A LEVEL NON-EXAM ASSESSMENT

A study app that helps plan your studies an also help focus on the necessary subject you need (Students planner)

# 

Saim Kazi

Contents

[Analysis 2](#_Toc513514936)

[Problem – 2](#_Toc1247787365)

[Procrastinating - 2](#_Toc494467350)

[Responsibilities- 2](#_Toc725986775)

[Lack of resources / vast information - 2](#_Toc1096824686)

[Over studying 3](#_Toc97183488)

[Feedback 3](#_Toc935528246)

[Conclusion 4](#_Toc978564588)

[Why this problem 4](#_Toc114290131)

[Possible Solution 5](#_Toc318507994)

[Stakeholders 5](#_Toc992026754)

[Interview / survey 6](#_Toc30514964)

[Question and answers 6](#_Toc1150140576)

[Research (Similar programs) And (website) (problems) 8](#_Toc1490715316)

[Todoist 8](#_Toc1369115508)

[Tick Tick 12](#_Toc686250492)

[Microsoft To do 15](#_Toc906751099)

[Any.do 17](#_Toc239361148)

[OBJECTIVES- 20](#_Toc1861428858)

[DESIGN- 23](#_Toc109669566)

[Design Section 23](#_Toc2074214699)

[Chosen Tools and Technologies 23](#_Toc1236349453)

[OBJECTIVES 24](#_Toc1387108730)

[Functional Objectives 25](#_Toc1869216329)

[Interface Design- 25](#_Toc240630911)

[Explanation – 26](#_Toc232110410)

[Flow charts - 47](#_Toc1728632905)

[CODED DESIGN 50](#_Toc1782257166)

[UML DESIGN 67](#_Toc1086678446)

[Flashcard - 68](#_Toc1185556264)

[MINI GAME - 68](#_Toc623541361)

[Full system - 69](#_Toc1741940590)

[Database Design 70](#_Toc2086211550)

[2. Table Definitions 70](#_Toc1332157723)

[3. Relationships 73](#_Toc401108705)

[4.SQL query code 73](#_Toc773401389)

[Algorithms- 75](#_Toc1347221126)

[Linear Search – 75](#_Toc268686425)

[Bubble Sort – 76](#_Toc1952038461)

[Binary search - 77](#_Toc950139974)

[Data flow- 78](#_Toc918641215)

[File Structure- 79](#_Toc142929618)

[Arithmetic calculation - 79](#_Toc593355360)

[Calculation for priority system 79](#_Toc1667613241)

[Calculating available time slot 80](#_Toc1458432436)

[Calculating Trend Analysis – 82](#_Toc916420156)

[Technical solution 83](#_Toc1492569676)

[CODE 84](#_Toc1912366146)

[Eclipse.py 84](#_Toc25957662)

[Login.py 85](#_Toc139938777)

[Main.py 94](#_Toc1463056493)

[Server.py 100](#_Toc2118450673)

[Home\_page.py 100](#_Toc1216813579)

[Taskviewer.py 104](#_Toc2029822503)

[Pomodoro\_timer.py 110](#_Toc31532594)

[Calander.py 117](#_Toc610191585)

[Tree\_game\_v.py 132](#_Toc1954586766)

[Scores.py 163](#_Toc872283856)

[Userpanel.py 167](#_Toc729258818)

[Flashcard\_v.py 174](#_Toc1358884633)

[Calculator\_normal.py 190](#_Toc572429466)

[Graph\_bar.py 202](#_Toc1627217893)

[Quotes.py 214](#_Toc31636444)

[Resources.py 215](#_Toc1145169537)

[Availability.py 218](#_Toc1859449829)

[Analysis.py 223](#_Toc825175811)

[User\_profile.py 232](#_Toc1678823991)

[ExamCountdown.py- 236](#_Toc2100471701)

[Program flow - 241](#_Toc873382303)

[Technical used 242](#_Toc1009888618)

[Linear search - 242](#_Toc1251923236)

[Binary search - 243](#_Toc2130891139)

[Tree traversal - 244](#_Toc476490579)

[Calling parameterised Web service APIs - 249](#_Toc325386367)

[Server-Side scipting- 252](#_Toc1845647034)

[Aggregate SQL functions - 256](#_Toc416660290)

[Cross-table parameterised SQL 257](#_Toc1148054782)

[List Operations- 258](#_Toc445144079)

[Files(s) organised for direct access 260](#_Toc288736039)

[Complex scientific/mathematical/robotics/control/business model 260](#_Toc1439040174)

[Complex User-defined algorithm 263](#_Toc1365078881)

[Bubble sort 266](#_Toc61821646)

[Classes 266](#_Toc11762424)

[Inheritance - 268](#_Toc1948324058)

[Composite key - 268](#_Toc104115414)

[Polymorphism- 274](#_Toc504681926)

[Dynamic Object generation 276](#_Toc497779463)

[Simple mathematical calculation - 280](#_Toc2029782650)

[Singel dimensional array - 280](#_Toc1884075318)

[File handeling - 280](#_Toc1502506750)

[Multi-dimensional Array 282](#_Toc48191003)

[Dictionary - 282](#_Toc217400659)

[Simple datbase model - 282](#_Toc1836092332)

[Simple OOP model - 282](#_Toc1983450504)

[Simple Object generation of object 283](#_Toc112541683)

[Simple user defined - 283](#_Toc1824766695)

[Non-SQL Table access 284](#_Toc298809981)

[Writing and reading files 285](#_Toc1319128997)

[Files organised for sequential access - 285](#_Toc1826936026)

[Records - 285](#_Toc1320848751)

[Recursive 286](#_Toc943297376)

[Calling Web service APIs and parsing JSON/XML 288](#_Toc891240905)

[Stacks - 290](#_Toc1663642423)

# 

# Analysis

## Problem –

After years of being spoon-fed by their teachers, students are thrown into the world of education, yet at one of the crucial points that determines your career, A levels and beyond in university this two is the path to your carrier GCSE was the first steppingstone, with lots of help from teacher. This leads to a great deal of stress because, in addition to not knowing how to study, some students also do not know what to study or how to organize their future studies. This causes pupils to attempt, give up, try again, concentrate on other topics, or even study with no thought process at all and it can also lead to many different emotions such as discouragement, stress, panic, and worse case depression.

### Procrastinating -

Students after going home they have 7-8 hours of free time, shockingly there is 21% of students that use this time to study and revise the other 79% they use this time to relax and do nothing but be on social media, they do not know how to balance the time between studies and entertainment. This leads to negligence and leads to cumulative stress building onto that person, and it would end up leading them to depression as the human will feel trapped, due to not being able to escape the person will feel defeated, useless and will start thinking negative and to make them feel good again they will use the leisure time to play games or distraction themselves from their responsibilities as the easiest way to coup with stress is to forget about it . This can also be a result of not enough dopamine from studying , dopamine is the main hormone / feeling we use to motivte out self to do it , for example games give dopamine as the user has a goal to complete and when they complete the goal they feel a sense of realife and satisfaction which you do not get in studying as student say its “uninteresting”.

### Responsibilities-

On the other hand, some people have less time to study as they must take care of their siblings or just have other curriculums that they may have to do such as martial arts, gym, school clubs or driving lessons etc. This will also hinder the person's study time and affect their focus as if the person will get fatigued, which affects how they will study and how focused they are. Some students even help around the house hoover, cook, this takes away a lot of time from the students and leaves them tired from all the work they may have done. With all the studies

### Lack of resources / vast information -

Some students know how to study, but they do not know how to get resources from, what to look at or what is useful for their studies, because of the vast amount of information available online and in textbooks, self-studying students may feel overwhelmed and struggle to determine what is most important to focus on. which leads a lot of students in the wrong direction and leads them to a plumet as they do not know how to self-study. “The study, which surveyed 8,353 students from 25 colleges, reports that 84 percent of respondents found "getting started" to be the hardest part of research projects.”

<https://www.huffpost.com/entry/college-students-dont-know-how-to-research-study-shows_n_781269> .

### Over studying

Also, Students should not use all the leisure to study or use their time not to they should be able to make a good balance, which this will help they studies and in the long term as they are enjoying their free time and being productive and getting help and maximizing their study time. Over studying can lead to burnout, the amount of your focus, Decreased Productivity, an increase of anxiety and stress, poor physical health,Risk of Academic Plateau or Decline. Over studying can contribute to heightened levels of stress and anxiety. Constantly pushing oneself to study beyond reasonable limits can lead to feelings of overwhelm, anxiety about performance, and a sense of burden that will be built upon oneself which can make some mental problems. Over studying can lead to mental and physical exhaustion, resulting in burnout. This can manifest as feelings of fatigue, lack of motivation, and decreased interest in studying or other activities. Despite the intention to improve academic performance through over studying, there is a risk of reaching a plateau or even experiencing a decline in performance due to diminishing returns. Continuous study without adequate breaks and variation in study methods may not yield optimal results. Neglecting physical health needs such as adequate sleep, nutrition, and exercise while over studying can have detrimental effects on overall well-being.

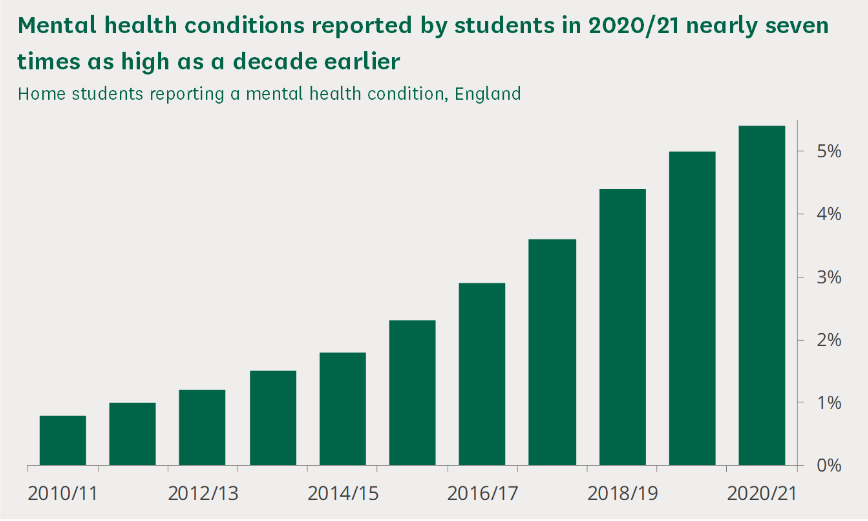
### Feedback

Another problem is that the students cannot gauge their understanding of their problems as they cannot get feedback. Feedback from teachers or peers is invaluable for correcting mistakes and misconceptions. When studying independently, students may not realize they have made errors or misunderstood concepts without someone to point them out and provide guidance on how to correct them. Even when self-studying students seek feedback online or through resources like textbooks, the quality of feedback may vary. They may encounter conflicting information or inadequate explanations, which can hinder their learning progress. And without any feedback the students would not know their progress like if their excelling on the topic or their needs to be more attendance, the data that the students get from the feedback helps them move forwards and either get less time on the subject or he needs more attention of the subject which help planning. Knowing that you are doing good in the subject gives confidence to the students and allows them to not procrastinate as they will feel a sense of joy from all the challenging work. Not only joy they will be able to gauge their understanding of their topic.

### Conclusion

In summary, there are a variety of reasons why studying is beneficial for students. These reasons include the ability to achieve their goals with the greatest amount of efficiency, to prioritize their mental and physical health, to stay up with their academics, to enjoy life, and to not completely sacrifice the things they love.

## Why this problem

This is a widespread problem with students around the world as students face this difficulty with little to no help. This is also a problem I have faced throughout my study life, and I have experienced this which is why I am facing this problem and making a solution so no other person will feel how I felt during this period. I have also researched how the problems affect the person well-being, and in this report mental well-being is rising at a fast rate 2016 was only at 3% and in 2020/2021 it increases 2% and its now 5% and still increasing,https://commonslibrary.parliament.uk/research-briefings/cbp-8593 (all the information there)

I have also asked my head of year about the pass rate of our school and she has given me the past 2 years , and thanks to the data I have seen a decrease in pass rates as in 2022 is 70% to 56% in 2023 and this is thanks to covid as it split the flow of education and now more than ever we will see a really big plumet in the pass rate as education is evolving but the ways of studying isn’t , that’s why I'm solving this problem as students need support , validation and a general guide to how they should tackle their studies . I have experience this at the beginning of my A level I was overwhelmed with studies, responsibilities and making a name for myself, and if I had such an app with me, I would not be such stressed and be able to balance everything and get some time for myself. This is a problem which is not answered correctly.

## Possible Solution

With the variety of problems that students face in their education, students find themself unable to plan and how to tackle the situation, they need a planner timetable, but we all know that life would not allow all those plans to happen which is why I am making a Revision planner.

I am going to make a program that will make a timetable for the subject that the user is struggling with or the subject that they want to focus on, without neglecting the other subject. But the app needs to be flexible to meet with the user timetable which is why I am going to also allow the user to input their activities into the program and either push or delay their study period for later or even remove that certain block of studies. This solves the problem of being overwhelmed as it will create time and allow the user to get time to themself. The user will also be able to make changes to future events as they see fit. Students face the problem of procrastination when they study so I figured I should make the app into a point-based system, so it allows students to engage into their studies and compete with other users / friends. This app is going to be more like a reminder app, I want the app to take your data of your test, and the possible mini quizzes that is included in the app and make it into a graph to see how your education is going so far.

The program will focus the person studies while giving the user the space and time they need to cool down, this is because the student who is packed with their life, they have a harder time to study and slowly they will slowly degrade the person mental well-being and eventually lead them to bad grades and even worse depression.

The program will also fix other problems like boredom and the dopamine problem I will be making a mini game that focuses on revision like a quizz , snakce game or other type of revision game tools .

This will give the user motivation to study as they will be see their fruit of their effort by keep playing a game , the more the user gets the answer correct the more achieved they will feel making them actually focused on the revision not trying to lose in the game .

Another problem is the revison material the students need to find / get. The internet is an ever evolving each day with many guides and many different way of studying and revising which makes it hard to trust any of the website or even know if the resources are correct which is why I will be making a resource library that will help the user have their resources on the get go . I might use an Api but I am not sure at this point . this will solve the problem of searching for the right resource to study as the user will be going through on a endless search .

And finally the last problem that affect students studying is the fact they do not see themselves grow or decrease , a person needs to see that their hardwork is being paid off and beign used or shown that they are getting better . To solve this I will show the user a graph that shows their trend and also an analysis page where the user will be getting in depth analysis of the given subject . This solves the problem of self growth , this is a big deal as some students will be growing slowly while other eponentioally and comparing one self to someone who getting more then them can automatically make a sense of dissatisfaction . The best way to solve it again is to show the user the score increase and their future increase giving them a sense of beliefe in them self .

## Stakeholders

My program is aimed at the students as they do not know how to balance their time as their naive and lost, so the program is focused on students who need help and has an interest in studying and will give time and follow the scheduel. This program is also aimed at students or even people who have a busy schedule and are unable to find time to balance free time and work. Alot of people either have too much free time or they work too much, and they can't find an appropriate balance, having too much free time it sounds good but it's bad as the person will forget its studies and start to get lazy and when their exam is fast approaching they realize the importance and start to panic and try to get the content of a whole 2 years in 1 month, stress over flows them which doesn't help them at all. On the other hand, students who are studying every day for extended period trying to become better each day they are hindering their learning abilities as they are not getting enough free time to rest their brain and body, this will lead to a burnout affecting the students mental and emotional state.

Also, this app can also be used by other people as it is not entirely directed to only students it can also be used by other people such as: office workers, teachers and normal citizens as the features will be useful to them. But the app is focused on the students mainly in 6th form and university students as most of the features help study and plan their study and help find resources for their studies

## Interview / survey

I have interview one of my tuition teacher Mr. Rohman as they have the most knowledge about student and how they struggle, he has seen students struggle with their studies and balancing their life, so I found fit to interview them and move forwards with project. I also interview other suitable teachers and even students to know their different opinions. I interview Mr. Islam who is my old science teacher , and the student named Maisha who studies business and math at college level .

R = Mr. Rohman, I = Mr. Islam , M=Maisha

### Question and answers

**What are the main reasons students do not study?**

R=Students don’t study due to the reason that it’s boring, not interesting and annoying and all of this is fueled by their own emotion pushing repos abilities to the end. It could be a many different factors as well and a main factor is life, what i mean by that is that students dont have a linear life so emotion will fluctuate one day will be mentally stressful or sad or even angry. I believe students should make a timetable depending on how much free time they have and make it flexible.

I= I have seen in my years of teaching is that students can’t balance and understand between which subject they should be studying more and which less priority. What back my saying is the point when I give math homework almost everyone does it while if i give science 20% comes back done and that's the reason of math being harder and even tho the students are doing good in the subject, they still focusing on math more giving less time to science.

M=I am student myself and I cannot balance my life schedule taking care of my siblings and also study when i start studying i spend 15 –30 to see what subject to revise and I usually mess up and revise the same subject again due to liking that subject and I get more streesed more stress gets applied the more unmotivated I get

**What helps students balance their study and leisure time?**

R=A student's planner, I really like the idea of planning out your studies which is why I gave them their own student planner hoping they use it for their study . Planning the study before the week it can give some reassurance to the user and a sense of comfort as they got a plan and know what they ment to do givin them. I always belive planning you time

I=I believe a score tracker or having a frequent quiz / past paper that they do to check if they are doing good in a said subject or else the students will never come to a satisfied result and will neglect other subject and take all their free time. So, the students so see their score and decided how much to study for their individua subject

M= What helped me was to give my self goals such as 2 hour a day or complete section one for today e.tc

**What was the result of the given method?**

R=The class was using it for couple of weeks, and before Christmas arrived the class was divided into 3 diverse groups one which still using the study planner and doing their homework, another without a study planner and still doing work and the other group with no study planner and not completing their homework. The method helped remind students about their homework or the work they need to do.

I=I gave this method to all my students only 4 students started using it, and I see 3 of the students are completing their homework and looking less tired than the week prior

M= I don’t panic as I have a set goal that I will meet if I take it slow

**What feature do you think would help the student?**

R=I think that a constant reminder will help students study, as one of my students said that she was so busy she forgot the homework she was meant to do tomorrow but thanks to her putting a sticky note on her desk she was able to remember it. I also to see their test results and progress of their studies

I=The recommended feature I can give is properly a priority tracker type thing where the user enters their scores, and the program will calculate what level of priority the subject should be and maybe how many hours they should give it

M= A minigame so when I am tired and not bothered to actully revise I want to play a mini game that's makes me more interested in the work

**Do you think more leisure time is important or more study time?**

R=I would say having a balance is important, but I also think having more time to relax is better as it will allow them to have their own time after a hard day of work

I= rest is needed due to the reason of the brain being a muscle and it needs rest if they don't get sufficient rest, it’ll be overworked and will be

M=I think a bit of you time is much better than studying all the time, I have burned out Mutiple times due to all the stuff going on and still forcing to study

**What affects the student's mental health when studying?**

I=Stress, pressure, expectation, self-doubt all of these are the things that affect their study and all of them lead to overwhelming studying and into a spiral of unhappiness.

R=other factors then studying, such as motivation some students see their mark 30,40,50 0r even 20,50,30 and they get discouraged but I feel like if they see a trend where their progress is going up then they will be more motivated

M=the teacher pressuring to do everything they ask for having mocks the next day after ur coursework submission not able to balance that makes me more depressed and sad about my school life

**Do you think an assistant will help them focus on their studies?**

R= maybe something that creates timetable or a plan for the week might help but I do not see the reason for any other assistance after all every student study their own way .

I=I personally believe someone who remembers their task and manages their time to suit their needs will be a great tool as it cares about the user's wellbeing but also not putting them behind their studies. Studies will feel easier as it will not be a burden.

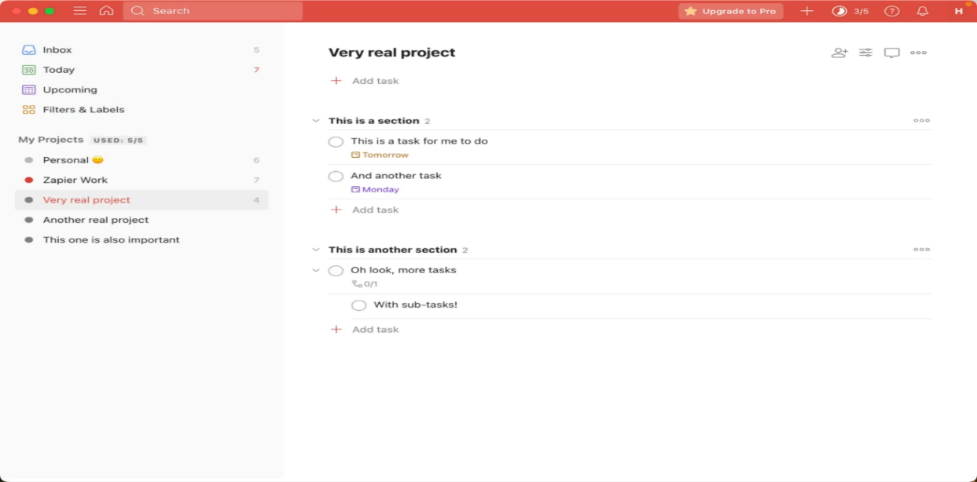
M=The first thing I want to see is all my task that i need to do to remember

## Research (Similar programs) And (website) (problems)

I have researched different apps like my project and want to see the pros and cons of each of the apps so that I can get what people liked from each of the apps and hopefully implement the idea into my app and make the app a fluent a functional for the students and be super helpful for students as they need much planning and much help as they can

### Todoist -

The app is simple and plain, the app focuses on more on function than aesthetic. This app allows the user to store and categories each task into its individual group allowing the user to have an organized workspace. The app is good for breaking down tasks and allowing the user to see clearly what to do. The app also allows color co-ordinate the groups for simple and efficient way to differentiate between ask

This app has an uncomplicated way to add due dates and options to be able to change by just dragging the item from one end to another, its flexible enough to adapt to most workflows but not so complicated as to overwhelm AI assistants and keyboard navigations. The app is simplistic looking, focusing on the functions more than on its looks. It has easy to understand functions and is easy to differentiate the functions.

A screenshot of a computer

Description automatically generatedA screenshot of a calendar

Description automatically generated

The app also has an option to set the date that it should be set for. This can go up to for 2 years, the app also allows to set the exact time or before a task

Quick add button and clear, which makes for an easy access to make new task

A screenshot of a computer

Description automatically generated

This app also has 4 different priority levels, this feature allows the person to set a priority and focus on that

A screenshot of a computer

Description automatically generatedIt also allows the user to add a habit to the app and allow AI to help them to allow their goal to be attainable instead of unreasonable demands that the user thinks he should be able to do. It also has another feature to help the user

A screenshot of a computer

Description automatically generated The app had the option to add a description so the user can say a specific subject and what exactly to focus on

A screenshot of a project

Description automatically generated

This is also good as it allows the user to categorize their task into different sections

A screenshot of a computer task

Description automatically generated

I don’t like the way that the task is displayed as it looks boring and it’s more like a checklist than a planner as it doesn’t put it in a time slot. it’s nice to people who has lot of time, as it doesn’t seem like it feels their schedule but it’s not helpful for the people who has a lot of tasks to complete during the day as they will not get a notification about their task, therefore forgetting their task.

A white background with black lines

Description automatically generated

The priority is hard to understand which is more important as it doesn’t highlight the important so isn’t eye catching

A screenshot of a computer

Description automatically generatedbut the problem is that the app is hard to navigate as it's not easy to understand at first it takes time to get the understanding of the app it's also really boring as the app is has simple colors, the only thing that you can do is to change the color of the app but even, so the app doesn’t look pleasing and just plain

SUMMARY –

It’s a good app to plan your day and help you choose when to do your task it more like a checklist then a time planner or a study planner as the user inputs their task and it’s up to you when to complete it. After using the app, I liked how it was fluent and useful, but after some time it got boring and can get confusing as you cannot differentiate between notification and actual task, also the routine was a mess and prior

PROS –

Easy to make task, priority levels, AI assistance, what’s the program being used for, able to plan and set time and date

CONS –

Priority level confusing, Boring, unable to see time slot, unpractical, the app is confusing not enough clarification

### Tick Tick

- pinned notification and widgets on mobile or Computer easy to add task, task can be organized using lists, tags, due dates and priorities and able to add subtask to any task build in pomodoro timer (25 minutes’ work and 5 minutes’ break) and allowing third party apps and allow to see all your appointments in one place priority level allowing you to review how many days you did or didn't exercise and diet commitment

A screenshot of a computer

Description automatically generated

The app has a pleasant and welcoming appearance to it and due to the feature that the app can allow you to drag the task from the

A screenshot of a computer

Description automatically generatedIt also chooses the category of the to-do-list that you want, its good as it puts all the category, but it has limited section, so you cannot add more than there is

A screenshot of a computer

Description automatically generated

A calendar with many colored squares

Description automatically generatedA screenshot of a computer

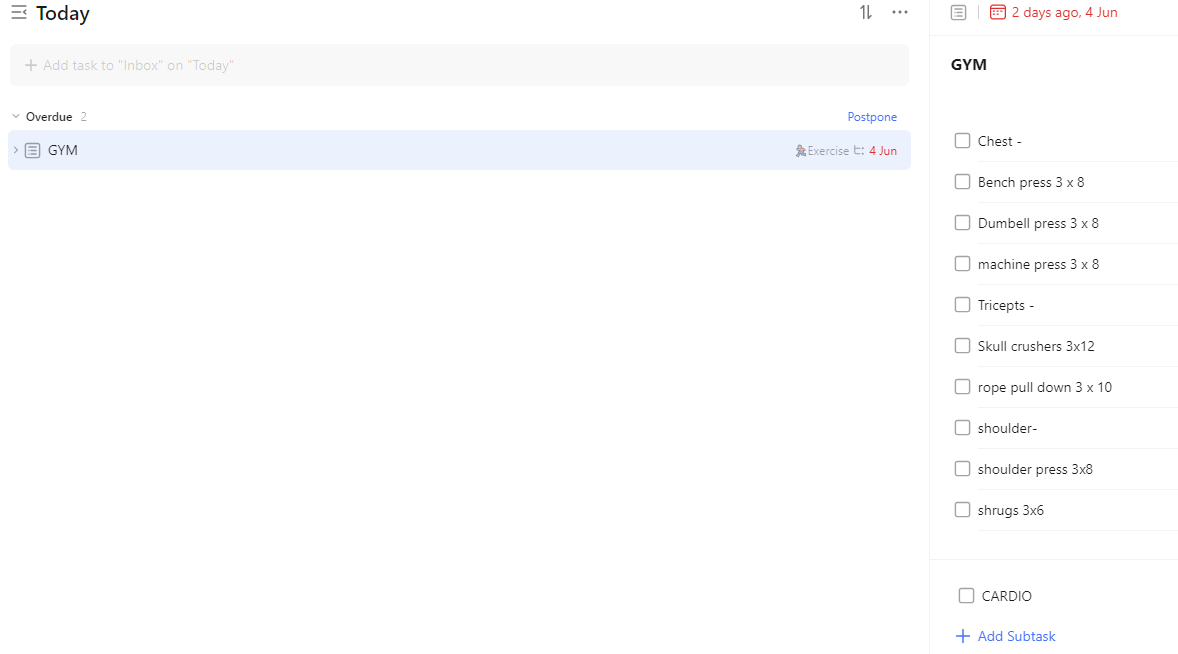
Description automatically generated

The app asks you of which feature you want with your app; this can also be added later but the fact its s

A screenshot of a clock

Description automatically generatedA screenshot of a computer

Description automatically generatedhow's a preview of the feature allows the user to see what they want

You can make tick boxes so when you are doing a big task you can break it up and make it into little task which slowly you can tick off. The amount of detail you can put on the description opens lots of possibilities and a lot of flexibilities that allows the user to freely make task as they wish for

A white background with black text

Description automatically generatedYou can also make a Sub-Name which you can break the task even further down, good for studying, or good to break down a project.

A close-up of a person

Description automatically generatedThis is super useful as the user doesn’t have to think before they do the task as they already broke down the problem into small task which allows the user to do the task and tick it off. Good feature too allows to see each individual task

You can make multiple Sub task and you can make as many of the sub-name as you wish and as many different tasks as you want. Also due to the reason that you can also repeat the task, which makes it super useful for people who has a routine at the gym, allowing them to see what workout they have done

A screenshot of a computer

Description automatically generated

The exploded view is perfect to see the whole month plans and see where there will be free

A screenshot of a phone

Description automatically generatedThe app includes the Pomodoro study method and a stopwatch, this is useful for students such as me who love to use the Pomodoro method as I don’t have to go to another tab to see my task and how much left I have until I finish my allocated time

Screens screenshot of a phone

Description automatically generated

The are also had a preview of which time you will end when u start the timer, this is useful if you have a class in the next hour and you have forgotten it as it will remind you and keep you on track. The feature also has an interrupt to make sure you are sure of quitting the section

A screenshot of a computer

Description automatically generated

app has habit system which allows the user to set a routine like cleaning their room, university lesson or even revision planning there is endless possibilities of which the user can do. Feature is super useful for people who has lots of chores and study.A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedThe App also had an area which allows the user to assign which level of priority the following task is by just dragging the task to the corresponding level.

The app also has search feature which allows to see the task faster due to the feature, this is efficient for the user who has a lot of task and cant find the specific task,

A screenshot of a computer

Description automatically generated

SUMMARY –

Overall, the App is super fluent and functional and efficient for students who are living alone and must balance a lot of studies and chores, The app is plain but not so boring that I don’t mind its look. But on the other hand, the app is confusing for new begging, and some features are hidden in the app and not clear, the app has a boring home page and seems like there is nothing, you also must go through different sections to see the stuff you want. There isn’t a quick add button.

Pros –

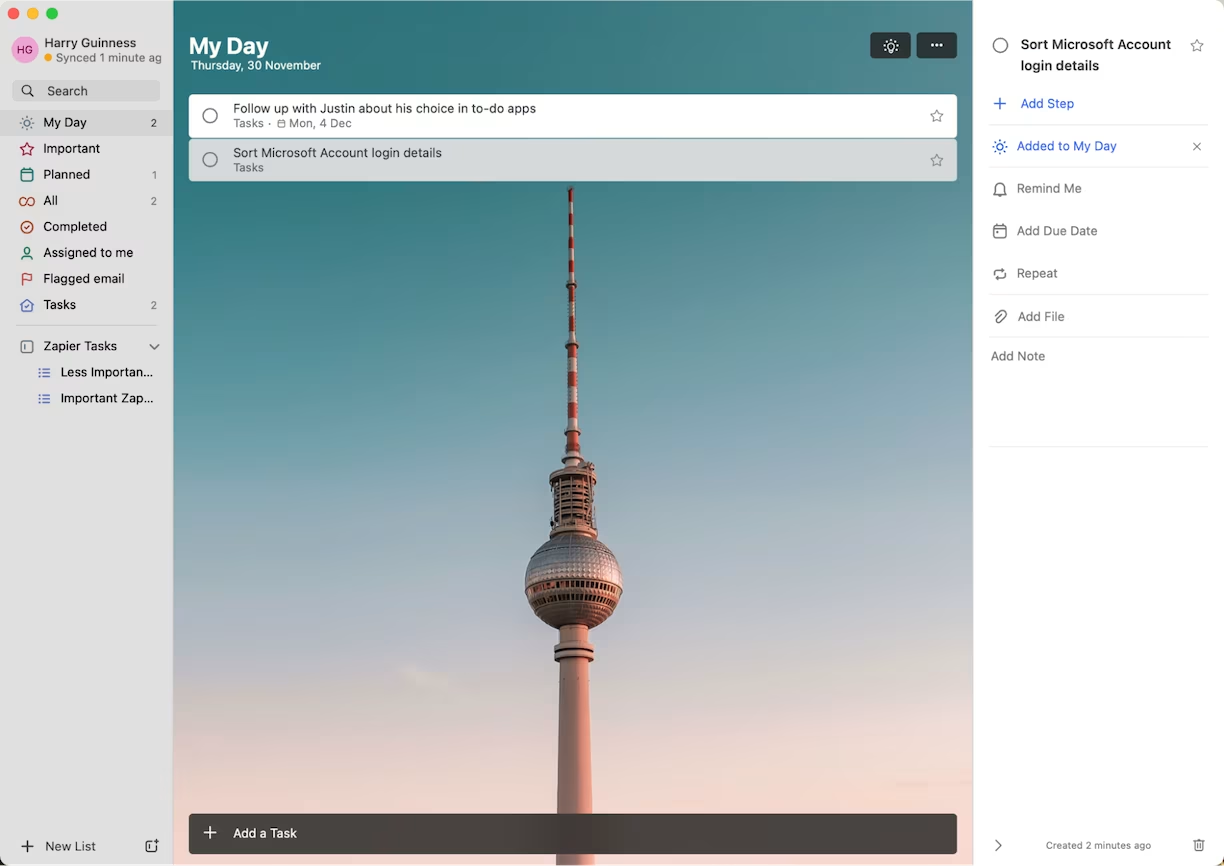
Different features, section for each subject / task, A routine can be made, exploded view of the month, search feature, priority assignment, in app timer / stopwatch and the app allows you to make mini task in the big task.

Cons -

The app is super confusing and doesn’t have a quick add task button which means you have to add the task in a certain area, the app has limited sections so you cannot make many.

### Microsoft To do -

the main interface is clean and friendly, and adding task is quick a lot of flexibility deep integration with Microsoft's ecosystem, this app shows all the unopen email and if an outlook sets a task it put it on the task. window user can add tasks using Cortana or by typing in the Start menu, for example, you can type "add rice to my shopping list," and rice will be added to list called "shopping " Good looking to-do list app, customizable background



As soon as you enter the website you will get a quick run through how to use the app and what kind of features there is on the app and what I can do. The app also has a quick add button which allows you to quickly add tasks. The app looks boring and doesn’t have many features it only allows the user to repeat, add due date and set a reminder for the user. The different feature is that the app allows to change the background when needed

A screenshot of a computer

Description automatically generated

SUMMARY – Good app but it doesn’t have much feature that benefits students it’s more like a list then a student planner the noticeable thing is that the app has all the feature before you add the task

PROS – good for listing and reminders

CONS – Not good for students planning and not good for organize the subject as you wish

### Any.do

The app has a clean and very responsive interface. The app is very customizable and friendly to use, easy to understand and does its necessary tasks without having a composed of useless data, this makes it so that the user has a smooth experience without confusing the user with useless information. The app has multiple customizations such as background, customizable priority tags, task classification e.g.(home,work,tax).

A screenshot of a computer

Description automatically generated I like how the app asks if the user is going to have a team or other people with them during these moments. This may be useful to keep some students in check and motivated in their studies

Screens screenshot of a computer

Description automatically generated

The app has different task features they can do such as planning a trip or doing chores and call back a person categorizing different task in the way that the surer can differentiate which task is which and what to do on each task proves to be super useful and efficient this

makes the app more into a checker for oneself and makes it really interesting and useful for the user as it allows it to remember to do certain things. And the certain task has a timestamp so the user will get reminded of their task and the latest they should do.A blue and white background

Description automatically generated

The app has a beautiful home page totally different to the home page we had seen before and makes it more welcoming to the user than other apps that looked boring and annoying to look at it has clear symbols and instructions on the home page. This particle page displayed all the tasks that the user may have. The app also has a quick find button on the top right which allows for a quick search of the task and a notification bar to see what task are on reminder, simple but super useful for people who are busy with life

A screenshot of a phone

Description automatically generated

This is how it looks like if the user has task to do , the page is beautiful and full of life and well thought of and the page looks really simple but easy to understand and thank to the box on the side you can organize the task whichever way you want add to the different list or give it a tag (priority) , you can also add an attachment and also add subtask really easy to see and really flexible for the user. There is also a description area, and a reminder option to make sure that the user doesn’t forget the task they have assigned.

A screenshot of a computer

Description automatically generated

This page allows the user to see the task that will have to be done today. The page is simple but pleasant to the human eye as it is not just black and white. On the side there is a feature that allows the user to fill the area with the task with the corresponding task on the time that the user want to do, the app also has an engine that suggest the user on which task that they can fit in their schedule

A screenshot of a computer

Description automatically generatedThis page shows all the task in the upcoming 7 days which allows the user to see a quick and easy overview of the person work or even the person study. you can also add task easily same as the All my task page, this is fast and efficient way of adding and overview your week.

A screenshot of a computer

Description automatically generatedThis is an expanded view of the top section this shows all program (action, task) in each hours of the day and u can drag and customize when you want to do the task , you can also use this area for add task in the certain tiem block this is super useful to see your full week task and timing and with this you can plan to meet your friends or make time for your leisure time.

Screens screenshot of a phone

Description automatically generatedIn my list section you can see each section and what task is assigned, which makes it easier to look at if it’s subject or even routine passed also has an add button again to add stuff directly in the corresponding list.

SUMMARY –

Any.do is a good app as it has many features that complement its efficient way of app style its customizable allowing the user to have any background as they wish for. The app is pleasant and is nice, and thanks to the view calander and other feature a person can plan and be able to manage their time and not forget the task as the app has a build in reminder which super useful for keeping on track with the assignment or project that the user may do.

PROS -

You can add task in any window making it easier to add task, very easy to maneuver, search function allowing for quick finding task that they need , the app is useful for many reason different task (study , groceries , chores or work ) , a calendar with time allowing to see what time the user is going to be free , Also a calendar with just the task that need to be done and the task in the following 7 days .

CONS-

Limited number of tags not apple to show which is more important and which is less important, you cannot drag and drop task on the calendar view making 2 times harder to use and also is hard to get the exact time with it making it annoying to use

Final –

|  |  |
| --- | --- |
| The app | The Function I am going to to use |
| Todoist | * How the app is being used like (work, study, general use) * AI assistant (Help with making a timetable and making it duable) * Able to see the whole month Calander (able to plan) * Priority level |
| Tick Tick | * Pomodoro method / timer (Good for students to get breaks and sufficient time for studies) * Able drag and drop task to certain priority, timetable, category * Making sub-task to break down a problem / task / project * Able to make a list in the description (making tick boxes to the description) |
| Microsoft to-do | * Reminder system * Clean UI * Calabder focus |
| Any.do | * Calander view with time to see availability * A task overview of the week * Task of the day * Add buttons in all the area (easy to add task) * Notification section * Reminder section * Categories * Easy to use * Very customizable |

In conclusion I have seen that the app Any.do has a nice UI and a easy to use and properly the idea for UI set up but for functionality Tick Tick is better And I might use some of the other functions it provides.

Moving forwards, I need to see how I am going to make a program that will create the timetable for as that's the only feature that I have no idea about as there are no example apps.

## Research evaluation with solution –

After doing research on the problem, taking interviews and looking at existing problems I have seen that the main reasons why the students struggle to study is that they are:

It means the students who want to study they unable to study efficiently due to unable to understand what subject to study and what subject to not to study as there is numerous amounts of cases where students will study one subject over the other due to their familiarity with the subject or their worry about such subject.

To solve this problem, I will be making a program that will make a timetable depending on the user availability and scores . To calculate the priority level for any subject we need to use weighted subject the more weight the more priority it gets and it will then get more time allocation then the other subject and the time allocation will be calculated by taking the end period – the start period to get the specific day study time .

The next problem that students face is the drive to actually study for their subject and that is because there is no dopamine , dopamine is a hormone where a person get when they get win or get a rewards for their action which this hormone is also the reason that people get motivated to do actions ,that’s why lots of young people play games due to this dopamine .

To Solve the problem is simple I have to make a minigame that will let the user play through and every time they get a question right they make an action which makes them feel awarded for and by doing this it gives a motivation to the student to try to get more questions right and therefor giving them a sense of reward / drive to do better in their subject . This will fix the motivation issue and boredom issue aswell as now students have something to look for and test themselves with while also not being bored about it .

Another problem is that students do not know where to go in the internet to get their resources , So I will make a section in the program where they will be able to type their subject and get their resources from there .

Also there is also another feature that will suit the program and that is having flashcard system. This is helpful for students for many reason and one for them is for active recall and doing a flashcard app will help students remember more .

## OBJECTIVES-

1. The user needs to be able to login and signup
   1. The system should validate with the database for right credential
      1. If the wrong credential was inputted, then it should show and error
      2. If the right credential shows a welcome screen and take it to the main program
   2. If the user signups it should pull up another frame / window for the sign up
      1. The sign-up program should validate if the username was taken or not
         1. Display error message if username is taken
      2. The system should check if the password macthes
         1. The system displays error message if the password doesnt match
      3. Make sure that both username and password are 4 characters long
         1. If they do not meet such requirements then show error
         2. Do not crash the code
2. Finish setting up the user program
   1. The program should check if they have details or not if they dont then take it to the following program
      1. If the user is directed here, then let the user enter their subject and availability
         1. Do not let the user get past if there is no subject inputed
         2. And show an errore when no subject was added
3. The user should be able to see, add and complete task from the home page
   1. Show the task in different section
      1. The task will need to be split on today, tomorrow and upcoming
   2. The task shoudl have complete button to signify competition
      1. When the task is completed, the task will not be deleted on the same time only after closure
      2. Task should have a clear indication of their completation
   3. The user should be able to have a task adder
      1. The task adder shoudl contain label, date and time for specific storage
      2. If the there is no label, system should show error
      3. When the task is added it should show both on the calander and the homepage
4. When the user calls Calander program it should display, and add task
   1. The calander will be able to show next week or the previous week
      1. When the user presses the next week it will load all the next week task
         1. The task shouldn’t overlap with this week task
   2. Make the app smooth and seamless
      1. Smooth creation
   3. The task saved on the database should be displayed at their corresponding date and time
      1. It should show a notification for confirmation
      2. It should update the Calander
   4. The user should be able to save the task on the database
      1. Allow the user to add task in the cell they click on
         1. Make sure the right row and column is extracted and saved in the database
      2. Make sure the label is filled
         1. If no label, then display the error and tell the user to enter the label
5. The user should have a section for flashcard (adding, deleting and displaying)
   1. It should show all the subject like a menu
      1. The program shoudl retrieve all the subject and display the name
         1. Gets the subject from the specific user\_id
         2. Ges the subject from the specific subject\_id also
   2. Should show all the selected flashcard if there is any if not tell the user to add cards
      1. Show all existing flashcard from the specific user\_id
      2. Add the flashcard to the specific user\_id
         1. The program should validate if the user inputed a front and a meaning for the card
   3. Let the user to either add more flashcard or view the card
      1. The button should allow to swicth to view teh cards
      2. The viewing of the card should have flipping, deleting, and next card button
         1. When deleting the succession of deletion shoudl be displayed
   4. The user should be able to return to menu
      1. The user should be able to go back to main menu
6. The user should have a pomodoro timer section allowing to have 2 option (25 min, 5 mins)
   1. They should be a start, pause, reset
      1. Start
         1. Should show countdown and a notification when the timer is done
         2. Reset back to the beginning waiting for the user input
      2. Pasue
         1. Should pause at the specific time it was paused
      3. Reset
         1. It should reset back to the 00:00
7. The user should be able to get an overview of their subject scores, availability, the resources, and enter scores, also an Indepth analysis of the specific subject
   1. A graph to show the users results and progress
      1. The user should be able to see which year, the month period (last 3 month, or 6 month)
         1. If the graph has no data make sure the lines are on 0 and doesn’t show error
         2. A clear difference between the different subjects
   2. The user should be able to change their availability
      1. If their no availability shows 0 not error
      2. If the user puts the wrong time show error eg(15:00-14:00)
   3. The user should be to enter the scores at anytime of the year or month
      1. When they add a score make sure the subject it corresponds to is entered or else tell the user to add the subject with an error
      2. Give the flexibility to add at any time date and month and year
   4. The user should be able to enter the subject name and get the resources with the name
      1. If no subject entered show error
      2. When the subject is entered and its valid then show the resources
   5. The analysis program should be able to see Indepth summary for the specific subject
8. When the user play the mini game animation should show the user health depletion and also the boss
   1. For every action the user makes false or correct show a notification for such move
   2. Animation should play for boss attack or player attack
   3. When the user gets hit or losses damage show notification
9. The program should be able to make a time take by takign availability asn the latest scores
   1. Let the program differentiate between a study day and off day
      1. Days with no availability should be let go for time allocation
   2. The program should fecth the last score and generate a schedule
      1. Make a schedule depending on scheduel
      2. Store teh schedule to the database

# DESIGN-

## Design Section

The design of the project will include ease of use and intuitiveness of the system, catering to the primary functionality of organizing and managing a student's study timetable. The system will further be extended with features that enhance its usability and versatility to accommodate varied user needs beyond mere schedule management.

The desing of the project will include functionality and user-friendly UI, making it easy to use. The program should have a timetable generator

## Chosen Tools and Technologies

To develop the application, I will use the following tools and technologies:

**Frontend: CustomTkinter**

CustomTkinter , a Python GUI library I choose this library as it's something I'm familiar with and easy to code and it's better to normal Tkinter as it gives more advanced customization options and gives a modern design to make the app look better and appealing making so the app is a bit more customisation.

Key benefits of using CustomTkinter:

* Enhanced customization options for widgets, including colors, shapes, and fonts.
* A modern and clean design, improving the app’s visual appeal.
* Simpler implementation of complex UI features compared to vanilla Tkinter.

Disadvantages:

* Doesn’t have already made buttons like Flet
* Program needs a lot of detail for each individual button
* Hard to do some aesthetic features

**Backend: MySQL Database**

The backend will be supported by a MySQL database. MySQL is chosen for its robust querying capabilities and efficiency in handling structured data. This database will store and manage all necessary information, such as user details, study schedules, and any additional features that I migh use .

Key reasons for selecting MySQL:

* Its ability to handle complex queries effectively.
* High performance and scalability, ensuring smooth operation as the app grows.
* Compatibility with Python, allowing seamless integration with the frontend.

**Database Management Tool: XAMPP**

XAMPP is a app that allow the user to manage their database with ease and due to being open-source platform that simplifies the setup of a local server environment. Its intuitive interface and ease to use make it suitable for management and ideal for my project as there is goign to me many different datastores. Although XAMPP supports other database I will primarily use MySQL because I'm familiar with it then the other databases .

Key advantages of XAMPP:

* Easy installation and configuration.
* User-friendly interface for easy management of database operations.
* Built-in support for running and testing locally hosted applications.

## OBJECTIVES

### Functional Objectives

The application will provide the following core functionalities:

1. **Study Timetable Management:**
   * Allow users to create, edit, and delete study schedules.
   * Provide options to categorize tasks by subject priority, depending on the user scores .
2. **Extended Features:**
   * Mini Game to help to make studying fun..
   * Additional resources or links section for users to store helpful materials.
   * Flashcard so the user can revise his own way and have access to it anytime they want
   * Pomodoro timer allowing a user to have in build timer so they can study with the app open
   * Have a task viewer so the user can see their task
   * Analyse the subject and give suggestion to the user needs
   * Sove the user problems (such as why they can't focus e.tc and other propblems like tired confused etc.)
   * Have an overall view of a subject in the given years

**Design Goals**

1. **Scalability:** Design the database and application structure to accommodate additional features or user data as needed.
2. **Performance:** Optimize both the frontend and backend to ensure fast response times and a smooth user experience.
3. **Security:** Implement appropriate measures to protect user data, such as password encryption and secure database connections.
4. Make it all appealing not black and white

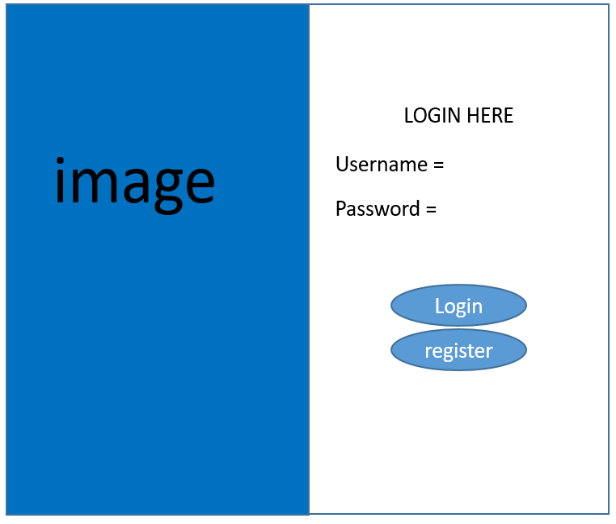
## Interface Design-

### Explanation –

I will show Illustration of how I want the app to look like how each page will interact and what they would have in each section I am also going to break down each function and later adding the code that I might use for the certain features. I am also going to use the apps that I have researched for my app lay out and I might look for some references

### Sketches

#### LOGIN PAGE



This is going to be the design for the login page where 'I'm going to have a picture of my choice on the left and a simple login UI. The Register button overwrites the login frame and then pops up the register window. THE LOGIN gets the username and password check with database if TRUE CARRIES ON the username and password check with .

The image will correspond with the theme that I migh use in the future I might as well add a featuere where they can change the login picture as it would add some customization

Pseudocode-

BEGIN

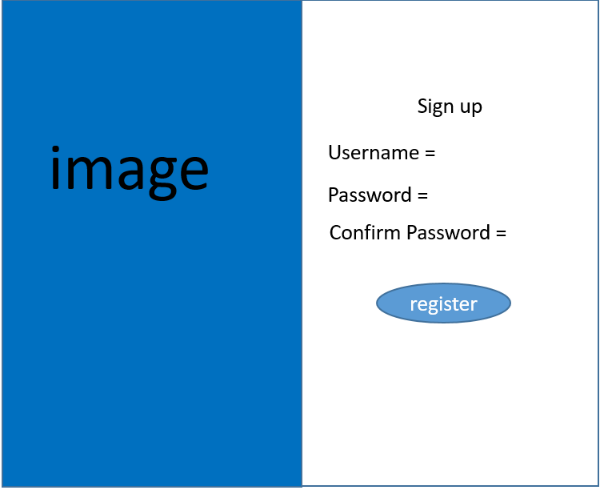
// Display Login Form

DISPLAY\_FORM("LOGIN HERE")

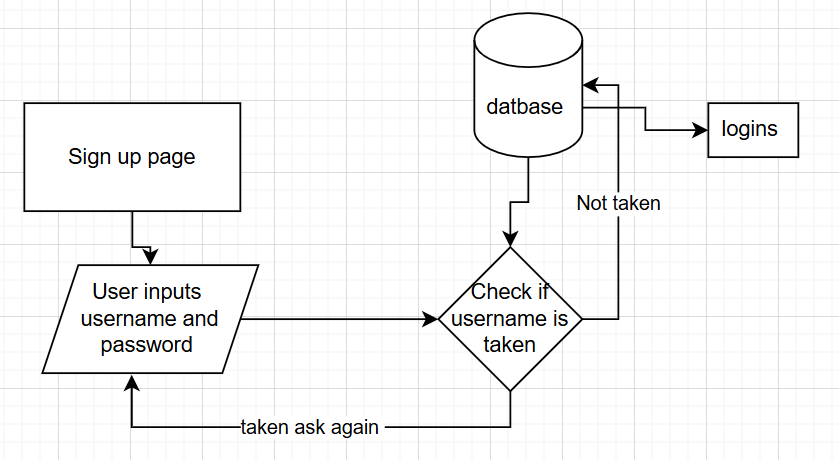
// Input Fields  
username ← INPUT\_FIELD("Username")  
password ← INPUT\_FIELD("Password", HIDDEN)  
  
// Buttons  
login\_button ← CREATE\_BUTTON("Login")  
register\_button ← CREATE\_BUTTON("Register")  
  
// Login function  
ON\_CLICK(login\_button) DO  
 user ← VERIFY\_CREDENTIALS(username, password)  
   
 IF user THEN  
 SHOW\_MESSAGE("Login Successful!")  
 REDIRECT\_TO("Main.py")  
 ELSE  
 SHOW\_ERROR("Invalid credentials")  
 END IF  
END ON\_CLICK  
  
// Register Functionality  
ON\_CLICK(register\_button) DO  
 REDIRECT\_TO("Sign Up Page")  
END ON\_CLICK

END

#### REGISTER –



username (checks if username is greater than 4 characters) if TRUE then check password is greater than 4 characters then check if confirm password is the sane as password then stores the user details into the database. When the registration is complete it will go back to the login page where the user must put their details into login so they can enter the main program



Pseudocode -

BEGIN

DISPLAY\_FORM("Sign Up")

// Input Fields  
username ← INPUT\_FIELD("Username")  
password ← INPUT\_FIELD("Password", HIDDEN)  
confirm\_password ← INPUT\_FIELD("Confirm Password", HIDDEN)  
// Register Button  
register\_button ← CREATE\_BUTTON("Register")  
  
ON\_CLICK(register\_button) DO  
 IF password == confirm\_password THEN  
 // Create new user account  
 new\_user ← CREATE\_USER(username, password)  
 IF new\_user THEN  
 SHOW\_MESSAGE("Registration Successful!")  
 GO\_TO("Login Page")  
 ELSE  
 SHOW\_ERROR("Username already exists")  
 END IF  
 ELSE  
 SHOW\_ERROR("Passwords don't match")  
 END IF  
END ON\_CLICK

END

#### Calendar-

In this page I will put an algorithm so when it runs it will make a timetable it will take data from user-availability and then also get the latest exam score and weight the subject making a timetable for the student.To make a custom calendar I will need to make each individual cells to host the events that will be stored from the database I have to make a loop for the cells until they satisfy the timing . I'm doing this cause custom tkinter / tkinter gives a calendar, but it does have cells or anything to store any events only thing they can do is select date, therefore I must do it the other way which is make my own calendar. The frame should consist of a next week button, previous week button and time at the top allowing for visualization and then the Calander app should be simple and easy to understand instead of being complicated .  
A black and white calendar

AI-generated content may be incorrect.

 The middle is going to display the current time, and the right button allows you to see next week, and the left button allows the user to see the previous week

This is the Calander where the task is displayed. The user can add and edit their task from here. In this place there will be individual cells that allow them to add or edit tasks. In here it will also display the user timetable and display when they meant to study. I may also add a highlight for the current date.

**STEPS -**

**Setup the Interface:**

* Add Title: "Availability Calendar"
* Set up the calendar as a grid with days across the top and times along the left side

(might need a looping algorithm )

**Calendar Grid Structure:**

* Days of the week (MON through SUN) as column headers
* Time slots (7am through 12pm) as row headers
* Create clickable buttons

**Basic Functions:**

* When a user clicks on a cell, highlight it to show selection
* Save function will store which time slots are selected
* Clear function will remove all highlights
* Load function will restore previous selections

Psudo code -

I will use OOP for the programme I will do that as it would easily be opened because the main program will be in OOP and the behaviours and other task that may be done cannot be doesn’t efficiently without OOP. To get the data I will use linear search with OOP to organize the data as the events will be stores in the corresponding rows and column. The calendar will use the grid function to get the exact location of the event.

BEGIN PROGRAM

// Setup basic containers  
time\_data ← EMPTY\_CONTAINER  
day\_labels ← LIST\_OF\_DAY\_NAMES  
time\_labels ← LIST\_OF\_TIME\_INTERVALS  
ui\_elements ← EMPTY\_GRID\_STRUCTURE  
  
// frame work  
DISPLAY\_WINDOW()  
CREATE\_TITLE\_BLOCK()  
  
// Build visual matrix  
SETUP\_GRID\_FRAMEWORK()  
FOR EACH day IN day\_labels DO  
 ADD\_DAY\_MARKER(day)  
END FOR  
FOR EACH time IN time\_labels DO  
 ADD\_TIME\_MARKER(time)  
END FOR  
  
// Generate interactive components  
FOR EACH day IN day\_labels DO  
 FOR EACH time IN time\_labels DO  
 element ← MAKE\_INTERACTIVE\_BOX(day, time)  
 SET\_ELEMENT\_BEHAVIOR(element, TOGGLE\_ACTION)  
 ui\_elements[day][time] ← element  
 END FOR  
END FOR

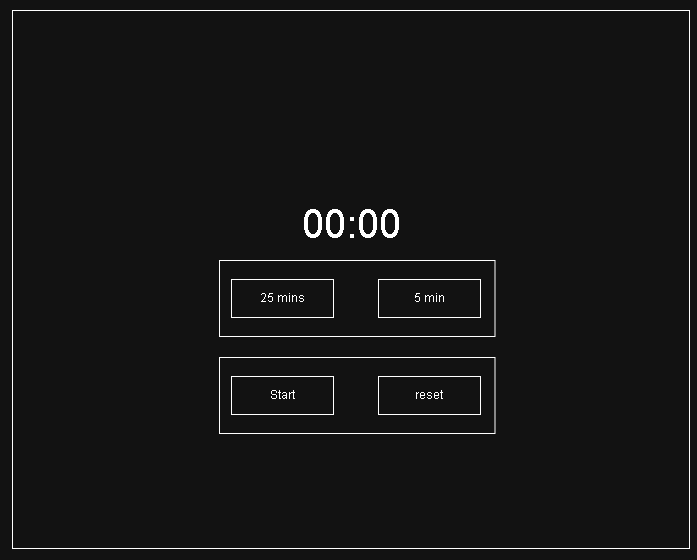
END PROGRAM

#### Pomodoro page –

The Pomodoro page I wanted to be content heavy so that the user will know what happening in the upcoming hours a reminder like system where even if the user is revising, he will not miss the time. I wanted some opinions for the design, so I asked the student named maisha she has said to me that the design is functional but not practical and that is because of the following.

“The different information will attract me more to them instead of helping me focus on my revision and distract me, usually when I study, I want peace and full focus on the subject and if i am in the zone on one subject i may want to focus on the subject.”

With this conclusion I have realized the page should be more clean and less distracting than the current design.

I updated the design to the current design its simpler and more just focused on the timer. No distraction will be given to the person.

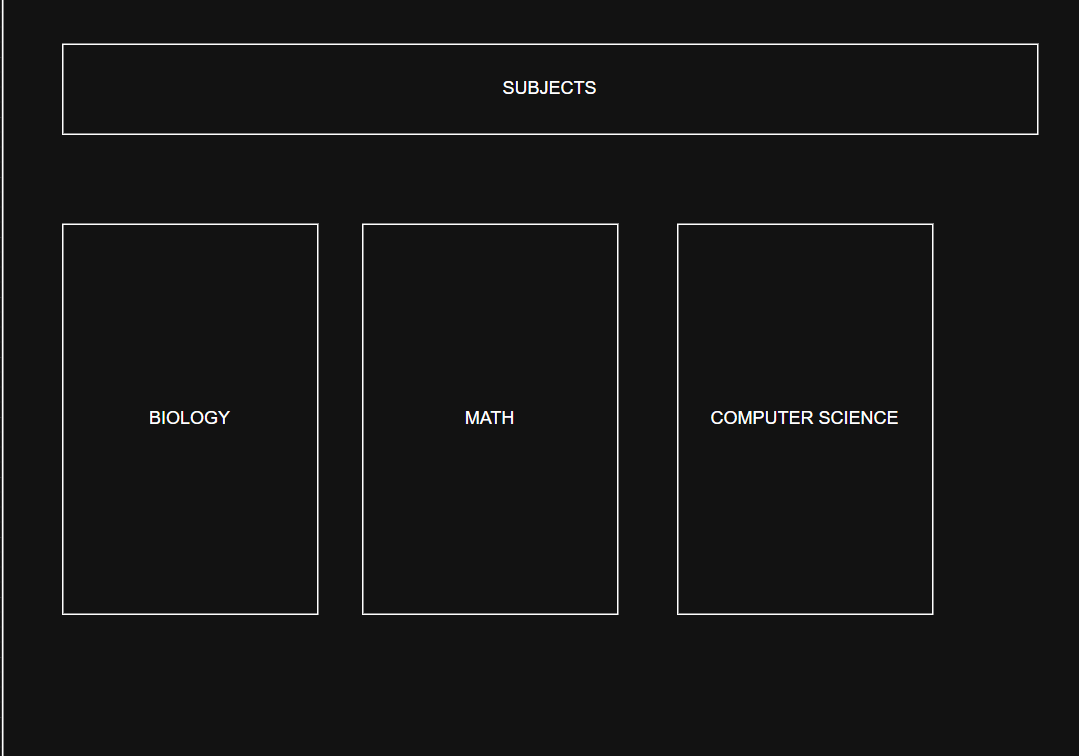
**Psudocode -**

// Initialize   
time\_units ← EMPTY\_COLLECTION  
control\_mechanisms ← EMPTY\_SET  
display\_component ← NULL  
  
// Setup visual   
INIT\_DISPLAY\_SURFACE()  
CREATE\_PRIMARY\_DISPLAY("00:00")  
  
// Configure adjustable parameters  
time\_units["primary"] ← 25  
time\_units["secondary"] ← 5  
  
// Build user control   
FOR EACH unit IN time\_units DO  
 CREATE\_CONFIGURATION\_OPTION(unit)  
END FOR  
  
// Add controls  
control\_mechanisms["activation"] ← CREATE\_TRIGGER("Start")  
control\_mechanisms["termination"] ← CREATE\_TRIGGER("Reset")  
  
// Define operational behaviors  
FUNCTION INITIATE\_SEQUENCE()  
 BEGIN\_COUNTING\_PROCESS(time\_units["primary"])  
 UPDATE\_DISPLAY\_COMPONENT()  
END FUNCTION  
  
FUNCTION HALT\_SEQUENCE()  
 STOP\_ALL\_PROCESSES()  
 RESTORE\_DEFAULT\_DISPLAY()  
END FUNCTION  
  
FUNCTION MODIFY\_PARAMETERS()  
 ADJUST\_CONFIGURABLE\_VALUES(time\_units)  
 REFRESH\_INTERFACE()  
END FUNCTION

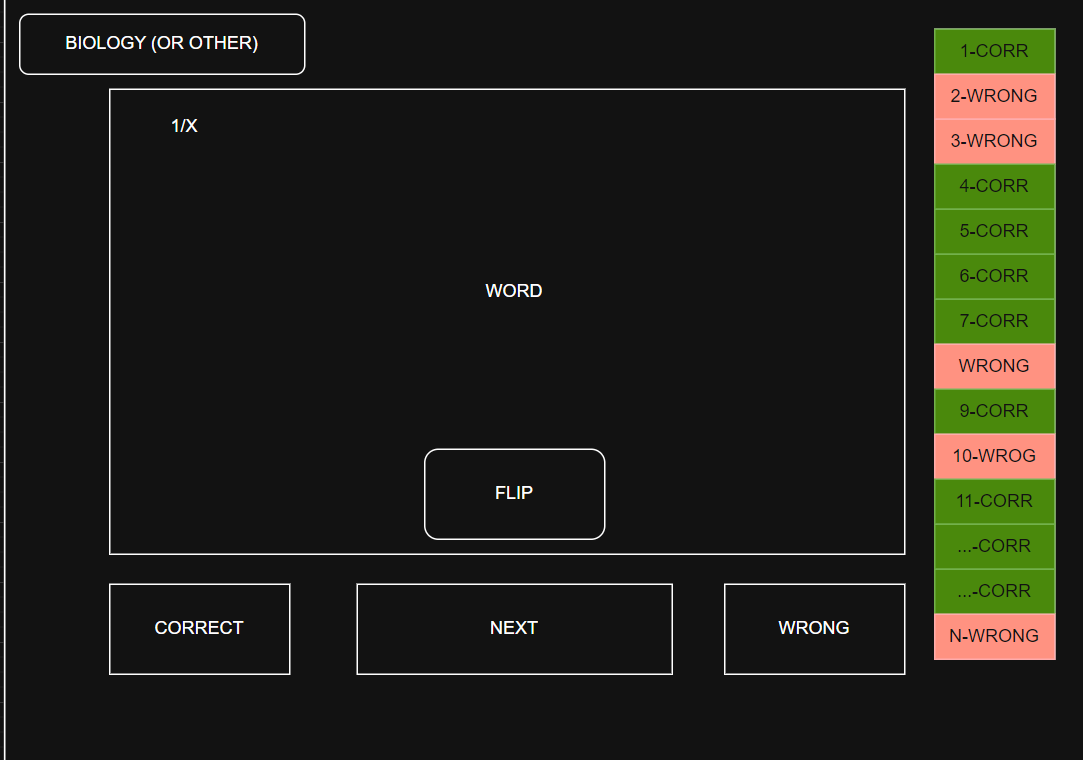
END PROGRAM

#### Flashcards-

This is the base idea for the flashcard application. When the user opens the flashcard, a different subject will appear Infront of the user and if they select let's say Math then all the flashcard

flashcard. The user will be taken to the following screen where they can add flashcard or study with them .

This is the page where the user will be directed to here they can add the name of the flascard and the meaning of the flashcard at the bottom is where the user will displat all their flashcard and also when clicked that when they can delete it and make amends to the card.

After the user press study then they can view their flashcard and of course revise with it. The page will contain a flip button which will allow to see the meaning mark it right or wrong (probably use a Json file) to store the right and wrong answer for that section 

After consideration the right and wrong storer will be overkill and waste of storage as if the user has 100 flashcard it has to show 100 flashcard on the right making them miniscule and can slow the process the down

The flashcard app will need to be reliant on the database as the user needs such data on the go or anywhere, he logs in from. Data collision or confusion can happen with the subject\_name as many users can have the same subject, so I need to unique subject\_id for the user\_id such as user\_id 1 has subjct math subject\_id 2 and user\_id 2 has subject math subject\_id 4.

Reasearch -

<https://youtu.be/eOdbvneI33M?si=eq2DSBxO9TnfR8CV>



<https://youtu.be/EZZ2ChtxzSA?si=g6CzYE0i0AWAIIsa>



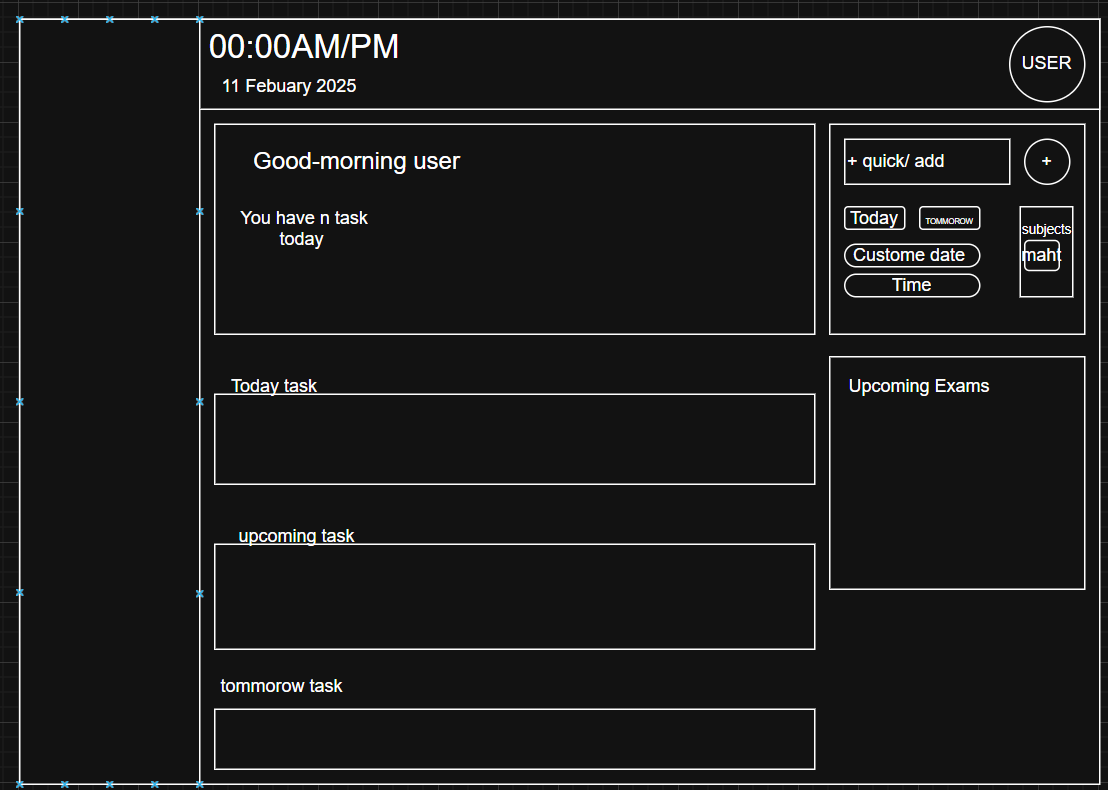
Analysis-  
this is the design for an Indepth a specific subject. On this page it should be the result of the past 3 month and depending on the user feeling about the subject the program will give advice for the user to be able to focus the user can also say how they feel about and what they should do to improve better. In this page it will be user input depended as it for the user support. I might use recursion on each box so that if their is any changes then it will automatically update

#### Homepage-

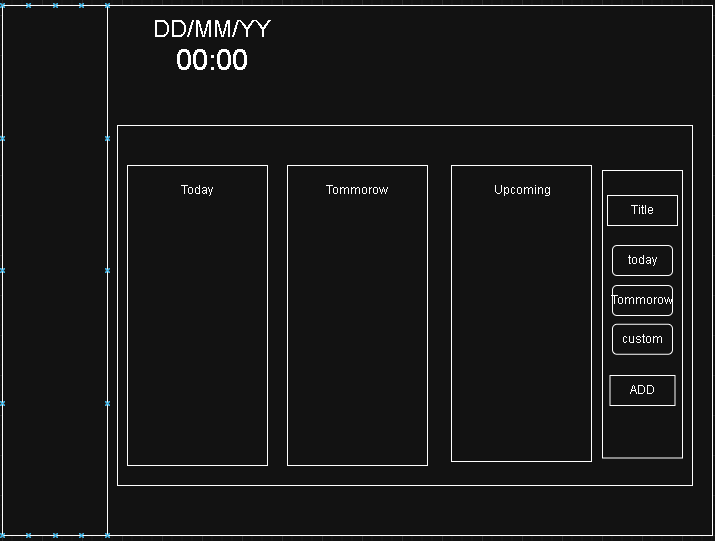
I need a homepage so that when the user logs in, they will receive today task and the time and date I don’t want to bombard them with information they don’t need but want to remind them of the task they are needed to do.

The design should show:

* Today's / Tomorrow / upcoming task task
* A quick add task button
* The user names
* Date and time

The following design hits all the objectives but the today upciming adn tommorow task are a bit annoying to look at and feel like a waste of space . I askesd the user about the design and they said they want a clean design .

I decided to make a new design for the home page, and I decided to make it more pact and more meaningful. These changes also allow me to make a simpler code as the is less frames.

This is the new design. When the user presses the custom button, the Calander will pop up then the user will allow to make an event at any given date. The task will be in the corresponding area.

**Prototype** – use pseudocode to demonstrate

BEGIN

// inilisazatin

FUNCTION main(user)

setup()

load()

show(user)

buttons()

run()

END FUNCTION

// screen attribute

FUNCTION setup()

make\_window()

fullscreen()

END FUNCTION

// load correspon images

FUNCTION load()

get\_images()

END FUNCTION

// load user id and such

FUNCTION show(user)

welcome(user)

END FUNCTION

// create the tabs

FUNCTION buttons()

Possible side bar

END FUNCTION

FUNCTION run()

WHILE true DO

update()

END WHILE

END FUNCTION

END

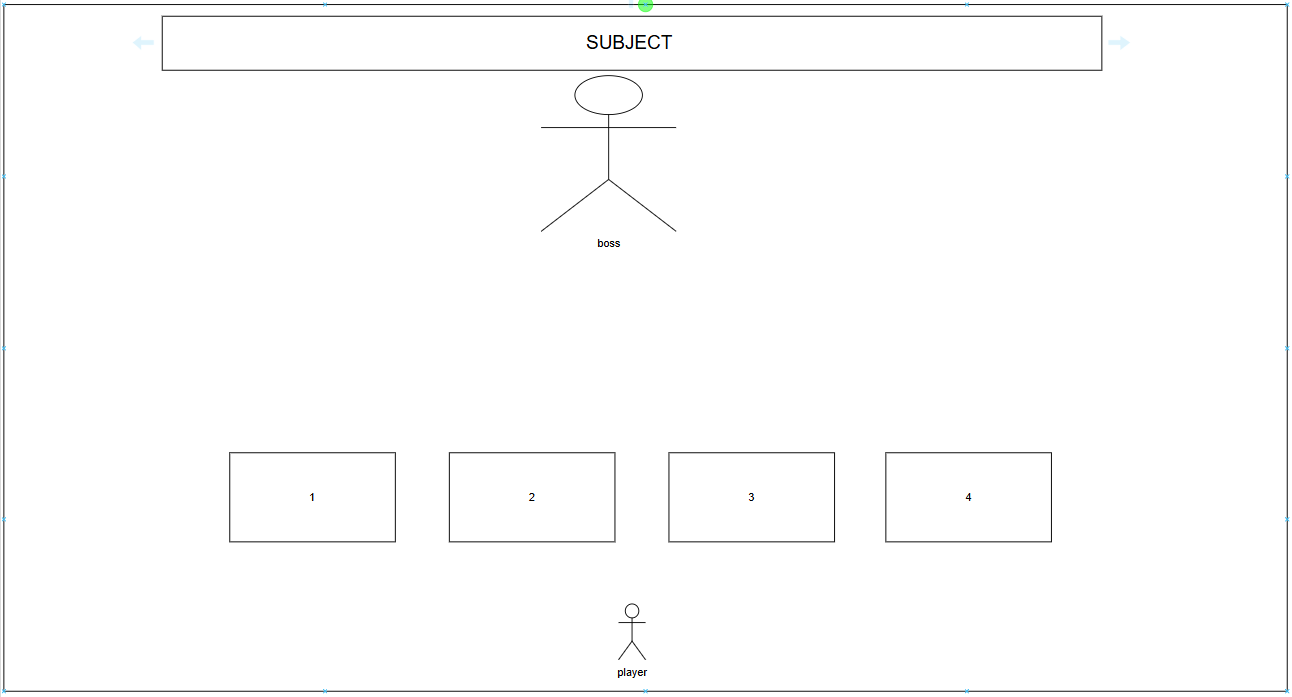
#### Mini Game-

For my objectives I have said I want a minigame which will help the user to have some dopamine to excite them and have a quick revision .

##### Research –

In my research I found different minigames like memory matching where you connect related stuff but it's hard to make the cards flip in CustomTkinter and gets boring once you memorize everything. Word puzzles are easier to code but don't really feel like proper learning. Knowledge trees look cool showing how concepts connect but would be super complicated to build in CustomTkinter. Trivia board games let you move around by answering questions but need too much screen space and would be tricky to code.

After looking at everything I think a quiz combat system makes the most sense. If the user gets an answer right they can dodge attacks and hit the boss, wrong answers mean they take damage. I need to figure out how animations would work with frame sequences in CustomTkinter for movement and attacks. I thought about using PyGame which would be better for animations but that means opening a separate window and I want everything to stay in the main app. This quiz game gives that dopamine hit when you answer correctly and lets users see their progress through health bars while keeping everything in one place.



The part should do :

* Have animation of moving
* Attacking animation
* Have multiple subjects
* Have levels
* Make it look nice and interesting
* Smooth fuctiion
* And make it so that if they get the wrong answer they get punished
* Damage taking animation

For the boss and player -   
i can either make it or use png online , For each level it can update the

##### Flow chart

The flow chart allows me to see the direction the program takes . For example the program lets the user pics the subjects then fecthes the questions and answer from the data storage then display the question with the corresponding possible answers , the program should do some animation to make it more captivating for the user to make them interested of the minigame then this is where the gaem actually start after getting the question and answers the user will choose the option they think it’s the right answers and the program should know if it’s the correct or wrong answer . If its correct then boss takes damage wrong then user takes damage and this keep happening until the boss is 0 or user 0 if boss = 0 then next level if player – 0 restart the game



Pseudo code -

BEGIN

// Game Initialization

current\_level ← 1

player\_health ← 3

boss\_health ←3

questions ← LOAD\_QUESTIONS()

// Main Game Loop  
WHILE player\_health > 0 AND boss\_health > 0 DO  
 // Player Turn  
 current\_question ← GET\_RANDOM\_QUESTION(questions, current\_level)  
 DISPLAY\_QUESTION(current\_question)  
 player\_answer ← GET\_PLAYER\_INPUT()  
  
 // Check Answer  
 IF VERIFY\_ANSWER(current\_question, player\_answer) THEN  
 // Correct Answer - Player Attacks  
 PLAY\_ANIMATION("player\_attack")  
 boss\_damage ← CALCULATE\_DAMAGE(current\_level)  
 boss\_health ← boss\_health - boss\_damage  
 PLAY\_ANIMATION("boss\_takes\_damage")  
   
 // Load Next Question  
 questions ← REMOVE\_QUESTION(questions, current\_question)  
 IF COUNT\_REMAINING\_QUESTIONS(questions) == 0 THEN  
 current\_level ← current\_level + 1  
 questions ← LOAD\_QUESTIONS(current\_level)  
 END IF  
 ELSE  
 // Wrong Answer - Boss Attacks  
 PLAY\_ANIMATION("boss\_attack")  
 player\_damage ← CALCULATE\_BOSS\_DAMAGE(current\_level)  
 player\_health ← player\_health - player\_damage  
 PLAY\_ANIMATION("player\_takes\_damage")  
 END IF  
  
 // Check Game Over Conditions  
 IF boss\_health <= 0 THEN  
 PLAY\_ANIMATION("boss\_defeated")  
 DISPLAY\_MESSAGE("VICTORY!")  
 EXIT  
 ELSE IF player\_health <= 0 THEN  
 PLAY\_ANIMATION("game\_over")  
 DISPLAY\_MESSAGE("GAME OVER")  
 EXIT  
 END IF  
END WHILE

END

Picture -

Because I cannot make my own design, I will use online picture for the boss  
I will use this friendly design as its appealing and nice I got the picutre from the following website. <https://imgbin.com/png/SQvxgssS/mega-man-8-mega-man-6-mega-man-the-power-battle-boss-png.>

There is room for improvement Of course I can pick any picture that i need and the animation I will need to do it by myself and need to learn how to do it .

Reseach -

TO help me make this minigame i decided to do some research as I do not know how to make the game in tkinter . I have found the followign videos to help me . In the fisrt playlist the person makes a space attacker game where the alines move randomly and the user should move away and also shoot . In this I used the animation and moving techniques the video used to make my boss and player move making it more reactive. Then I got help from another video on how to make a quizz on tkinter the first link I followed the flow on how the person made the quizz but I needed the quizz to handle correct and wrong answer on the screen so I found another video to help me handle them and using what I learned I made the coded design .

<https://youtu.be/ocdbsGpu_B4?si=ZE3oMNA76sc-9fW>\_



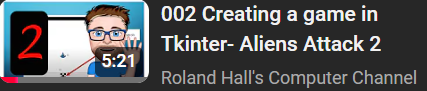
<https://youtu.be/gfV1a3ri1tk?si=wv0eQzGcB4EFmeER>



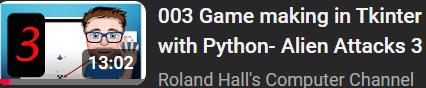
<https://youtu.be/j-r14INfdvM?si=nvkWM3MTEUhbuyic>



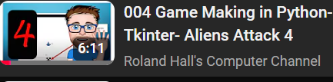
<https://youtu.be/JEIOKz48BgU?si=d5SEfXQM-Ns2xQCL>



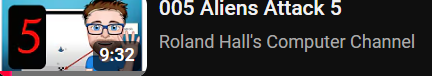
<https://youtu.be/h0hpHbvsXIM?si=TctvYSXhEgNnLvpP>



<https://youtu.be/pdl-kZnEY28?si=IUEar7gWavjU-Osy>



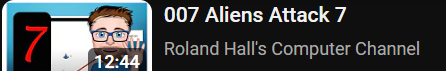
<https://youtu.be/1VfaSKPBMeI?si=tFQQw2Lkd7d5rdLU>



<https://youtu.be/JtoIu9vPYQI?si=YbRWa7HHUz4py5c1>

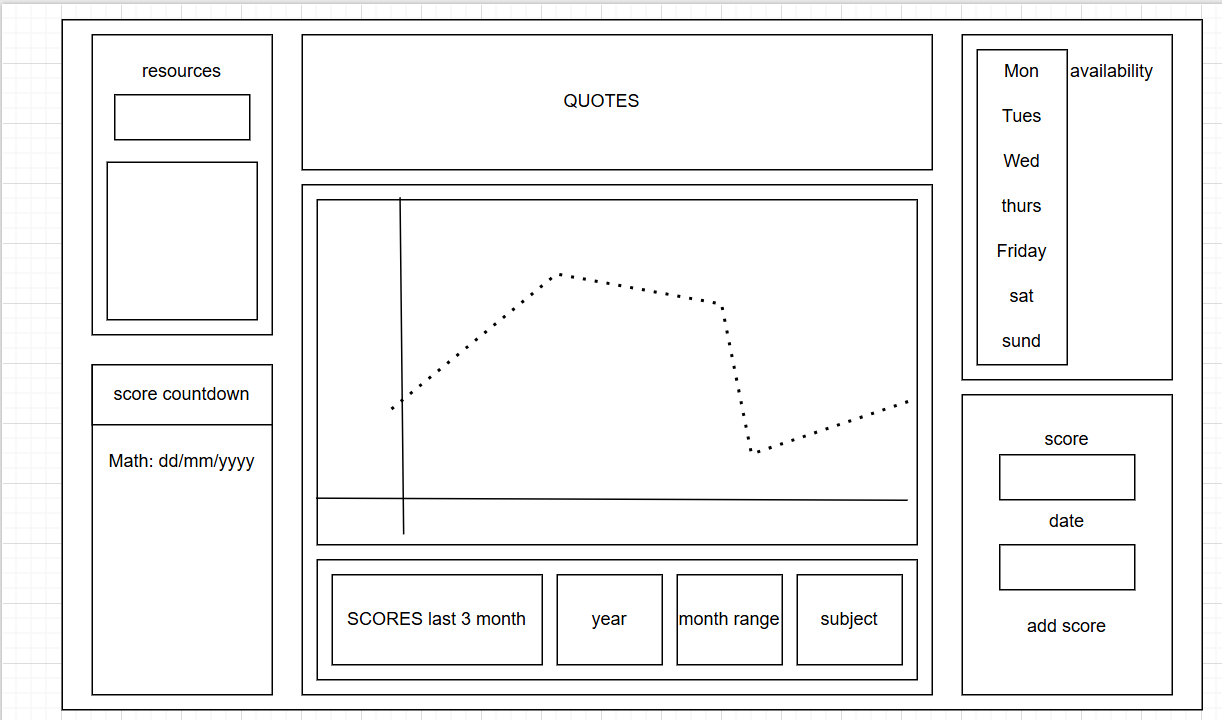


<https://youtu.be/bINdzlbF1W8?si=l6z1_PLlsskq6Jf4>



#### Userpanel-

This is the userpanel area where the user can get resources for their subject solving the problem not knowing where to get revision materials and in the userpanel . Then the user can see their result with the graph the graph should allow the uer to see any years any month range and to see a specific subject or all subjects this allows the user to see their data . then there is the availability section where the user can change their availability . here the user should be able to analyse the specific subject I need to add a button to whre the user can press and goes to the analyse section



##### Plan-

The plan for this side of the project is to make multiple files one named userpanel.py and make this the main backbone this will hold the other programs in their places the userpane will be able to change corresponding to the theme . Then I will have 6 other files , one for the resources which will inherit the frame from usernpanel , one for quotes again inherits the frame and same for , availability , graph,score adding , exam countdown and a button for analysis .

###### Resources –

This part of the code the program will search through the database to find the resource and create a button where it will take it to the website we need research for that on how we can access the internet by clicking a button .

I used this video to help me to code the resources program this shows me how to open website links on tkinter which is what I need . I will store all the link in a json file under their corresponding name .

<https://youtu.be/fPjMgtlsvnc?si=Zma6Vr6FknFE5lyi>

A screenshot of a video

AI-generated content may be incorrect.

###### The exam countodown –

For the exam countdown the user will need to enter the day/month/year its on and then takes that date and minus from today this allows for an accurate time then it will store into a json file . I will need to use the date time library to get the dates to do the calculation

Psudo code –

The psudo code is a prototype of how the calculation will work and how to get the relevant data needed to calculate the days left

BEGIN

TODAY ← GETDATE()

FOR EACH exam IN exams

examDate ← ConvertToDate(exam.date)

daysLeft ← examDate - TODAY

IF daysLeft = 1 THEN

exam.remaining ← "Tomorrow"

ELSE IF daysLeft < 0 THEN

exam.remaining ← "Passed"

ELSE

exam.remaining ← daysLeft + " days left"

END IF

END FOR

FOR EACH exam IN exams

IF exam.remaining = "Passed" THEN

REMOVE exam FROM exams

END IF

END FOR

END

###### Availability –

The availibaility frame will just loop the list until all of the days where accountet for then if the user changes any availability then it will save in the database on the amount of hours but it will also save in a json file .

The json file is to store the visual aspect of the availability as the database will store the amount of hours in that day not the period so the json will just tell the user “this is the time range” and will just show which time to which time they entered . The json file will be also used in calculating the allocation for the scheduel as it will be used to add the subject in the specific time range .

###### Adding scores –

The program will be simple the user will be asked to select from the subjects for example (math , science , tech) after selecing the subject they will be asked to select which month and date and also year after selecting those then the score they received after inputing the score the data will be inserted into the database and shown to the graph .

###### Quotes –

The program will just randomly get a quote and store it in another file properlt named todayQuote and so for this day it will only show the quote tommorow it will show a different quote

PsudoCode-

I made psudo code to represent my logic and how it will only show the specific quote for today

BEGIN

TODAY ← FormatDate(CurrentDate(), "YYYY-MM-DD")

OPEN "last\_quote.dat" FOR READ AS QuoteFile

IF NOT EOF(QuoteFile) THEN

SavedDate ← ReadLine(QuoteFile)

SavedQuote ← ReadLine(QuoteFile)

CLOSE QuoteFile

IF SavedDate = TODAY THEN

RETURN SavedQuote

END IF

ELSE

CLOSE QuoteFile

END IF

# Load quotes and tips from data files

OPEN "quotes.dat" FOR READ AS QuotesFile

QuotesList ← ReadAllLines(QuotesFile)

CLOSE QuotesFile

OPEN "tips.dat" FOR READ AS TipsFile

TipsList ← ReadAllLines(TipsFile)

CLOSE TipsFile

AllMessages ← QuotesList + TipsList

RandomIndex ← RANDOM(0, LENGTH(AllMessages) - 1)

SelectedQuote ← AllMessages[RandomIndex]

OPEN "last\_quote.dat" FOR WRITE AS SaveFile

WriteLine(SaveFile, TODAY)

WriteLine(SaveFile, SelectedQuote)

CLOSE SaveFile

RETURN SelectedQuote

END

### Flow charts –

I am also show how each interaction of the user will go to which part of the program and where the user will go too . The flow chart allows me to see the process that each section can do as I need to see wich app does what and when does it get stores simple function like task viwer will be also made into a flow chart so show the thinking that I might use in the code . Flow chart is a great way for me to keep up with the coding as when I start coding I might lose track on what the code should do .

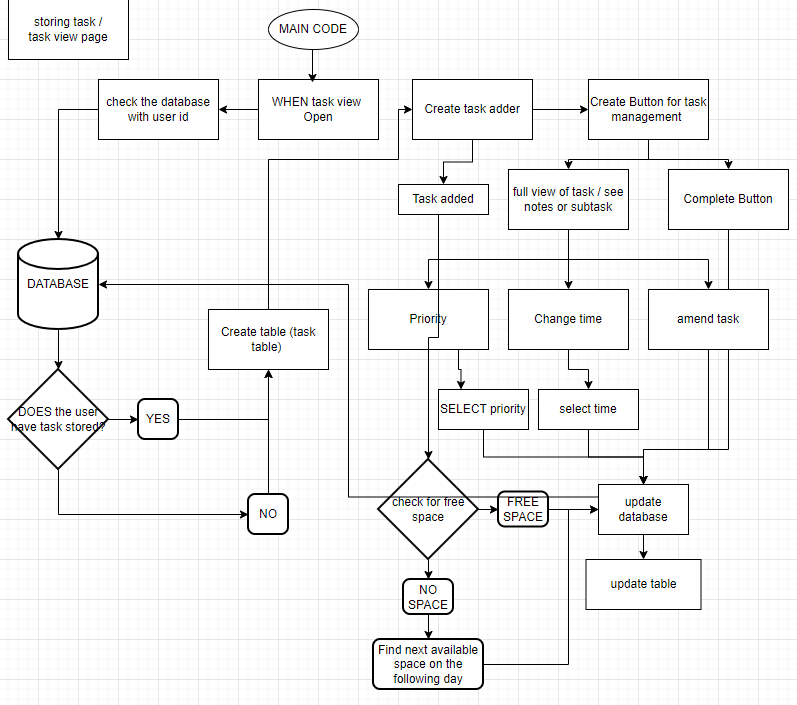
This flow chart is for after the login process and what the program shoud do after the user logs in for the first time or is loging back up .

When the user logins in the user loads the content / data like task and all the stuff then taskes the content to the progress tracker which will display graph and also allow the user to analyze in the said subject . In this page the user will also be allowed to change the availablity setting (the time they want to study) and also they can add new scores to their subject e.tc . Then there is the task viwer this app should get all the events strored in te database and display them also delete when the user completes them . The calander view is the main area where the user will be able to see their schedule that the program made for them and any addition here should transfer to the database and the user in clander view should be able to add task . Timer takes no data it just has controls which the user needs to interact for to work

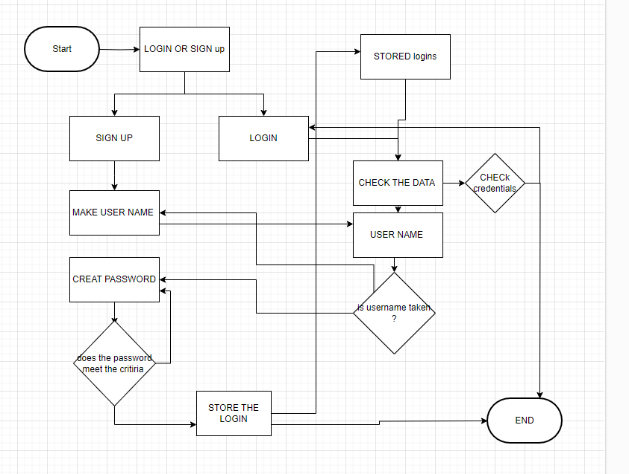
A black screen with white text

Description automatically generated

This is the task viewr app also know as the homepage as when u open the home page taskviwer will come the program should check the database for events if there is events add them to the column but if there is no event just show no data . The taskviewr should have a task adder have different priority levels and allow the tk to change the time . The complete button or a check box should be added so the user can show they completed the said task . When adding task it shouldn’t add task in an area where there is already an event present . So the program should be stopping the user to add more events in the exact day and time . The program should also allow th user to select the priority level to show which is first and which time the user should do the task .



This the flow chart for the Login and sign up page . The Program should check when the person sign up if the username is taken or not and if it not taken then check the username is more then 4 character if thast True then check if the password is more then 4 characters then if the password are the same then store the credential to the database then the main program statrs



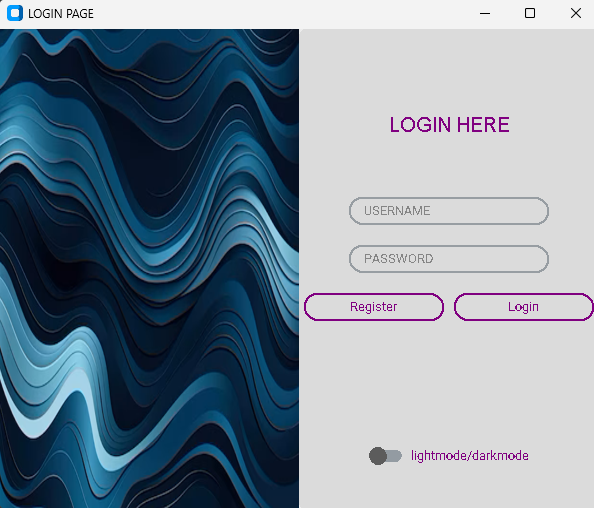
### CODED DESIGN

#### Login

Here I have coded the design to see how it looks there is no function but just place holders for the main code function and all of them will be implemented later as I might need to make Chages to the ui and it will be hard to do the function while changing the UI .  
The Login system is simple to code. It has 2 frames a left and a right, the left for pictures and the right for the login inputs and login button and signup buttons.   
the problem I encountered is that the login button when you go full screen, they also move making it annoying so when I'm making the code I will make another frame for the buttons and the inputs so it will stay in the structure.

##### User feedback-

Add more words like subtitles make the register button smaller than this one so i dont get confused between register and login and add a little info at the bottom.



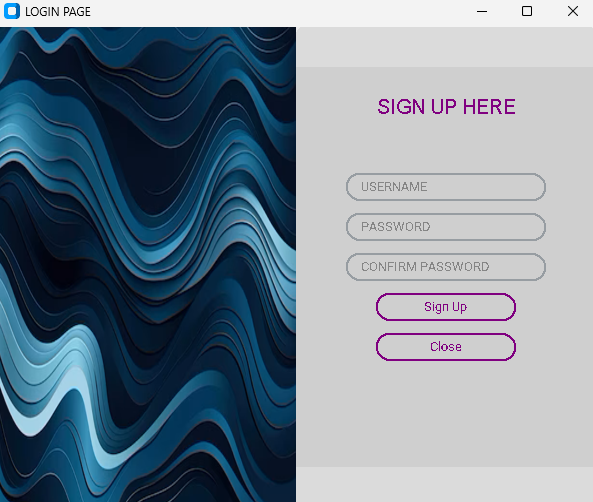
#### SignUp

To make the sign-up page simple, all you do is press the register button which takes you to the function which creates a new frame on top of the right frame overlapping the login inputs which allows the user not to confuse the login and signup.

I did this by using pack in custom tkinter and when the user successfully signups then the signup button will produce a successful signup and then destroy the signup frame to remove it and go back to the login page.

##### UserFeedback -

The sign-up frame looks unnatural, and the rounded boxes are also a bit weird I do not know what it is but feel weird. I feel like that you should make a smaller frame and has the full focus on the signup. Maybe change the color to a dark scheme and make it more like a rounded rectangle



This is the notification box that I'm using for the user to know if they logged in or they put in the incorrect credential this is also for the signup as if they do not meet a specific word lenght then they get an error .

Instead of using a notification box from tkinter libary I used placed a frame in the middle and put the notification in the middle I did this to make it look appealing for the user .

It follows the same logic as the signup page but instrad of packing it I have to use place (place is where I can place any element in the window (0.5,0.5) is the middle )

UserFeedback-

Make it more interactive so like the box is red for errors and green for success and the fact that its in the middle and you must click ok just to retry it gets annoying so can you make the box at the bottom and make it disappear after 3-5 seconds and when u login succesfully do like a loadign screen animation . It will feel more complete and nice to use



Research -

I found this video which gave me idea to how to lay out my login page but this is actually a great tutorial for custom tkinter as it goes through the basics of custom tkinter . helping me making this design

<https://youtu.be/Miydkti_QVE?si=gXQ1FJnEwy42ZEDp>

A black and white sign with white text

AI-generated content may be incorrect.

#### HomePage-

This is the homepage, as soon as the login is successful the user gets into the home page the top is the date and time and the bottom is the task and I have added a task to see if the creation work I also added the quick add button to see how it looks .

I did this by making 2 different files, So I don’t get confused.

The backbone for for the page is called home\_page.py this is where the looks and aesthetics can be changed, such as colors time positions welcome position, the welcome text e.t.c

The main function to see the task is the taskviwer.py The program gets the task and then filters them to the corrects days such as if the day is 12/01/2025 and the tasks are :

12/01/2025 (Math) and 13/01/2025 (Tech).

It will show Math on today Colum and Tech on the tomorrow column and any other task will be in the upcoming task .The filter system is simple get the today date then tomorrow = today +1 , we use this values in a if statement then it puts it in the following list.

Another frame is in the task\_viwer this is for the quick add which is on the right of the 3 column. The add frame will store to the database , and the custom date will pop up a calender that you can select what date to store the event on a drpo down menu also for the time.

I connected the taskviewr with homepage using oop inheritance what i did was that the task view will inherit the frame location from the homepage and create at the set location .

I split the files like this as it the taskviwer will has a complex fecth and filtering process and also adding the task and having the function in the same file as the homepage it will clutter the program making it harder for changes in the future .

So I have split the files for further development in the future adding more function if I want too

##### Code -

This code is my code that i made for the testing session.

Datetime – is a libary that gets the date and time from the system (the device the user is on )

Allowing to get an accurate details .

I used this in the program to get todays date for the filtered system and timedelta is to get time difference / date difference / even second's difference. This allows me to get tomorrow's date accurately.

tdy= datetime.today().date()

tmr= tdy + timedelta(days=1)

self.filt\_tasks = {"Today": [], "Tomorrow": [], "Upcoming": []}

for task in self.tasks:

task\_date = datetime.strptime(task["date"], "%Y-%m-%d").date()

if task\_date == tdy:

self.filt\_tasks["Today"].append(task)

elif task\_date == tmr:

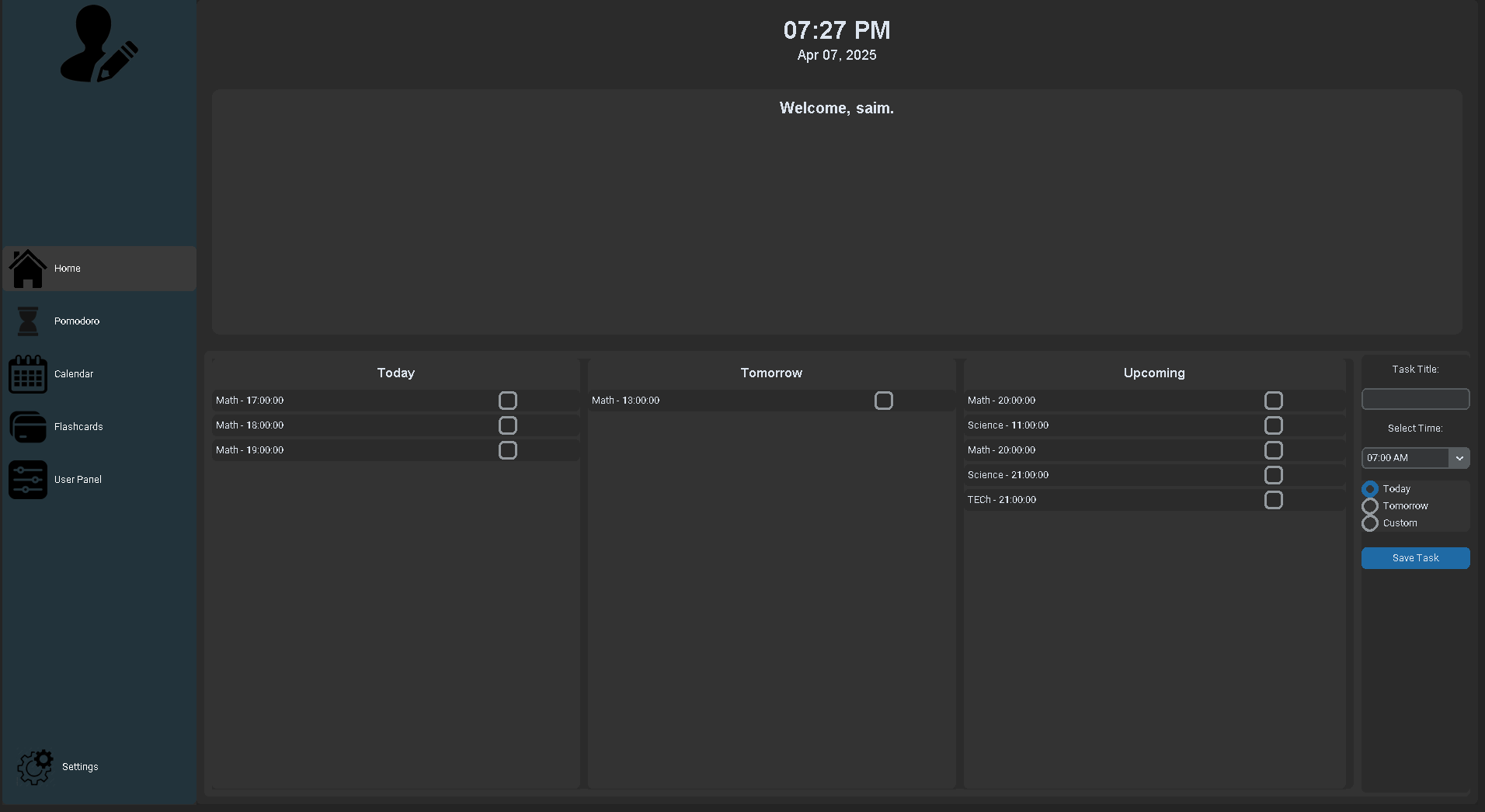
self.filtered\_tasks["Tomorrow"].append(task)

else:

self.filtered\_tasks["Upcoming"].append(task)

##### USER FEEDBACK-

The design is nice but i feel like you can make the welcome more dynamic like adding good morning , good evening and good afternoon depending on the hours. And i feel like it is too dark. Can you add some color scheme to it and maybe make it more appealing and make each column the same size so even though there is a task it will not expand or shrink the other columns. And maybe make the task viewer a bit bigger and centered.



#### POMODORO TIMER -

In this page I have implemented the timer option and how it changed to stop , pause . The time selection changes by making another frame on top of the other frame .

The timer function is simple (or countdown function)-

The function keep calling itself after (1000 ) which is 1 second in python and then subtract from remaining seconds variable I have this variable so that the timer gets updated and also store (its not store in a stack or list just store in the variable ) when its pause and after subtracting from the variable seconds\_remaining its gets divmod to convert the seconds to minutes and also seconds for display . The only way for the timer to go down the variabe running = True if it False it doesn't countdown this helps to pause the timer the running variable is a global variable

##### CODE-

def timer(remaining\_sec):

global running, time

time = remaining\_sec # Keep track of remaining time

if running:

mins, secs = divmod(remaining\_sec, 60) # timer

timer = f'{mins:02}:{secs:02}'

timer\_labl.configure(text=timer)

if remaining\_sec > 0:

root.after(1000, timer, remaining\_sec - 1)

else:

timer\_labl.configure(text="Time's up!") # timer finished

running = False

##### User feedback -

I do not like how empty is feels could you also add a bar that fills up or something and change the color of the buttons to make it more friendly and more welcoming right now it just looks so boring and maybe make the timer bigger so it easier to see and please make a differentiation between the study time and break time as its confusing which thing Im doing when .

##### Reasearch –

I used this small video for the timer function it only tells me how the timer will count down

And how the divmod can be used to display the text .  
I will properly play around it making it more suitable for my program there is another video which I will use for the looping mechanism

<https://youtu.be/1g65KSKyqZw?si=DlFckcHh6dvTWwRs>

A screenshot of a computer

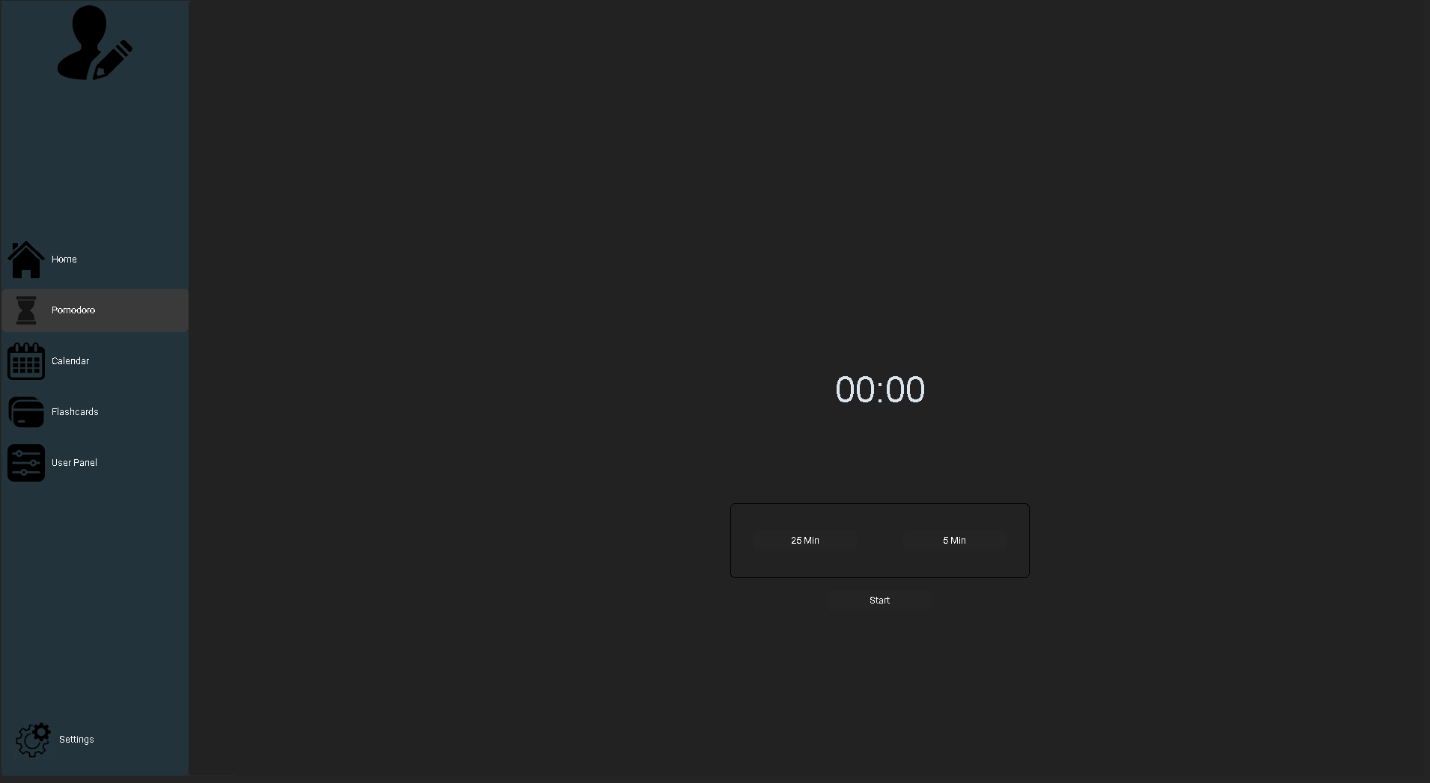
AI-generated content may be incorrect.

<https://youtu.be/Au2De-mYaXw?si=alBVcQaf_SLY4PUI>

A screen shot of a phone

AI-generated content may be incorrect.

##### UI-POMODORO



#### Calander -

This was the most challenging process to make as I have to make each single button (cells) with a loop that feels it out . Each cell is a custom tkinter button due to the reason that the user should be able to edit / add any event to the corresponding cells and store it to the database fetching the date and time is corresponding to the row and column and converting that to 24 hr format and also a date/month/year format .

I solved this by -

* Taking the start of the week and then offsetting it corresponding to the column number so if today is Monday then offset =0 but if its friday = 4 and gets the date using this
* For hour format I created a list that holds all the value of the house so the row its on its corresponds to the list so if its 9:00 AM return = 2 which is the list is 9:00 AM then the code take the 9:00 am(12hr format) and converts it to 9:00:00 (24 hr format)
* Then saving these details to the Database I am converting to 24 hr format as it will be easier to store in the database as IF i store it in 12 hour format i will need a special case to store it as well

##### Code -

Ima show the key code make the design

# Days and time slots

self.days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

self.hours = [

"07:00 AM", "08:00 AM", "09:00 AM", "10:00 AM", "11:00 AM",

"12:00 PM", "01:00 PM", "02:00 PM", "03:00 PM", "04:00 PM",

"05:00 PM", "06:00 PM", "07:00 PM", "08:00 PM", "09:00 PM",

"10:00 PM", "11:00 PM", "12:00 AM"

]

# Get today's date and determine the day index (0 = Monday, 6 = Sunday)

today = datetime.datetime.today()

self.today\_day = today.weekday() # Monday = 0, Sunday = 6

self.today\_date = today.date() # Current date for events

# Current week start date

today = datetime.datetime.today()

self.start\_of\_week = today - datetime.timedelta(days=today.weekday())

# Create the header for the days with dates

for col, day in enumerate(self.days):

day\_date = self.start\_of\_week + datetime.timedelta(days=col)

header\_text = f"{day}\n{day\_date.day}"

# The day of the week properties

header\_label = ctk.CTkLabel(master=self.calendar\_frame,text=header\_text,fg\_color="#333", text\_color="white",font=("Arial", 12, "bold"),width=12,height=2,)

header\_label.grid(row=0, column=col + 1, padx=2, pady=2)

# Create the time slots for the calendar

for row, hour in enumerate(self.hours):

time\_label = ctk.CTkLabel(master=self.calendar\_frame,text=hour,fg\_color="#444",text\_color="white",font=("Arial", 10),width=12,height=2,)

time\_label.grid(row=row + 1, column=0, padx=2, pady=2)

# Configure grid to make cells resize proportionally

for col in range(8):

self.calendar\_frame.grid\_columnconfigure(col, weight=1, uniform="equal")

for row in range(1, len(self.hours) + 1):

self.calendar\_frame.grid\_rowconfigure(row, weight=1, uniform="equal")

# Create the event cells (empty initially)

self.event\_cells = {}

# Loops to create all cells for the calendar

for row in range(1, len(self.hours) + 1):

for col in range(len(self.days)):

# Highlight today's cells with different color

is\_today = (col == self.today\_day and

(self.strt\_of\_week + datetime.timedelta(days=col)).date() == self.today\_date)

event\_frm = ctk.CTkFrame(master=self.calendar\_frm,width=120,height=50,corner\_radius=5,fg\_color="#0C2D48" if is\_today else "#333",)

event\_frm.grid(row=row, column=col + 1, padx=2, pady=2, sticky="nsew")

# Bind click event to open event editing

event\_frm.bind("<Button-1>", lambda event, r=row - 1, c=col: def add\_edit\_event(self, row, col):

event\_date = (self.start\_of\_week + datetime.timedelta(days=col)).date()

event\_time = self.hours[row]

# Convert event\_time to 24-hour format (for database storage)

event\_time\_24hr = datetime.datetime.strptime(event\_time, "%I:%M %p").strftime("%H:%M:%S")

print(f"Selected: Date: {event\_date}, Time: {event\_time} ({event\_time\_24hr})")

#Store(event\_date,event\_time\_24hr)

##### UserFeedback-

The calander is not fully filling the spaces and making it look weird teh design is clean and small but I feel like if the calander was expanded to fill the window it will be better . Also, when testing this out If I keep going to the right to look at my future events, I must go all the way back to today and i can also miss and go too far back making is super annoying . In the future can you add a button to quickly go back and can you fully expand the calender to fit the whole screen and also color scheme looks weird it doesn’t have an appeal to it

##### Personal feedback -

When i ran the program, I seen that when i wanted to see the saved event it had mixed outputs, and the displayed events sometimes become varied and details that i did not want. I would understand that if it was database error but in this stage the events are stored in a json file which mimics a database but even, so the problem is adding and edit function as there is a problem with storing the event in the list. And also, a buggy editing event after editing it doesn’t change the event in the Calander even though it says it was changed . Again could be the variables maybe

##### Reasearch -

The research I have used to make the Calander I liked how the person loop to create the buttons and when the person select said button it return a value which the program can use to store , I also took the idea of next week and previouse week from this video and also changing the button configuration when the event is there .   
I have followed the way the perosn makes the code but I have not copied it word by word I only used the loop to make the buttons , also the changes when event is present and then how next week and previouse weeks changes .

<https://youtu.be/b_5iKyI1bJw?si=mY_lyPGtIc9v75IV>



A screenshot of a computer

AI-generated content may be incorrect.

#### Flashcard-

The flashcard app / program is made out 3 distinct section. Main menu which is the area where the user will see the subjects they want to see the flashcards for and when they press the subjects it will only show that subject flashcard . Then we have the adding/ view section this section is where the user can either add another flashcard onto the subject or start the flashcard to start revising for the subject . Then we are taken to flashcard view this si where the user can see their flashcard at first it will show the front meaning and then when the user press the flip button it will flip the flashcard. The user can also delete the flashcard or go to the next flashcard . This application is database heavy and takes the values across the program .

In this program is the first I decided to connect to the database too as a file structure would properly simulate the effects it needs to do.

##### MainMenu –

In this first page I needed to use the database and fetch all subjects with the corresponding user\_id after executing the sql code we then take the result to a loop where it creates ctkButton for each subject\_name as ,you can see in the picture then the presses one of the buttons which It will take to the next window by calling the function flahcard and it takes the subject\_name with it .

##### SQL query –

"SELECT DISTINCT subject\_name FROM subjects WHERE user\_id=%s",(self.user\_id)

##### CODE –

To make this happen we fetch the subject from the database then we clear any buttons on the frame then we create the frame to contain the subjects then we create the subjects button to select from .

def create\_flashcards():

unique\_subjects = fecth\_subj\_from\_db()

if not unique\_subjects:

messagebox.showinfo("Info", "No subjects available.")

return

for widget in self.button\_frm.winfo\_children():

widget.destroy()

for subject in unique\_subjects:

btn = ctk.CTkButton(self.button\_frm, text=subject,height=400,width=300,

command=lambda subject=subject: open\_subject\_flashcards(subject))

btn.pack(padx=10,pady=5,side="left")

# Create a frame for buttons

self.button\_frm = ctk.CTkFrame(self,width=500)

self.button\_frm.place(relx=0.5, rely=0.5, anchor="center")

create\_flashcards()

#### Add/view window –

When this window is opened first it will go through the SQL statement get every flashcard on the subject\_name and display it. The user will be able to add flashcards in this section and see their flashcard as well and then when they press flip to add meaning they another box gets added and they can add the flashcard to the database. The flashcard is stored like this the word is stored as front and the and the meaning as back which can be used for the viewing flashcard program.

##### SQL query -

SELECT DISTINCT flashcards.front FROM subjects JOIN flashcards On subjects.subject\_id=flashcards.subject\_id WHERE subjects.subject\_name=%s

#### Viewing window -

This window is where ethe user actually use their flashcard they have stored what the program does takes the all of the flashcards with the subject\_name and then stored in a list called flashcards in to flip the cards we can use a BOOLEAN operation to track if the user want to show the word or the meaning of the word . We can also delete the flashcard we do this by taking the front the word and using that with a WHERE clause and delete it .

##### SQL query –

SELECT flashcards.front AS word, flashcards.back AS meaning FROM subjects JOIN flashcards ON subjects.subject\_id = flashcards.subject\_id WHERE subjects.subject\_name = %s

##### Problem –

If I have multiple user with the same subject they will show both user flashcards which I do not want that and the other problem is if I want to delete the card if there is multiple meaning for one words it deleted all of the instances for example aggregation in computer science relates to OOP but in math it measn combining multiple values in a single value , they have the same name but different meaning and with the current code it deletes from all the time it has that name .

But the aesthetic function of getting and display the function was met making it alright to take ti forwards and just fix the bugs. to make it better I will need to add a flashcard unique id so it deletes that flashcard and no other flashcard and I have to also use the user\_id to get only the user can see their own flashcard

##### UserFeedback-

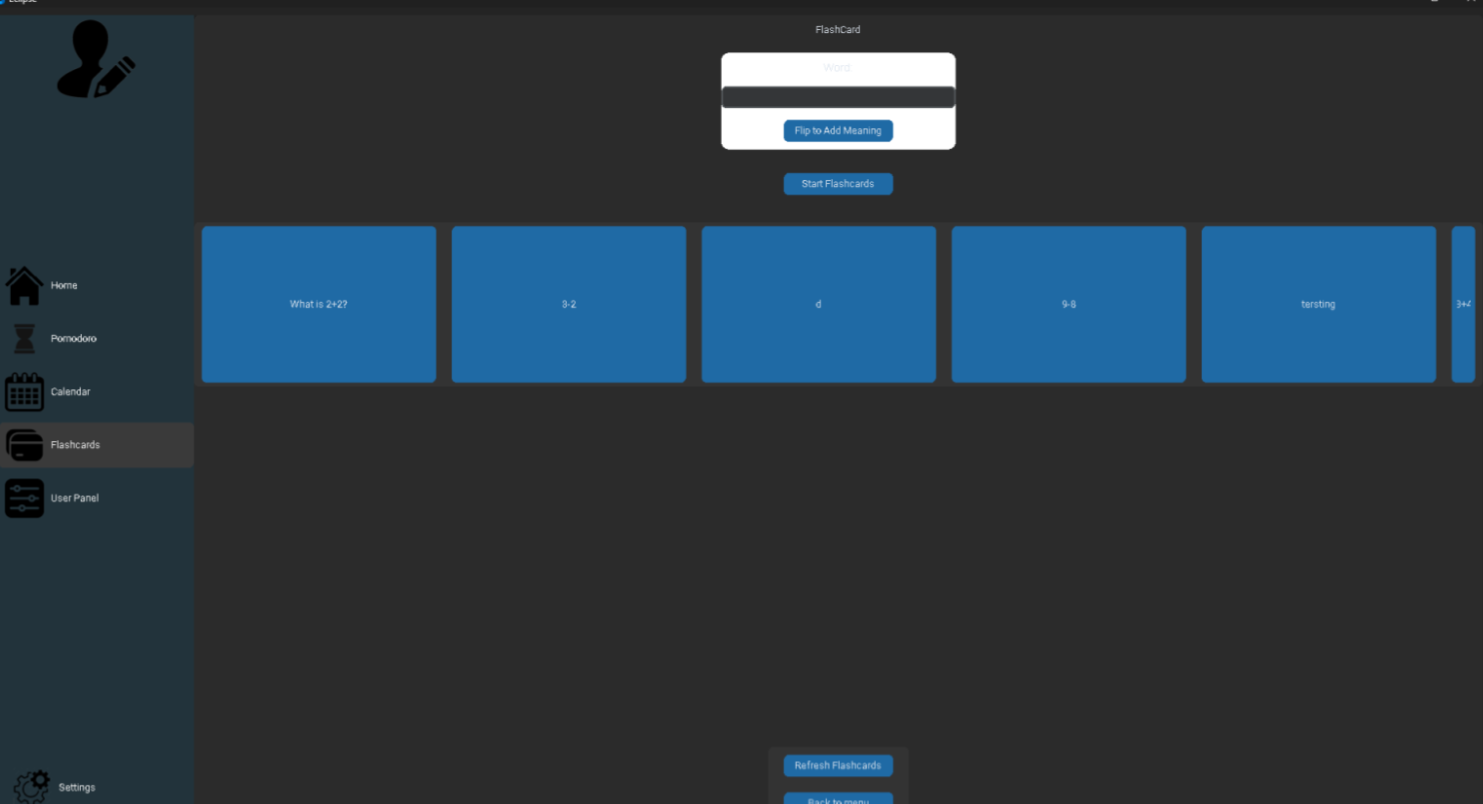
The page has no welcome it look boring and simple but I do like the concept that you are going forwards with the design is really nice and unique but maybe add more of color scheme to it and maybe make it feel like more of a menu page and have some stats like maybe the amount of flashcard the user has stored. But I do like the idea of how when I select the subject it shows all the selected subject instead of having all of the flashcard like other app do . But when I went to add the flashcard the box was to small to add a detailed flashcard and it was weid the formatting wasn’t optimal or the slightest bit aesthetical. If you could can you make the flashcard add a bit bigger and the back button to the top left and also starting the flashcard the button really doesn’t stand out making me confused where I needed to press to start viewing the flashcard . on the other hand when I did go to the viewing of flashcard the actual viewing is big and immersive in some kind of way it is really good lay out but maybe separate the control buttons from the flashcard and then take the back button all the way to the top left . Also could you change the colors for delete , flip , and next card to something like red , purple and then blue or something like that .

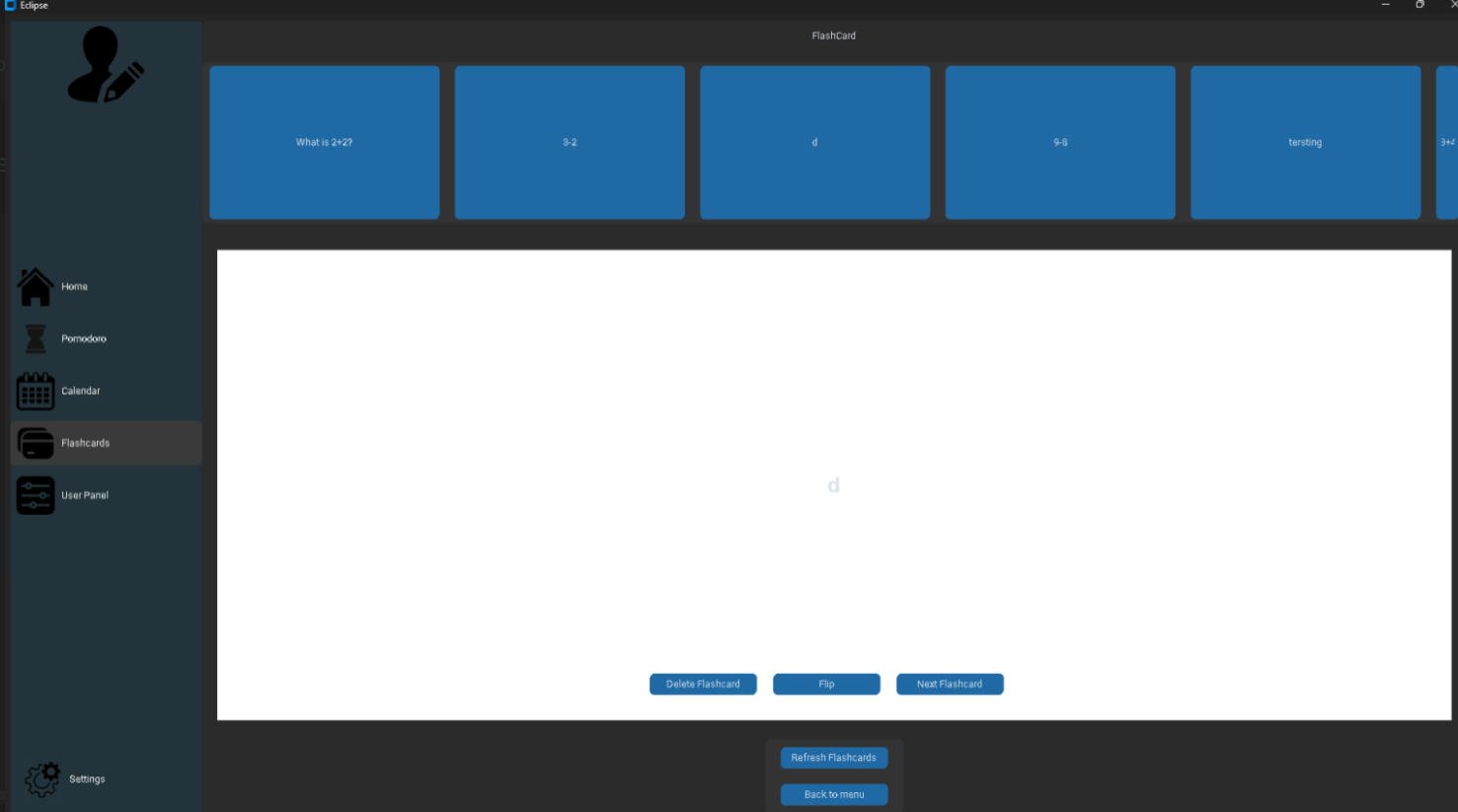
##### Research –

I have done some research on how I could make the flahcard how they flip or like how will it function , and I have found some videos to actually help me I have taken the functionality of the code but not fully copied the code I used it as a start for my flashcard <https://youtu.be/eOdbvneI33M?si=o4n4WHOlrlbZjmTC>

A screenshot of a computer

AI-generated content may be incorrect.





#### Userpanel -

This is the userpanel for the code , the userpanel main goal is like a studyhelper where it has resources visualization for their scores and exam countdow , availability and also an analysis . The usepanel is composed is many programs and the userpanel is just a skeleton that holds all the other codes .

##### Resource -

The resources program is where they can search resources that are store in the dataset, I made a simple search enter box and then takes all the values with that subject name and then loops it through to make a frame to dipsly the title stored then the describtion will also display then a link button will be made that connect the resources with the website

##### Availability -

To make availability I have an array that stores the days and loop through to make the labe and give the corresponding entry box

###### Code –

frm = ctk.CTkFrame(self)

frm.pack(padx=10, pady=10, fill="both", expand=True)

ctk.CTkLabel(frm, text="Enter availability (e.g., 9-12, 14-16 or 0 for no availability):",

font=("Arial", 12, "bold")).pack()

for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]:

row\_frame = ctk.CTkFrame(frm)

row\_frame.pack(fill="x", pady=2)

ctk.CTkLabel(row\_frame, text=day, font=("Arial", 10, "bold"), width=10).pack(side="left", padx=5)

entry = ctk.CTkEntry(row\_frame, width=150, height=10)

Problem -

##### Examcountdown-

This is the section where the user enters an exam date and it does small calculations to display the user the remaining days to the exam . we do this by getting todays date and the selected date and get the different between the dates .

###### Code -

for exam in sorted(exams, key=lambda x: datetime.strptime(x["exam\_date"], "%d-%m-%Y")):

exam\_day = datetime.strptime(exam["exam\_date"], "%d-%m-%Y")

days\_left = (exam\_day - today).days

countdown\_text += f"• {exam['subject']} - {exam['exam\_date']} " \

f"({'Tomorrow' if days\_left == 1 else f'{days\_left} days left'})\n"

##### Graph –

For the graph aspect I have used matploitlib to make the graph as its very versatile as it can have many different type of showing the result for example it can create a line graph , a bar graph , a scatter graph , and also a pie graph , this allows me to get a customizable range of option to show the user their result. But I decided to stick with the line graph as is more user friendly and less cluttered then the other graph .

###### Research –

I have also made a control panel to let the user control the time range and year range wand which subject to focus on . Because I have never made graph I have find some good videos on how to use matplotlib One of the video helps me understand how the library work to visualize while the other shows me how to control the graph

<https://youtu.be/xqkNWkV_D2o?si=tn2Dhli5G7w0dQR4>

<https://youtu.be/p-xJsc6LSx0?si=k-BjA4z86ZkvtSJU>

##### Scores-

This program allows to enter to the user score on their selected date and it will aso show to the graph for now I have simple json file structure to see if the inputed values get saves .

#### MiniGame –

I decided to go with a quizz action where the user gets the right answers and attacks the boss until it dies giving them a sense of accomplishment . I decided to use tree traversal for the minigame quiz element as then I can control the difficulty criteria and what question can come up for example if the plater get 3 answers right they go to medium question but if they get 2 asnwers wrong then they do not go to the medium question .

##### Tree traversal –

The program uses a tree traversal system where it allows me the developer to control the flow of the level difficulty as each node represent a question and depending on the answer they will get that question for example if a questions where [2+2 , 3-2 , 45- 5,] and the question was [2+2] and the user puts in 4 then the question [2+2] will not show again but if the user gets the answer wrong it will not remove 2+2 but it will move to the next question [3-2] and when the other question in the section is completed then it will return back to [2+2] to let the user try again .

How it works simple –

* Root Node : the starting difficulty (easy queston)
* Correct answers : goes to the right and cariees on with the question
* Wrong answer : carries on to the next set of question but it will return back to the question they got wrong

CODE -   
I coded the idea and make simple ui to go with it to see how it would work



##### User feeback –

“Can you make it more bigger it doesn’t fill the screen making it a bit akward and everythgi else is fine but I do want to select the subject I want to be quizzed on could you add that feature . Also can you have player and boss images boxes look boring ”

##### Updated UI –

I took the user consideration into account and made it so that the user can select their subject I also added images to the boss and player , and I have made it so that the screen looks full not empty like before

A screenshot of a game

AI-generated content may be incorrect.

##### Research –

TO help me make this minigame i decided to do some research as I do not know how to make the game in tkinter . I have found the followign videos to help me . In the fisrt playlist the person makes a space attacker game where the alines move randomly and the user should move away and also shoot . In this I used the animation and moving techniques the video used to make my boss and player move making it more reactive. Then I got help from another video on how to make a quizz on tkinter the first link I followed the flow on how the person made the quizz but I needed the quizz to handle correct and wrong answer on the screen so I found another video to help me handle them and using what I learned I made the coded design .

<https://youtu.be/ocdbsGpu_B4?si=ZE3oMNA76sc-9fW>\_

A screenshot of a computer

AI-generated content may be incorrect.

<https://youtu.be/gfV1a3ri1tk?si=wv0eQzGcB4EFmeER>

A black background with white text

AI-generated content may be incorrect.

<https://youtu.be/j-r14INfdvM?si=nvkWM3MTEUhbuyic>

A close up of a sign

AI-generated content may be incorrect.

<https://youtu.be/JEIOKz48BgU?si=d5SEfXQM-Ns2xQCL>

A black background with white text

AI-generated content may be incorrect.

<https://youtu.be/h0hpHbvsXIM?si=TctvYSXhEgNnLvpP>

A black background with white text

AI-generated content may be incorrect.

<https://youtu.be/pdl-kZnEY28?si=IUEar7gWavjU-Osy>

A screenshot of a computer

AI-generated content may be incorrect.

<https://youtu.be/1VfaSKPBMeI?si=tFQQw2Lkd7d5rdLU>

A black background with white text

AI-generated content may be incorrect.

<https://youtu.be/JtoIu9vPYQI?si=YbRWa7HHUz4py5c1>

A black background with white text

AI-generated content may be incorrect.

<https://youtu.be/bINdzlbF1W8?si=l6z1_PLlsskq6Jf4>

A black background with white text

AI-generated content may be incorrect.

## Pictures used in the program

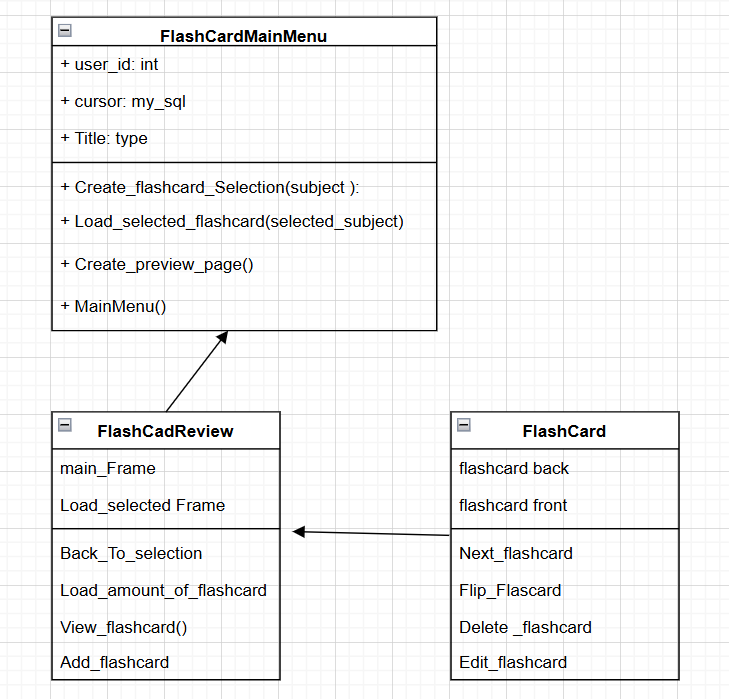
Pictures I used for the picture

|  |  |  |
| --- | --- | --- |
| Picture | Url | Uses |
|  | <https://stock.adobe.com/uk/search?k=%22gear+icon%22&asset_id=740563580> | I used this picture for the setting button |
|  | <https://stock.adobe.com/uk/search?filters%5Bcontent_type%3Aphoto%5D=1&filters%5Bcontent_type%3Aillustration%5D=1&filters%5Bcontent_type%3Azip_vector%5D=1&filters%5Bcontent_type%3Avideo%5D=1&filters%5Bcontent_type%3Atemplate%5D=1&filters%5Bcontent_type%3A3d%5D=1&filters%5Bcontent_type%3Aimage%5D=1&order=relevance&safe_search=1&limit=100&search_page=1&search_type=autosuggest&acp=0&aco=squere+ouline&k=square+outline&get_facets=0&asset_id=647289384> | This is the picture to signify the go fullscreen |
|  | <https://pngtree.com/freepng/hourglass-vector-icon_4015279.html> | Pomodor icon |
|  | <https://pngtree.com/freepng/vector-calendar-icon_4155724.html> | Calendar icon |
|  | <https://www.flaticon.com/free-icon/filter_3126528?term=control+panel&page=1&position=8&origin=tag&related_id=3126528> | Userpanel icon |
|  | <https://freepngimg.com/png/11763-home-png-image> | Homepage icon |
|  | <https://imgbin.com/png/SQvxgssS/mega-man-8-mega-man-6-mega-man-the-power-battle-boss-png> | Boss and player image |
|  | <https://www.cleanpng.com/png-video-game-computer-icons-clip-art-video-game-825689/download-png.html> | Minigaem icon |

## UML DESIGN

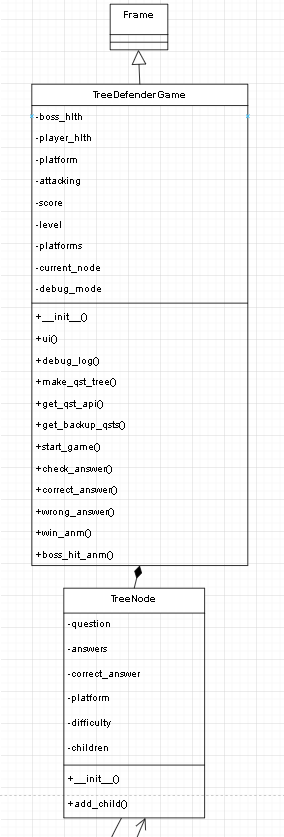
### Flashcard –

In the flashcard application I will have to use Ploymorphism . this is bevasue the program gos throgu h main menu then flashcard review and then flashcard what the menu does is allows the user to select the subject they wan to see , and therefore the code will change accordingly to the user selected subject .



### MINI GAME -

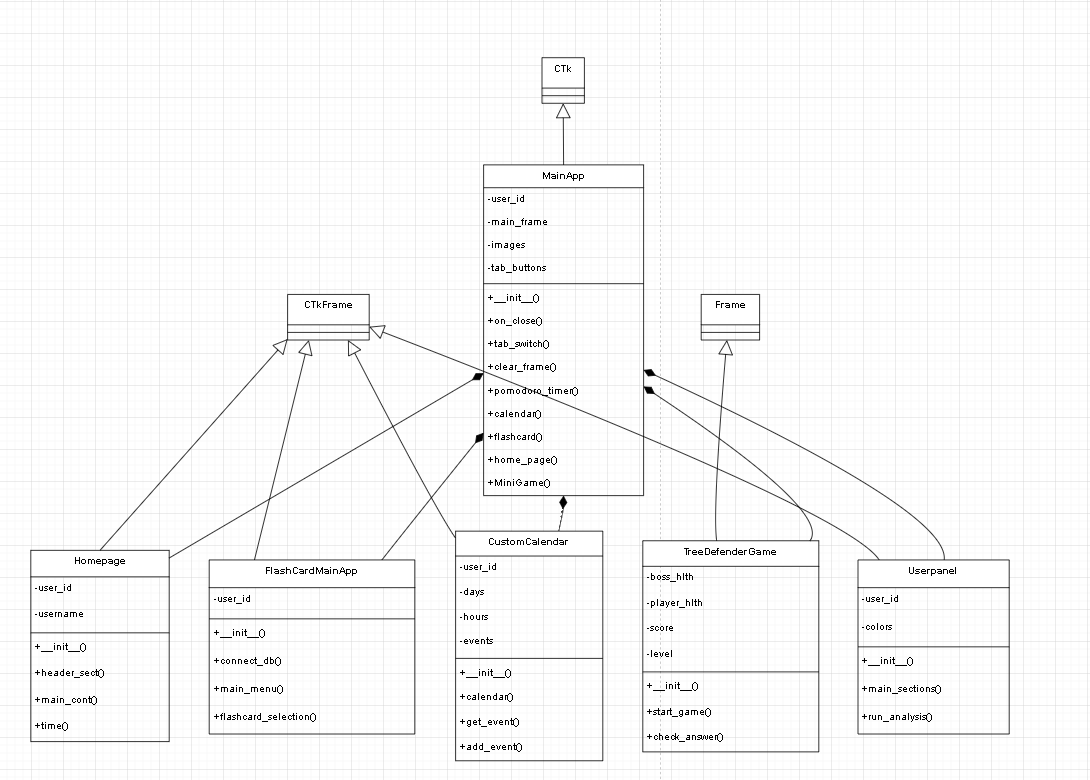
Main Menu to MainGame where 2 different objects will be making player and boss and a funcition to validate.

In this OOP diagram It will use : Inheritance , Composition and aggregation , Encapsulation,Polymorphism **Inheritance -** Tkinter frame class which measn it inherits all the Gui container . which includes widget containment ,layout containment ,and event handling , **Composition –**TreeDefenderGame creates TreeNode objects and own their life cycle - TreeNode exist to serve the game , theirs lifecycle is tied to the game instance , **Encapsulatin –** keeping attributs private to the class , providing and manipulating the data .

### Full system -

This is the OOP system that I migth use for the main app on which app inherits what they need . The main app is goign to be the parent and every other tabs will be inherit or depends on the parent node . The Main App will be the container for the other programs to stay in with such as containing the application frame and the main frame needs to be consistant if not then the code will be making many applications frame wich will back log and lag the pc. So, the Main app is the skeleton of each program as it holds the size location and fucntion .

As soon as the Main App from Login it will open the welcome page which is linked to finishing the set up for the users but if the user doesn't need to always go through this page as this is only if they do not have availability or subject stored .



## **Database Design**

The database structure consists of five main entities with relationships as illustrated in the provided. It can change depending on the needs the code may require later but this is the main way that the database will interact with each other

* **Users**: Central entity storing user login information
* **Subjects**: Stores subjects that connect each user
* **Scores**: Records test scores for each subject within month and year
* **Events**: Tracks calendar events and study sessions
* **Flashcards**: Stores learning materials for revision

### 2. Table Definitions

#### 2.1 Users Table (nea\_users)

This table stores authentication and user identity information.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Data Type** | **Constraints** | **Description** |
| user\_id | int(11) | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for each user |
| username | varchar(50) | NOT NULL, UNIQUE | User login name |
| password | varchar(50) | NOT NULL | Hashed password for authentication |

**Notes:**

* In a production environment, passwords should be stored using secure hashing algorithms (e.g., encrypt)
* Consider adding additional user profile fields like email, date\_joined, last\_login

#### 2.2 Subjects Table (nea\_subjects)

This table stores the academic subjects for each user.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Data Type** | **Constraints** | **Description** |
| subject\_id | int(11) | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for each subject |
| user\_id | int(11) | FOREIGN KEY | Reference to the user who owns this subject |
| subject\_name | varchar(50) | NOT NULL | Name of the academic subject |

**Notes:**

* Consider adding additional fields like color\_code for UI customization
* Could add difficulty\_level or priority fields to help with timetable generation

#### 2.3 Scores Table (nea\_scores)

This table tracks academic performance for each subject over time.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Data Type** | **Constraints** | **Description** |
| subject\_id | int(11) | FOREIGN KEY | Reference to the subject |
| month | int(11) | CHECK (month BETWEEN 1 AND 12) | Month of the score record |
| year | int(11) | NOT NULL | Year of the score record |
| score | float | NOT NULL, CHECK (score BETWEEN 0 AND 100) | Performance score (percentage) |
| user\_id | int(11) | FOREIGN KEY | Reference to the user |

**Notes:**

* The combination of subject\_id,will be foreign key that will help store the scores

#### 2.4 Events Table (nea\_events)

This table stores calendar events including study sessions and deadlines.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Data Type** | **Constraints** | **Description** |
| user\_id | int(11) | FOREIGN KEY | Reference to the user |
| date | text | NOT NULL | Date of the event |
| time | text | NOT NULL | Time of the event |
| title | text | NOT NULL | Title/description of the event |

**Notes:**

* Consider adding an event\_id as PRIMARY KEY for better reference
* Using DATE and TIME data types would be more appropriate than TEXT

#### 2.5 Flashcards Table (nea\_flashcards)

This table stores study materials in flashcard format.

|  |  |  |  |
| --- | --- | --- | --- |
| **Column** | **Data Type** | **Constraints** | **Description** |
| card\_id | int(11) | PRIMARY KEY, AUTO\_INCREMENT | Unique identifier for each flashcard |
| subject\_id | int(11) | FOREIGN KEY | Reference to the subject |
| user\_id | int(11) | FOREIGN KEY | Reference to the user |
| front | text | NOT NULL | Question or prompt side of flashcard |
| back | text | NOT NULL | Answer or response side of flashcard |

**Notes:**

* Consider adding created\_date and last\_reviewed fields
* A difficulty\_rating could help with spaced repetition algorithms
* Tags field might help with organization

### 3. Relationships

1. **Users to Subjects**: One-to-Many

One user can have multiple subjects

Each subject belongs to exactly one user

1. **Subjects to Scores**: One-to-Many

One subject can have multiple score records

Each score record belongs to exactly one subject

1. **Users to Scores**: One-to-Many

One user can have multiple score records

Each score record belongs to exactly one user

1. **Users to Events**: One-to-Many

One user can have multiple events

Each event belongs to exactly one user

1. **Users to Flashcards**: One-to-Many

One user can have multiple flashcards

Each flashcard belongs to exactly one user

1. **Subjects to Flashcards**: One-to-Many

One subject can have multiple flashcards

Each flashcard belongs to exactly one subject

### 4.SQL query code

#### Creating tables

All the queries used to make the tables are here. This is going to be a rough outline of what the database may look like when making I might make changes to the database as I run into challenges. The database will hold all the user's data such as scores, availability, events, and the flashcards, and that is due to the reason of the user will be keep changing their data and if they log in on another device they should be able to access it but this also makes it so that the memory used will be heavy and specially for the flashcards the user doesn't have a limit to how much flashcard they can store and due to that the table will be memory heavy .

-- Users Table

CREATE TABLE nea\_users ( user\_id INT (11) PRIMARY KEY AUTO\_INCREMENT, username VARCHAR (50) NOT NULL UNIQUE, password VARCHAR (50) NOT NULL);

-- Subjects Table

CREATE TABLE nea\_subjects ( subject\_id INT (11) PRIMARY KEY AUTO\_INCREMENT, user\_id INT (11) NOT NULL, subject\_name VARCHAR (50) NOT NULL, FOREIGN KEY (user\_id) REFERENCES nea\_users(user\_id) ON DELETE CASCADE).

-- Scores Table

CREATE TABLE nea\_scores ( subject\_id INT(11) NOT NULL, month INT(11) NOT NULL CHECK (month BETWEEN 1 AND 12), year INT(11) NOT NULL, score FLOAT NOT NULL CHECK (score BETWEEN 0 AND 100), user\_id INT(11) NOT NULL, PRIMARY KEY (subject\_id, month, year), FOREIGN KEY (subject\_id) REFERENCES nea\_subjects(subject\_id) ON DELETE CASCADE, FOREIGN KEY (user\_id) REFERENCES nea\_users(user\_id) ON DELETE CASCADE );

-- Events Table

CREATE TABLE nea\_events ( event\_id INT (11) PRIMARY KEY AUTO\_INCREMENT, user\_id INT (11) NOT NULL, date TEXT NOT NULL, time TEXT NOT NULL, title TEXT NOT NULL, FOREIGN KEY (user\_id) REFERENCES nea\_users(user\_id) ON DELETE CASCADE);

-- Flashcards Table

CREATE TABLE nea\_flashcards ( card\_id INT(11) PRIMARY KEY AUTO\_INCREMENT, subject\_id INT(11) NOT NULL, user\_id INT(11) NOT NULL, front TEXT NOT NULL, back TEXT NOT NULL, FOREIGN KEY (subject\_id) REFERENCES nea\_subjects(subject\_id) ON DELETE CASCADE, FOREIGN KEY (user\_id) REFERENCES nea\_users(user\_id) ON DELETE CASCADE );

-- Availability Table

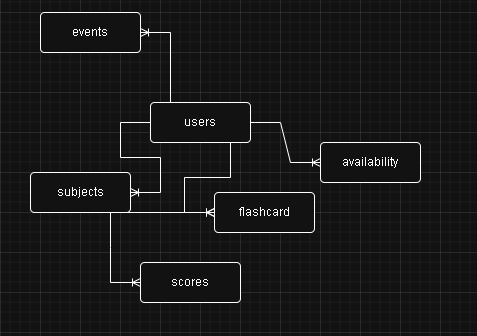
CREATE TABLE availability (id int(11) NOT NULL, user\_id int(11) NOT NULL, day varchar(20) NOT NULL, hours longtext CHARACTER SET ,);

To get the scores -

SELECT subjects.subject\_name,scores.score,score.month,score.year FROM subjects score JOIN SELECT subject\_id , score , month , year ROW\_NUMBER() OVER (subject\_id ORDER BY year DESC , month DESC ) as rownumber FROM scores ) scores ON subjects.subject\_id = scores.subject\_id AND scores.rownumber = 1 WHERE subject.user\_id = s%

To get the use availability

#### DATABASE relationship design



This is the database design that i might use when making the code. It isn’t finalized but this is how it would look if i made the database. The database only holds the user details and it's most user centered due to the fact that if the user wants to log in from another device, they will get the information they need.

## Algorithms-

### Linear Search –

An algorithm where it goes through each of the components until the desired item is found. The algorithm is good for small and data set as it's an O(n) complexity so the larger the data set the more time it takes to get the desired item. This can be used for getting resources as it will have a small dataset as there is a set limit to number of subjects in the world .

The algorithm will be used for the analysis page , more specifically the feelings program , the user will input how they feel it will take that and go through the dataset to get the advice

Pseudo code -

BEGIN

FUNCTION linear\_search\_feelings(user\_input)

TRY

feelings\_data ← READ\_JSON\_FILE(feelings\_file)

user\_words ← LOWERCASE(user\_input).SPLIT()

recommendations ← []

FOR EACH feeling IN feelings\_data DO

IF LOWERCASE(feeling) IN user\_words THEN

ADD feelings\_data[feeling] TO recommendations

END IF

END FOR

RETURN recommendations

CATCH Exception

RETURN []

END TRY

END FUNCTION

END

### Bubble Sort –

Bubble sort is where it goes through each indevidual element in the list and if the value is less then the previous then it will swap the elemtn and it goes through each element . Bubble sort has a O(n^2) complexity which will make it complicated in a massicve unsorted list . I will use bubble sort for the graph as the data will be store randomly as the user can add a score of today today but then it add a score he got yesdterady making it unorganized which is why before we display the graph we use bubbe sort as an exempt handler to get all the score in the right ordern

PsudoCode-

BEGIN

FUNCTION bubble\_sort(data)

sorted\_data ← {}

FOR EACH subject, scores IN data DO

sorted\_scores ← COPY(scores)

n ← LENGTH(sorted\_scores)

FOR i ← 0 TO n-1 DO

FOR j ← 0 TO n-i-2 DO

IF sorted\_scores[j]['month'] > sorted\_scores[j+1]['month'] THEN

SWAP sorted\_scores[j], sorted\_scores[j+1]

END IF

END FOR

END FOR

sorted\_data[subject] ← sorted\_scores

END FOR

RETURN sorted\_data

END FUNCTION

END

### Binary search -

Binary sort, it's where it divides an organized data set and sees if the said value is above or below and decides which direction to go. It has O(logn) Complexity. I will be using binary search for the resources as then it will be much easier and faster to get the desired subject. I need to make sure the user Input is all lower case to make it easier to get the data as it will match . I used Binary search as the data strucutre will be organized and their will be many subject so we need a quick but efficient search which is why I used binary search

PSUDO CODE-

BEGIN

FUNCTION binary\_search(subject)

// Prepare sorted list of subjects (case-insensitive)

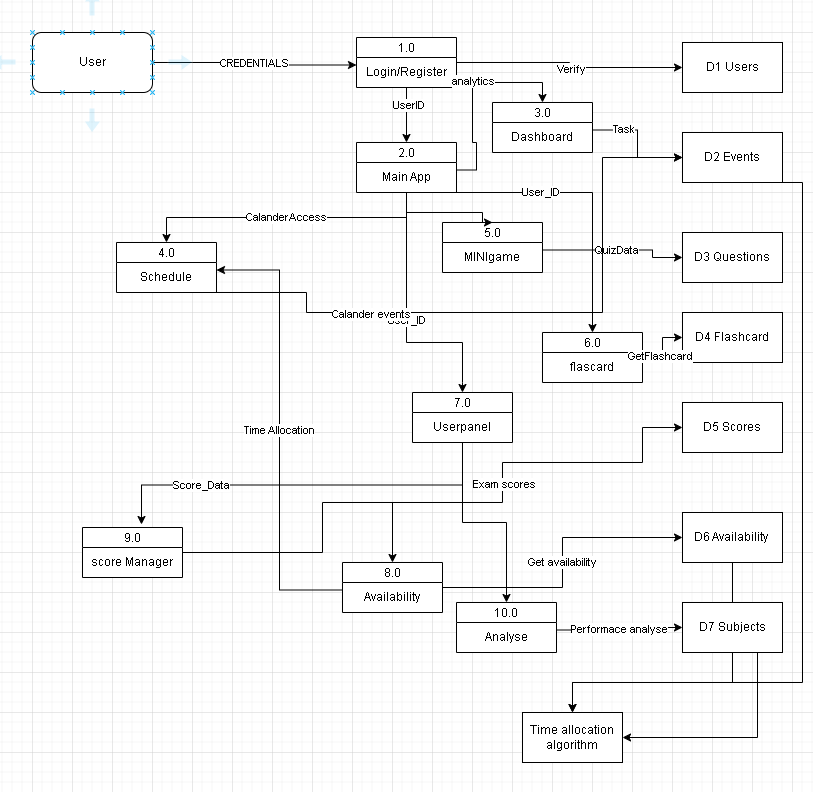
subjects ← SORT(resources.keys(), CASE\_INSENSITIVE)

// Initialize search boundaries  
 low ← 0  
 high ← LENGTH(subjects) - 1  
   
 // Binary search loop  
 WHILE low <= high DO  
 // Calculate midpoint  
 mid ← (low + high) / 2 (integer division)  
 current\_subject ← subjects[mid]  
   
 // Case-insensitive comparison  
 IF LOWERCASE(current\_subject) == LOWERCASE(subject) THEN  
 RETURN current\_subject // Exact match found  
 ELSE IF LOWERCASE(current\_subject) < LOWERCASE(subject) THEN  
 low ← mid + 1 // Search right half  
 ELSE  
 high ← mid - 1 // Search left half  
 END IF  
 END WHILE  
   
 RETURN NULL // No match found  
END FUNCTION

END

## Data flow-

This diagram will show where the data will go through out the code and which app will need wich storage and attribute and what program will amend or change in the database not all the storage is database is aso json and api from the website. In the data flow it also shows the allocation alorithm that will allocate the subject revision time which is events is stored from the algorithm and scores is taken for the alogorithm and avilability .



## Arithmetic calculation -

### Calculation for priority system

**weighted time allocation formula**

The weighted time allocation formula takes the score flips the score which means inversing the score so that mean if the user scores 30 then weight of 70 is given and so it's more prioritized and the same is done with the other scores and then the score is divided with the total weight and the times with the Total time in that week and that will give us the amount of hours in that week

#### Psudocode-

BEGIN

FUNCTION normalize\_weight(weight) normalized\_weight ← 0

IF weight IS INSTANCE OF LIST THEN  
 latest\_entry ← FIRST ITEM IN weight  
 FOR EACH entry IN weight DO  
 IF (entry["year"] > latest\_entry["year"]) OR   
 (entry["year"] == latest\_entry["year"]

AND entry["month"] > latest\_entry["month"]) THEN  
 latest\_entry ← entry  
 END IF  
 END FOR  
 normalized\_weight ← 100 - latest\_entry["score"]  
 ELSE  
 normalized\_weight ← 100 - TO\_INTEGER(weight)  
 END IF  
 RETURN normalized\_weight  
END FUNCTION

END

### Calculating available time slot

The user available time will be gotten by the user as they will enter their data for each day of the week. Such as Monday = 14:00-16:00 that 2 hours on Monday. We do this by taking the second value and minus the first value and remainder of the value is the number of hours they have on that day they are then saved to a list or the database and then they are used to calculate the study allocation. This is repeated for the days in a week and all of the values added together to make the total study time in a week.

More simplified version

#### Psudo code -

FUNCTION CalculateTotalWeeklyHours(availability)

totalHours ← 0

FOR day FROM 1 TO 7

CASE OF day

WHEN 1 THEN dayName ← "Monday"

WHEN 2 THEN dayName ← "Tuesday"

WHEN 3 THEN dayName ← "Wednesday"

WHEN 4 THEN dayName ← "Thursday"

WHEN 5 THEN dayName ← "Friday"

WHEN 6 THEN dayName ← "Saturday"

WHEN 7 THEN dayName ← "Sunday"

END CASE

hoursList ← availability[dayName]

IF Length(hoursList) = 0 THEN

// No hours available for this day

CONTINUE

END IF

FOR i ← 0 TO Length(hoursList) - 1

startTime ← hoursList[i]

endTime ← startTime + 1 // Each block is 1 hour

totalHours ← totalHours + (endTime - startTime)

NEXT i

NEXT day

RETURN totalHours

END FUNCTION

### Calculating Trend Analysis –

For my analyze program I have a graph that shows the past 3 month and makes a trend calculating and also calculates the next predicted trend. I did this to show the increase or decrease in learning.

 we need to 3 calculations for this –

* Calculating the current improvement percentage we need to first get last score then first exam and we subtract last with first and then divide with first then after that times it by 100 to get current improvement increase

* Calculating the Predicted Next Movement percentage (to get a prediction percentage if the user study as he intends too ) to get this prediction we need to calculate the percentage difference in between the assesment then we take that and divide it with first score and the times by 100

ASI = Avarage score increase

n=Number of assessments

* Next Score Predicted this predicts the the next score the user might get . we

Do this with sequential mathematics so  we can predict the next score if the user growth in the predicted way

k = number of future assessment to predict

# Technical solution

## CODE

[Eclipse.py](#_Eclipse.py)

[Login.py](#_Login.py)

[Main.py](#_Main.py)

[Server.py](#_Server.py)

[Home\_page.py](#_Home_page.py)

[Taskviewer.py](#_Taskviewer.py)

[Pomodoro\_timer.py](#_Pomodoro_timer.py)

[Calander.py](#_Calander.py)

[Tree\_game\_v.py](#_Tree_game_v.py)

[Scores.py](#_Scores.py)

[Userpanel.py](#_Userpanel.py)

[Flashcard\_v.py](#_Flashcard_v.py)

[Examcountdown.py](#_Examcountdown.py)

[Calculator\_normal.py](#_Calculator_normal.py)

[Graph\_bar.py](#_Graph_bar.py)

[Quotes.py](#_Quotes.py)

[Resources.py](#_Resources.py)

[Availability.py](#_Availability.py)

[Analyse.py](#_Analysis.py)

### Eclipse.py

# eclipse\_app.py - Main entry point for the Eclipse application

from Login import login\_page

# This is the main entry point - run this file to start the application

if \_\_name\_\_ == "\_\_main\_\_":

login\_page()

### Login.py

from customtkinter import \*

import customtkinter as ctk

from PIL import Image, ImageTk

import pymysql

from server import \*

# Main login window creator

def login\_page():

app = ctk.CTk()

app.title("ECLIPSE LEARNING")

# Set window

app.geometry("1000x600")

app.minsize(1000, 600)

ctk.set\_appearance\_mode("dark")

ctk.set\_default\_color\_theme("blue")

# Load the background

image = Image.open("Pics/wave.png")

background\_image = CTkImage(dark\_image=image, light\_image=image, size=(500, 600))

#image = CTkLabel(left\_panel, text="", image=background\_image) # keep this just in case

#image.place(relx=0.5, rely=0.5, anchor="center")

# Setup the main frame - this holds everything

main\_frm = CTkFrame(app, fg\_color="#16161A")

main\_frm.pack(fill="both", expand=True)

# Left side for the bg image

left\_frm = CTkFrame(main\_frm, fg\_color="#16161A", width=500)

left\_frm.pack(side="left", fill="both", expand=True)

# Right panel - this is where login stuff goes

right\_panel = CTkFrame(main\_frm, fg\_color="#16161A", width=500)

right\_panel.pack(side="right", fill="both")

# Container for login form

login\_frame = CTkFrame(right\_panel, fg\_color="transparent", width=400)

login\_frame.place(relx=0.5, rely=0.5, anchor="center")

image = CTkLabel(left\_frm, text="", image=background\_image)

image.place(relx=0.5, rely=0.5, anchor="center")

# Sign up window function

def sign\_up\_page():

# Handle the signup button click

def signup\_handler():

# Grab the data from form

username\_signup = username\_inpt.get().strip()

print(username\_signup) # debug

password\_signup = password\_entry\_box.get()

print(password\_signup) # debug

confirm\_password\_signup = confirm\_password\_entry.get()

print(confirm\_password\_signup) # debug

# List to store validation errors

errors = []

# Super basic validation

if not username\_signup:

errors.append("Username cannot be empty")

print("Username cannot be empty")

elif len(username\_signup) < 4:

errors.append("Username must be at least 4 characters")

print("Username must be at least 4 characters")

if not password\_signup:

errors.append("Password cannot be empty")

print("no password")

elif len(password\_signup) < 4:

errors.append("Password must be at least 4 characters")

if password\_signup != confirm\_password\_signup:

errors.append("Passwords do not match")

# Check if username exists in DB

try:

query = "SELECT \* FROM users WHERE username = %s"

cursor.execute(query, (username\_signup,))

result = cursor.fetchone()

if result:

errors.append("Username is already taken")

print("DB username taken")

except Exception as e:

errors.append(f"Database error: {str(e)}")

print("DB fail")

# Show errors

if errors:

error\_frame = CTkFrame(signup\_frame, fg\_color="red", corner\_radius=10)

error\_frame.place(relx=0.5, rely=0.9, anchor="center", relwidth=0.9)

# Make a bullet list of errors

text\_error = "\n".join([f"• {error}" for error in errors])

error\_label = CTkLabel(error\_frame, text=text\_error, font=("Helvetica", 10), text\_color="black", wraplength=300,justify="left")

error\_label.pack(padx=10, pady=10)

# Auto-destroy errors after 5s

error\_frame.after(3000, error\_frame.destroy)

return

# Create the user

try:

query = "INSERT INTO users (username, password) VALUES (%s, %s)"

cursor.execute(query, (username\_signup, password\_signup))

db\_connection.commit() # save to db

print("user created")

# Show success message

show\_message("Success", "Your account has been created successfully!","#2CB67D", lambda: [signup\_frame.destroy(), show\_login\_frame()])

except Exception as e:

show\_message("Error",f"Failed to create account: {str(e)}","red", ) # red color for error

# Helper function for popup messages

def show\_message(title, message, color, on\_close):

message\_frame = CTkFrame(signup\_frame, fg\_color=color,corner\_radius=10,border\_width=1,border\_color=color)

message\_frame.place(relx=0.5, rely=0.5, anchor="center", relwidth=0.8, relheight=0.4)

title\_label = CTkLabel(message\_frame, text=title, font=("Helvetica", 16, "bold"), text\_color=color)

title\_label.pack(pady=(20, 10))

msg\_label = CTkLabel(message\_frame, text=message, font=("Helvetica", 12), text\_color="#FFFFFE",wraplength=300)

msg\_label.pack(pady=10)

# Close button

close\_button = CTkButton(message\_frame, text="Close", font=("Helvetica", 12),fg\_color=color,text\_color="#FFFFFE",hover\_color=color,corner\_radius=10,command=lambda: [message\_frame.destroy(), on\_close() if on\_close else None])

close\_button.pack(pady=10)

# signup popup frame

signup\_frame = CTkFrame(right\_panel, fg\_color="#242629",corner\_radius=10,border\_width=1,border\_color="#94A1B2",width=400, height=500) # make sure the sizing is alr

signup\_frame.place(relx=0.5, rely=0.5, anchor="center")

# Header part

header\_frame = CTkFrame(signup\_frame, fg\_color="transparent")

header\_frame.pack(fill="x", padx=20, pady=20)

header = CTkLabel(header\_frame, text="Create an Account", font=("Helvetica", 20, "bold"), text\_color="#FFFFFE")

header.pack(side="left")

close\_btn = CTkButton(header\_frame, text="✕", font=("Helvetica", 12),fg\_color="transparent",text\_color="#94A1B2",hover\_color="#16161A",width=30,height=30,corner\_radius=15,command=signup\_frame.destroy)

close\_btn.pack(side="right")

container = CTkFrame(signup\_frame, fg\_color="transparent")

container.pack(fill="both", expand=True, padx=20, pady=10)

# input fields

username\_label = CTkLabel(container, text="Username", font=("Helvetica", 12), text\_color="#FFFFFE",anchor="w")

username\_label.pack(fill="x", pady=5)

username\_inpt = CTkEntry(container, font=("Helvetica", 12),fg\_color="#16161A",text\_color="#FFFFFE",border\_color="#94A1B2",corner\_radius=10,height=40)

username\_inpt.pack(fill="x", pady=10)

password\_label = CTkLabel(container, text="Password", font=("Helvetica", 12), text\_color="#FFFFFE",anchor="w")

password\_label.pack(fill="x", pady=5)

password\_entry\_box = CTkEntry( container, font=("Helvetica", 12),fg\_color="#16161A",text\_color="#FFFFFE",border\_color="#94A1B2",corner\_radius=10,height=40,show="●" ) # hide password

password\_entry\_box.pack(fill="x", pady=10)

confirm\_password\_label = CTkLabel(container, text="Confirm Password", font=("Helvetica", 12), text\_color="#FFFFFE",anchor="w")

confirm\_password\_label.pack(fill="x", pady=5)

confirm\_password\_entry = CTkEntry(container, font=("Helvetica", 12),fg\_color="#16161A",text\_color="#FFFFFE",border\_color="#94A1B2",corner\_radius=10,height=40,show="●") # hide password

confirm\_password\_entry.pack(fill="x", pady=10)

# Sign Up button

signup\_button = CTkButton(container, text="Sign Up", font=("Helvetica", 12, "bold"),fg\_color="#7F5AF0", text\_color="#FFFFFE",hover\_color="#7F5AF0", corner\_radius=10,height=45,command=signup\_handler)

signup\_button.pack(fill="x", pady=10)

# back to login button

back\_to\_login = CTkFrame(container, fg\_color="transparent")

back\_to\_login.pack(fill="x", pady=10)

login\_text = CTkLabel(back\_to\_login, text="Already have an account?", font=("Helvetica", 10), text\_color="#94A1B2")

login\_text.pack(side="left")

back\_to\_login = CTkButton(back\_to\_login, text="Log In", font=("Helvetica", 10),fg\_color="transparent",text\_color="#7F5AF0",hover\_color="blue", hover=False,width=50,height=20,command=lambda: signup\_frame.destroy())

back\_to\_login.pack(side="left", padx=5)

# Function to show the login form

def show\_login\_frame():

# Handle login button press

def handler\_for\_login():

username\_inpt = username\_entry.get().strip()

password\_inpt = password\_entry.get()

# Basic validation

if not username\_inpt or not password\_inpt:

error("Please enter both username and password")

return

# Check login in DB

try:

query = "SELECT user\_id, username FROM users WHERE username = %s AND password = %s"

cursor.execute(query, (username\_inpt, password\_inpt))

result = cursor.fetchone()

if result:

# Login worked

user\_id, username = result

# Show success and proceed

show\_success(f"Welcome back, {username}!", user\_id)

else:

# Bad login

error("Invalid username or password")

except Exception as e:

# DB error

error(f"Database error: {str(e)}")

# Show error messages

def error(message):

error = CTkFrame(login\_frame, fg\_color="red",corner\_radius=10)

error.place(relx=0.5, rely=0.85, anchor="center", relwidth=0.9)

error\_label = CTkLabel(error, text=message, font=("Helvetica", 10), text\_color="black")

error\_label.pack(padx=10, pady=10)

# Auto-hide after 5s

error.after(5000, error.destroy)

# Success login handler

def show\_success(message, user\_id):

# Green success popup

success\_frame = CTkFrame(right\_panel, fg\_color="#2CB67D",corner\_radius=10,border\_width=1,border\_color="#2CB67D")

success\_frame.place(relx=0.5, rely=0.5, anchor="center", relwidth=0.8, relheight=0.5)

success\_icon = CTkLabel(success\_frame, text="✓", font=("Arial", 48, "bold"), text\_color="#2CB67D")

success\_icon.pack(pady=20)

success\_msg = CTkLabel(success\_frame, text=message, font=("Helvetica", 16, "bold"), text\_color="black")

success\_msg.pack(pady=10)

loading = CTkLabel(success\_frame, text="Preparing your dashboard...", font=("Helvetica", 12), text\_color="#94A1B2")

loading.pack(pady=10)

# Fake loading bar

progress\_container = CTkFrame(success\_frame, fg\_color="#16161A", height=10, width=200, corner\_radius=5)

progress\_container.pack(pady=10)

progress\_bar = CTkFrame(progress\_container, fg\_color="#2CB67D", height=10, width=0, corner\_radius=5)

progress\_bar.place(x=0, y=0)

# Make the progress bar fill up

def loading\_animation(value=0):

if value <= 100:

width = (value / 100) \* 200

progress\_bar.configure(width=width)

# Increase by 2% each time - not perfect but looks ok

success\_frame.after(30, lambda: loading\_animation(value + 2))

else:

# Ready to launch the main app

success\_frame.after(500, lambda: [app.destroy(), launch\_main\_app(user\_id)])

# Start the progress animation

success\_frame.after(500, lambda: loading\_animation())

# Launch the main app

def launch\_main\_app(user\_id):

# Import the main app only when needed

from main import MainApp

main = MainApp(user\_id)

main.mainloop()

# title

login\_title = CTkLabel(login\_frame, text="Welcome Back", font=("Helvetica", 24, "bold"), text\_color="#FFFFFE" )

login\_title.pack(pady=5)

login\_subtitle = CTkLabel(login\_frame, text="Sign in to continue your learning journey", font=("Helvetica", 12), text\_color="#94A1B2" )

login\_subtitle.pack(pady=20)

# app logo

branding\_frm = CTkFrame(left\_frm, fg\_color="transparent")

branding\_frm.place(relx=0.5, rely=0.2, anchor="center")

logo\_frm = CTkFrame(branding\_frm, fg\_color="#7F5AF0", width=80, height=80, corner\_radius=40)

logo\_frm.pack(pady=20)

logo\_txt = CTkLabel(logo\_frm, text="Eclipse", font=("Arial", 40, "bold"), text\_color="#FFFFFE")

logo\_txt.place(relx=0.5, rely=0.5, anchor="center")

# user input fields

username\_label = CTkLabel(login\_frame, text="Username", font=("Helvetica", 12), text\_color="#FFFFFE" ,anchor="w")

username\_label.pack(fill="x", pady=5)

username\_entry = CTkEntry(login\_frame, font=("Helvetica", 12),fg\_color="#16161A" ,text\_color="#FFFFFE",border\_color="#94A1B2",corner\_radius=10,height=45,width=400)

username\_entry.pack(fill="x", pady=10)

password\_label = CTkLabel(login\_frame, text="Password", font=("Helvetica", 12), text\_color="#FFFFFE" ,anchor="w")

password\_label.pack(fill="x", pady=5)

password\_entry = CTkEntry(login\_frame, font=("Helvetica", 12),fg\_color="#16161A" ,text\_color="#FFFFFE" ,border\_color="#94A1B2" ,corner\_radius=10,height=45,width=400,show="●") # Hide password with bullet

password\_entry.pack(fill="x", pady=10)

# buttons

login\_button = CTkButton(login\_frame, text="Log In", font=("Helvetica", 12, "bold"),fg\_color="#7F5AF0",text\_color="#FFFFFE",hover\_color="#7F5AF0", corner\_radius=10,height=45,width=400,command=handler\_for\_login )

login\_button.pack(pady=10)

register\_frame = CTkFrame(login\_frame, fg\_color="transparent")

register\_frame.pack(pady=5)

register\_text = CTkLabel(register\_frame, text="Don't have an account?", font=("Helvetica", 10), text\_color="#94A1B2")

register\_text.pack(side="left")

register\_Button = CTkButton( register\_frame, text="Sign Up", font=("Helvetica", 10),fg\_color="transparent",text\_color="#7F5AF0",hover\_color="blue",hover=False,width=60,height=20,command=sign\_up\_page)

register\_Button.pack(side="left", padx=5)

# Just some info text

frm\_info = CTkFrame(login\_frame, fg\_color="#16161A" , corner\_radius=10)

frm\_info.pack(fill="x", pady=20)

I\_label = CTkLabel( frm\_info, text="Welcome to eclipse ", font=("Helvetica", 10), text\_color="#94A1B2")

I\_label.pack(pady=5)

I = CTkLabel( frm\_info, text="PLease remember your password", font=("Helvetica", 10), text\_color="#94A1B2")

I.pack()

# might add more info here later if needed

I\_2 = CTkLabel(frm\_info, text="...", font=("Helvetica", 10), text\_color="#94A1B2")

I\_2.pack(pady=5)

# Show login form initially

show\_login\_frame()

# Start the app

app.mainloop()

if \_\_name\_\_ == "\_\_main\_\_":

login\_page()

### Main.py

from customtkinter import \*

import customtkinter as ctk

from customtkinter import CTkImage

from PIL import Image

from pomodoro\_timer import pomodoro\_timer

from flashcard\_v import FlashCardMainApp

from calander import CustomCalendar

from userpanel import Userpanel

from home\_page import Homepage

from user\_profile import UserProfile

from tree\_game\_v import TreeDefenderGame

from calculator\_normal import check\_setup, complete\_setup

class MainApp(ctk.CTk):

def \_\_init\_\_(self, user\_id):

super().\_\_init\_\_()

self.user\_id = user\_id

self.user\_id\_tp = tuple(str(user\_id))

# Configure main window

self.title("Eclipse")

self.screen\_width = self.winfo\_screenwidth()

self.screen\_height = self.winfo\_screenheight()

set\_appearance\_mode("dark")

# Protocol for closing the window

self.protocol("WM\_DELETE\_WINDOW", self.on\_close)

# Main container

self.main\_frame = ctk.CTkFrame(self)

self.main\_frame.pack(fill="both", expand=True, padx=10, pady=10)

if check\_setup(self.user\_id):

self.finish\_singup()

else:

self.start\_app()

def finish\_singup(self):

#Show a welcome screen

for widget in self.main\_frame.winfo\_children():

widget.destroy()

welcome\_frm = ctk.CTkFrame(self.main\_frame)

welcome\_frm.pack(fill="both", expand=True, padx=20, pady=20)

# Welcome message

ctk.CTkLabel(welcome\_frm, text="Welcome to Eclipse Learning!", font=("Arial", 24, "bold")).pack(pady=(50, 20))

ctk.CTkLabel(welcome\_frm, text="Before you begin, we need to set up your availability and subjects.", font=("Arial", 16),wraplength=600).pack(pady=(0, 30))

# Info

info\_frm = ctk.CTkFrame(welcome\_frm)

info\_frm.pack(fill="x", padx=50, pady=20)

ctk.CTkLabel(info\_frm, text="We'll need you to:", font=("Arial", 14, "bold"),anchor="w").pack(fill="x", pady=(10, 5), padx=10)

ctk.CTkLabel(info\_frm, text="• Add your subjects (e.g., Math, English, Science)", font=("Arial", 14),anchor="w").pack(fill="x", pady=5, padx=30)

ctk.CTkLabel(info\_frm, text="• Set your weekly availability for each day", font=("Arial", 14),anchor="w").pack(fill="x", pady=5, padx=30)

ctk.CTkLabel(info\_frm, text="• This helps us create an optimal study schedule for you", font=("Arial", 14),anchor="w").pack(fill="x", pady=(5, 10), padx=30)

# Button to continue to setup

ctk.CTkButton(welcome\_frm, text="Continue to Setup", font=("Arial", 16, "bold"),width=250, height=50,command=self.start\_setup).pack(pady=40)

def start\_setup(self):

#Show a setup screen

complete\_setup(self.user\_id, callback=self.start\_app)

def start\_app(self):

for widget in self.main\_frame.winfo\_children():

widget.destroy()

# Tab bar (left sidebar)

self.Func\_frm = CTkFrame(master=self.main\_frame, fg\_color="#22333B", width=250)

self.Func\_frm.pack\_propagate(False) # Prevent size changes

self.Func\_frm.pack(side="left", fill="y")

logo\_frm = CTkFrame(self.Func\_frm, fg\_color="transparent", width=50, height=200)

logo\_frm.pack(fill="both", expand=False, side="top")

CTkFrame(self.Func\_frm, fg\_color="transparent", width=50, height=200).pack(fill="both", expand=False, side="top")

# Application

self.application\_frm = CTkFrame(master=self.main\_frame, fg\_color="#222222")

self.application\_frm.pack(side="right", expand=True, fill="both")

self.tab\_buttons = {}

tab\_container = CTkFrame(self.Func\_frm, fg\_color="transparent")

tab\_container.pack(fill="both", side="top")

# Load images

self.images = {

'logo': CTkImage(Image.open("Pics/user\_profile.png"), size=(100, 100)),

'timer': CTkImage(Image.open("Pics/hourglass\_fb.png"), size=(50, 50)),

'calendar': CTkImage(Image.open("Pics/calander.png"), size=(50, 50)),

'flashcards': CTkImage(Image.open("Pics/flashcard.png"), size=(50, 50)),

'user': CTkImage(Image.open("Pics/userpanel.png"), size=(50, 50)),

'home': CTkImage(Image.open("Pics/home.png"), size=(50, 50)),

'settings': CTkImage(Image.open("Pics/Setting.png"), size=(50, 50)),

'full\_screen': CTkImage(Image.open("Pics/window.png"), size=(50, 50)),

'minigame': CTkImage(Image.open("Pics/minigame.png"), size=(50, 50)),

}

# buttons

tabs = [

('home', "Home", self.home\_page),

('timer', "Pomodoro", self.pomodortimer),

('calendar', "Calendar", self.calendar),

('flashcards', "Flashcards", self.flashcard),

('user', "User Panel", self.user\_panel),

('minigame', "MiniGame", self.MiniGame)

]

Logo = CTkButton(logo\_frm,image=self.images['logo'],text="",compound="left",anchor="center",fg\_color="transparent", text\_color="white",hover\_color="#3a3a3a",command=self.user\_profile)

Logo.pack(side="top", fill="x", pady=2)

for tab\_id, text, command in tabs:

btn = CTkButton(tab\_container,text=text,image=self.images[tab\_id],compound="left",anchor="w",fg\_color="transparent", text\_color="white",hover\_color="#3a3a3a",command=command)

btn.pack(fill="x", pady=5)

self.tab\_buttons[tab\_id] = btn

self.settings\_btn = CTkButton(self.Func\_frm,image=self.images['settings'],text="Settings",compound="left",anchor="w",fg\_color="transparent", text\_color="white",hover\_color="#3a3a3a",command=self.settings)

self.settings\_btn.pack(side="bottom", fill="x", pady=20, padx=10)

# start homepage

self.tab\_swicth('Home')

self.home\_page()

def on\_close(self):

#event close

if hasattr(self, 'homepage'):

self.homepage.task\_viewr.save\_deletions()

self.destroy()

def tab\_swicth(self, tab\_id):

for btn in self.tab\_buttons.values():

btn.configure(fg\_color="transparent")

# Highlight active tab

if tab\_id in self.tab\_buttons:

self.tab\_buttons[tab\_id].configure(fg\_color="#3a3a3a")

def clear\_frame(self):

if hasattr(self, 'homepage') and self.homepage:

self.homepage.task\_viewr.save\_deletions()

for widget in self.application\_frm.winfo\_children():

widget.destroy()

def pomodortimer(self):

self.clear\_frame()

self.tab\_swicth('timer')

pomodoro\_timer(self.application\_frm)

def user\_profile(self):

self.clear\_frame()

self.tab\_swicth('user\_profile')

self.user\_profile = UserProfile(self.application\_frm, self.user\_id)

self.user\_profile.pack(fill="both", expand=True)

def calendar(self):

self.clear\_frame()

self.tab\_swicth('calendar')

self.calendar = CustomCalendar(self.application\_frm, self.user\_id)

self.calendar.pack(fill="both", expand=True)

def flashcard(self):

self.clear\_frame()

self.tab\_swicth('flashcards')

self.flashcard\_app = FlashCardMainApp(self.application\_frm, self.user\_id\_tp)

self.flashcard\_app.pack(fill="both", expand=True)

def user\_panel(self):

self.clear\_frame()

self.tab\_swicth('user')

self.user\_panel = Userpanel(self.application\_frm, user\_id=self.user\_id)

self.user\_panel.pack(fill="both", expand=True)

def home\_page(self):

self.clear\_frame()

self.tab\_swicth('home')

self.homepage = Homepage(self.application\_frm, self.user\_id, main\_frame=self)

self.homepage.pack(fill="both", expand=True)

def MiniGame(self):

self.clear\_frame()

self.tab\_swicth('minigame')

self.minigame = TreeDefenderGame(self.application\_frm)

self.minigame.pack(fill="both", expand=True)

def settings(self):

self.clear\_frame()

def toggle\_fullscreen():

self.attributes("-fullscreen", not self.attributes("-fullscreen"))

self.settings\_panel = CTkFrame(self.application\_frm, fg\_color="#3a3a3a")

self.settings\_panel.pack(side="bottom", fill="both", expand=True)

fullscreen\_btn = CTkButton(self.settings\_panel,text="Toggle Fullscreen",image=self.images['full\_screen'],command=toggle\_fullscreen)

fullscreen\_btn.grid(row=1, column=0)

fullscreen\_info = CTkLabel(self.settings\_panel,text="allows you to go full screen")

fullscreen\_info.grid(row=1, column=1)

### Server.py

from customtkinter import \*

import customtkinter as ctk

import mysql.connector

import pymysql

db\_connection = pymysql.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = db\_connection.cursor()

### Home\_page.py

import customtkinter as ctk

import time

from datetime import datetime

import mysql.connector

from task\_viwer import TaskView

class Homepage(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, main\_frame, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

self.main\_frm = main\_frame

self.user\_id = user\_id

self.username = self.get\_username()

# Create main

self.content\_frm = ctk.CTkFrame(self, fg\_color="transparent")

self.content\_frm.pack(fill="both", expand=True, padx=20, pady=20)

self.header\_sect()

# Main content area

self.main\_cont()

# Update time

self.time()

def header\_sect(self):

# Header frame

header\_frm = ctk.CTkFrame(self.content\_frm, fg\_color="transparent")

header\_frm.pack(fill="x", pady=(0, 20))

# Left section with user greeting

left\_sect = ctk.CTkFrame(header\_frm, fg\_color="transparent")

left\_sect.pack(side="left", anchor="w")

# Current time of day greeting

crnt\_hour = datetime.now().hour

if crnt\_hour < 12:

greeting = "Good morning"

elif crnt\_hour < 18:

greeting = "Good afternoon"

else:

greeting = "Good evening"

greeting\_labl = ctk.CTkLabel(left\_sect, text=f"{greeting}, {self.username}",font=("Arial", 24, "bold"),text\_color="#7F5AF0" )

greeting\_labl.pack(anchor="w")

# Current date with better formatting

self.date\_labl = ctk.CTkLabel(left\_sect, text="",font=("Arial", 16),text\_color="#94A1B2" )

self.date\_labl.pack(anchor="w", pady=(5, 0))

# Right section with clock

right\_sect = ctk.CTkFrame(header\_frm, fg\_color="transparent")

right\_sect.pack(side="right", anchor="e")

self.time\_labl = ctk.CTkLabel(right\_sect, text="",font=("Arial", 42, "bold"),text\_color="#7F5AF0" )

self.time\_labl.pack(anchor="e")

def main\_cont(self):

# Main content frame with card design

main\_card = ctk.CTkFrame(self.content\_frm, corner\_radius=15,fg\_color="#242629", border\_width=1,border\_color="#333333")

main\_card.pack(fill="both", expand=True, padx=10, pady=10)

card\_header = ctk.CTkFrame(main\_card, fg\_color="transparent")

card\_header.pack(fill="x", padx=20, pady=(20, 10))

welcome\_labl = ctk.CTkLabel(card\_header, text=f"Welcome back, {self.username}",font=("Arial", 22, "bold"),text\_color="#FFFFFF")

welcome\_labl.pack(side="left")

action\_frm = ctk.CTkFrame(card\_header, fg\_color="transparent")

action\_frm.pack(side="right")

# refresh

refresh\_btn = ctk.CTkButton(action\_frm,text="Refresh",font=("Arial", 12),fg\_color="#7F5AF0",hover\_color="#614AD3",corner\_radius=8,height=32,width=100,command=lambda: self.task\_viewr.get\_tasks() )

refresh\_btn.pack(side="right", padx=5)

# Task viewer section

tasks\_cont = ctk.CTkFrame(main\_card, fg\_color="transparent")

tasks\_cont.pack(fill="both", expand=True, padx=15, pady=15)

task\_header = ctk.CTkFrame(tasks\_cont, fg\_color="transparent", height=40)

task\_header.pack(fill="x", pady=(0, 10))

task\_title = ctk.CTkLabel(task\_header,text="Your Tasks",font=("Arial", 18, "bold"),text\_color="#FFFFFF")

task\_title.pack(side="left")

self.task\_viewr = TaskView(tasks\_cont, self.user\_id)

self.task\_viewr.pack(fill="both", expand=True, padx=5, pady=5)

def time(self):

crnt\_time = time.strftime("%I:%M %p")

crnt\_date = datetime.now().strftime("%A, %B %d, %Y")

self.time\_labl.configure(text=crnt\_time)

self.date\_labl.configure(text=crnt\_date)

self.after(1000, self.time) # Update every second

def get\_username(self):

try:

connection = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = connection.cursor()

cursor.execute("SELECT username FROM users WHERE user\_id=%s", (self.user\_id,))

username = cursor.fetchone()

connection.close()

return username[0] if username else "Guest"

except Exception as e:

print(f"Error fetching username: {e}")

return "Guest"

if \_\_name\_\_ == '\_\_main\_\_':

user\_id = 1

app = Homepage(user\_id)

app.mainloop()

### Taskviewer.py

import customtkinter as ctk

import mysql.connector

from datetime import datetime, timedelta

from tkcalendar import Calendar

import functools

DB\_CONFIG = {

'user': 'root', # MySQL username

'password': '', # MySQL password

'host': 'localhost', #host

'database': 'nea', # database name

}

class TaskView(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

self.user\_id = user\_id

self.tasks = self.get\_tasks()

self.deleted\_tasks = [] # Stack to store deleted tasks

self.filtered\_tasks = {"Today": [], "Tomorrow": [], "Upcoming": []}

self.filter\_tsk()

self.ui()

# Make the TaskView frame fill the entire width of the parent

self.pack(fill="both", expand=True)

def get\_tasks(self):

connection = mysql.connector.connect(\*\*DB\_CONFIG)

cursor = connection.cursor(dictionary=True)

today = datetime.today().date()

cursor.execute("""

SELECT title, date, time

FROM events

WHERE user\_id = %s AND date >= %s

ORDER BY date ASC

""", (self.user\_id, today.strftime('%Y-%m-%d')))

tasks = cursor.fetchall()

connection.close()

return tasks

def filter\_tsk(self):

today = datetime.today().date()

tomorrow = today + timedelta(days=1)

self.filtered\_tasks = {"Today": [], "Tomorrow": [], "Upcoming": []}

for task in self.tasks:

task\_date = datetime.strptime(task["date"], "%Y-%m-%d").date()

if task\_date == today:

self.filtered\_tasks["Today"].append(task)

elif task\_date == tomorrow:

self.filtered\_tasks["Tomorrow"].append(task)

else:

self.filtered\_tasks["Upcoming"].append(task)

def Mark\_deletion(self, task, checkbox):

# Marks a task for deletion

if task in self.deleted\_tasks:

self.deleted\_tasks.remove(task)

checkbox.configure(fg\_color=("#3B8ED0", "#1F6AA5"))

else:

self.deleted\_tasks.append(task)

checkbox.configure(fg\_color="red")

def save\_deletions(self):

# Delete all marked tasks

if self.deleted\_tasks:

connection = mysql.connector.connect(\*\*DB\_CONFIG)

cursor = connection.cursor()

for task in self.deleted\_tasks:

cursor.execute("DELETE FROM events WHERE user\_id = %s AND title = %s AND date = %s AND time = %s",

(self.user\_id, task['title'], task['date'], task['time']))

connection.commit()

connection.close()

self.tasks = self.get\_tasks()

self.filter\_tsk()

self.clear\_ui()

self.ui()

def add\_task(self):

frame = ctk.CTkFrame(self)

frame.pack(pady=5, fill="both", side="right")

ctk.CTkLabel(frame, text="Task Title:").pack(pady=5)

title\_entry = ctk.CTkEntry(frame)

title\_entry.pack(pady=5, fill="x")

ctk.CTkLabel(frame, text="Select Time:").pack(pady=5)

time\_slots = [datetime.strptime(f"{hour}:00", "%H:%M").strftime("%I:%M %p") for hour in range(7, 24)]

time\_opt = ctk.CTkComboBox(frame, values=time\_slots)

time\_opt.pack(pady=5, fill="x")

time\_opt.set(time\_slots[0])

# Radio Buttons for Date Selection

date\_selection = ctk.CTkFrame(frame)

date\_selection.pack(pady=10, fill="x")

selected\_date\_option = ctk.StringVar(value="Today")

custom\_date = ctk.StringVar(value=datetime.today().date().strftime('%Y-%m-%d'))

today = ctk.CTkRadioButton(date\_selection, text="Today", variable=selected\_date\_option, value="Today",fg\_color="#7f5af0")

tomorrow = ctk.CTkRadioButton(date\_selection, text="Tomorrow", variable=selected\_date\_option, value="Tomorrow",fg\_color="#7f5af0")

custom = ctk.CTkRadioButton(date\_selection, text="Custom", variable=selected\_date\_option, value="Custom",fg\_color="#7f5af0")

today.pack(anchor="w")

tomorrow.pack(anchor="w")

custom.pack(anchor="w")

# Date display for custom date

display\_date = ctk.CTkLabel(date\_selection, text=f"Selected date: {custom\_date.get()}")

display\_date.pack(anchor="w", pady=(5, 0))

def open\_cstm\_date():

if selected\_date\_option.get() == "Custom":

cal\_window = ctk.CTkToplevel(self)

cal\_window.title("Select Date")

cal\_window.geometry("300x300")

cal\_window.attributes('-topmost', True) # Keep window on top

cal = Calendar(cal\_window, selectmode="day", date\_pattern="yyyy-MM-dd")

cal.pack(pady=20)

def select\_date():

selected\_date = cal.get\_date()

custom\_date.set(selected\_date)

display\_date.configure(text=f"Selected date: {selected\_date}")

cal\_window.destroy()

select\_button = ctk.CTkButton(cal\_window, text="Select", command=select\_date)

select\_button.pack(pady=10)

def handler\_date():

if selected\_date\_option.get() == "Custom":

display\_date.pack(anchor="w", pady=(5, 0))

open\_cstm\_date()

else:

display\_date.pack\_forget()

# Configure radio buttons to call handle\_radio\_change

custom.configure(command=handler\_date)

today.configure(command=handler\_date)

tomorrow.configure(command=handler\_date)

# Select custom date button

date\_button = ctk.CTkButton(

date\_selection,

text="Select Custom Date",

command=open\_cstm\_date

)

date\_button.pack(pady=5)

def save\_task():

title = title\_entry.get().strip()

time\_inpt = time\_opt.get().strip()

date\_opt = selected\_date\_option.get()

if not title or not time\_inpt:

return

if date\_opt == "Today":

task\_date = datetime.today().date().strftime('%Y-%m-%d')

elif date\_opt == "Tomorrow":

task\_date = (datetime.today().date() + timedelta(days=1)).strftime('%Y-%m-%d')

else:

task\_date = custom\_date.get() # Get the selected custom date

conv\_24hr = datetime.strptime(time\_inpt, "%I:%M %p").strftime("%H:%M:%S")

connection = mysql.connector.connect(\*\*DB\_CONFIG)

cursor = connection.cursor()

cursor.execute("INSERT INTO events (user\_id, title, date, time) VALUES (%s, %s, %s, %s)",

(self.user\_id, title, task\_date, conv\_24hr))

connection.commit()

connection.close()

self.tasks = self.get\_tasks()

self.filter\_tsk()

self.clear\_ui()

self.ui()

ctk.CTkButton(frame, text="Save Task", command=save\_task,fg\_color="#7f5af0").pack(pady=10)

def clear\_ui(self):

# Clears UI

for widget in self.winfo\_children():

widget.destroy()

def ui(self):

# Creates UI

task\_frame = ctk.CTkFrame(self)

task\_frame.pack(pady=10, padx=10, fill="both", expand=True, side="left") # Main frame for tasks

for section in ["Today", "Tomorrow", "Upcoming"]:

frame = ctk.CTkFrame(task\_frame)

frame.pack(side="left", padx=(0, 10), fill="both", expand=True) # Task sections fill available space

ctk.CTkLabel(frame, text=section, font=("Arial", 16, "bold")).pack(pady=5)

task\_list = self.filtered\_tasks[section]

if task\_list:

for task in task\_list:

task\_frame\_inner = ctk.CTkFrame(frame)

task\_frame\_inner.pack(fill="x", pady=2)

ctk.CTkLabel(task\_frame\_inner, text=f"{task['title']} - {task['time']}").pack(side="left", padx=5)

checkbox = ctk.CTkCheckBox(task\_frame\_inner, text="")

checkbox.configure(command=functools.partial(self.Mark\_deletion, task, checkbox))

checkbox.pack(side="right", padx=5)

else:

ctk.CTkLabel(frame, text="No tasks", font=("Arial", 12, "italic")).pack(pady=2, fill='x')

self.add\_task()

### Pomodoro\_timer.py

import customtkinter as ctk

from PIL import Image, ImageTk

def pomodoro\_timer(main\_app):

# Set appearance mode and color theme

ctk.set\_appearance\_mode("dark")

# Get screen dimensions

screen\_width = main\_app.winfo\_screenwidth()

screen\_height = main\_app.winfo\_screenheight()

# Container frame

container = ctk.CTkFrame(main\_app, fg\_color="#16161A")

container.pack(fill="both", expand=True)

global seconds, running, time

seconds = 0

running = False

time = seconds

# Header section with title

header\_frm = ctk.CTkFrame(container, fg\_color="transparent")

header\_frm.pack(fill="x", padx=20, pady=(20, 10))

title\_labl = ctk.CTkLabel(header\_frm, text="Pomodoro Timer", font=("Arial", 24, "bold"),text\_color="#FFFFFE" )

title\_labl.pack(side="left")

# Main content

content\_frm = ctk.CTkFrame(container, fg\_color="transparent")

content\_frm.pack(fill="both", expand=True, padx=20, pady=10)

# Left side - Timer card

timer\_card = ctk.CTkFrame(content\_frm, fg\_color="#242629" , corner\_radius=15)

timer\_card.pack(fill="both", expand=True, pady=10)

inner\_frm = ctk.CTkFrame(timer\_card, fg\_color="transparent")

inner\_frm.pack(fill="both", expand=True, padx=20, pady=20)

inner\_frm.grid\_columnconfigure(0, weight=3) # Left side (controls)

inner\_frm.grid\_columnconfigure(1, weight=2) # Right side (timer)

# Left side controls and session

controls\_side = ctk.CTkFrame(inner\_frm, fg\_color="transparent")

controls\_side.grid(row=0, column=0, sticky="nsew", padx=(0, 10))

# right side timer

timer\_side = ctk.CTkFrame(inner\_frm, fg\_color="transparent")

timer\_side.grid(row=0, column=1, sticky="nsew", padx=(10, 0))

# Session type

session\_frm = ctk.CTkFrame(controls\_side, fg\_color="transparent")

session\_frm.pack(anchor="center", pady=(20, 30))

session\_type = ctk.CTkLabel(session\_frm, text="Focus Session", font=("Arial", 18, "bold"),text\_color="#7F5AF0")

session\_type.pack()

# Progress bar

progress\_frm = ctk.CTkFrame(controls\_side, fg\_color="transparent")

progress\_frm.pack(fill="x", pady=(0, 30))

progress\_bg = ctk.CTkProgressBar(progress\_frm, width=300,height=12,corner\_radius=6,fg\_color="#333333",progress\_color="#7F5AF0")

progress\_bg.pack()

progress\_bg.set(0) # Start empty

# Timer label -

timer\_labl = ctk.CTkLabel(timer\_side, text="00:00", font=("Arial", 120, "bold"),text\_color="#FFFFFE" )

timer\_labl.place(relx=0.5, rely=0.5, anchor="center")

# Timer selection controls - on left side

select\_timer = ctk.CTkFrame(controls\_side, fg\_color="transparent")

select\_timer.pack(fill="x", pady=(0, 20))

button\_frm = ctk.CTkFrame(controls\_side, fg\_color="transparent")

button\_frm.pack(pady=30)

# Define all functions first to avoid scope issues

def timer\_25\_min():

global seconds, time

seconds = 25 \* 60

time = seconds

timer\_labl.configure(text="25:00") # Update the label

session\_type.configure(text="Focus Session", text\_color="#7F5AF0")

progress\_bg.configure(progress\_color="#7F5AF0")

progress\_bg.set(0)

def timer\_5\_min():

global seconds, time

seconds = 5 \* 60

time = seconds

timer\_labl.configure(text="05:00") # Update the label

session\_type.configure(text="Break Session", text\_color="#2CB67D")

progress\_bg.configure(progress\_color="#2CB67D")

progress\_bg.set(0)

def countdown(remaining\_seconds):

global running, time

time = remaining\_seconds # Keep track of remaining time

if running:

mins, secs = divmod(remaining\_seconds, 60)

timer = f'{mins:02}:{secs:02}'

timer\_labl.configure(text=timer)

# Update progress bar

if seconds > 0: # Prevent division by zero

progress = 1.0 - (remaining\_seconds / seconds)

progress\_bg.set(progress)

if remaining\_seconds > 0:

container.after(1000, countdown, remaining\_seconds - 1)

else:

# Timer completed

timer\_labl.configure(text="Time's up!")

running = False

# Show completion notification

show\_completion()

# Reset buttons to start

reset\_timer()

def timer\_controller():

# Clear existing buttons if they exist

for widget in button\_frm.winfo\_children():

widget.destroy()

# Pause button

pause\_btn = ctk.CTkButton(button\_frm,text="Pause",font=("Arial", 16, "bold"),fg\_color="#555555",text\_color="white",hover\_color="#444444",corner\_radius=8,width=100,height=45,command=pause\_timer)

pause\_btn.pack(side="left", padx=5)

# Reset button

reset\_btn = ctk.CTkButton(button\_frm,text="Reset",font=("Arial", 16, "bold"),fg\_color="#E45858",text\_color="white",hover\_color="#C0392B",corner\_radius=8,width=100,height=45,command=reset\_timer)

reset\_btn.pack(side="left", padx=5)

# Make pause button accessible to other functions

return pause\_btn

def pause\_timer():

global running

running = False

# Replace with unpause button

for widget in button\_frm.winfo\_children():

widget.destroy()

# Unpause button

unpause\_button = ctk.CTkButton(button\_frm,text="Resume",font=("Arial", 16, "bold"),fg\_color="#7F5AF0",text\_color="white",hover\_color="#6344D0",corner\_radius=8,width=100,height=45,command=unpause\_timer)

unpause\_button.pack(side="left", padx=5)

# Reset button

reset\_button = ctk.CTkButton(button\_frm,text="Reset",font=("Arial", 16, "bold"),fg\_color="#E45858",text\_color="white",hover\_color="#C0392B",corner\_radius=8,width=100,height=45,command=reset\_timer)

reset\_button.pack(side="left", padx=5)

def unpause\_timer():

global running

running = True

timer\_controller() # Restore normal buttons

countdown(time) # Resume from where left off

def reset\_timer():

global running, seconds, time

running = False

# Reset display

mins, secs = divmod(seconds, 60)

timer\_labl.configure(text=f"{mins:02}:{secs:02}")

progress\_bg.set(0)

# Reset timer

time = seconds

# Restore start btn

for widget in button\_frm.winfo\_children():

widget.destroy()

# Start button

start\_btn = ctk.CTkButton(button\_frm,text="Start",font=("Arial", 16, "bold"),fg\_color="#7F5AF0",text\_color="white",hover\_color="#6344D0",corner\_radius=8,width=100,height=45,command=start\_timer)

start\_btn.pack(side="left", padx=5)

def show\_completion():

# Create notification popup

notify\_frm = ctk.CTkFrame(container, fg\_color="#242629" ,corner\_radius=15)

notify\_frm.place(relx=0.5, rely=0.5, anchor="center", width=400, height=250)

# Emoji for completion

emoji\_labl = ctk.CTkLabel(notify\_frm,text="🎉",font=("Arial", 40))

emoji\_labl.pack(pady=(20, 0))

# Completion message

complete\_label = ctk.CTkLabel(notify\_frm,text="Time's Up!",font=("Arial", 22, "bold"),text\_color="#FFFFFE" )

complete\_label.pack(pady=(10, 5))

# Subtext

if session\_type.cget("text") == "Focus Session":

msg = "Great job! Take a break before your next focus session."

next\_session = "Break"

next\_color = "#2CB67D"

next\_command = timer\_5\_min

else:

msg = "Break time is over. Ready to get back to work?"

next\_session = "Focus"

next\_color = "#7F5AF0"

next\_command = timer\_25\_min

message\_label = ctk.CTkLabel(notify\_frm,text=msg,font=("Arial", 14),text\_color="#94A1B2" ,wraplength=350)

message\_label.pack(pady=(0, 20))

# Next session button

next\_button = ctk.CTkButton(notify\_frm,text=f"Start {next\_session} Session",font=("Arial", 14),fg\_color=next\_color,text\_color="white",hover\_color="#6344D0" if next\_session == "Focus" else "#247A58",corner\_radius=8,width=200,height=40,command=lambda: [next\_command(), notify\_frm.destroy(), start\_timer()])

next\_button.pack(pady=5)

# Close button

close\_button = ctk.CTkButton(notify\_frm,text="Close",font=("Arial", 14),fg\_color="transparent",text\_color="#FFFFFE" ,hover\_color="#16161A",corner\_radius=8,border\_width=1,border\_color="#94A1B2" ,width=200,height=40,command=notify\_frm.destroy)

close\_button.pack(pady=5)

# Function to start the timer - adapted from original code

def start\_timer():

global running, time, seconds

if seconds > 0: # Ensure a time has been selected

pause\_button = timer\_controller()

if not running:

running = True

time = seconds # Start from the selected time

countdown(time)

else:

# Show notification to select time

notify\_frm = ctk.CTkFrame(container, fg\_color="#242629" ,corner\_radius=15)

notify\_frm.place(relx=0.5, rely=0.5, anchor="center", width=400, height=200)

notify\_labl = ctk.CTkLabel(notify\_frm,text="Please select a time first",font=("Arial", 18, "bold"),text\_color="#FFFFFE" )

notify\_labl.pack(pady=(40, 20))

close\_btn = ctk.CTkButton(notify\_frm,text="Close",font=("Arial", 14),fg\_color="#7F5AF0",text\_color="white",hover\_color="#6344D0",corner\_radius=8,width=120,height=40,command=notify\_frm.destroy)

close\_btn.pack(pady=10)

# Focus Duration

focus\_labl = ctk.CTkLabel(select\_timer, text="Focus Duration", font=("Arial", 14),text\_color="#94A1B2" )

focus\_labl.pack(pady=(0, 5))

focus\_btn = ctk.CTkButton(select\_timer,text="25 min",font=("Arial", 12),fg\_color="#7F5AF0",text\_color="white",hover\_color="#6344D0",corner\_radius=8,width=120,height=30,command=timer\_25\_min)

focus\_btn.pack(pady=(0, 15))

# Break Duration

break\_label = ctk.CTkLabel(select\_timer, text="Break Duration", font=("Arial", 14),text\_color="#94A1B2" )

break\_label.pack(pady=(0, 5))

break\_btn = ctk.CTkButton(select\_timer,text="5 min",font=("Arial", 12),fg\_color="#2CB67D",text\_color="white",hover\_color="#247A58",corner\_radius=8,width=120,height=30,command=timer\_5\_min)

break\_btn.pack()

# Start button

start\_button = ctk.CTkButton(button\_frm,text="Start",font=("Arial", 16, "bold"),fg\_color="#7F5AF0",text\_color="white",hover\_color="#6344D0",corner\_radius=8,width=100,height=45,command=start\_timer)

start\_button.pack(side="left", padx=5)

### Calander.py

import customtkinter as ctk

import tkinter as tk

from tkinter import simpledialog, messagebox

import mysql.connector

import datetime

from calculator\_normal import tm\_allocation

class CustomCalendar(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

tm\_allocation(user\_id)

self.user\_id = user\_id

# Set overall appearance

self.configure(fg\_color="#1E1E2E")

self.header\_frm = ctk.CTkFrame(self, fg\_color="#1E1E2E", corner\_radius=0)

self.header\_frm.pack(fill="x", padx=20, pady=(20, 10))

title\_lbl = ctk.CTkLabel(self.header\_frm, text="My Schedule", font=("Arial", 24, "bold"), text\_color="#EAEAEA")

title\_lbl.pack(side="left")

quick\_add\_btn = ctk.CTkButton(self.header\_frm,text="+ Add Event",font=("Arial", 14),fg\_color="#7C3AED", hover\_color="#6D28D9",corner\_radius=8,width=120,height=36,command=lambda: self.add\_event(None, None) )

quick\_add\_btn.pack(side="right")

self.DB() # DB

# Days and time slots

self.days = ["Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"]

self.hours = [

"07:00 AM", "08:00 AM", "09:00 AM", "10:00 AM", "11:00 AM",

"12:00 PM", "01:00 PM", "02:00 PM", "03:00 PM", "04:00 PM",

"05:00 PM", "06:00 PM", "07:00 PM", "08:00 PM", "09:00 PM",

"10:00 PM", "11:00 PM", "12:00 AM"

]

tdy = datetime.datetime.today()

self.tdy\_day = tdy.weekday()

self.tdy\_date = tdy.date()

tdy = datetime.datetime.today()

self.strt\_week = tdy - datetime.timedelta(days=tdy.weekday())

self.time\_frm = ctk.CTkFrame(self, corner\_radius=12, fg\_color="#282838", height=60)

self.time\_frm.pack(padx=20, pady=(5, 15), fill="x")

self.prev\_week\_btn = ctk.CTkButton(self.time\_frm, text="← Previous Week", command=self.previous\_week, width=140,height=36,font=("Arial", 14),fg\_color="#444454",hover\_color="#555565",corner\_radius=8)

self.prev\_week\_btn.pack(side="left", padx=15, pady=10)

self.week\_labl = ctk.CTkLabel(self.time\_frm, text=self.this\_week(), font=("Arial", 16, "bold"), text\_color="#EAEAEA")

self.week\_labl.pack(side="left", expand=True)

self.next\_week\_btn = ctk.CTkButton(self.time\_frm, text="Next Week →", command=self.next\_week, width=140,height=36,font=("Arial", 14),fg\_color="#444454",hover\_color="#555565",corner\_radius=8)

self.next\_week\_btn.pack(side="right", padx=15, pady=10)

self.time\_frame = ctk.CTkFrame(self, fg\_color="#1E1E2E", corner\_radius=0)

self.time\_frame.pack(fill="x", padx=20)

self.time\_labl = ctk.CTkLabel(self.time\_frame, text="", font=("Arial", 14), text\_color="#AAAAAA")

self.time\_labl.pack(side="right", pady=5)

# Today button

self.today\_btn = ctk.CTkButton(self.time\_frame,text="Today",command=self.today,width=100,height=30,font=("Arial", 13),fg\_color="#7C3AED",hover\_color="#6D28D9",corner\_radius=8)

self.today\_btn.pack(side="left", pady=5)

self.calendar\_frm = ctk.CTkFrame(self, fg\_color="#1E1E2E", corner\_radius=0)

self.calendar\_frm.pack(fill="both", expand=True, padx=20, pady=10)

self.calendar\_frm = ctk.CTkFrame(self.calendar\_frm, corner\_radius=12, fg\_color="#282838",border\_width=1,border\_color="#3E3E4E")

self.calendar\_frm.pack(fill="both", expand=True, padx=0, pady=0)

self.events = {}

self.event\_cell = {}

self.calendar()

self.time()

def this\_week(self):

start\_date = self.strt\_week.date()

end\_date = start\_date + datetime.timedelta(days=6)

if start\_date.month == end\_date.month:

return f"{start\_date.strftime('%B %d')} - {end\_date.strftime('%d, %Y')}"

else:

return f"{start\_date.strftime('%B %d')} - {end\_date.strftime('%B %d, %Y')}"

def today(self):

today = datetime.datetime.today()

self.strt\_week = today - datetime.timedelta(days=today.weekday())

self.week\_labl.configure(text=self.this\_week())

self.calendar()

def DB(self):

self.conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

self.cursor = self.conn.cursor()

def calendar(self):

# Clear

for widget in self.calendar\_frm.winfo\_children():

widget.destroy()

header\_frm = ctk.CTkFrame(self.calendar\_frm, fg\_color="#282838", corner\_radius=0, height=60)

header\_frm.grid(row=0, column=0, columnspan=8, sticky="ew")

time = ctk.CTkLabel(header\_frm,text="",width=80,fg\_color="#282838",corner\_radius=0)

time.grid(row=0, column=0, padx=2, pady=2, sticky="nsew")

for col, day in enumerate(self.days):

day\_date = self.strt\_week + datetime.timedelta(days=col)

today = day\_date.date() == self.tdy\_date

header\_frm = ctk.CTkFrame(self.calendar\_frm,fg\_color="#7C3AED" if today else "#373747",corner\_radius=8 if today else 0)

header\_frm.grid(row=0, column=col + 1, padx=4, pady=4, sticky="nsew")

day\_name = ctk.CTkLabel(header\_frm,text=day,font=("Arial", 14, "bold"),text\_color="#FFFFFF")

day\_name.pack(pady=(8, 0))

date\_num = ctk.CTkLabel(header\_frm,text=str(day\_date.day),font=("Arial", 18, "bold"),text\_color="#FFFFFF")

date\_num.pack(pady=(0, 8))

# Create tiem slot

for row, hour in enumerate(self.hours):

time\_label = ctk.CTkLabel(master=self.calendar\_frm,text=hour,fg\_color="#373747",text\_color="#CCCCCC",font=("Arial", 12),width=80,height=30,corner\_radius=4)

time\_label.grid(row=row + 1, column=0, padx=4, pady=4, sticky="nsew")

for col in range(8):

self.calendar\_frm.grid\_columnconfigure(col, weight=1 if col > 0 else 0, uniform="equal" if col > 0 else "")

for row in range(1, len(self.hours) + 1):

self.calendar\_frm.grid\_rowconfigure(row, weight=1, uniform="equal")

self.event\_cell = {}

self.get\_event()

# loops itself until all the necessary cells are made (this is the cells / events that the user can add)

for row in range(1, len(self.hours) + 1):

for col in range(len(self.days)):

today\_cell = col == self.tdy\_day and (self.strt\_week + datetime.timedelta(days=col)).date() == self.tdy\_date

event\_frm = ctk.CTkFrame(master=self.calendar\_frm,corner\_radius=6,fg\_color="#2E2E3E",border\_width=1,border\_color="#3E3E4E")

event\_frm.grid(row=row, column=col + 1, padx=4, pady=4, sticky="nsew")

if today\_cell:

indicator = ctk.CTkFrame(event\_frm,width=4,corner\_radius=2,fg\_color="#7C3AED" )

indicator.place(x=0, y=0, relheight=1)

# binding mouse input

event\_frm.bind("<Button-1>", lambda event, r=row - 1, c=col: self.add\_edit\_evnt(r, c))

self.event\_cell[(row - 1, col)] = event\_frm

event = self.events.get((row - 1, col))

if event:

self.show\_event(row - 1, col, event)

def show\_event(self, row, col, event\_title):

event\_frame = self.event\_cell[(row, col)]

# Clear existing content

for widget in event\_frame.winfo\_children():

widget.destroy()

# Create event button

event\_button = ctk.CTkButton(master=event\_frame, text=event\_title,corner\_radius=6, fg\_color="#7C3AED", hover\_color="#6D28D9",text\_color="white", font=("Arial", 12),command=lambda: self.edit\_event(row, col))

event\_button.place(relx=0.5, rely=0.5, anchor="center", relwidth=0.9, relheight=0.8)

def get\_event(self):

# Fetch events from the db

start\_date = self.strt\_week.date()

end\_date = start\_date + datetime.timedelta(days=6)

self.cursor.execute("""

SELECT date, time, title FROM events WHERE date BETWEEN %s AND %s AND user\_id = %s

""", (start\_date, end\_date, self.user\_id))

rows = self.cursor.fetchall()

self.events = {}

# dynamic obhject genration

for date, time, title in rows:

try:

dt = datetime.datetime.strptime(time, "%H:%M:%S")

formatted\_time = dt.strftime("%I:%M %p")

if formatted\_time in self.hours:

time\_index = self.hours.index(formatted\_time)

day\_offset = (datetime.datetime.strptime(date, "%Y-%m-%d").date() - start\_date).days

self.events[(time\_index, day\_offset)] = title

except ValueError as e:

print(f"Error processing event with time '{time}': {e}")

# show the last week

def previous\_week(self):

self.strt\_week -= datetime.timedelta(weeks=1)

self.week\_labl.configure(text=self.this\_week())

self.calendar()

# changes the calander to the next week

def next\_week(self):

# Go to the next week

self.strt\_week += datetime.timedelta(weeks=1)

self.week\_labl.configure(text=self.this\_week())

self.calendar()

# adds the events or go through changes in the event

def add\_edit\_evnt(self, row, col):

# Check if there's already an event in the selected cell

current\_event = self.events.get((row, col), None)

def decision():

action = self.action\_txt.get().strip().lower()

self.action\_frame.destroy() # Close the prompt frame after getting input

if action == 'yes':

self.edit\_event(row, col)

elif action == 'no':

pass # Do nothing and exit

else:

messagebox.showerror("Invalid Input", "Please type 'Yes' or 'No'.") # Show an error message

self.add\_edit\_evnt(row, col) # Reopen the prompt for correct input

# Polymorphism: Same method handles both adding AND editing

if current\_event:

self.action\_frame = ctk.CTkFrame(master=self, width=350, height=220, corner\_radius=12, fg\_color="#282838",border\_width=1,border\_color="#3E3E4E")

self.action\_frame.place(relx=0.5, rely=0.4, anchor="center")

self.action\_label = ctk.CTkLabel(master=self.action\_frame,text=f"Event at {self.hours[row]} on {self.days[col]}:\n{current\_event}\n\nDo you want to edit this event? (Yes/No)",font=("Arial", 14),text\_color="#EAEAEA")

self.action\_label.place(relx=0.5, rely=0.4, anchor="center")

self.action\_txt = ctk.CTkEntry(master=self.action\_frame, placeholder\_text="Type Yes or No",width=200,height=35,font=("Arial", 14),corner\_radius=6,fg\_color="#373747",border\_color="#4E4E5E")

self.action\_txt.place(relx=0.5, rely=0.65, anchor="center")

self.action\_button = ctk.CTkButton(master=self.action\_frame, command=decision, text="Confirm",width=120,height=35,font=("Arial", 14),fg\_color="#7C3AED",hover\_color="#6D28D9",corner\_radius=6)

self.action\_button.place(relx=0.5, rely=0.85, anchor="center")

else:

self.add\_event(row, col)

def add\_event(self, row, col):

self.add\_event\_frame = ctk.CTkFrame(master=self, width=450, height=350, corner\_radius=12, fg\_color="#282838",border\_width=1,border\_color="#3E3E4E")

self.add\_event\_frame.place(relx=0.5, rely=0.5, anchor="center")

# Frame title

frame\_title = ctk.CTkLabel(master=self.add\_event\_frame, text="Create New Event", font=("Arial", 20, "bold"), text\_color="#EAEAEA")

frame\_title.place(relx=0.5, rely=0.1, anchor="center")

# (Dynamic object generation)

if row is None or col is None:

date\_label = ctk.CTkLabel(master=self.add\_event\_frame, text="Date:", font=("Arial", 14), text\_color="#EAEAEA")

date\_label.place(relx=0.1, rely=0.25, anchor="w")

week\_dates = []

for i in range(7):

date = self.strt\_week + datetime.timedelta(days=i)

week\_dates.append(f"{self.days[i]} ({date.day}/{date.month})")

self.date\_combo = ctk.CTkComboBox(master=self.add\_event\_frame,values=week\_dates,width=200,height=35,font=("Arial", 14),corner\_radius=6,dropdown\_font=("Arial", 14))

self.date\_combo.place(relx=0.6, rely=0.25, anchor="w")

self.date\_combo.set(week\_dates[0]) # Default to first day

time\_label = ctk.CTkLabel(master=self.add\_event\_frame, text="Time:", font=("Arial", 14), text\_color="#EAEAEA")

time\_label.place(relx=0.1, rely=0.35, anchor="w")

self.time\_combo = ctk.CTkComboBox(master=self.add\_event\_frame,values=self.hours,width=200,height=35,font=("Arial", 14),corner\_radius=6,dropdown\_font=("Arial", 14))

self.time\_combo.place(relx=0.6, rely=0.35, anchor="w")

self.time\_combo.set(self.hours[0])

title\_y\_position = 0.45

else:

slot\_info = ctk.CTkLabel(master=self.add\_event\_frame, text=f"Day: {self.days[col]} | Time: {self.hours[row]}", font=("Arial", 14), text\_color="#AAAAAA")

slot\_info.place(relx=0.5, rely=0.2, anchor="center")

title\_y\_position = 0.35

# Title Label and Entry

title\_label = ctk.CTkLabel(master=self.add\_event\_frame, text="Event Title:", font=("Arial", 14), text\_color="#EAEAEA")

title\_label.place(relx=0.1, rely=title\_y\_position, anchor="w")

self.event\_title\_entry = ctk.CTkEntry(master=self.add\_event\_frame, placeholder\_text="Enter event title",width=300,height=35,font=("Arial", 14),corner\_radius=6,fg\_color="#373747",border\_color="#4E4E5E")

self.event\_title\_entry.place(relx=0.5, rely=title\_y\_position+0.1, anchor="center")

# Buttons

button\_frame = ctk.CTkFrame(master=self.add\_event\_frame,fg\_color="transparent")

button\_frame.place(relx=0.5, rely=0.85, anchor="center")

self.save\_button = ctk.CTkButton(master=button\_frame,text="Save Event",command=lambda: self.save\_event(row, col),width=150,height=40,font=("Arial", 14),fg\_color="#7C3AED",hover\_color="#6D28D9",corner\_radius=6)

self.save\_button.pack(side="left", padx=10)

self.cancel\_button = ctk.CTkButton(master=button\_frame,text="Cancel",command=lambda: self.add\_event\_frame.place\_forget(),width=120,height=40,font=("Arial", 14),fg\_color="#444454",hover\_color="#555565",corner\_radius=6)

self.cancel\_button.pack(side="left", padx=10)

def save\_event(self, row, col):

# Save the event details

event\_name = self.event\_title\_entry.get().strip()

if row is None or col is None:

date\_str = self.date\_combo.get()

day\_indx = self.days.index(date\_str.split()[0][:3]) # Get day index from the selected text

event\_date = (self.strt\_week + datetime.timedelta(days=day\_indx)).date()

event\_time = self.time\_combo.get()

time\_indx = self.hours.index(event\_time)

else:

event\_date = (self.strt\_week + datetime.timedelta(days=col)).date()

event\_time = self.hours[row]

time\_indx = row

try:

# convert 12 format to 24 format

event\_time\_24hr = datetime.datetime.strptime(event\_time, "%I:%M %p").strftime("%H:%M:%S")

if event\_name:

try:

self.cursor.execute("""

INSERT INTO events (user\_id, title, date, time)

VALUES (%s, %s, %s, %s)

""", (self.user\_id, event\_name, event\_date, event\_time\_24hr))

self.conn.commit()

if row is not None and col is not None:

self.events[(row, col)] = {

"title": event\_name,

}

# Refresh

self.calendar()

messagebox.showinfo("Success", "Event added successfully!")

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to save event: {e}")

finally:

# Hide the event entry frame after saving

self.add\_event\_frame.destroy()

else:

messagebox.showwarning("Warning", "Event title cannot be empty.")

except ValueError as e:

print(f"Error converting time '{event\_time}' to 24-hour format: {e}")

def show\_event(self, row, col, event\_name):

event\_frame = self.event\_cell[(row, col)]

# Clear

for widget in event\_frame.winfo\_children():

widget.destroy()

if event\_name:

event\_button = ctk.CTkButton(master=event\_frame, text=event\_name, fg\_color="#7C3AED", hover\_color="#6D28D9",text\_color="white", font=("Arial", 12),corner\_radius=6,command=lambda: self.edit\_event(row, col))

event\_button.place(relx=0.5, rely=0.5, anchor="center", relwidth=0.9, relheight=0.8)

else:

# Empty cell

pass

def edit\_event(self, row, col):

# Retrieve from the db

current\_event = self.events.get((row, col), None)

if current\_event:

current\_title = current\_event["title"]

else:

current\_title = ""

# pop up frame

edit\_popup = ctk.CTkFrame(master=self, width=400, height=300, corner\_radius=12, fg\_color="#282838",border\_width=1,border\_color="#3E3E4E")

edit\_popup.place(relx=0.5, rely=0.5, anchor="center")

title = ctk.CTkLabel(master=edit\_popup, text="Edit Event", font=("Arial", 20, "bold"), text\_color="#EAEAEA")

title.place(relx=0.5, rely=0.15, anchor="center")

name\_labl = ctk.CTkLabel(edit\_popup, text="Event Title:", font=("Arial", 14), text\_color="#EAEAEA")

name\_labl.place(relx=0.1, rely=0.3, anchor="w")

name\_entry = ctk.CTkEntry(edit\_popup,width=250,height=35,font=("Arial", 14),corner\_radius=6,fg\_color="#373747",border\_color="#4E4E5E")

name\_entry.insert(0, current\_title) # Set current event title as default

name\_entry.place(relx=0.5, rely=0.4, anchor="center")

time\_labl = ctk.CTkLabel(edit\_popup, text="Select Time:", font=("Arial", 14), text\_color="#EAEAEA")

time\_labl.place(relx=0.1, rely=0.55, anchor="w")

time\_combo = ctk.CTkComboBox(edit\_popup, values=self.hours, width=250,height=35,font=("Arial", 14),corner\_radius=6,dropdown\_font=("Arial", 14))

time\_combo.set(self.hours[row]) # Set the current time slot as default

time\_combo.place(relx=0.5, rely=0.65, anchor="center")

# Buttons

button\_frm = ctk.CTkFrame(edit\_popup, fg\_color="transparent")

button\_frm.place(relx=0.5, rely=0.85, anchor="center")

# Confirm button to save the changes

confirm\_btn = ctk.CTkButton(button\_frm, text="Save Changes", width=150,height=40,font=("Arial", 14),fg\_color="#7C3AED",hover\_color="#6D28D9",corner\_radius=6,command=lambda: self.confirm\_edit(row, col, name\_entry.get(), time\_combo.get(), edit\_popup))

confirm\_btn.pack(side="left", padx=10)

delete\_btn = ctk.CTkButton(button\_frm, text="Delete", width=100,height=40,font=("Arial", 14),fg\_color="#E11D48", hover\_color="#BE123C",corner\_radius=6,command=lambda: self.delete\_event(row, col, edit\_popup))

delete\_btn.pack(side="left", padx=10)

def delete\_event(self, row, col, popup):

if messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete this event?"):

try:

event\_date = (self.strt\_week + datetime.timedelta(days=col)).date()

old\_time\_obj = datetime.datetime.strptime(self.hours[row], "%I:%M %p")

old\_time = old\_time\_obj.strftime("%H:%M:%S")

self.cursor.execute("""

DELETE FROM events

WHERE user\_id = %s AND date = %s AND time = %s

""", (self.user\_id, event\_date, old\_time))

self.conn.commit()

# Remove from local events dictionary

if (row, col) in self.events:

del self.events[(row, col)]

# Close popup

popup.destroy()

# Refresh calendar

self.calendar()

messagebox.showinfo("Success", "Event deleted successfully!")

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to delete event: {e}")

def confirm\_edit(self, row, col, new\_name, new\_time, popup):

if new\_name:

if new\_time != self.hours[row]:

new\_time\_index = self.hours.index(new\_time)

if (row, col) in self.events:

del self.events[(row, col)]

self.events[(new\_time\_index, col)] = {"title": new\_name}

# Update the db

try:

event\_date = (self.strt\_week + datetime.timedelta(days=col)).date()

old\_time = datetime.datetime.strptime(self.hours[row], "%I:%M %p").strftime("%H:%M:%S")

new\_time = datetime.datetime.strptime(new\_time, "%I:%M %p").strftime("%H:%M:%S")

self.cursor.execute("""

UPDATE events

SET title = %s, time = %s

WHERE date = %s AND time = %s AND user\_id = %s

""", (new\_name, new\_time, event\_date, old\_time, self.user\_id))

self.conn.commit()

# Refresh the calendar

self.calendar()

messagebox.showinfo("Success", "Event updated successfully!")

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to update event: {e}")

else:

self.events[(row, col)] = {"title": new\_name}

# Update the db

try:

event\_date = (self.strt\_week + datetime.timedelta(days=col)).date()

old\_time = datetime.datetime.strptime(self.hours[row], "%I:%M %p").strftime("%H:%M:%S")

self.cursor.execute("""

UPDATE events

SET title = %s

WHERE date = %s AND time = %s AND user\_id = %s

""", (new\_name, event\_date, old\_time, self.user\_id))

self.conn.commit()

self.calendar()

messagebox.showinfo("Success", "Event updated successfully!")

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to update event: {e}")

popup.destroy()

else:

messagebox.showwarning("Warning", "Event name cannot be empty.")

def time(self):

curnt\_time = datetime.datetime.now().strftime("%A, %B %d, %Y | %I:%M:%S %p")

self.time\_labl.configure(text=curnt\_time)

# Call this function every 1000 milliseconds (1 second)

self.after(1000, self.time)

### Tree\_game\_v.py

import tkinter as tk

from tkinter import messagebox, font, ttk

from PIL import Image, ImageTk

import os

import random

import math

import requests

import json

import sys

class TreeNode:

def \_\_init\_\_(self, question, answers, correct\_answer, platform, difficulty=1):

self.question = question

self.answers = answers

self.correct\_answer = correct\_answer

self.platform = platform

self.difficulty = difficulty

self.children = {

"correct": None, # correct answer

"incorrect": None # wrong answer

}

def add\_child(self, node, path\_type="correct"):

# cprrect and wrong nodes

self.children[path\_type] = node

return node

class TreeDefenderGame(tk.Frame):

def \_\_init\_\_(self, parent, \*\*kwargs):

super().\_\_init\_\_(parent, \*\*kwargs)

# Store the parent

self.parent = parent

# Game background

self.bg\_color = "#1A2A3A" # Darker blue background

self.configure(bg=self.bg\_color)

self.main\_frm = tk.Frame(self, bg=self.bg\_color)

self.main\_frm.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Game state

self.boss\_hlth = 3

self.player\_hlth = 3

self.crnt\_platform\_indx = -1 # Track player position

self.attacking = False

self.allowed\_move = True # New flag to control player movement

self.score = 0

self.level = 1

self.platforms = []

self.crnt\_node = None # Current question node

self.debug\_mode = True # Enable debug output

# Available subjects

self.subj = ["Math", "Geography", "Science", "History", "Literature", "Mixed"]

self.selected\_subj = tk.StringVar(value=self.subj[0])

self.assests()

# GUI Setup

self.ui()

def assests(self):

# Create placeholder images if they don't exist

if not os.path.exists("window.png"):

self.backup\_entity("player.png", (40, 40), "blue")

if not os.path.exists("boss.png"):

self.backup\_entity("home.png", (100, 100), "red")

def backup\_entity(self, filename, size, color):

from PIL import Image, ImageDraw

img = Image.new('RGBA', size, color)

img.save(filename)

def ui(self):

self.title\_font = font.Font(family="Arial", size=24, weight="bold")

self.question\_font = font.Font(family="Arial", size=16)

self.button\_font = font.Font(family="Arial", size=14, weight="bold")

# Grid configuration to ensure resizing works

self.main\_frm.grid\_columnconfigure(0, weight=1)

# Game

self.title\_label = tk.Label(self.main\_frm, text="TREE DEFENDER",

font=self.title\_font,

fg="#F1C40F", bg=self.bg\_color)

self.title\_label.grid(row=0, column=0, pady=(10, 10), sticky="ew")

self.subject\_frame = tk.Frame(self.main\_frm, bg=self.bg\_color)

self.subject\_frame.grid(row=1, column=0, pady=(0, 10), sticky="ew")

self.subject\_frame.grid\_columnconfigure(0, weight=1)

self.subject\_frame.grid\_columnconfigure(1, weight=1)

self.subject\_labl = tk.Label(self.subject\_frame, text="Select Subject:", font=self.button\_font, fg="white", bg=self.bg\_color)

self.subject\_labl.grid(row=0, column=0, padx=10, sticky="e")

self.subject\_select = ttk.Combobox(self.subject\_frame, textvariable=self.selected\_subj,values=self.subj, width=15, font=self.button\_font)

self.subject\_select.grid(row=0, column=1, padx=10, sticky="w")

self.canvas\_frm = tk.Frame(self.main\_frm, bg=self.bg\_color, bd=2, relief=tk.SUNKEN)

self.canvas\_frm.grid(row=2, column=0, pady=10, sticky="nsew")

self.main\_frm.grid\_rowconfigure(2, weight=3) # Give canvas most of the vertical space

self.canvas = tk.Canvas(self.canvas\_frm, bg=self.bg\_color, highlightthickness=0)

self.canvas.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

# Player and boss elements

self.player\_img = ImageTk.PhotoImage(Image.open("pics/player.png").resize((50, 50)))

self.player = self.canvas.create\_image(100, 350, image=self.player\_img)

self.boss\_img = ImageTk.PhotoImage(Image.open("pics/boss.png").resize((120, 120)))

self.boss = self.canvas.create\_image(650, 100, image=self.boss\_img)

# Create platforms

platform\_width = 100

platform\_height = 40

platform\_colors = ["#1ABC9C", "#2ECC71", "#3498DB", "#9B59B6"]

for i in range(4):

x = 150 + i \* 150

y = 350

platform = self.canvas.create\_rectangle(

x - platform\_width/2, y + 30,

x + platform\_width/2, y + 30 + platform\_height,

fill=platform\_colors[i], outline="#FFFFFF", width=2

)

self.platforms.append({"shape": platform, "x": x, "y": y + 30 + platform\_height/2})

# Add platform number

self.canvas.create\_text(x, y + 30 + platform\_height/2,

text=str(i+1), fill="white", font=self.button\_font)

# Health and score displays

heart\_emoji = "❤️"

self.boss\_health\_txt = self.canvas.create\_text(

650, 40, text=f"Boss Health: {self.boss\_hlth} {heart\_emoji}",

fill="#E74C3C", font=self.title\_font

)

self.player\_health\_txt = self.canvas.create\_text(

150, 40, text=f"Your Health: {self.player\_hlth} {heart\_emoji}",

fill="#2ECC71", font=self.title\_font

)

self.score\_txt = self.canvas.create\_text(

400, 40, text=f"Score: {self.score} | Level: {self.level}",

fill="#F1C40F", font=self.title\_font

)

if self.debug\_mode:

self.debug\_text = self.canvas.create\_text(

400, 380, text="Debug: Ready",

fill="#FFFFFF", font=("Arial", 10)

)

# Question display

self.question\_frm = tk.Frame(self.main\_frm, bg=self.bg\_color, padx=20, pady=10,

bd=2, relief=tk.GROOVE)

self.question\_frm.grid(row=3, column=0, pady=(10, 10), sticky="ew")

self.question\_frm.grid\_columnconfigure(0, weight=1)

self.question\_lbl = tk.Label(self.question\_frm, text="", font=self.question\_font, wraplength=700, fg="white", bg=self.bg\_color)

self.question\_lbl.pack(pady=10, fill=tk.X)

self.controls\_frm = tk.Frame(self.main\_frm, bg=self.bg\_color)

self.controls\_frm.grid(row=4, column=0, pady=(10, 10), sticky="ew")

self.controls\_frm.grid\_columnconfigure(0, weight=1)

# Button frame

self.btn\_frm = tk.Frame(self.controls\_frm, bg=self.bg\_color)

self.btn\_frm.pack(fill=tk.X)

self.btn\_frm.grid\_columnconfigure(0, weight=1)

self.btn\_frm.grid\_columnconfigure(1, weight=1)

self.btn\_frm.grid\_columnconfigure(2, weight=1)

self.btn\_frm.grid\_columnconfigure(3, weight=1)

self.btns = []

for i in range(4):

btn = tk.Button(self.btn\_frm, text="", width=15, height=2,bg=platform\_colors[i], fg="white", font=self.button\_font,

command=lambda idx=i: self.check\_answer(idx))

btn.grid(row=0, column=i, padx=10, sticky="ew")

self.btns.append(btn)

self.strt\_btn\_frm = tk.Frame(self.main\_frm, bg=self.bg\_color)

self.strt\_btn\_frm.grid(row=5, column=0, pady=(10, 10), sticky="ew")

# Start button with glow effect

self.strt\_btn = tk.Button(self.strt\_btn\_frm, text="Start Game", font=self.button\_font,

bg="#F1C40F", fg="#000000", relief=tk.RAISED, bd=3,

padx=20, pady=10, command=self.start\_game)

self.strt\_btn.pack()

# Loading indicator and tree path visualization in status frame

self.status\_frm = tk.Frame(self.main\_frm, bg=self.bg\_color, bd=1, relief=tk.SUNKEN)

self.status\_frm.grid(row=6, column=0, pady=(5, 5), sticky="ew")

self.loading\_labl = tk.Label(self.status\_frm, text="", font=self.button\_font,

fg="#F1C40F", bg=self.bg\_color)

self.loading\_labl.pack(pady=5)

self.tree\_path = tk.Label(self.status\_frm, text="Question Tree Path: Root",

font=("Arial", 10), fg="#F1C40F", bg=self.bg\_color)

self.tree\_path.pack(pady=5)

# Add blinking effect to start button

self.str\_btn\_animation()

self.canvas.bind("<Configure>", self.on\_canvas\_resize)

def on\_canvas\_resize(self, event):

# depending on the size of the window the boss will need to chang position and also the player has to change the position

width = event.width

height = event.height

# Calculate new positions

canvas\_center\_x = width / 2

boss\_x = width - 150

player\_x = 100

# Update

self.canvas.coords(self.boss, boss\_x, 100)

self.canvas.coords(self.player, player\_x, height - 80)

self.canvas.coords(self.boss\_health\_txt, boss\_x, 40)

self.canvas.coords(self.player\_health\_txt, player\_x + 50, 40)

self.canvas.coords(self.score\_txt, canvas\_center\_x, 40)

# Update position

if self.debug\_mode:

self.canvas.coords(self.debug\_text, canvas\_center\_x, height - 20)

platform\_spacing = (width - 300) / 3

for i in range(4):

x = 150 + i \* platform\_spacing

y = height - 80 + 30 # Position below player

# Update platform rectangle

platform\_width = 100

platform\_height = 40

self.canvas.coords(

self.platforms[i]["shape"],

x - platform\_width/2, y,

x + platform\_width/2, y + platform\_height

)

# Update platform position data

self.platforms[i]["x"] = x

self.platforms[i]["y"] = y + platform\_height/2

# Update platform number text (need to get the text item first)

# Get all items at platform position to find the text

items = self.canvas.find\_overlapping(x-1, y+platform\_height/2-1, x+1, y+platform\_height/2+1)

for item in items:

if self.canvas.type(item) == "text":

self.canvas.coords(item, x, y + platform\_height/2)

def debug\_log(self, message):

#Helper function me to see where the program ends up in the code

if self.debug\_mode:

print(f"DEBUG: {message}")

self.canvas.itemconfig(self.debug\_text, text=f"Debug: {message}")

def str\_btn\_animation(self):

current\_color = self.strt\_btn.cget("background")

next\_color = "#F39C12" if current\_color == "#F1C40F" else "#F1C40F"

self.strt\_btn.config(background=next\_color)

self.after(500, self.str\_btn\_animation)

def make\_qst\_tree(self, subject):

#Create a tree of questions

try:

self.debug\_log(f"Creating question tree for subject: {subject}")

self.loading\_labl.config(text="Building question tree...")

# Get questions from API or fallback

questions = self.get\_qst\_api(subject, count=15)

# Group questions by difficulty (if available from API), or assign difficulties

easy\_questions = [q for q in questions if q.difficulty <= 1]

medium\_questions = [q for q in questions if q.difficulty == 2]

hard\_questions = [q for q in questions if q.difficulty >= 3]

# redistrubution if not enough question

if len(easy\_questions) < 5:

random.shuffle(questions)

easy\_questions = questions[:5]

for q in easy\_questions:

q.difficulty = 1

if len(medium\_questions) < 5:

remaining = [q for q in questions if q not in easy\_questions]

if len(remaining) >= 5:

medium\_questions = remaining[:5]

else:

medium\_questions = remaining + random.sample(easy\_questions, 5 - len(remaining))

for q in medium\_questions:

q.difficulty = 2

if len(hard\_questions) < 5:

used = easy\_questions + medium\_questions

remaining = [q for q in questions if q not in used]

if len(remaining) >= 5:

hard\_questions = remaining[:5]

else:

hard\_questions = remaining + random.sample(medium\_questions, 5 - len(remaining))

for q in hard\_questions:

q.difficulty = 3

#5 question limit

easy\_questions = easy\_questions[:5]

medium\_questions = medium\_questions[:5]

hard\_questions = hard\_questions[:5]

# Create question adn levels

root = easy\_questions[0]

self.debug\_log("Building question tree structure...")

# Track nodes

added\_nodes = {root}

# filter question

current\_correct = root

for i in range(1, len(easy\_questions)):

if easy\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(easy\_questions[i], "correct")

added\_nodes.add(easy\_questions[i])

if medium\_questions and medium\_questions[0] not in added\_nodes:

current\_correct = current\_correct.add\_child(medium\_questions[0], "correct")

added\_nodes.add(medium\_questions[0])

for i in range(1, len(medium\_questions)):

if medium\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(medium\_questions[i], "correct")

added\_nodes.add(medium\_questions[i])

if hard\_questions and hard\_questions[0] not in added\_nodes:

current\_correct = current\_correct.add\_child(hard\_questions[0], "correct")

added\_nodes.add(hard\_questions[0])

for i in range(1, len(hard\_questions)):

if hard\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(hard\_questions[i], "correct")

added\_nodes.add(hard\_questions[i])

# create the incorrect paths

nodes\_process = [root]

while nodes\_process:

node = nodes\_process.pop(0)

if node.children["correct"]:

nodes\_process.append(node.children["correct"])

if node.difficulty == 1:

avl = [q for q in easy\_questions if q not in added\_nodes]

if not avl:

node.children["incorrect"] = root

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

elif node.difficulty == 2:

avl = [q for q in easy\_questions + medium\_questions[:2] if q not in added\_nodes]

if not avl:

node.children["incorrect"] = medium\_questions[0] if medium\_questions else easy\_questions[0]

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

elif node.difficulty == 3:

avl = [q for q in medium\_questions if q not in added\_nodes]

if not avl:

node.children["incorrect"] = medium\_questions[0] if medium\_questions else root

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

self.loading\_labl.config(text="")

self.debug\_log("Question tree built successfully")

return root

except Exception as e:

self.debug\_log(f"Error creating question tree: {str(e)}")

# Create a simple fallback tree

return self.make\_backup\_tree(subject)

def make\_backup\_tree(self, subject):

#Create a simple fallback tree if API or tree creation fails

backup\_questions = self.get\_backup\_qsts(subject)

if not backup\_questions:

return None

root = backup\_questions[0]

current = root

for i in range(1, len(backup\_questions)):

current.add\_child(backup\_questions[i], "correct")

current = backup\_questions[i]

# Create loops for incorrect answers

for i in range(len(backup\_questions)):

if i >= 3:

target\_index = i - 3

backup\_questions[i].add\_child(backup\_questions[target\_index], "incorrect")

else:

backup\_questions[i].add\_child(root, "incorrect")

return root

def get\_qst\_api(self, subject, count=15):

#Fetch questions from API

self.loading\_labl.config(text="Loading questions...")

try:

# This is a placeholder for the actual API call

api\_url = "<https://opentdb.com/api.php>"

# Map the subjects to API categories

category\_map = {

"Math": 19,

"Geography": 22,

"Science": 17,

"History": 23,

"Literature": 10,

"Mixed": None

}

# Prepare parameters

params = {

"amount": count,

"type": "multiple" # Multiple choice questions

}

# Add category if not Mixed

if subject != "Mixed" and category\_map[subject]:

params["category"] = category\_map[subject]

# Make API request

self.debug\_log(f"Fetching questions for subject: {subject}")

response = requests.get(api\_url, params=params)

data = response.json()

# Check if we got valid response

if response.status\_code == 200 and data["response\_code"] == 0:

questions = []

for i, item in enumerate(data["results"]):

# Get all answers and shuffle them

all\_answers = [item["correct\_answer"]] + item["incorrect\_answers"]

random.shuffle(all\_answers)

# Find the index of the correct answer

correct\_index = all\_answers.index(item["correct\_answer"])

# Assign difficulty based on position in results (first 5 easy, next 5 medium, last 5 hard)

difficulty = 1 if i < 5 else (2 if i < 10 else 3)

# Create TreeNode

node = TreeNode(

question=item["question"],

answers=all\_answers,

correct\_answer=correct\_index,

platform=item["category"],

difficulty=difficulty

)

questions.append(node)

self.loading\_labl.config(text="")

return questions

else:

# API error - use fallback questions

self.debug\_log(f"API Error: {data.get('response\_code', 'Unknown')}")

self.loading\_labl.config(text="Could not load questions. Using defaults.")

return self.get\_backup\_qsts(subject)

except Exception as e:

# Handle any errors (network issues, etc.)

self.debug\_log(f"Error fetching questions: {str(e)}")

self.loading\_labl.config(text="Network error. Using default questions.")

return self.get\_backup\_qsts(subject)

def get\_backup\_qsts(self, subject):

#Provide fallback questions

fallback\_questions = []

if subject == "Math" or subject == "Mixed":

fallback\_questions.extend([

TreeNode("What is 5 + 3?", ["6", "7", "8", "9"], 2, "Math", 1),

TreeNode("What is 12 × 3?", ["33", "36", "39", "42"], 1, "Math", 1),

TreeNode("What is the square root of 64?", ["6", "7", "8", "9"], 2, "Math", 2),

TreeNode("What is 20% of 80?", ["12", "16", "20", "24"], 1, "Math", 2),

TreeNode("If x + 5 = 12, what is x?", ["5", "6", "7", "8"], 2, "Math", 3)

])

if subject == "Geography" or subject == "Mixed":

fallback\_questions.extend([

TreeNode("What is the capital of France?", ["Berlin", "Madrid", "Paris", "Rome"], 2, "Geography", 1),

TreeNode("What is the capital of Japan?", ["Beijing", "Seoul", "Tokyo", "Bangkok"], 2, "Geography", 1),

TreeNode("Which is the largest ocean?", ["Atlantic", "Indian", "Arctic", "Pacific"], 3, "Geography", 2),

TreeNode("What is the capital of Australia?", ["Sydney", "Melbourne", "Canberra", "Perth"], 2, "Geography", 2),

TreeNode("Which country is known as the Land of the Rising Sun?", ["China", "Japan", "Thailand", "Korea"], 1, "Geography", 3)

])

if subject == "Science" or subject == "Mixed":

fallback\_questions.extend([

TreeNode("Which planet is known as the Red Planet?", ["Earth", "Mars", "Jupiter", "Venus"], 1, "Science", 1),

TreeNode("Which element has the symbol 'O'?", ["Gold", "Silver", "Oxygen", "Osmium"], 2, "Science", 1),

TreeNode("What is the largest planet in our solar system?", ["Earth", "Mars", "Jupiter", "Saturn"], 2, "Science", 2),

TreeNode("What is the chemical symbol for water?", ["Wa", "H2O", "W", "HO"], 1, "Science", 2),

TreeNode("Which animal has the longest lifespan?", ["Elephant", "Giant Tortoise", "Whale", "Human"], 1, "Science", 3)

])

if subject == "History" or subject == "Mixed":

fallback\_questions.extend([

TreeNode("In which year did World War II end?", ["1943", "1945", "1947", "1950"], 1, "History", 1),

TreeNode("Who was the first President of the United States?", ["Thomas Jefferson", "George Washington", "John Adams", "Benjamin Franklin"], 1, "History", 1),

TreeNode("In which year did the Titanic sink?", ["1910", "1912", "1915", "1918"], 1, "History", 2),

TreeNode("Which empire was ruled by Caesar?", ["Greek", "British", "Roman", "Persian"], 2, "History", 2),

TreeNode("Which war is also known as the Great War?", ["Civil War", "World War I", "World War II", "Cold War"], 1, "History", 3)

])

if subject == "Literature" or subject == "Mixed":

fallback\_questions.extend([

TreeNode("Who wrote 'Romeo and Juliet'?", ["Charles Dickens", "William Shakespeare", "Jane Austen", "Mark Twain"], 1, "Literature", 1),

TreeNode("Who wrote 'Pride and Prejudice'?", ["Emily Brontë", "Jane Austen", "Virginia Woolf", "Charlotte Brontë"], 1, "Literature", 1),

TreeNode("Who created Sherlock Holmes?", ["Agatha Christie", "Arthur Conan Doyle", "Edgar Allan Poe", "H.G. Wells"], 1, "Literature", 2),

TreeNode("Which book features a character named Harry Potter?", ["The Hobbit", "Harry Potter and the Philosopher's Stone", "The Lion, the Witch and the Wardrobe", "Percy Jackson"], 1, "Literature", 2),

TreeNode("Who wrote 'To Kill a Mockingbird'?", ["J.D. Salinger", "Harper Lee", "F. Scott Fitzgerald", "Ernest Hemingway"], 1, "Literature", 3)

])

# For Mixed, we already added some from each category

# If not Mixed and we're here, we have all the questions for the specific subject

# Shuffle the fallback questions

random.shuffle(fallback\_questions)

# Limit to 15 questions maximum

return fallback\_questions[:15]

def start\_game(self):

# Disable subject dropdown

self.subject\_select.config(state="disabled")

# Remove start button

self.strt\_btn\_frm.pack\_forget()

# Hide any messages

self.loading\_labl.config(text="")

# Get selected subject

subject = self.selected\_subj.get()

self.debug\_log(f"Starting game with subject: {subject}")

# Create question tree

self.question\_tree\_root = self.make\_qst\_tree(subject)

# Start with the root node

self.crnt\_node = self.question\_tree\_root

self.path\_history = ["Root"]

# Animate boss entrance

self.boss\_entry\_anm()

def boss\_entry\_anm(self):

# Animate the boss flying in

current\_x = 800

current\_y = 100

def updt\_boss\_pos(x, y):

if x > 650:

self.canvas.coords(self.boss, x, y)

self.after(50, lambda: updt\_boss\_pos(x - 10, y))

else:

self.Intro\_msg()

self.canvas.coords(self.boss, current\_x, current\_y)

updt\_boss\_pos(current\_x, current\_y)

def Intro\_msg(self):

subject = self.selected\_subj.get()

messagebox.showinfo("Tree Defender", f"The {subject} Syntax Wave Boss is attacking the knowledge tree! Answer correctly to defend and defeat the boss!")

# Display the first question (root of the tree)

self.current\_qst()

def current\_qst(self):

#Display the current question from the tree traversal

if not self.crnt\_node:

self.debug\_log("No current node! Showing victory")

self.win\_anm()

return

# Update tree path visualization

self.tree\_path.config(text=f"Question Tree Path: {' -> '.join(self.path\_history)}")

# Update question with animation effect

self.question\_lbl.config(text="")

question\_text = f"[{self.crnt\_node.platform}] {self.crnt\_node.question}"

def animate\_txt(text, i=0):

if i <= len(text):

self.question\_lbl.config(text=text[:i])

self.after(30, lambda: animate\_txt(text, i + 1))

animate\_txt(question\_text)

# Enable and update answer buttons

for i, ans in enumerate(self.crnt\_node.answers):

self.btns[i].config(text=ans, state=tk.NORMAL)

# Reset player movement flag to allow movement

self.allowed\_move = True

def check\_answer(self, chosen\_index):

if self.attacking or not self.allowed\_move:

self.debug\_log("Player tried to move while locked")

return # Prevent input during attacks or when movement is locked

# Disable buttons during animation

for btn in self.btns:

btn.config(state=tk.DISABLED)

# Move player to selected platform

self.move\_player(chosen\_index)

if chosen\_index == self.crnt\_node.correct\_answer:

self.correct\_answer()

else:

self.wrong\_answer(chosen\_index)

def move\_player(self, platform\_index):

if not self.allowed\_move:

self.debug\_log("Movement blocked - player cannot move")

return

self.crnt\_platform\_indx = platform\_index

target\_x = self.platforms[platform\_index]["x"]

target\_y = self.platforms[platform\_index]["y"] - 25 # Position above platform

current\_x, current\_y = self.canvas.coords(self.player)

self.movement\_anm(current\_x, current\_y, target\_x, target\_y)

def movement\_anm(self, current\_x, current\_y, target\_x, target\_y, steps=20):

if steps <= 0:

return

# Calculate movement

dx = (target\_x - current\_x) / steps

dy = (target\_y - current\_y) / steps

# Update position

self.canvas.coords(self.player, current\_x + dx, current\_y + dy)

# Continue animation

self.after(20, lambda: self.movement\_anm(

current\_x + dx, current\_y + dy, target\_x, target\_y, steps - 1))

def correct\_answer(self):

# Visual feedback for correct answer

platform = self.platforms[self.crnt\_node.correct\_answer]["shape"]

original\_color = self.canvas.itemcget(platform, "fill")

def platform\_anm(count=6):

if count > 0:

color = "#2ECC71" if count % 2 == 0 else original\_color

self.canvas.itemconfig(platform, fill=color)

self.after(200, lambda: platform\_anm(count - 1))

else:

self.canvas.itemconfig(platform, fill=original\_color)

self.attack\_boss()

platform\_anm()

messagebox.showinfo("Correct!", "✅ Excellent! Time to counter-attack!")

def attack\_boss(self):

# Create player attack projectile

player\_x, player\_y = self.canvas.coords(self.player)

boss\_x, boss\_y = self.canvas.coords(self.boss)

self.debug\_log(f"Creating attack from player to boss")

# Create an energy bolt

bolt = self.canvas.create\_oval(

player\_x - 10, player\_y - 10,

player\_x + 10, player\_y + 10,

fill="#2ECC71", outline="#FFFFFF"

)

def update\_bolt(obj, x, y):

# Calculate distance to boss

distance = math.sqrt((boss\_x - x)\*\*2 + (boss\_y - y)\*\*2)

if distance > 20: # Stop when we get close to the boss

# Calculate trajectory

dx = (boss\_x - x) / 10

dy = (boss\_y - y) / 10

# Move the bolt

self.canvas.move(obj, dx, dy)

x += dx

y += dy

self.after(30, lambda: update\_bolt(obj, x, y))

else:

# Bolt hit the boss

self.debug\_log("Bolt hit boss")

self.canvas.delete(obj)

self.boss\_hit\_anm()

update\_bolt(bolt, player\_x, player\_y)

def boss\_hit\_anm(self):

# Visual effect for boss hit

boss\_x, boss\_y = self.canvas.coords(self.boss)

# Flag to ensure update\_boss\_health is called only once

self.hit\_processed = False

# Flash effect

explosion = self.canvas.create\_oval(

boss\_x - 30, boss\_y - 30,

boss\_x + 30, boss\_y + 30,

fill="#E74C3C", outline="#F1C40F", width=2

)

self.debug\_log("Starting explosion effect")

def fade\_explosion(obj, size=30, alpha=1.0):

if size < 60 and alpha > 0:

self.canvas.coords(

obj,

boss\_x - size, boss\_y - size,

boss\_x + size, boss\_y + size

)

size += 3

alpha -= 0.1

self.after(50, lambda: fade\_explosion(obj, size, alpha))

else:

self.canvas.delete(obj)

# Only call update\_boss\_health once

if not self.hit\_processed:

self.hit\_processed = True

self.debug\_log("Explosion complete, calling update\_boss\_health")

self.upd\_boss\_health()

fade\_explosion(explosion)

def upd\_boss\_health(self):

self.debug\_log(f"Boss health before: {self.boss\_hlth}")

self.boss\_hlth -= 1

self.score += 100 \* self.crnt\_node.difficulty # Higher score for harder questions

self.debug\_log(f"Boss health after: {self.boss\_hlth}")

# Update display

heart\_emoji = "❤️"

self.canvas.itemconfig(

self.boss\_health\_txt,

text=f"Boss Health: {self.boss\_hlth} {heart\_emoji}"

)

self.canvas.itemconfig(

self.score\_txt,

text=f"Score: {self.score} | Level: {self.level}"

)

# Update the current node and traverse to the next correct node

next\_node = self.crnt\_node.children["correct"]

if next\_node:

self.debug\_log(f"Moving to next node in 'correct' path")

self.crnt\_node = next\_node

# Update path history

difficulty\_emoji = "🟢" if next\_node.difficulty == 1 else ("🟡" if next\_node.difficulty == 2 else "🔴")

self.path\_history.append(f"{difficulty\_emoji} Q{len(self.path\_history)}")

else:

# No more nodes in the tree, player wins!

self.debug\_log("No more nodes in the 'correct' path - Victory!")

self.crnt\_node = None

# Check if boss is defeated

if self.boss\_hlth <= 0:

self.debug\_log("Boss defeated! Triggering victory animation")

self.win\_anm()

else:

self.debug\_log("Boss still alive, continuing game")

self.after(1000, self.continue\_game)

def win\_anm(self):

# Make boss disappear with explosion effect

boss\_x, boss\_y = self.canvas.coords(self.boss)

# Create multiple explosion particles

particles = []

for \_ in range(10):

angle = random.uniform(0, 3.14 \* 2)

speed = random.uniform(2, 8)

dx = speed \* math.cos(angle)

dy = speed \* math.sin(angle)

size = random.randint(5, 15)

color = random.choice(["#E74C3C", "#F39C12", "#F1C40F"])

particle = self.canvas.create\_oval(

boss\_x - size, boss\_y - size,

boss\_x + size, boss\_y + size,

fill=color

)

particles.append({

"obj": particle,

"dx": dx,

"dy": dy,

"life": 20

})

def upd\_particle():

still\_alive = False

for p in particles:

if p["life"] > 0:

self.canvas.move(p["obj"], p["dx"], p["dy"])

p["life"] -= 1

p["dx"] \*= 0.95

p["dy"] \*= 0.95

still\_alive = True

else:

self.canvas.delete(p["obj"])

if still\_alive:

self.after(50, upd\_particle)

else:

self.canvas.delete(self.boss)

self.victor\_msg()

upd\_particle()

def victor\_msg(self):

subject = self.selected\_subj.get()

messagebox.showinfo("Victory!", f"🎉 You defeated the {subject} Syntax Wave Boss!\nYou completed the knowledge tree with a score of {self.score}!")

# Show a diagram of the path taken through the tree

path\_str = " -> ".join(self.path\_history)

messagebox.showinfo("Tree Path", f"Your path through the knowledge tree:\n{path\_str}")

# Ask if player wants to continue to next level

if messagebox.askyesno("Next Level", "Do you want to proceed to the next level?"):

self.level += 1

self.boss\_hlth = 3 + self.level # Increase difficulty

self.canvas.itemconfig(

self.score\_txt,

text=f"Score: {self.score} | Level: {self.level}"

)

# Recreate boss

self.boss\_img = ImageTk.PhotoImage(Image.open("pics/boss.png").resize((120, 120)))

self.boss = self.canvas.create\_image(800, 100, image=self.boss\_img)

# Update boss health display

heart\_emoji = "❤️"

self.canvas.itemconfig(

self.boss\_health\_txt,

text=f"Boss Health: {self.boss\_hlth} {heart\_emoji}"

)

# Get selected subject and create new question tree

subject = self.selected\_subj.get()

self.question\_tree\_root = self.make\_qst\_tree(subject)

# Reset path history and start with the root node

self.crnt\_node = self.question\_tree\_root

self.path\_history = ["Root"]

# Start new round

self.boss\_entry\_anm()

else:

# Recreate boss

self.boss\_img = ImageTk.PhotoImage(Image.open("pics/boss.png").resize((120, 120)))

self.boss = self.canvas.create\_image(800, 100, image=self.boss\_img)

# Reset game for a new session

self.reset\_game()

def reset\_game(self):

#Reset the game

# Reset game state

self.boss\_hlth = 3

self.player\_hlth = 3

self.score = 0

self.level = 1

self.crnt\_node = None

self.path\_history = ["Root"]

self.attacking = False

self.allowed\_move = True

# Update displays

heart\_emoji = "❤️"

self.canvas.itemconfig(

self.boss\_health\_txt,

text=f"Boss Health: {self.boss\_hlth} {heart\_emoji}"

)

self.canvas.itemconfig(

self.player\_health\_txt,

text=f"Your Health: {self.player\_hlth} {heart\_emoji}"

)

self.canvas.itemconfig(

self.score\_txt,

text=f"Score: {self.score} | Level: {self.level}"

)

# Reset player and boss positions

self.canvas.coords(self.boss, 650, 100)

self.canvas.coords(self.player, 100, 350)

# Enable subject selection

self.subject\_select.config(state="normal")

# Clear question text and buttons

self.question\_lbl.config(text="")

for btn in self.btns:

btn.config(text="", state=tk.DISABLED)

# Clear tree path label

self.tree\_path.config(text="Question Tree Path: Root")

# Show the start button frame again - make sure it's visible!

self.strt\_btn\_frm.grid(row=5, column=0, pady=(10, 10), sticky="ew")

# Make sure the start button is visible

self.strt\_btn.pack()

# Update loading label

self.loading\_labl.config(text="Select a subject and click 'Start Game' to begin!")

def wrong\_answer(self, chosen\_index):

# Set the attack flag and prevent player movement during attack sequence

self.attacking = True

self.allowed\_move = False

self.debug\_log(f"Wrong answer selected: {chosen\_index}. Player locked in place.")

# Store the chosen platform index - this ensures the attack hits the right platform

self.target\_platform = chosen\_index

# Visual feedback for wrong platform

platform = self.platforms[chosen\_index]["shape"]

original\_color = self.canvas.itemcget(platform, "fill")

self.canvas.itemconfig(platform, fill="#E74C3C")

# Show correct answer

correct\_platform = self.platforms[self.crnt\_node.correct\_answer]["shape"]

self.canvas.itemconfig(correct\_platform, fill="#2ECC71")

# Show message before attack

messagebox.showinfo("Incorrect!", f"❌ Wrong answer! The correct answer was: {self.crnt\_node.answers[self.crnt\_node.correct\_answer]}")

messagebox.showinfo("Boss Attack", "The boss is attacking!")

# Boss attack animation

self.animate\_boss\_attack(chosen\_index)

# Reset platform colors after a delay

def reset\_colors():

self.canvas.itemconfig(platform, fill=original\_color)

self.canvas.itemconfig(correct\_platform, fill=original\_color)

self.after(2000, reset\_colors)

def animate\_boss\_attack(self, target\_platform):

# Get positions

boss\_x, boss\_y = self.canvas.coords(self.boss)

target\_x = self.platforms[target\_platform]["x"]

target\_y = self.platforms[target\_platform]["y"] - 25

# Create attack projectile

fireball = self.canvas.create\_text(

boss\_x, boss\_y, text="🔥", font=("Arial", 100),fill="red",

)

self.debug\_log(f"Boss attack launched at platform {target\_platform}")

# Animate projectile

def update\_fireball(obj, x, y, steps=0):

# Calculate direction vector

dx = (target\_x - x) / 20

dy = (target\_y - y) / 20

# Check if we've reached target (or close enough)

distance = math.sqrt((target\_x - x)\*\*2 + (target\_y - y)\*\*2)

if distance > 20 and steps < 100: # Add steps limit as failsafe

# Move the fireball

self.canvas.move(obj, dx, dy)

x += dx

y += dy

self.after(20, lambda: update\_fireball(obj, x, y, steps + 1))

else:

# Create explosion effect

self.debug\_log(f"Fireball reached target at ({x:.2f}, {y:.2f})")

# Clear the fireball

self.canvas.delete(obj)

explosion = self.canvas.create\_oval(

target\_x - 20, target\_y - 20,

target\_x + 20, target\_y + 20,

fill="#E74C3C", outline="#F1C40F", width=2

)

def fade\_explosion(exp\_obj, size=20):

if size < 40:

self.canvas.coords(

exp\_obj,

target\_x - size, target\_y - size,

target\_x + size, target\_y + size

)

size += 2

self.after(30, lambda: fade\_explosion(exp\_obj, size))

else:

self.canvas.delete(exp\_obj)

# Explicitly apply damage here

self.debug\_log("Explosion complete - applying damage to player")

self.apply\_damage\_to\_player()

fade\_explosion(explosion)

update\_fireball(fireball, boss\_x, boss\_y)

def apply\_damage\_to\_player(self):

"""Apply damage to player regardless of position, as they've committed to the wrong answer"""

self.debug\_log("Player taking damage from boss attack")

# Reduce player health

self.player\_hlth -= 1

# Update health display with animation

heart\_emoji = "❤️"

# Flash the health text red

def flash\_health\_text(count=6):

if count > 0:

color = "#FF0000" if count % 2 == 0 else "#2ECC71"

self.canvas.itemconfig(self.player\_health\_txt, fill=color)

self.after(100, lambda: flash\_health\_text(count - 1))

else:

self.canvas.itemconfig(

self.player\_health\_txt,

text=f"Your Health: {self.player\_hlth} {heart\_emoji}",

fill="#2ECC71"

)

self.check\_game\_over()

flash\_health\_text()

# Visual feedback for player hit

player\_x, player\_y = self.canvas.coords(self.player)

hit\_effect = self.canvas.create\_oval(

player\_x - 25, player\_y - 25,

player\_x + 25, player\_y + 25,

fill="#E74C3C", outline="white", width=2

)

def fade\_hit(obj, opacity=1.0):

if opacity > 0:

# Can't change opacity directly in tkinter, so simulate it

self.after(100, lambda: fade\_hit(obj, opacity - 0.2))

else:

self.canvas.delete(obj)

fade\_hit(hit\_effect)

def check\_game\_over(self):

if self.player\_hlth <= 0:

subject = self.selected\_subj.get()

messagebox.showerror("Game Over", f"💀 You were defeated by the {subject} Syntax Wave Boss!")

# Show path taken before defeat

path\_str = " -> ".join(self.path\_history)

messagebox.showinfo("Tree Path", f"Your path through the knowledge tree:\n{path\_str}")

# Ask if player wants to restart

if messagebox.askyesno("Play Again?", "Would you like to play again with the same subject?"):

# Reset game to start state with same subject

self.debug\_log("Resetting game to initial state with same subject")

# Reset all game variables

self.boss\_hlth = 3

self.player\_hlth = 3

self.score = 0

self.level = 1

self.crnt\_node = None

self.path\_history = ["Root"]

self.attacking = False

self.allowed\_move = True

# Update displays

heart\_emoji = "❤️"

self.canvas.itemconfig(

self.boss\_health\_txt,

text=f"Boss Health: {self.boss\_hlth} {heart\_emoji}"

)

self.canvas.itemconfig(

self.player\_health\_txt,

text=f"Your Health: {self.player\_hlth} {heart\_emoji}"

)

self.canvas.itemconfig(

self.score\_txt,

text=f"Score: {self.score} | Level: {self.level}"

)

# Reset player and boss positions

self.canvas.coords(self.boss, 650, 100)

self.canvas.coords(self.player, 100, 350)

# Disable subject selection since we're using the same subject

self.subject\_select.config(state="disabled")

# Clear question text and buttons

self.question\_lbl.config(text="")

for btn in self.btns:

btn.config(text="", state=tk.DISABLED)

# Clear tree path label

self.tree\_path.config(text="Question Tree Path: Root")

# Create a new question tree based on the selected subject

subject = self.selected\_subj.get()

self.question\_tree\_root = self.make\_qst\_tree(subject)

# Set current node to the root of the tree

self.crnt\_node = self.question\_tree\_root

# Simply call start\_game directly to restart

self.start\_game()

else:

# Reset the game completely

self.reset\_game()

else:

self.attacking = False

# Tree traversal happens here for incorrect answers

# Move to the incorrect branch of the tree

next\_node = self.crnt\_node.children["incorrect"]

if next\_node:

self.debug\_log(f"Moving to next node in 'incorrect' path")

self.crnt\_node = next\_node

# Update path history for incorrect path

difficulty\_emoji = "🟢" if next\_node.difficulty == 1 else ("🟡" if next\_node.difficulty == 2 else "🔴")

self.path\_history.append(f"{difficulty\_emoji} Q{len(self.path\_history)}\*") # \* marks incorrect path

else:

# No incorrect path defined, stay on current node and retry

self.debug\_log("No 'incorrect' path defined - retrying current question")

self.allowed\_move = True # Re-enable player movement

self.debug\_log("Attack sequence complete, player movement re-enabled")

self.after(1000, self.continue\_game)

def continue\_game(self):

for btn in self.btns:

btn.config(state=tk.DISABLED)

# Reset attack flag

self.attacking = False

self.allowed\_move = True # Ensure player can move again

# Tree traversal - display the current node (either next correct node or incorrect branch)

if self.crnt\_node:

self.current\_qst()

else:

# No more nodes in the tree, player wins!

self.win\_anm()

### Scores.py

import customtkinter as ctk

from tkinter import messagebox

from tkcalendar import Calendar

import mysql.connector # Import MySQL connector

DB\_CONFIG = {

'user': 'root',

'password': '',

'host': 'localhost',

'database': 'nea',

'buffered': 'True'

}

class ScoreManager(ctk.CTkFrame): # inheritance

def \_\_init\_\_(self, parent,user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

self.user\_id=user\_id

self.subj = self.get\_subj(user\_id)

self.ui()

def get\_subj(self, user\_id):

subjects = []

try:

# Connect to the MySQL database

connection = mysql.connector.connect(\*\*DB\_CONFIG)

cursor = connection.cursor()

# Execute the query to fetch subjects for user\_id = 1

cursor.execute("SELECT subject\_name FROM subjects WHERE user\_id = %s", (user\_id,))

rows = cursor.fetchall()

# Extract subject names from the query results

subjects = [row[0] for row in rows]

except mysql.connector.Error as e:

messagebox.showerror("Database Error", f"An error occurred while fetching subjects: {e}")

finally:

if connection.is\_connected():

cursor.close()

connection.close()

return subjects

def cstm\_date(self):

top = ctk.CTkToplevel(self)

top.title("Select Date")

# Get screen dimensions

screen\_width = top.winfo\_screenwidth()

screen\_height = top.winfo\_screenheight()

# Set the size of the calendar window

calendar\_width = 300

calendar\_height = 300

# Calculate x and y coordinates for center position

x = (screen\_width // 2) - (calendar\_width // 2)

y = (screen\_height // 2) - (calendar\_height // 2)

# Set the geometry of the calendar window

top.geometry(f"{calendar\_width}x{calendar\_height}+{x}+{y}")

top.attributes('-topmost', True) # Ensure the window stays on top

cal = Calendar(top, selectmode='day', year=2024, month=1, day=1)

cal.pack(pady=20)

def grab\_date():

selected\_date = cal.get\_date()

month, day, year = map(int, selected\_date.split("/"))

self.yr\_var.set(year) # Year will be full 4-digit format

self.month\_var.set(month)

self.date\_disp\_var.set(f"Selected Date: {day}/{month}/{year}")

top.destroy()

ctk.CTkButton(top, text="Select", command=grab\_date).pack()

def add\_score(self):

subject = self.subj\_var.get()

year\_str = self.yr\_var.get() # Get the year as a string

month = int(self.month\_var.get())

score = float(self.score\_var.get()) # Score can be a float as per your table definition

# Ensure the year is a full four-digit number

try:

year = int(year\_str)

if year < 100: # If the year is less than 100, assume it's in the 2000s

year += 2000

except ValueError:

messagebox.showerror("Input Error", "Year must be a valid integer.")

return

print(f"Adding score: Subject={subject}, Year={year}, Month={month}, Score={score}")

try:

# Connect to the DB

connection = mysql.connector.connect(\*\*DB\_CONFIG)

cursor = connection.cursor()

cursor.execute("SELECT subject\_id FROM subjects WHERE subject\_name = %s AND user\_id = %s", (subject,self.user\_id))

subject\_id\_row = cursor.fetchone()

if not subject\_id\_row:

messagebox.showerror("Database Error", "Subject not found in the database.")

return

subject\_id = subject\_id\_row[0]

#NOTE Close the first cursor and create a new one for the INSERT

cursor.close()

cursor = connection.cursor()

# Insert the score into the scores table

cursor.execute("INSERT INTO scores (subject\_id, month, year, score) VALUES (%s, %s, %s, %s)",

(subject\_id, month, year, score))

connection.commit()

messagebox.showinfo("Success", "Score added successfully!")

except mysql.connector.Error as e:

messagebox.showerror("Database Error", f"An error occurred while adding the score: {e}")

finally:

if connection.is\_connected():

cursor.close()

connection.close()

def ui(self):

frame = ctk.CTkFrame(self) # Ensure it's created in the context of ScoreManager

frame.pack(pady=10, padx=10, fill='both', expand=True)

self.subj\_var = ctk.StringVar()

self.yr\_var = ctk.StringVar()

self.month\_var = ctk.StringVar()

self.score\_var = ctk.StringVar()

self.date\_disp\_var = ctk.StringVar(value="No date selected")

ctk.CTkLabel(frame, text="Subject:").grid(row=0, column=0, pady=5)

ctk.CTkComboBox(frame, variable=self.subj\_var, values=self.subj).grid(row=0, column=1, pady=5)

ctk.CTkLabel(frame, text="Date:").grid(row=1, column=0, pady=5)

date\_btn = ctk.CTkButton(frame, text="Select Date", command=self.cstm\_date)

date\_btn.grid(row=1, column=1, pady=5)

ctk.CTkLabel(frame, textvariable=self.date\_disp\_var, text\_color="white").grid(row=2, column=1, pady=5)

ctk.CTkLabel(frame, text="Score:").grid(row=3, column=0, pady=5)

ctk.CTkEntry(frame, textvariable=self.score\_var).grid(row=3, column=1, pady=5)

ctk.CTkButton(frame, text="Add Score", command=self.add\_score).grid(row=4, columnspan=2, pady=10)

### Userpanel.py

import customtkinter as ctk

import tkinter as tk

from tkinter import ttk

from graph\_bar import ScoreVisualizer

from resources import ResourceLibrary

from v3.v3.fianl.analyse import StudyAdvisorApp

from quotes import MotivationQuote

from examcountdow import ExamCountdown

from availability import AvailabilityEditor

from scores import ScoreManager

class Userpanel(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

self.user\_id = user\_id

self.configure(fg\_color="#111827")

# Configure the grid with proper weights

self.columnconfigure(0, weight=1)

self.columnconfigure(1, weight=2)

self.columnconfigure(2, weight=1)

self.rowconfigure(0, weight=0)

self.rowconfigure(1, weight=1)

# Create the UI components

self.header()

self.main\_sections()

def header(self):

#Create a header bar

hdr = ctk.CTkFrame(self, fg\_color="#1E293B", corner\_radius=0, height=50) # hdr = header

hdr.grid(row=0, column=0, columnspan=3, sticky="ew", padx=0, pady=(0, 0))

# Title

title = ctk.CTkLabel(hdr, text="ECLIPSE DASHBOARD", font=("Arial", 16, "bold"),text\_color="#FFFFFF")

title.pack(side="left", padx=20, pady=8)

# User info

user\_frm = ctk.CTkFrame(hdr, fg\_color="transparent")

user\_frm.pack(side="right", padx=20, pady=8)

user\_dipl = ctk.CTkLabel( user\_frm, text=f"👤 User ID: {self.user\_id}", font=("Arial", 12),text\_color="#FFFFFF") # dipl = display

user\_dipl.pack(side="right")

def main\_sections(self):

padding = {"padx": 8, "pady": 8}

# I use this to see

# 1. (LEFT COLUMN)

left\_sect = self.section\_frame(0, "Resources & Deadlines", padding)

self.resources(left\_sect)

self.exam\_countdown(left\_sect)

# 2. (CENTER COLUMN)

center\_sect = self.section\_frame(1, "Progress & Performance", padding)

self.Run\_quotes(center\_sect)

self.Graph(center\_sect)

# 3. (RIGHT COLUMN)

right\_sect = self.section\_frame(2, "Schedule & Scores", padding)

self.availability(right\_sect)

self.score\_manager(right\_sect)

self.analysis(right\_sect)

def section\_frame(self, column, title, padding):

# main frame for section

section = ctk.CTkFrame(self, fg\_color="transparent", corner\_radius=0)

section.grid(row=1, column=column, sticky="nsew", \*\*padding)

section.columnconfigure(0, weight=1)

section.rowconfigure(0, weight=0)

section.rowconfigure(1, weight=1)

section.rowconfigure(2, weight=1)

section.rowconfigure(3, weight=0)

# Add the section title

title\_frame = ctk.CTkFrame(section, fg\_color="#1F2A3C", corner\_radius=8, height=36)

title\_frame.grid(row=0, column=0, sticky="ew", padx=0, pady=(0, 8))

# Title text

title\_label = ctk.CTkLabel(title\_frame, text=title, font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title\_label.pack(pady=8)

return section

def resources(self, parent\_frame):

resources\_frm = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

resources\_frm.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

resource\_ttl\_frm = ctk.CTkFrame(resources\_frm, fg\_color="transparent", height=30)

resource\_ttl\_frm.pack(fill="x", padx=15, pady=(10, 0))

resource\_ttl = ctk.CTkLabel(resource\_ttl\_frm, text="📚 Learning Resources", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

resource\_ttl.pack(anchor="w")

resource\_cont = ctk.CTkFrame(resources\_frm, fg\_color="#2D3748")

resource\_cont.pack(fill="both", expand=True, padx=15, pady=10)

self.resource\_lib = ResourceLibrary(resource\_cont)

self.resource\_lib.pack(fill="both", expand=True)

def exam\_countdown(self, parent\_frame):

sched\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

sched\_frame.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Exam schedule title

sched\_ttl\_frame = ctk.CTkFrame(sched\_frame, fg\_color="transparent", height=30)

sched\_ttl\_frame.pack(fill="x", padx=15, pady=(10, 0))

sched\_title = ctk.CTkLabel(sched\_ttl\_frame, text="📅 Exam Schedule", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

sched\_title.pack(anchor="w")

# Initialize Exam Countdown - DO NOT set fg\_color here

exam\_container = ctk.CTkFrame(sched\_frame, fg\_color="#2D3748")

exam\_container.pack(fill="both", expand=True, padx=15, pady=10)

self.countdown = ExamCountdown(exam\_container)

self.countdown.pack(fill="both", expand=True)

def Run\_quotes(self, parent\_frame):

quote = ctk.CTkFrame(parent\_frame, fg\_color="#8B5CF6", corner\_radius=8)

quote.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Initialize Motivation Quote directly in the purple frame

self.quotes = MotivationQuote(quote)

# Important: Don't change its fg\_color, which could cause rendering issues

self.quotes.pack(fill="both", expand=True, padx=10, pady=10)

def Graph(self, parent\_frame):

performa = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

performa.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Performance title

title\_frame = ctk.CTkFrame(performa, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="📊 Performance Metrics", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

self.progress = ctk.CTkFrame(performa, fg\_color="#2D3748", height=400)

self.progress.pack(fill="both", expand=True, padx=15, pady=10)

# Important: Don't override ScoreVisualizer's internal styling

self.graph = ScoreVisualizer(self.progress, self.user\_id)

self.graph.pack(fill="both", expand=True)

def availability(self, parent\_frame):

# Container for the first component (Availability)

avail\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

avail\_frame.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Availability title

title\_frame = ctk.CTkFrame(avail\_frame, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="⏰ Weekly Availability", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

# Initialize Availability Editor - Important: give it enough height

avail\_container = ctk.CTkFrame(avail\_frame, fg\_color="#2D3748", height=350)

avail\_container.pack(fill="both", expand=True, padx=15, pady=10)

# Create the availability editor without changing its fg\_color

availability = AvailabilityEditor(avail\_container, self.user\_id)

availability.pack(fill="both", expand=True)

def score\_manager(self, parent\_frame):

# Container for the second component (Score Manager)

score\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

score\_frame.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Score manager title

title\_frame = ctk.CTkFrame(score\_frame, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="🎯 Score Manager", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

container = ctk.CTkFrame(score\_frame, fg\_color="#2D3748", height=300)

container.pack(fill="both", expand=True, padx=15, pady=10)

# Create score manager without changing its fg\_color

score\_manager = ScoreManager(container, self.user\_id)

score\_manager.pack(fill="both", expand=True)

def analysis(self, parent\_frame):

btn\_frm = ctk.CTkFrame(parent\_frame, fg\_color="transparent")

btn\_frm.grid(row=3, column=0, sticky="nsew", padx=0, pady=(0, 0))

self.analysis\_brn = ctk.CTkButton(btn\_frm, text='📈 Analyze Performance', command=self.run\_analysis,fg\_color="#EF4444",hover\_color="#E73535",font=("Arial", 14, "bold"),corner\_radius=6,height=45)

self.analysis\_brn.pack(fill="x", pady=0)

def run\_analysis(self):

for widget in self.grid\_slaves():

if int(widget.grid\_info()["column"]) == 1 and int(widget.grid\_info()["row"]) == 1:

widget.destroy()

break

# create analysis ui

analysis\_sect = ctk.CTkFrame(self, fg\_color="transparent", corner\_radius=0)

analysis\_sect.grid(row=1, column=1, sticky="nsew", padx=8, pady=8)

title\_frm = ctk.CTkFrame(analysis\_sect, fg\_color="#1F2A3C", corner\_radius=8, height=36)

title\_frm.pack(fill="x", pady=(0, 8))

title = ctk.CTkLabel(title\_frm, text="Advanced Analytics", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(pady=8)

content\_frame = ctk.CTkFrame(analysis\_sect, fg\_color="#1F2A3C", corner\_radius=8, height=600)

content\_frame.pack(fill="both", expand=True, pady=(0, 8))

self.analysis = StudyAdvisorApp(content\_frame,self.user\_id)

self.analysis.pack(fill="both", expand=True, padx=15, pady=15)

# Return button

btn\_frame = ctk.CTkFrame(analysis\_sect, fg\_color="transparent")

btn\_frame.pack(fill="x")

return\_btn = ctk.CTkButton(btn\_frame,text="← Return to Dashboard",command=self.exit,fg\_color="#1F2A3C",hover\_color="#324056",font=("Arial", 14),corner\_radius=6,height=40)

return\_btn.pack(fill="x")

def exit(self):

#Return to the main dashboard view

for widget in self.grid\_slaves():

if int(widget.grid\_info()["column"]) == 1 and int(widget.grid\_info()["row"]) == 1:

widget.destroy()

break

# Recreate the center section

center\_sect = self.section\_frame(1, "Progress & Performance", {"padx": 8, "pady": 8})

self.Run\_quotes(center\_sect)

self.Graph(center\_sect)

### Flashcard\_v.py

import customtkinter as ctk

from tkinter import messagebox

from textwrap import fill

import mysql.connector

class FlashCardMainApp(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

# Extract user\_id properly

if isinstance(user\_id, tuple):

self.user\_id = user\_id[0] # Extract first element from tuple

else:

self.user\_id = user\_id # Simple attributes

print(self.user\_id)

# Set dark theme appearance

self.configure(fg\_color="#121212") # Dark background

self.main\_menu()

def connect\_db(self):

# try to connect to DB

try:

self.conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea",

buffered=True

)

self.cursor = self.conn.cursor(buffered=True)

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to connect to DB: {e}")

def main\_menu(self):

# connect to DB

self.connect\_db()

# Header

header = ctk.CTkFrame(self, fg\_color="#121212", corner\_radius=0)

header.pack(fill="x", pady=(20, 10))

# Title

title\_labl = ctk.CTkLabel(header, text="Flashcards", font=("Arial", 28, "bold"),text\_color="#FFFFFF")

title\_labl.pack(side="left", padx=20)

# Subtitle

subtitle\_labl = ctk.CTkLabel(header, text="Study and memorize key concepts with interactive flashcards", font=("Arial", 14),text\_color="#AAAAAA")

subtitle\_labl.pack(side="left", padx=10)

# Statistics Frame

stats\_frm = ctk.CTkFrame(self, fg\_color="#1E1E1E", corner\_radius=10)

stats\_frm.pack(side="top", anchor="e", padx=20, pady=10)

# get the stats from DB

try:

self.cursor.execute("""

SELECT

COUNT(front)

FROM flashcards

WHERE user\_id = %s

""", (self.user\_id,))

stats\_data = self.cursor.fetchone()

# Default valus if nothing exists

total = stats\_data[0] if stats\_data else 0

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to load stats: {e}")

total, reviewed, pending = 0, 0, 0

# Create stat cards inside the frame (3 columns)

card\_frm = ctk.CTkFrame(stats\_frm, fg\_color="#1E1E1E")

card\_frm.pack(padx=20, pady=20)

# Stats cards

stats = [

{"value": str(total), "label": "REVISION", "color": "#8A2BE2"}, # Purple

{"value": "0", "label": "WORK DONE", "color": "#9370DB"}, # Medium purple

{"value": "7", "label": "FAILS", "color": "#9932CC"} # Dark orchid

]

for i, stat in enumerate(stats):

stat\_card = ctk.CTkFrame(card\_frm, fg\_color="#1E1E1E")

stat\_card.grid(row=0, column=i, padx=20)

# Value

value\_labl = ctk.CTkLabel(stat\_card, text=stat["value"], font=("Arial", 24, "bold"),text\_color=stat["color"])

value\_labl.pack(anchor="center", pady=(0, 5))

# Label

label = ctk.CTkLabel(stat\_card, text=stat["label"], font=("Arial", 12),text\_color="#AAAAAA")

label.pack(anchor="center")

# Subject selection section

subj\_section = ctk.CTkFrame(self, fg\_color="#121212", corner\_radius=0)

subj\_section.pack(fill="x", pady=20)

# Subject heading

subj\_heading = ctk.CTkLabel(subj\_section, text="Select a Subject", font=("Arial", 20, "bold"),text\_color="#FFFFFF")

subj\_heading.pack(anchor="w", padx=20, pady=(0, 10))

# Search bar

search\_frm = ctk.CTkFrame(subj\_section, fg\_color="#1E1E1E", corner\_radius=20, height=40)

search\_frm.pack(fill="x", padx=20, pady=10)

search\_icon = ctk.CTkLabel(search\_frm, text="🔍", font=("Arial", 14), text\_color="#AAAAAA")

search\_icon.pack(side="left", padx=(15, 5))

search\_entry = ctk.CTkEntry(search\_frm, placeholder\_text="Search subjects...",border\_width=0,fg\_color="#1E1E1E",text\_color="#FFFFFF",font=("Arial", 14),height=38,width=400)

search\_entry.pack(side="left", fill="x", expand=True, padx=(0, 20))

# Create the scroll frame for subjects

self.button\_frm = ctk.CTkScrollableFrame(self,width=900,height=400,fg\_color="#121212",scrollbar\_fg\_color="#121212",scrollbar\_button\_color="#8A2BE2",scrollbar\_button\_hover\_color="#9370DB")

self.button\_frm.pack(fill="both", expand=True, padx=20, pady=10)

# Fill with subjects

self.flashcard\_selection()

def flashcard\_selection(self):

def get\_select\_subj\_db():

#Fetch subjects from dp

try:

self.cursor.execute("SELECT DISTINCT subject\_name FROM subjects WHERE user\_id=%s", (self.user\_id,))

results = self.cursor.fetchall()

return [row[0] for row in results] # Extract the subject names

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Database query failed: {e}")

return []

def open\_subj\_flashcard(subject):

#Open the subject

self.flash\_card(subject)

def create\_flashcards():

#unique subject button

unique\_subjects = get\_select\_subj\_db()

if not unique\_subjects:

# Create an empty state display

empty\_frm = ctk.CTkFrame(self.button\_frm, fg\_color="#121212")

empty\_frm.pack(fill="both", expand=True, pady=50)

empty\_labl = ctk.CTkLabel(empty\_frm, text="No subjects available yet", font=("Arial", 18),text\_color="#AAAAAA")

empty\_labl.pack(pady=10)

add\_btn = ctk.CTkButton(empty\_frm,text="+ Add Your First Subject",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=10,height=40)

add\_btn.pack(pady=10)

return

# Clear existing buttons before adding new ones

for widget in self.button\_frm.winfo\_children():

widget.destroy()

# Add our subjects as cards

subj\_card\_frm = ctk.CTkFrame(self.button\_frm, fg\_color="#121212")

subj\_card\_frm.pack(fill="both", expand=True)

# Configure grid with 3 columns

subj\_card\_frm.grid\_columnconfigure(0, weight=1)

subj\_card\_frm.grid\_columnconfigure(1, weight=1)

subj\_card\_frm.grid\_columnconfigure(2, weight=1)

# Add subject cards in a grid

for i, subject in enumerate(unique\_subjects):

row, col = divmod(i, 3) # 3 columns

subject\_card = ctk.CTkButton(subj\_card\_frm,text=subject,font=("Arial", 16, "bold"),fg\_color="#1E1E1E",hover\_color="#2A2A2A",corner\_radius=10,height=150,width=280,command=lambda s=subject: open\_subj\_flashcard(s))

subject\_card.grid(row=row, column=col, padx=10, pady=10, sticky="nsew")

# Create the flashcards

create\_flashcards()

def flash\_card(self, subject):

# Clear the main frame before showing flashcards

for widget in self.winfo\_children():

widget.destroy()

class Flashcard:

def \_\_init\_\_(self, word, meaning):

self.word = word

self.meaning = meaning

class FlashcardApp:

def \_\_init\_\_(self, main\_frame, cursor, connection, subject,user\_id, parent\_app):

self.main\_frame = main\_frame

self.parent\_app = parent\_app

self.cursor = cursor

self.connection = connection

self.subject = subject

self.user\_id=user\_id

self.flashcards = []

self.current\_flashcard = None

self.showing\_word = True

self.current\_index = 0

# Action\_frame

Action\_frm = ctk.CTkFrame(main\_frame, fg\_color="#121212", corner\_radius=0)

Action\_frm.pack(fill="x", pady=(20, 10))

# Back button

back\_btn = ctk.CTkButton(Action\_frm,text="← Back",font=("Arial", 14),fg\_color="#1E1E1E",hover\_color="#2A2A2A",corner\_radius=8,command=self.return\_back)

back\_btn.pack(side="left", padx=20)

subject\_title = ctk.CTkLabel(Action\_frm, text=f"Flashcards: {subject}", font=("Arial", 22, "bold"),text\_color="#FFFFFF")

subject\_title.pack(side="left", padx=10)

self.content\_frm = ctk.CTkFrame(main\_frame, fg\_color="#121212", corner\_radius=0)

self.content\_frm.pack(fill="both", expand=True, padx=20, pady=10)

# Tab selector frame

tab\_selector\_frm = ctk.CTkFrame(self.content\_frm, fg\_color="#1E1E1E", corner\_radius=10, height=50)

tab\_selector\_frm.pack(fill="x", pady=(0, 10))

# swicthing from add to study

def switch\_tab(tab\_name):

# hasattr = check if attribute exists

if hasattr(self, 'create\_frame'):

self.create\_frme.pack\_forget()

if hasattr(self, 'study\_tab\_frame'):

self.study\_tab\_frm.pack\_forget()

self.create\_tab\_btn.configure(fg\_color="#8A2BE2" if tab\_name == "Create" else "#1E1E1E",text\_color="white")

self.study\_tab\_btn.configure(fg\_color="#8A2BE2" if tab\_name == "Study" else "#1E1E1E",text\_color="white")

if tab\_name == "Create":

if not hasattr(self, 'create\_tab\_frame'):

self.create\_new\_flashcard()

self.create\_frme.pack(fill="both", expand=True)

else:

if not hasattr(self, 'study\_tab\_frame'):

self.setup\_study\_tab()

self.study\_tab\_frm.pack(fill="both", expand=True)

# Create tab buttons

self.create\_tab\_btn = ctk.CTkButton(tab\_selector\_frm,text="Create",font=("Arial", 14),fg\_color="#8A2BE2", hover\_color="#9370DB",corner\_radius=8,height=40,width=120,command=lambda: switch\_tab("Create"))

self.create\_tab\_btn.pack(side="left", padx=10, pady=5)

self.study\_tab\_btn = ctk.CTkButton(tab\_selector\_frm,text="Study",font=("Arial", 14),fg\_color="#1E1E1E", hover\_color="#2A2A2A",corner\_radius=8,height=40,width=120,command=lambda: switch\_tab("Study"))

self.study\_tab\_btn.pack(side="left", padx=10, pady=5)

self.create\_new\_flashcard()

self.load\_flashcards\_fromDB()

self.amount\_flashcard()

def create\_new\_flashcard(self):

# Create frame

self.create\_frme = ctk.CTkFrame(self.content\_frm, fg\_color="#121212", corner\_radius=0)

self.create\_frme.pack(fill="both", expand=True)

frame = ctk.CTkFrame(self.create\_frme, fg\_color="#1E1E1E", corner\_radius=10)

frame.pack(padx=20, pady=20, fill="x")

# Title

title = ctk.CTkLabel(frame, text="Create New Flashcard", font=("Arial", 18, "bold"),text\_color="#FFFFFF")

title.pack(pady=(20, 30))

# Word input

front\_labl = ctk.CTkLabel(frame, text="Term", font=("Arial", 14),text\_color="#FFFFFF")

front\_labl.pack(anchor="w", padx=30)

self.front\_entry = ctk.CTkEntry(frame,placeholder\_text="Enter term or concept",font=("Arial", 14),fg\_color="#2A2A2A",text\_color="#FFFFFF",height=40,width=500)

self.front\_entry.pack(padx=30, pady=(5, 20), fill="x")

self.back\_label = ctk.CTkLabel(frame, text="Definition", font=("Arial", 14),text\_color="#FFFFFF")

self.back\_entry = ctk.CTkEntry(frame,placeholder\_text="Enter definition or explanation",font=("Arial", 14),fg\_color="#2A2A2A",text\_color="#FFFFFF",height=40,width=500)

# button

self.add\_meaning = ctk.CTkButton(frame,text="Continue",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=8,height=40,command=self.flip\_meaning)

self.add\_meaning.pack(pady=20)

self.add\_btn = ctk.CTkButton(frame,text="Add Flashcard",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=8,height=40,command=self.add\_cards)

# Flashcard preview section

view\_frame = ctk.CTkFrame(self.create\_frme, fg\_color="#121212", corner\_radius=0)

view\_frame.pack(fill="both", expand=True, padx=20, pady=20)

view\_lbl = ctk.CTkLabel(view\_frame, text="Your Flashcards", font=("Arial", 18, "bold"),text\_color="#FFFFFF")

view\_lbl.pack(anchor="w", pady=(0, 10))

self.flashcard\_preview = ctk.CTkScrollableFrame(view\_frame,fg\_color="#121212",scrollbar\_fg\_color="#121212",scrollbar\_button\_color="#8A2BE2",scrollbar\_button\_hover\_color="#9370DB",orientation="horizontal")

self.flashcard\_preview.pack(fill="x", pady=5)

self.amount\_flashcard()

def setup\_study\_tab(self):

self.study\_tab\_frm = ctk.CTkFrame(self.content\_frm, fg\_color="#121212", corner\_radius=0)

# Main frame

self.flashcard\_frm = ctk.CTkFrame(self.study\_tab\_frm, fg\_color="#1E1E1E", corner\_radius=10)

self.flashcard\_frm.pack(fill="both", expand=True, padx=20, pady=20)

# Flashcard content

self.flashcard\_content = ctk.CTkLabel(self.flashcard\_frm, text="Click 'Start Studying' to begin",font=("Arial", 24, "bold"),text\_color="#FFFFFF",wraplength=700)

self.flashcard\_content.place(relx=0.5, rely=0.5, anchor="center")

# Control buttons

button\_frm = ctk.CTkFrame(self.study\_tab\_frm, fg\_color="#121212", corner\_radius=0)

button\_frm.pack(fill="x", padx=20, pady=(0, 20))

self.start\_btn = ctk.CTkButton(button\_frm,text="Start Studying",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=8,height=40,width=150,command=self.start\_cards)

self.start\_btn.pack(side="left", padx=(0, 10))

self.flip\_btn = ctk.CTkButton(button\_frm,text="Flip Card",font=("Arial", 14),fg\_color="#1E1E1E",hover\_color="#2A2A2A",corner\_radius=8,height=40,width=120,command=self.flip\_card)

self.next\_btn = ctk.CTkButton(button\_frm,text="Next Card",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=8,height=40,width=120,command=self.next\_card)

self.delete\_btn = ctk.CTkButton(button\_frm,text="Delete Card",font=("Arial", 14),fg\_color="#E74C3C", corner\_radius=8,height=40,width=120,command=self.delete\_card)

# load the flashcard

def amount\_flashcard(self):

def load\_flashcards():

selected\_sub = self.subject

try:

query = "SELECT DISTINCT flashcards.front FROM subjects JOIN flashcards On subjects.subject\_id=flashcards.subject\_id WHERE subjects.subject\_name=%s AND flashcards.user\_id=%s"

self.cursor.execute(query, (selected\_sub,self.user\_id))

results = self.cursor.fetchall()

return [row[0] for row in results] # Extract the subject names

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Database query failed: {e}")

return []

#Creates the flashcard

flashfront = load\_flashcards()

for widget in self.flashcard\_preview.winfo\_children():

widget.destroy()

if not flashfront:

empty\_label = ctk.CTkLabel(self.flashcard\_preview, text="No flashcards available. Create your first flashcard above.",font=("Arial", 14),text\_color="#AAAAAA")

empty\_label.pack(pady=20)

return

for subject in flashfront:

card\_preview = ctk.CTkButton(self.flashcard\_preview, text=subject,font=("Arial", 14),fg\_color="#2A2A2A",hover\_color="#3A3A3A",corner\_radius=8,height=150,width=200)

card\_preview.pack(side="left", padx=10, pady=5)

def return\_back(self):

# Clear content and go back

for widget in self.main\_frame.winfo\_children():

widget.destroy()

# Recreate the main menu

self.parent\_app.main\_menu()

def flip\_meaning(self):

word = self.front\_entry.get().strip()

if word:

# Hide word elements

#Dynamic object genrations

self.add\_meaning.pack\_forget()

# Show meaning elements

self.back\_label.pack(anchor="w", padx=30)

self.back\_entry.pack(padx=30, pady=(5, 20), fill="x")

self.add\_btn.pack(pady=20)

else:

messagebox.showerror("Error", "Please enter a term")

def add\_cards(self):

word = self.front\_entry.get().strip()

meaning = self.back\_entry.get().strip()

if word and meaning:

try:

cursor= self.connection.cursor(buffered=True)

# Get subject\_id

subject\_query = "SELECT subject\_id FROM subjects WHERE subject\_name = %s"

self.cursor.execute(subject\_query, (self.subject,))

result = self.cursor.fetchone()

if not result:

messagebox.showerror("Error", "Subject not found In DB")

return

subject\_id = result[0] # Get the subject id

# clear any pending result

while self.cursor.nextset():

pass

# MAKE fresh cursor for the insertion

insert\_cursor = self.connection.cursor()

# ADD into database

query = "INSERT INTO flashcards (front, back, subject\_id, user\_id) VALUES (%s, %s, %s, %s)"

insert\_cursor.execute(query, (word, meaning, subject\_id, self.user\_id))

self.connection.commit()

insert\_cursor.close() # Close the dedicated cursor

# Update local list

self.flashcards.append(Flashcard(word, meaning))

# Clear input fields

self.front\_entry.delete(0, 'end')

self.back\_entry.delete(0, 'end')

# Reset form

self.back\_label.pack\_forget()

self.back\_entry.pack\_forget()

self.add\_btn.pack\_forget()

self.add\_meaning.pack(pady=20)

# Show success message

messagebox.showinfo("Success", "Flashcard added successfully!")

# Refresh flashcards

self.load\_flashcards\_fromDB()

self.amount\_flashcard()

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to add flashcard: {e}")

# Print the full error to console for debugging

import traceback

traceback.print\_exc()

else:

messagebox.showwarning("Input Error", "Please provide both a term and a definition")

def load\_flashcards\_fromDB(self):

try:

cursor = self.connection.cursor(buffered=True)

query = "SELECT flashcards.front AS word, flashcards.back AS meaning FROM subjects JOIN flashcards ON subjects.subject\_id = flashcards.subject\_id WHERE subjects.subject\_name = %s AND flashcards.user\_id=%s" # simple Datbase Model

self.cursor.execute(query, (self.subject,self.user\_id,))

results = self.cursor.fetchall()

self.flashcards = [Flashcard(row[0], row[1]) for row in results]

if not self.flashcards:

self.flashcard\_content.configure(

text=f"No flashcards for {self.subject}.\nCreate some on the 'Create' tab.",

font=("Arial", 18)

)

else:

# Update the count in start button

self.start\_btn.configure(text=f"Start Studying ({len(self.flashcards)})")

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Database query failed: {e}")

def start\_cards(self):

if not self.flashcards:

messagebox.showinfo("No Flashcards", "Please create some flashcards first.")

# Switch to create tab

self.create\_tab\_btn.invoke()

return

# Hide start button

self.start\_btn.pack\_forget()

# Show control buttons

self.flip\_btn.pack(side="left", padx=(0, 10))

self.next\_btn.pack(side="left", padx=(0, 10))

self.delete\_btn.pack(side="left")

# Show first flashcard

self.next\_card()

def delete\_card(self):

if self.current\_flashcard:

# Get the word of the current flashcard

word\_to\_delete = self.current\_flashcard.word

if messagebox.askyesno("Confirm Delete", f"Are you sure you want to delete '{word\_to\_delete}'?"):

try:

# Get subject\_id

subject\_query = "SELECT subject\_id FROM subjects WHERE subject\_name = %s"

self.cursor.execute(subject\_query, (self.subject,))

subject\_result = self.cursor.fetchone()

if not subject\_result:

messagebox.showerror("Error", "Subject not found.")

return

subject\_id = subject\_result[0] # Get the subject\_id

# Delete the flashcard from the database

delete\_query = "DELETE FROM flashcards WHERE front = %s AND subject\_id = %s"

self.cursor.execute(delete\_query, (word\_to\_delete, subject\_id))

self.connection.commit()

# Remove the flashcard from the local list

self.flashcards = [fc for fc in self.flashcards if fc.word != word\_to\_delete]

messagebox.showinfo("Success", f"Flashcard '{word\_to\_delete}' deleted successfully.")

# Update flashcards display

if self.flashcards:

self.next\_card()

else:

self.flashcard\_content.configure(

text="No flashcards available.\nCreate some on the 'Create' tab.",

font=("Arial", 18)

)

# Hide control buttons and show start button

self.flip\_btn.pack\_forget()

self.next\_btn.pack\_forget()

self.delete\_btn.pack\_forget()

self.start\_btn.pack(side="left", padx=(0, 10))

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Failed to delete flashcard: {e}")

else:

messagebox.showwarning("Warning", "No flashcard is currently displayed.")

# Update flashcard preview

self.amount\_flashcard()

def next\_card(self):

if self.flashcards:

self.current\_index = (self.current\_index + 1) % len(self.flashcards)

self.current\_flashcard = self.flashcards[self.current\_index]

self.showing\_word = True

# Update content with card counter

self.flashcard\_content.configure(

text=self.current\_flashcard.word,

font=("Arial", 28, "bold")

)

# Show card counter at bottom of frame

self.flashcard\_content\_subtitle = ctk.CTkLabel(self.flashcard\_frm, text=f"Card {self.current\_index + 1} of {len(self.flashcards)}",font=("Arial", 14),text\_color="#AAAAAA")

self.flashcard\_content\_subtitle.place(relx=0.5, rely=0.9, anchor="center")

else:

self.flashcard\_content.configure(

text="No flashcards available!",

font=("Arial", 18)

)

messagebox.showwarning("Warning", "No flashcards available.")

self.return\_back()

def flip\_card(self):

if self.current\_flashcard:

if self.showing\_word:

self.flashcard\_content.configure(

text=self.current\_flashcard.meaning,

font=("Arial", 20)

)

self.showing\_word = False

else:

self.flashcard\_content.configure(

text=self.current\_flashcard.word,

font=("Arial", 28, "bold")

)

self.showing\_word = True

# Create the flashcard app

FlashcardApp(self, self.cursor, self.conn, subject, self.user\_id,self)

### Calculator\_normal.py

import mysql.connector

import json

import os

from datetime import datetime, timedelta

import random

import tkinter as tk

import customtkinter as ctk

from tkinter import messagebox

def save\_plan(user\_id, plan):

conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = conn.cursor()

cursor.execute("DELETE FROM events WHERE user\_id = %s", (user\_id,)) # Clear old plans for the user

for date, time, title in plan:

cursor.execute("INSERT INTO events (user\_id, date, time, title) VALUES (%s, %s, %s, %s)", (user\_id, date, time, title))

conn.commit()

conn.close()

def user\_availability(user\_id):

#Load availability data

default\_availability = {day: [] for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]}

try:

conn = mysql.connector.connect(host="localhost", user="root", password="", database="nea")

cursor = conn.cursor(dictionary=True)

cursor.execute("SELECT day, hours FROM availability WHERE user\_id = %s", (user\_id,))

results = cursor.fetchall()

conn.close()

if results:

# Convert database to format

availability = {day: [] for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]}

for row in results:

day = row['day']

hours\_str = row['hours']

if hours\_str and hours\_str.strip():

try:

# Try to parse as comma-separated values first

hours = [int(h.strip()) for h in hours\_str.split(',') if h.strip().isdigit()]

except:

# If parsing fails, use empty list

hours = []

else:

hours = []

availability[day] = hours

except Exception as e:

print(f"Error loading availability: {e}")

return default\_availability

def subj\_weight(weight):

if isinstance(weight, list):

latest\_entry = max(weight, key=lambda x: (x["year"], x["month"]))

return 100 - latest\_entry["score"] # Prioritize LOW scores

return 100 - int(weight)

def subj\_scores(user\_id):

#Get subject scores from database

try:

# get sore from db

conn = mysql.connector.connect(host="localhost", user="root", password="", database="nea")

cursor = conn.cursor(dictionary=True)

# Get subjects with their latest scores

cursor.execute("""

SELECT s.subject\_name, sc.score, sc.month, sc.year

FROM subjects s

LEFT JOIN (

SELECT subject\_id, score, month, year,

ROW\_NUMBER() OVER (PARTITION BY subject\_id ORDER BY year DESC, month DESC) as rn

FROM scores

) sc ON s.subject\_id = sc.subject\_id AND sc.rn = 1

WHERE s.user\_id = %s

""", (user\_id,))

results = cursor.fetchall()

conn.close()

if results:

return {row['subject\_name']: subj\_weight(row['score']) for row in results if row['score'] is not None}

except Exception as e:

print(f"Error getting subject scores: {e}")

# Return empty dict

return {}

# Add these functions for checking if setup is needed

def check\_setup(user\_id):

# check if user completed setup

daily\_hours = user\_availability(user\_id)

conn = mysql.connector.connect(host="localhost", user="root", password="", database="nea")

cursor = conn.cursor(dictionary=True)

# Get subjects with their latest scores

cursor.execute("""

SELECT subject\_name FROM subjects WHERE user\_id = %s

""", (user\_id,))

subjects = cursor.fetchall()

conn.close()

has\_hours = any(len(hours) >= 0 for hours in daily\_hours.values())

has\_subjects = len(subjects) > 0

return not (has\_hours and has\_subjects)

def complete\_setup(user\_id, callback=None):

if check\_setup(user\_id):

setup\_ui(user\_id, callback)

else:

tm\_allocation(user\_id)

if callback:

callback()

def setup\_ui(user\_id, callback=None):

#Create a setup window

global callback\_function

callback\_function = callback

setup\_window = ctk.CTk()

setup\_window.title("Initial Setup")

setup\_window.geometry("600x500")

ctk.set\_appearance\_mode("dark")

ctk.set\_default\_color\_theme("dark-blue")

main = ctk.CTkFrame(setup\_window)

main.pack(fill="both", expand=True, padx=20, pady=20)

# Welcome message

welcome\_msg = ctk.CTkLabel(main, text="Welcome to Eclipse Learning!", font=("Arial", 18, "bold"))

welcome\_msg.pack(pady=(10, 20))

info\_msg = ctk.CTkLabel(main, text="Please complete your initial setup to get started.", font=("Arial", 14))

info\_msg.pack(pady=(0, 20))

tabview = ctk.CTkTabview(main, width=550, height=350)

tabview.pack(fill="both", expand=True, padx=20, pady=20)

# Create tabs

tab\_subj = tabview.add("Subjects")

tab\_avail = tabview.add("Availability")

# SUBJECTS TAB

subjects\_frame = ctk.CTkFrame(tab\_subj)

subjects\_frame.pack(fill="both", expand=True, padx=20, pady=20)

ctk.CTkLabel(subjects\_frame, text="Add your subjects:", font=("Arial", 14, "bold")).pack(pady=(0, 10))

# Subject entry

add\_subj\_frame = ctk.CTkFrame(subjects\_frame)

add\_subj\_frame.pack(fill="x", pady=10)

subject\_entry = ctk.CTkEntry(add\_subj\_frame, placeholder\_text="Enter subject name", width=300)

subject\_entry.pack(side="left", padx=10)

subj\_list = []

def add\_subject():

subject = subject\_entry.get().strip()

if subject and subject not in subj\_list:

subj\_list.append(subject)

subject\_entry.delete(0, 'end')

update\_subjects\_display()

add\_button = ctk.CTkButton(add\_subj\_frame, text="Add", command=add\_subject)

add\_button.pack(side="right", padx=10)

# Subjects display area

subj\_display = ctk.CTkFrame(subjects\_frame)

subj\_display.pack(fill="both", expand=True, pady=10)

def update\_subjects\_display():

# Clear current display

for widget in subj\_display.winfo\_children():

widget.destroy()

# Show subjects

for i, subject in enumerate(subj\_list):

subj\_frame = ctk.CTkFrame(subj\_display)

subj\_frame.pack(fill="x", pady=2)

ctk.CTkLabel(subj\_frame, text=subject, width=300).pack(side="left", padx=10)

# Add remove button

remove\_btn = ctk.CTkButton(subj\_frame, text="Remove", width=80,

command=lambda s=subject: remove\_subject(s))

remove\_btn.pack(side="right", padx=10)

def remove\_subject(subject):

subj\_list.remove(subject)

update\_subjects\_display()

# availability frame

avail\_frame = ctk.CTkFrame(tab\_avail)

avail\_frame.pack(fill="both", expand=True, padx=20, pady=20)

ctk.CTkLabel(avail\_frame, text="Set your weekly availability:",

font=("Arial", 14, "bold")).pack(pady=(0, 10))

ctk.CTkLabel(avail\_frame, text="Enter time ranges for each day (e.g., 9-12, 14-16)",

font=("Arial", 12)).pack(pady=(0, 5))

ctk.CTkLabel(avail\_frame, text="Or enter '0' to indicate no availability for that day",

font=("Arial", 12)).pack(pady=(0, 10))

days = ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]

day\_entries = {}

for day in days:

day\_frame = ctk.CTkFrame(avail\_frame)

day\_frame.pack(fill="x", pady=5)

ctk.CTkLabel(day\_frame, text=day, width=100).pack(side="left", padx=10)

entry = ctk.CTkEntry(day\_frame, placeholder\_text="e.g., 9-12, 14-16 or 0", width=300)

entry.pack(side="right", padx=10)

day\_entries[day] = entry

# Save button

def save\_setup():

# Validate subjects

if not subj\_list:

messagebox.showerror("Error", "Please add at least one subject.")

return

avail\_data = {}

for day, entry in day\_entries.items():

text = entry.get().strip()

time = []

# store if its 0

if text == "0":

time = [] # list

elif text:

ranges = [r.strip() for r in text.split(',')]

for time\_range in ranges:

if '-' in time\_range:

try:

start, end = map(int, time\_range.split('-'))

if start < end and 0 <= start <= 23 and 1 <= end <= 24:

time.extend(range(start, end))

else:

messagebox.showerror("Error", f"Invalid time range for {day}: {time\_range}\nHours must be between 0-23 and end time must be greater than start time.")

return

except ValueError:

messagebox.showerror("Error", f"Invalid format for {day}: {time\_range}\nUse format like '9-12'")

return

avail\_data[day] = time

try:

# Save subjects to db

conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = conn.cursor()

# Add subjects to the db

for subject in subj\_list:

try:

cursor.execute("INSERT INTO subjects (user\_id, subject\_name) VALUES (%s, %s)",

(user\_id, subject))

except:

pass

# save to db

try:

# Clear existing records for this user

cursor.execute("DELETE FROM availability WHERE user\_id = %s", (user\_id,))

# Insert new availability data with minimal approach

for day, hours in avail\_data.items():

if not hours:

# If no hours, insert with NULL for hours

cursor.execute(

"INSERT INTO availability (user\_id, day, hours) VALUES (%s, %s, NULL)",

(user\_id, day)

)

else:

# For days with hours, use a very simple comma-separated string

# Limiting to avoid any potential issues

if len(hours) > 10:

hours = hours[:10] # Limit to first 10 hours to avoid any potential size issues

hours\_str = ','.join(str(h) for h in hours)

cursor.execute(

"INSERT INTO availability (user\_id, day, hours) VALUES (%s, %s, %s)",

(user\_id, day, hours\_str)

)

except Exception as e:

print(f"Warning: Could not save availability to database: {e}")

conn.commit()

conn.close()

# Close the setup window

setup\_window.destroy()

try:

tm\_allocation(user\_id)

except Exception as e:

print(f"Warning: Failed to calculate time allocation: {e}")

# If callback function was provided, call it

if callback\_function is not None:

callback\_function()

except Exception as e:

messagebox.showerror("Error", f"Failed to save setup: {e}")

return

save\_button = ctk.CTkButton(main, text="Save and Continue",

command=save\_setup,

font=("Arial", 14, "bold"))

save\_button.pack(pady=20)

# Run the setup window

setup\_window.mainloop()

def tm\_allocation(user\_id):

print("start of allocation")

# Get availability and subjects

daily\_hours = user\_availability(user\_id)

subjects = subj\_scores(user\_id)

total\_tm = sum(len(hours) for hours in daily\_hours.values())

print(f"\nTotal weekly study time: {total\_tm} hours")

if not subjects:

print("Error: No subjects found. Please add subjects first.")

return False

if total\_tm == 0:

print("Error:No time allocated")

return False

adjusted\_weights = {subject: score for subject, score in subjects.items()}

total\_weight = sum(adjusted\_weights.values())

if total\_weight == 0:

print("Warning: Total weight is 0, using equal weights for all subjects")

adjusted\_weights = {subject: 1 for subject in subjects}

total\_weight = len(subjects)

allocation = {

subject: max(round((weight / total\_weight) \* total\_tm), 1)

for subject, weight in adjusted\_weights.items()

}

allocation\_sum = sum(allocation.values())

if allocation\_sum != total\_tm and allocation:

difference = total\_tm - allocation\_sum

try:

weak\_subj = min(allocation, key=allocation.get)

allocation[weak\_subj] += difference

except ValueError:

print("Warning: Could not adjust allocation, using equal allocation")

if subjects:

equal\_time = total\_tm // len(subjects)

allocation = {subject: equal\_time for subject in subjects}

# Print allocation for debugging

print("\nTime allocation per subject:")

for subject, hours in allocation.items():

print(f"{subject}: {hours} hours")

start\_dt = datetime.today()

start\_week = start\_dt - timedelta(days=start\_dt.weekday())

daily\_alloc = []

hours\_left = allocation.copy()

day\_m = {

0: "Monday", 1: "Tuesday", 2: "Wednesday",

3: "Thursday", 4: "Friday", 5: "Saturday", 6: "Sunday"

}

for i in range(7): # loop

day\_name = day\_m[i]

hours = daily\_hours.get(day\_name, [])

if not hours:

continue

date = (start\_week + timedelta(days=i)).strftime('%Y-%m-%d')

slots = 0

avble\_subjects = sorted(hours\_left.keys(), key=lambda x: hours\_left[x], reverse=True)

while slots < len(hours) and avble\_subjects:

for subject in avble\_subjects:

if hours\_left[subject] > 0:

start\_time = hours[slots]

end\_time = start\_time + 1

time\_str = f"{start\_time:02}:00:00"

daily\_alloc.append((date, time\_str, subject))

hours\_left[subject] -= 1

if (slots + 1 < len(hours) and hours\_left[subject] > 0 and

hours[slots + 1] == end\_time and random.random() < 0.5):

start\_time = hours[slots + 1]

end\_time = start\_time + 1

time\_str = f"{start\_time:02}:00:00"

daily\_alloc.append((date, time\_str, subject))

hours\_left[subject] -= 1

slots += 1

slots += 1

break

# Update available subjects list

avble\_subjects = [s for s in avble\_subjects if hours\_left[s] > 0]

if slots >= len(hours) or not avble\_subjects:

break

save\_plan(user\_id, daily\_alloc)

print("plan saved")

for date, time, title in daily\_alloc:

print(f"{date} - {time}: {title}")

print("DONE")

return True

if \_\_name\_\_ == "\_\_main\_\_":

# Just print a message that the module is loaded

print("Study time allocation module loaded.")

### Graph\_bar.py

import mysql.connector

import matplotlib.pyplot as plt

import numpy as np

import tkinter as tk

from tkinter import ttk

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from scipy.interpolate import CubicSpline

from datetime import datetime

# My db settings

DB\_STUFF = {

'user': 'root',

'password': '',

'host': 'localhost',

'database': 'nea',

}

class ScoreVisualizer(tk.Frame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

# Make sure user\_id is an int not a tuple (was getting weird errors before)

self.user\_id = int(user\_id) if isinstance(user\_id, (int, str)) else user\_id[0]

# Keep track of callbacks so they don't disappear

self.callbacks = []

# Dark theme setup

self.configure(bg='#141414')

self.style = ttk.Style()

self.style.configure('TFrame', background='#141414')

self.style.configure('TLabelframe', background='#141414', foreground='white')

self.style.configure('TLabelframe.Label', background='#141414', foreground='white')

self.style.configure('TCheckbutton', background='#141414', foreground='black')

self.style.configure('TCombobox', background='#141414', foreground='white')

# Get the data from DB

my\_data = self.load\_data()

if not my\_data:

print("No data found! Using empty dataset instead")

my\_data = {}

sorted\_stuff = self.bubble\_sort(my\_data) # sort using my bubble sort implementation

# Main container

self.main\_frm = tk.Frame(self, bg='#141414')

self.main\_frm.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

self.plot\_frm = tk.Frame(self.main\_frm)

self.plot\_frm.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

self.option\_frm = tk.Frame(self.main\_frm, bg='#141414')

self.option\_frm.pack(side=tk.BOTTOM, fill=tk.Y, padx=10, pady=10)

# Checkboxes for subject selection

subj\_vars = self.Subject\_slctr(self.option\_frm, sorted\_stuff,

lambda: self.update\_plot(sorted\_stuff, self.plot\_frm, subj\_vars, time\_range, yr))

time\_range = self.month\_range\_slctr(self.option\_frm,

lambda: self.update\_plot(sorted\_stuff, self.plot\_frm, subj\_vars, time\_range, yr))

yr = self.year\_slctr(self.option\_frm,

lambda: self.update\_plot(sorted\_stuff, self.plot\_frm, subj\_vars, time\_range, yr))

# Show latest scores in a box

self.last\_scores(self.option\_frm, sorted\_stuff)

# Draw initial graph with everything selected

self.update\_plot(sorted\_stuff, self.plot\_frm, subj\_vars, time\_range, yr)

# Clean up when closing

self.bind("<Destroy>", self.cleanup)

def cleanup(self, event=None):

for cb in self.callbacks:

try:

self.after\_cancel(cb)

except Exception:

pass

# Close matplotlib figures

plt.close('all')

def last\_scores(self, parent, data):

# This shows the most recent scores for quick reference

latest\_frame = ttk.LabelFrame(parent, text="Latest Scores", style='TLabelframe')

latest\_frame.pack(side=tk.RIGHT, fill=tk.X, padx=10, pady=10)

self.style.configure('Latest.TLabel',

background='#141414',

foreground='white',

font=('Arial', 10))

# Find recent scores

latest = {}

for subj, scores in data.items():

if scores:

sorted\_scores = sorted(scores,

key=lambda x: (x['year'], x['month']),

reverse=True)

last\_score = sorted\_scores[0]

latest[subj] = f"{last\_score['score']} ({self.get\_month\_name(last\_score['month'])} {last\_score['year']})"

# Show the scores in a graph

for col, (subj, score) in enumerate(latest.items()):

ttk.Label(latest\_frame,

text=f"{subj}:",

style='Latest.TLabel'

).grid(row=0, column=col, sticky='w', padx=5, pady=2)

ttk.Label(latest\_frame, text=score, style='Latest.TLabel',foreground='#1f77b4' ).grid(row=1, column=col, sticky='e', padx=5, pady=2)

for kid in latest\_frame.winfo\_children():

kid.grid\_configure(padx=5, pady=2)

@staticmethod

def get\_month\_name(month\_num):

months = ["January", "February", "March", "April", "May", "June",

"July", "August", "September", "October", "November", "December"]

return months[month\_num - 1] # adjust for 0-indexing

@staticmethod

def get\_current\_year\_and\_month():

now = datetime.now()

return now.year, now.month

@staticmethod

def get\_time\_range\_months(year, time\_range, current\_year, current\_month):

# This was tricky to get right!

if year == current\_year:

# Don't show future months

if time\_range == "3 Months":

start = max(1, current\_month - 2)

return list(range(start, current\_month + 1))

elif time\_range == "6 Months":

start = max(1, current\_month - 5)

return list(range(start, current\_month + 1))

else: # Full Year

return list(range(1, current\_month + 1))

else:

# For past/future years show different parts of the year

if time\_range == "3 Months":

return list(range(10, 13)) if year < current\_year else list(range(1, 4))

elif time\_range == "6 Months":

return list(range(7, 13)) if year < current\_year else list(range(1, 7))

else: # Full Year

return list(range(1, 13))

def load\_data(self):

# Get scores from database

try:

# Connect to MySQL

conn = mysql.connector.connect(\*\*DB\_STUFF)

cursor = conn.cursor(dictionary=True)

# Debug stuff

print(f"Looking up user\_id: {self.user\_id} (type: {type(self.user\_id)})")

# Get subjects first

cursor.execute("SELECT subject\_id, subject\_name FROM subjects WHERE user\_id = %s", (self.user\_id,))

subjects = cursor.fetchall()

# Start with empty lists for each subject

all\_data = {subject['subject\_name']: [] for subject in subjects}

# Then get scores if we have subjects

if subjects:

subj\_ids = [subject['subject\_id'] for subject in subjects]

# Make SQL placeholders

placeholders = ', '.join(['%s'] \* len(subj\_ids))

cursor.execute(f"""

SELECT subjects.subject\_name, scores.month, scores.year, scores.score

FROM scores

JOIN subjects ON subjects.subject\_id = scores.subject\_id

WHERE subjects.subject\_id IN ({placeholders})

""", tuple(subj\_ids))

results = cursor.fetchall()

print(f"Found {len(results)} scores for {len(subjects)} subjects")

# Organize into our data structure

for row in results:

subj = row['subject\_name']

month = row['month']

year = row['year']

score = row['score']

all\_data[subj].append({'month': month, 'year': year, 'score': score})

return all\_data

except mysql.connector.Error as err:

print(f"DB Error: {err}")

return {}

except Exception as e:

print(f"Something went wrong: {e}")

return {}

finally:

if 'conn' in locals() and conn.is\_connected():

cursor.close()

conn.close()

print("DB connection closed")

@staticmethod

def bubble\_sort(data): # Used bubble sort

sorted\_data = {}

for subj, scores in data.items():

sorted\_scores = scores.copy()

n = len(sorted\_scores)

for i in range(n):

for j in range(0, n-i-1):

if sorted\_scores[j]['month'] > sorted\_scores[j+1]['month']:

sorted\_scores[j], sorted\_scores[j+1] = sorted\_scores[j+1], sorted\_scores[j]

sorted\_data[subj] = sorted\_scores

return sorted\_data

def plot\_scores(self, data, frame, selected\_subjects, time\_range, year):

# Clear old graph

for widget in frame.winfo\_children():

widget.destroy()

# Stop memory leaks

plt.close('all')

# Dark theme looks way better

plt.style.use('dark\_background')

fig, ax = plt.subplots(figsize=(10, 5))

# My colors - blue for Math, pink for Science, etc.

colors = ['#1f77b4', '#ff69b4', '#ff4500', '#00cc00', '#9400d3', '#ff7f00']

line\_styles = ['-', '--', '-.', ':']

# Match dark theme

fig.patch.set\_facecolor('#1a1a1a')

ax.set\_facecolor('#1a1a1a')

current\_year, current\_month = self.get\_current\_year\_and\_month()

months = self.get\_time\_range\_months(year, time\_range, current\_year, current\_month)

# Fallback if something went wrong

if not months:

months = list(range(1, 13))

month\_names = [self.get\_month\_name(m) for m in months]

x = np.arange(len(month\_names))

has\_any\_data = False

plot\_lines = [] # For legend

# Handle both cases - with data and without

if not data or not selected\_subjects:

# Empty graph case

empty\_line, = ax.plot(x, [0] \* len(x), label="No subjects available",

marker='o', markersize=8, markerfacecolor='gray',

markeredgecolor='white', markeredgewidth=1.5,

color='gray', linestyle='--', linewidth=1.5)

plot\_lines.append(empty\_line)

ax.text(0.5, 0.5, "No subjects or data available",

horizontalalignment='center', verticalalignment='center',

transform=ax.transAxes, color='white', fontsize=14)

else:

# Plot each selected subject

for i, subj in enumerate(selected\_subjects):

if subj not in data:

continue

scores = data[subj]

my\_color = colors[i % len(colors)]

my\_style = line\_styles[i % len(line\_styles)]

# Only show scores from selected year

year\_scores = [entry for entry in scores if entry['year'] == year]

# Always show a line even with no scores

if not year\_scores:

print(f"No {year} scores for {subj}, showing zero line")

# Empty line with markers

line, = ax.plot(x, [0] \* len(x), label=f"{subj}",

marker='o', markersize=8, markerfacecolor=my\_color,

markeredgecolor='white', markeredgewidth=1.5,

color=my\_color, linestyle='--', linewidth=2)

plot\_lines.append(line)

has\_any\_data = True

continue

has\_any\_data = True

score\_month = {entry['month']: entry['score'] for entry in year\_scores}

# Get score for each month (0 if missing)

values = [score\_month.get(m, 0) for m in months]

# Simple line for 1 point

if len(values) < 2:

line, = ax.plot(x, values, label=subj, marker='o', markersize=8,

color=my\_color, linestyle=my\_style, linewidth=2)

plot\_lines.append(line)

else:

# Try to make a smooth curve

try:

# This looks nicer than straight lines

spline = CubicSpline(x, values)

x\_smooth = np.linspace(0, len(month\_names)-1, 300)

y\_smooth = spline(x\_smooth)

line, = ax.plot(x\_smooth, y\_smooth, label=subj, color=my\_color,

linestyle=my\_style, linewidth=2.5)

plot\_lines.append(line)

# Add dots for actual data points

ax.scatter(x, values, s=50, color=my\_color, edgecolor='#ffffff',

linewidth=1, zorder=4)

except Exception as e:

print(f"Spline error: {e}")

# Just do regular line if spline fails

line, = ax.plot(x, values, label=subj, marker='o', markersize=8,

color=my\_color, linestyle=my\_style, linewidth=2)

plot\_lines.append(line)

# Still show something even with no data for this year

if not has\_any\_data and selected\_subjects:

# Zero lines for each subject

for i, subj in enumerate(selected\_subjects):

if subj not in data:

continue

my\_color = colors[i % len(colors)]

line, = ax.plot(x, [0] \* len(x), label=f"{subj}",

marker='o', markersize=8, markerfacecolor=my\_color,

markeredgecolor='white', markeredgewidth=1.5,

color=my\_color, linestyle='--', linewidth=2)

plot\_lines.append(line)

has\_any\_data = True

# Message to explain why it's empty

ax.text(0.5, 0.5, "No scores for selected subjects in this time period",

horizontalalignment='center', verticalalignment='center',

transform=ax.transAxes, color='white', fontsize=14)

# Graph settings

ax.set\_ylim(-5, 105) # Scores 0-100 with a bit of padding

# Make text visible on dark bg

ax.xaxis.label.set\_color('white')

ax.yaxis.label.set\_color('white')

ax.title.set\_color('white')

ax.tick\_params(axis='x', colors='white')

ax.tick\_params(axis='y', colors='white')

# Subtle grid

ax.grid(color='#333333', linestyle='-', linewidth=0.5, alpha=0.8)

# Add legend if we have any plots

if plot\_lines:

legend = ax.legend(handles=plot\_lines, facecolor='#1a1a1a', edgecolor='#404040',

fontsize=10, labelcolor='white')

if legend:

for txt in legend.get\_texts():

txt.set\_color("white")

ax.set\_xlabel("Month")

ax.set\_ylabel("Score")

ax.set\_title(f"Student Scores for {year} ({time\_range})")

ax.set\_xticks(x)

ax.set\_xticklabels(month\_names, rotation=0)

# Add to the frame

canvas = FigureCanvasTkAgg(fig, master=frame)

canvas.draw()

canvas.get\_tk\_widget().pack(fill=tk.BOTH, expand=True)

def Subject\_slctr(self, parent, data, update\_callback):

# Subject selector checkboxes

checkbox\_frame = ttk.LabelFrame(parent, text="Select Subjects")

checkbox\_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)

style = ttk.Style()

style.configure("White.TCheckbutton",

foreground="white",

background='#141414')

subject\_vars = {}

if data:

# Add checkboxes for each subject

for subj in data.keys():

var = tk.BooleanVar(value=True) # checked by default

cb = ttk.Checkbutton(checkbox\_frame,

text=subj,

variable=var,

command=update\_callback,

style="White.TCheckbutton")

cb.pack(anchor=tk.W, padx=5, pady=2)

subject\_vars[subj] = var

else:

# Handle no data case

var = tk.BooleanVar(value=True)

cb = ttk.Checkbutton(checkbox\_frame,

text="No subjects",

variable=var,

state="disabled",

style="White.TCheckbutton")

cb.pack(anchor=tk.W, padx=5, pady=2)

subject\_vars["No subjects"] = var

# Message explaining why

tk.Label(checkbox\_frame, text="No subject data available",

bg='#141414', fg='gray').pack(pady=5)

return subject\_vars

def month\_range\_slctr(self, parent, update\_callback):

# Time range selector

time\_frame = ttk.LabelFrame(parent, text="Select Time Range")

time\_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)

time\_range = tk.StringVar(value="6 Months") # default to 6 months

choices = ["Full Year", "6 Months", "3 Months"]

dropdown = ttk.Combobox(time\_frame, textvariable=time\_range, values=choices, state="readonly")

dropdown.pack(padx=5, pady=5)

# Need to keep ref to callback

self.dropdown\_callback = lambda event: update\_callback()

dropdown.bind("<<ComboboxSelected>>", self.dropdown\_callback)

return time\_range

def year\_slctr(self, parent, update\_callback):

# Year selector

year\_frame = ttk.LabelFrame(parent, text="Select Year")

year\_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=10, pady=10)

current\_year, \_ = self.get\_current\_year\_and\_month()

year = tk.IntVar(value=current\_year)

# Might need to add more years later

years = [2024, 2025, 2026]

dropdown = ttk.Combobox(year\_frame, textvariable=year, values=years, state="readonly")

dropdown.pack(padx=5, pady=5)

# Keep ref to callback

self.year\_callback = lambda event: update\_callback()

dropdown.bind("<<ComboboxSelected>>", self.year\_callback)

return year

def update\_plot(self, data, frame, subject\_vars, time\_range, year):

# Figure out which subjects are checked

selected = [s for s, var in subject\_vars.items() if var.get()]

# And redraw everything

self.plot\_scores(data, frame, selected, time\_range.get(), year.get())

### Quotes.py

import customtkinter as ctk

import json

import random

from datetime import datetime

# Load quotes and tips

def load\_quotes():

with open("quotes.json", "r") as file:

data = json.load(file)

return data["quotes"], data["tips"]

# Select a quote or tip for the day

def daily\_msg():

tdy = datetime.today().strftime("%Y-%m-%d")

try:

with open("last\_message.json", "r") as file:

last\_data = json.load(file)

if last\_data["date"] == tdy:

return last\_data["message"]

except FileNotFoundError:

pass

# Pick a new quote or tip

quotes, tips = load\_quotes()

message = random.choice(quotes + tips)

# Save the message to keep it for the day

with open("last\_message.json", "w") as file:

json.dump({"date": tdy, "message": message}, file)

return message

# CustomTkinter UI

class MotivationQuote(ctk.CTkFrame):

def \_\_init\_\_(self,parent,\*args,\*\*kwargs):

super().\_\_init\_\_(parent,\*args,\*\*kwargs)

self.labl = ctk.CTkLabel(self, text="📢 Today's Motivation", font=("Arial", 20))

self.labl.pack(pady=20)

self.msg = daily\_msg()

self.quote\_label = ctk.CTkLabel(self, text=self.msg, wraplength=400, font=("Arial", 16))

self.quote\_label.pack(pady=20)

def refresh\_message(self):

self.msg = daily\_msg()

self.quote\_label.configure(text=self.msg)

### Resources.py

import tkinter as tk

import customtkinter as ctk

from tkinter import ttk, messagebox

import json

import webbrowser

class ResourceLibrary(ctk.CTkFrame):

def \_\_init\_\_(self,parent,\*args,\*\*kwargs):

super().\_\_init\_\_(parent,\*args,\*\*kwargs)

self.resources = {}

self.resources\_loader()

self.GUI()

def resources\_loader(self):

try:

with open("resources.json", "r") as file:

self.resources = json.load(file)

except FileNotFoundError:

messagebox.showerror("Error", "The file 'resources.json' was not found.")

except json.JSONDecodeError:

messagebox.showerror("Error", "Error decoding 'resources.json'. Please check the file format.")

except Exception as e:

messagebox.showerror("Error", f"Failed to load resources: {e}")

def GUI(self):

main\_frame = ctk.CTkFrame(self)

main\_frame.pack(padx=10, pady=10, fill=tk.BOTH, expand=True)

ctk.CTkLabel(main\_frame, text="Enter Subject:").pack(pady=5)

self.subject\_entry = ctk.CTkEntry(main\_frame)

self.subject\_entry.pack(pady=5)

self.search\_button = ctk.CTkButton(main\_frame,fg\_color="#7f5af0", text="Search", command=self.get\_resource)

self.search\_button.pack(pady=5)

self.resource\_frame = ctk.CTkScrollableFrame(main\_frame)

self.resource\_frame.pack(pady=10, fill=tk.BOTH, expand=True)

def get\_resource(self):

for widget in self.resource\_frame.winfo\_children():

widget.destroy()

input\_subject = self.subject\_entry.get().strip().lower()

if not input\_subject:

messagebox.showinfo("Input Required", "Please enter a subject to search.")

return

matched\_subject = self.binary\_search\_match(input\_subject)

if not matched\_subject:

messagebox.showinfo("Not Found", "No matching subject found.")

return

resources = self.resources.get(matched\_subject, [])

if not resources:

messagebox.showinfo("No Resources", "No resources available for this subject.")

return

for resource in resources:

frame = ctk.CTkFrame(self.resource\_frame, corner\_radius=5, border\_width=1)

frame.pack(pady=5, fill=tk.X)

title = ctk.CTkLabel(frame, text=resource["title"], font=("Arial", 12, "bold"))

title.pack(anchor="w", padx=5, pady=2)

description = ctk.CTkLabel(frame, text=resource["description"], wraplength=550)

description.pack(anchor="w", padx=5, pady=2)

link = ctk.CTkButton(frame, text="View Resource", command=lambda url=resource["link"]: webbrowser.open(url))

link.pack(anchor="w", padx=5, pady=5)

# binary search to get the reource

def binary\_search\_match(self, subject):

subjects = sorted(self.resources.keys(), key=lambda s: s.lower())

low, high = 0, len(subjects) - 1

while low <= high:

mid = (low + high) // 2

mid\_subject = subjects[mid]

mid\_subject\_lower = mid\_subject.lower()

if mid\_subject\_lower == subject:

return mid\_subject

elif mid\_subject\_lower < subject:

low = mid + 1

else:

high = mid - 1

return None

if \_\_name\_\_ == "\_\_main\_\_":

app = ResourceLibrary()

app.mainloop()

### Availability.py

import customtkinter as ctk

from tkinter import messagebox

import json

import re

import os

import mysql.connector

from calculator\_normal import tm\_allocation

class AvailabilityEditor(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id):

super().\_\_init\_\_(parent)

self.txt\_entries = {}

self.user\_id = user\_id

self.avl = self.load\_data\_avl()

self.ui()

def load\_data\_avl(self):

#Load from db then if that fails load from file

default\_avl = {day: [] for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]}

# First try loading from database

try:

conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = conn.cursor()

cursor.execute("SELECT day, hours FROM availability WHERE user\_id = %s", (self.user\_id,))

results = cursor.fetchall()

if results:

db\_avl = default\_avl.copy()

for day, hours\_str in results:

# Default to empty list (which means "0" in the UI)

hours\_list = []

# Only try to parse if hours is not NULL and not empty

if hours\_str:

try:

# Convert the comma-separated string back to a list of integers

hours\_list = [int(h) for h in hours\_str.split(',') if h.strip()]

except (ValueError, AttributeError) as e:

print(f"Error parsing hours for {day}: {e}")

# Store the hours list (empty or with values)

db\_avl[day] = hours\_list

print(f"Loaded availability from database for user {self.user\_id}: {db\_avl}")

return db\_avl

except Exception as e:

print(f"Error loading availability from database: {e}")

finally:

if 'conn' in locals() and conn and conn.is\_connected():

cursor.close()

conn.close()

# If database load failed, try loading from file

try:

# Try to load from user-specific JSON file

filename = f"availability\_data/availability\_{self.user\_id}.json"

if os.path.exists(filename) and os.path.getsize(filename) > 0:

with open(filename, "r") as file:

data = json.load(file)

print(f"Loaded availability from file for user {self.user\_id}: {data}")

return data

except Exception as e:

print(f"Error loading availability from file: {e}")

print(f"Using default availability for user {self.user\_id}")

return default\_avl

def save\_data(self, data):

#Save availability

try:

try:

# First attempt to save to database

self.save\_DB(data)

except Exception as e:

print(f"Warning: Couldn't save to database, using file storage only: {e}")

# If database fails, make sure we save to file

self.save\_file(data)

messagebox.showinfo("Success", "Availability saved successfully!")

# Run the time allocation calculation

tm\_allocation(self.user\_id)

except Exception as e:

messagebox.showerror("Error", f"Failed to save availability: {e}")

def save\_DB(self, data):

"""Attempt to save to database - this is a best-effort approach"""

conn = None

try:

conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = conn.cursor()

# First try removing existing records for this user

cursor.execute("DELETE FROM availability WHERE user\_id = %s", (self.user\_id,))

# Now insert new records

for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]:

hours = data.get(day, [])

if not hours:

# No hours for this day

cursor.execute(

"INSERT INTO availability (user\_id, day, hours) VALUES (%s, %s, NULL)",

(self.user\_id, day)

)

print(f"Saved {day} with NULL hours to database")

else:

if len(hours) > 24: # Max 24 hours in a day

hours = hours[:24]

hours\_str = ','.join(str(h) for h in hours)

cursor.execute(

"INSERT INTO availability (user\_id, day, hours) VALUES (%s, %s, %s)",

(self.user\_id, day, hours\_str)

)

print(f"Saved {day} with hours {hours\_str} to database")

conn.commit()

print(f"Successfully saved all availability data to database for user {self.user\_id}")

# save to file as backup

self.save\_file(data)

except Exception as e:

print(f"Database save error: {e}")

if conn:

conn.rollback()

raise

finally:

if conn and conn.is\_connected():

cursor.close()

conn.close()

def save\_file(self, data):

try:

os.makedirs("availability\_data", exist\_ok=True)

filename = f"availability\_data/availability\_{self.user\_id}.json"

with open(filename, "w") as file:

json.dump(data, file)

print(f"Saved availability data to file: {filename}")

except Exception as e:

print(f"Error saving to file: {e}")

def update(self):

updated\_data = {}

for day, entry in self.txt\_entries.items():

text = entry.get().strip()

times = []

# Check for "0" entry which explicitly indicates no availability

if text == "0":

# User explicitly says not available on this day

times = [] # Empty list indicates no availability

elif text:

matches = re.findall(r"(\d{1,2})-(\d{1,2})", text)

for start, end in matches:

start, end = int(start), int(end)

if start < end:

times.extend(range(start, end))

updated\_data[day] = times

self.save\_data(updated\_data)

def ui(self):

frm = ctk.CTkFrame(self)

frm.pack(padx=10, pady=10, fill="both", expand=True)

ctk.CTkLabel(frm, text="Enter availability (e.g., 9-12, 14-16 or 0 for no availability):",

font=("Arial", 12, "bold")).pack()

for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]:

row\_frame = ctk.CTkFrame(frm)

row\_frame.pack(fill="x", pady=2)

ctk.CTkLabel(row\_frame, text=day, font=("Arial", 10, "bold"), width=10).pack(side="left", padx=5)

entry = ctk.CTkEntry(row\_frame, width=150, height=10)

# Get times from availability dictionary, defaulting to empty list

existing\_times = self.avl.get(day, [])

# Print debug info

print(f"Day: {day}, Times: {existing\_times}")

if not existing\_times:

# No times for this day

entry.insert(0, "0") # Insert "0" to indicate no availability

else:

# Sort times to ensure proper range creation

existing\_times.sort()

# Group consecutive times into ranges

time\_ranges = []

if existing\_times:

start = existing\_times[0]

current\_end = start

for i in range(1, len(existing\_times)):

if existing\_times[i] != existing\_times[i - 1] + 1:

# End of a consecutive range

time\_ranges.append(f"{start}-{current\_end + 1}")

start = existing\_times[i]

current\_end = existing\_times[i]

# Add the final range

time\_ranges.append(f"{start}-{current\_end + 1}")

entry.insert(0, ", ".join(time\_ranges))

entry.pack(side="right", padx=5)

self.txt\_entries[day] = entry

ctk.CTkButton(self, text="Save", command=self.update, font=("Arial", 10, "bold"),).pack(pady=10)

### Analysis.py

import mysql.connector

import json

import datetime

import webbrowser

from pathlib import Path

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

import tkinter as tk

from tkinter import ttk,scrolledtext

import customtkinter as ctk

DB\_CONFIG = {

'user': 'root', # MySQL username

'password': '', # MySQL password

'host': 'localhost', #host

'database': 'nea', # database name

}

# Set appearance for CustomTkinter (dark mode by default)

ctk.set\_appearance\_mode("dark")

ctk.set\_default\_color\_theme("dark-blue")

def db():

return mysql.connector.connect(\*\*DB\_CONFIG)

class StudyAdvisorApp(ctk.CTkFrame):

def \_\_init\_\_(self,parent,user\_id,\*args,\*\*kwargs):

super().\_\_init\_\_(parent,\*args,\*\*kwargs)

self.user\_id=user\_id

# File paths

self.scores\_file = Path("scores.json")

self.feelings\_file = Path("feelings.json")

self.methods\_file = Path("learning\_methods.json")

self.ui()

self.load\_subj()

# Start recursive update of recommendation frames

self.recom\_loop()

def ui(self):

# Main container frame

self.main\_frm = ctk.CTkFrame(self, fg\_color="#1f2a3c")

self.main\_frm.pack(fill="both", expand=True, padx=20, pady=20)

# Subject Selection

subj\_labl = ctk.CTkLabel(self.main\_frm, text="Select Subject:", font=("Arial", 14))

subj\_labl.grid(row=0, column=0, padx=10, pady=10, sticky="w")

self.subj\_var = tk.StringVar()

self.subj\_combobox = ttk.Combobox(self.main\_frm, textvariable=self.subj\_var, state="readonly")

self.subj\_combobox.grid(row=0, column=1, padx=10, pady=10, sticky="ew")

self.subj\_combobox.bind("<<ComboboxSelected>>", lambda e: self.update\_scores())

# Scores

scores\_title = ctk.CTkLabel(self.main\_frm, text="Last 3 Scores:", font=("Arial", 14))

scores\_title.grid(row=1, column=0, padx=10, pady=5, sticky="w")

self.scores\_labl = ctk.CTkLabel(self.main\_frm, text="", font=("Arial", 12))

self.scores\_labl.grid(row=1, column=1, padx=10, pady=5, sticky="w")

# Graph & Trend Panel

plot\_container = ctk.CTkFrame(self.main\_frm, fg\_color="#2e2e2e")

plot\_container.grid(row=2, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")

plot\_container.columnconfigure(0, weight=1)

plot\_container.columnconfigure(1, weight=1)

self.figure, self.ax = plt.subplots(figsize=(5, 2))

self.figure.patch.set\_facecolor('#333333') # Figure background

self.ax.set\_facecolor('#333333') # Plot background

self.canvas = FigureCanvasTkAgg(self.figure, master=plot\_container)

self.canvas.get\_tk\_widget().grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

self.trend\_frm = ctk.CTkFrame(plot\_container, fg\_color="#3e3e3e")

self.trend\_frm.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

trend\_title = ctk.CTkLabel(self.trend\_frm, text="Performance Trend", font=("Arial", 14, "bold"))

trend\_title.pack(anchor="w", padx=10, pady=5)

self.crnt\_improve\_labl = ctk.CTkLabel(self.trend\_frm, text="Current Improvement: N/A", font=("Arial", 12))

self.crnt\_improve\_labl.pack(anchor="w", padx=10, pady=5)

self.predicted\_improve\_labl = ctk.CTkLabel(self.trend\_frm, text="Predicted Next Increase: N/A", font=("Arial", 12))

self.predicted\_improve\_labl.pack(anchor="w", padx=10, pady=5)

struggle\_frm = ctk.CTkFrame(self.main\_frm, fg\_color="#2e2e2e")

struggle\_frm.grid(row=3, column=0, columnspan=2, padx=10, pady=10, sticky="ew")

struggle\_labl = ctk.CTkLabel(struggle\_frm, text="What are you struggling with?", font=("Arial", 14))

struggle\_labl.grid(row=0, column=0, columnspan=2, padx=10, pady=5, sticky="w")

self.struggle\_vars = {

"memory": tk.BooleanVar(),

"focus": tk.BooleanVar(),

"understanding": tk.BooleanVar(),

"motivation": tk.BooleanVar()

}

cb\_memory = ctk.CTkCheckBox(struggle\_frm, text="Remembering key terms", variable=self.struggle\_vars["memory"])

cb\_memory.grid(row=1, column=0, padx=10, pady=5, sticky="w")

cb\_understanding = ctk.CTkCheckBox(struggle\_frm, text="Understanding concepts", variable=self.struggle\_vars["understanding"])

cb\_understanding.grid(row=2, column=0, padx=10, pady=5, sticky="w")

cb\_focus = ctk.CTkCheckBox(struggle\_frm, text="Staying focused", variable=self.struggle\_vars["focus"])

cb\_focus.grid(row=1, column=1, padx=10, pady=5, sticky="w")

cb\_motivation = ctk.CTkCheckBox(struggle\_frm, text="Staying motivated", variable=self.struggle\_vars["motivation"])

cb\_motivation.grid(row=2, column=1, padx=10, pady=5, sticky="w")

# Feeling Input

feeling\_labl = ctk.CTkLabel(self.main\_frm, text="How are you feeling?", font=("Arial", 14))

feeling\_labl.grid(row=4, column=0, padx=10, pady=10, sticky="w")

self.feeling\_entry = ctk.CTkEntry(self.main\_frm, width=250)

self.feeling\_entry.grid(row=4, column=1, padx=10, pady=10, sticky="w")

# Recommendation Sections Container

rec\_container = ctk.CTkFrame(self.main\_frm, fg\_color="#2e2e2e")

rec\_container.grid(row=5, column=0, columnspan=2, padx=10, pady=10, sticky="nsew")

rec\_container.columnconfigure((0, 1, 2), weight=1)

# Performance Analysis

analysis\_frm = ctk.CTkFrame(rec\_container, fg\_color="#3e3e3e")

analysis\_frm.grid(row=0, column=0, padx=5, pady=5, sticky="nsew")

analysis\_labl = ctk.CTkLabel(analysis\_frm, text="Performance Analysis", font=("Arial", 12, "bold"))

analysis\_labl.pack(anchor="w", padx=5, pady=5)

self.analysis\_txt = tk.scrolledtext.ScrolledText(analysis\_frm, height=8, wrap=tk.WORD, bg="#333333", fg="white")

self.analysis\_txt.pack(fill="both", expand=True, padx=5, pady=5)

# Struggle-Specific Methods

method\_frm = ctk.CTkFrame(rec\_container, fg\_color="#3e3e3e")

method\_frm.grid(row=0, column=1, padx=5, pady=5, sticky="nsew")

method\_labl = ctk.CTkLabel(method\_frm, text="Struggle-Specific Methods", font=("Arial", 12, "bold"))

method\_labl.pack(anchor="w", padx=5, pady=5)

self.method\_txt = tk.scrolledtext.ScrolledText(method\_frm, height=8, wrap=tk.WORD, bg="#333333", fg="white")

self.method\_txt.pack(fill="both", expand=True, padx=5, pady=5)

# Frame to hold resource buttons

self.resource\_button\_frm = ctk.CTkFrame(method\_frm, fg\_color="#3e3e3e")

self.resource\_button\_frm.pack(fill="x", padx=5, pady=5)

# Feeling-Based Suggestions

suggestion\_frm = ctk.CTkFrame(rec\_container, fg\_color="#3e3e3e")

suggestion\_frm.grid(row=0, column=2, padx=5, pady=5, sticky="nsew")

suggestion\_labl = ctk.CTkLabel(suggestion\_frm, text="Feeling-Based Suggestions", font=("Arial", 12, "bold"))

suggestion\_labl.pack(anchor="w", padx=5, pady=5)

self.suggestion\_text = tk.scrolledtext.ScrolledText(suggestion\_frm, height=8, wrap=tk.WORD, bg="#333333", fg="white")

self.suggestion\_text.pack(fill="both", expand=True, padx=5, pady=5)

# Configure grid weights for resizing

self.main\_frm.columnconfigure(1, weight=1)

self.main\_frm.rowconfigure(5, weight=1)

def load\_subj(self):

try:

conn = db()

cursor = conn.cursor()

cursor.execute("SELECT subject\_id, subject\_name FROM subjects WHERE user\_id=%s",(self.user\_id,))

subjects = cursor.fetchall()

conn.close()

if subjects:

self.subj\_map = {str(sub[0]): sub[1] for sub in subjects}

self.subj\_combobox['values'] = list(self.subj\_map.values())

self.subj\_var.set(list(self.subj\_map.values())[0])

self.update\_scores()

except Exception as e:

print(f"Error loading subjects: {e}")

def update\_scores(self):

subj\_name = self.subj\_var.get()

subj\_id = next((k for k, v in self.subj\_map.items() if v == subj\_name), None)

if not subj\_id:

self.scores\_labl.configure(text="No scores available")

return

try:

conn = db()

cursor = conn.cursor(dictionary=True)

cursor.execute("""

SELECT score, month, year

FROM scores

WHERE subject\_id = %s

ORDER BY year DESC, month DESC

LIMIT 3

""", (subj\_id,))

scores = cursor.fetchall()

conn.close()

if scores:

scores\_txt = ", ".join(f"{entry['score']}" for entry in scores)

self.scores\_labl.configure(text=scores\_txt)

self.plot\_scores(scores)

self.update\_trend(scores)

else:

self.scores\_labl.configure(text="No scores available")

except Exception as e:

self.scores\_labl.configure(text="Error loading scores")

print(e)

def plot\_scores(self, scores):

self.ax.clear()

self.figure.patch.set\_facecolor('#333333')

self.ax.set\_facecolor('#333333')

if scores:

try:

score\_values = [int(entry['score']) for entry in scores]

attempts = list(range(1, len(score\_values) + 1))

self.ax.plot(attempts, score\_values, marker='o', linestyle='-', color='b', label="Scores")

self.ax.fill\_between(attempts, score\_values, min(score\_values) - 5, color='blue', alpha=0.2)

self.ax.set\_title("Score Progression", fontsize=12, fontweight="bold", color='white')

self.ax.set\_xlabel("Attempts", fontsize=10, color='white')

self.ax.set\_ylabel("Scores", fontsize=10, color='white')

self.ax.tick\_params(colors='white')

self.ax.spines['bottom'].set\_color('white')

self.ax.spines['left'].set\_color('white')

self.ax.grid(True, linestyle='--', alpha=0.6, color='white')

self.ax.legend(facecolor='#333333', edgecolor='white')

self.ax.set\_ylim(min(score\_values) - 5, max(score\_values) + 5)

if len(score\_values) > 1:

percentage\_change = ((score\_values[-1] - score\_values[0]) / score\_values[0]) \* 100 if score\_values[0] != 0 else 0

self.ax.text(attempts[-1], score\_values[-1], f"{percentage\_change:.2f}% Change", fontsize=10, color='red')

except Exception as ex:

print(f"Error plotting scores: {ex}")

else:

self.ax.text(0.5, 0.5, "No scores available", fontsize=12, ha='center', transform=self.ax.transAxes, color='white')

self.canvas.draw()

def update\_trend(self, scores):

if scores and len(scores) >= 2:

try:

score\_values = [int(entry['score']) for entry in scores]

first, last = score\_values[0], score\_values[-1]

current\_improve = ((last - first) / first) \* 100 if first != 0 else 0

intervals = len(score\_values) - 1

avg\_increase = ((last - first) / intervals) if intervals > 0 else 0

predicted\_improve = (avg\_increase / first) \* 100 if first != 0 else 0

self.crnt\_improve\_labl.configure(text=f"Current Trend: {current\_improve:.2f}%")

self.predicted\_improve\_labl.configure(text=f"Predicted Next Increase: {predicted\_improve:.2f}%")

except Exception as ex:

self.crnt\_improve\_labl.configure(text="Current Trend: N/A")

self.predicted\_improve\_labl.configure(text="Predicted Next Increase: N/A")

print(f"Error computing trend: {ex}")

else:

self.crnt\_improve\_labl.configure(text="Current Trend: N/A")

self.predicted\_improve\_labl.configure(text="Predicted Next Increase: N/A")

def get\_recom(self):

recommendations = []

try:

with open(self.methods\_file, 'r') as f:

methods = json.load(f)

except Exception as e:

methods = {}

selected\_struggles = [key for key, var in self.struggle\_vars.items() if var.get()]

for struggle in selected\_struggles:

if struggle in methods:

method = methods[struggle]

rec\_text = (f"Recommended Method: {method['name']}\n"

f"Description: {method['description']}\nSteps:\n")

rec\_text += "\n".join(f"• {step}" for step in method['steps'])

recommendations.append((rec\_text, method.get('resource', '')))

return recommendations

def linear\_search\_feelings(self, user\_input):

try:

with open(self.feelings\_file, 'r') as f:

feelings\_dt = json.load(f)

user\_words = user\_input.lower().split()

recom = []

for feeling in feelings\_dt:

if feeling.lower() in user\_words:

recom.extend(feelings\_dt[feeling])

return recom

except Exception as e:

return []

def get\_performance\_recom(self, scores):

if not scores:

return []

try: # basic mathematical calculation

score\_values = [int(entry['score']) for entry in scores]

avg = sum(score\_values) / len(score\_values)

except Exception as e:

avg = 0

recom = []

if avg >= 70:

recom.append("High Performance Detected!")

recom.append("- Use advanced practice questions")

recom.append("- Teach concepts to peers")

elif 50 <= avg < 70:

recom.append("Moderate Understanding")

recom.append("- Focus on weak areas")

recom.append("- Use spaced repetition")

else:

recom.append("Needs Improvement")

recom.append("- Daily fundamental reviews")

recom.append("- Seek teacher assistance")

return recom

def generate\_recom(self):

# Clear previous recommendations

self.analysis\_txt.delete("1.0", tk.END)

self.method\_txt.delete("1.0", tk.END)

self.suggestion\_text.delete("1.0", tk.END)

for widget in self.resource\_button\_frm.winfo\_children():

widget.destroy()

subject = self.subj\_var.get()

try:

with open(self.scores\_file, 'r') as f:

data = json.load(f)

subject\_data = data.get(subject, [])

last\_three = subject\_data[-3:]

except Exception as e:

last\_three = []

#Performance Analysis

performance\_recs = self.get\_performance\_recom(last\_three)

if performance\_recs:

for rec in performance\_recs:

self.analysis\_txt.insert(tk.END, f"• {rec}\n")

else:

self.analysis\_txt.insert(tk.END, "No performance analysis available.\n")

#Struggle-Specific Methods

struggle\_recs = self.get\_recom()

if struggle\_recs:

for rec\_text, resource in struggle\_recs:

self.method\_txt.insert(tk.END, rec\_text + "\n")

if resource:

btn = ttk.Button(self.resource\_button\_frm, text="Go to Resource",

command=lambda url=resource: webbrowser.open(url))

btn.pack(pady=2, anchor="w")

self.method\_txt.insert(tk.END, "---------------------\n")

else:

self.method\_txt.insert(tk.END, "No specific struggle selected.\n")

#Feeling-Based Suggestions

feeling\_input = self.feeling\_entry.get().strip()

feeling\_recs = self.linear\_search\_feelings(feeling\_input)

if feeling\_recs:

for rec in feeling\_recs:

self.suggestion\_text.insert(tk.END, f"• {rec}\n")

else:

self.suggestion\_text.insert(tk.END, "No feeling-based suggestions available.\n")

def recom\_loop(self):

self.generate\_recom()

self.after(2000, self.recom\_loop)

if \_\_name\_\_ == "\_\_main\_\_":

app = StudyAdvisorApp()

app.mainloop()

### User\_profile.py

import mysql.connector

import hashlib

from customtkinter import \*

class UserProfile(CTkFrame):

def \_\_init\_\_(self, parent, user\_id):

super().\_\_init\_\_(parent)

self.user\_id = int(user\_id) # Ensure user\_id is an integer

self.username, self.password\_hash = self.user\_info()

self.db()

self.ui()

def db(self):

#Connect to MySQL database

self.conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

self.cursor = self.conn.cursor()

self.cursor.execute("""

CREATE TABLE IF NOT EXISTS subjects (

id INT AUTO\_INCREMENT PRIMARY KEY,

user\_id INT,

subject VARCHAR(255) NOT NULL

)

""")

self.conn.commit()

def user\_info(self):

#Retrieve username and password hash from MySQL database

conn = mysql.connector.connect(

host="localhost",

user="root",

password="",

database="nea"

)

cursor = conn.cursor()

cursor.execute("SELECT username, password FROM users WHERE user\_id = %s", (self.user\_id,))

user = cursor.fetchone()

conn.close()

if user:

return user[0], user[1]

return "Unknown", ""

def ui(self):

#Create UI

CTkLabel(self, text=f"User: {self.username}", font=("Arial", 16)).pack(pady=10)

CTkLabel(self, text=f"Password Hash: {self.password\_hash}", font=("Arial", 12)).pack(pady=5)

self.subject\_entry = CTkEntry(self, placeholder\_text="Enter Subject")

self.subject\_entry.pack(pady=5)

add\_button = CTkButton(self, text="Add Subject", command=self.add\_subject)

add\_button.pack(pady=5)

self.subject\_frame = CTkFrame(self)

self.subject\_frame.pack(pady=5, fill="both", expand=True)

self.load\_subjects()

def load\_subjects(self):

#Load subjects from the DB

for widget in self.subject\_frame.winfo\_children():

widget.destroy()

self.cursor.execute("SELECT subject\_name FROM subjects WHERE user\_id = %s", (self.user\_id,))

subjects = self.cursor.fetchall()

# convert subject to buttons

for subject in subjects:

subject\_button = CTkButton(self.subject\_frame, text=subject[0], command=lambda s=subject[0]: self.edit\_window(s))

subject\_button.pack(pady=2, fill="x")

def edit\_window(self, subject):

#Open a popup window to edit the subject

popup = CTkToplevel(self)

popup.title("Edit Subject")

popup.geometry("300x150")

popup.attributes('-topmost', True)

CTkLabel(popup, text="Edit Subject:").pack(pady=5)

subject\_entry = CTkEntry(popup)

subject\_entry.insert(0, subject)

subject\_entry.pack(pady=5)

def save\_changes():

new\_subject = subject\_entry.get().strip()

if new\_subject:

self.cursor.execute("UPDATE subjects SET subject\_name = %s WHERE user\_id = %s AND subject\_name = %s", (new\_subject, self.user\_id, subject))

self.conn.commit()

self.load\_subjects()

popup.destroy()

save\_button = CTkButton(popup, text="Save", command=save\_changes)

save\_button.pack(pady=5)

delete\_button = CTkButton(popup, text="Delete", command=lambda: self.delete\_subject(subject, popup))

delete\_button.pack(pady=5)

def add\_subject(self):

#Add a new subject

subject = self.subject\_entry.get().strip()

if subject:

self.cursor.execute("INSERT INTO subjects (user\_id, subject\_name) VALUES (%s, %s)", (self.user\_id, subject))

self.conn.commit()

self.load\_subjects()

self.subject\_entry.delete(0, "end")

def delete\_subject(self, subject, popup=None):

#Delete the subject

self.cursor.execute("DELETE FROM subjects WHERE user\_id = %s AND subject\_name = %s", (self.user\_id, subject))

self.conn.commit()

self.load\_subjects()

if popup:

popup.destroy()

def cls\_con(self):

#Close the database when deleted

self.conn.close()

### ExamCountdown.py-

import customtkinter as ctk

from tkinter import messagebox

from tkcalendar import Calendar

import json

import os

from datetime import datetime

ctk.set\_appearance\_mode("System")

ctk.set\_default\_color\_theme("blue")

class ExamCountdown(ctk.CTkFrame):

def \_\_init\_\_(self,parent,\*args,\*\*kwargs):

super().\_\_init\_\_(parent,\*args,\*\*kwargs)

self.FILE\_PATH = "exam\_details.json"

self.ui()

self.update\_countdown()

def ui(self):

self.main\_frm = ctk.CTkFrame(self)

self.main\_frm.pack(padx=20, pady=20, fill="both", expand=True)

self.title\_labl = ctk.CTkLabel(

self.main\_frm, text="Exam Schedule", font=("Arial", 16, "bold")

)

self.title\_labl.pack(pady=10)

self.countdown = ctk.CTkLabel(

self.main\_frm, text="", justify="left"

)

self.countdown.pack(pady=10)

self.button\_frm = ctk.CTkFrame(self.main\_frm, fg\_color="transparent")

self.button\_frm.pack(pady=10)

self.add\_btn = ctk.CTkButton(

self.button\_frm, text="Add Exam Date", command=self.add\_window,fg\_color="#7f5af0"

)

self.add\_btn.pack(side="left", padx=5)

self.manage\_btn = ctk.CTkButton(

self.button\_frm, text="Manage Exams", command=self.manage\_window,fg\_color="#7f5af0"

)

self.manage\_btn.pack(side="left", padx=5)

def save\_details(self, date, subject):

try:

formatted\_date = datetime.strptime(date, "%m/%d/%y").strftime("%d-%m-%Y")

data = self.load\_details()

data.append({"exam\_date": formatted\_date, "subject": subject})

with open(self.FILE\_PATH, "w") as file:

json.dump(data, file, indent=4)

self.update\_countdown()

return True

except Exception as e:

messagebox.showerror("Error", f"Error saving exam: {str(e)}")

return False

def load\_details(self):

if os.path.exists(self.FILE\_PATH):

with open(self.FILE\_PATH, "r") as file:

data = json.load(file)

return data if isinstance(data, list) else []

return []

def delete\_exam(self, exam\_date, subject):

exams = [exam for exam in self.load\_details()

if not (exam["exam\_date"] == exam\_date and exam["subject"] == subject)]

with open(self.FILE\_PATH, "w") as file:

json.dump(exams, file, indent=4)

self.update\_countdown()

def update\_countdown(self):

exams = self.load\_details()

today = datetime.today()

countdown\_text = "Upcoming Exams:\n\n"

for exam in sorted(exams, key=lambda x: datetime.strptime(x["exam\_date"], "%d-%m-%Y")):

exam\_day = datetime.strptime(exam["exam\_date"], "%d-%m-%Y")

days\_left = (exam\_day - today).days

if days\_left < 0:

self.delete\_exam(exam["exam\_date"], exam["subject"])

continue

countdown\_text += f"• {exam['subject']} - {exam['exam\_date']} " \

f"({'Tomorrow' if days\_left == 1 else f'{days\_left} days left'})\n"

self.countdown.configure(

text=countdown\_text if exams else "No upcoming exams!"

)

def add\_window(self):

AddExamWindow(self)

def manage\_window(self):

ManageExamsWindow(self)

class AddExamWindow(ctk.CTkToplevel):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent)

self.parent = parent

self.title("Add Exam")

self.geometry("350x350")

self.\_setup\_window()

self.\_setup\_ui()

def \_setup\_window(self):

self.transient(self.parent)

self.grab\_set()

self.lift()

self.focus\_force()

def \_setup\_ui(self):

self.cal\_labl = ctk.CTkLabel(self, text="Select Exam Date:")

self.cal\_labl.pack(pady=5)

self.cal = Calendar(self, date\_pattern='mm/dd/yy')

self.cal.pack(pady=5)

self.subject\_labl = ctk.CTkLabel(self, text="Enter Subject:")

self.subject\_labl.pack(pady=5)

self.subject\_entry = ctk.CTkEntry(self)

self.subject\_entry.pack(pady=5)

self.save\_btn = ctk.CTkButton(

self, text="Save Exam Details", command=self.\_save\_exam

)

self.save\_btn.pack(pady=10)

def \_save\_exam(self):

subject = self.subject\_entry.get()

if not subject:

messagebox.showerror("Error", "Please enter a subject!")

return

if self.parent.save\_details(self.cal.get\_date(), subject):

self.destroy()

class ManageExamsWindow(ctk.CTkToplevel):

def \_\_init\_\_(self, parent):

super().\_\_init\_\_(parent)

self.parent = parent

self.title("Manage Exams")

self.geometry("400x400")

self.\_setup\_window()

self.\_setup\_ui()

self.\_refresh\_list()

def \_setup\_window(self):

self.transient(self.parent)

self.grab\_set()

self.lift()

self.focus\_force()

def \_setup\_ui(self):

self.scroll\_frm = ctk.CTkScrollableFrame(self)

self.scroll\_frm.pack(fill="both", expand=True, padx=10, pady=10)

def \_refresh\_list(self):

for widget in self.scroll\_frm.winfo\_children():

widget.destroy()

for exam in self.parent.load\_exam\_details():

item\_frm = ctk.CTkFrame(self.scroll\_frm)

item\_frm.pack(fill="x", pady=2)

label = ctk.CTkLabel(item\_frm,text=f"{exam['subject']} - {exam['exam\_date']}")

label.pack(side="left", padx=5)

btn\_frm = ctk.CTkFrame(item\_frm, fg\_color="transparent")

btn\_frm.pack(side="right")

edit\_btn = ctk.CTkButton(btn\_frm,text="Edit",width=50,command=lambda e=exam: EditExamWindow(self.parent, e))

edit\_btn.pack(side="left", padx=2)

delete\_btn = ctk.CTkButton(btn\_frm,text="Delete",width=50,fg\_color="#d44942",hover\_color="#a33732",command=lambda e=exam: self.\_delete\_exam(e))

delete\_btn.pack(side="left", padx=2)

def \_delete\_exam(self, exam):

self.parent.delete\_exam(exam["exam\_date"], exam["subject"])

self.\_refresh\_list()

class EditExamWindow(ctk.CTkToplevel):

def \_\_init\_\_(self, parent, exam):

super().\_\_init\_\_(parent)

self.parent = parent

self.exam = exam

self.title("Edit Exam")

self.geometry("350x350")

self.\_setup\_window()

self.\_setup\_ui()

def \_setup\_window(self):

self.transient(self.parent)

self.grab\_set()

self.lift()

self.focus\_force()

def \_setup\_ui(self):

self.cal\_labl = ctk.CTkLabel(self, text="Select New Exam Date:")

self.cal\_labl.pack(pady=5)

self.cal = Calendar(self, date\_pattern='mm/dd/yy')

self.cal.pack(pady=5)

self.subj\_labl = ctk.CTkLabel(self, text="Enter New Subject:")

self.subj\_labl.pack(pady=5)

self.subj\_entry = ctk.CTkEntry(self)

self.subj\_entry.insert(0, self.exam["subject"])

self.subj\_entry.pack(pady=5)

self.save\_btn = ctk.CTkButton(

self, text="Save Changes", command=self.\_save\_changes

)

self.save\_btn.pack(pady=10)

def \_save\_changes(self):

new\_subject = self.subj\_entry.get()

if not new\_subject:

messagebox.showerror("Error", "Please enter a subject!")

return

try:

new\_date = datetime.strptime(self.cal.get\_date(), "%m/%d/%y").strftime("%d-%m-%Y")

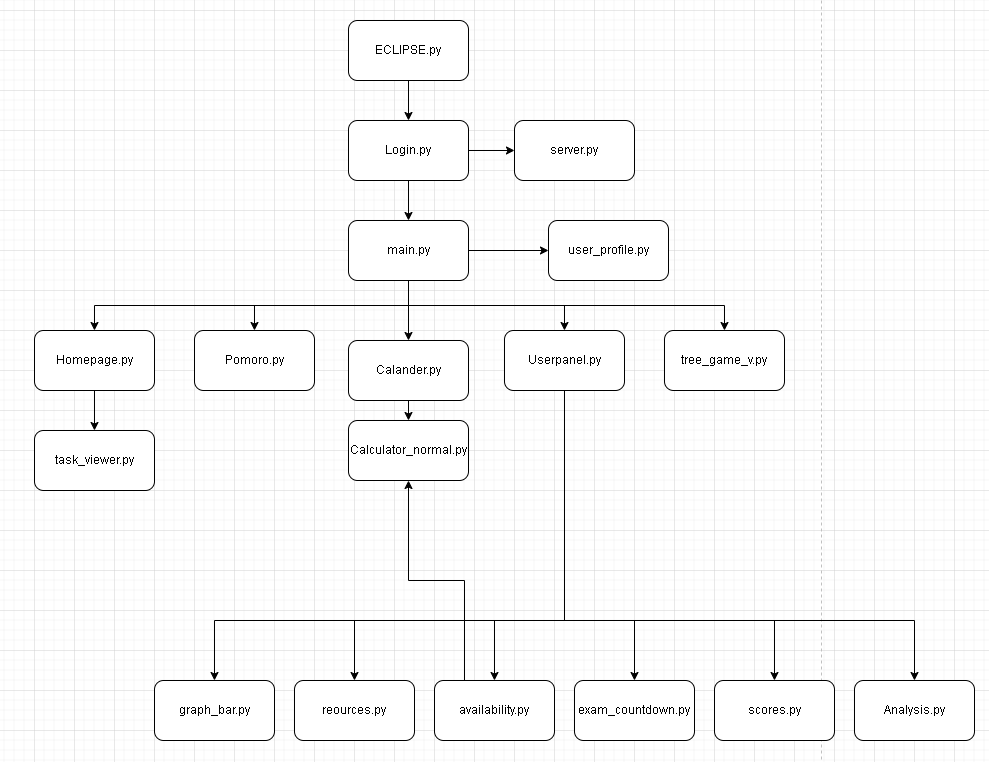
self.parent.delete\_exam(self.exam["exam\_date"], self.exam["subject"])

if self.parent.save\_exam\_details(self.cal.get\_date(), new\_subject):

self.destroy()

except ValueError:

messagebox.showerror("Error", "Invalid date format!")

Program flow - 

## Technical used

### Linear search -

I have used linear search for this part of the code as the user will enter how they feel, and their linear search will go through each feeling and get the data. I used linear search due to the data set being small and would be more efficient than other algorithm due to the fact thast the data set is small . What the the code does for the linear search to be needed is that , the user will enter their feeling such as tired , unfocused .confused , e.t.c and when such value is entered and after 2000 seconds the next cycle of the recursion repeats and then shows the result of the linear search and it will display the following .

def linear\_search\_feelings(self, user\_input):

try:

with open(self.feelings\_file, 'r') as f:

feelings\_dt = json.load(f)

user\_words = user\_input.lower().split()

recom = []

for feeling in feelings\_dt:

if feeling.lower() in user\_words:

recom.extend(feelings\_dt[feeling])

return recom

except Exception as e:

return []

### Binary search -

I used a binary search to get the resources as there can be lots of subjects that can be stored and using a linear search it will take long to actually get such resources as it would be O(n) complexity, and the amount of subject can be in the thousands. That why I used binary search it half every time it searched and makes it better for this situation as it would take less time to get the subject as O(logn).

Summary -

The binary search gets resources from the json file reources.json where each subject is organized Alphabetically order the user enters the subject, they want resources for, the code gets the subject and the lowers the input text. Then goes through the resources when it gets such subject then it will display the reources .

def binary\_search\_match(self, subject):

subjects = sorted(self.resources.keys(), key=lambda s: s.lower())

low, high = 0, len(subjects) - 1

while low <= high:

mid = (low + high) // 2

mid\_subject = subjects[mid]

mid\_subject\_lower = mid\_subject.lower()

if mid\_subject\_lower == subject:

return mid\_subject

elif mid\_subject\_lower < subject:

low = mid + 1

else:

high = mid - 1

return None

### Tree traversal -

I used tree traversal to navigate the user through the question as if the user gets a right answer it will display the next set of question and not display the same question again but if it goes through the wrong answer then the question will be displayed after the user answered all the other questions. I used this system as it more convenient to know which is right and wrong answer without checking each time if the answer, they have choosen macth all the users must do is go to the desired box and then if its the right node then carries on with the game. I also used tree traversal to also split the question from easy to medium and hard so after the user has enough righ answers it will display the next set of questions makign the game more dynamic

This line of code handles the wrogn answer and the moves to the wrong node and then shows the user they have incorrect answer and then continouse the game . (This code moves to the wrong node )

# Tree traversal happens here for incorrect answers

# Move to the incorrect branch of the tree

next\_node = self.crnt\_node.children["incorrect"]

if next\_node:

self.debug\_log(f"Moving to next node in 'incorrect' path")

self.crnt\_node = next\_node

# Update path history for incorrect path

difficulty\_emoji = "🟢" if next\_node.difficulty == 1 else ("🟡" if next\_node.difficulty == 2 else "🔴")

self.path\_history.append(f"{difficulty\_emoji} Q{len(self.path\_history)}\*") # \* marks incorrect path

else:

# No incorrect path defined, stay on current node and retry

self.debug\_log("No 'incorrect' path defined - retrying current question")

self.allowed\_move = True # Re-enable player movement

self.debug\_log("Attack sequence complete, player movement re-enabled")

self.after(1000, self.continue\_game)

This does the same thing but it goes down to the correct node

next\_node = self.crnt\_node.children["correct"]

if next\_node:

self.debug\_log(f"Moving to next node in 'correct' path")

self.crnt\_node = next\_node

# Update path history

difficulty\_emoji = "🟢" if next\_node.difficulty == 1 else ("🟡" if next\_node.difficulty == 2 else "🔴")

self.path\_history.append(f"{difficulty\_emoji} Q{len(self.path\_history)}")

else:

# No more nodes in the tree, player wins!

self.debug\_log("No more nodes in the 'correct' path - Victory!")

self.crnt\_node = None

This function makes the question tree for the game first it get 15 questions from the API and then checks if there is difficulties on the questions and then it check if thir is 5 question . Then the function builds the tree structure using the first easy question as the root node , then creating a correct path that progressws through easy then medium then a hard question adn does the sdame with the incorrect answer but inseatd of goign throug dificuty it will use the same section(dificulty) to for teh next question instead of goign down .

If the user gets answers correct it will go through the main path and gradually increase the difficulty adn when it gets wrong they go to another path

def make\_qst\_tree(self, subject):

#Create a tree of questions

try:

self.debug\_log(f"Creating question tree for subject: {subject}")

self.loading\_labl.config(text="Building question tree...")

# Get questions from API or fallback

questions = self.get\_qst\_api(subject, count=15)

# Group questions by difficulty (if available from API), or assign difficulties

easy\_questions = [q for q in questions if q.difficulty <= 1]

medium\_questions = [q for q in questions if q.difficulty == 2]

hard\_questions = [q for q in questions if q.difficulty >= 3]

# redistrubution if not enough question

if len(easy\_questions) < 5:

random.shuffle(questions)

easy\_questions = questions[:5]

for q in easy\_questions:

q.difficulty = 1

if len(medium\_questions) < 5:

remaining = [q for q in questions if q not in easy\_questions]

if len(remaining) >= 5:

medium\_questions = remaining[:5]

else:

medium\_questions = remaining + random.sample(easy\_questions, 5 - len(remaining))

for q in medium\_questions:

q.difficulty = 2

if len(hard\_questions) < 5:

used = easy\_questions + medium\_questions

remaining = [q for q in questions if q not in used]

if len(remaining) >= 5:

hard\_questions = remaining[:5]

else:

hard\_questions = remaining + random.sample(medium\_questions, 5 - len(remaining))

for q in hard\_questions:

q.difficulty = 3

#5 question limit

easy\_questions = easy\_questions[:5]

medium\_questions = medium\_questions[:5]

hard\_questions = hard\_questions[:5]

# Create question adn levels

root = easy\_questions[0]

self.debug\_log("Building question tree structure...")

# Track nodes

added\_nodes = {root}

# filter question

current\_correct = root

for i in range(1, len(easy\_questions)):

if easy\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(easy\_questions[i], "correct")

added\_nodes.add(easy\_questions[i])

if medium\_questions and medium\_questions[0] not in added\_nodes:

current\_correct = current\_correct.add\_child(medium\_questions[0], "correct")

added\_nodes.add(medium\_questions[0])

for i in range(1, len(medium\_questions)):

if medium\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(medium\_questions[i], "correct")

added\_nodes.add(medium\_questions[i])

if hard\_questions and hard\_questions[0] not in added\_nodes:

current\_correct = current\_correct.add\_child(hard\_questions[0], "correct")

added\_nodes.add(hard\_questions[0])

for i in range(1, len(hard\_questions)):

if hard\_questions[i] not in added\_nodes:

current\_correct = current\_correct.add\_child(hard\_questions[i], "correct")

added\_nodes.add(hard\_questions[i])

# create the incorrect paths

nodes\_process = [root]

while nodes\_process:

node = nodes\_process.pop(0)

if node.children["correct"]:

nodes\_process.append(node.children["correct"])

if node.difficulty == 1:

avl = [q for q in easy\_questions if q not in added\_nodes]

if not avl:

node.children["incorrect"] = root

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

elif node.difficulty == 2:

avl = [q for q in easy\_questions + medium\_questions[:2] if q not in added\_nodes]

if not avl:

node.children["incorrect"] = medium\_questions[0] if medium\_questions else easy\_questions[0]

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

elif node.difficulty == 3:

avl = [q for q in medium\_questions if q not in added\_nodes]

if not avl:

node.children["incorrect"] = medium\_questions[0] if medium\_questions else root

else:

incorrect\_node = random.choice(avl)

node.children["incorrect"] = incorrect\_node

added\_nodes.add(incorrect\_node)

nodes\_process.append(incorrect\_node)

self.loading\_labl.config(text="")

self.debug\_log("Question tree built successfully")

return root

except Exception as e:

self.debug\_log(f"Error creating question tree: {str(e)}")

# Create a simple fallback tree

return self.make\_backup\_tree(subject)

### Calling parameterised Web service APIs -

I used an API for the game as I cannot make questions for all the subjects as there is loads of subjects to account for and is easier to get it from the website then making my own questions. We use the request libary to request from the website multiple questions and get 15 questions for the selected subject. To get a working Api we need a website that gives the api for the user, I used the open trivia database website for the API as it has loads of question on the different subjects and has an open api. After gettting the question for the API, the program gos through and makes the tree taking in variables like the possible answers and also the correct answers. I also made sure that if the API doesn’t work then we have fallback question just in case the network fails

<https://opentdb.com/>

#Fetch questions from API

self.loading\_labl.config(text="Loading questions...")

try:

api\_url = "<https://opentdb.com/api.php>"

# Map the subjects to API categories

category\_map = {

"Math": 19,

"Geography": 22,

"Science": 17,

"History": 23,

"Literature": 10,

"Mixed": None

}

# Prepare parameters

params = {

"amount": count,

"type": "multiple" # Multiple choice questions

}

# Add category if not Mixed

if subject != "Mixed" and category\_map[subject]:

params["category"] = category\_map[subject]

# Make API request

self.debug\_log(f"Fetching questions for subject: {subject}")

response = requests.get(api\_url, params=params)

data = response.json()

# Check if we got valid response

if response.status\_code == 200 and data["response\_code"] == 0:

questions = []

for i, item in enumerate(data["results"]):

# Get all answers and shuffle them

all\_answers = [item["correct\_answer"]] + item["incorrect\_answers"]

random.shuffle(all\_answers)

# Find the index of the correct answer

correct\_index = all\_answers.index(item["correct\_answer"])

# Assign difficulty based on position in results (first 5 easy, next 5 medium, last 5 hard)

difficulty = 1 if i < 5 else (2 if i < 10 else 3)

# Create TreeNode

node = TreeNode(

question=item["question"],

answers=all\_answers,

correct\_answer=correct\_index,

platform=item["category"],

difficulty=difficulty

)

questions.append(node)

self.loading\_labl.config(text="")

return questions

else:

# API error - use fallback questions

self.debug\_log(f"API Error: {data.get('response\_code', 'Unknown')}")

self.loading\_labl.config(text="Could not load questions. Using defaults.")

return self.get\_backup\_qsts(subject)

except Exception as e:

# Handle any errors (network issues, etc.)

self.debug\_log(f"Error fetching questions: {str(e)}")

self.loading\_labl.config(text="Network error. Using default questions.")

return self.get\_backup\_qsts(subject)

### Server-Side scipting-

Server-side scripting is where the program handles the request to go to the website before hand . In my program I have used this in 2 places one in resources where the user ask for resource and then makes a button with the correspondign website and then the other in the struggling function in analyze.py. I take the url from the json files and then transfer it to the button correspondingly and when the user presses the button it will open teh browser on their computer and go to the selected website .

This is the json file which i have store the title and the websites url (I do not own these websites )

Resources.json

{

"Math": [

{

"title": "Algebra Basics",

"description": "A beginner-friendly guide to algebra.",

"link": "<https://www.khanacademy.org/math/algebra>"

},

{

"title": "Geometry Essentials",

"description": "Learn the fundamentals of geometry.",

"link": "<https://www.mathsisfun.com/geometry/index.html>"

}

],

"Science": [

{

"title": "Introduction to Physics",

"description": "Basic concepts of physics with examples.",

"link": "<https://www.physicsclassroom.com/>"

},

{

"title": "Biology 101",

"description": "Understanding life and living organisms.",

"link": "<https://www.biologyonline.com/>"

}

],

"History": [

{

"title": "World History Overview",

"description": "An overview of major historical events.",

"link": "<https://www.history.com/>"

},

{

"title": "Ancient Civilizations",

"description": "Learn about the great civilizations of the past.",

"link": "<https://www.ancient.eu/>"

}

],

"Programming": [

{

"title": "Python for Beginners",

"description": "A step-by-step guide to learning Python.",

"link": "<https://realpython.com/>"

},

{

"title": "JavaScript Essentials",

"description": "Learn JavaScript from scratch.",

"link": "<https://developer.mozilla.org/en-US/docs/Web/JavaScript>"

}

]

}

Learning\_method.json

{

"memory": {

"name": "Active Recall",

"description": "Science-backed method to improve memory retention",

"steps": [

"Create flashcards with questions on front/answers on back",

"Regularly test yourself without looking at notes",

"Use spaced repetition software like Anki"

],

"resource": "<https://www.learningscientists.org/retrieval-practice>"

},

"focus": {

"name": "Pomodoro Technique",

"description": "Time management method for sustained focus",

"steps": [

"Set timer for 25 minutes of focused work",

"Take 5-minute break after each session",

"After 2 sessions, take 15-30 minute break"

],

"resource": "<https://todoist.com/productivity-methods/pomodoro-technique>"

},

"motivation":{

"name": "Gamification",

"description": "Using game design elements to boost motivation",

"steps": [

"Set clear goals and rewards for achieving them",

"Track progress and celebrate milestones",

"Use leaderboards or social sharing to stay motivated"

],

"resource": "<https://www.gamification.co/what-is-gamification/>"

},

"understanding": {

"name": "Feynman Technique",

"description": "Learn by teaching concepts to others",

"steps": [

"Choose a concept to learn",

"Explain it in simple terms",

"Identify gaps in understanding",

"Review and simplify further"

],

"resource": "<https://collegeinfogeek.com/feynman-technique/>"

}

}

The program finds the subject in the json file and then it extracts the data from the json file display anythign that not the url code and makes the url code into a button and when its clicked on it will take the user to the website.

Example -

If the user press Math, then it will display alebra basics and then a button which will take us to the website

Resources.py in the function get\_resource()

for resource in resources:

frame = ctk.CTkFrame(self.resource\_frame, corner\_radius=5, border\_width=1)

frame.pack(pady=5, fill=tk.X)

title = ctk.CTkLabel(frame, text=resource["title"], font=("Arial", 12, "bold"))

title.pack(anchor="w", padx=5, pady=2)

description = ctk.CTkLabel(frame, text=resource["description"], wraplength=550)

description.pack(anchor="w", padx=5, pady=2)

link = ctk.CTkButton(frame, text="View Resource", command=lambda url=resource["link"]: webbrowser.open(url))

link.pack(anchor="w", padx=5, pady=5)

Same method has been used for the struggle reccomendation

Analyse.py in the function generate\_recom()

#Struggle-Specific Methods

struggle\_recs = self.get\_recom()

if struggle\_recs:

for rec\_text, resource in struggle\_recs:

self.method\_txt.insert(tk.END, rec\_text + "\n")

if resource:

btn = ttk.Button(self.resource\_button\_frm, text="Go to Resource",

command=lambda url=resource: webbrowser.open(url))

btn.pack(pady=2, anchor="w")

self.method\_txt.insert(tk.END, "---------------------\n")

else:

self.method\_txt.insert(tk.END, "No specific struggle selected.\n")

### Aggregate SQL functions -

I need to use aggregated sql to get specific data from teh database and the database can have a big data set what aggregates does it get the said amount of in some cases count the amount of data in that table .

In Flashcard\_v.py we use a simple COUNT which is count for aggregation . I use count to the code to get the amount of flashcard the user has in total in that

try:

self.cursor.execute("""

SELECT

COUNT(front)

FROM flashcards

WHERE user\_id = %s

""", (self.user\_id,))

stats\_data = self.cursor.fetchone()

# Default valus if nothing exists

total = stats\_data[0] if stats\_data else 0

In analyse.py I have used the LIMIT And DESC function to get the latest 3 scores the user has stored in the selected subject **update\_score function**

def update\_scores(self):

subj\_name = self.subj\_var.get()

subj\_id = next((k for k, v in self.subj\_map.items() if v == subj\_name), None)

if not subj\_id:

self.scores\_labl.configure(text="No scores available")

return

try:

conn = db()

cursor = conn.cursor(dictionary=True)

cursor.execute("""

SELECT score, month, year

FROM scores

WHERE subject\_id = %s

ORDER BY year DESC, month DESC

LIMIT 3

""", (subj\_id,))

In this part of the program in calculator\_normal.py the function uses the sql statment ROW\_NUMBER() to get the latest score for each subject which can be seen as aggregations THis is in the **calulator\_normal.py (subj\_sccoresfunction)**

cursor.execute(f"""

SELECT subjects.subject\_name, scores.month, scores.year, scores.score

FROM scores

JOIN subjects ON subjects.subject\_id = scores.subject\_id

WHERE subjects.subject\_id IN ({placeholders})

""", tuple(subj\_ids))

results = cursor.fetchall()

### Cross-table parameterised SQL

This is needed to use the data we got in other tables in the database for example in flashcard we have the user\_id and we use this to get the subject\_id and subjects name to display , and flashcard table store user\_id and subject\_id and so we use the subject\_id to select which subject to select adn user\_id for the specific user flashcard .

The same goes for teh graph bar but we using the subject\_id and joining it with the scores suibject\_id to get the score for that specific subject , and we doign this as the user cannot differentiate between the subject if their name is just 1,2 and 3 so we JOIN them to get the subject name , scores month ,year,score

Graph\_bar.py LOAD DATA FUNCTION

placeholders = ', '.join(['%s'] \* len(subj\_ids))

cursor.execute(f"""

SELECT subjects.subject\_name, scores.month, scores.year, scores.score

FROM scores

JOIN subjects ON subjects.subject\_id = scores.subject\_id

WHERE subjects.subject\_id IN ({placeholders})

""", tuple(subj\_ids))

Flashcard\_v.py in the function load\_flascards\_fromDB

def load\_flashcards\_fromDB(self):

try:

cursor = self.connection.cursor(buffered=True)

query = "SELECT flashcards.front AS word, flashcards.back AS meaning FROM subjects JOIN flashcards ON subjects.subject\_id = flashcards.subject\_id WHERE subjects.subject\_name = %s AND flashcards.user\_id=%s" # simple Datbase Model

self.cursor.execute(query, (self.subject,self.user\_id,))

results = self.cursor.fetchall()

Flashcard\_v,py in the function **amount\_flashcard()**

def amount\_flashcard(self):

def load\_flashcards():

selected\_sub = self.subject

try:

query = "SELECT DISTINCT flashcards.front FROM subjects JOIN flashcards On subjects.subject\_id=flashcards.subject\_id WHERE subjects.subject\_name=%s AND flashcards.user\_id=%s"

self.cursor.execute(query, (selected\_sub,self.user\_id))

results = self.cursor.fetchall()

### List Operations-

List operation is where the list can be append or change the list .

In Analyse.py in the function update\_scores the code takes the latest 3 scores and puts in a list and then comments it in order (this is just to show the user their scores for the latest 3 month)

if scores:

scores\_txt = ", ".join(f"{entry['score']}" for entry in scores)

self.scores\_labl.configure(text=scores\_txt)

In analyse.py in the function of plot\_scores() we convert the scores into an interger then it creates a list which squently store it is using range , and then this list are used for plotting the score progression graph .

if scores:

try:

score\_values = [int(entry['score']) for entry in scores]

attempts = list(range(1, len(score\_values) + 1))

Availability.py in load\_data\_avl() this splits the hours for every comma it appears into a list then the h.strips() it filters out empty value or items . This creates a clean list for the hour values

if results:

db\_avl = default\_avl.copy()

for day, hours\_str in results:

# Default to empty list (which means "0" in the UI)

hours\_list = []

# Only try to parse if hours is not NULL and not empty

if hours\_str:

try:

# Convert the comma-separated string back to a list of integers

hours\_list = [int(h) for h in hours\_str.split(',') if h.strip()]

In task \_viwer.py in the function mark\_deletion() teh function marks the task to either be deleted or remove it from the deleted\_task . In the program when the user goes to another window then the task gets deleted so it gets stored in the deleted\_tasks. The code check if the task is in the deleted\_tasks and if it is it removes it (this is when the user click again to uncomplete the task) or else it appends it to the list.

def mark\_deletion(self, task, checkbox):

# Marks a task for deletion or unmarks it.

if task in self.deleted\_tasks:

self.deleted\_tasks.remove(task)

checkbox.configure(fg\_color=("#3B8ED0", "#1F6AA5"))

else:

self.deleted\_tasks.append(task)

checkbox.configure(fg\_color="red")

### Files(s) organised for direct access

This json file saves the availability in a format that the user can see like 13-15 instead of 2 hours which will store in the database. The json file is used for also a backp availability just in case

Calculator\_normal.py in the function load\_json()

def load\_json(filename, default\_data):

if os.path.exists(filename):

with open(filename, "r") as file:

return json.load(file)

return default\_data

def save\_json(filename, data):

with open(filename, "w") as file:

json.dump(data, file, indent=4)

Calculator\_normal.py in the function get\_user\_avl()

def get\_user\_avl(user\_id):

#Load availability data

default\_aval = {day: [] for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]}

try:

# First try to load from user-specific JSON file

if not os.path.exists("availability\_data"):

os.makedirs("availability\_data")

filename = f"availability\_data/availability\_{user\_id}.json"

if os.path.exists(filename) and os.path.getsize(filename) > 0:

try:

with open(filename, "r") as file:

data = json.load(file)

if isinstance(data, dict) and all(day in data for day in ["Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday", "Sunday"]):

return data

except json.JSONDecodeError:

print(f"Warning: Invalid JSON in {filename}")

This code load the quotes.json to get the quotes and then stores to the last\_mesage.json so the specific quotes will be repeated during the day

Quotes.py in the function load\_quotes()

def load\_quotes():

with open("quotes.json", "r") as file:

data = json.load(file)

return data["quotes"], data["tips"]

# Select a quote or tip for the day

def daily\_msg():

tdy = datetime.today().strftime("%Y-%m-%d")

try:

with open("last\_message.json", "r") as file:

last\_data = json.load(file)

if last\_data["date"] == tdy:

return last\_data["message"]

except FileNotFoundError:

Pass

I stored the resources in a different files to make it easier to add more resources which makes it easy updates .

Resources.py in the function resources\_loader()

def resources\_loader(self):

try:

with open("resources.json", "r") as file:

self.resources = json.load(file)

except FileNotFoundError:

messagebox.showerror("Error", "The file 'resources.json' was not found.")

except json.JSONDecodeError:

messagebox.showerror("Error", "Error decoding 'resources.json'. Please check the file format.")

except Exception as e:

messagebox.showerror("Error", f"Failed to load resources: {e}")

This store the exam detail in structure dates and subject(title)

Examcountdow.py in the function save\_details()

def save\_details(self, date, subject):

try:

formatted\_date = datetime.strptime(date, "%m/%d/%y").strftime("%d-%m-%Y")

data = self.load\_details()

data.append({"exam\_date": formatted\_date, "subject": subject})

with open(self.FILE\_PATH, "w") as file:

json.dump(data, file, indent=4)

self.update\_countdown()

return True

except Exception as e:

messagebox.showerror("Error", f"Error saving exam: {str(e)}")

return False

def load\_details(self):

if os.path.exists(self.FILE\_PATH):

with open(self.FILE\_PATH, "r") as file:

data = json.load(file)

return data if isinstance(data, list) else []

return []

### Complex scientific/mathematical/robotics/control/business model

The program takes in a sequences of students' scores, then caclulates the percentage change between teh first and the most reecent scores to create a trend , and then gets the avarage rate of change to then predict the next score (imporvcemtn percentage ) which then is displayed for the user to see what path they are working on and what it may lead to if they keep studying makign them more confident .

def update\_trend(self, scores):

if scores and len(scores) >= 2:

try:

score\_values = [int(entry['score']) for entry in scores]

first, last = score\_values[0], score\_values[-1]

current\_improve = ((last - first) / first) \* 100 if first != 0 else 0

intervals = len(score\_values) - 1

avg\_increase = ((last - first) / intervals) if intervals > 0 else 0

predicted\_improve = (avg\_increase / first) \* 100 if first != 0 else 0

self.current\_improve\_label.configure(text=f"Current Trend: {current\_improve:.2f}%")

self.predicted\_improve\_label.configure(text=f"Predicted Next Increase: {predicted\_improve:.2f}%")

except Exception as ex:

self.current\_improve\_label.configure(text="Current Trend: N/A")

self.predicted\_improve\_label.configure(text="Predicted Next Increase: N/A")

print(f"Error computing trend: {ex}")

else:

self.current\_improve\_label.configure(text="Current Trend: N/A")

self.predicted\_improve\_label.configure(text="Predicted Next Increase: N/A")

### Complex User-defined algorithm

The program is the main timetable maker where it takes the user availability data then get the subject performance scores and inverts them so if math has 30 score it will have 70 weight making more prioritized then normalizes the weights to distribute total available study time then creates a minimal study time per subject then the weakest subject will be allocated more time and then the program will randomize the sybjct allocation so it's not algorithmic for the user .

Calularo\_normal.py in the function tm\_allocation()

def tm\_allocation(user\_id):

print(f"Welcome to the Study Time Allocation Program for user {user\_id}!")

# Get availability and subjects

daily\_hours = get\_user\_avl(user\_id)

subjects = subj\_scores(user\_id)

total\_time = sum(len(hours) for hours in daily\_hours.values())

print(f"\nTotal weekly study time: {total\_time} hours")

# Check if we have any subjects or available hours

if not subjects:

print("Error: No subjects found. Please add subjects first.")

return False

if total\_time == 0:

print("Error: No available hours found. Please set your availability first.")

return False

# Use the stored weight directly

adjusted\_weights = {subject: score for subject, score in subjects.items()}

total\_weight = sum(adjusted\_weights.values())

if total\_weight == 0:

print("Warning: Total weight is 0, using equal weights for all subjects")

adjusted\_weights = {subject: 1 for subject in subjects}

total\_weight = len(subjects)

weighted\_allocation = {

subject: max(round((weight / total\_weight) \* total\_time), 1)

for subject, weight in adjusted\_weights.items()

}

allocation\_sum = sum(weighted\_allocation.values())

if allocation\_sum != total\_time and weighted\_allocation: # Make sure weighted\_allocation is not empty

difference = total\_time - allocation\_sum

# Safely get the weakest subject or just use the first subject if they're all equal

try:

weakest\_subject = min(weighted\_allocation, key=weighted\_allocation.get)

weighted\_allocation[weakest\_subject] += difference

except ValueError: # In case weighted\_allocation is empty

print("Warning: Could not adjust allocation, using equal allocation")

if subjects:

equal\_time = total\_time // len(subjects)

weighted\_allocation = {subject: equal\_time for subject in subjects}

# Print allocation for debugging

print("\nTime allocation per subject:")

for subject, hours in weighted\_allocation.items():

print(f"{subject}: {hours} hours")

start\_date = datetime.today()

start\_week = start\_date - timedelta(days=start\_date.weekday())

dly\_subject\_allocation = []

remaining\_hours = weighted\_allocation.copy()

# Map day index to day name

day\_map = {

0: "Monday", 1: "Tuesday", 2: "Wednesday",

3: "Thursday", 4: "Friday", 5: "Saturday", 6: "Sunday"

}

for i in range(7): # Iterate through all 7 days

day\_name = day\_map[i]

hours = daily\_hours.get(day\_name, [])

if not hours:

continue

date = (start\_week + timedelta(days=i)).strftime('%Y-%m-%d')

slot\_index = 0

available\_subjects = sorted(remaining\_hours.keys(), key=lambda x: remaining\_hours[x], reverse=True)

while slot\_index < len(hours) and available\_subjects:

for subject in available\_subjects:

if remaining\_hours[subject] > 0:

start\_time = hours[slot\_index]

end\_time = start\_time + 1

time\_str = f"{start\_time:02}:00:00"

dly\_subject\_allocation.append((date, time\_str, subject))

remaining\_hours[subject] -= 1

if (slot\_index + 1 < len(hours) and remaining\_hours[subject] > 0 and

hours[slot\_index + 1] == end\_time and random.random() < 0.5):

start\_time = hours[slot\_index + 1]

end\_time = start\_time + 1

time\_str = f"{start\_time:02}:00:00"

dly\_subject\_allocation.append((date, time\_str, subject))

remaining\_hours[subject] -= 1

slot\_index += 1

slot\_index += 1

break

# Update available subjects list

available\_subjects = [s for s in available\_subjects if remaining\_hours[s] > 0]

if slot\_index >= len(hours) or not available\_subjects:

break

### Bubble sort

The bubble sort is used to sort the student score data by month and year within each subject as the score can be entered at random times and not sorting them can make the score appear at weird times in the graph\_bar section , and we need them to be organized so the user can see their score choronological order .

Bubble sort is a complexity which is good for a small data structure . But after runnign the code its not the most efficient as the program get all the scores then sorts them and if the user store in all the month in the years it can quickly become a large data set to store . But for now this is sufficient for the code

Graph\_bar.py in teh function bubble\_sort()

def bubble\_sort(data): # Bubble sort algorithm

sorted\_data = {}

for subject, scores in data.items():

sorted\_scores = scores.copy()

n = len(sorted\_scores)

for i in range(n):

for j in range(0, n-i-1):

if sorted\_scores[j]['month'] > sorted\_scores[j+1]['month']:

sorted\_scores[j], sorted\_scores[j+1] = sorted\_scores[j+1], sorted\_scores[j]

sorted\_data[subject] = sorted\_scores

return sorted\_data

### Classes

I used classes to connect the different windows into a singualr frame without keep on making a frame when it opened I made the main.py as the parent class and every other window such as homepage.py ,flashcard\_v.py ,calender.py . All of the files are child class to main.py

Graph\_bar.py

class ScoreVisualizer(tk.Frame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs) # Initialize the Tkinter root window

# Store user\_id as an integer and make sure it's a non-tuple parameter for SQL

self.user\_id = int(user\_id) if isinstance(user\_id, (int, str)) else user\_id[0]

# Store callback references to prevent garbage collection

self.callbacks = []

Calendar.py

class CustomCalendar(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

calculate\_time\_allocation(user\_id)

self.user\_id = user\_id

# Set overall appearance

self.configure(fg\_color="#1E1E2E") # Dark background color

# Create header with nice title

self.header\_frame = ctk.CTkFrame(self, fg\_color="#1E1E2E", corner\_radius=0)

self.header\_frame.pack(fill="x", padx=20, pady=(20, 10))

Flashcar.py

class FlashCardMainApp(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

# Extract user\_id properly

if isinstance(user\_id, tuple):

self.user\_id = user\_id[0]

else:

self.user\_id = user\_id

print(self.user\_id)

# Set dark theme appearance

self.configure(fg\_color="#121212") # Dark background

self.main\_menu()

e.t.c

### Inheritance -

Inherits is used after the login.py as explained before the main.py is the parent class and all other files are the child class which inherits the attribute of the frame from the main.py

class FlashCardMainApp(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

class CustomCalendar(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

class ScoreManager(ctk.CTkFrame): # inheritance

def \_\_init\_\_(self, parent,user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

self.user\_id=user\_id

self.subj = self.get\_subj(user\_id)

self.ui()

e.t.c

### Composition-

. Composition is a oop techinque with a has-a relationship and userpanel fits that description as the userpanel has availablityEditor , scorecvisuallizer , quotes, resourceslibary,study advisor and scoreManager . The userpanel is made out of these other program and each program doesn’t inherit behaviours it just contains them and manages them .

Userpanel.py in teh function main\_sections()

# 1. (LEFT COLUMN)

left\_sect = self.section\_frame(0, "Resources & Deadlines", padding)

self.resources(left\_sect)

self.exam\_countdown(left\_sect)

# 2. (CENTER COLUMN)

center\_sect = self.section\_frame(1, "Progress & Performance", padding)

self.Run\_quotes(center\_sect)

self.Graph(center\_sect)

# 3. (RIGHT COLUMN)

right\_sect = self.section\_frame(2, "Schedule & Scores", padding)

self.availability(right\_sect)

self.score\_manager(right\_sect)

self.analysis(right\_sect)

Resources()

def resources(self, parent\_frame):

resources\_frm = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

resources\_frm.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

resource\_ttl\_frm = ctk.CTkFrame(resources\_frm, fg\_color="transparent", height=30)

resource\_ttl\_frm.pack(fill="x", padx=15, pady=(10, 0))

resource\_ttl = ctk.CTkLabel(resource\_ttl\_frm, text="📚 Learning Resources", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

resource\_ttl.pack(anchor="w")

resource\_cont = ctk.CTkFrame(resources\_frm, fg\_color="#2D3748")

resource\_cont.pack(fill="both", expand=True, padx=15, pady=10)

self.resource\_lib = ResourceLibrary(resource\_cont)

self.resource\_lib.pack(fill="both", expand=True)

Exam\_countdown()

def exam\_countdown(self, parent\_frame):

sched\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

sched\_frame.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Exam schedule title

sched\_ttl\_frame = ctk.CTkFrame(sched\_frame, fg\_color="transparent", height=30)

sched\_ttl\_frame.pack(fill="x", padx=15, pady=(10, 0))

sched\_title = ctk.CTkLabel(sched\_ttl\_frame, text="📅 Exam Schedule", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

sched\_title.pack(anchor="w")

# Initialize Exam Countdown - DO NOT set fg\_color here

exam\_container = ctk.CTkFrame(sched\_frame, fg\_color="#2D3748")

exam\_container.pack(fill="both", expand=True, padx=15, pady=10)

self.countdown = ExamCountdown(exam\_container)

self.countdown.pack(fill="both", expand=True)

Quote ()

def Run\_quotes(self, parent\_frame):

quote = ctk.CTkFrame(parent\_frame, fg\_color="#8B5CF6", corner\_radius=8)

quote.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Initialize Motivation Quote directly in the purple frame

self.quotes = MotivationQuote(quote)

# Important: Don't change its fg\_color, which could cause rendering issues

self.quotes.pack(fill="both", expand=True, padx=10, pady=10)

Graph()

def Graph(self, parent\_frame):

performa = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

performa.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Performance title

title\_frame = ctk.CTkFrame(performa, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="📊 Performance Metrics", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

self.progress = ctk.CTkFrame(performa, fg\_color="#2D3748", height=400)

self.progress.pack(fill="both", expand=True, padx=15, pady=10)

# Important: Don't override ScoreVisualizer's internal styling

self.graph = ScoreVisualizer(self.progress, self.user\_id)

self.graph.pack(fill="both", expand=True)

Availability()

def availability(self, parent\_frame):

# Container for the first component (Availability)

avail\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

avail\_frame.grid(row=1, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Availability title

title\_frame = ctk.CTkFrame(avail\_frame, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="⏰ Weekly Availability", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

# Initialize Availability Editor - Important: give it enough height

avail\_container = ctk.CTkFrame(avail\_frame, fg\_color="#2D3748", height=350)

avail\_container.pack(fill="both", expand=True, padx=15, pady=10)

# Create the availability editor without changing its fg\_color

availability = AvailabilityEditor(avail\_container, self.user\_id)

availability.pack(fill="both", expand=True)

Score\_manager()

def score\_manager(self, parent\_frame):

# Container for the second component (Score Manager)

score\_frame = ctk.CTkFrame(parent\_frame, fg\_color="#1F2A3C", corner\_radius=8)

score\_frame.grid(row=2, column=0, sticky="nsew", padx=0, pady=(0, 8))

# Score manager title

title\_frame = ctk.CTkFrame(score\_frame, fg\_color="transparent", height=30)

title\_frame.pack(fill="x", padx=15, pady=(10, 0))

title = ctk.CTkLabel(title\_frame, text="🎯 Score Manager", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(anchor="w")

container = ctk.CTkFrame(score\_frame, fg\_color="#2D3748", height=300)

container.pack(fill="both", expand=True, padx=15, pady=10)

# Create score manager without changing its fg\_color

score\_manager = ScoreManager(container, self.user\_id)

score\_manager.pack(fill="both", expand=True)

Run\_analysis()

def run\_analysis(self):

for widget in self.grid\_slaves():

if int(widget.grid\_info()["column"]) == 1 and int(widget.grid\_info()["row"]) == 1:

widget.destroy()

break

# create analysis ui

analysis\_sect = ctk.CTkFrame(self, fg\_color="transparent", corner\_radius=0)

analysis\_sect.grid(row=1, column=1, sticky="nsew", padx=8, pady=8)

title\_frm = ctk.CTkFrame(analysis\_sect, fg\_color="#1F2A3C", corner\_radius=8, height=36)

title\_frm.pack(fill="x", pady=(0, 8))

title = ctk.CTkLabel(title\_frm, text="Advanced Analytics", font=("Arial", 14, "bold"),text\_color="#FFFFFF")

title.pack(pady=8)

content\_frame = ctk.CTkFrame(analysis\_sect, fg\_color="#1F2A3C", corner\_radius=8, height=600)

content\_frame.pack(fill="both", expand=True, pady=(0, 8))

self.analysis = StudyAdvisorApp(content\_frame,self.user\_id)

self.analysis.pack(fill="both", expand=True, padx=15, pady=15)

# Return button

btn\_frame = ctk.CTkFrame(analysis\_sect, fg\_color="transparent")

btn\_frame.pack(fill="x")

return\_btn = ctk.CTkButton(btn\_frame,text="← Return to Dashboard",command=self.exit,fg\_color="#1F2A3C",hover\_color="#324056",font=("Arial", 14),corner\_radius=6,height=40)

return\_btn.pack(fill="x")

### Polymorphism-

Is a technique were depending on the situation the oject will change appropriate to the situation and in the code this i sused in flashcard\_v.py wherer in the class FlashCardMainApp the user selects the desired subject and then the class FlashcardApp is called to handle that specific subject showing and adding to that selected subject .

Polymorphism is used in this function where first it gets the subjects from the database, then it creates the subject button, and it calls open\_subj\_flashcard with the specific subject which turn calls flash\_card which then in the class FlashcardApp is created with the selected subject and get all the specific subject flashcards .

def flashcard\_selection(self):

def get\_select\_subj\_db():

#Fetch subjects from dp

try:

self.cursor.execute("SELECT DISTINCT subject\_name FROM subjects WHERE user\_id=%s", (self.user\_id,))

results = self.cursor.fetchall()

return [row[0] for row in results] # Extract the subject names

except mysql.connector.Error as e:

messagebox.showerror("Error", f"Database query failed: {e}")

return []

def open\_subj\_flashcard(subject):

#Open the subject

self.flash\_card(subject)

def create\_flashcards():

#unique subject button

unique\_subjects = get\_select\_subj\_db()

if not unique\_subjects:

# Create an empty state display

empty\_frm = ctk.CTkFrame(self.button\_frm, fg\_color="#121212")

empty\_frm.pack(fill="both", expand=True, pady=50)

empty\_labl = ctk.CTkLabel(empty\_frm, text="No subjects available yet", font=("Arial", 18),text\_color="#AAAAAA")

empty\_labl.pack(pady=10)

add\_btn = ctk.CTkButton(empty\_frm,text="+ Add Your First Subject",font=("Arial", 14),fg\_color="#8A2BE2",hover\_color="#9370DB",corner\_radius=10,height=40)

add\_btn.pack(pady=10)

return

# Clear existing buttons before adding new ones

for widget in self.button\_frm.winfo\_children():

widget.destroy()

# Add our subjects as cards

subj\_card\_frm = ctk.CTkFrame(self.button\_frm, fg\_color="#121212")

subj\_card\_frm.pack(fill="both", expand=True)

# Configure grid with 3 columns

subj\_card\_frm.grid\_columnconfigure(0, weight=1)

subj\_card\_frm.grid\_columnconfigure(1, weight=1)

subj\_card\_frm.grid\_columnconfigure(2, weight=1)

# Add subject cards in a grid

for i, subject in enumerate(unique\_subjects):

row, col = divmod(i, 3) # 3 columns

subject\_card = ctk.CTkButton(subj\_card\_frm,text=subject,font=("Arial", 16, "bold"),fg\_color="#1E1E1E",hover\_color="#2A2A2A",corner\_radius=10,height=150,width=280,command=lambda s=subject: open\_subj\_flashcard(s))

subject\_card.grid(row=row, column=col, padx=10, pady=10, sticky="nsew")

# Create the flashcards

create\_flashcards()

### Dynamic Object generation

Dynamic object generation is when an object is created depending on the user inputs or is made during the run time of the code this description macthes the events creation in calendar.py where the user has a stored evetns and the program generates it when its runed. All users have diffetnet evetns which is why it has to be made after the login page.

This function gets the events

def get\_event(self):

# Fetch events from the db

start\_date = self.strt\_week.date()

end\_date = start\_date + datetime.timedelta(days=6)

self.cursor.execute("""

SELECT date, time, title FROM events WHERE date BETWEEN %s AND %s AND user\_id = %s

""", (start\_date, end\_date, self.user\_id))

rows = self.cursor.fetchall()

self.events = {}

# dynamic obhject genration

for date, time, title in rows:

try:

dt = datetime.datetime.strptime(time, "%H:%M:%S")

formatted\_time = dt.strftime("%I:%M %p")

if formatted\_time in self.hours:

time\_index = self.hours.index(formatted\_time)

day\_offset = (datetime.datetime.strptime(date, "%Y-%m-%d").date() - start\_date).days

self.events[(time\_index, day\_offset)] = title

except ValueError as e:

print(f"Error processing event with time '{time}': {e}")

And this iteration creates the events in the calendar first it makes the cells and if there is an event makes the event. The event from the database is store in a list called self.events which then get through the function show\_event() and makes the event

# loops itself until all the necessary cells are made (this is the cells / events that the user can add)

for row in range(1, len(self.hours) + 1):

for col in range(len(self.days)):

today\_cell = col == self.tdy\_day and (self.strt\_week + datetime.timedelta(days=col)).date() == self.tdy\_date

event\_frm = ctk.CTkFrame(master=self.calendar\_frm,corner\_radius=6,fg\_color="#2E2E3E",border\_width=1,border\_color="#3E3E4E")

event\_frm.grid(row=row, column=col + 1, padx=4, pady=4, sticky="nsew")

if today\_cell:

indicator = ctk.CTkFrame(event\_frm,width=4,corner\_radius=2,fg\_color="#7C3AED" )

indicator.place(x=0, y=0, relheight=1)

# binding mouse input

event\_frm.bind("<Button-1>", lambda event, r=row - 1, c=col: self.add\_edit\_evnt(r, c))

self.event\_cell[(row - 1, col)] = event\_frm

event = self.events.get((row - 1, col))

if event:

self.show\_event(row - 1, col, event)

### Simple mathematical calculation -

In the examcountdow.py i used a simple minus calculation to get the remainign days to the exam

def update\_countdown(self):

exams = self.load\_details()

today = datetime.today()

countdown\_text = "Upcoming Exams:\n\n"

for exam in sorted(exams, key=lambda x: datetime.strptime(x["exam\_date"], "%d-%m-%Y")):

exam\_day = datetime.strptime(exam["exam\_date"], "%d-%m-%Y")

days\_left = (exam\_day - today).days

And in analyse.py I used the calculation to get the avarage to see if the user is having a high performance , a mid performance or low performance .

def get\_performance\_recom(self, scores):

if not scores:

return []

try: # basic mathematical calculation

score\_values = [int(entry['score']) for entry in scores]

avg = sum(score\_values) / len(score\_values)

### Singel dimensional array -

I used this array in calander.py to make the Y axis which is the hours that he user can see

self.hours = [

"07:00 AM", "08:00 AM", "09:00 AM", "10:00 AM", "11:00 AM",

"12:00 PM", "01:00 PM", "02:00 PM", "03:00 PM", "04:00 PM",

"05:00 PM", "06:00 PM", "07:00 PM", "08:00 PM", "09:00 PM",

"10:00 PM", "11:00 PM", "12:00 AM"

]

And is also used in flashcard\_v.py to store the flashcards

self.flashcards = []

### File handeling -

In examcoundow.py I have used file handelign to save the exam details so when the program is called it can be used for later

def save\_details(self, date, subject):

try:

formatted\_date = datetime.strptime(date, "%m/%d/%y").strftime("%d-%m-%Y")

data = self.load\_details()

data.append({"exam\_date": formatted\_date, "subject": subject})

with open(self.FILE\_PATH, "w") as file:

json.dump(data, file, indent=4)

self.update\_countdown()

return True

except Exception as e:

messagebox.showerror("Error", f"Error saving exam: {str(e)}")

return False

I also have a function to load the details

def load\_details(self):

if os.path.exists(self.FILE\_PATH):

with open(self.FILE\_PATH, "r") as file:

data = json.load(file)

return data if isinstance(data, list) else []

return []

### Multi-dimensional Array

Multiple dimensinal array is where the array stores both x and y position and in thsi case the program gets the event then stores it with eht corresponding date and time (column and row)

self.event\_cells = {} # Store event cells as { (date, hour): cell } (Multi-Dimensional Arrays)

### Dictionary -

A dictionary is a set of key-value pairs which mean if the user enters a key and it macthes that key should have a value in in this example in resources.py the program opens resouces.json where the subject such as math will have the value of the webiste

def resources\_loader(self):

try:

with open("resources.json", "r") as file:

self.resources = json.load(file)

except FileNotFoundError:

messagebox.showerror("Error", "The file 'resources.json' was not fou

**Resources.json (half)**

{

"Math": [

{

"title": "Algebra Basics",

"description": "A beginner-friendly guide to algebra.",

"link": "<https://www.khanacademy.org/math/algebra>"

},

{

"title": "Geometry Essentials",

"description": "Learn the fundamentals of geometry.",

"link": "<https://www.mathsisfun.com/geometry/index.html>"

}

],

"Science": [

{

"title": "Introduction to Physics",

"description": "Basic concepts of physics with examples.",

"link": "<https://www.physicsclassroom.com/>"

},

{

"title": "Biology 101",

"description": "Understanding life and living organisms.",

"link": "<https://www.biologyonline.com/>"

}

],

### Simple datbase model -

This query shows 2 interlinked tables

def load\_flashcards\_fromDB(self):

try:

cursor = self.connection.cursor(buffered=True)

query = "SELECT flashcards.front AS word, flashcards.back AS meaning FROM subjects JOIN flashcards ON subjects.subject\_id = flashcards.subject\_id WHERE subjects.subject\_name = %s AND flashcards.user\_id=%s" # simple Datbase Model

### Simple OOP model -

class FlashCardMainApp(ctk.CTkFrame):

def \_\_init\_\_(self, parent, user\_id, \*args, \*\*kwargs):

super().\_\_init\_\_(parent, \*args, \*\*kwargs)

# Extract user\_id properly

if isinstance(user\_id, tuple):

self.user\_id = user\_id[0] # Extract first element from tuple

else:

self.user\_id = user\_id # Simple attributes

# Set dark theme appearance

self.configure(fg\_color="#121212") # Dark background

self.main\_menu()

### Simple Object generation of object

I used a simple object generation of object where if the usr press the start button it creates these controls

def timer\_controller():

# Clear existing buttons if they exist

for widget in button\_frm.winfo\_children():

widget.destroy()

# Pause button

pause\_btn = ctk.CTkButton(button\_frm,text="Pause",font=("Arial", 16, "bold"),fg\_color="#555555",text\_color="white",hover\_color="#444444",corner\_radius=8,width=100,height=45,command=pause\_timer)

pause\_btn.pack(side="left", padx=5)

# Reset button

reset\_btn = ctk.CTkButton(button\_frm,text="Reset",font=("Arial", 16, "bold"),fg\_color="#E45858",text\_color="white",hover\_color="#C0392B",corner\_radius=8,width=100,height=45,command=reset\_timer)

reset\_btn.pack(side="left", padx=5)

# Make pause button accessible to other functions

return pause\_btn

### Simple user defined -

In availability

updated\_data = {}

for day, entry in self.txt\_entries.items():

text = entry.get().strip()

times = []

# Check for "0" entry which explicitly indicates no availability

if text == "0":

# User explicitly says not available on this day

times = [] # Empty list indicates no availability

elif text:

matches = re.findall(r"(\d{1,2})-(\d{1,2})", text)

for start, end in matches:

start, end = int(start), int(end)

if start < end:

times.extend(range(start, end))

updated\_data[day] = times

self.save\_data(updated\_data)

### Non-SQL Table access

In examcountdown.py the exam detaisl are stored in a table format specifically in fate and subject format

def save\_details(self, date, subject):

try:

formatted\_date = datetime.strptime(date, "%m/%d/%y").strftime("%d-%m-%Y")

data = self.load\_details()

data.append({"exam\_date": formatted\_date, "subject": subject})

with open(self.FILE\_PATH, "w") as file:

json.dump(data, file, indent=4)

self.update\_countdown()

return True

except Exception as e:

messagebox.showerror("Error", f"Error saving exam: {str(e)}")

return False

### Writing and reading files

I read the user availability and also make changes when changes are made in .

Calculator\_normal.py

def load\_json(filename, default\_data):

if os.path.exists(filename):

with open(filename, "r") as file:

return json.load(file)

return default\_data

def save\_json(filename, data):

with open(filename, "w") as file:

json.dump(data, file, indent=4)

### Files organised for sequential access -

def load\_quotes():

with open("quotes.json", "r") as file:

data = json.load(file)

return data["quotes"], data["tips"]

# Select a quote or tip for the day

def get\_daily\_message():

today = datetime.today().strftime("%Y-%m-%d")

try:

with open("last\_message.json", "r") as file:

last\_data = json.load(file)

if last\_data["date"] == today:

return last\_data["message"]

except FileNotFoundError:

pass

# Pick a new quote or tip

quotes, tips = load\_quotes()

message = random.choice(quotes + tips)

# Save the message to keep it for the day

with open("last\_message.json", "w") as file:

json.dump({"date": today, "message": message}, file)

return message

### Records -

**Exam\_details.json**

[

{

"exam\_date": "02-04-2025",

"subject": "Math"

}

]

def save\_details(self, date, subject):

try:

formatted\_date = datetime.strptime(date, "%m/%d/%y").strftime("%d-%m-%Y")

data = self.load\_details()

data.append({"exam\_date": formatted\_date, "subject": subject})

with open(self.FILE\_PATH, "w") as file:

json.dump(data, file, indent=4)

self.update\_countdown()

return True

except Exception as e:

messagebox.showerror("Error", f"Error saving exam: {str(e)}")

return False

### Recursive

I used recursion here so that each section gets updates in one go insteado of having 3 individual section looping for the user to input a value it gets recalled until the user moves away frpm this section . But it is laggy and buggy making it not ideal for the situation

def generate\_recom(self):

# Clear previous recommendations

self.analysis\_txt.delete("1.0", tk.END)

self.method\_txt.delete("1.0", tk.END)

self.suggestion\_text.delete("1.0", tk.END)

for widget in self.resource\_button\_frm.winfo\_children():

widget.destroy()

subject = self.subj\_var.get()

try:

with open(self.scores\_file, 'r') as f:

data = json.load(f)

subject\_data = data.get(subject, [])

last\_three = subject\_data[-3:]

except Exception as e:

last\_three = []

#Performance Analysis

performance\_recs = self.get\_performance\_recom(last\_three)

if performance\_recs:

for rec in performance\_recs:

self.analysis\_txt.insert(tk.END, f"• {rec}\n")

else:

self.analysis\_txt.insert(tk.END, "No performance analysis available.\n")

#Struggle-Specific Methods

struggle\_recs = self.get\_recom()

if struggle\_recs:

for rec\_text, resource in struggle\_recs:

self.method\_txt.insert(tk.END, rec\_text + "\n")

if resource:

btn = ttk.Button(self.resource\_button\_frm, text="Go to Resource",

command=lambda url=resource: webbrowser.open(url))

btn.pack(pady=2, anchor="w")

self.method\_txt.insert(tk.END, "---------------------\n")

else:

self.method\_txt.insert(tk.END, "No specific struggle selected.\n")

#Feeling-Based Suggestions

feeling\_input = self.feeling\_entry.get().strip()

feeling\_recs = self.linear\_search\_feelings(feeling\_input)

if feeling\_recs:

for rec in feeling\_recs:

self.suggestion\_text.insert(tk.END, f"• {rec}\n")

else:

self.suggestion\_text.insert(tk.END, "No feeling-based suggestions available.\n")

def recom\_loop(self):

self.generate\_recom()

self.after(2000, self.recom\_loop)

### Queue -

This is a queue not a stack As it follows first in first out (FiFO) which is a queue behaviour and I have used a queue to store the correct path and if they go too the correct path the questions gets poped out of the queue

nodes\_process = [root]

while nodes\_process:

node = nodes\_process.pop(0)

if node.children["correct"]:

nodes\_process.append(node.children["correct"])

# Testing-

I have went through my program and tested it to see if any of the program would not work or break , and after going through a vast amount of test these are the result I have come up with . I also asked my users to try out the app and give me their error point so I can make a note of that .

## Manual Testing

I carried out 54 test all documented in the following table . In the program I tested

* User account creation
* User account validation
* Password length and if it macthes
* Error handelign for missing or incorrect inputs
* Button functionality
* Calendar date/timer input behavior
* Display updae
* And user features

## User testing –

I have asked my stakeholders to test the program to see what error they might have run into and they have reported me back of these errors –

* Graph doesn’t upate automatically
* Performance doesn’t show up
* And events custom created by the user will be deleted when the timtable is made

2 of the erros are in my document but not the last one I did not notice that the code deletes all the events in the week just to make the timetable. I need to make another database table to store the schedule and keep the user events at another table. This fix will allow the deletion and update of the schedule not affect actual user custom events . While I have a fix for this I do not know why the performance is not showing . it may need further investigation

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test ID | Purpose | Input | Expected Output | Actual result | Pass/Fail | Time stamp |
| 1 | SignUp opens | Click the signup button | A new gram will pop out | The frame was made successfully | Pass | 0:14 |
| 2 | Check credential is taken | Username already in database | If taken report error | Username is taken error shown | pass | 0:37 |
| 3 | Check passwords length is greater or equal too 4 | 1,2 – or any 2-character passwords | Error shows saying passwords at least 4 characters | Error is shown and doesn’t tell it to enter atleast 4 character | pass | 0:49 |
| 4 | Check if password macthes | 2 differetn passwords | Error message saying password don’t match | Error is shown and says password don’t macth | Pass | 0:58 |
| 5 | User created | Valid credentials | User created succesffulyy | User was created properly | pass | 1:06 |
| 6 | Closing the button and not making a new login form | Click the close button | Closes the success message and the signup | Closed both | pass | 1:14 |
| 7 | Check if password is entered or username | Password or username not both | Error password and username should be entered | Shows the error | pass | 1:21  And  1:26 |
| 8 | Check for righ credentials | Invalid password or username | Error password or username not valid | Shows the error | pass | 1:37  And  1:44 |
| 9 | Check for the right credial | Valid password and username | Loading screen pop up with success mesasage | Show the success and loading screen pop up | pass | 1:52 |
| 10 | Check if user has completed setup or not | N/A  No input | A new window should pop up telling the user to finish setup | Window pop up | pass | 1:55 |
| 11 | Check if there is an entry for the subject | No input , and click add button | Error add atleast one subject | No window but doesn’t add the empty subject | fail | 2:13 |
| 12 | Check if atleast one subject was added | No subject was added and the use presses save and continue | Error atleast add one subject and let them try again | Error shown and lets them go back the other window pops up bug needs to be fixed | pass | 2:27 |
| 13 | Check if availability is enteres in 24 hour format | Any number smaller then the first number so like 13-12 | Error ask the user to enter in 24 hour format | Error is show | pass | 3:41 |
| 14 | If no availability entere no error | Nothin on one of the days | It goes through to the main program | Nothing is shown | pass | 3:55 |
| 15 | Load main program | No input | Main program to pop up | Main program didn’t pop up had to click it to make it pop up | fail | 4:04 |
| 16 | Filtered task | Add a valid Task | Goes to the selected section | Went to the specific section | pass | 4.38  And  5:03 |
| 17 | Different event time storage | Select any other time expect 7 am | Add task with the correct time | Addes successfully with the correct time | pass | 5:03 |
| 18 | Custom data opens when adding a task | The user presses the custom button | Calander pop up and lets the user select a date | Calender pop up and correct date was implemented | pass | 5:21 |
| 19 | Function of the button select custom date | User presses it | Calander pop up and allow to change the date | Calander pop up with correct change | pass | 5:34 |
| 20 | After creation able to change the event | Click on event and change | Event details pop up to change | Nothing | Fail |  |
| 21 | When the task is completed it doesn’t delete directly | Click on the tick box | Tick box turn red | It turns red | pass | 7:27 |
| 22 | When the task is completed it doesn’t delete directly only after closure | Stays there for a bit then move away from window then come back | It will not delete but when the user moves away to another window then it deltes | Deletes all the task once the user moves awat from the window | pass | 7:48 |
| 23 | Calander fetches events properly | No input | All the stored events display at the correct date and time | Correct location of events | pass | 5:43  And 6:02 |
| 24 | Next week button and previous week button function as they should | Click on one of the buttons | Display next week | It displays the next week | pass | 5:58 |
| 25 | Today button function as it should | User presed the button | Goes back to todays date and the week | Correct week and date | pass | 6:16 |
| 26 | Quick add button | Pressed button enter event name and date | Saves it in databae and also display in the calendar | Saves properly  Shows success of saved event | pass | 6:33 |
| 27 | The user can add a task by clicking on the position they want and add a task | A title | Success notification and it should show in the calendar | Event addes succefully and is shown | pass | 7:09 |
| 28 | Select the timer | Click either 25 or 5 minute and start | Show the value and count down | Count down initiated | pass | 8:14 |
| 29 | Pasue and reset button works | Press the button | If pause pressed pause the timer  If reset pressed rest the timer | Timer Is paused  Timer resets | pass | 8:26 pause  8:28 reset |
| 30 | Timer option locked when running | They click either 25 or 5 min | No timer changer | Timer changed for a split second then went back | fail | 8:47 |
| 31 | Show user subject | No input | User subjects | Shows them correctly | pass | 9:01 |
| 32 | Adding flashcard | Front and back | Flashcard added success fully | Flashcard added | pass | 9:30 |
| 33 | Auto update flashcard | No input | New flashcard displayed | No flashcard displayed | Bug the first one didn’t add ,the one after the first oen work normally | 9:42 |
| 34 | Study view works | User pressed start studying | The user will be able to see through each flash card allow to flip them and also delete them and go to the next card | Flip successful  Next card successful  Delete success(with sure notification) | pass | 10:59  11:02  11:26 |
| 35 | Examcountdow work ? | Input exam name and date | Shows the days left until exam | Shows remaining days | pass | 12:06 |
| 36 | Resources (binary seach get the subject entered ) | Any subject name such as math | Show the resources | It shows the resources for math | Pass | 12:25 |
| 37 | The button pulls up the website | Click of the button | Show the website to use the resources | Goes to website | pass | 12:43 |
| 38 | Availability show with the time interval | No input | It shows the stored availability | Shows | pass | 13:36 |
| 39 | Add scores | Add score date and also the subject you got it for | Success of adding the score | It success | pass | 13:49  14:15 |
| 40 | graphupdates | No input | Shows the new inputed values | Nothing changed | Fail  Only shows when u refresh the page | 14:27 |
| 41 | The graph control work | Change the period of tiem or range of month or the subject | Shows the specific grap with the corresspoding year and month range with the selected subject | Correctly changes | pass | 14:53 |
| 42 | Schedule been made with the score | Nothing | In calender or task view their should be subject name created to represent the schedule with the corresponding availability focusing on the high weighted subject | Shows the time table with the lowest achivig subject being repeated more times | passs | 17:44 |
| 43 | Analyse pop up nicely | Analyse button pressed | The analyse program | The return back to menu is out of the range only after I make changed then does it show | Fail or else pass as the whole app come nicely | 21:45 |
| 44 | Performance analysis | Nothing | Gives feedback on the suer performance | No performance | fail | 22:04 |
| 45 | Struggle | Click one of the option | Shows the clicked struggle method to fix | Shows the struggle | pass | 22:38 |
| 46 | Does it take you to the website | Click the resource | Shows the website | It does show | pass | 22:52 |
| 47 | Feelings | Enter user feeling | Correspondign tip to the feeling | Yes pull up | pass | 23:25 |
| 48 | Recursion work properly | Input feeling and select struggle | Updates the frame | Updates the frame | pass | 22:46 |
| 49 | Quesiton and answers taken out of the API | Choose the subject | Should show question | Yes show question specially from api | pass | 24:01 |
| 50 | Boss entrance | Click start | Animation for boss work | Yes the boss moves , but return back for some reason | Pass, with bug | 23:57 |
| 51 | Player and boss takes damage | N/A | When taking damage flash health and reduce by one | Reduces health and flash | Pass | 24:40  24:54  25:01  25.21 |
| 52 | Boss die ask the user if they want next level | N/A | Boss is dead and a window should pop up asking to carry on | Window pop up and going to next level success | pass | 25:34 |
| 53 | Player dies ask to restart or go back | N/A | Show pop up for death and also a choice to carry on or change subject | Death is confirmed and restart is also confirmed | Pass | 25:56 |
| 54 | Lock the select subject so the user cannot change when game started | Tries to click and change another subject | Doesn’t change or drop down the menu | Doesn’t drop the menu | pass | 27:07 |

## Summary –

The testing was 30 minutes long with 54 test conducted and with 47 of those test coming out successfully which leaves 7 test coming back as failed , which shows errors in my app which need to be addressed when making upgrade to the program , but 87% of the test was successful which is not a bad percentage . In the testing I thoroughly evaluated user authentication, task management, calendar features, data validation, error handling, UI navigation, game elements, and analytics functionality, and found failures in in empty subject handling (Test #11), main program loading (Test #15), event details display (Test #20), timer locking (Test #30), graph updates (Test #40), analyze feature navigation (Test #43), and performance analysis (Test #44).

# Evaluation –

## Testing summary –(summary of the one above )

In my testing, I conducted 54 tests over 30 minutes with an 87% success rate (47 passes, 7 failures). The evaluation covered user authentication, task management, calendar features, data validation, error handling, UI navigation, game elements, and analytics. Failures occurred in empty subject handling (#11), program loading (#15), event details display (#20), timer locking (#30), graph updates (#40), analyze feature navigation (#43), and performance analysis (#44). These issues will need to be addressed in future updates.

## Success criteria

My solution successfully fulfilled most of the key requirements that I put at the start of my project. The user authentication system (Criterion 1) achieved full functionality, with proper validation for credentials, showing errors when needed, and letting users signup securely. Testing showed all validation checks worked fine, but the main program sometimes had issues loading (Test #15).

The task management part (Criterion 3) worked well, with tasks correctly shown as today, tomorrow, and upcoming. The complete button worked how I wanted, keeping completed tasks until closure. But the calendar feature (Criterion 4) had some problems - while users could navigate weeks fine, sometimes the event details wouldn't show right (Test #20).

The flashcard system (Criterion 5) met all requirements, letting users add, delete, and view cards by subject, and checked if cards had both front and back content. The Pomodoro timer (Criterion 6) had all the controls I planned, but sometimes the timer would lock up (Test #30).

The analytics section (Criterion 7) only partly met requirements - it showed subject scores and let users change their availability, but the graphs didn't always update right (Test #40) and the performance analysis sometimes gave wrong results (Test #44). The mini-game animations (Criterion 8) worked as planned, showing the right notifications and properly animating player and boss health changes during gameplay. The timetable feature (Criterion 9) also performed well, correctly differentiating between study days and off days and creating schedules based on the latest scores.

My code demonstrates good integration with MySQL databases, proper handling of user interfaces with Tkinter/CustomTkinter, and effective data management with both database storage and JSON backups. Future improvements could include better password security through hashing, more consistent time format handling across the application, and fixing the occasional graphing and analysis issues in the analytics dashboard.

## Strength and limitation –

### Strength –

My code has a very user friendly experience and also a very functional one as the user all they have to do insert their score and availability and the timetable will be made with ease , also making the program gamified makes it more intriguing and also interesting as user will naturally check the game and then of course makes them more motivated about the program . Another strength is that the program has a deep analyss of the user studies and can give advice depending on the user input , this not only allows to see if the user is failing and if they study what they can get this also helps the user to no suffer alone it will be there stepping stone to get help .

### Limitation –

The program is some part look rough doesn’t look as I wanted too and also there is no security on the program which is definitely need to be considered in the future development , and also the event situation where the schedule deletes the user custom events causing logic error . the app is no fit to be for a device such as a laptop or phoen only for desktop with 768 width .

## UserFeedback –

The code was made for the students who are struggling balancing their subject or their life and so I decided to ask my stakeholder and my supervisor to try my code and give me critical feedback which will help me improve my code If I was going to develop it further more .

First I have asked my stakeholder / a fellow students who is having struggle balancing their math revision with their business presentation there are the perfect user to try the code as they are essentially unable to balance their time and having a hard time trying to find resources to study from aswell .

**For context –**

The stakeholder name is maisha , and she studies business and math the business course she doing is a level 3 btec where their college have frequent business presentation where each week she will be needing to complete the presentation for the following week . But she also need to study for math and she quoted “impossible to balance them ”.

So I gave them my code for 7 days to let them use it , And she has came back and said the following .

“The app feel completes like the app you can get from the app store it runs smoothly . I like how the program looks its friendly and also very helpful even tho It doesn't have much resources I feel like if their was more resources it would make It more useful but there is a minor flaw in the system well actually two flaws that wasn’t accounted for and both of them are for the study planner . What the problems are is the fact that If I am a new user which just started the year I will not get a schedule which I think was the main purpose of the code to make the user a timetable well if the user needs to have a score to make a timetable then the user will be set behind for the first 3 month as that is usually when we take test . Another problems is the fact that when the user add their own events it successfully adds the events but the problem is the fact if I generate my study schedule the events I manually enters do not stay they also get deleted , which I disliked about the program or else I really do like how it creates a nice dynamic schedule the schedule actually helped me study for my math and also help me meet the deadline , that means the program does help users from getting in the right path and balance both subject. “

In conclusion, The stakeholder liked the program and it did help them on their studies but they did run into problems such as the events being deleted when the study plan is made and also the problem of not making a timetable when they’re just starting off .   
  
I also asked my supervisors from my tuition Mr. Rohman, Mr. Islam to test out my code I have went there personally and connect their screen board with my laptop . Mr.Rohman is a math teacher who taught students from year 11 to year 13 , same goes for Mr.Islam but he teaches science instead of math . Both of the teacher has seen the studens struggle and so I have asked them to test my code with their students , I went to their tuition session and loaded the program for the teacher to use and test .

**Mr rohamn said –**

“Your Eclipse application underwent testing from my class for 2 days, we used your app to generate on the topics they where struggling with . The application inetrefacesis quit friendly and very easy to understand which is why even the students who has hard on the device was able to use it .   
We as a class we decided to use the pomodoro timer , we did 25 mins of study then a 5 minute breaks , and I will be honest it deifnetly seen some increase In focus levels I was very surprised that it worked and I seen that students ask more questions now due to the increase of concentration level.

But I do have a problem with the flashcard it works as it intends for all words based subject but not math as it doesn’t have the notation for the math formulas , this was a problem for me and my class as we are a math class and writing formula is required .

Or else the program actually is realty helpful for my class as they can see their performance and decide what to do moving forward . therefore I come to a conclusion that he application is really helpful for studenst to visualize their path and see what they need to do moving forward “

In conclusion the supervisor has tested my program or used my program as class tool and it was successful even though it wasn’t build for such purpose the program actually made a change in the class and made a difference . He said that the scheduler helped to organize the class , the pomodor increased the focus level .

But he has suggested that to make the flahcard app more versatile allowing to add any type of words even formurlas which I might consider in the future

**Mr Islam –**

"I personally tested your Eclipse app with a few of my A-Level science students during their tutorial sessions, and I must say, it addresses many of the organization challenges they typically face. The study schedule generator is quite impressive - it creates realistic timetables that actually consider students' availability patterns. The dark-themed interface seems to resonate well with the students who tried it.

I found that when using the resource library, the science material seemed a bit sparse compared to other subjects. The few students who tested it were looking for more specific resources on topics like organic chemistry and quantum physics, but the options were pretty limited. Having a way for teachers to add custom resources would enhance its educational value.

On the plus side, those motivational quotes and the tree game are a nice touch. They provide a brief mental break without completely distracting from the academic focus. If you could expand the science resources it would benefit the students further .”

In conclusion , Mr islam has tested the app with the class . He was impressed with the schedule generator and make a realistic schedule which is the main purpose of the code , this is good if it meets a realistic schedule . The problem is that their isn’t much resources for the students so Mr islam wants to add his own resources .

Summary –

After reviewing feedback from the three users—Maisha (student), Mr. Rohman (math teacher), and Mr. Islam (science teacher)—several important insights emerge about the Eclipse Learning application.

Common Strengths

All users appreciated the application's clean, modern interface and found it intuitive to navigate. The pomodoro timer was highlighted as particularly effective, with Mr. Rohmana as he said it increased the class concentration level. The study schedule generator, when working properly, was praised for creating realistic and manageable timetables that helped with balancing multiple subjects.

Common Areas for Improvement

Some consistent issues were identified across multiple users:

1. Resource Availability: Users mentioned that expanding the resource library would increase the application's utility, particularly for specific subjects like science.
2. Technical Issues: Maisha identified two critical functional issues with the scheduler that impact user experience: the inability to generate schedules for new users without prior scores, and the deletion of manually added events when generating study plans.

Priority for developing –

* Fixing the core functionality issues with the scheduler to ensure it works for new users and preserves manually entered events.
* Adding support for mathematical and scientific notation in the flashcard system.
* Expanding the resource library, particularly for science subjects.

## Improvements –

After doing testing and getting the user feedback I have realized their one major problem in my program. Which is when the program generates the specific user timetable it also deletes all of the user stored events (events not made by the program ). To fix this I might have too make another database table called schedule which will store the schedule there and also update the schedule when necessary .

To solve this problem first we need to –

* Make a new table
* Make a new function to save the plan (in calculator\_normal.py)
* Make a new function to get the event (in calendar )

### New database table –

This table will not conflict with the other events the user may store and the program will only make changed to this table not the event table .

CREATE TABLE schedule(

user\_id INT NOT NULL,

subject\_id) INT NOT NULL,

date DATE NOT NULL ,

time TIME NOT NULL,

FOREIGN KEY (user\_id) REFERENCES users(user\_id),

FOREIGN KEY (subject\_id) REFERENCES subjects(subject\_id) );

### Update the calculator\_normal.py–

Update to save On the new table instead of event table . The program calculator\_normal is used to make the scheduele and the function called save\_plan saves the schedule . to make the schedules save onto the new table all I have to do is change the table

OLD -

def save\_plan(user\_id, plan):

conn = mysql.connector.connect(host="localhost", user="root", password="", database="nea")

cursor = conn.cursor()

cursor.execute("DELETE FROM events WHERE user\_id = %s", (user\_id,)) # Clear old plans for the user

for date, time, title in plan:

cursor.execute("INSERT INTO events (user\_id, date, time, title) VALUES (%s, %s, %s, %s)", (user\_id, date, time, title))

conn.commit()

conn.close()

NEW

def save\_plan(user\_id, plan):

conn = mysql.connector.connect(host="localhost", user="root", password="", database="nea")

cursor = conn.cursor()

cursor.execute("DELETE FROM schedule WHERE user\_id = %s", (user\_id,)) # Clear old plans for the user

for date, time, title in plan:

cursor.execute("INSERT INTO events (user\_id, date, time, title) VALUES (%s, %s, %s, %s)", (user\_id, date, time, title))

conn.commit()

conn.close()

### Update Calander.py

Update the calendar get\_event function to let both events to be displayed , and we do this by getting the event and then the schedule

Old

def get\_event(self):

# Fetch events from the db

start\_date = self.strt\_week.date()

end\_date = start\_date + datetime.timedelta(days=6)

self.cursor.execute("""

SELECT date, time, title FROM events WHERE date BETWEEN %s AND %s AND user\_id = %s

""", (start\_date, end\_date, self.user\_id))

rows = self.cursor.fetchall()

self.events = {}

# dynamic obhject genration

for date, time, title in rows:

try:

dt = datetime.datetime.strptime(time, "%H:%M:%S")

formatted\_time = dt.strftime("%I:%M %p")

if formatted\_time in self.hours:

time\_index = self.hours.index(formatted\_time)

day\_offset = (datetime.datetime.strptime(date, "%Y-%m-%d").date() - start\_date).days

self.events[(time\_index, day\_offset)] = title

except ValueError as e:

print(f"Error processing event with time '{time}': {e}")

NEW

def get\_event(self):

# Fetch events from the db

start\_date = self.strt\_week.date()

end\_date = start\_date + datetime.timedelta(days=6)

# events

self.cursor.execute("""

SELECT date, time, title FROM events WHERE date BETWEEN %s AND %s AND user\_id = %s

""", (start\_date, end\_date, self.user\_id))

user\_events = self.cursor.fetchall()

# timetable event

self.cursor.execute("""

SELECT date, time, title FROM events WHERE date BETWEEN %s AND %s AND user\_id = %s

""", (start\_date, end\_date, self.user\_id))

study\_ses = self.cursor.fetchall()

all\_events = user\_events + study\_ses

self.events = {}

# dynamic obhject genration

for date, time, title in all\_events:

try:

dt = datetime.datetime.strptime(time, "%H:%M:%S")

formatted\_time = dt.strftime("%I:%M %p")

if formatted\_time in self.hours:

time\_index = self.hours.index(formatted\_time)

day\_offset = (datetime.datetime.strptime(date, "%Y-%m-%d").date() - start\_date).days

self.events[(time\_index, day\_offset)] = title

except ValueError as e:

print(f"Error processing event with time '{time}': {e}")