MaxQ App Design Document

Table of Contents

1 Defining the Problem

My client is a lifelong learner studying mathematics and computer science. He usually studies alone with lesson materials he finds online, which makes it difficult for him to navigate foreign topics. Additionally, without some kind of system to monitor his studies, the client can be easily distracted and struggles with time management. He is also especially susceptible to disorganisation, procrastination, and lack of motivation.

Furthermore, the halting of popular music streaming services in his country, such as Spotify, SoundCloud, and Deezer, poses a problem for the client because music motivates him to study and focus. Currently, he is discouraged from listening to music because he needs to buy expensive vinyl records or download audio files from the web to do so, the latter of which is impractical without a music player. He emphasises that he still has access to *mp3* music files this way.

For these reasons, the client has been seeing a decline in his productivity lately and wants that to change. He would like to incorporate break times in his routine to prevent burnout systematically. To keep track of his lessons, he would like to list his goals for the day and check them off as he accomplishes them. To keep track of time, he would like to see how long he had studied in a day compared to past days. Lastly, he would like a way to listen to music in his studies to make the experience more enjoyable. (*Appendix I*)

2 Solution Rationale

The client and I identified three obstacles in his learning: lack of music, no daily goals, and absence of a proper break system. We agreed that a program containing a music player, timer, and to-do list would address these issues (*Appendix I*). Additionally, a database will store music details, preferences, statistics, and goals. The client can add audio files as songs, which come from the internet (file download). The user's daily study time for the past week will be displayed, letting him align productivity with goals. A login system will increase accessibility, only enabled when the user explicitly logs out to prevent hassle.

3 Software Justification

Python was chosen for its useful libraries, including audio file management. Python's simplicity reduces the time needed to read library documentation and implement external functions. To accommodate, PyCharm was chosen as the development platform as it provides debugging, code completion, and Python-specific support libraries, minimising hidden bugs. As a language-specialised platform, PyCharm will be simpler to configure than other alternatives like VS Code.

For GUI development, Python's built-in Tkinter library was chosen because of its lightweight properties whilst still providing support to construct a basic program. Being an internal library also entails that Tkinter will be using less memory than other external alternatives.

SQLite was preferred over MySQL or PHP for the database because it is lightweight, compatible with Python through the *sqlite3* module, but still sufficient for the program's needs. SQLite's integration with Python improves compatibility, reliability, and efficiency in both Pycharm development and future deployment.

4 Success Criteria

1. **To have a working timer that calls for breaks and study times accordingly.** The user can control how long their breaks and study times will be. The timer visually counts down and rings to symbolise the end of any period. This feature should be implemented with the use of recursion and third party time libraries.
2. **To be able to play and control music**. It should be accessible and functional during active study periods, and wherever the user goes in the program. The player should be able to play, pause, repeat, skip, and rewind songs. This feature should be implemented with complex or dynamic data structures including stacks, queues, 2D arrays, and custom data types.
3. **To keep track of the user's study time in a database.** The data will be used in the analytics page and should be automatically updated every time the user finishes a study period. The data will be put in contrast to his previous days so that the client can align his productivity with his goals. This feature should be implemented with techniques pertaining to database management systems including table structures and SQL queries.
4. **To allow the user to input preferences such as the length of breaks and study periods, colouring options, and music player volume.** The options will be displayed in the settings page and the user's saved settings will be uploaded to the database, making sure that the program remembers those preferences the next time the user starts the program. This feature should be implemented through validation and verification techniques as well as database systems.
5. **To provide a space for a task list and user goals**. The user can freely remove and add tasks to it so that he can regulate his goals for the day in a manageable manner. The space would act as a mini syllabus for him to set and follow. This feature should also be implemented through database systems and verification and validation techniques.
6. **To be able to allow the user to add songs to the music library**. The feature should be able to let the user choose any audio file from his computer library and add it as a song to the program. The user should also be able to rename and remove songs. This technique will be implemented through complex data structures and databases.
7. **The program should have a login system and partition user data.** This would allow multiple users to use the same program on the same computer, and the user's data is protected. A logout button should be provided. This technique will be implemented through database systems.
8. **To feature a simplistic user interface that reduces distractions, especially during active study periods.** The active studying page should be minimalistic to reduce distractions. This technique will be implemented through the use of third person GUI libraries such as Tkinter.

5 Flowchart Design Overview

1. System Flowchart (Homepage)

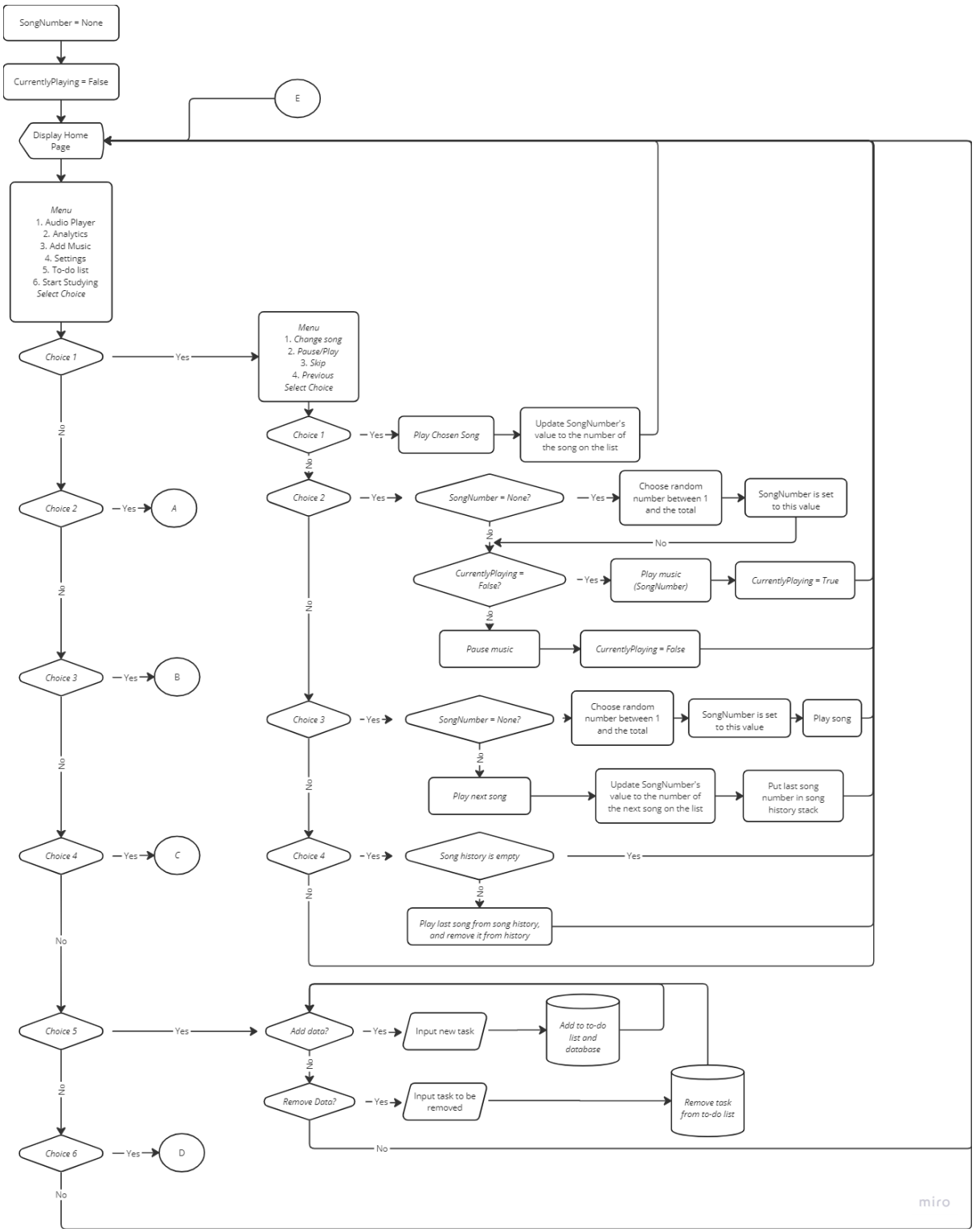


Image B.1

The lettered circles, from A to E, represent connections to other pages. Although connected to the system flowchart, they are depicted in modulated flowcharts to simplify the figure.

2. Login Page Flowchart

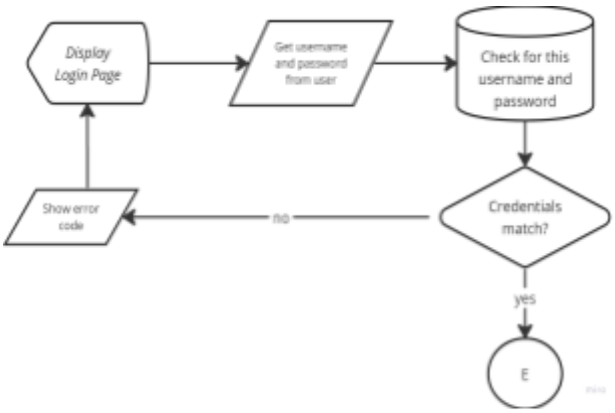


Image B.2

3. Register Page Flowchart

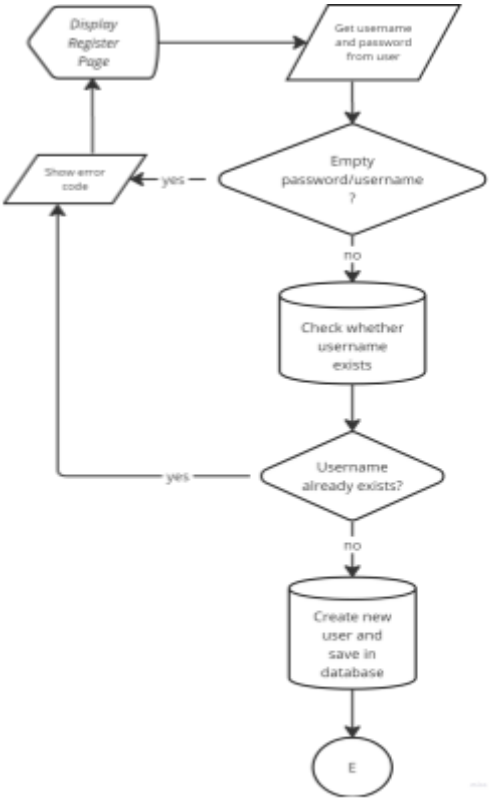


Image B.3

4. Analytics Page Flowchart

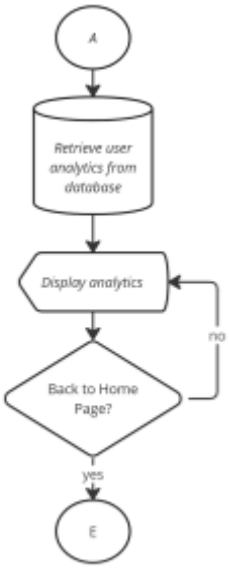


Image B.4

5. Add Music Page Flowchart

Image B.5

6. Settings Page Flowchart

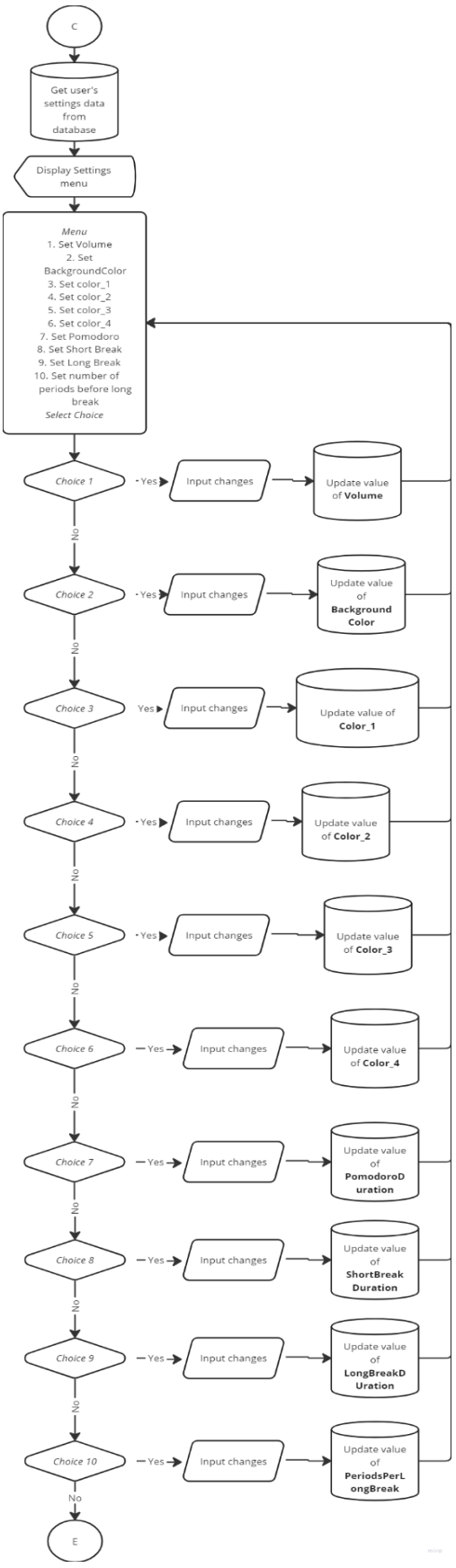


Image B.6

7. Start Studying Page Flowchart

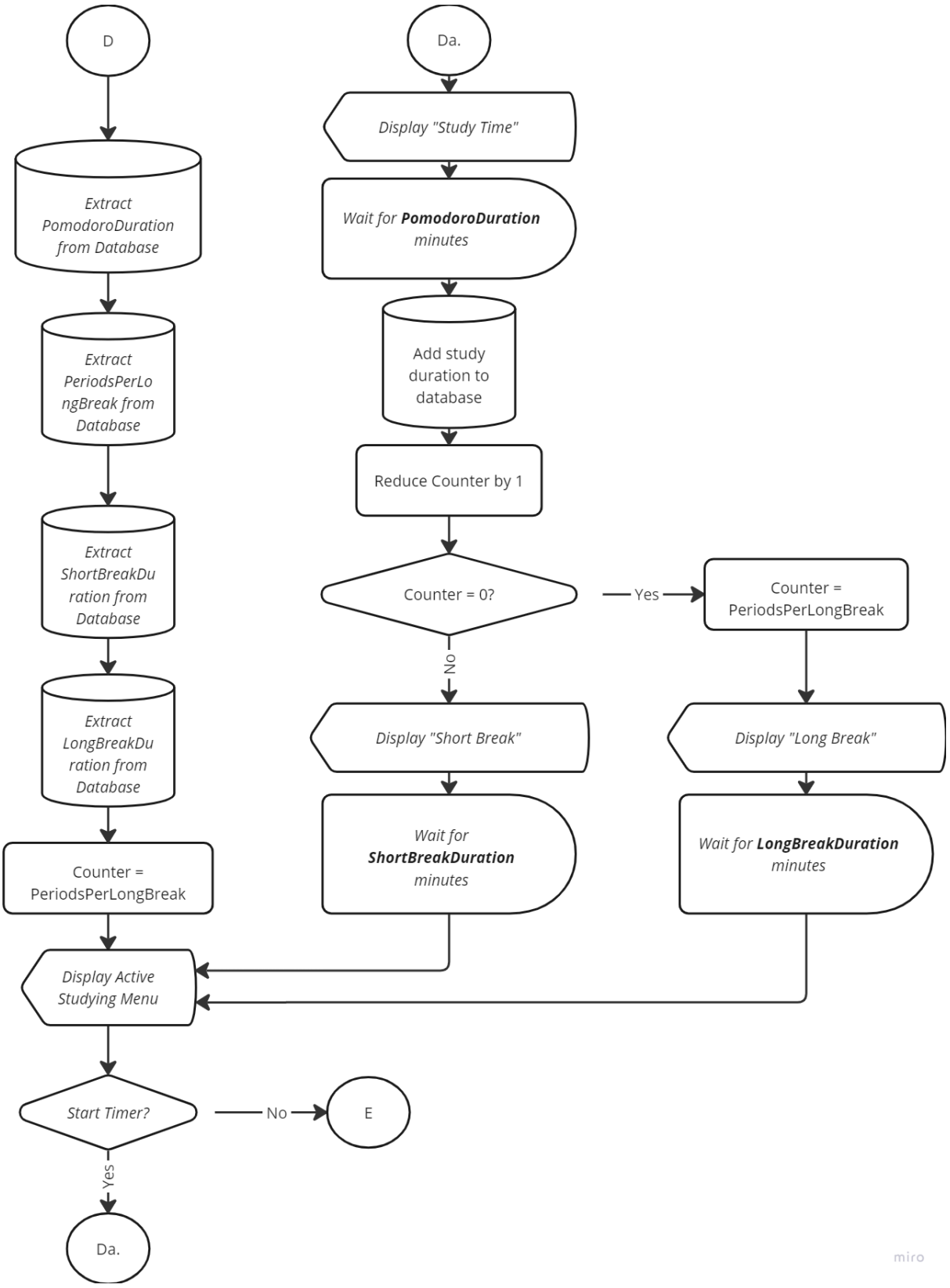


Image B.7

6 User Interface Design Prototype

* Refer to Appendix II for full transcript of client feedback

1. Login Screen

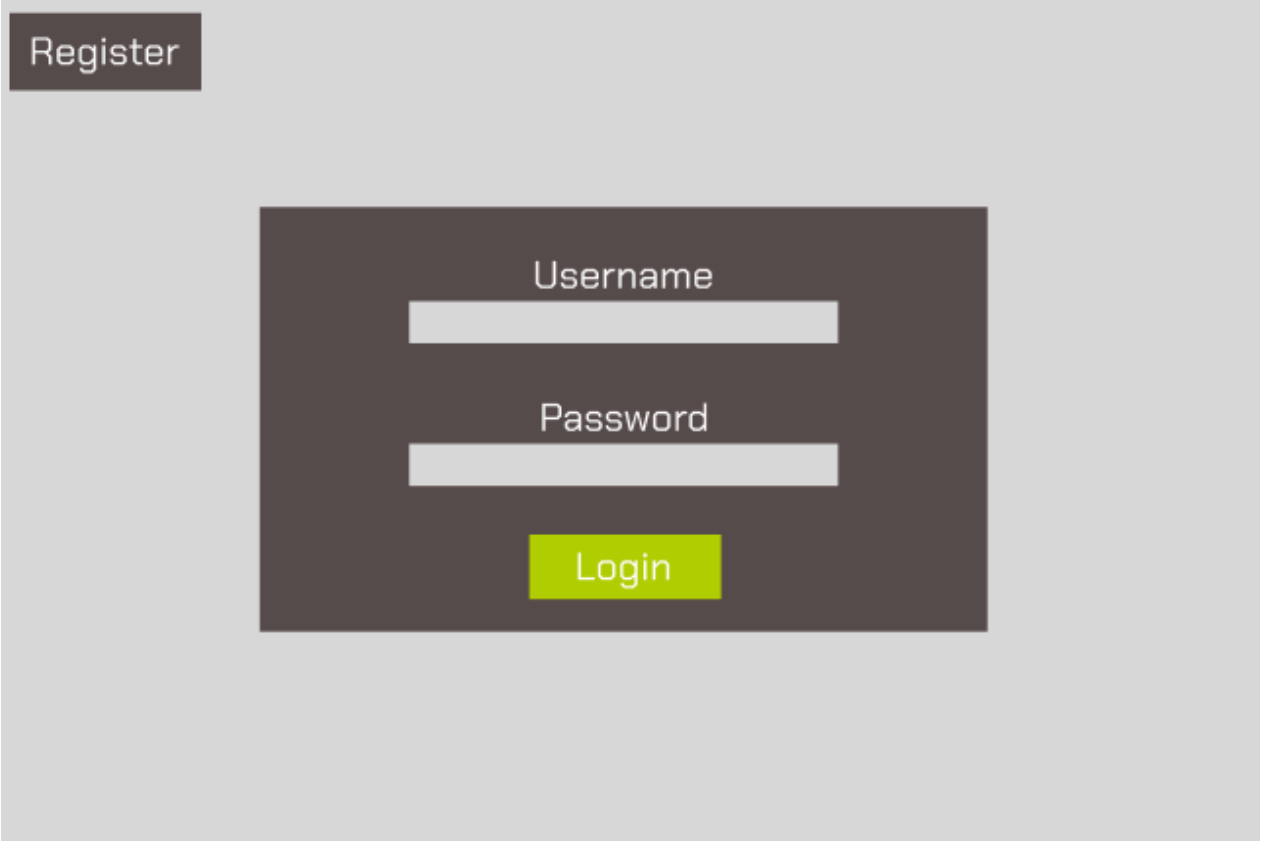


Image B.8

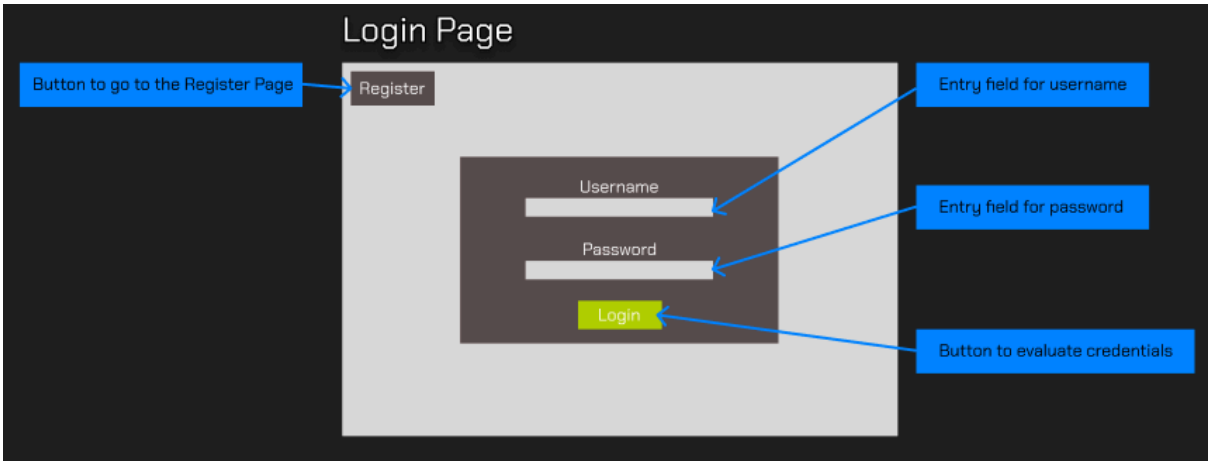


Image B.9 Client feedback: Use a darker background

2. Register Screen

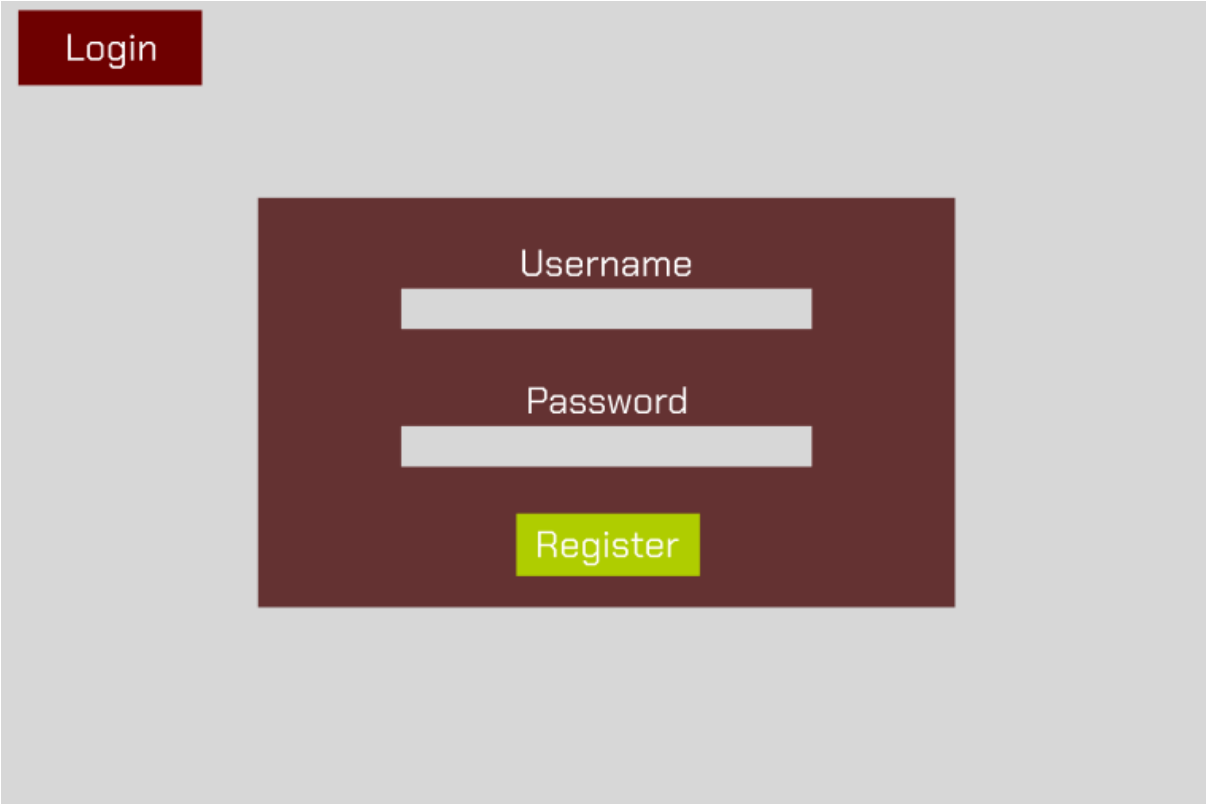


Image B.10

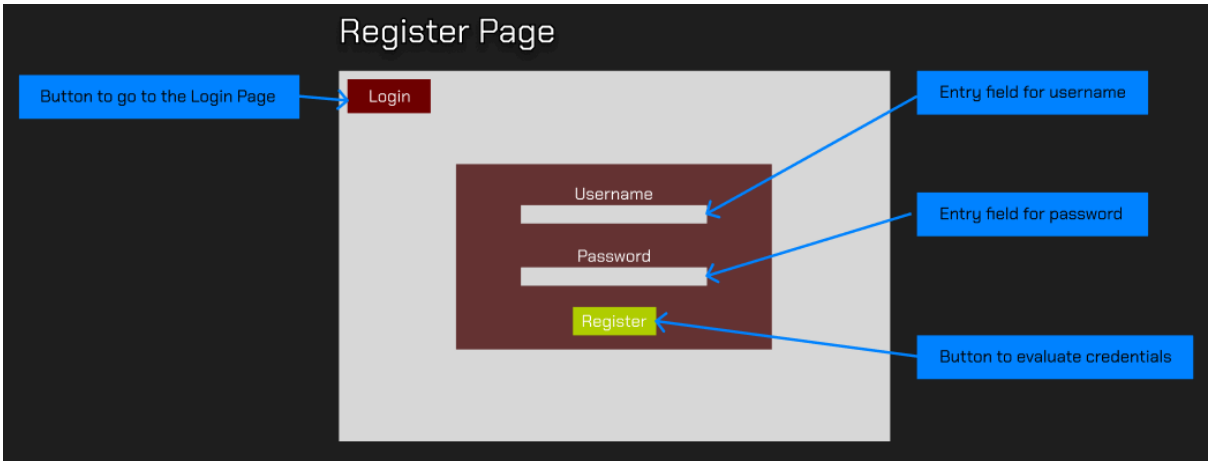


Image B.11 Client feedback: Use a darker background

3. Home Screen

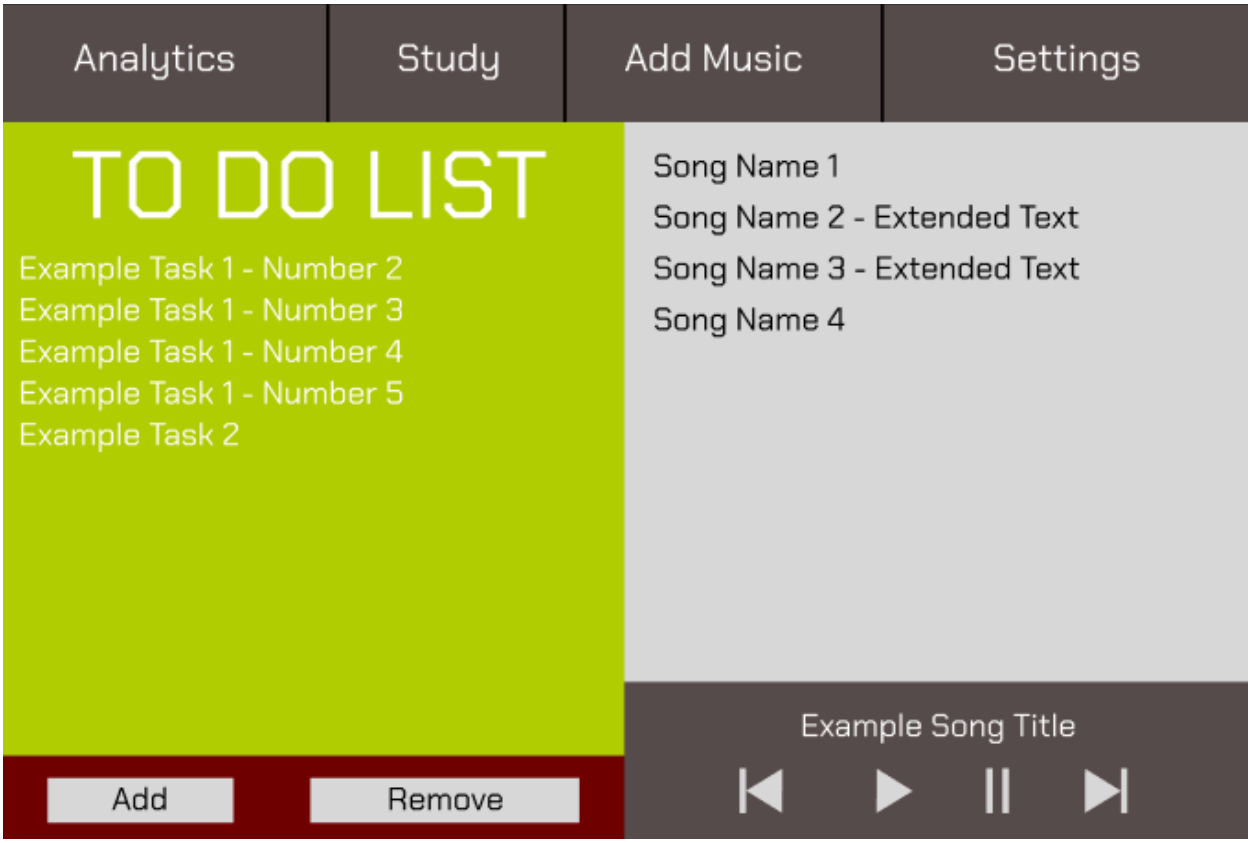


Image B.12

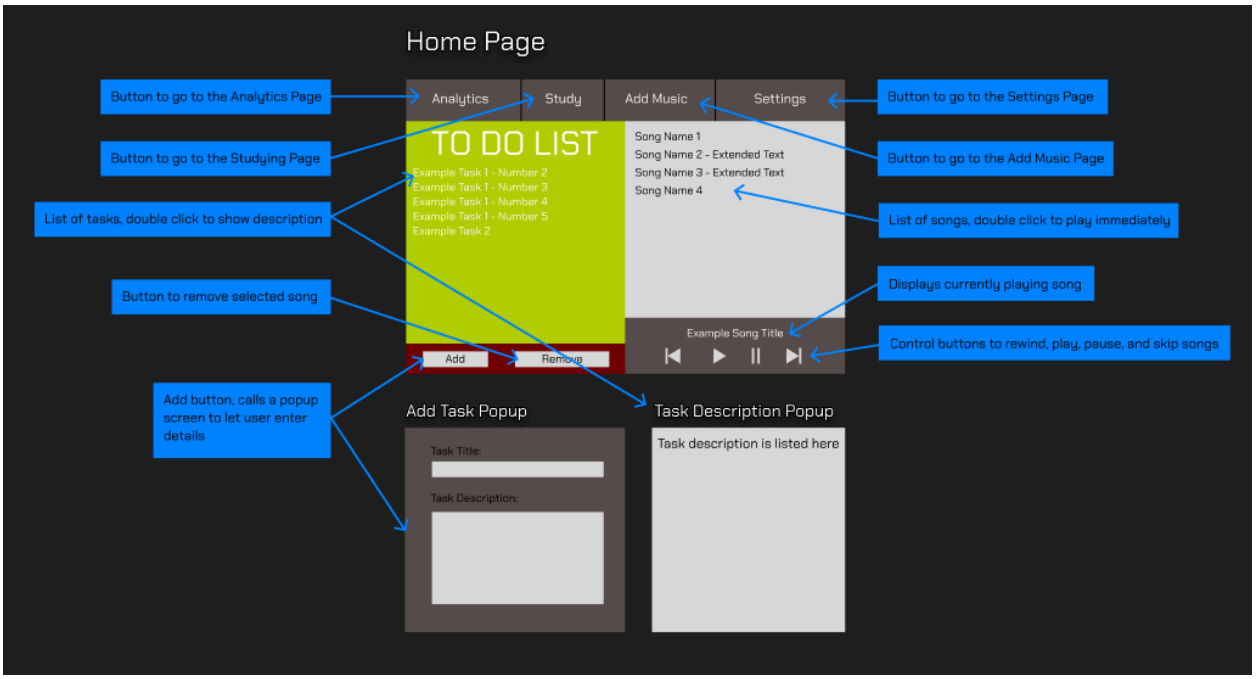


Image B.13 Client feedback: Use a darker background and less flashy colours

4. Analytics Screen

Image B.14

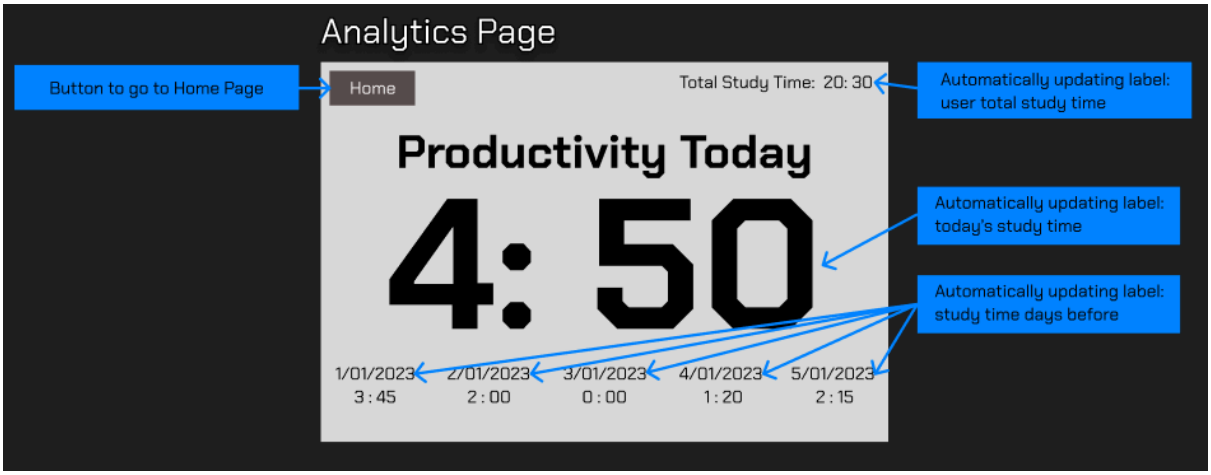


Image B.15 Client feedback: Use a darker background

5. Studying Screen

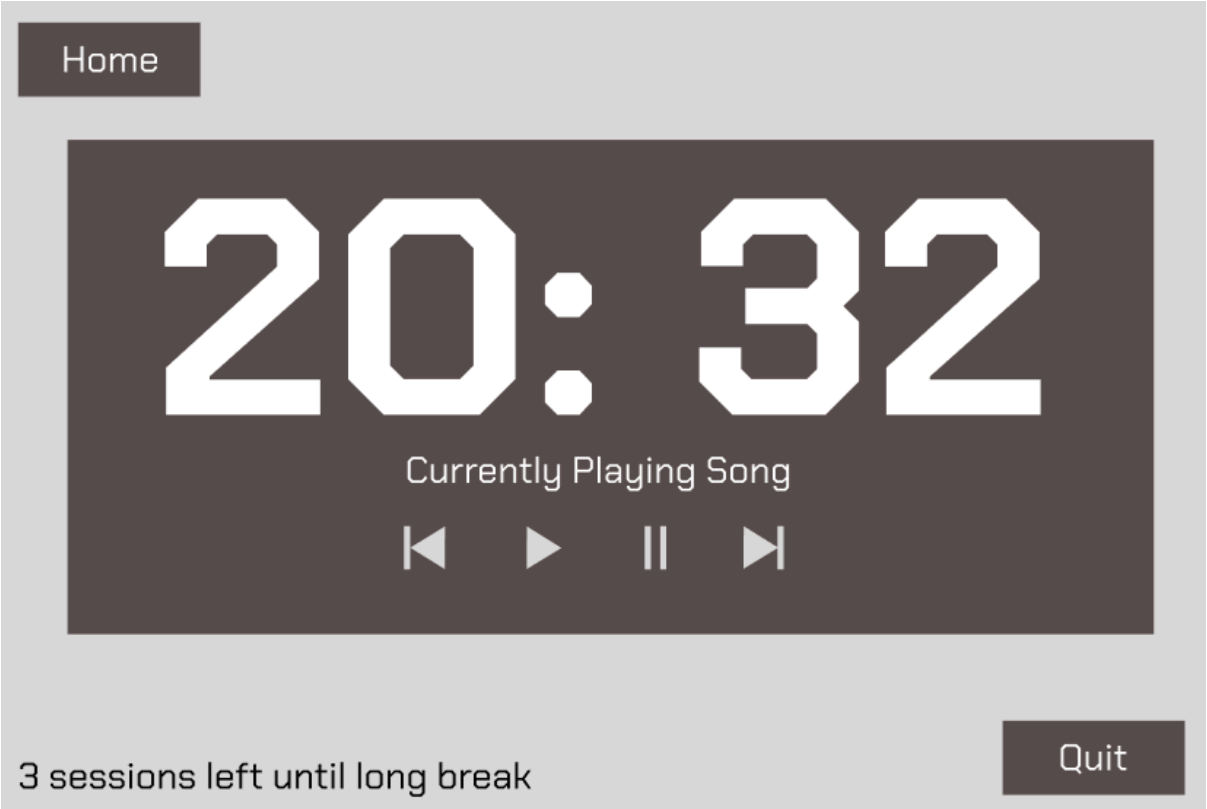


Image B.16

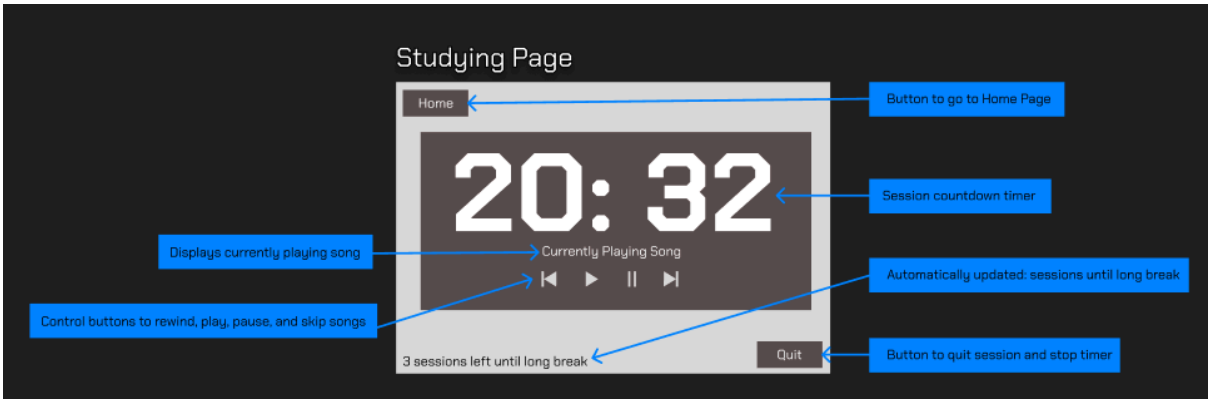


Image B.17 Client feedback: Use a darker background, the page is too cluttered so remove the player buttons and song title. Try to make the clock less distracting.

6. Add Music Screen

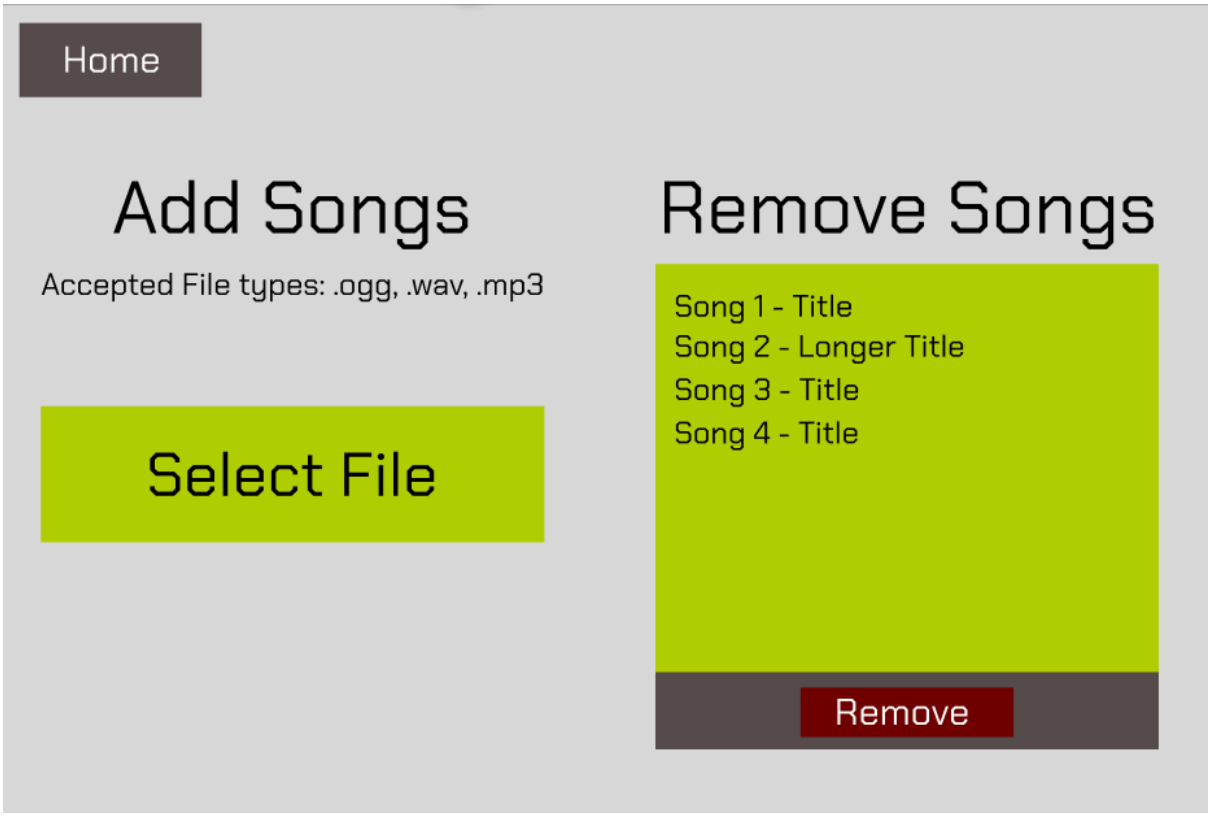


Image B.18

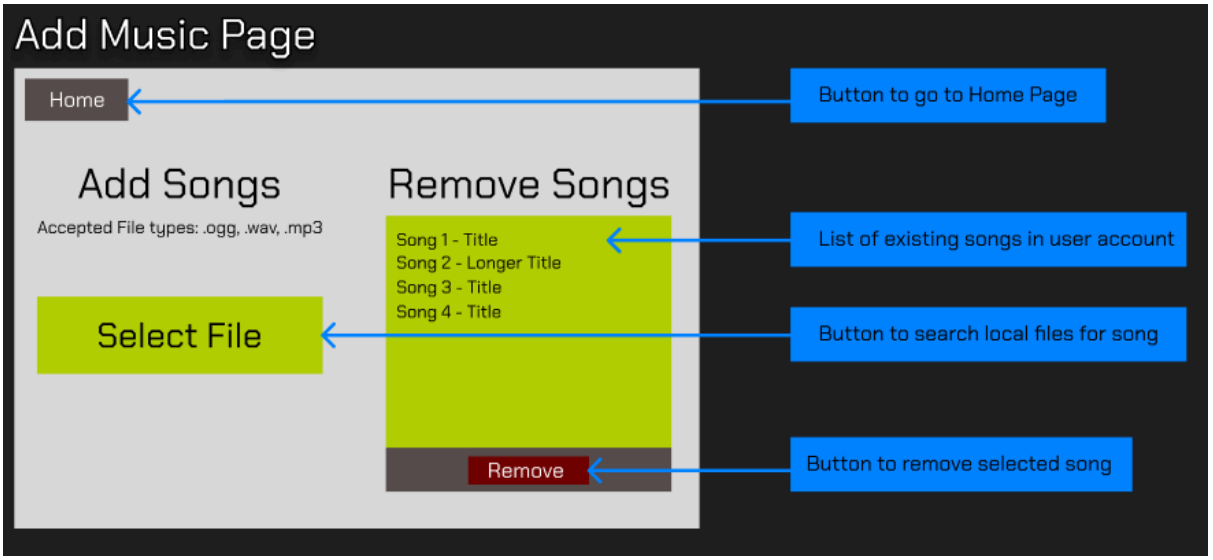


Image B.19 Client feedback: Use a darker background

7. Settings Page

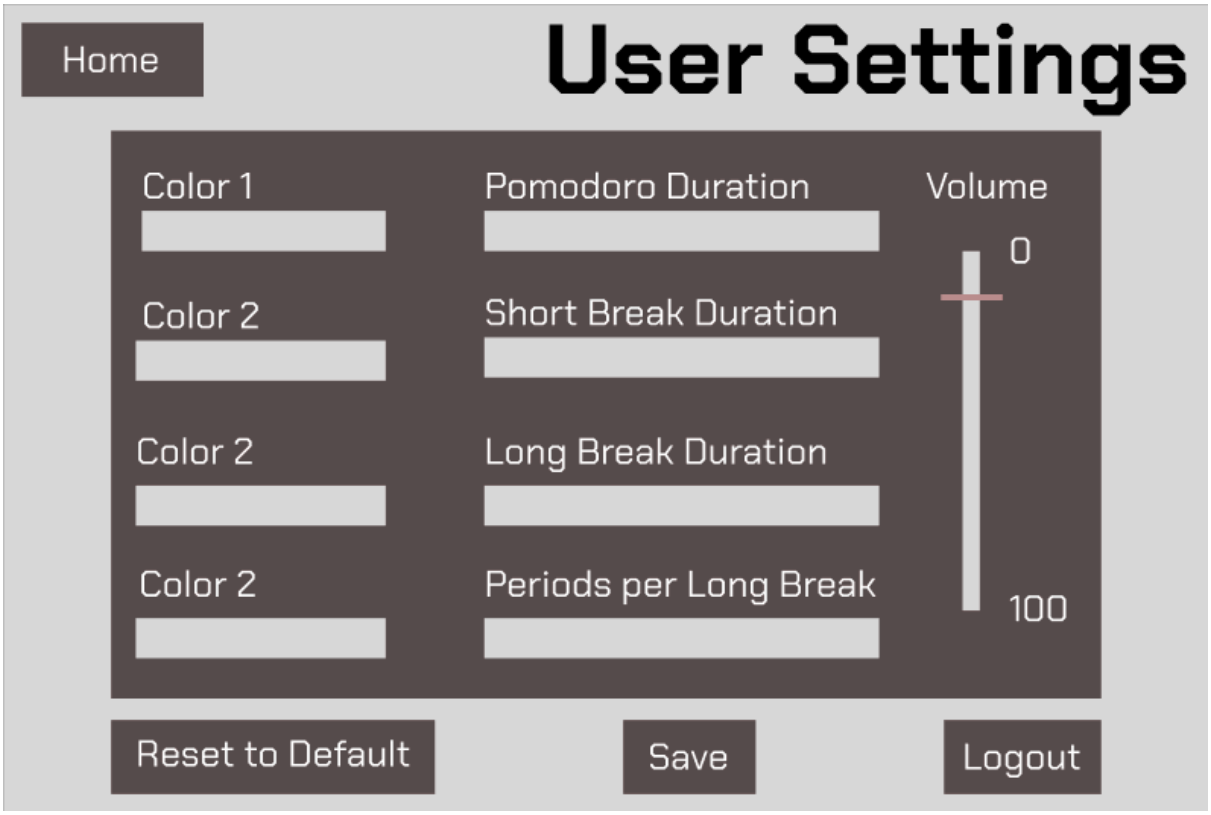


Image B.20

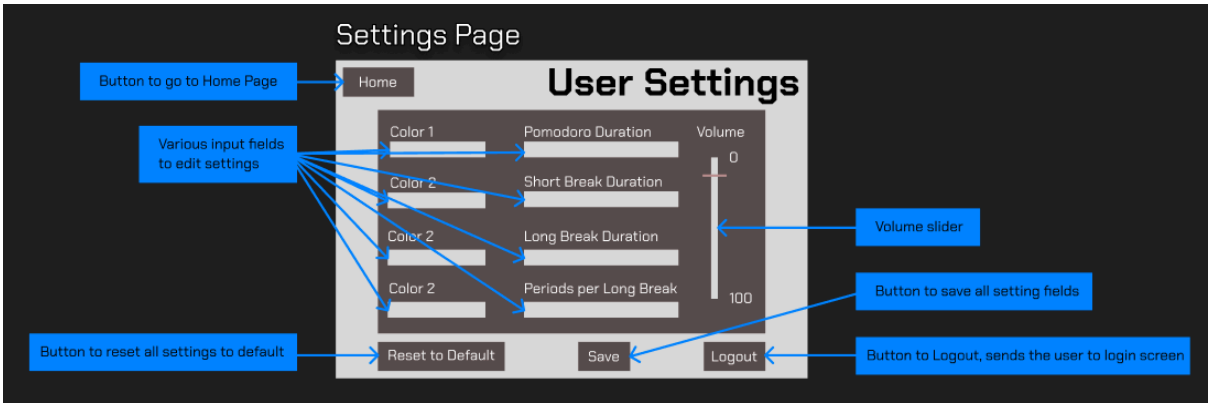


Image B.21 Client feedback: Use a darker background. Place the buttons elsewhere and spread out the entry fields more. The volume slider is confusing.

7 Entity Relationship Diagram

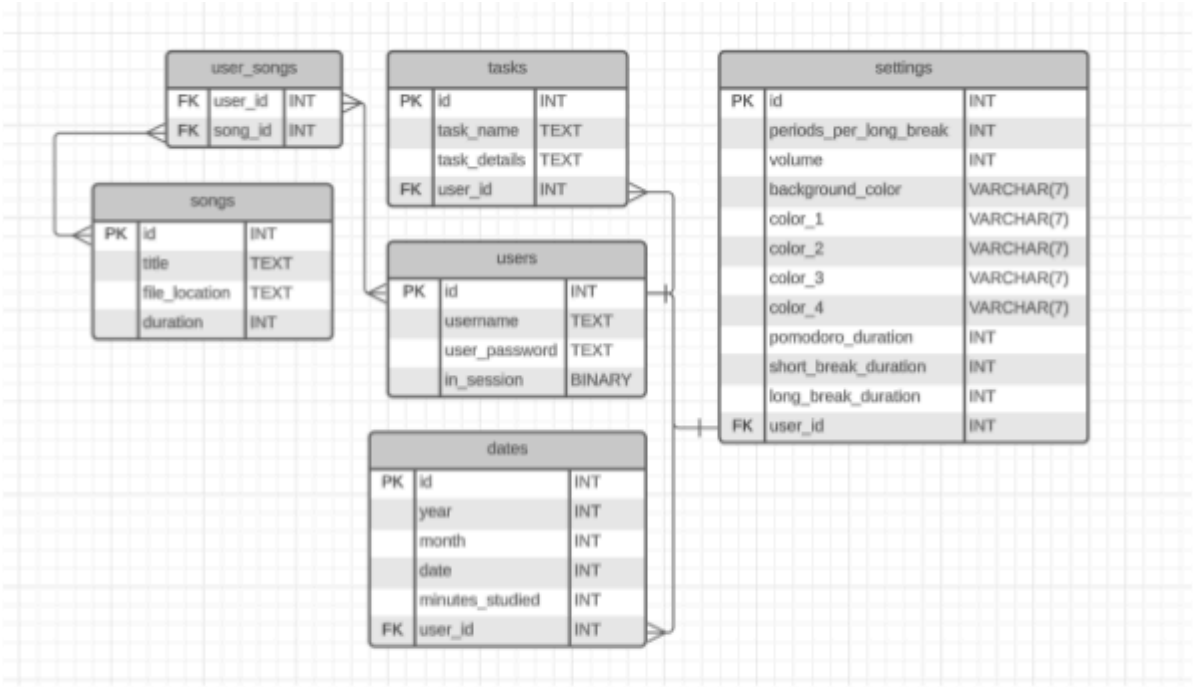


Image B.22

8 Unified Modeling Language (UML) Class Diagrams

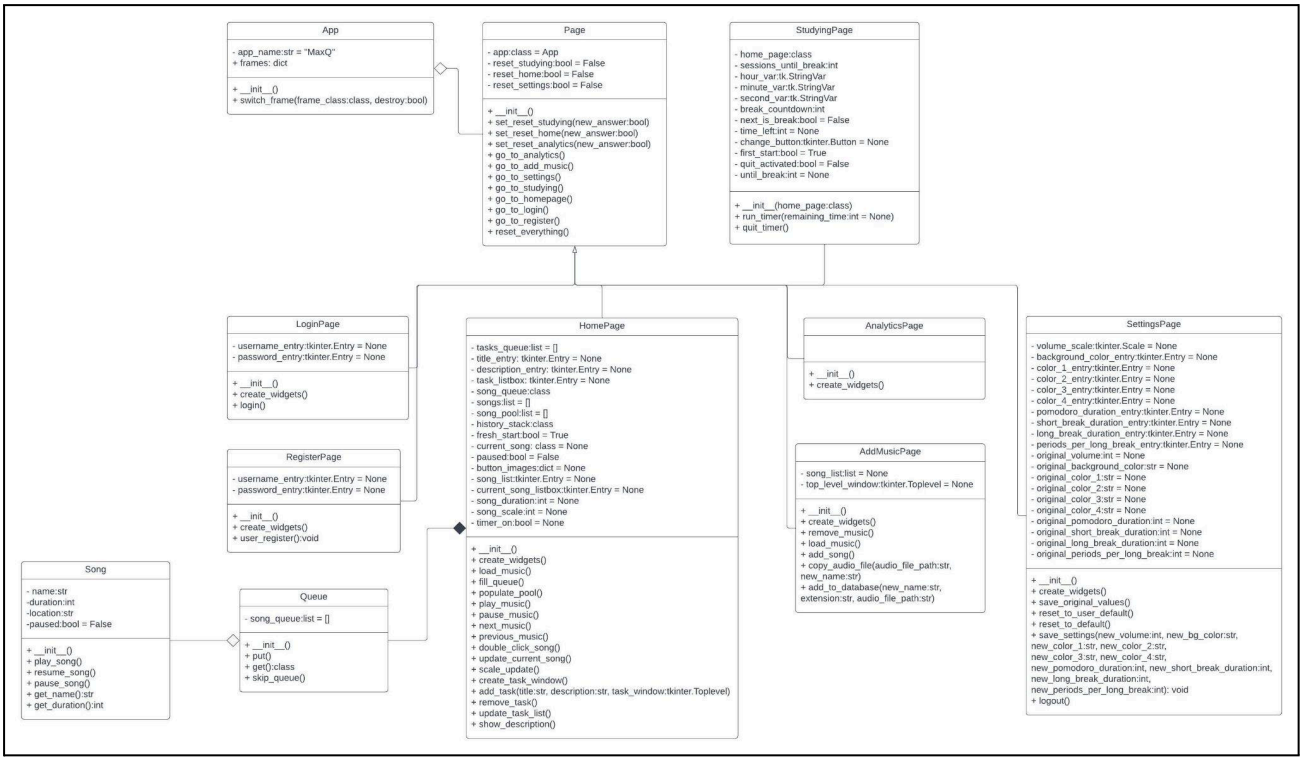


Image B.23

Since the diagram could not fit in one page in a readable format, the image is divided into three smaller parts and presented below:

Top side:

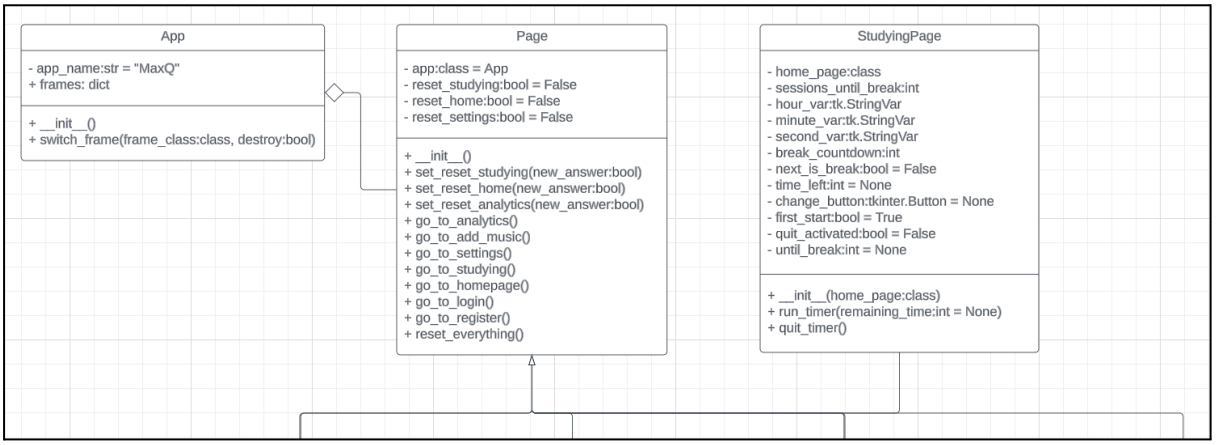


Image B.23.1

Left side:

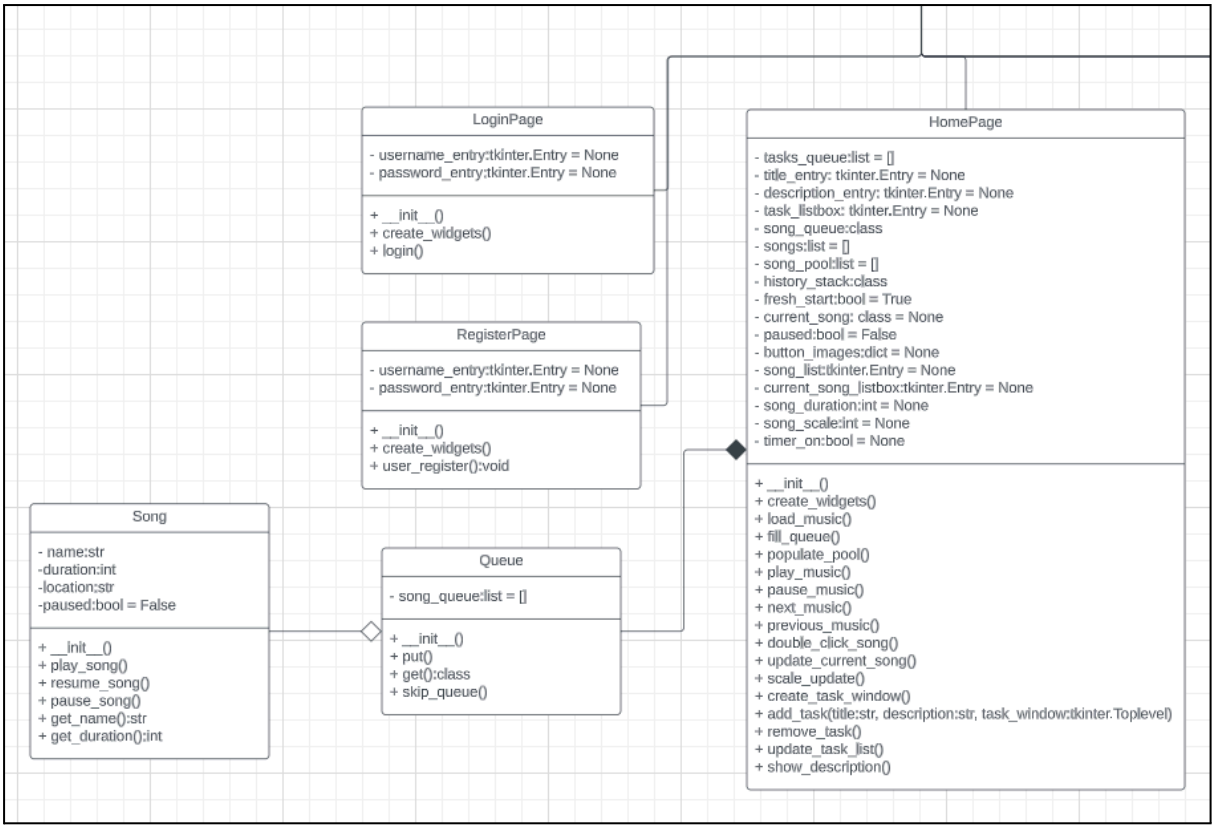


Image B.23.2

Right side:

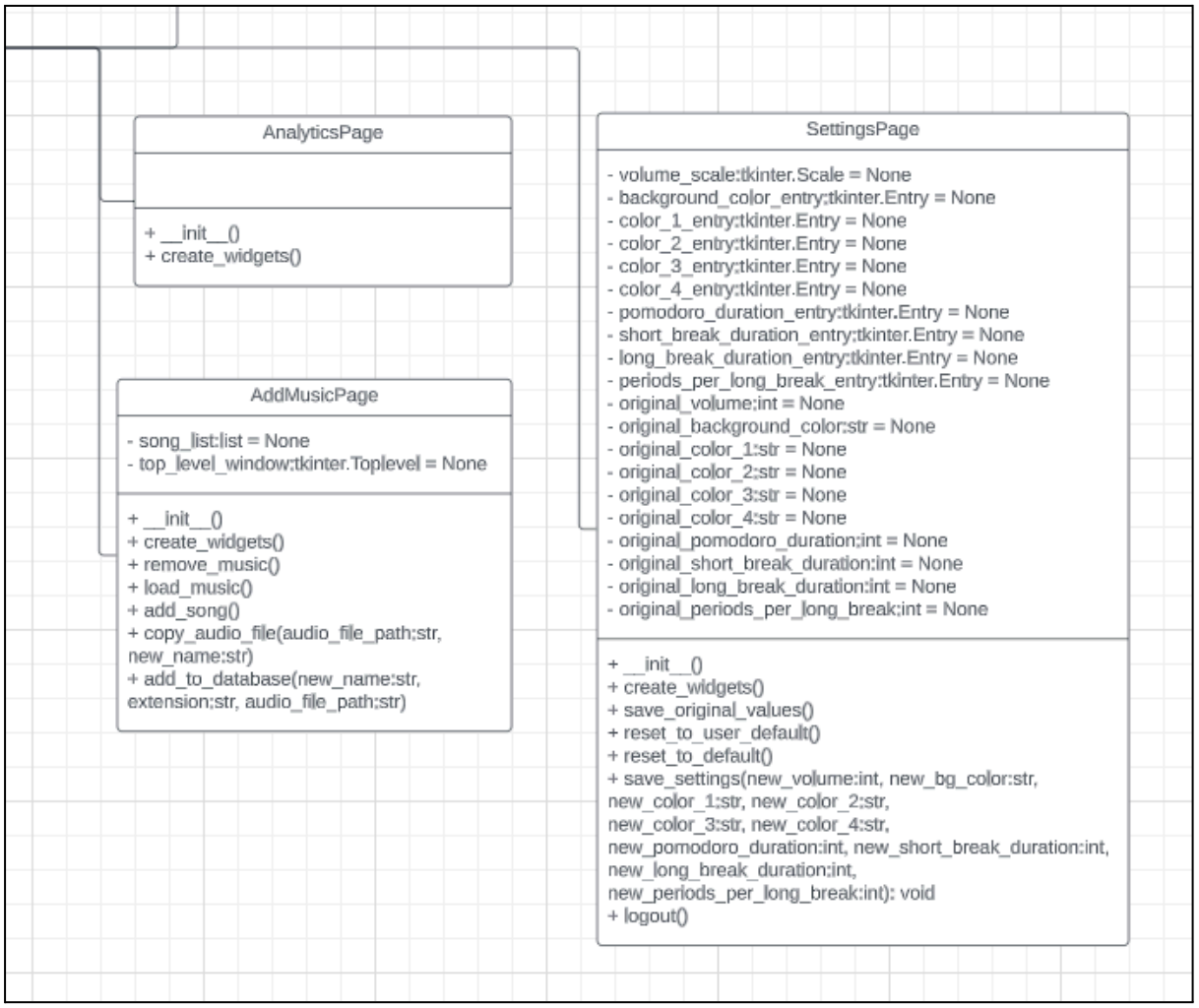


Image B.23.3

9 Data Dictionary

Table B.1 Users Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	id	Integer	Used as ID element for this table	1
2	username	Text	User's username	maxim_kudinov
3	user_password	Text	Password for user's account	Cctp^@7#I710
4	in_session	Binary	Determines whether or not the user is in session. One for true and zero for false	0

Table B.2 Settings Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	id	Integer	Used as ID element for this table	1
2	volume	Integer	Stores a measure for the user's preferred song volume	50
3	background_color	Varchar(7)	Stores the hexadecimal code for the background colour in the program	#FFFFFF
4	color_1	Varchar(7)	Stores the hexadecimal code for colour 1 in the program	#FFFFFF
5	color_2	Varchar(7)	Stores the hexadecimal code for colour 2 in the program	#FFFFFF
6	color_3	Varchar(7)	Stores the hexadecimal code for colour 3 in the program	#FFFFFF
7	color_4	Varchar(7)	Stores the hexadecimal code for colour 4 in the program	#FFFFFF
8	pomodoro_duration	Integer	Stores how long every study session is in minutes	25
9	short_break_duration	Integer	Stores how long every short break is in minutes	5
10	long_break_duration	Integer	Stores how long every long break is in minutes	10
11	periods_per_long_break	Integer	Stores the number of study sessions before a long break	3
12	user_id	Integer	Stores the user id of the user associated with this settings entry	1

Table B.3 Tasks Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	id	Integer	Used as ID element for this table	1
2	task_name	Text	The title of the task	Move to Italy
3	task_details	Text	The description of the task	Do so before December, change name to Giacomo
4	user_id	Integer	Stores the user id of the user associated with this task	2

Table B.4 Dates Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	id	Integer	Used as ID element for this table	1
2	year	Integer	The year this record was recorded	2023
3	month	Integer	The number of the month this record was recorded	7
4	day	Integer	The day this record was recorded	22
5	minutes_studied	Integer	The amount of minutes the associated user studied at that date	350
6	user_id	Integer	The id of the user associated with this record	2

Table B.5 Songs Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	id	Integer	Used as ID element for this table	1
2	title	Text	The song title	Arctic Monkeys - Do I Wanna Know
3	file_location	Text	The location of the file containing the song	resources/Music\Arctic Monkeys - Do I Wanna Know.ogg
4	duration	Integer	Stores how long the songs is in seconds	223

Table B.6 User_songs Table Data Dictionary

No.	Variable Name	Type	Description	Example
1	user_id	Integer	The id of the user associated with this record and song	1
2	song_id	Integer	The id of the song associated with this record and user	14

10 Test Plan

Table B.7 Test Plan Table

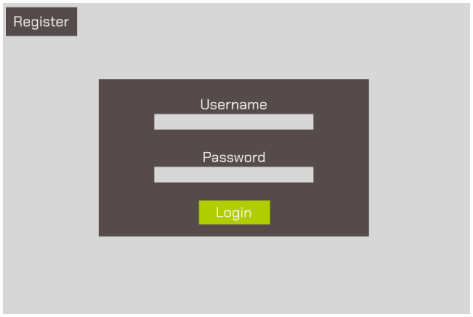
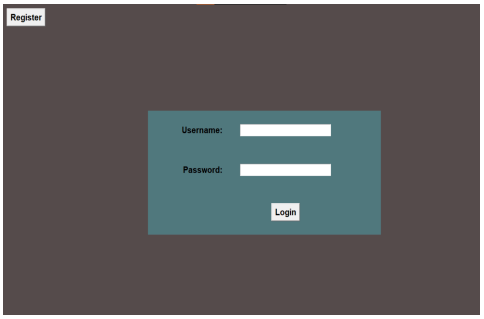
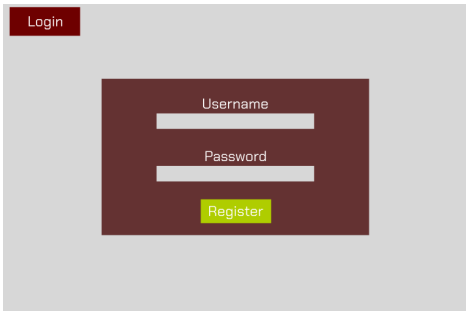
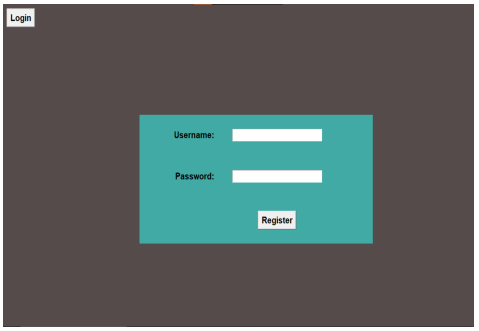
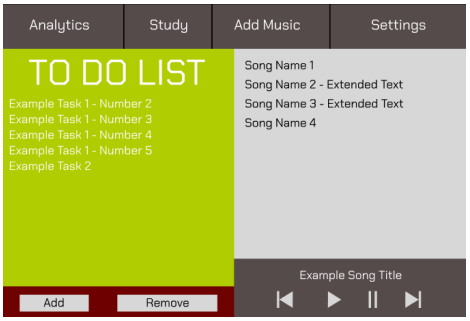
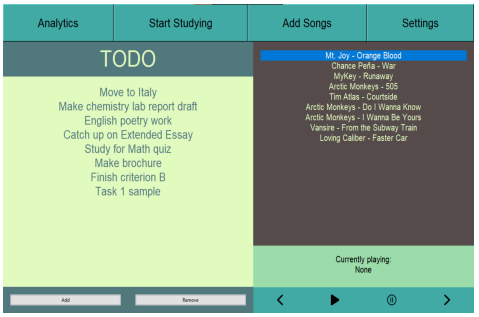
No	Description	Input Data/Instructions	Expected Results	Success Criteria (Refer to Section 4)
1	User Account Information Test	Add data to a user's account and check if any other users reflect that change. Also test by creating a new account to see if the new data was added there (A) Add a task to the "Justrene" account and check to see if the "Maxim" account contains that task.	All users must be independent entities, so no change should be observed to any other account (A) The task is not found in "Maxim"	7
2	Data Retention Test	Add data to the user's account and close the program. Then reopen it to see if the information is retained. (A) Add a task to the todo list (B) Remove a task from the todo list (C) Update colours in settings	The information should still be in the account for display for all types of data. (A) Task is accurately displayed after the application restarts (B) Task is no longer displayed after the application restarts (C) The colours in the settings change as well as the program's colour scheme	3, 4, 6
3	Data consistency	Update variables in one part of the application and check if the database accurately records it. (A) Change the volume in settings (B) Change the pomodoro timer in settings (C) Add a new song from the Add Music page (D) Remove a song from the Add Music page	The database records the change, which can be seen by the updation of the same record's displays or effects all over the program. (A) The music player's volume changes in accordance with the number (B) The timer counts down from the new duration (C) The Home page displays the new song (D) The Home page no longer displays the song	3, 4, 6

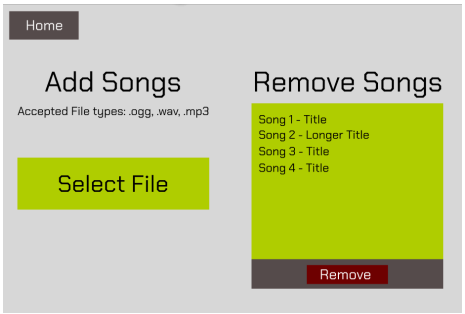
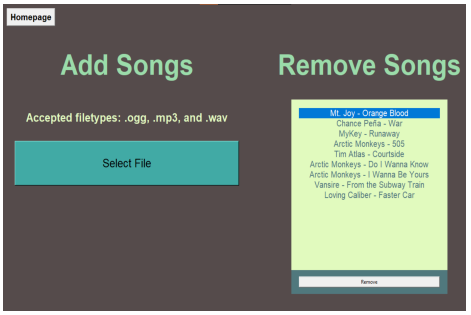
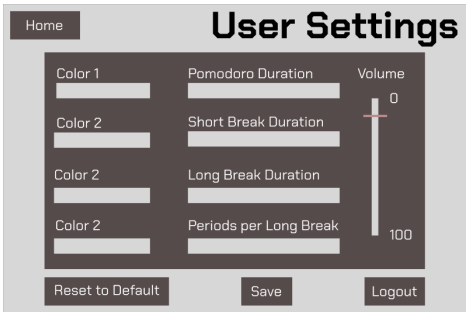
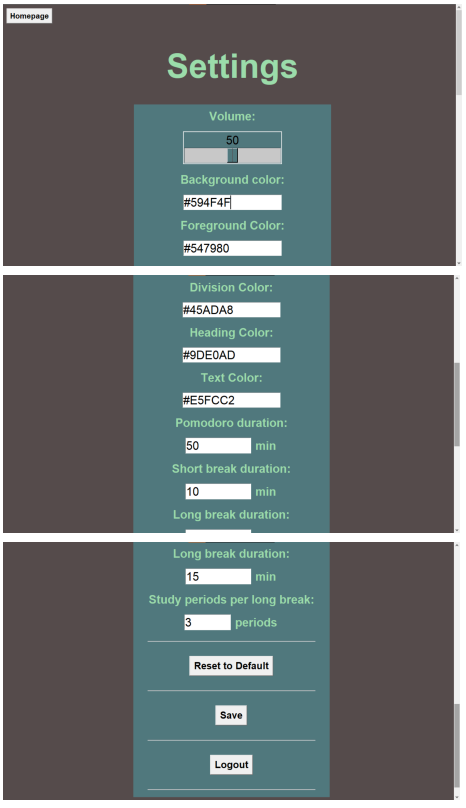
No	Description	Input Data/Instructions	Expected Results	Success Criteria (Refer to Section 4)
4	Erroneous Values in the Login and Register pages	Feed bad values for both pages and test the program's reaction. (A) Input an empty username in the Register page (B) Input an empty password in the Register page (C) Input an already existing name in the Register page (D) Input the wrong credentials in the Login page	The program responds accordingly to each case. (A) The program displays an error citing that the field cannot be empty. (B) The program displays an error citing that the field cannot be empty. (C) The program displays an error citing that the name has been taken. (D) The program displays an error citing that the username and password do not align.	7
5	Pomodoro timer variable reset test	Run the pomodoro timer several times and check if the counter resets when exiting the application. (A) In default settings, do 2 study sessions and close the program. Then, execute it again.	The counter should default to the full amount every time the program starts. (A) The program counter displays 3, the default value, instead of 1.	1
6	Pomodoro long break test	Run the pomodoro timer several times and check to see if the long break triggers after the set amount of periods	The long break should trigger as soon as the last period ends, and the cycle starts again	1
7	Button Functionality Test	Click all the buttons in the program	Each button leads to the appropriate screen or executes the desired command.	1, 2, 4, 6
8	Song Queue and History Test	After the player has been playing for a while, rewind to see which songs are played.	Songs should be played in reverse chronological order starting from the song last played.	2

No.	Description	Input Data/Instructions	Expected Results	Success Criteria (Refer to Section 4)
9	Double Click Test	Test to see if double clicks are properly binded throughout the program for specific uses. (A) Double click a song in the song library (B) Double click a task in the todo list	The appropriate function is executed or the appropriate screen is displayed (A) The double-clicked song plays immediately and fits into the song history (B) The task's description is displayed in a different window	2, 5
10	Test Analytics Page	Check to see if the analytics page immediately reflects the accurate number of minutes studied after the user finishes a study session.	The numbers are updated accordingly in the page	3
11	Deletion of Incomplete Data Entries	If an entry is incomplete, the database should not store that data. (A) Quit a study session in the middle. (B) Fill in a task but do not save it (C) Change the settings but do not save it	No changes should be made to the database (A) Analytics page does not change (B) Todo list does not change (C) Settings does not change after the user goes to another page and back	-
12	Test the Add Music Page File Explorer	Check to see if the file explorer can retrieve audio files in unconventional locations and put them in a secured folder	The explorer has access to the entire computer's file system and any audio file can be safely added to the program	6

11 Client Feedback for Graphical User Interface

Table C.1 Client Feedback for GUI

Screen	Prototype	Results	Feedback
Login			-
Register			-
Home			Final version has a better colour scheme
Analytics			-

Screen	Prototype	Results	Feedback
Studying			Colouring looks off, should be more distinct
Add Music			-
Settings			Final version is better due to the scroll function. It is also less cluttered.

*Refer to Appendix III for full transcript of client feedback

12 List of Techniques

Technique 1: Object-Oriented Programming

1. **Encapsulation**
    a. Accessors

```
class Song:
    def __init__(self, name, duration, location):
        self.name = name
        self.duration = duration
        self.location = location
        self.paused = False

    2 usages (2 dynamic)
    def play_song(self):...

    1 usage (1 dynamic)
    def resume_song(self):...

    1 usage (1 dynamic)
    def pause_song(self):...

    2 usages (2 dynamic)
    def get_name(self):  # Accessor for name
        return self.name

    2 usages (2 dynamic)
    def get_duration(self):  # Accessor for duration
        return self.duration
```

The Song class contains accessors so program methods can access its name or duration without endangering the data already in the class (Janssen).

b. Mutators

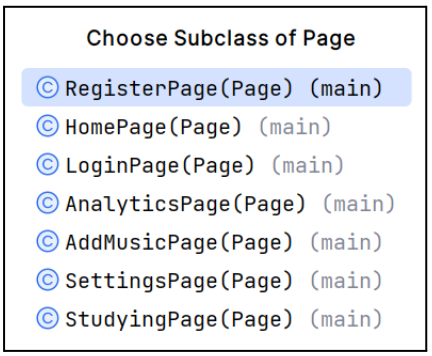
```
class Page(tk.Frame):
    def __init__(self):...

    2 usages
    def set_reset_studying(self, new_answer):
        if new_answer:
            self.reset_studying = True
        else:
            self.reset_studying = False

    2 usages
    def set_reset_home(self, new_answer):
        if new_answer:
            self.reset_home = True
        else:
            self.reset_home = False
```

The page class contains mutators to provide a simple interface for other parts of the application to modify key variables without endangering the other data.

2. Inheritance

```
class Page(tk.Frame):
    def __init__(self):
        super().__init__()
        # Connect to app, background color, and grid the frame
        self.app = App
        self.configure(bg=background_color)
        self.grid()

        # Boolean attributes to help reset pages
        self.reset_studying = False
        self.reset_home = False
        self.reset_settings = False

    2 usages
    def set_reset_studying(self, new_answer):...

    2 usages
    def set_reset_home(self, new_answer):...

    2 usages
    def go_to_analytics(self):...
```

```
2 usages
def go_to_add_music(self):...

2 usages
def go_to_settings(self):...

2 usages
def go_to_studying(self):...

7 usages (1 dynamic)
def go_to_homepage(self):...

4 usages
def go_to_login(self):...

2 usages
def go_to_register(self):...

1 usage
def reset_everything(self):...
```



The Page class is designed to include all the necessary attributes for all types of pages in the program, such as setting the app class, background, and grid. Additionally, the class incorporates methods to refresh or jump to any page. All other pages are subclasses of this class which shortens the code and opens the possibilities for new pages.

3. Complex Data Structures

a. Queues

```
class Queue:
    # Create custom queue object
    def __init__(self):
        self.song_queue = []

    1 usage
    def put(self, item):
        self.song_queue.append(item)

    18 usages (15 dynamic)
    def get(self):
        new_item = self.song_queue[0]
        self.song_queue.remove(new_item)
        return new_item

    1 usage
    def skip_queue(self, item):
        self.song_queue.insert(0, item)
```

```
new_song = random.choice(self.song_pool)
self.song_queue.put(new_song)
```

```
# Get a song from the queue
self.current_song = self.song_queue.get()
```

Queues are used to keep track of future songs to play in the music player. A custom class was made to incorporate the skip_queue() method, which puts an item in the front of the queue. This method was necessary to put songs back in the front of the queue when users skip to the previous song.

b. Stacks

```
self.history_stack = LifoQueue()  # To keep track of song history
```

```
# Update stack
self.history_stack.put(self.current_song)
```

```
# Get a song from the stack
self.current_song = self.history_stack.get(block=False)
```

Stacks are used for song history because the LIFO principle ensures that the most recently played song is retrieved first (GeeksforGeeks).

c. 2D Arrays

```
song_data = list()  # Initialize the song data array
```

```
song_data[:] = []  # Empty the song data array
for row in results:
    song_data.append(list(row))  # Append the list [title, file_location, duration] for every song
# song_data is now a 2d array, each row a song and each list in each row the song's attributes
for i in range(len(song_data)):
    new_song = Song(song_data[i][0], song_data[i][2], song_data[i][1])  # Create a Song object for each
    self.songs.append(new_song)  # Add to a new list of Song objects
```

A 2D array is created to map out each song's attributes from the database to later pass to the Song class.

Technique 2: Python and Algorithmic Thinking

1. Recursive Timekeeping

```
def scale_update(self):
    if self.timer_on:  # Ensures the timer is not turned off
        if self.song_scale < self.song_duration:  # Ensures time elapsed is less than the song duration
            self.song_scale += 1  # Keeps track of time elapsed
        else:
            self.next_music()  # Play next song if current one is over
    self.after(1000, self.scale_update)  # Time based recursion
```

This function creates an internal clock to monitor music progression. It utilises the after() method to call this function again after 1 second. Recursion gives the program a way to count the elapsed number of seconds to determine when a song ends (Halverson).

```
if not self.quit_activated:  # Ensures user has not quit the timer
    if clock_time > 0:  # Recurse while timer is not over
        self.after(1000, self.run_timer, clock_time - 1)  # Third argument is passed to second argument
    else:
        messagebox.showinfo("Session Over", "Congratulations, you have finished a session!")
        # Change an interface to reflect the timer's termination
        self.change_button.config(text="Start", command=self.run_timer)
        self.first_start = True  # Reset timer
else:
    self.quit_activated = False  # Timer terminates, but quit is reset
```

The Pomodoro timer's recursion works similarly but counts down the seconds instead. The function terminates if the remaining time is 0.

2. Conditional Statements

a. Nested if

```
if self.timer_on:  # Ensures the timer is not turned off
    if self.song_scale < self.song_duration:  # Ensures time elapsed is less than the song duration
        self.song_scale += 1  # Keeps track of time elapsed
    else:
        self.next_music()  # Play next song if current one is over
self.after(1000, self.scale_update)  # Time based recursion
```

Nested ifs create more complex conditions that can more effectively control the flow of the program. In this case, nested ifs are used to verify the conditions in which the next music is played.

b. Ladder if

```
# Copy the file to the destination directory in ogg format
if extension == ".ogg":
    shutil.copyfile(audio_file_path, destination_file_path)
elif extension == ".mp3":
    audio = AudioSegment.from_mp3(audio_file_path)
    audio.export(destination_file_path, format="ogg")
elif extension == ".wav":
    audio = AudioSegment.from_wav(audio_file_path)
    audio.export(destination_file_path, format="ogg")
```

```
# Get raw time in seconds
if session_type == 0:
    clock_time = int(user_settings[7]) * 60
elif session_type == 1:
    clock_time = int(user_settings[8]) * 60
elif session_type == 2:
    clock_time = int(user_settings[9]) * 60
else:
    print("Session type incorrectly inputted")
    clock_time = 0
```

Ladder ifs are useful for determining different paths of action based on different conditions. This technique dealt with different input situations, such as audio file types and upcoming session types.

3. For Loops

```
# Create and add frames to dictionary
for F in (HomePage, AnalyticsPage, AddMusicPage, SettingsPage, LoginPage, RegisterPage):
    frame = F()
    self.frames[F] = frame
    frame.grid(row=0, column=0, sticky="nsew")
```

The for loops shorten code by simultaneously generating instances of multiple classes at once. The same type of code is repeated all over the program where repetition is required.

4. **Lambda Functions**

```
self.song_list.bind("<Double-Button-1>", lambda event: self.double_click_song())
```

```
add_button = ttk.Button(task_window, text="Add",
                        command=lambda: self.add_task(self.title_entry.get(), self.description_entry.get(),
                                                      task_window))
```

Lambda functions are used because they are concise but sufficient to perform the simple task of passing arguments to a method, in this case, double_click_song() and add_task().

Technique 3: SQLite Database

1. **Database Structure**

Database Name: MaxQ
Database Tables:

- users
- dates
- tasks
- settings
- songs
- user_songs

```
(venv) PS C:\Users\Justrene\PycharmProjects\MaxQ> sqlite3 MaxQ.db
SQLite version 3.42.0 2023-05-16 12:36:15
```

```
sqlite> .tables
dates       settings    songs       tasks       user_songs  users
```

2. **Establishing Connection**

```
# Initialize database
connection = sqlite3.connect("MaxQ.db")
cursor = connection.cursor()
```

The statement establishes a connection with the SQLite3 database and initialises a cursor object to execute commands. If the database does not exist, it will be created and stored in the same folder as the program app.

3. Table Creation and Verification

```
try:
    query = '''
        CREATE TABLE user_songs(
            user_id INT NOT NULL,
            song_id INT NOT NULL,
            FOREIGN KEY(user_id) REFERENCES users(id),
            FOREIGN KEY(song_id) REFERENCES songs(id)
        );
    '''
    cursor.execute(query)
except sqlite3.Error as error:
    print("user_songs table found, proceeding...")
```

```
try:
    query = '''
        CREATE TABLE users(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT,
            user_password TEXT,
            in_session BINARY
        );
    '''
    cursor.execute(query)
except sqlite3.Error as error:
    print("users table found, proceeding...")
```

```
try:
    query = '''
        CREATE TABLE tasks(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            task_name TEXT NOT NULL,
            task_description TEXT,
            user_id INT,
            FOREIGN KEY(user_id) REFERENCES users(id)
        );
    '''
    cursor.execute(query)
except sqlite3.Error as error:
    print("tasks table found, proceeding...")
```

```
try:
    query = '''
        CREATE TABLE songs(
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            title TEXT,
            file_location TEXT,
            duration INT
        );
    '''
    cursor.execute(query)
except sqlite3.Error as error:
    print("songs table found, proceeding...")
```

For every table in the database, its creation query is stored and executed every time the program is run. If the table does not exist or is missing, it will be promptly created again. However, if the table already exists, the cursor.execute() method will raise an exception. This ensures that there are no duplicates or missing tables.

4. Database Queries

a. Select

```
cursor.execute("SELECT minutes_studied FROM dates WHERE year = ? AND month = ? AND day = ? "
               "AND user_id = ?", (year, month, day, current_user[0],))
results = cursor.fetchall()
```

This query retrieves information from the indicated columns in a table from all entries where the set condition is met. A similar query is also done to extract data from the database in different parts of the program.

b. Insert

```
cursor.execute("INSERT INTO users (username, user_password, in_session) VALUES (?, ?, 0)", (username,
                                                                                            password,))
connection.commit()
```

This query adds new records to tables by first specifying the intended columns. A similar query is done to insert data into the database in different parts of the program.

c. Update

```
cursor.execute("UPDATE users SET in_session = 1 WHERE id = ?", (results[0][0],))
connection.commit()
```

This query updates the user's status in the database. It is also used in areas where database entries have to be updated.

d. Delete

```
cursor.execute("DELETE FROM user_songs WHERE user_id = ? AND song_id IN (SELECT id FROM songs WHERE "
               "title = ?)", (current_user[0], name,))
cursor.execute("DELETE FROM songs WHERE title = ?", (name,))
```

This query deletes records from the database, such as when the user removes songs.

e. Create

```
query = '''
    CREATE TABLE settings(
        id INTEGER PRIMARY KEY AUTOINCREMENT,
        volume INT,
        background_color VARCHAR(7),
        color_1 VARCHAR(7),
        color_2 VARCHAR(7),
        color_3 VARCHAR(7),
        color_4 VARCHAR(7),
        pomodoro_duration INT,
        short_break_duration INT,
        long_break_duration INT,
        periods_per_long_break INT,
        user_id INT,
        FOREIGN KEY(user_id) REFERENCES users(id)
    );
'''
cursor.execute(query)
```

This query is used to recreate missing tables to make the program more foolproof.

Technique 4: Form Validation and Exception Handling

1. Integer Validation

```
# Validate information / Error handling
if not str(new_volume).isdigit() or not 0 <= new_volume <= 100:  # Must be an integer between 0 and 100
    messagebox.showinfo("Settings could not be saved", "Please recheck your input values")
    return
```

```
durations = [new_pomodoro_duration, new_short_break_duration, new_long_break_duration]
for duration in durations:  # Checks every duration value at once
    if not str(duration).isdigit() or not 0 <= int(duration) <= 120:  # Must be an integer between 0 and 120
        messagebox.showinfo("Settings could not be saved", "Please recheck your input values")
        return
if not str(new_periods_per_long_break).isdigit() or not 0 <= int(new_periods_per_long_break) <= 10:
    # Must be an integer between 0 and 10
    messagebox.showinfo("Settings could not be saved", "Please recheck your input values")
    return
```

Uses isdigit() as the most efficient way to check that the input is purely numerical and thus an integer.

2. Hexadecimal Validation

```
color_variables = [new_bg_color, new_color_1, new_color_2, new_color_3, new_color_4]
for color_variable in color_variables:  # Checks every single color filled in
    if color_variable.startswith('#'):  # First step: Must have a hashtag in front
        color_variable = color_variable[1:]  # Splice of the hashtag and check the rest
        try:
            int(color_variable, 16)  # Check if the remaining string is a valid hexadecimal number
        except ValueError:
            messagebox.showinfo("Settings could not be saved", "Please recheck your input values")
            return
    else:
        messagebox.showinfo("Settings could not be saved", "Please recheck your input values")
        return
```

First, startswith() is used to check that the string starts with a hashtag. Then the hashtag is spliced off and the program attempts to convert the remaining to a hexadecimal number with the int() function. When an error arises, the try-except block catches it.

3. Mandatory Fields Validation

```
def user_register(self):
    username = self.username_entry.get()  # Field for user entry
    password = self.password_entry.get()  # Field for user entry
    if not username or not password:  # If either fields are empty
        tk.messagebox.showinfo("Error", "Fields cannot be empty")  # An error raised
        return  # Quit method
```

All mandatory fields in the program have a validation process to ensure they are filled.

4. Unique Value Validation

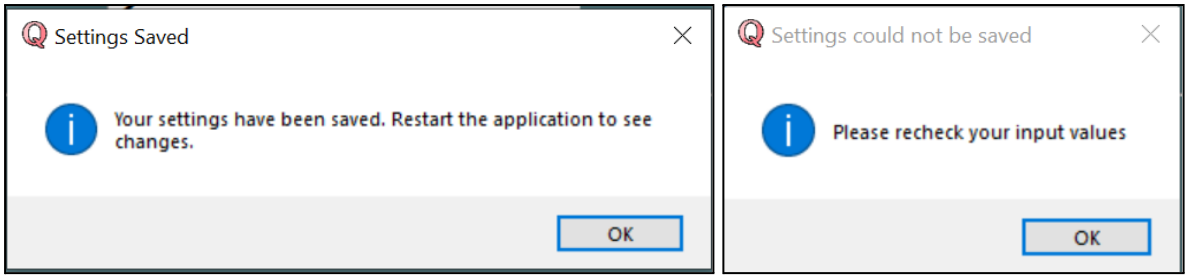
```
cursor.execute("SELECT id FROM users WHERE username = ?", (username,))  # Search for a user with the same name
results = cursor.fetchall()
if results:  # If the database finds a result
    tk.messagebox.showinfo("Error", "Username taken")  # Raise an error
```

Some database entries require a value unique from anything already in the database, so a validation process is set for each.

5. Notification Window

```
# Show a confirmation message
messagebox.showinfo("Settings Saved", "Your settings have been saved. Restart the application to see changes.")
```

The showinfo() method from the tkinter library displays the success status of various stages of the validation process.



Technique 5: Libraries

1. **pygame**

```
# Initialize pygame
pygame.init()
pygame.mixer.init()
```

```
def play_song(self):
    pygame.mixer.music.load(self.location)
    pygame.mixer.music.play()
```

```
def resume_song(self):
    self.paused = False
    pygame.mixer.music.unpause()
```

```
def pause_song(self):
    self.paused = True
    pygame.mixer.music.pause()
```

```
# Initialize pygame volume
pygame.mixer.music.set_volume(float(volume/100))
pygame.mixer.music.pause()
```

After initialisation, this library can load, play, pause, and unpause a song, as well as set its volume.

2. **pydub**

```
from pydub import AudioSegment
```

```
elif extension == ".mp3":
    audio = AudioSegment.from_mp3(audio_file_path)
    audio.export(destination_file_path, format="ogg")
elif extension == ".wav":
    audio = AudioSegment.from_wav(audio_file_path)
    audio.export(destination_file_path, format="ogg")
```

This library imports AudioSegment, which converts audio files into pygame's accepted filetype (.ogg).

```
# Find song duration
audio_file = AudioSegment.from_file(destination_file_path, format="ogg")
song_duration = len(audio_file) / 1000
```

It is also used to find the duration of a song.

3. datetime

```
# Get today's date
year = datetime.datetime.now().year
month = datetime.datetime.now().month
day = datetime.datetime.now().day
```

```
date = (datetime.datetime.now() - datetime.timedelta(days=(i + 1)))
```

This library gets the date to update the analytics database. It can also display the date *i* days ago for some *i* to keep track of the past.

4. json

```
# Import default settings from JSON file
with open("resources/settings.json", "r") as f:
    settings = json.load(f)
```

The library reads the default settings kept in a JSON file for security purposes.

5. queue

```
import queue
from queue import LifoQueue
```

```
self.history_stack = LifoQueue()  # To keep track of song history
```

```
# Update stack
self.history_stack.put(self.current_song)
```

```
# Get a song from the stack
self.current_song = self.history_stack.get(block=False)
```

Required library to import the stack object (LifoQueue). Its get() and put() methods effectively keep track of the user's play history.

6. shutil

```
# Copy the file to the destination directory in ogg format
if extension == ".ogg":
    shutil.copyfile(audio_file_path, destination_file_path)
```

Duplicates files into protected folder

7. **tkinter**

```
# Create content frame
frame = tk.Frame(self, bg=background_color)
frame.rowconfigure(0, weight=1)
frame.rowconfigure(1, weight=1)
frame.rowconfigure(2, weight=1)
frame.columnconfigure(0, weight=1)
frame.columnconfigure(1, weight=1)
frame.columnconfigure(2, weight=1)
frame.pack(side='top', expand=True, fill='both')
```

```
start_studying = tk.Button(menubar, text="Start Studying", font=('Arial', 20), bg=color_2,
                           activebackground=color_1, height=2, command=self.go_to_studying)
start_studying.grid(row=0, column=1, sticky="nsew")
```

Tkinter is used to instantiate the application and design the GUI. It is the backbone of this program.

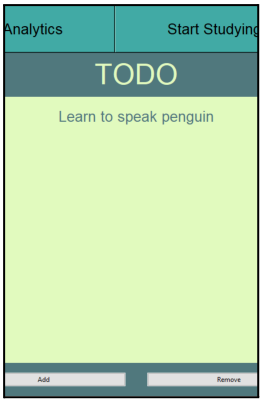
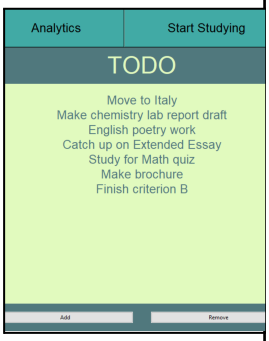
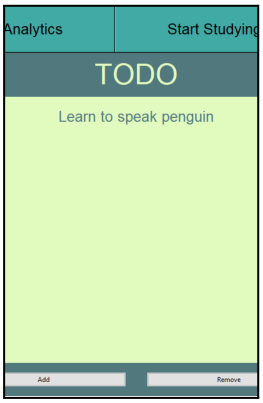
8. **Functions**
    a. to_hours

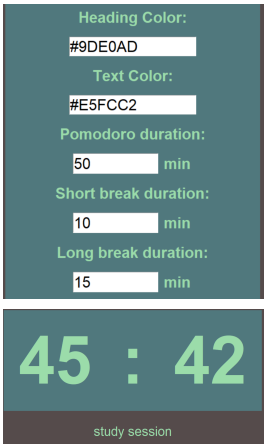
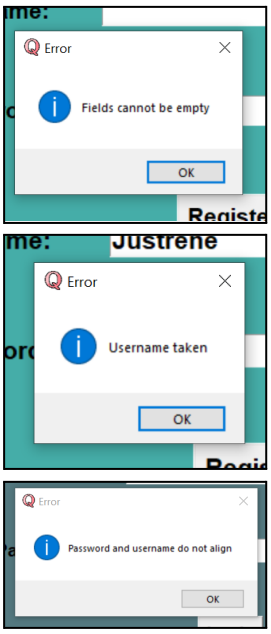
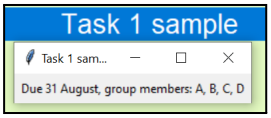
```
# Create vital functions
3 usages
def to_hours(minutes):
    spare = minutes % 60
    hours = (minutes - spare)/60
    return spare, int(hours)
```

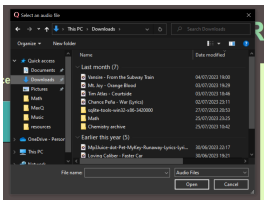
Function to split total minutes into hours and minutes

13 Test Results

Table C.2 Contrasting Test with Expected Results

No.	Proof	Description	Evaluation
1	"Justrene"  "Maxim"  	Added a task in "Justrene", not reflected in "Maxim".	Satisfactory
2	 	Task and setting changes maintained after app closure.	Satisfactory

No.	Proof	Description	Evaluation
3		Volume, timer, song changes reflected throughout. Settings change after restart.	Satisfactory
4		Manages empty fields, existing usernames, and bad credentials well.	Satisfactory
5		Break counter always 3 when break started	Satisfactory
6		Long break triggers properly	Satisfactory
7	-	All buttons functional	Satisfactory
8	-	Music player properly tracks songs	Satisfactory
9		Song plays when double clicked. Task displays description when double clicked.	Satisfactory

No.	Proof	Description	Evaluation
10		Analytics page updates instantaneously	Satisfactory
11	 (Recorded after user quits the timer halfway)	Settings, tasks, timer not updated if not completed or not saved	Satisfactory
12		File Explorer has access to all of the user's files	Satisfactory

*Refer to Section 10

14 Success Criteria Evaluation

Table E.1 Success Criteria Evaluation

No	Personal Evaluation	Client Feedback	Conclusion
1	The timer calls for break and study times accordingly, helping the client focus on his work while having breaks to maximise his productivity.	The client is satisfied with how the timer has paced his studying schedule, noticing considerable improvements to how long he studies daily.	Satisfactory
2	The music player works as intended, managing the client's downloaded files like a streaming service. He can listen to any song he wants freely, which motivates him.	The music player has increased his productivity by helping him focus. The feature also enables him to study more hours every day.	Satisfactory
3	The client maintains productivity by comparing himself to his previous study days even with varying study and break lengths, using the analytics page for consistency.	The client has set daily study goals that he is able to match with the analytics page, ensuring he stays on track in his syllabus.	Satisfactory
4	Being able to manage the timer lengths lets the user choose how intense their study sessions are. It works in tandem with the analytics page to make sure the user studies consistently everyday.	Controlling how long his breaks and study sessions are have encouraged him to push through on days when he has no motivation.	Satisfactory
5	The todo list space serves as a daily syllabus. It promotes focus and organisation by letting the client define an endpoint and the steps to get there.	The client uses this space to keep track of important lessons in progressive order to master a topic. The todo list has increased his study efficiency.	Satisfactory
6	The add music page filters audio from the user's computer, renames them, and stores them securely. It compliments the music player to make a streaming-like experience as requested.	The page allows him to curate a playlist from his own downloads, functioning as a streaming service to help his study focus and motivation.	Satisfactory
7	The login and register pages facilitate program extensibility for multiple users. Users are automatically logged in, enhancing the study experience, while allowing easy sharing of the program with others.	He is satisfied with the added extensibility for his friends or family while still being a non-distracting feature with the autologin function.	Satisfactory
8	The colour and font choices decrease battery usage and eye strain, whilst successfully promoting the user to focus on their work.	The colour choices promote focus by avoiding distractions, but the program's awkward proportions sometimes hinder ease of use.	Adequate

*Refer to Section 4 and Appendix IV

15 Future Developments

Client Recommendations

1. Put a playlist function in the music player for users with different "moods" while studying, allowing for a more personalised experience.
2. Show more information about the user's study analytics beyond the last 5 days. A search and filter function as well as line graphs to display the information were recommended. Visual representations communicate information better.
3. Implement a due date function in the to-do list linked to a schedule page that allows the user to view their tasks in chronological order.
4. Improve the stylistic choices in the GUIs of the Analytics and Studying Page. The client has recommended reducing empty spaces by increasing font size and adding divisions. Example improvements are shown below.
    a. Studying Page
        i. Current Version

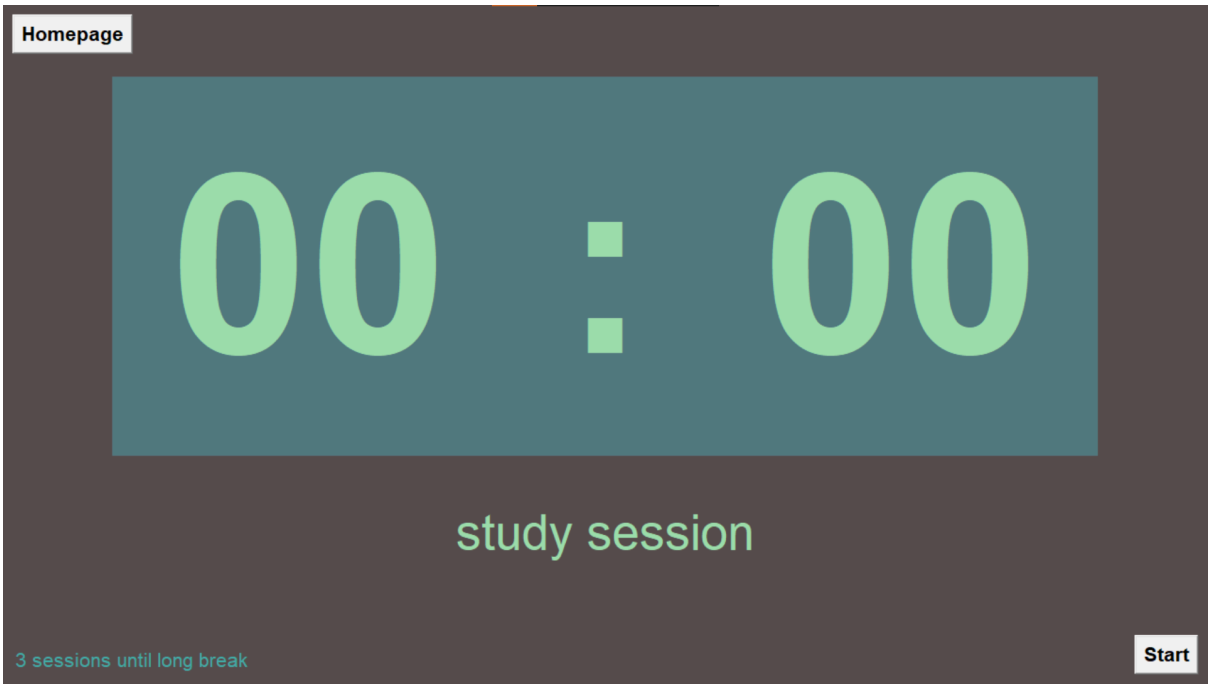


Image E.1

ii. Example Improvement

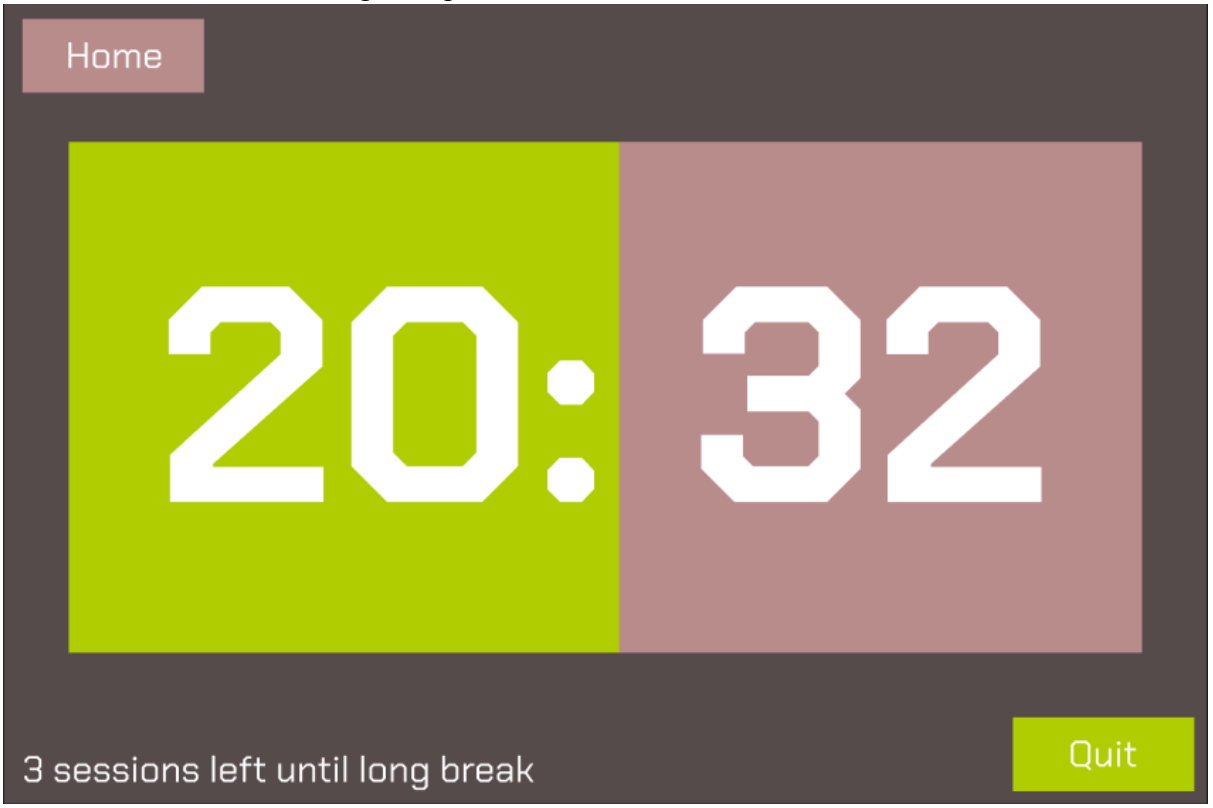


Image E.2

b. Analytics Page

i. Current Version

Image E.3

ii. Example Improvement

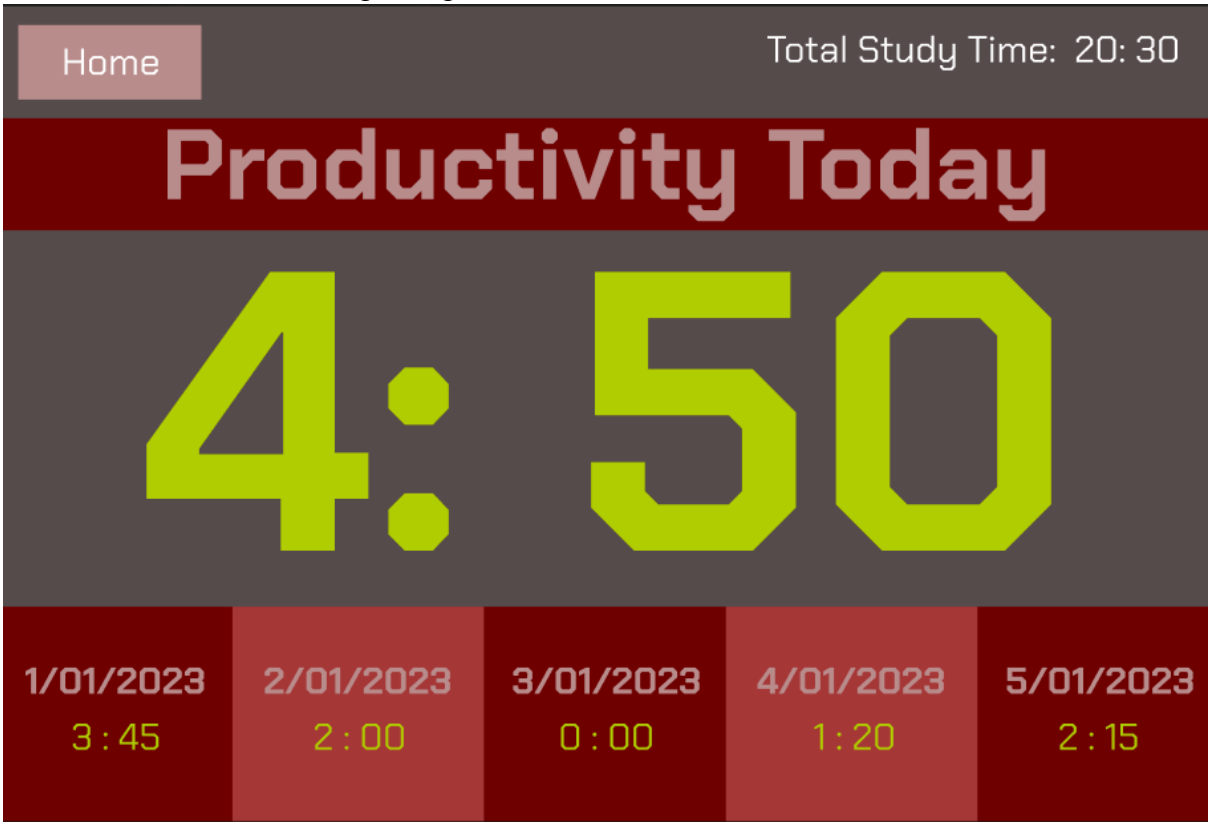


Image E.4

Developer Recommendations

1. Let users choose their own songs in the music player's queue, whilst still queuing random songs if the user does not make any additions. This allows for greater flexibility and personalisation.
2. Add a task priority feature. The to-do list is arranged based on these priorities, taking due dates into account. This allows for better time management and decision-making when learning a topic. The feature can possibly be implemented with an Eisenhower Matrix.
3. Arrange tasks in a calendar instead of a to-do list clipboard. This feature allows for better foresight for the user in planning their study routine.

Libraries

pygame
https://www.pygame.org/docs/
https://www.pygame.org/docs/ref/mixer.html
pydub
https://pypi.org/project/pydub/
https://audiosegment.readthedocs.io/en/latest/audiosegment.html
json
https://docs.python.org/3/library/json.html
queue
https://docs.python.org/3/library/queue.html
shutil
https://docs.python.org/3/library/shutil.html
tkinter
https://docs.python.org/3/library/tkinter.htm

Bibliography

"#6 Repeat and Autoplay Songs | Music Player In Tkinter | Tutorial on Music Player." *YouTube*, uploaded by CodeLoop - By Ritik, 20 Apr. 2022, https://www.youtube.com/watch?v=Ta7gPv9c1O8. Accessed 1 July 2023.

Halverson, Shaun. "How To Code A Countdown Timer In Python With Tkinter | Tutorial For Beginners | Visual Studio 2022." *YouTube*, uploaded by Shaun Halverson, 30 Dec. 2021, https://www.youtube.com/watch?v=k5mQwgSjKs8. Accessed 21 July 2023.

"How To Create Apps In Python Using Tkinter!" *YouTube*, YouTube, 31 Oct. 2022, https://www.youtube.com/watch?v=eoy9_S0sfP4. Accessed 13 July 2023.

Janssen, Thorben. "OOP Concept for Beginners: What Is Encapsulation." *Stackify*, 1 May 2023, stackify.com/oop-concept-for-beginners-what-is-encapsulation/.

"Preparing Environment For Pydub | Installation of FFmpeg & Microsoft Build Tools | Python Audio Text." *YouTube*, uploaded by Nileg Production, 16 Jan. 2022, https://www.youtube.com/watch?v=d2Y0lGrRoMI. Accessed 31 July 2023.

"SQLite Tutorial." *Tutorialspoint*, www.tutorialspoint.com/sqlite/. Accessed 25 July 2023.

"Stack Data Structure." *GeeksforGeeks*, 26 July 2023, www.geeksforgeeks.org/stack-data-structure/.

"Tkinter Beginner Course - Python GUI Development." *YouTube*, YouTube, 29 Sep. 2021, https://www.youtube.com/watch?v=ibf5cx221hk. Accessed 10 June 2023.