# **FOCUS TIMER - PROJECT DOCUMENTATION**

## **PROJECT STATEMENT**

**Focus Timer** is a comprehensive productivity application built with Python and Pygame that helps users manage their time effectively using the Pomodoro technique. The application combines timer functionality with website blocking, task management, analytics tracking, and multi-user support to create a complete productivity ecosystem. Users can track their focus sessions, manage tasks, analyze their productivity patterns, and compete with others through a leaderboard system.

## **AIM / OBJECTIVE OF THE PROJECT**

### **Primary Objectives:**

1. **Enhance User Productivity**: Implement Pomodoro timer technique with customizable focus/break intervals
2. **Eliminate Distractions**: Provide website blocking during focus sessions to maintain concentration
3. **Task Management**: Enable users to create, track, and complete tasks integrated with timer sessions
4. **Performance Analytics**: Track and visualize productivity metrics across time periods
5. **Multi-User Support**: Allow multiple users to maintain separate profiles and compete on leaderboards
6. **Gamification**: Motivate users through achievement tracking and competitive leaderboards

### **Secondary Objectives:**

* Secure user authentication with password hashing
* Persistent data storage using MySQL database
* Intuitive graphical interface with modern design
* Background music integration for focus sessions
* Cross-session data persistence and user session management

## 

## **USER TYPES AND OPERATIONS**

### **Single User Type: ALL USERS**

**Access Level**: All users have equal access to all features

**Note**: There is no administrator vs regular user distinction. Every user account has full access to all application features, including creating new users and managing accounts.

**Operations Available to All Users**:

1. **Authentication & Account Management**
   * Create new user accounts (via "Manage Users" in Settings)
   * Login to any existing account
   * Switch between user accounts
   * View all existing users
   * Session management (login required on app startup)
2. **Timer Management**
   * Start/Stop/Pause focus timer
   * Set custom timer duration (default 25 minutes)
   * View real-time countdown and progress ring
   * Track session history
3. **Task Management**
   * Create new tasks with descriptions
   * Mark tasks as complete/incomplete
   * Delete completed tasks
   * View task completion statistics (X/Y completed)
   * Tasks are user-specific (each user has their own task list)
4. **Website Blocking**
   * Add websites to personal block list
   * Remove websites from block list
   * Automatic blocking during active timer sessions
   * Block lists are user-specific (each user has their own blocked sites)
5. **Analytics & Statistics**
   * View daily focus time and session count
   * Track weekly productivity trends
   * Monitor monthly progress
   * View all-time statistics
   * Access global leaderboard (compare with all users)
   * Statistics are user-specific
6. **Settings & Configuration**
   * Switch between user accounts
   * Manage personal blocked websites
   * View application information
   * Access user management portal
   * Configure database connection (first-time setup)
7. **Music Control**
   * Play/pause background music
   * Adjust volume levels
   * Music player integration during focus sessions

### **Data Privacy**

While all users can create accounts and switch between them, user data remains private:

* Each user has their own tasks
* Each user has their own blocked websites
* Each user has their own session history
* Only leaderboard rankings are shared across users

## 

## 

## 

## **USER-DEFINED MODULES / FUNCTIONS**

### **Module 1: main.py - Application Entry Point**

**Purpose**: Main application launcher and home screen (Refer pg for code)

**Key Functions**:

1. **main()**
   * Initializes pygame and application window
   * Loads snowfall video background animation
   * Renders FOCUS text with hover effects
   * Handles navigation to Timer, Settings, and Analytics
   * Manages volume controls
   * Implements smooth animations and transitions
   * Continues running after subprocesses return (keeps session alive)
2. **get\_mysql\_config()**
   * Creates/loads MySQL configuration from mysql\_config.json
   * Provides GUI popup for MySQL credentials on first run
   * Validates database connection settings
   * Stores configuration persistently
3. **initialize\_database()**
   * Creates focus\_app database if not exists
   * Creates required tables: users, timer\_sessions, todos, blocked\_sites
   * Sets up table schemas with proper constraints
   * Handles connection errors gracefully
4. **get\_current\_user()**
   * Reads user session from current\_user.txt
   * Returns user\_id and username
   * Fallback to default user if file missing

**User Interaction Flow**:

* User starts application → Login required → FOCUS screen displayed
* Click "FOCUS" → Opens timer with current user
* Click Settings icon → Opens settings panel
* Click Analytics icon → Opens analytics dashboard
* All navigation preserves user session

### **Module 2: login\_screen.py - User Authentication**

**Purpose**: Handles user login and first-time setup (Refer pg for code)

**Key Functions**:

1. **login\_screen()**
   * Displays login interface with user selection
   * Shows list of available users from database
   * Handles password input (masked display)
   * Validates credentials against hashed passwords
   * Saves session to current\_user.txt on success
   * Redirects to user\_management if no users exist
2. **verify\_password(username, password)**
   * Retrieves user from database by username
   * Hashes input password using SHA-256
   * Compares hashed password with stored password\_hash
   * Returns user data on success, None on failure
3. **save\_current\_user(user\_id, username)**
   * Writes user session to text file
   * Format: Line 1: user\_id, Line 2: username
   * Enables session persistence across screens
4. **get\_all\_users()**
   * Queries database for all user accounts
   * Returns list of {id, username} dictionaries
   * Used to populate user selection dropdown

**User Interaction Flow**:

* No users exist → Auto-opens user\_management
* Users exist → Display login screen with user list
* User selects account → Enters password → Login successful → Save session

### **Module 3: user\_management.py - User Account Creation**

**Purpose**: Create and manage user accounts (Refer pg for code)

**Key Functions**:

1. **user\_management\_screen(first\_time=False)**
   * Displays user creation interface
   * Input fields for username and password
   * Shows existing users list
   * "Create User" button with validation
   * "Continue" button when users exist
   * Tab/Enter keyboard navigation
2. **create\_user(username, password)**
   * Validates username (min 3 characters, unique)
   * Validates password (min 4 characters)
   * Hashes password using SHA-256
   * Inserts new user into database
   * Returns success/failure message
   * Handles duplicate username errors
3. **hash\_password(password)**
   * Uses SHA-256 cryptographic hashing
   * Converts password to hexadecimal digest
   * Ensures secure password storage
4. **get\_database\_connection()**
   * Loads MySQL config from mysql\_config.json
   * Creates Database instance
   * Returns connection object or None

**User Interaction Flow**:

* First time (no users) → Welcome message → Create account → Continue to login
* Subsequent access → Add new users → View existing users → Back to settings

### **Module 4: timer.py - Focus Timer & Task Management**

**Purpose**: Core Pomodoro timer with integrated task list and website blocking

(Refer pg for code)

**Key Classes**:

1. **TimerApp(user\_id, username)**
   * Main timer application class
   * Manages timer state, UI, and user interactions

**Key Functions**:

1. **\_\_init\_\_(user\_id, username)**
   * Initializes timer with user context
   * Loads user's blocked sites from database
   * Creates TodoList instance for task management
   * Sets up video background and UI elements
   * Initializes database connection
2. **start\_timer()**
   * Begins countdown from set duration
   * Activates website blocking
   * Plays alert sound on start
   * Records session start time
   * Updates UI to show "STOP" button
3. **stop\_timer()**
   * Pauses/stops current session
   * Deactivates website blocking
   * Calculates elapsed time
   * Saves session to database
   * Updates UI to show "START" button
4. **activate\_blocking()**
   * Reads blocked sites for current user
   * Modifies Windows hosts file (requires admin)
   * Redirects blocked domains to 127.0.0.1
   * Creates backup of original hosts file
5. **deactivate\_blocking()**
   * Restores original hosts file from backup
   * Removes blocking entries
   * Re-enables blocked websites
6. **save\_session()**
   * Records completed session to database
   * Stores: user\_id, duration, completed\_tasks, timestamp
   * Updates analytics data
7. **handle\_events(event)**
   * Processes keyboard/mouse input
   * Start/stop button clicks
   * Back button navigation
   * Task list interactions
   * Settings button for blocked sites
8. **draw(screen)**
   * Renders timer display with large digits
   * Draws circular progress ring
   * Shows task list panel
   * Displays back button and controls
   * Updates visual elements each frame

**TodoList Sub-Module**:

1. **TodoList(x, y, width, height, user\_id, db)**
   * Task management panel within timer
2. **load\_todos()**
   * Fetches user's tasks from database
   * Populates task list
3. **save\_todos()**
   * Saves task state to database asynchronously
   * Updates task completion status
4. **handle\_event(event)**
   * Add new task input
   * Toggle task completion checkbox
   * Clear completed tasks button
5. **draw(screen)**
   * Renders task panel with header
   * Shows completed count (X/Y)
   * Displays individual tasks with checkboxes
   * "Clear completed" button

**User Interaction Flow**:

* Timer screen loads with user's tasks
* User enters custom duration (default 25:00)
* Click START → Timer begins, websites blocked, countdown starts
* User completes tasks (checkbox clicks)
* Click STOP → Timer pauses, session saved, websites unblocked
* Click BACK → Return to main screen

### **Module 5: settings.py - Application Settings**

**Purpose**: User preferences, website management, account switching (Refer pg for code)

**Key Functions**:

1. **settings\_screen()**
   * Main settings interface with tabbed navigation
   * Validates current user session
   * Four tabs: Account, Websites, Database, About
2. **Tab: Account**
   * **get\_current\_user()**: Displays logged-in user
   * **User switching**: Shows all users, click to switch
   * Updates current\_user.txt on account switch
   * Immediate account change without re-login
3. **Tab: Websites (Blocked Sites)**
   * **load\_blocked\_sites()**: Fetches user's blocked websites
   * **Add website**: Input field + "Add" button
   * **Remove website**: Click on website in list
   * **save\_blocked\_site(user\_id, site)**: Adds to database
   * **delete\_blocked\_site(user\_id, site)**: Removes from database
   * Real-time updates to blocked sites list
4. **Tab: Database**
   * **MySQL Configuration**: Display connection settings
   * **Connection Status**: Shows if database is connected
   * **Reconfigure**: Option to update MySQL credentials
   * Test connection functionality
5. **Tab: About**
   * Application name and version
   * Feature list
   * Developer credits
   * Technology stack information

**User Interaction Flow**:

* Open settings from main screen
* Navigate between tabs
* Account tab: Switch users by clicking on username
* Websites tab: Add/remove blocked sites
* Back button: Return to main screen

### **Module 6: AnalyticsDashboard.py - Productivity Analytics**

**Purpose**: Visualize user productivity metrics and leaderboard (Refer pg for code)

**Key Functions**:

1. **analytics\_dashboard()**
   * Main dashboard interface
   * Loads current user from session
   * Displays statistics and charts
2. **get\_user\_statistics(user\_id)**
   * Queries database for user's timer sessions
   * Calculates:
     + Total focus time (all sessions summed)
     + Total sessions count
     + Daily average focus time
     + Weekly statistics
     + Monthly statistics
3. **get\_leaderboard\_data()**
   * Aggregates all users' total focus time
   * Ranks users by total time
   * Returns top users with rankings
4. **draw\_time\_chart(screen, sessions)**
   * Visualizes focus time over last 7/30 days
   * Bar chart representation
   * X-axis: Dates, Y-axis: Minutes
5. **format\_time(seconds)**
   * Converts seconds to "Xh Ym" format
   * Handles display of duration statistics

**Display Panels**:

1. **Today's Stats**: Focus time and sessions today
2. **This Week**: Total time and sessions this week
3. **This Month**: Monthly productivity metrics
4. **All Time**: Lifetime statistics
5. **Leaderboard**: Rankings of all users
6. **Time Chart**: Visual representation of productivity trends

**User Interaction Flow**:

* View personal statistics (daily/weekly/monthly/all-time)
* Compare with other users on leaderboard
* Analyze productivity trends through charts
* Back to main screen

### **Module 7: database.py - Database Operations**

**Purpose**: MySQL database interface and data persistence (Refer pg for code)

**Key Class**: Database(mysql\_config)

**Key Functions**:

1. **\_\_init\_\_(mysql\_config)**
   * Establishes MySQL connection
   * Validates connection parameters
   * Creates database and tables if needed
2. **create\_tables()**
   * Creates users table
   * Creates timer\_sessions table
   * Creates todos table
   * Creates blocked\_sites table
   * Sets up indexes and constraints
3. **get\_todos(user\_id)**
   * Fetches all tasks for user
   * Returns list of {id, task, completed} dictionaries
4. **save\_todos(user\_id, todos)**
   * Deletes existing todos for user
   * Inserts updated todo list
   * Maintains task state
5. **get\_blocked\_sites(user\_id)**
   * Retrieves user's blocked websites
   * Returns list of site URLs
6. **add\_blocked\_site(user\_id, site)**
   * Inserts new blocked site for user
   * Prevents duplicates
7. **remove\_blocked\_site(user\_id, site)**
   * Deletes blocked site from database
8. **save\_session(user\_id, duration, completed\_tasks)**
   * Records timer session to database
   * Stores duration in seconds
   * Tracks task completion count
   * Timestamps session with current datetime
9. **get\_sessions(user\_id, start\_date=None, end\_date=None)**
   * Queries timer sessions for user
   * Optional date range filtering
   * Returns session history
10. **get\_user\_stats(user\_id)**
    * Aggregates statistics:
      + Total focus time
      + Total sessions
      + Average session duration
      + Streak data

**Connection Management**:

* Auto-reconnect on connection loss
* Error handling for all database operations
* Transaction support for data consistency

### **Module 8: admin\_launcher.py - Application Launcher**

**Purpose**: Ensures application runs with administrator privileges (Refer pg for code)

**Key Functions**:

1. **is\_admin()**
   * Checks if process has admin rights
   * Uses Windows API (ctypes)
   * Returns True/False
2. **run\_as\_admin()**
   * Relaunches script with admin privileges
   * Uses ShellExecuteW with "runas" verb
   * Required for hosts file modification (website blocking)
3. **main()**
   * Checks admin status
   * Requests elevation if needed
   * Runs main.py as subprocess
   * Displays warning if admin rights denied

**User Interaction Flow**:

* Batch file runs admin\_launcher.py
* Check admin privileges
* If not admin: Show UAC prompt
* If admin granted: Run main.py
* If admin denied: Show warning, continue with limited features

## **FILES AND DATABASE STRUCTURE**

### **Configuration Files**

#### **1. mysql\_config.json**

**Type**: JSON Configuration File

**Purpose**: Store MySQL database connection credentials

**Fields**:

{

"host": "localhost", // MySQL server address

"user": "root", // MySQL username

"password": "", // MySQL password

"database": "focus\_app", // Database name

"port": 3306 // MySQL port (default 3306)

}

**Usage**: Created on first run, loaded by database.py

#### **2. current\_user.txt**

**Type**: Plain Text File

**Purpose**: Store current user session

**Format**:

Line 1: user\_id (integer)

Line 2: username (string)

**Example**:

1

Amar

**Lifecycle**:

* Created on successful login
* Deleted on app startup (forces login)
* Read by all modules to identify current user
* Updated when user switches accounts in settings

### **Database: focus\_app**

**Database Name**: focus\_app

**Database Type**: MySQL

**Purpose**: Persistent storage for all application data

#### **Table 1: users**

**Purpose**: Store user accounts and credentials

**Structure**:

A close-up of a white background

AI-generated content may be incorrect.

**Fields**:

**A screenshot of a computer

AI-generated content may be incorrect.**

A screenshot of a computer

AI-generated content may be incorrect.

#### **Table 2: timer\_sessions**

**Purpose**: Record all completed focus timer sessions

**Structure**:

A screen shot of a computer code

AI-generated content may be incorrect.

**Fields**:

**A screenshot of a computer

AI-generated content may be incorrect.**

**Analytics Usage**:

* Sum of duration = Total focus time
* Count of records = Total sessions
* GROUP BY DATE(session\_date) = Daily statistics
* GROUP BY user\_id, ORDER BY SUM(duration) = Leaderboard

#### **Table 3: todos**

**Purpose**: Store user tasks and completion status

**Structure**:

A computer code with text

AI-generated content may be incorrect.

**Fields**:

**A screenshot of a computer

AI-generated content may be incorrect.Sample Data**:

A screenshot of a computer

AI-generated content may be incorrect.

**Task Management**:

* Load tasks: SELECT \* FROM todos WHERE user\_id = ?
* Save tasks: DELETE FROM todos WHERE user\_id = ?; INSERT INTO todos ...
* Completed count: SELECT COUNT(\*) FROM todos WHERE user\_id = ? AND completed = 1

#### **Table 4: blocked\_sites**

**Purpose**: Store websites blocked during focus sessions

**Structure**:

A screenshot of a computer code

AI-generated content may be incorrect.

**Fields**:

**A screenshot of a computer

AI-generated content may be incorrect.Sample Data**:

A screenshot of a computer

AI-generated content may be incorrect.

**Blocking Mechanism**:

* Load sites: SELECT site FROM blocked\_sites WHERE user\_id = ?
* Add to Windows hosts file: 127.0.0.1 youtube.com
* Remove from hosts file on timer stop
* Each user has independent block list

### **Media Files**

#### **Video Files**

* **snowfall.mp4**: Background animation for main screen
* Format: MP4 video
* Loops continuously using OpenCV
* Provides aesthetic winter theme

#### **Image Files**

* **guy.png**: Settings icon (character illustration)
* **Leaderboard.png**: Analytics button icon
* **font.ttf**: Custom font file for UI text

## **SYSTEM ARCHITECTURE**

### **Application Flow:**

Batch File (Focus.bat)

↓

admin\_launcher.py (Check/Request Admin Rights)

↓

main.py (Entry Point)

↓

Check current\_user.txt exists? → NO → login\_screen.py

↓ ↓

| Check users exist? → NO → user\_management.py

| ↓ ↓

| User Login Create Account

| ↓ ↓

| Save current\_user.txt ← ← ← ← ┘

↓ ↓

main() Function (FOCUS Screen) ← ← ← ← ← ┘

↓

User Actions:

- Click FOCUS → timer.py (user\_id, username)

- Click Settings → settings.py

- Click Analytics → AnalyticsDashboard.py

↓

All subprocesses exit back to main() loop

(Session persists in current\_user.txt)

### **Security Features:**

1. **Password Hashing**: SHA-256 cryptographic hashing
2. **No Plain Text Passwords**: Only hashes stored in database
3. **Session Management**: Temporary session file, cleared on restart
4. **Database Credentials**: Stored locally in JSON (not in code)
5. **Admin Privileges**: Required for website blocking (hosts file modification)

### **Data Persistence:**

* **MySQL Database**: All user data, sessions, tasks, blocked sites
* **Configuration Files**: mysql\_config.json, current\_user.txt
* **Session State**: Maintained across screens during app runtime
* **Analytics History**: Complete session history for trend analysis

## **TECHNICAL SPECIFICATIONS**

**Programming Language**: Python 3.x

**GUI Framework**: Pygame

**Database**: MySQL

**External Libraries**:

* pygame: Graphics and UI
* mysql-connector-python: Database connectivity
* cv2 (OpenCV): Video playback
* hashlib: Password hashing
* ctypes: Windows API access (admin privileges)
* subprocess: Process management
* json: Configuration management
* datetime: Timestamp handling

**Supported Platform**: Windows (hosts file modification requires Windows)

**Minimum Requirements**:

* Python 3.7+
* MySQL Server 5.7+
* Windows 7+
* Administrator privileges (for website blocking)
* 2GB RAM
* 100MB disk space

# **SOURCE CODE:**

## admin\_launcher.py:

import os

import sys

import ctypes

import subprocess

def is\_admin():

    """Check if the script is running with administrator privileges."""

    try:

        return ctypes.windll.shell32.IsUserAnAdmin()

    except:

        return False

def run\_as\_admin():

    """Relaunch the script with administrator privileges."""

    try:

        script = os.path.abspath(sys.argv[0])

        params = ' '.join([f'"{arg}"' for arg in sys.argv[1:]])

        # Use ShellExecute to run as admin

        ctypes.windll.shell32.ShellExecuteW(

            None, "runas", sys.executable, f'"{script}" {params}', None, 1

        )

        return True

    except Exception as e:

        print(f"Failed to run as admin: {e}")

        return False

def main():

    """Main launcher that ensures admin rights"""

    if not is\_admin():

        print("Not running as administrator. Requesting admin privileges...")

        if run\_as\_admin():

            print("Relaunching with admin rights...")

            sys.exit(0)

        else:

            print("Failed to get admin rights. Some features may not work.")

            # Continue anyway but show warning

            import tkinter as tk

            from tkinter import messagebox

            root = tk.Tk()

            root.withdraw()

            messagebox.showwarning(

                "Admin Rights Required",

                "Website blocking requires administrator privileges.\n\n"

                "The application will continue, but website blocking may not work.\n"

                "To enable blocking, please run as Administrator."

            )

            root.destroy()

    # Run main.py as subprocess (includes login at startup!)

    while True:

        try:

            import subprocess

            result = subprocess.run([sys.executable, "main.py"])

            # If main.py exits normally, break the loop

            if result.returncode == 0:

                break

            # If error, also break

            else:

                break

        except Exception as e:

            print(f"Error launching main application: {e}")

            input("Press Enter to exit...")

            break

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

    main()

## AnalyticsDashboard.py

A screenshot of a computer

AI-generated content may be incorrect.

import pygame

import sys

import subprocess

from database import Database

import datetime

import math

# Initialize pygame

pygame.init()

# Screen dimensions - fullscreen for better experience

SCREEN\_INFO = pygame.display.Info()

SCREEN\_WIDTH, SCREEN\_HEIGHT = SCREEN\_INFO.current\_w, SCREEN\_INFO.current\_h

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Study Analytics Dashboard")

# Colors - matching timer.py theme

BACKGROUND = (25, 25, 50)

PANEL\_BG = (40, 40, 80)

DARK\_PANEL = (30, 30, 60)

TEXT\_COLOR = (255, 255, 255)

HIGHLIGHT = (100, 200, 255)

ACCENT\_GOLD = (255, 215, 0)

ACCENT\_SILVER = (192, 192, 192)

BRONZE = (205, 127, 50)

BUTTON\_COLOR = (70, 130, 180)

BUTTON\_HOVER = (100, 160, 210)

SUCCESS\_GREEN = (100, 200, 150)

WARNING\_YELLOW = (255, 200, 0)

SHADOW\_COLOR = (15, 15, 30)

BORDER\_COLOR = (100, 100, 150)

TODO\_ITEM\_BG = (45, 45, 75)  # For dropdown items

# Fonts

title\_font = pygame.font.SysFont('timesnewroman', 60, bold=True)

header\_font = pygame.font.SysFont('timesnewroman', 36, bold=True)

text\_font = pygame.font.SysFont('timesnewroman', 24)

small\_font = pygame.font.SysFont('timesnewroman', 20)

tiny\_font = pygame.font.SysFont('timesnewroman', 16)

button\_font = pygame.font.SysFont('timesnewroman', 20, bold=True)

# Create database instance

db = Database()

class Button:

    def \_\_init\_\_(self, x, y, width, height, text, color=BUTTON\_COLOR, hover\_color=BUTTON\_HOVER):

        self.rect = pygame.Rect(x, y, width, height)

        self.text = text

        self.color = color

        self.hover\_color = hover\_color

        self.is\_hovered = False

    def draw(self, screen):

        # Shadow

        shadow = pygame.Rect(self.rect.x + 4, self.rect.y + 4,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Button

        color = self.hover\_color if self.is\_hovered else self.color

        pygame.draw.rect(screen, color, self.rect, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=12)

        text\_surface = button\_font.render(self.text, True, TEXT\_COLOR)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def is\_clicked(self, pos):

        return self.rect.collidepoint(pos)

    def update\_hover(self, pos):

        self.is\_hovered = self.rect.collidepoint(pos)

def get\_current\_user():

    """Get current user from file"""

    try:

        with open("current\_user.txt", "r") as f:

            lines = f.readlines()

            if len(lines) >= 2:

                user\_id = int(lines[0].strip())

                username = lines[1].strip()

                return user\_id, username

    except:

        pass

    return 1, "Default User"

def get\_all\_users():

    """Get all users from database"""

    try:

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            cursor.execute("SELECT id, username FROM users ORDER BY username")

            users = cursor.fetchall()

            cursor.close()

            return users

    except Exception as e:

        print(f"Error loading users: {e}")

    return []

def switch\_user(user\_id, username):

    """Switch to a different user"""

    try:

        with open("current\_user.txt", "w") as f:

            f.write(f"{user\_id}\n{username}\n")

        return True

    except:

        return False

def format\_time(minutes):

    """Format minutes into readable time"""

    if minutes is None:

        return "0m"

    minutes = int(minutes)

    if minutes < 60:

        return f"{minutes}m"

    else:

        hours = minutes // 60

        mins = minutes % 60

        if mins == 0:

            return f"{hours}h"

        else:

            return f"{hours}h {mins}m"

def get\_personal\_analytics(user\_id):

    """Get personal analytics data from database"""

    if db.connection is None:

        print("Database connection not available")

        return None

    try:

        analytics = db.get\_study\_analytics(user\_id)

        return analytics

    except Exception as e:

        print(f"Error getting analytics data: {e}")

        return None

def draw\_stat\_card(screen, x, y, width, height, title, value, subtitle="", icon=""):

    """Draw an improved stat card with shadow and icon"""

    # Shadow

    shadow\_rect = pygame.Rect(x + 5, y + 5, width, height)

    pygame.draw.rect(screen, SHADOW\_COLOR, shadow\_rect, border\_radius=12)

    # Card background

    card\_rect = pygame.Rect(x, y, width, height)

    pygame.draw.rect(screen, PANEL\_BG, card\_rect, border\_radius=12)

    pygame.draw.rect(screen, BORDER\_COLOR, card\_rect, 2, border\_radius=12)

    # Icon (if provided)

    if icon:

        icon\_surface = header\_font.render(icon, True, HIGHLIGHT)

        screen.blit(icon\_surface, (x + 15, y + 15))

        title\_x = x + 60

    else:

        title\_x = x + 15

    # Title

    title\_surface = small\_font.render(title, True, HIGHLIGHT)

    screen.blit(title\_surface, (title\_x, y + 20))

    # Main value (centered)

    value\_surface = header\_font.render(str(value), True, TEXT\_COLOR)

    value\_rect = value\_surface.get\_rect(center=(x + width//2, y + height//2 + 10))

    screen.blit(value\_surface, value\_rect)

    # Subtitle

    if subtitle:

        subtitle\_surface = tiny\_font.render(subtitle, True, ACCENT\_SILVER)

        screen.blit(subtitle\_surface, (x + width//2 - subtitle\_surface.get\_width()//2, y + height - 25))

def draw\_progress\_bar(screen, x, y, width, height, progress, color=HIGHLIGHT):

    """Draw a horizontal progress bar"""

    # Background

    bg\_rect = pygame.Rect(x, y, width, height)

    pygame.draw.rect(screen, DARK\_PANEL, bg\_rect, border\_radius=5)

    # Fill

    if progress > 0:

        fill\_width = int(width \* min(progress, 1.0))

        fill\_rect = pygame.Rect(x, y, fill\_width, height)

        pygame.draw.rect(screen, color, fill\_rect, border\_radius=5)

    # Border

    pygame.draw.rect(screen, BORDER\_COLOR, bg\_rect, 1, border\_radius=5)

def draw\_mini\_chart(screen, x, y, width, height, data\_points, color=HIGHLIGHT):

    """Draw a simple line chart"""

    if not data\_points or len(data\_points) < 2:

        return

    # Background

    chart\_rect = pygame.Rect(x, y, width, height)

    pygame.draw.rect(screen, DARK\_PANEL, chart\_rect, border\_radius=8)

    # Find min/max for scaling

    max\_val = max(data\_points) if data\_points else 1

    min\_val = min(data\_points) if data\_points else 0

    range\_val = max\_val - min\_val if max\_val != min\_val else 1

    # Draw points and lines

    points = []

    for i, val in enumerate(data\_points):

        px = x + (i / (len(data\_points) - 1)) \* width

        py = y + height - ((val - min\_val) / range\_val) \* (height - 20) - 10

        points.append((int(px), int(py)))

    # Draw line

    if len(points) > 1:

        pygame.draw.lines(screen, color, False, points, 3)

    # Draw points

    for point in points:

        pygame.draw.circle(screen, color, point, 5)

        pygame.draw.circle(screen, TEXT\_COLOR, point, 3)

def draw\_analytics\_dashboard(screen, user\_id, username, analytics, all\_users=None, show\_user\_dropdown=False):

    """Draw the enhanced analytics dashboard"""

    screen.fill(BACKGROUND)

    # Header section with gradient effect

    header\_height = 120

    pygame.draw.rect(screen, DARK\_PANEL, (0, 0, SCREEN\_WIDTH, header\_height))

    # Title (moved right to avoid overlap with back button)

    title\_text = title\_font.render("STUDY ANALYTICS", True, HIGHLIGHT)

    screen.blit(title\_text, (200, 25))

    # User info (now a dropdown/clickable area)

    user\_info\_text = f"Student: {username}"

    user\_info = text\_font.render(user\_info\_text, True, ACCENT\_SILVER)

    user\_info\_rect = user\_info.get\_rect(topleft=(200, 85))

    screen.blit(user\_info, user\_info\_rect)

    # Dropdown indicator

    dropdown\_arrow = tiny\_font.render("▼", True, ACCENT\_SILVER)

    screen.blit(dropdown\_arrow, (user\_info\_rect.right + 10, 87))

    # Store clickable area for user dropdown

    global user\_dropdown\_rect

    user\_dropdown\_rect = pygame.Rect(200, 85, user\_info\_rect.width + 30, 25)

    # Draw user dropdown menu if active

    if show\_user\_dropdown and all\_users:

        dropdown\_width = 300

        dropdown\_height = min(len(all\_users) \* 40 + 10, 400)

        dropdown\_x = 200

        dropdown\_y = 115

        # Shadow

        shadow\_rect = pygame.Rect(dropdown\_x + 4, dropdown\_y + 4, dropdown\_width, dropdown\_height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow\_rect, border\_radius=10)

        # Dropdown background

        dropdown\_rect = pygame.Rect(dropdown\_x, dropdown\_y, dropdown\_width, dropdown\_height)

        pygame.draw.rect(screen, PANEL\_BG, dropdown\_rect, border\_radius=10)

        pygame.draw.rect(screen, BORDER\_COLOR, dropdown\_rect, 2, border\_radius=10)

        # Draw user options

        global user\_option\_rects

        user\_option\_rects = []

        for i, user in enumerate(all\_users[:10]):  # Show max 10 users

            option\_y = dropdown\_y + 5 + i \* 40

            option\_rect = pygame.Rect(dropdown\_x + 5, option\_y, dropdown\_width - 10, 35)

            user\_option\_rects.append((option\_rect, user['id'], user['username']))

            # Highlight current user or hovered

            if user['id'] == user\_id:

                pygame.draw.rect(screen, HIGHLIGHT, option\_rect, border\_radius=8)

                text\_color = BACKGROUND

            else:

                pygame.draw.rect(screen, TODO\_ITEM\_BG, option\_rect, border\_radius=8)

                text\_color = TEXT\_COLOR

            # User name

            user\_text = small\_font.render(user['username'], True, text\_color)

            screen.blit(user\_text, (option\_rect.x + 10, option\_rect.y + 8))

    # Current date

    date\_text = tiny\_font.render(datetime.datetime.now().strftime("%B %d, %Y"), True, ACCENT\_SILVER)

    screen.blit(date\_text, (SCREEN\_WIDTH - 200, 90))

    if not analytics or not analytics.get('records'):

        # No data state - improved

        no\_data\_y = SCREEN\_HEIGHT // 2 - 100

        # Message

        no\_data\_text = header\_font.render("No Study Data Yet", True, TEXT\_COLOR)

        screen.blit(no\_data\_text, (SCREEN\_WIDTH//2 - no\_data\_text.get\_width()//2, no\_data\_y + 20))

        instruction = text\_font.render("Start using the focus timer to track your study sessions!", True, ACCENT\_SILVER)

        screen.blit(instruction, (SCREEN\_WIDTH//2 - instruction.get\_width()//2, no\_data\_y + 80))

        # Tips box

        tips\_y = no\_data\_y + 140

        tips\_width = 600

        tips\_rect = pygame.Rect(SCREEN\_WIDTH//2 - tips\_width//2, tips\_y, tips\_width, 100)

        pygame.draw.rect(screen, PANEL\_BG, tips\_rect, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, tips\_rect, 2, border\_radius=12)

        tip1 = small\_font.render("• Complete study sessions to see analytics", True, TEXT\_COLOR)

        tip2 = small\_font.render("• Rate your focus to track performance", True, TEXT\_COLOR)

        screen.blit(tip1, (SCREEN\_WIDTH//2 - tip1.get\_width()//2, tips\_y + 25))

        screen.blit(tip2, (SCREEN\_WIDTH//2 - tip2.get\_width()//2, tips\_y + 60))

        return

    # Stats cards section

    cards\_y = header\_height + 30

    card\_width = 280

    card\_height = 140

    spacing = 30

    start\_x = (SCREEN\_WIDTH - (4 \* card\_width + 3 \* spacing)) // 2

    # Card 1: Total Study Time

    total\_time = analytics['records'].get('total\_study\_time', 0) or 0

    draw\_stat\_card(screen, start\_x, cards\_y, card\_width, card\_height,

                   "Total Study Time", format\_time(total\_time), "All time", "")

    # Card 2: Total Sessions

    total\_sessions = analytics['records'].get('total\_sessions', 0) or 0

    draw\_stat\_card(screen, start\_x + card\_width + spacing, cards\_y, card\_width, card\_height,

                   "Study Sessions", str(total\_sessions), "Completed", "")

    # Card 3: Average Session

    avg\_session = analytics['records'].get('avg\_session\_length', 0) or 0

    draw\_stat\_card(screen, start\_x + 2\*(card\_width + spacing), cards\_y, card\_width, card\_height,

                   "Avg Session", format\_time(int(avg\_session)), "Per session", "")

    # Card 4: Best Focus

    best\_focus = analytics['records'].get('best\_focus', 0) or 0

    focus\_stars = "★" \* int(best\_focus) if best\_focus > 0 else "—"

    draw\_stat\_card(screen, start\_x + 3\*(card\_width + spacing), cards\_y, card\_width, card\_height,

                   "Peak Focus", focus\_stars, f"{best\_focus:.1f}/5" if best\_focus > 0 else "Not rated", "")

    # Weekly trend section

    trend\_y = cards\_y + card\_height + 40

    # Left side - Weekly chart

    chart\_section\_width = SCREEN\_WIDTH // 2 - 60

    # Section header

    trend\_header = header\_font.render("Weekly Trends", True, HIGHLIGHT)

    screen.blit(trend\_header, (50, trend\_y))

    if analytics.get('weekly\_data') and len(analytics['weekly\_data']) > 0:

        chart\_y = trend\_y + 60

        chart\_height = 250

        # Chart background panel

        chart\_panel = pygame.Rect(50, chart\_y, chart\_section\_width, chart\_height + 80)

        pygame.draw.rect(screen, PANEL\_BG, chart\_panel, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, chart\_panel, 2, border\_radius=12)

        # Extract data for chart

        weekly\_minutes = [week.get('total\_minutes', 0) or 0 for week in analytics['weekly\_data'][:6]]

        # Draw mini chart

        draw\_mini\_chart(screen, 70, chart\_y + 20, chart\_section\_width - 40, 180, weekly\_minutes)

        # Labels (moved up to avoid overlap)

        label\_y = chart\_y + 205

        for i in range(min(6, len(analytics['weekly\_data']))):

            label\_x = 70 + (i / 5) \* (chart\_section\_width - 40) if len(analytics['weekly\_data']) > 1 else chart\_section\_width // 2

            week\_label = tiny\_font.render(f"W{analytics['weekly\_data'][i]['week']}", True, ACCENT\_SILVER)

            screen.blit(week\_label, (int(label\_x) - 10, label\_y))

        # Summary stats below chart (with more spacing)

        summary\_y = chart\_y + 250

        if len(weekly\_minutes) > 0:

            avg\_weekly = sum(weekly\_minutes) / len(weekly\_minutes)

            trend\_text = tiny\_font.render(f"Avg: {format\_time(int(avg\_weekly))}/week", True, ACCENT\_SILVER)

            screen.blit(trend\_text, (60, summary\_y))

            # Trend indicator (moved down)

            if len(weekly\_minutes) >= 2:

                if weekly\_minutes[-1] > weekly\_minutes[-2]:

                    trend\_icon = "↗ Improving"

                    trend\_color = SUCCESS\_GREEN

                elif weekly\_minutes[-1] < weekly\_minutes[-2]:

                    trend\_icon = "↘ Declining"

                    trend\_color = WARNING\_YELLOW

                else:

                    trend\_icon = "→ Stable"

                    trend\_color = ACCENT\_SILVER

                trend\_surf = tiny\_font.render(trend\_icon, True, trend\_color)

                screen.blit(trend\_surf, (60, summary\_y + 22))

    # Right side - Completed Tasks

    tasks\_x = SCREEN\_WIDTH // 2 + 30

    tasks\_header = header\_font.render("Completed Tasks", True, HIGHLIGHT)

    screen.blit(tasks\_header, (tasks\_x, trend\_y))

    # Tasks panel background

    tasks\_panel\_y = trend\_y + 60

    tasks\_panel\_height = 420

    tasks\_panel\_width = SCREEN\_WIDTH // 2 - 60

    tasks\_panel = pygame.Rect(tasks\_x, tasks\_panel\_y, tasks\_panel\_width, tasks\_panel\_height)

    # Shadow

    tasks\_shadow = pygame.Rect(tasks\_panel.x + 5, tasks\_panel.y + 5,

                               tasks\_panel.width, tasks\_panel.height)

    pygame.draw.rect(screen, SHADOW\_COLOR, tasks\_shadow, border\_radius=12)

    # Panel background

    pygame.draw.rect(screen, PANEL\_BG, tasks\_panel, border\_radius=12)

    pygame.draw.rect(screen, BORDER\_COLOR, tasks\_panel, 2, border\_radius=12)

    # Get tasks from database

    completed\_tasks = []

    if db and db.connection:

        try:

            cursor = db.connection.cursor(dictionary=True)

            # Try to get tasks with completed\_at column

            try:

                query = """

                    SELECT task, completed\_at

                    FROM todos

                    WHERE user\_id = %s

                    AND completed = 1

                    ORDER BY id DESC

                    LIMIT 10

                """

                cursor.execute(query, (user\_id,))

                completed\_tasks = cursor.fetchall()

            except:

                # Fallback if completed\_at doesn't exist - just get completed tasks

                query = """

                    SELECT task

                    FROM todos

                    WHERE user\_id = %s

                    AND completed = 1

                    ORDER BY id DESC

                    LIMIT 10

                """

                cursor.execute(query, (user\_id,))

                completed\_tasks = cursor.fetchall()

            cursor.close()

        except Exception as e:

            print(f"Error loading tasks: {e}")

            completed\_tasks = []

    # Draw tasks inside panel

    if completed\_tasks and len(completed\_tasks) > 0:

        tasks\_list\_y = tasks\_panel\_y + 20

        for i, task\_data in enumerate(completed\_tasks[:8]):  # Show top 8

            task\_y\_pos = tasks\_list\_y + i \* 48

            task\_width = tasks\_panel\_width - 30

            # Task card

            task\_rect = pygame.Rect(tasks\_x + 15, task\_y\_pos, task\_width, 40)

            pygame.draw.rect(screen, TODO\_ITEM\_BG, task\_rect, border\_radius=8)

            pygame.draw.rect(screen, BORDER\_COLOR, task\_rect, 1, border\_radius=8)

            # Checkmark

            check\_text = text\_font.render("✓", True, SUCCESS\_GREEN)

            screen.blit(check\_text, (tasks\_x + 25, task\_y\_pos + 8))

            # Task name

            task\_name = task\_data.get('task', 'Unknown task')

            if len(task\_name) > 30:

                task\_name = task\_name[:30] + "..."

            name\_surf = small\_font.render(task\_name, True, TEXT\_COLOR)

            screen.blit(name\_surf, (tasks\_x + 55, task\_y\_pos + 10))

    else:

        # No completed tasks - centered message

        no\_tasks\_text = text\_font.render("No completed tasks yet", True, ACCENT\_SILVER)

        no\_tasks\_rect = no\_tasks\_text.get\_rect(center=(tasks\_panel.centerx, tasks\_panel.centery))

        screen.blit(no\_tasks\_text, no\_tasks\_rect)

    # Best day insight at bottom

    insight\_y = SCREEN\_HEIGHT - 100

    if analytics.get('best\_day') and analytics['best\_day'] and analytics['best\_day'].get('day\_name'):

        insight\_width = 700

        insight\_rect = pygame.Rect(SCREEN\_WIDTH//2 - insight\_width//2, insight\_y, insight\_width, 70)

        # Shadow

        shadow = pygame.Rect(insight\_rect.x + 4, insight\_rect.y + 4, insight\_rect.width, insight\_rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Background

        pygame.draw.rect(screen, PANEL\_BG, insight\_rect, border\_radius=12)

        pygame.draw.rect(screen, ACCENT\_GOLD, insight\_rect, 2, border\_radius=12)

        # Text

        best\_day = analytics['best\_day']['day\_name']

        insight\_text = text\_font.render(f"Your most productive day: {best\_day}", True, ACCENT\_GOLD)

        screen.blit(insight\_text, (insight\_rect.x + 30, insight\_rect.y + 15))

        tip\_text = tiny\_font.render("Schedule important study sessions on this day for best results!", True, ACCENT\_SILVER)

        screen.blit(tip\_text, (insight\_rect.x + 30, insight\_rect.y + 45))

def main():

    # Get current user

    user\_id, username = get\_current\_user()

    # Get all users for dropdown

    all\_users = get\_all\_users()

    # Get analytics data

    analytics = get\_personal\_analytics(user\_id)

    clock = pygame.time.Clock()

    # Create buttons

    back\_button = Button(30, 30, 140, 50, "← Back")

    refresh\_button = Button(SCREEN\_WIDTH - 170, 30, 140, 50, "Refresh")

    # Dropdown state

    show\_user\_dropdown = False

    # Global variables for click detection

    global user\_dropdown\_rect, user\_option\_rects

    user\_dropdown\_rect = None

    user\_option\_rects = []

    running = True

    while running:

        mouse\_pos = pygame.mouse.get\_pos()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

            elif event.type == pygame.MOUSEBUTTONDOWN:

                if back\_button.is\_clicked(mouse\_pos):

                    pygame.quit()

                    try:

                        subprocess.run([sys.executable, "main.py"]); sys.exit()  # Restart main

                    except Exception as e:

                        return

                elif refresh\_button.is\_clicked(mouse\_pos):

                    analytics = get\_personal\_analytics(user\_id)

                    all\_users = get\_all\_users()

                # Check user dropdown toggle

                elif user\_dropdown\_rect and user\_dropdown\_rect.collidepoint(mouse\_pos):

                    show\_user\_dropdown = not show\_user\_dropdown

                # Check user selection from dropdown

                elif show\_user\_dropdown and user\_option\_rects:

                    for rect, uid, uname in user\_option\_rects:

                        if rect.collidepoint(mouse\_pos):

                            if uid != user\_id:

                                # Switch user

                                if switch\_user(uid, uname):

                                    user\_id = uid

                                    username = uname

                                    analytics = get\_personal\_analytics(user\_id)

                            show\_user\_dropdown = False

                            break

                else:

                    # Click outside dropdown - close it

                    show\_user\_dropdown = False

        # Update button hover states

        back\_button.update\_hover(mouse\_pos)

        refresh\_button.update\_hover(mouse\_pos)

        # Draw everything with error handling

        try:

            draw\_analytics\_dashboard(screen, user\_id, username, analytics, all\_users, show\_user\_dropdown)

            back\_button.draw(screen)

            refresh\_button.draw(screen)

        except Exception as e:

            # If drawing fails, show error message

            screen.fill(BACKGROUND)

            error\_title = header\_font.render("Error Loading Dashboard", True, WARNING\_YELLOW)

            error\_msg = text\_font.render(str(e), True, TEXT\_COLOR)

            screen.blit(error\_title, (SCREEN\_WIDTH//2 - error\_title.get\_width()//2, SCREEN\_HEIGHT//2 - 50))

            screen.blit(error\_msg, (SCREEN\_WIDTH//2 - error\_msg.get\_width()//2, SCREEN\_HEIGHT//2))

            back\_button.draw(screen)

            print(f"Dashboard error: {e}")

            import traceback

            traceback.print\_exc()

        pygame.display.flip()

        clock.tick(30)

    pygame.quit()

    sys.exit()

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

    main()

## auth.py

import hashlib

from tkinter import simpledialog, messagebox, Tk

from database import db

USER\_FILE = "user\_data.txt"  # We'll keep this for migration

def hash\_password(password):

    return hashlib.sha256(password.encode()).hexdigest()

def create\_account(root):

    """Create new account in MySQL database"""

    while True:

        username = simpledialog.askstring("Create Account", "Choose a username:", parent=root)

        if not username:

            return None

        # Check if username already exists

        if db.user\_exists(username):

            messagebox.showerror("Error", "Username already exists. Choose a different one.", parent=root)

            continue

        pwd1 = simpledialog.askstring("Create Password", "Enter a new password:", show='\*', parent=root)

        if not pwd1:

            return None

        pwd2 = simpledialog.askstring("Confirm Password", "Confirm your password:", show='\*', parent=root)

        if pwd1 != pwd2:

            messagebox.showerror("Error", "Passwords do not match. Try again.", parent=root)

        else:

            # Create user in database

            user\_id = db.create\_user(username, pwd1)

            if user\_id:

                messagebox.showinfo("Success", "Account created successfully!", parent=root)

                return {"id": user\_id, "username": username}

            else:

                messagebox.showerror("Error", "Failed to create account. Please try again.", parent=root)

                return None

def verify\_password(root):

    """Verify user against MySQL database"""

    # First, check if we need to migrate from file-based system

    migrate\_from\_file()

    # Check if any users exist in database

    if not db.user\_exists():

        return create\_account(root)

    username = simpledialog.askstring("Login", "Enter your username:", parent=root)

    if not username:

        return None

    password = simpledialog.askstring("Login", "Enter your password:", show='\*', parent=root)

    if not password:

        return None

    user\_data = db.verify\_user(username, password)

    if user\_data:

        return user\_data

    else:

        messagebox.showerror("Error", "Invalid username or password!", parent=root)

        return None

def migrate\_from\_file():

    """Migrate from file-based authentication to database"""

    import os

    import json

    if os.path.exists(USER\_FILE) and not db.user\_exists():

        try:

            with open(USER\_FILE, 'r') as f:

                content = f.read().strip()

            # Try to parse as JSON (new format)

            try:

                user\_data = json.loads(content)

                username = user\_data.get('username', 'User')

                password\_hash = user\_data.get('password\_hash', '')

                # Create user in database

                if username and password\_hash:

                    # We can't recover the password, so create with default password

                    db.create\_user(username, "password123")

                    print("✅ Migrated user from file to database")

            except json.JSONDecodeError:

                # Old format - just the password hash

                password\_hash = content

                db.create\_user("User", "password123")

                print("✅ Migrated user from old file format to database")

            # Backup the old file

            os.rename(USER\_FILE, USER\_FILE + ".backup")

        except Exception as e:

            print(f"❌ Error migrating from file: {e}")

## config.py

import os

from PIL import ImageFont

# Global font variables

app\_font = ("Segoe UI", 10)

app\_font\_bold = ("Segoe UI", 10, "bold")

app\_font\_large = ("Segoe UI", 12)

def load\_fonts():

    """Load custom fonts for the application"""

    global app\_font, app\_font\_bold, app\_font\_large

    try:

        # Try to load the custom TTF font

        font\_path = "font.ttf"

        if os.path.exists(font\_path):

            print("Custom TTF font found: font.ttf")

            # For TTF files with tkinter, we use the font file directly

            app\_font = ("Arial", 10)  # Using Arial as fallback for custom font

            app\_font\_bold = ("Arial", 10, "bold")

            app\_font\_large = ("Arial", 12)

            # Test loading with PIL (for any image/text operations)

            try:

                pil\_font = ImageFont.truetype(font\_path, 12)

                print(f"Successfully loaded TTF font with PIL: {font\_path}")

            except Exception as e:

                print(f"PIL font loading error: {e}")

        else:

            print("Using system fonts (font.ttf not found)")

    except Exception as e:

        print(f"Error loading font: {e}")

# Load fonts when module is imported

load\_fonts()

## database.py

import mysql.connector

from datetime import datetime, timedelta

import hashlib

import os

import time

class Database:

    def \_\_init\_\_(self, config=None):

        # Use provided config or default

        if config:

            self.config = config

        else:

            # Try to load from config file if exists

            config\_file = "mysql\_config.json"

            if os.path.exists(config\_file):

                try:

                    import json

                    with open(config\_file, 'r') as f:

                        config = json.load(f)

                        self.config = config

                except:

                    self.config = {

                        'host': 'localhost',

                        'user': 'root',

                        'password': '',

                        'database': 'focus\_app',

                        'port': 3306,

                        'auth\_plugin': 'mysql\_native\_password'

                    }

            else:

                self.config = {

                    'host': 'localhost',

                    'user': 'root',

                    'password': '',

                    'database': 'focus\_app',

                    'port': 3306,

                    'auth\_plugin': 'mysql\_native\_password'

                }

        self.connection = None

        self.connect()

    def connect(self):

        """Connect to MySQL database with retry logic"""

        max\_retries = 3

        retry\_delay = 2  # seconds

        for attempt in range(max\_retries):

            try:

                print(f"Attempting to connect to MySQL database (attempt {attempt + 1}/{max\_retries})...")

                self.connection = mysql.connector.connect(\*\*self.config)

                print("✅ Connected to local MySQL database")

                self.initialize\_database()

                return True

            except mysql.connector.Error as err:

                print(f"❌ Database connection error: {err}")

                if attempt < max\_retries - 1:

                    print(f"Retrying in {retry\_delay} seconds...")

                    time.sleep(retry\_delay)

                else:

                    # Last attempt failed, try to create database

                    print("All connection attempts failed. Trying to create database...")

                    return self.create\_database()

        return False

    def create\_database(self):

        """Create database and tables if they don't exist"""

        try:

            print("Attempting to create database...")

            # First, try to connect without database specified

            temp\_config = self.config.copy()

            if 'database' in temp\_config:

                temp\_config.pop('database')

            # Try with and without password

            temp\_configs\_to\_try = []

            # Try current password

            temp\_configs\_to\_try.append(temp\_config)

            # Try empty password if current one isn't empty

            if temp\_config.get('password', ''):

                temp\_config\_no\_pass = temp\_config.copy()

                temp\_config\_no\_pass['password'] = ''

                temp\_configs\_to\_try.append(temp\_config\_no\_pass)

            # Try common passwords

            common\_passwords = ['password', '123456', 'root', 'admin']

            for common\_pass in common\_passwords:

                temp\_config\_common = temp\_config.copy()

                temp\_config\_common['password'] = common\_pass

                temp\_configs\_to\_try.append(temp\_config\_common)

            conn = None

            for test\_config in temp\_configs\_to\_try:

                try:

                    print(f"Trying connection with password: {'\*' \* len(test\_config.get('password', '')) if test\_config.get('password') else '(empty)'}")

                    conn = mysql.connector.connect(\*\*test\_config)

                    print(f"✅ Connected with password: {'\*' \* len(test\_config.get('password', '')) if test\_config.get('password') else '(empty)'}")

                    break

                except mysql.connector.Error:

                    continue

            if not conn:

                print("❌ Could not connect to MySQL with any password. Please check your MySQL installation.")

                print("\nTroubleshooting steps:")

                print("1. Make sure MySQL service is running")

                print("2. Try default MySQL installation credentials:")

                print("   - Username: root")

                print("   - Password: (empty) or 'password' or 'root'")

                print("3. If you set a custom password, edit mysql\_config.json")

                return False

            cursor = conn.cursor()

            # Create database if it doesn't exist

            cursor.execute("CREATE DATABASE IF NOT EXISTS focus\_app")

            cursor.execute("USE focus\_app")

            # Create users table

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS users (

                    id INT PRIMARY KEY AUTO\_INCREMENT,

                    username VARCHAR(50) UNIQUE NOT NULL,

                    password\_hash VARCHAR(255) NOT NULL,

                    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

                )

            """)

            # Create study\_sessions table (enhanced for analytics)

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS study\_sessions (

                    id INT PRIMARY KEY AUTO\_INCREMENT,

                    user\_id INT NOT NULL,

                    start\_time DATETIME NOT NULL,

                    end\_time DATETIME,

                    duration\_minutes INT,

                    focus\_rating INT,  # 1-5 stars

                    subject\_tag VARCHAR(50),  # What were you studying?

                    notes TEXT,

                    distractions\_count INT DEFAULT 0,

                    FOREIGN KEY (user\_id) REFERENCES users(id)

                )

            """)

            # Create blocked\_sites table

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS blocked\_sites (

                    id INT PRIMARY KEY AUTO\_INCREMENT,

                    user\_id INT NOT NULL,

                    website VARCHAR(255) NOT NULL,

                    added\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

                    FOREIGN KEY (user\_id) REFERENCES users(id),

                    UNIQUE KEY unique\_user\_site (user\_id, website)

                )

            """)

            # Create todos table for task management

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS todos (

                    id INT PRIMARY KEY AUTO\_INCREMENT,

                    user\_id INT NOT NULL,

                    task VARCHAR(500) NOT NULL,

                    completed BOOLEAN DEFAULT FALSE,

                    created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

                    FOREIGN KEY (user\_id) REFERENCES users(id)

                )

            """)

            # Create personal\_records table for achievements

            cursor.execute("""

                CREATE TABLE IF NOT EXISTS personal\_records (

                    id INT PRIMARY KEY AUTO\_INCREMENT,

                    user\_id INT NOT NULL,

                    record\_type VARCHAR(50) NOT NULL,  # 'longest\_session', 'best\_focus', etc.

                    record\_value FLOAT NOT NULL,

                    achieved\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

                    FOREIGN KEY (user\_id) REFERENCES users(id),

                    UNIQUE KEY unique\_user\_record (user\_id, record\_type)

                )

            """)

            conn.commit()

            cursor.close()

            conn.close()

            # Update config with database name

            self.config['database'] = 'focus\_app'

            # Now connect to the new database with updated config

            self.connection = mysql.connector.connect(\*\*self.config)

            print("✅ Database and tables created successfully")

            return True

        except mysql.connector.Error as err:

            print(f"❌ Error creating database: {err}")

            print("\n⚠️  IMPORTANT: MySQL Setup Required")

            print("=" \* 40)

            print("Please ensure MySQL is installed and running.")

            print("\nFor Windows:")

            print("1. Install MySQL from mysql.com or use XAMPP")

            print("2. Start MySQL service")

            print("3. Common default passwords: (empty), 'password', 'root'")

            print("\nThe app will create a 'mysql\_config.json' file")

            print("if you need to specify custom credentials.")

            return False

    def initialize\_database(self):

        """Initialize database by creating tables if they don't exist"""

        cursor = self.connection.cursor()

        try:

            # Check if study\_sessions table exists

            cursor.execute("SHOW TABLES LIKE 'study\_sessions'")

            if not cursor.fetchone():

                # Create all tables

                self.create\_database()

            else:

                # Check if study\_sessions table has focus\_rating column

                cursor.execute("SHOW COLUMNS FROM study\_sessions LIKE 'focus\_rating'")

                if not cursor.fetchone():

                    # Add focus\_rating column if it doesn't exist

                    try:

                        cursor.execute("ALTER TABLE study\_sessions ADD COLUMN focus\_rating INT")

                        cursor.execute("ALTER TABLE study\_sessions ADD COLUMN subject\_tag VARCHAR(50)")

                        cursor.execute("ALTER TABLE study\_sessions ADD COLUMN distractions\_count INT DEFAULT 0")

                        self.connection.commit()

                        print("✅ Added analytics columns to study\_sessions table")

                    except mysql.connector.Error as err:

                        print(f"Note: Could not add columns (might already exist): {err}")

        except mysql.connector.Error as err:

            print(f"Error checking tables: {err}")

            # Try to create database

            self.create\_database()

        cursor.close()

    def hash\_password(self, password):

        """Hash password for storage"""

        return hashlib.sha256(password.encode()).hexdigest()

    def create\_user(self, username, password):

        """Create a new user"""

        try:

            cursor = self.connection.cursor()

            password\_hash = self.hash\_password(password)

            cursor.execute(

                "INSERT INTO users (username, password\_hash) VALUES (%s, %s)",

                (username, password\_hash)

            )

            self.connection.commit()

            user\_id = cursor.lastrowid

            cursor.close()

            return user\_id

        except mysql.connector.Error as err:

            print(f"Error creating user: {err}")

            return None

    def verify\_user(self, username, password):

        """Verify user credentials"""

        try:

            cursor = self.connection.cursor(dictionary=True)

            password\_hash = self.hash\_password(password)

            cursor.execute(

                "SELECT id, username FROM users WHERE username = %s AND password\_hash = %s",

                (username, password\_hash)

            )

            user = cursor.fetchone()

            cursor.close()

            return user

        except mysql.connector.Error as err:

            print(f"Error verifying user: {err}")

            return None

    def get\_user\_by\_username(self, username):

        """Get user by username"""

        try:

            cursor = self.connection.cursor(dictionary=True)

            cursor.execute(

                "SELECT id, username, created\_at FROM users WHERE username = %s",

                (username,)

            )

            user = cursor.fetchone()

            cursor.close()

            return user

        except mysql.connector.Error as err:

            print(f"Error getting user: {err}")

            return None

    def get\_all\_users(self):

        """Get all users (for login screen)"""

        try:

            cursor = self.connection.cursor()

            cursor.execute("SELECT id, username FROM users ORDER BY username")

            users = cursor.fetchall()

            cursor.close()

            return users

        except mysql.connector.Error as err:

            print(f"Error getting users: {err}")

            return []

    def update\_username(self, user\_id, new\_username):

        """Update username"""

        try:

            cursor = self.connection.cursor()

            cursor.execute(

                "UPDATE users SET username = %s WHERE id = %s",

                (new\_username, user\_id)

            )

            self.connection.commit()

            cursor.close()

            return cursor.rowcount > 0

        except mysql.connector.Error as err:

            print(f"Error updating username: {err}")

            return False

    def update\_password(self, user\_id, new\_password):

        """Update password"""

        try:

            cursor = self.connection.cursor()

            password\_hash = self.hash\_password(new\_password)

            cursor.execute(

                "UPDATE users SET password\_hash = %s WHERE id = %s",

                (password\_hash, user\_id)

            )

            self.connection.commit()

            cursor.close()

            return cursor.rowcount > 0

        except mysql.connector.Error as err:

            print(f"Error updating password: {err}")

            return False

    def delete\_user(self, user\_id):

        """Delete user and all their data"""

        try:

            cursor = self.connection.cursor()

            # Delete in correct order (due to foreign keys)

            cursor.execute("DELETE FROM blocked\_sites WHERE user\_id = %s", (user\_id,))

            cursor.execute("DELETE FROM study\_sessions WHERE user\_id = %s", (user\_id,))

            cursor.execute("DELETE FROM personal\_records WHERE user\_id = %s", (user\_id,))

            cursor.execute("DELETE FROM users WHERE id = %s", (user\_id,))

            self.connection.commit()

            cursor.close()

            return True

        except mysql.connector.Error as err:

            print(f"Error deleting user: {err}")

            return False

    # Blocked sites management

    def add\_blocked\_site(self, user\_id, website):

        """Add a website to block list"""

        try:

            cursor = self.connection.cursor()

            cursor.execute(

                "INSERT INTO blocked\_sites (user\_id, website) VALUES (%s, %s)",

                (user\_id, website)

            )

            self.connection.commit()

            cursor.close()

            return True

        except mysql.connector.Error as err:

            print(f"Error adding blocked site: {err}")

            return False

    def get\_blocked\_sites(self, user\_id):

        """Get all blocked sites for a user"""

        try:

            cursor = self.connection.cursor()

            cursor.execute(

                "SELECT website FROM blocked\_sites WHERE user\_id = %s ORDER BY website",

                (user\_id,)

            )

            sites = [row[0] for row in cursor.fetchall()]

            cursor.close()

            return sites

        except mysql.connector.Error as err:

            print(f"Error getting blocked sites: {err}")

            return []

    def remove\_blocked\_site(self, user\_id, website):

        """Remove a website from block list"""

        try:

            cursor = self.connection.cursor()

            cursor.execute(

                "DELETE FROM blocked\_sites WHERE user\_id = %s AND website = %s",

                (user\_id, website)

            )

            self.connection.commit()

            cursor.close()

            return cursor.rowcount > 0

        except mysql.connector.Error as err:

            print(f"Error removing blocked site: {err}")

            return False

    # Study sessions management (enhanced for analytics)

    def record\_study\_session(self, user\_id, duration\_minutes, focus\_rating=None, subject\_tag=None, distractions\_count=0, notes=None):

        """Record a study session with analytics data"""

        try:

            cursor = self.connection.cursor()

            start\_time = datetime.now()

            end\_time = start\_time + timedelta(minutes=duration\_minutes)

            cursor.execute("""

                INSERT INTO study\_sessions

                (user\_id, start\_time, end\_time, duration\_minutes, focus\_rating, subject\_tag, distractions\_count, notes)

                VALUES (%s, %s, %s, %s, %s, %s, %s, %s)

            """, (user\_id, start\_time, end\_time, duration\_minutes, focus\_rating, subject\_tag, distractions\_count, notes))

            self.connection.commit()

            session\_id = cursor.lastrowid

            # Update personal records

            self.\_update\_personal\_records(user\_id, duration\_minutes, focus\_rating)

            cursor.close()

            return session\_id

        except mysql.connector.Error as err:

            print(f"Error recording study session: {err}")

            return None

    def \_update\_personal\_records(self, user\_id, duration\_minutes, focus\_rating):

        """Update personal records for achievements"""

        try:

            cursor = self.connection.cursor()

            # Check for longest session record

            cursor.execute("""

                SELECT record\_value FROM personal\_records

                WHERE user\_id = %s AND record\_type = 'longest\_session'

            """, (user\_id,))

            result = cursor.fetchone()

            if not result or duration\_minutes > result[0]:

                # New longest session!

                cursor.execute("""

                    INSERT INTO personal\_records (user\_id, record\_type, record\_value)

                    VALUES (%s, 'longest\_session', %s)

                    ON DUPLICATE KEY UPDATE record\_value = VALUES(record\_value)

                """, (user\_id, duration\_minutes))

            # Check for best focus record

            if focus\_rating:

                cursor.execute("""

                    SELECT record\_value FROM personal\_records

                    WHERE user\_id = %s AND record\_type = 'best\_focus'

                """, (user\_id,))

                result = cursor.fetchone()

                if not result or focus\_rating > result[0]:

                    # New best focus!

                    cursor.execute("""

                        INSERT INTO personal\_records (user\_id, record\_type, record\_value)

                        VALUES (%s, 'best\_focus', %s)

                        ON DUPLICATE KEY UPDATE record\_value = VALUES(record\_value)

                    """, (user\_id, focus\_rating))

            self.connection.commit()

            cursor.close()

        except mysql.connector.Error as err:

            print(f"Error updating records: {err}")

    # Analytics functions (for AnalyticsDashboard.py)

    def get\_study\_analytics(self, user\_id):

        """Get comprehensive study analytics for dashboard"""

        try:

            cursor = self.connection.cursor(dictionary=True)

            analytics = {}

            # Today's date and 30 days ago

            today = datetime.now().date()

            thirty\_days\_ago = today - timedelta(days=30)

            # Weekly study time (last 4 weeks)

            cursor.execute("""

                SELECT

                    YEAR(start\_time) as year,

                    WEEK(start\_time) as week,

                    SUM(duration\_minutes) as total\_minutes,

                    COUNT(\*) as session\_count,

                    AVG(focus\_rating) as avg\_focus

                FROM study\_sessions

                WHERE user\_id = %s

                GROUP BY YEAR(start\_time), WEEK(start\_time)

                ORDER BY year DESC, week DESC

                LIMIT 4

            """, (user\_id,))

            analytics['weekly\_data'] = cursor.fetchall()

            # Daily study time for last 7 days

            cursor.execute("""

                SELECT

                    DATE(start\_time) as date,

                    SUM(duration\_minutes) as daily\_minutes,

                    COUNT(\*) as sessions,

                    AVG(focus\_rating) as avg\_focus

                FROM study\_sessions

                WHERE user\_id = %s AND DATE(start\_time) >= DATE\_SUB(CURDATE(), INTERVAL 7 DAY)

                GROUP BY DATE(start\_time)

                ORDER BY date DESC

            """, (user\_id,))

            analytics['daily\_data'] = cursor.fetchall()

            # Personal records

            cursor.execute("""

                SELECT

                    MAX(duration\_minutes) as longest\_session,

                    AVG(duration\_minutes) as avg\_session\_length,

                    SUM(duration\_minutes) as total\_study\_time,

                    COUNT(\*) as total\_sessions,

                    MAX(focus\_rating) as best\_focus,

                    AVG(focus\_rating) as avg\_focus\_lifetime

                FROM study\_sessions

                WHERE user\_id = %s

            """, (user\_id,))

            analytics['records'] = cursor.fetchone()

            # Most productive day of week

            cursor.execute("""

                SELECT

                    DAYNAME(start\_time) as day\_name,

                    AVG(duration\_minutes) as avg\_minutes,

                    COUNT(\*) as session\_count

                FROM study\_sessions

                WHERE user\_id = %s

                GROUP BY DAYNAME(start\_time)

                ORDER BY avg\_minutes DESC

                LIMIT 1

            """, (user\_id,))

            analytics['best\_day'] = cursor.fetchone()

            # Subject/tag analysis

            cursor.execute("""

                SELECT

                    subject\_tag,

                    SUM(duration\_minutes) as total\_time,

                    COUNT(\*) as session\_count,

                    AVG(focus\_rating) as avg\_focus

                FROM study\_sessions

                WHERE user\_id = %s AND subject\_tag IS NOT NULL AND subject\_tag != ''

                GROUP BY subject\_tag

                ORDER BY total\_time DESC

                LIMIT 5

            """, (user\_id,))

            analytics['subjects'] = cursor.fetchall()

            # Get actual personal records from personal\_records table

            cursor.execute("""

                SELECT record\_type, record\_value, achieved\_at

                FROM personal\_records

                WHERE user\_id = %s

            """, (user\_id,))

            analytics['personal\_records'] = cursor.fetchall()

            cursor.close()

            return analytics

        except mysql.connector.Error as err:

            print(f"Error getting analytics: {err}")

            return None

    def get\_recent\_sessions(self, user\_id, limit=10):

        """Get recent study sessions"""

        try:

            cursor = self.connection.cursor(dictionary=True)

            cursor.execute("""

                SELECT

                    start\_time, duration\_minutes, focus\_rating, subject\_tag, notes

                FROM study\_sessions

                WHERE user\_id = %s

                ORDER BY start\_time DESC

                LIMIT %s

            """, (user\_id, limit))

            sessions = cursor.fetchall()

            cursor.close()

            return sessions

        except mysql.connector.Error as err:

            print(f"Error getting recent sessions: {err}")

            return []

    def get\_study\_time\_by\_period(self, user\_id, days=30):

        """Get study time for a specific period"""

        try:

            cursor = self.connection.cursor()

            cursor.execute("""

                SELECT SUM(duration\_minutes)

                FROM study\_sessions

                WHERE user\_id = %s AND start\_time >= DATE\_SUB(NOW(), INTERVAL %s DAY)

            """, (user\_id, days))

            result = cursor.fetchone()

            cursor.close()

            return result[0] or 0

        except mysql.connector.Error as err:

            print(f"Error getting study time: {err}")

            return 0

    def get\_todos(self, user\_id):

        """Get all todos for a user"""

        try:

            cursor = self.connection.cursor(dictionary=True)

            cursor.execute("""

                SELECT id, task, completed

                FROM todos

                WHERE user\_id = %s

                ORDER BY created\_at ASC

            """, (user\_id,))

            todos = cursor.fetchall()

            cursor.close()

            return todos

        except Exception as err:

            print(f"Error getting todos: {err}")

            return []

    def save\_todos(self, user\_id, todos):

        """Save todos for a user (replaces all existing todos)"""

        try:

            cursor = self.connection.cursor()

            # Delete existing todos for this user

            cursor.execute("DELETE FROM todos WHERE user\_id = %s", (user\_id,))

            # Insert new todos

            for task, completed in todos:

                cursor.execute("""

                    INSERT INTO todos (user\_id, task, completed)

                    VALUES (%s, %s, %s)

                """, (user\_id, task, completed))

            self.connection.commit()

            cursor.close()

            return True

        except Exception as err:

            print(f"Error saving todos: {err}")

            self.connection.rollback()

            return False

## guy.png

## A statue of a person AI-generated content may be incorrect.

## Leaderboard.py

## A colorful podium with a cup on top AI-generated content may be incorrect.

## login\_screen.py

A screenshot of a login screen

AI-generated content may be incorrect.

import pygame

import sys

import json

import os

import hashlib

# Initialize pygame

pygame.init()

# Screen dimensions

SCREEN\_INFO = pygame.display.Info()

SCREEN\_WIDTH, SCREEN\_HEIGHT = SCREEN\_INFO.current\_w, SCREEN\_INFO.current\_h

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Focus Timer - Login")

# Colors (matching theme)

BACKGROUND = (25, 25, 50)

PANEL\_BG = (40, 40, 80)

DARK\_PANEL = (30, 30, 60)

TEXT\_COLOR = (255, 255, 255)

HIGHLIGHT = (100, 200, 255)

def hash\_password(password):

    '''Hash password using SHA-256'''

    return hashlib.sha256(password.encode()).hexdigest()

ACCENT\_SILVER = (192, 192, 192)

BUTTON\_PRIMARY = (70, 130, 180)

BUTTON\_HOVER = (100, 160, 210)

BUTTON\_SUCCESS = (80, 180, 120)

INPUT\_BG = (40, 40, 60)

BORDER\_COLOR = (100, 100, 150)

SHADOW\_COLOR = (15, 15, 30)

ERROR\_RED = (255, 100, 100)

SUCCESS\_GREEN = (100, 200, 150)

# Fonts

title\_font = pygame.font.SysFont('timesnewroman', 72, bold=True)

header\_font = pygame.font.SysFont('timesnewroman', 36, bold=True)

text\_font = pygame.font.SysFont('timesnewroman', 24)

small\_font = pygame.font.SysFont('timesnewroman', 20)

button\_font = pygame.font.SysFont('timesnewroman', 22, bold=True)

def get\_database\_connection():

    """Create a database connection"""

    try:

        config\_file = "mysql\_config.json"

        if os.path.exists(config\_file):

            with open(config\_file, 'r') as f:

                mysql\_config = json.load(f)

        else:

            mysql\_config = {

                'host': 'localhost',

                'user': 'root',

                'password': '',

                'database': 'focus\_app',

                'port': 3306

            }

        from database import Database

        return Database(mysql\_config)

    except Exception as e:

        print(f"Database connection error: {e}")

        return None

def get\_all\_users():

    """Get all users from database"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            cursor.execute("SELECT id, username FROM users ORDER BY username")

            users = cursor.fetchall()

            cursor.close()

            return users

    except Exception as e:

        print(f"Error loading users: {e}")

    return []

def verify\_password(username, password):

    """Verify user password by comparing hashes"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            username = username.strip()

            password = password.strip()

            # Hash the input password

            input\_hash = hash\_password(password)

            # Get user and stored hash

            cursor.execute("SELECT id, username, password\_hash FROM users WHERE username = %s", (username,))

            user = cursor.fetchone()

            cursor.close()

            db.connection.close()

            if not user:

                return None

            stored\_hash = user.get('password\_hash', '').strip()

            # Compare hashes

            if stored\_hash and stored\_hash == input\_hash:

                return {'id': user['id'], 'username': user['username']}

            return None

    except Exception as e:

        print(f"Login error: {e}")

        return None

def save\_current\_user(user\_id, username):

    """Save current user to file"""

    try:

        with open("current\_user.txt", "w") as f:

            f.write(f"{user\_id}\n{username}\n")

        return True

    except:

        return False

class Button:

    def \_\_init\_\_(self, x, y, width, height, text, color=BUTTON\_PRIMARY, hover\_color=BUTTON\_HOVER):

        self.rect = pygame.Rect(x, y, width, height)

        self.text = text

        self.color = color

        self.hover\_color = hover\_color

        self.hovered = False

    def draw(self, screen):

        # Shadow

        shadow = pygame.Rect(self.rect.x + 4, self.rect.y + 4,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Button

        color = self.hover\_color if self.hovered else self.color

        pygame.draw.rect(screen, color, self.rect, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=12)

        # Text

        text\_surface = button\_font.render(self.text, True, TEXT\_COLOR)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.hovered = self.rect.collidepoint(mouse\_pos)

    def is\_clicked(self, mouse\_pos, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            return self.rect.collidepoint(mouse\_pos)

        return False

class UserButton:

    def \_\_init\_\_(self, x, y, width, height, username, user\_id):

        self.rect = pygame.Rect(x, y, width, height)

        self.username = username

        self.user\_id = user\_id

        self.hovered = False

        self.selected = False

    def draw(self, screen):

        # Shadow

        shadow = pygame.Rect(self.rect.x + 3, self.rect.y + 3,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=10)

        # Background

        if self.selected:

            color = HIGHLIGHT

            text\_color = BACKGROUND

        elif self.hovered:

            color = BUTTON\_HOVER

            text\_color = TEXT\_COLOR

        else:

            color = PANEL\_BG

            text\_color = TEXT\_COLOR

        pygame.draw.rect(screen, color, self.rect, border\_radius=10)

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=10)

        # Username

        text\_surface = text\_font.render(self.username, True, text\_color)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.hovered = self.rect.collidepoint(mouse\_pos)

    def is\_clicked(self, mouse\_pos, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            return self.rect.collidepoint(mouse\_pos)

        return False

def draw\_input\_box(screen, x, y, width, height, text, active, is\_password=False):

    """Draw an input box"""

    # Shadow

    shadow = pygame.Rect(x + 3, y + 3, width, height)

    pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=10)

    # Box

    color = HIGHLIGHT if active else INPUT\_BG

    pygame.draw.rect(screen, color, (x, y, width, height), border\_radius=10)

    pygame.draw.rect(screen, BORDER\_COLOR, (x, y, width, height), 2, border\_radius=10)

    # Text

    if is\_password:

        display\_text = "•" \* len(text)

    else:

        display\_text = text

    if active:

        display\_text += "|"

    text\_surface = text\_font.render(display\_text, True, TEXT\_COLOR)

    screen.blit(text\_surface, (x + 15, y + 12))

def login\_screen():

    """Main login screen"""

    global screen  # Declare at the start of the function

    clock = pygame.time.Clock()

    # Get all users

    all\_users = get\_all\_users()

    if not all\_users:

        # No users in database - open user management as subprocess

        print("No users found! Opening user management...")

        pygame.quit()

        import subprocess

        subprocess.run([sys.executable, "user\_management.py"])

        # After user management, check if users were created

        all\_users = get\_all\_users()

        if not all\_users:

            # Still no users, exit

            return None

        # Users created, restart login screen

        pygame.init()

        screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

        pygame.display.set\_caption("Focus Timer - Login")

        return login\_screen()

    # State

    selected\_user = None

    password\_input = ""

    password\_active = False

    error\_message = ""

    success\_message = ""

    # Create user buttons (VERTICAL LAYOUT)

    user\_buttons = []

    button\_width = 400

    button\_height = 55

    spacing\_y = 15

    start\_x = SCREEN\_WIDTH // 2 - button\_width // 2

    start\_y = 280

    # Limit to 8 users visible (can add scrolling later if needed)

    max\_visible\_users = 8

    visible\_users = all\_users[:max\_visible\_users]

    for i, user in enumerate(visible\_users):

        y = start\_y + i \* (button\_height + spacing\_y)

        user\_buttons.append(UserButton(start\_x, y, button\_width, button\_height,

                                       user['username'], user['id']))

    # Password input box (positioned below user list)

    password\_box\_width = 400

    password\_box\_height = 50

    password\_box\_x = SCREEN\_WIDTH // 2 - password\_box\_width // 2

    # Calculate position based on visible users

    num\_visible = len(visible\_users)

    list\_height = num\_visible \* (button\_height + spacing\_y)

    password\_box\_y = start\_y + list\_height + 40

    password\_box\_rect = pygame.Rect(password\_box\_x, password\_box\_y, password\_box\_width, password\_box\_height)

    # Login button

    login\_button = Button(SCREEN\_WIDTH // 2 - 100, password\_box\_y + 80, 200, 55,

                         "Login", BUTTON\_SUCCESS, BUTTON\_HOVER)

    # Add User button (bottom right)

    add\_user\_button = Button(SCREEN\_WIDTH - 200, SCREEN\_HEIGHT - 100, 180, 50,

                             "+ Add User", BUTTON\_PRIMARY, BUTTON\_HOVER)

    running = True

    while running:

        mouse\_pos = pygame.mouse.get\_pos()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                return None, None

            elif event.type == pygame.MOUSEBUTTONDOWN:

                # Check user button clicks

                for btn in user\_buttons:

                    if btn.is\_clicked(mouse\_pos, event):

                        selected\_user = (btn.user\_id, btn.username)

                        # Deselect all, select clicked

                        for b in user\_buttons:

                            b.selected = False

                        btn.selected = True

                        error\_message = ""

                        success\_message = ""

                # Check password box click

                if password\_box\_rect.collidepoint(mouse\_pos):

                    password\_active = True

                else:

                    password\_active = False

                # Check login button

                if login\_button.is\_clicked(mouse\_pos, event):

                    if selected\_user and password\_input:

                        user = verify\_password(selected\_user[1], password\_input)

                        if user:

                            # Success!

                            save\_current\_user(user['id'], user['username'])

                            success\_message = "Login successful!"

                            error\_message = ""

                            pygame.time.wait(500)

                            return user['id'], user['username']

                        else:

                            error\_message = "Incorrect password!"

                            success\_message = ""

                    else:

                        error\_message = "Please select a user and enter password"

                        success\_message = ""

                # Check add user button

                if add\_user\_button.is\_clicked(mouse\_pos, event):

                    pygame.quit()

                    pygame.quit()

                    import subprocess

                    subprocess.run([sys.executable, "user\_management.py"])

                    pygame.init()

                    screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

                    pygame.display.set\_caption("Focus Timer - Login")

                    if result == "continue" or result == "back":

                        # Restart login screen

                        pygame.init()

                        screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

                        pygame.display.set\_caption("Focus Timer - Login")

                        return login\_screen()

                    else:

                        return None, None

            elif event.type == pygame.KEYDOWN and password\_active:

                if event.key == pygame.K\_RETURN:

                    # Try to login

                    if selected\_user and password\_input:

                        user = verify\_password(selected\_user[1], password\_input)

                        if user:

                            save\_current\_user(user['id'], user['username'])

                            success\_message = "Login successful!"

                            error\_message = ""

                            pygame.time.wait(500)

                            return user['id'], user['username']

                        else:

                            error\_message = "Incorrect password!"

                            success\_message = ""

                elif event.key == pygame.K\_BACKSPACE:

                    password\_input = password\_input[:-1]

                elif event.key == pygame.K\_ESCAPE:

                    password\_active = False

                elif len(password\_input) < 30:

                    password\_input += event.unicode

        # Update hover states

        for btn in user\_buttons:

            btn.check\_hover(mouse\_pos)

        login\_button.check\_hover(mouse\_pos)

        add\_user\_button.check\_hover(mouse\_pos)

        # Draw everything

        screen.fill(BACKGROUND)

        # Title

        title\_text = title\_font.render("FOCUS TIMER", True, HIGHLIGHT)

        title\_rect = title\_text.get\_rect(center=(SCREEN\_WIDTH // 2, 100))

        screen.blit(title\_text, title\_rect)

        # Subtitle

        subtitle = header\_font.render("Select Your Account", True, ACCENT\_SILVER)

        subtitle\_rect = subtitle.get\_rect(center=(SCREEN\_WIDTH // 2, 200))

        screen.blit(subtitle, subtitle\_rect)

        # User buttons

        for btn in user\_buttons:

            btn.draw(screen)

        # Password label

        if selected\_user:

            password\_label = text\_font.render(f"Password for {selected\_user[1]}:", True, TEXT\_COLOR)

            screen.blit(password\_label, (password\_box\_x, password\_box\_y - 35))

            # Password input

            draw\_input\_box(screen, password\_box\_x, password\_box\_y, password\_box\_width,

                          password\_box\_height, password\_input, password\_active, is\_password=True)

            # Login button

            login\_button.draw(screen)

        # Error/Success messages

        if error\_message:

            error\_surf = text\_font.render(error\_message, True, ERROR\_RED)

            error\_rect = error\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, password\_box\_y + 150))

            screen.blit(error\_surf, error\_rect)

        if success\_message:

            success\_surf = text\_font.render(success\_message, True, SUCCESS\_GREEN)

            success\_rect = success\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, password\_box\_y + 150))

            screen.blit(success\_surf, success\_rect)

        # Instructions

        if not selected\_user:

            instruction = small\_font.render("Click on your name to continue", True, ACCENT\_SILVER)

            instruction\_rect = instruction.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 100))

            screen.blit(instruction, instruction\_rect)

        # Add user button

        add\_user\_button.draw(screen)

        pygame.display.flip()

        clock.tick(60)

    return None, None

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

    result = login\_screen()

    if result and result[0] is not None:

        user\_id, username = result

        print(f"Logged in as: {username} (ID: {user\_id})")

    pygame.quit()

    sys.exit()

## main.py

A screenshot of a computer

AI-generated content may be incorrect.

A person looking at the camera

AI-generated content may be incorrect.

## import sys

## import subprocess

## import ctypes

## import time

## import os

## import json

## # Flag to skip slow checks on subsequent runs

## \_FIRST\_RUN = not os.path.exists(".app\_initialized")

## def check\_and\_install\_modules():

## """Check for required modules and install missing ones"""

## if not \_FIRST\_RUN:

## # Skip module check - already done in this session

## return

## 

## required\_modules = [

## ('pygame', 'pygame'),

## ('cv2', 'opencv-python'),

## ('numpy', 'numpy'),

## ('PIL', 'pillow'), # PIL is the module name, pillow is the package

## ('mysql.connector', 'mysql-connector-python')

## ]

## 

## missing\_modules = []

## 

## print("🔍 Checking required modules...")

## 

## for module\_name, package\_name in required\_modules:

## try:

## \_\_import\_\_(module\_name)

## print(f"✅ {module\_name} is installed")

## except ImportError:

## print(f"❌ {module\_name} is missing")

## missing\_modules.append(package\_name)

## 

## if missing\_modules:

## print(f"\n📦 Installing {len(missing\_modules)} missing module(s)...")

## for package in missing\_modules:

## try:

## print(f"Installing {package}...")

## subprocess.check\_call([sys.executable, '-m', 'pip', 'install', package])

## print(f"✅ Successfully installed {package}")

## except subprocess.CalledProcessError:

## print(f"❌ Failed to install {package}")

## else:

## print("✅ All required modules are installed")

## 

## print()

## # Check and install modules before anything else

## check\_and\_install\_modules()

## def is\_admin():

## try:

## return ctypes.windll.shell32.IsUserAnAdmin()

## except:

## return False

## # Check if running as admin - but don't exit immediately

## if \_FIRST\_RUN:

## if not is\_admin():

## print("⚠️ WARNING: Not running as administrator")

## print("=========================================")

## print("Website blocking may not work properly.")

## print("Some features require admin privileges.")

## print()

## print("The app will start, but website blocking may fail.")

## print("To enable full functionality, run as administrator.")

## print()

## time.sleep(2) # Give time to read the warning

## admin\_warning = True

## else:

## print("✅ Running with administrator privileges")

## print("🌐 Website blocking enabled")

## admin\_warning = False

## else:

## # On subsequent runs, just check silently

## admin\_warning = not is\_admin()

## # ===== MYSQL CONFIGURATION POPUP WITH PYGAME =====

## # Global color constants

## BUTTON\_PRIMARY = (70, 130, 180)

## BUTTON\_HOVER = (100, 160, 210)

## BUTTON\_DANGER = (200, 60, 60)

## def get\_mysql\_config():

## """Get MySQL configuration with PyGame popup"""

## config\_file = "mysql\_config.json"

## 

## # Check if config file exists

## if os.path.exists(config\_file):

## try:

## with open(config\_file, 'r') as f:

## config = json.load(f)

## print("✅ Loaded MySQL configuration from file")

## return config

## except:

## pass

## 

## # First time setup - show PyGame popup

## print("Opening MySQL configuration window...")

## 

## try:

## import pygame

## import cv2

## import numpy as np

## 

## # Initialize PyGame for the popup

## pygame.init()

## 

## # Window settings

## POPUP\_WIDTH = 600

## POPUP\_HEIGHT = 600

## screen = pygame.display.set\_mode((POPUP\_WIDTH, POPUP\_HEIGHT))

## pygame.display.set\_caption("MySQL Database Setup")

## 

## # Colors for dark mode

## DARK\_BG = (20, 20, 30)

## DARK\_PANEL = (30, 30, 45)

## TEXT\_COLOR = (220, 220, 220)

## HIGHLIGHT = (100, 150, 255)

## BUTTON\_COLOR = (50, 100, 180)

## BUTTON\_HOVER = (70, 120, 200)

## INPUT\_BG = (40, 40, 60)

## INPUT\_BORDER = (70, 70, 100)

## 

## # Fonts

## title\_font = pygame.font.SysFont('Arial', 28, bold=True)

## text\_font = pygame.font.SysFont('Arial', 18)

## small\_font = pygame.font.SysFont('Arial', 14)

## input\_font = pygame.font.SysFont('Arial', 16)

## 

## # Load video background

## try:

## video\_cap = cv2.VideoCapture('snowfall.mp4')

## video\_available = True

## except:

## video\_available = False

## print("⚠️ Could not load video background")

## 

## def get\_video\_frame():

## if video\_available:

## ret, frame = video\_cap.read()

## if not ret:

## video\_cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

## ret, frame = video\_cap.read()

## 

## if ret:

## # Crop and resize for square aspect ratio

## height, width = frame.shape[:2]

## size = min(height, width)

## start\_x = (width - size) // 2

## start\_y = (height - size) // 2

## cropped = frame[start\_y:start\_y+size, start\_x:start\_x+size]

## 

## # Resize to popup size

## resized = cv2.resize(cropped, (POPUP\_WIDTH, POPUP\_HEIGHT))

## 

## # Darken the video for better text visibility

## resized = cv2.addWeighted(resized, 0.3, np.zeros\_like(resized), 0.7, 0)

## 

## # Convert to PyGame surface

## resized = cv2.cvtColor(resized, cv2.COLOR\_BGR2RGB)

## return pygame.surfarray.make\_surface(resized.swapaxes(0, 1))

## return None

## 

## # Input fields (database is auto-created as 'focus\_app')

## input\_fields = [

## {"label": "Host:", "value": "localhost", "type": "text", "active": False, "rect": None},

## {"label": "Username:", "value": "root", "type": "text", "active": False, "rect": None},

## {"label": "Password:", "value": "", "type": "password", "active": False, "rect": None},

## {"label": "Port:", "value": "3306", "type": "number", "active": False, "rect": None}

## ]

## 

## # Buttons

## test\_button = {"text": "Test Connection", "rect": None, "hover": False}

## save\_button = {"text": "Save & Continue", "rect": None, "hover": False}

## 

## # Current active input field

## active\_field = None

## 

## def draw\_popup():

## # Get video frame

## video\_frame = get\_video\_frame()

## if video\_frame:

## screen.blit(video\_frame, (0, 0))

## else:

## screen.fill(DARK\_BG)

## 

## # Draw semi-transparent overlay

## overlay = pygame.Surface((POPUP\_WIDTH, POPUP\_HEIGHT), pygame.SRCALPHA)

## overlay.fill((20, 20, 30, 200)) # Semi-transparent dark

## screen.blit(overlay, (0, 0))

## 

## # Draw title

## title = title\_font.render("MySQL Database Setup", True, HIGHLIGHT)

## screen.blit(title, (POPUP\_WIDTH//2 - title.get\_width()//2, 30))

## 

## # Draw description

## desc\_lines = [

## "This app needs MySQL to store your study data.",

## "Please enter your MySQL credentials below.",

## "Database 'focus\_app' will be created automatically."

## ]

## 

## for i, line in enumerate(desc\_lines):

## desc = text\_font.render(line, True, TEXT\_COLOR)

## screen.blit(desc, (POPUP\_WIDTH//2 - desc.get\_width()//2, 80 + i\*25))

## 

## # Draw input fields

## field\_height = 35

## field\_width = 400

## start\_y = 180

## 

## for i, field in enumerate(input\_fields):

## y\_pos = start\_y + i \* (field\_height + 15)

## 

## # Draw label

## label = text\_font.render(field["label"], True, TEXT\_COLOR)

## screen.blit(label, (50, y\_pos + 8))

## 

## # Draw input box

## input\_rect = pygame.Rect(200, y\_pos, field\_width, field\_height)

## field["rect"] = input\_rect

## 

## # Input box background

## box\_color = HIGHLIGHT if field["active"] else INPUT\_BORDER

## pygame.draw.rect(screen, INPUT\_BG, input\_rect, border\_radius=5)

## pygame.draw.rect(screen, box\_color, input\_rect, 2, border\_radius=5)

## 

## # Draw input text

## display\_text = field["value"]

## if field["type"] == "password" and field["value"]:

## display\_text = "\*" \* len(field["value"])

## 

## if display\_text:

## text\_color = TEXT\_COLOR

## else:

## display\_text = "Click to enter..."

## text\_color = (100, 100, 120)

## 

## text\_surface = input\_font.render(display\_text, True, text\_color)

## text\_x = input\_rect.x + 10

## screen.blit(text\_surface, (text\_x, y\_pos + 8))

## 

## # Draw cursor if active

## if field["active"] and int(pygame.time.get\_ticks() / 500) % 2 == 0:

## cursor\_x = text\_x + text\_surface.get\_width()

## pygame.draw.line(screen, TEXT\_COLOR,

## (cursor\_x, y\_pos + 5),

## (cursor\_x, y\_pos + field\_height - 5), 2)

## 

## # Draw buttons

## button\_y = start\_y + len(input\_fields) \* (field\_height + 15) + 30

## button\_width = 180

## button\_height = 40

## button\_spacing = 30

## 

## # Test button

## test\_rect = pygame.Rect(POPUP\_WIDTH//2 - button\_width - button\_spacing//2,

## button\_y, button\_width, button\_height)

## test\_button["rect"] = test\_rect

## 

## test\_color = BUTTON\_HOVER if test\_button["hover"] else BUTTON\_COLOR

## pygame.draw.rect(screen, test\_color, test\_rect, border\_radius=8)

## pygame.draw.rect(screen, HIGHLIGHT, test\_rect, 2, border\_radius=8)

## 

## test\_text = text\_font.render(test\_button["text"], True, TEXT\_COLOR)

## screen.blit(test\_text, (test\_rect.centerx - test\_text.get\_width()//2,

## test\_rect.centery - test\_text.get\_height()//2))

## 

## # Save button

## save\_rect = pygame.Rect(POPUP\_WIDTH//2 + button\_spacing//2,

## button\_y, button\_width, button\_height)

## save\_button["rect"] = save\_rect

## 

## save\_color = BUTTON\_HOVER if save\_button["hover"] else BUTTON\_COLOR

## pygame.draw.rect(screen, save\_color, save\_rect, border\_radius=8)

## pygame.draw.rect(screen, HIGHLIGHT, save\_rect, 2, border\_radius=8)

## 

## save\_text = text\_font.render(save\_button["text"], True, TEXT\_COLOR)

## screen.blit(save\_text, (save\_rect.centerx - save\_text.get\_width()//2,

## save\_rect.centery - save\_text.get\_height()//2))

## 

## # Draw help text at bottom

## help\_text = small\_font.render("Press TAB to navigate • Press ENTER to save", True, (150, 150, 150))

## screen.blit(help\_text, (POPUP\_WIDTH//2 - help\_text.get\_width()//2, POPUP\_HEIGHT - 40))

## 

## pygame.display.flip()

## 

## def test\_mysql\_connection():

## """Test MySQL connection"""

## import mysql.connector

## try:

## config = {

## 'host': input\_fields[0]['value'],

## 'user': input\_fields[1]['value'],

## 'password': input\_fields[2]['value'],

## 'port': int(input\_fields[3]['value']) if input\_fields[3]['value'].isdigit() else 3306,

## 'connection\_timeout': 5

## }

## print(f"Testing MySQL connection...")

## conn = mysql.connector.connect(\*\*config)

## conn.close()

## print("Connection successful!")

## return True, "✅ MySQL OK! Database will be created."

## except mysql.connector.Error as e:

## print(f"MySQL error: {e}")

## return False, f"❌ MySQL Error: {str(e)[:100]}"

## except Exception as e:

## print(f"Connection error: {e}")

## return False, f"❌ Error: {str(e)[:100]}"

## 

## 

## def show\_message(message, is\_success=True):

## """Show a temporary message"""

## message\_color = (100, 200, 100) if is\_success else (220, 100, 100)

## 

## # Draw semi-transparent overlay

## overlay = pygame.Surface((POPUP\_WIDTH, POPUP\_HEIGHT), pygame.SRCALPHA)

## overlay.fill((0, 0, 0, 150))

## screen.blit(overlay, (0, 0))

## 

## # Draw message box

## box\_width = 500

## box\_height = 150

## box\_x = (POPUP\_WIDTH - box\_width) // 2

## box\_y = (POPUP\_HEIGHT - box\_height) // 2

## 

## pygame.draw.rect(screen, DARK\_PANEL, (box\_x, box\_y, box\_width, box\_height), border\_radius=10)

## pygame.draw.rect(screen, message\_color, (box\_x, box\_y, box\_width, box\_height), 2, border\_radius=10)

## 

## # Draw message text (split into lines if needed)

## words = message.split()

## lines = []

## current\_line = []

## 

## for word in words:

## test\_line = ' '.join(current\_line + [word])

## test\_surface = text\_font.render(test\_line, True, TEXT\_COLOR)

## if test\_surface.get\_width() < box\_width - 40:

## current\_line.append(word)

## else:

## lines.append(' '.join(current\_line))

## current\_line = [word]

## if current\_line:

## lines.append(' '.join(current\_line))

## 

## # Draw each line

## for i, line in enumerate(lines):

## line\_surface = text\_font.render(line, True, TEXT\_COLOR)

## screen.blit(line\_surface, (box\_x + (box\_width - line\_surface.get\_width()) // 2,

## box\_y + 30 + i \* 25))

## 

## # Draw continue instruction

## continue\_text = small\_font.render("Click anywhere to continue...", True, (150, 150, 150))

## screen.blit(continue\_text, (box\_x + (box\_width - continue\_text.get\_width()) // 2,

## box\_y + box\_height - 40))

## 

## pygame.display.flip()

## 

## # Wait for click

## waiting = True

## while waiting:

## for event in pygame.event.get():

## if event.type == pygame.QUIT:

## pygame.quit()

## sys.exit(0)

## if event.type == pygame.MOUSEBUTTONDOWN:

## waiting = False

## if event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE:

## waiting = False

## 

## # Main popup loop

## clock = pygame.time.Clock()

## running = True

## config\_saved = False

## 

## while running:

## mouse\_pos = pygame.mouse.get\_pos()

## 

## # Update button hover states

## test\_button["hover"] = test\_button["rect"] and test\_button["rect"].collidepoint(mouse\_pos)

## save\_button["hover"] = save\_button["rect"] and save\_button["rect"].collidepoint(mouse\_pos)

## 

## for event in pygame.event.get():

## if event.type == pygame.QUIT:

## running = False

## config\_saved = False

## 

## elif event.type == pygame.MOUSEBUTTONDOWN:

## # Check input field clicks

## clicked\_field = None

## for i, field in enumerate(input\_fields):

## if field["rect"] and field["rect"].collidepoint(mouse\_pos):

## clicked\_field = i

## break

## 

## if clicked\_field is not None:

## # Activate clicked field

## for i, field in enumerate(input\_fields):

## field["active"] = (i == clicked\_field)

## active\_field = clicked\_field

## 

## # Check button clicks

## elif test\_button["rect"] and test\_button["rect"].collidepoint(mouse\_pos):

## success, message = test\_mysql\_connection()

## show\_message(message, success)

## 

## elif save\_button["rect"] and save\_button["rect"].collidepoint(mouse\_pos):

## success, message = test\_mysql\_connection()

## if success:

## # Save configuration

## config = {

## 'host': input\_fields[0]["value"],

## 'user': input\_fields[1]["value"],

## 'password': input\_fields[2]["value"],

## 'database': 'focus\_app', # Hardcoded

## 'port': int(input\_fields[3]["value"]) if input\_fields[3]["value"].isdigit() else 3306

## }

## 

## try:

## with open(config\_file, 'w') as f:

## json.dump(config, f)

## print("✅ MySQL config saved!")

## show\_message("✅ Configuration saved!\n\nYou can edit mysql\_config.json later.", True)

## config\_saved = True

## running = False

## except Exception as e:

## show\_message(f"❌ Could not save configuration:\n{e}", False)

## else:

## show\_message(message, False)

## 

## else:

## # Clicked outside - deactivate all fields

## for field in input\_fields:

## field["active"] = False

## active\_field = None

## 

## elif event.type == pygame.KEYDOWN:

## if active\_field is not None:

## field = input\_fields[active\_field]

## 

## if event.key == pygame.K\_RETURN:

## # Move to next field or save

## if active\_field < len(input\_fields) - 1:

## for f in input\_fields:

## f["active"] = False

## input\_fields[active\_field + 1]["active"] = True

## active\_field += 1

## else:

## # Last field - trigger save

## success, message = test\_mysql\_connection()

## if success:

## config = {

## 'host': input\_fields[0]["value"],

## 'user': input\_fields[1]["value"],

## 'password': input\_fields[2]["value"],

## 'database': 'focus\_app', # Hardcoded

## 'port': int(input\_fields[3]["value"]) if input\_fields[3]["value"].isdigit() else 3306

## }

## 

## try:

## with open(config\_file, 'w') as f:

## json.dump(config, f)

## print("✅ MySQL config saved!")

## config\_saved = True

## running = False

## except Exception as e:

## show\_message(f"❌ Could not save: {e}", False)

## else:

## show\_message(message, False)

## 

## elif event.key == pygame.K\_TAB:

## # Move to next field

## for f in input\_fields:

## f["active"] = False

## next\_field = (active\_field + 1) % len(input\_fields)

## input\_fields[next\_field]["active"] = True

## active\_field = next\_field

## 

## elif event.key == pygame.K\_BACKSPACE:

## field["value"] = field["value"][:-1]

## 

## elif event.key == pygame.K\_ESCAPE:

## for f in input\_fields:

## f["active"] = False

## active\_field = None

## 

## else:

## # Regular character input

## if field["type"] == "number":

## if event.unicode.isdigit():

## field["value"] += event.unicode

## else:

## if len(field["value"]) < 50: # Limit length

## field["value"] += event.unicode

## 

## draw\_popup()

## clock.tick(60)

## 

## # Cleanup

## if video\_available:

## video\_cap.release()

## pygame.quit()

## 

## if config\_saved and os.path.exists(config\_file):

## with open(config\_file, 'r') as f:

## return json.load(f)

## else:

## print("❌ MySQL configuration was not saved.")

## return None

## 

## except Exception as e:

## print(f"❌ Error with PyGame popup: {e}")

## return get\_mysql\_config\_cli()

## def get\_mysql\_config\_cli():

## """Fallback to command line if PyGame fails"""

## config\_file = "mysql\_config.json"

## 

## print("\n" + "="\*50)

## print("FIRST TIME SETUP - MySQL DATABASE CONFIGURATION")

## print("="\*50)

## print("\nThis app uses MySQL to store your study data.")

## print("Please enter your MySQL credentials:")

## print()

## 

## host = input("MySQL Host [localhost]: ") or "localhost"

## user = input("MySQL Username [root]: ") or "root"

## password = input("MySQL Password (leave empty if none): ")

## port = input("Port [3306]: ") or "3306"

## 

## print("Database 'focus\_app' will be created automatically.")

## 

## config = {

## 'host': host,

## 'user': user,

## 'password': password,

## 'database': 'focus\_app', # Hardcoded

## 'port': int(port)

## }

## 

## try:

## with open(config\_file, 'w') as f:

## json.dump(config, f)

## print(f"\n✅ Configuration saved to {config\_file}")

## return config

## except Exception as e:

## print(f"❌ Could not save configuration: {e}")

## return config

## # Get MySQL configuration

## mysql\_config = get\_mysql\_config()

## if not mysql\_config:

## print("❌ Failed to configure MySQL. Exiting.")

## sys.exit(1)

## print("\n" + "="\*50)

## print("Initializing Database...")

## print("="\*50)

## def initialize\_database\_tables(config):

## '''Create database and tables if they don't exist'''

## if not \_FIRST\_RUN:

## return True

## 

## try:

## import mysql.connector

## 

## print("Connecting to MySQL...")

## conn = mysql.connector.connect(

## host=config['host'],

## user=config['user'],

## password=config['password'],

## port=config.get('port', 3306),

## connect\_timeout=10,

## autocommit=True

## )

## cursor = conn.cursor()

## 

## print("Creating database...")

## cursor.execute("CREATE DATABASE IF NOT EXISTS focus\_app")

## print("✅ Database 'focus\_app' ready")

## 

## cursor.execute("USE focus\_app")

## 

## print("Creating users table...")

## cursor.execute("""

## CREATE TABLE IF NOT EXISTS users (

## id INT AUTO\_INCREMENT PRIMARY KEY,

## username VARCHAR(255) NOT NULL UNIQUE,

## password\_hash VARCHAR(255) NOT NULL DEFAULT '',

## created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP

## )

## """)

## print("✅ Users table ready")

## 

## print("Creating blocked\_sites table...")

## cursor.execute("""

## CREATE TABLE IF NOT EXISTS blocked\_sites (

## id INT AUTO\_INCREMENT PRIMARY KEY,

## user\_id INT NOT NULL,

## website VARCHAR(255) NOT NULL,

## created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

## FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

## )

## """)

## print("✅ Blocked sites table ready")

## 

## print("Creating todos table...")

## cursor.execute("""

## CREATE TABLE IF NOT EXISTS todos (

## id INT AUTO\_INCREMENT PRIMARY KEY,

## user\_id INT NOT NULL,

## task VARCHAR(255) NOT NULL,

## completed BOOLEAN DEFAULT FALSE,

## created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

## FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

## )

## """)

## print("✅ Todos table ready")

## 

## print("Creating study\_sessions table...")

## cursor.execute("""

## CREATE TABLE IF NOT EXISTS study\_sessions (

## id INT AUTO\_INCREMENT PRIMARY KEY,

## user\_id INT NOT NULL,

## subject VARCHAR(255),

## duration INT NOT NULL,

## completed BOOLEAN DEFAULT FALSE,

## created\_at TIMESTAMP DEFAULT CURRENT\_TIMESTAMP,

## FOREIGN KEY (user\_id) REFERENCES users(id) ON DELETE CASCADE

## )

## """)

## print("✅ Study sessions table ready")

## 

## print("Closing database connection...")

## try:

## cursor.close()

## except:

## pass

## try:

## conn.close()

## except:

## pass

## 

## print("✅ Database initialization complete!\n")

## return True

## 

## except Exception as e:

## print(f"❌ Database initialization error: {e}")

## import traceback

## traceback.print\_exc()

## try:

## conn.close()

## except:

## pass

## return False

## initialize\_database\_tables(mysql\_config)

## # Create flag file to skip checks on subsequent runs

## if \_FIRST\_RUN:

## try:

## with open('.app\_initialized', 'w') as f:

## f.write('1')

## except:

## pass

## print("\n" + "="\*50)

## print("Starting Focus App...")

## print("="\*50)

## # Now import the rest of the modules

## import pygame

## import cv2

## import mysql.connector

## # Import the database module

## try:

## from database import Database

## # Pass the config to the Database class

## db = Database(mysql\_config)

## print("✅ Database connection established")

## except Exception as e:

## print(f"❌ Database error: {e}")

## print("App will run with limited functionality")

## db = None

## import music\_player

## class Button:

## def \_\_init\_\_(self, x, y, width, height, text, color=None, hover\_color=None, text\_color=(255, 255, 255)):

## self.rect = pygame.Rect(x, y, width, height)

## self.text = text

## self.color = color or BUTTON\_PRIMARY

## self.hover\_color = hover\_color or BUTTON\_HOVER

## self.text\_color = text\_color

## self.hovered = False

## 

## def draw(self, screen, font):

## current\_color = self.hover\_color if self.hovered else self.color

## pygame.draw.rect(screen, current\_color, self.rect, border\_radius=10)

## pygame.draw.rect(screen, (100, 100, 150), self.rect, 2, border\_radius=10)

## text\_surface = font.render(self.text, True, self.text\_color)

## text\_rect = text\_surface.get\_rect(center=self.rect.center)

## screen.blit(text\_surface, text\_rect)

## 

## def check\_hover(self, mouse\_pos):

## self.hovered = self.rect.collidepoint(mouse\_pos)

## 

## def is\_clicked(self, pos):

## return self.rect.collidepoint(pos)

## class ImageButton:

## def \_\_init\_\_(self, x, y, width, height, image\_path, hover\_image\_path=None):

## self.rect = pygame.Rect(x, y, width, height)

## try:

## self.image = pygame.image.load(image\_path).convert\_alpha()

## self.image = pygame.transform.scale(self.image, (width, height))

## 

## if hover\_image\_path and os.path.exists(hover\_image\_path):

## self.hover\_image = pygame.image.load(hover\_image\_path).convert\_alpha()

## self.hover\_image = pygame.transform.scale(self.hover\_image, (width, height))

## else:

## self.hover\_image = self.image

## except:

## # Fallback to a simple rectangle if image fails to load

## self.image = pygame.Surface((width, height))

## self.image.fill((100, 100, 100))

## self.hover\_image = pygame.Surface((width, height))

## self.hover\_image.fill((150, 150, 150))

## 

## self.is\_hovered = False

## 

## def draw(self, screen):

## if self.is\_hovered:

## screen.blit(self.hover\_image, self.rect)

## else:

## screen.blit(self.image, self.rect)

## 

## def is\_clicked(self, pos):

## return self.rect.collidepoint(pos)

## 

## def update\_hover(self, pos):

## self.is\_hovered = self.rect.collidepoint(pos)

## class VolumeControls:

## def \_\_init\_\_(self, screen\_width):

## self.screen\_width = screen\_width

## self.visible = False

## self.dragging = False

## 

## # Volume icon position (top right)

## self.icon\_rect = pygame.Rect(screen\_width - 70, 20, 50, 50)

## 

## # Volume panel (appears when icon clicked)

## self.panel\_rect = pygame.Rect(screen\_width - 200, 70, 180, 120)

## 

## # Slider

## self.slider\_rect = pygame.Rect(screen\_width - 190, 100, 160, 15)

## self.knob\_radius = 10

## self.knob\_x = self.slider\_rect.x + (music\_player.global\_music.get\_volume() \* self.slider\_rect.width)

## self.knob\_y = self.slider\_rect.y + self.slider\_rect.height // 2

## 

## # Music control buttons

## self.prev\_btn = pygame.Rect(screen\_width - 190, 130, 40, 30)

## self.play\_pause\_btn = pygame.Rect(screen\_width - 140, 130, 40, 30)

## self.next\_btn = pygame.Rect(screen\_width - 90, 130, 40, 30)

## 

## # Load PNG icons

## self.load\_icons()

## 

## self.font = pygame.font.SysFont('Arial', 12)

## 

## def load\_icons(self):

## """Load and resize PNG icons from assets folder"""

## try:

## # Load and resize icons

## pause\_img = pygame.image.load("assets/pause.png").convert\_alpha()

## play\_img = pygame.image.load("assets/play.png").convert\_alpha()

## next\_img = pygame.image.load("assets/next.png").convert\_alpha()

## previous\_img = pygame.image.load("assets/previous.png").convert\_alpha()

## 

## # Resize to fit buttons (20x20 for 40x30 buttons)

## self.pause\_icon = pygame.transform.scale(pause\_img, (20, 20))

## self.play\_icon = pygame.transform.scale(play\_img, (20, 20))

## self.next\_icon = pygame.transform.scale(next\_img, (20, 20))

## self.previous\_icon = pygame.transform.scale(previous\_img, (20, 20))

## 

## except Exception as e:

## print(f"Error loading icons: {e}")

## # Fallback to text if images fail to load

## self.pause\_icon = None

## self.play\_icon = None

## self.next\_icon = None

## self.previous\_icon = None

## 

## def handle\_event(self, event):

## mouse\_pos = pygame.mouse.get\_pos()

## 

## if event.type == pygame.MOUSEBUTTONDOWN:

## if self.icon\_rect.collidepoint(mouse\_pos):

## self.visible = not self.visible

## return True

## 

## if self.visible:

## # Check slider knob

## knob\_rect = pygame.Rect(self.knob\_x - self.knob\_radius, self.knob\_y - self.knob\_radius,

## self.knob\_radius \* 2, self.knob\_radius \* 2)

## if knob\_rect.collidepoint(mouse\_pos):

## self.dragging = True

## return True

## 

## # Check music control buttons

## if self.prev\_btn.collidepoint(mouse\_pos):

## music\_player.global\_music.previous\_song()

## return True

## elif self.play\_pause\_btn.collidepoint(mouse\_pos):

## music\_player.global\_music.toggle\_play\_pause()

## return True

## elif self.next\_btn.collidepoint(mouse\_pos):

## music\_player.global\_music.next\_song()

## return True

## 

## # Click outside panel to close

## if not self.panel\_rect.collidepoint(mouse\_pos):

## self.visible = False

## return True

## 

## elif event.type == pygame.MOUSEBUTTONUP:

## self.dragging = False

## 

## elif event.type == pygame.MOUSEMOTION and self.dragging:

## self.knob\_x = max(self.slider\_rect.left, min(event.pos[0], self.slider\_rect.right))

## volume = (self.knob\_x - self.slider\_rect.left) / self.slider\_rect.width

## music\_player.global\_music.set\_volume(volume)

## return True

## 

## return False

## 

## def draw(self, screen):

## # Draw volume icon (always visible)

## pygame.draw.rect(screen, (100, 100, 100), self.icon\_rect, border\_radius=5)

## pygame.draw.rect(screen, (200, 200, 200), self.icon\_rect, 2, border\_radius=5)

## 

## # Volume icon symbol

## icon\_font = pygame.font.SysFont('Arial', 20)

## icon\_text = icon\_font.render("♪", True, (255, 255, 255))

## screen.blit(icon\_text, (self.icon\_rect.centerx - 6, self.icon\_rect.centery - 10))

## 

## # Draw volume panel if visible

## if self.visible:

## # Panel background

## pygame.draw.rect(screen, (50, 50, 50), self.panel\_rect, border\_radius=8)

## pygame.draw.rect(screen, (150, 150, 150), self.panel\_rect, 2, border\_radius=8)

## 

## # Volume text

## volume\_text = self.font.render("Volume", True, (255, 255, 255))

## screen.blit(volume\_text, (self.panel\_rect.x + 10, self.panel\_rect.y + 15))

## 

## # Volume percentage

## volume\_percent = int(music\_player.global\_music.get\_volume() \* 100)

## percent\_text = self.font.render(f"{volume\_percent}%", True, (200, 200, 200))

## screen.blit(percent\_text, (self.panel\_rect.right - 40, self.panel\_rect.y + 15))

## 

## # Slider track

## pygame.draw.rect(screen, (100, 100, 100), self.slider\_rect)

## pygame.draw.rect(screen, (150, 150, 150), self.slider\_rect, 1)

## 

## # Slider knob

## pygame.draw.circle(screen, (100, 200, 200), (self.knob\_x, self.knob\_y), self.knob\_radius)

## pygame.draw.circle(screen, (50, 50, 50), (self.knob\_x, self.knob\_y), self.knob\_radius, 2)

## 

## # Music control buttons

## pygame.draw.rect(screen, (100, 100, 100), self.prev\_btn, border\_radius=3)

## pygame.draw.rect(screen, (100, 100, 100), self.play\_pause\_btn, border\_radius=3)

## pygame.draw.rect(screen, (100, 100, 100), self.next\_btn, border\_radius=3)

## 

## # Draw icons or fallback text

## if self.previous\_icon:

## screen.blit(self.previous\_icon, (self.prev\_btn.centerx - 10, self.prev\_btn.centery - 10))

## else:

## prev\_text = self.font.render("⏮", True, (255, 255, 255))

## screen.blit(prev\_text, (self.prev\_btn.centerx - 6, self.prev\_btn.centery - 6))

## 

## if music\_player.global\_music.is\_playing():

## if self.pause\_icon:

## screen.blit(self.pause\_icon, (self.play\_pause\_btn.centerx - 10, self.play\_pause\_btn.centery - 10))

## else:

## pause\_text = self.font.render("⏸", True, (255, 255, 255))

## screen.blit(pause\_text, (self.play\_pause\_btn.centerx - 6, self.play\_pause\_btn.centery - 6))

## else:

## if self.play\_icon:

## screen.blit(self.play\_icon, (self.play\_pause\_btn.centerx - 10, self.play\_pause\_btn.centery - 10))

## else:

## play\_text = self.font.render("▶", True, (255, 255, 255))

## screen.blit(play\_text, (self.play\_pause\_btn.centerx - 6, self.play\_pause\_btn.centery - 6))

## 

## if self.next\_icon:

## screen.blit(self.next\_icon, (self.next\_btn.centerx - 10, self.next\_btn.centery - 10))

## else:

## next\_text = self.font.render("⏭", True, (255, 255, 255))

## screen.blit(next\_text, (self.next\_btn.centerx - 6, self.next\_btn.centery - 6))

## def get\_current\_user():

## """Get current user from file - simple implementation"""

## try:

## with open("current\_user.txt", "r") as f:

## lines = f.readlines()

## if len(lines) >= 2:

## user\_id = int(lines[0].strip())

## username = lines[1].strip()

## return user\_id, username

## except:

## pass

## return 1, "Default User" # Fallback to default user

## # In the main() function of main.py, remove the settings\_button definition and drawing

## def main():

## pygame.init()

## screen\_width = 1000

## screen\_height = 600

## screen = pygame.display.set\_mode((screen\_width, screen\_height))

## pygame.display.set\_caption("FOCUS")

## clock = pygame.time.Clock()

## cap = cv2.VideoCapture('snowfall.mp4')

## 

## # Volume controls

## volume\_controls = VolumeControls(screen\_width)

## 

## # Add analytics button (top left corner)

## analytics\_button = ImageButton(20, 20, 50, 50, 'Leaderboard.png')

## 

## 

## # Animation variables for smooth transitions

## guy\_scale = 1.0 # Current scale for guy image

## guy\_target\_scale = 1.0 # Target scale

## focus\_glow\_alpha = 0 # Glow effect alpha

## fade\_in\_alpha = 0 # Startup fade-in

## 

## font\_small = pygame.font.Font("font.ttf", 20)

## def get\_video\_frame():

## ret, frame = cap.read()

## if ret:

## frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

## frame = cv2.resize(frame, (screen\_width, screen\_height))

## return pygame.surfarray.make\_surface(frame.swapaxes(0, 1))

## else:

## cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

## ret, frame = cap.read()

## frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

## frame = cv2.resize(frame, (screen\_width, screen\_height))

## return pygame.surfarray.make\_surface(frame.swapaxes(0, 1))

## guy\_image = pygame.image.load('guy.png').convert\_alpha()

## guy\_image = pygame.transform.scale(guy\_image, (310, 310))

## guy\_rect = guy\_image.get\_rect(center=(screen\_width//2, screen\_height//2 + 150))

## guy\_hitbox = pygame.Rect(guy\_rect.left, guy\_rect.top + 75, guy\_rect.width, guy\_rect.height - 100)

## font = pygame.font.SysFont('timesnewroman', 175)

## focus\_text = font.render("F O C U S", True, (255, 255, 255))

## focus\_rect = focus\_text.get\_rect(center=(screen\_width//2, screen\_height//2 - 50))

## font\_large = pygame.font.SysFont('timesnewroman', 200)

## focus\_text\_large = font\_large.render("F O C U S", True, (255, 255, 255))

## focus\_rect\_large = focus\_text\_large.get\_rect(center=(screen\_width//2, screen\_height//2 - 50))

## running = True

## focus\_hover = False

## guy\_hover = False

## while running:

## for event in pygame.event.get():

## if event.type == pygame.QUIT:

## running = False

## 

## # Handle volume controls first

## if volume\_controls.handle\_event(event):

## continue

## 

## elif event.type == pygame.MOUSEBUTTONDOWN:

## if analytics\_button.is\_clicked(event.pos):

## print("Analytics clicked")

## pygame.quit()

## cap.release()

## try:

## subprocess.run([sys.executable, "AnalyticsDashboard.py"])

## except:

## subprocess.run([sys.executable, "Leaderboard.py"])

## sys.exit() # Exit main, analytics will restart main when done

## 

## elif guy\_hitbox.collidepoint(event.pos):

## print("Guy clicked - opening settings")

## pygame.quit()

## cap.release()

## subprocess.run([sys.executable, "settings.py"])

## sys.exit() # Exit main, settings will restart main when done

## 

## elif focus\_rect.collidepoint(event.pos):

## print("="\*60)

## print("FOCUS CLICKED")

## print("="\*60)

## import os

## print(f"current\_user.txt exists: {os.path.exists('current\_user.txt')}")

## user\_id, username = get\_current\_user()

## print(f"Got user: {username} (ID: {user\_id})")

## pygame.quit()

## cap.release()

## subprocess.run([sys.executable, "timer.py", str(user\_id), username])

## sys.exit() # Exit main, timer will restart main when done

## mouse\_pos = pygame.mouse.get\_pos()

## focus\_hover = focus\_rect.collidepoint(mouse\_pos)

## guy\_hover = guy\_hitbox.collidepoint(mouse\_pos)

## analytics\_button.update\_hover(mouse\_pos)

## 

## # Smooth animations

## guy\_target\_scale = 1.16 if guy\_hover else 1.0

## guy\_scale += (guy\_target\_scale - guy\_scale) \* 0.3 # Smooth interpolation

## 

## focus\_glow\_alpha = min(255, focus\_glow\_alpha + 15) if focus\_hover else max(0, focus\_glow\_alpha - 15)

## 

## # Fade in on startup

## if fade\_in\_alpha < 255:

## fade\_in\_alpha = min(255, fade\_in\_alpha + 5)

## 

## screen.fill((0, 0, 0))

## 

## video\_frame = get\_video\_frame()

## screen.blit(video\_frame, (0, 0))

## 

## # Draw FOCUS text with glow effect on hover

## if focus\_glow\_alpha > 0:

## # Create glow surface

## glow\_surf = focus\_text\_large.copy()

## glow\_surf.set\_alpha(focus\_glow\_alpha // 2)

## screen.blit(glow\_surf, focus\_rect\_large)

## 

## if focus\_hover:

## screen.blit(focus\_text\_large, focus\_rect\_large)

## else:

## screen.blit(focus\_text, focus\_rect)

## 

## # Draw guy with smooth scaling

## current\_size = int(310 \* guy\_scale)

## scaled\_guy = pygame.transform.smoothscale(guy\_image, (current\_size, current\_size))

## scaled\_rect = scaled\_guy.get\_rect(center=guy\_rect.center)

## screen.blit(scaled\_guy, scaled\_rect)

## 

## # Draw volume controls

## volume\_controls.draw(screen)

## 

## # Draw analytics button

## analytics\_button.draw(screen)

## 

## 

## # REMOVED: settings\_button.draw(screen, font\_small)

## # Fade-in overlay

## if fade\_in\_alpha < 255:

## fade\_overlay = pygame.Surface((screen\_width, screen\_height))

## fade\_overlay.fill((0, 0, 0))

## fade\_overlay.set\_alpha(255 - fade\_in\_alpha)

## screen.blit(fade\_overlay, (0, 0))

## 

## pygame.display.flip()

## clock.tick(60) # Increased to 60 FPS for smoother animations

## pygame.quit()

## cap.release()

## sys.exit()

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

## import subprocess

## import sys

## import os

## 

## print("="\*60)

## print("MAIN.PY STARTED")

## print("="\*60)

## 

## # Check if session\_active.flag exists

## # If NO flag = Initial launch from batch file (delete old session, require login)

## # If flag EXISTS = Restart from timer/settings/analytics (keep session)

## 

## if not os.path.exists("session\_active.flag"):

## # Initial launch - clean up any old session files

## print("Initial launch - clearing old session")

## if os.path.exists("current\_user.txt"):

## os.remove("current\_user.txt")

## print("✓ Removed old current\_user.txt")

## 

## # Create flag to indicate session is now active

## with open("session\_active.flag", "w") as f:

## f.write("1")

## 

## # Open login screen

## subprocess.run([sys.executable, "login\_screen.py"])

## 

## # Check if user logged in

## if not os.path.exists("current\_user.txt"):

## print("Login cancelled. Exiting...")

## # Clean up flag

## if os.path.exists("session\_active.flag"):

## os.remove("session\_active.flag")

## sys.exit(0)

## else:

## # Restart from timer/settings/analytics - keep session

## print("Restarting from subprocess - keeping session")

## if not os.path.exists("current\_user.txt"):

## print("ERROR: Session lost! Requiring login...")

## subprocess.run([sys.executable, "login\_screen.py"])

## if not os.path.exists("current\_user.txt"):

## print("Login cancelled. Exiting...")

## sys.exit(0)

## 

## # User logged in, run main app

## print("DEBUG: About to call main()")

## main()

## print("DEBUG: main() returned")

## 

## music\_player.py

import pygame.mixer as mixer

import os

import threading

import glob

import yt\_dlp

import random

class GlobalMusicPlayer:

    def \_\_init\_\_(self):

        self.music\_state = "stopped"

        self.current\_volume = 0.5

        self.playlist = []

        self.current\_index = 0

        self.music\_folder = "focus\_songs"

        # Callback for UI refresh

        self.on\_playlist\_changed = None

        if not os.path.exists(self.music\_folder):

            os.makedirs(self.music\_folder)

        mixer.init()

        mixer.music.set\_volume(self.current\_volume)

        self.refresh\_playlist()

        # Auto-play on startup if there are songs

        if self.playlist:

            self.play\_random()

    def refresh\_playlist(self):

        self.playlist = []

        audio\_extensions = ['\*.mp3', '\*.wav', '\*.ogg', '\*.m4a', '\*.flac']

        for extension in audio\_extensions:

            pattern = os.path.join(self.music\_folder, extension)

            self.playlist.extend(glob.glob(pattern))

        self.playlist.sort()

        # Notify UI about playlist change

        if self.on\_playlist\_changed:

            self.on\_playlist\_changed()

    def play\_random(self):

        """Play a random song from the playlist"""

        if not self.playlist:

            return

        try:

            mixer.music.stop()

            # Select random song

            self.current\_index = random.randint(0, len(self.playlist) - 1)

            song\_path = self.playlist[self.current\_index]

            mixer.music.load(song\_path)

            mixer.music.play(-1)  # -1 means loop indefinitely

            self.music\_state = "playing"

        except Exception as e:

            print(f"Play error: {e}")

    def download\_by\_name(self, song\_name):

        def download():

            try:

                ydl\_opts = {

                    'format': 'bestaudio/best',

                    'outtmpl': f'{self.music\_folder}/%(title)s.%(ext)s',

                    'postprocessors': [{

                        'key': 'FFmpegExtractAudio',

                        'preferredcodec': 'mp3',

                        'preferredquality': '192',

                    }],

                    'default\_search': 'ytsearch',

                }

                with yt\_dlp.YoutubeDL(ydl\_opts) as ydl:

                    ydl.download([song\_name])

                self.refresh\_playlist()

                # Auto-play the newly downloaded song

                if self.playlist:

                    self.play\_random()

                return True

            except Exception as e:

                print(f"Download error: {e}")

                return False

        thread = threading.Thread(target=download, daemon=True)

        thread.start()

        return True

    def download\_from\_spotify(self, search\_term):

        """Search and download by song name"""

        return self.download\_by\_name(search\_term)

    def play(self, index=None):

        """Play specific song or random if no index provided"""

        if not self.playlist:

            return

        try:

            mixer.music.stop()

            if index is not None and 0 <= index < len(self.playlist):

                self.current\_index = index

                song\_path = self.playlist[index]

            else:

                # Play random song if no specific index

                self.play\_random()

                return

            mixer.music.load(song\_path)

            mixer.music.play(-1)

            self.music\_state = "playing"

        except Exception as e:

            print(f"Play error: {e}")

    def toggle\_play\_pause(self):

        """Toggle between play and pause - this is the main control method"""

        if not self.playlist:

            return

        if self.music\_state == "playing":

            mixer.music.pause()

            self.music\_state = "paused"

        elif self.music\_state == "paused":

            mixer.music.unpause()

            self.music\_state = "playing"

        else:  # stopped

            self.play\_random()

    def next\_song(self):

        """Play next random song"""

        if not self.playlist:

            return

        self.play\_random()

    def previous\_song(self):

        """Play previous random song"""

        if not self.playlist:

            return

        self.play\_random()

    def set\_volume(self, volume):

        self.current\_volume = max(0.0, min(1.0, volume))

        mixer.music.set\_volume(self.current\_volume)

    def get\_volume(self):

        return self.current\_volume

    def remove\_song(self, index):

        if 0 <= index < len(self.playlist):

            try:

                # Stop music if it's playing the song to be deleted

                if index == self.current\_index and self.music\_state == "playing":

                    mixer.music.stop()

                    self.music\_state = "stopped"

                song\_path = self.playlist[index]

                os.remove(song\_path)

                self.refresh\_playlist()

                # If songs remain, play a random one

                if self.playlist:

                    self.play\_random()

                return True

            except Exception as e:

                print(f"Remove error: {e}")

        return False

    def get\_current\_song(self):

        if self.playlist and 0 <= self.current\_index < len(self.playlist):

            return os.path.basename(self.playlist[self.current\_index])

        return "No song"

    def get\_playlist(self):

        return [os.path.basename(song) for song in self.playlist]

    def is\_playing(self):

        """Check if music is currently playing"""

        return self.music\_state == "playing"

    def get\_play\_pause\_icon(self):

        """Return the appropriate icon for play/pause state"""

        return "⏸" if self.is\_playing() else "▶"

# Global instance

global\_music = GlobalMusicPlayer()

## requirements.txt

Flask==2.3.3

flask-cors==4.0.0

PyJWT==2.8.0

mysql-connector-python==8.1.0

gunicorn==21.2.0

## run\_as\_admin.bat

@echo off

chcp 65001 >nul

echo Starting Focus Timer with Administrator privileges...

echo.

:: Check if Python is available

python --version >nul 2>&1

if errorlevel 1 (

    echo Error: Python is not installed or not in PATH

    echo Please install Python and ensure it's in your system PATH

    pause

    exit /b 1

)

:: Run the admin launcher

python admin\_launcher.py

:: If there's an error, pause so user can see the message

if errorlevel 1 pause

## settings.py

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

import pygame

import sys

import json

import os

import subprocess

# Initialize pygame

pygame.init()

# Screen dimensions

SCREEN\_INFO = pygame.display.Info()

SCREEN\_WIDTH, SCREEN\_HEIGHT = SCREEN\_INFO.current\_w, SCREEN\_INFO.current\_h

# Colors - UNIFIED THEME (matching timer.py and analytics)

BACKGROUND = (25, 25, 50)

PANEL\_BG = (40, 40, 80)

DARK\_PANEL = (30, 30, 60)

TEXT\_COLOR = (255, 255, 255)

HIGHLIGHT = (100, 200, 255)

ACCENT\_SILVER = (192, 192, 192)

ACCENT\_GOLD = (255, 215, 0)

BUTTON\_PRIMARY = (70, 130, 180)

BUTTON\_HOVER = (100, 160, 210)

BUTTON\_SUCCESS = (80, 180, 120)

BUTTON\_DANGER = (255, 100, 100)

INPUT\_BG = (40, 40, 60)

BORDER\_COLOR = (100, 100, 150)

SHADOW\_COLOR = (15, 15, 30)

SUCCESS\_GREEN = (100, 200, 150)

ERROR\_RED = (255, 100, 100)

WARNING\_YELLOW = (255, 200, 0)

# Fonts - consistent with timer

title\_font = pygame.font.SysFont('timesnewroman', 72, bold=True)

header\_font = pygame.font.SysFont('timesnewroman', 42, bold=True)

subheader\_font = pygame.font.SysFont('timesnewroman', 32, bold=True)

text\_font = pygame.font.SysFont('timesnewroman', 24)

small\_font = pygame.font.SysFont('timesnewroman', 20)

tiny\_font = pygame.font.SysFont('timesnewroman', 18)

button\_font = pygame.font.SysFont('timesnewroman', 22, bold=True)

def get\_database\_connection():

    """Create a database connection"""

    try:

        config\_file = "mysql\_config.json"

        if os.path.exists(config\_file):

            with open(config\_file, 'r') as f:

                mysql\_config = json.load(f)

        else:

            mysql\_config = {

                'host': 'localhost',

                'user': 'root',

                'password': '',

                'database': 'focus\_app',

                'port': 3306

            }

        from database import Database

        return Database(mysql\_config)

    except Exception as e:

        print(f"Database connection error: {e}")

        return None

def get\_current\_user():

    """Get current user from file"""

    try:

        with open("current\_user.txt", "r") as f:

            lines = f.readlines()

            if len(lines) >= 2:

                user\_id = int(lines[0].strip())

                username = lines[1].strip()

                return user\_id, username

    except:

        pass

    return None, None

def get\_all\_users():

    """Get all users from database"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            cursor.execute("SELECT id, username FROM users ORDER BY username")

            users = cursor.fetchall()

            cursor.close()

            db.connection.close()

            return users

    except Exception as e:

        print(f"Error loading users: {e}")

    return []

def get\_blocked\_sites(user\_id):

    """Get user's blocked sites"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            cursor.execute("SELECT website FROM blocked\_sites WHERE user\_id = %s ORDER BY website", (user\_id,))

            sites = cursor.fetchall()

            cursor.close()

            db.connection.close()

            return [site['website'] for site in sites]

    except Exception as e:

        print(f"Error loading blocked sites: {e}")

    return []

def add\_blocked\_site(user\_id, website):

    """Add a blocked site"""

    try:

        website = website.strip().lower()

        if not website:

            return False, "Website cannot be empty"

        # Remove http:// or https://

        website = website.replace('http://', '').replace('https://', '')

        website = website.replace('www.', '')

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor()

            # Check if already exists

            cursor.execute("SELECT id FROM blocked\_sites WHERE user\_id = %s AND website = %s",

                          (user\_id, website))

            if cursor.fetchone():

                cursor.close()

                db.connection.close()

                return False, "Website already blocked"

            # Try to add site

            try:

                cursor.execute("INSERT INTO blocked\_sites (user\_id, website) VALUES (%s, %s)",

                              (user\_id, website))

                db.connection.commit()

                cursor.close()

                db.connection.close()

                return True, "Website added successfully"

            except Exception as insert\_error:

                error\_msg = str(insert\_error)

                cursor.close()

                db.connection.close()

                # Check for foreign key constraint error

                if "foreign key constraint" in error\_msg.lower():

                    return False, f"User ID {user\_id} not found in database. Please log in again."

                elif "1452" in error\_msg:  # MySQL foreign key error code

                    return False, f"User ID {user\_id} not found in database. Please log in again."

                else:

                    return False, f"Error: {error\_msg}"

    except Exception as e:

        print(f"Error adding blocked site: {e}")

        import traceback

        traceback.print\_exc()

        return False, f"Error: {str(e)}"

    return False, "Could not connect to database"

def remove\_blocked\_site(user\_id, website):

    """Remove a blocked site"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor()

            cursor.execute("DELETE FROM blocked\_sites WHERE user\_id = %s AND website = %s",

                          (user\_id, website))

            db.connection.commit()

            cursor.close()

            return True, "Website removed successfully"

    except Exception as e:

        print(f"Error removing blocked site: {e}")

        return False, f"Error: {str(e)}"

    return False, "Could not connect to database"

class Button:

    def \_\_init\_\_(self, x, y, width, height, text, color=BUTTON\_PRIMARY, hover\_color=BUTTON\_HOVER):

        self.rect = pygame.Rect(x, y, width, height)

        self.text = text

        self.color = color

        self.hover\_color = hover\_color

        self.hovered = False

    def draw(self, screen):

        # Shadow

        shadow = pygame.Rect(self.rect.x + 4, self.rect.y + 4,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Button

        color = self.hover\_color if self.hovered else self.color

        pygame.draw.rect(screen, color, self.rect, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=12)

        # Text

        text\_surface = button\_font.render(self.text, True, TEXT\_COLOR)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.hovered = self.rect.collidepoint(mouse\_pos)

    def is\_clicked(self, mouse\_pos, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            return self.rect.collidepoint(mouse\_pos)

        return False

def draw\_input\_box(screen, x, y, width, height, text, active, label=""):

    """Draw an input box with label"""

    # Label

    if label:

        label\_surf = text\_font.render(label, True, TEXT\_COLOR)

        screen.blit(label\_surf, (x, y - 35))

    # Shadow

    shadow = pygame.Rect(x + 3, y + 3, width, height)

    pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=10)

    # Box

    color = HIGHLIGHT if active else INPUT\_BG

    pygame.draw.rect(screen, color, (x, y, width, height), border\_radius=10)

    pygame.draw.rect(screen, BORDER\_COLOR, (x, y, width, height), 2, border\_radius=10)

    # Text - only show actual text, no placeholder here

    display\_text = text

    if active and text:

        display\_text = text + "|"

    elif active and not text:

        display\_text = "|"

    if display\_text:

        text\_surface = text\_font.render(display\_text, True, TEXT\_COLOR)

        screen.blit(text\_surface, (x + 15, y + 12))

def settings\_screen():

    """Main settings screen"""

    # Create screen

    screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

    pygame.display.set\_caption("Focus Timer - Settings")

    clock = pygame.time.Clock()

    # Get current user

    user\_id, username = get\_current\_user()

    if not user\_id:

        # No user logged in - redirect to login

        print("No user logged in! Please login first.")

        pygame.quit()

        import login\_screen

        result = login\_screen.login\_screen()

        if result[0]:

            user\_id, username = result

            pygame.init()

            screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

            pygame.display.set\_caption("Focus Timer - Settings")

        else:

            return

    # Verify user exists in database

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

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

            user\_data = cursor.fetchone()

            cursor.close()

            db.connection.close()

            if not user\_data:

                # User doesn't exist in database anymore

                print(f"User ID {user\_id} not found in database!")

                pygame.quit()

                import login\_screen

                result = login\_screen.login\_screen()

                if result[0]:

                    user\_id, username = result

                    pygame.init()

                    screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

                    pygame.display.set\_caption("Focus Timer - Settings")

                else:

                    return

    except Exception as e:

        print(f"Error validating user: {e}")

    # State

    current\_tab = "account"  # account, websites, database, about

    blocked\_sites = get\_blocked\_sites(user\_id)  # Load at startup, not lazily

    website\_input = ""

    website\_input\_active = False

    message = ""

    message\_color = TEXT\_COLOR

    all\_users\_list = []  # For account switching

    user\_switch\_rects = []  # For click detection

    # UI Elements

    back\_button = Button(40, 40, 140, 50, "← Back")

    # Tab buttons

    tab\_y = 150

    tab\_width = 200

    tab\_height = 60

    tab\_spacing = 20

    tab\_x\_start = SCREEN\_WIDTH // 2 - (4 \* tab\_width + 3 \* tab\_spacing) // 2

    account\_tab = Button(tab\_x\_start, tab\_y, tab\_width, tab\_height, "Account")

    websites\_tab = Button(tab\_x\_start + tab\_width + tab\_spacing, tab\_y, tab\_width, tab\_height, "Websites")

    database\_tab = Button(tab\_x\_start + 2 \* (tab\_width + tab\_spacing), tab\_y, tab\_width, tab\_height, "Database")

    about\_tab = Button(tab\_x\_start + 3 \* (tab\_width + tab\_spacing), tab\_y, tab\_width, tab\_height, "About")

    # Content area

    content\_y = tab\_y + tab\_height + 40

    content\_width = SCREEN\_WIDTH - 200

    content\_height = SCREEN\_HEIGHT - content\_y - 100

    content\_x = 100

    # Website blocking elements

    website\_input\_rect = pygame.Rect(content\_x + 50, content\_y + 100, 400, 50)

    add\_website\_button = Button(content\_x + 470, content\_y + 100, 150, 50, "+ Add")

    # User management button

    manage\_users\_button = Button(SCREEN\_WIDTH - 220, 40, 180, 50, "Manage Users")

    running = True

    while running:

        mouse\_pos = pygame.mouse.get\_pos()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                running = False

            elif event.type == pygame.MOUSEBUTTONDOWN:

                # Back button

                if back\_button.is\_clicked(mouse\_pos, event):

                    pygame.quit()

                    subprocess.run([sys.executable, "main.py"])

                    return

                # Manage users button

                if manage\_users\_button.is\_clicked(mouse\_pos, event):

                    running = False  # Exit the loop first

                    pygame.quit()

                    import user\_management

                    user\_management.user\_management\_screen(first\_time=False)

                    return

                # Tab buttons

                if account\_tab.is\_clicked(mouse\_pos, event):

                    current\_tab = "account"

                    message = ""

                    # Load users for account switching

                    all\_users\_list = get\_all\_users()

                elif websites\_tab.is\_clicked(mouse\_pos, event):

                    current\_tab = "websites"

                    message = ""

                elif database\_tab.is\_clicked(mouse\_pos, event):

                    current\_tab = "database"

                    message = ""

                elif about\_tab.is\_clicked(mouse\_pos, event):

                    current\_tab = "about"

                    message = ""

                # Account tab - user switching

                if current\_tab == "account" and user\_switch\_rects:

                    for rect, user in user\_switch\_rects:

                        if rect.collidepoint(mouse\_pos) and user['id'] != user\_id:

                            # Switch to this user

                            try:

                                with open("current\_user.txt", "w") as f:

                                    f.write(f"{user['id']}\n{user['username']}")

                                print(f"Switched to user: {user['username']}")

                                running = False  # Exit the loop

                            except Exception as e:

                                print(f"Error switching user: {e}")

                                message = "Error switching account"

                                message\_color = ERROR\_RED

                            break

                # Website blocking tab actions

                if current\_tab == "websites":

                    if website\_input\_rect.collidepoint(mouse\_pos):

                        website\_input\_active = True

                    else:

                        website\_input\_active = False

                    # Add website button

                    if add\_website\_button.is\_clicked(mouse\_pos, event):

                        if website\_input:

                            success, msg = add\_blocked\_site(user\_id, website\_input)

                            message = msg

                            message\_color = SUCCESS\_GREEN if success else ERROR\_RED

                            if success:

                                website\_input = ""

                                blocked\_sites = get\_blocked\_sites(user\_id)

                    # Click on remove button

                    list\_y = content\_y + 200

                    for i, site in enumerate(blocked\_sites[:15]):

                        site\_rect = pygame.Rect(content\_x + 50, list\_y + i \* 45, content\_width - 150, 40)

                        remove\_btn = pygame.Rect(site\_rect.right + 10, site\_rect.y, 60, 40)

                        if remove\_btn.collidepoint(mouse\_pos):

                            success, msg = remove\_blocked\_site(user\_id, site)

                            message = msg

                            message\_color = SUCCESS\_GREEN if success else ERROR\_RED

                            if success:

                                blocked\_sites = get\_blocked\_sites(user\_id)

                            break

            elif event.type == pygame.KEYDOWN and website\_input\_active:

                if event.key == pygame.K\_RETURN:

                    if website\_input:

                        success, msg = add\_blocked\_site(user\_id, website\_input)

                        message = msg

                        message\_color = SUCCESS\_GREEN if success else ERROR\_RED

                        if success:

                            website\_input = ""

                            blocked\_sites = get\_blocked\_sites(user\_id)

                elif event.key == pygame.K\_BACKSPACE:

                    website\_input = website\_input[:-1]

                elif event.key == pygame.K\_ESCAPE:

                    website\_input\_active = False

                elif len(website\_input) < 100:

                    website\_input += event.unicode

        # Update hover states

        back\_button.check\_hover(mouse\_pos)

        manage\_users\_button.check\_hover(mouse\_pos)

        account\_tab.check\_hover(mouse\_pos)

        websites\_tab.check\_hover(mouse\_pos)

        database\_tab.check\_hover(mouse\_pos)

        about\_tab.check\_hover(mouse\_pos)

        if current\_tab == "websites":

            add\_website\_button.check\_hover(mouse\_pos)

        # Skip drawing if we're exiting

        if not running:

            continue

        # Draw everything

        screen.fill(BACKGROUND)

        # Draw header

        header\_rect = pygame.Rect(0, 0, SCREEN\_WIDTH, 120)

        pygame.draw.rect(screen, DARK\_PANEL, header\_rect)

        # Title

        title = header\_font.render("SETTINGS", True, HIGHLIGHT)

        screen.blit(title, (SCREEN\_WIDTH // 2 - title.get\_width() // 2, 35))

        # User info

        user\_text = text\_font.render(f"Logged in as: {username}", True, ACCENT\_SILVER)

        screen.blit(user\_text, (SCREEN\_WIDTH // 2 - user\_text.get\_width() // 2, 90))

        # Draw buttons

        back\_button.draw(screen)

        manage\_users\_button.draw(screen)

        # Draw tabs

        for tab in [account\_tab, websites\_tab, database\_tab, about\_tab]:

            # Highlight active tab

            if (tab == account\_tab and current\_tab == "account") or \

               (tab == websites\_tab and current\_tab == "websites") or \

               (tab == database\_tab and current\_tab == "database") or \

               (tab == about\_tab and current\_tab == "about"):

                # Draw active tab with highlight

                pygame.draw.rect(screen, HIGHLIGHT, tab.rect, border\_radius=12)

                pygame.draw.rect(screen, BORDER\_COLOR, tab.rect, 2, border\_radius=12)

                text\_surf = button\_font.render(tab.text, True, BACKGROUND)

                screen.blit(text\_surf, text\_surf.get\_rect(center=tab.rect.center))

            else:

                tab.draw(screen)

        # Draw content panel

        content\_panel = pygame.Rect(content\_x, content\_y, content\_width, content\_height)

        pygame.draw.rect(screen, PANEL\_BG, content\_panel, border\_radius=15)

        pygame.draw.rect(screen, BORDER\_COLOR, content\_panel, 2, border\_radius=15)

        # Draw tab content

        if current\_tab == "account":

            # Clear user switch rects for this frame

            user\_switch\_rects = []

            # Account settings

            content\_title = subheader\_font.render("Account Settings", True, HIGHLIGHT)

            screen.blit(content\_title, (content\_x + 50, content\_y + 30))

            y\_offset = content\_y + 100

            # Current account display

            current\_label = text\_font.render(f"Current Account: {username}", True, TEXT\_COLOR)

            screen.blit(current\_label, (content\_x + 50, y\_offset))

            y\_offset += 60

            # User ID

            id\_label = text\_font.render(f"User ID: {user\_id}", True, ACCENT\_SILVER)

            screen.blit(id\_label, (content\_x + 50, y\_offset))

            y\_offset += 100

            # Switch account section

            switch\_title = text\_font.render("Switch Account:", True, TEXT\_COLOR)

            screen.blit(switch\_title, (content\_x + 50, y\_offset))

            y\_offset += 50

            # Get all users if not loaded

            if not all\_users\_list:

                all\_users\_list = get\_all\_users()

            # Display user buttons for switching

            for i, user in enumerate(all\_users\_list[:8]):  # Show up to 8 users

                user\_y = y\_offset + i \* 55

                user\_rect = pygame.Rect(content\_x + 50, user\_y, 350, 45)

                # Store rect for click detection

                user\_switch\_rects.append((user\_rect, user))

                # Highlight current user

                if user['id'] == user\_id:

                    pygame.draw.rect(screen, HIGHLIGHT, user\_rect, border\_radius=10)

                    pygame.draw.rect(screen, BORDER\_COLOR, user\_rect, 2, border\_radius=10)

                    user\_text = text\_font.render(f"✓ {user['username']} (current)", True, BACKGROUND)

                    screen.blit(user\_text, (user\_rect.x + 15, user\_rect.y + 10))

                else:

                    # Other users - clickable

                    is\_hovered = user\_rect.collidepoint(mouse\_pos)

                    btn\_color = BUTTON\_HOVER if is\_hovered else BUTTON\_PRIMARY

                    pygame.draw.rect(screen, btn\_color, user\_rect, border\_radius=10)

                    pygame.draw.rect(screen, BORDER\_COLOR, user\_rect, 2, border\_radius=10)

                    user\_text = text\_font.render(user['username'], True, TEXT\_COLOR)

                    screen.blit(user\_text, (user\_rect.x + 15, user\_rect.y + 10))

            # Help text

            if len(all\_users\_list) > 1:

                help\_y = y\_offset + len(all\_users\_list[:8]) \* 55 + 30

                help\_text = small\_font.render("Click on an account to switch", True, ACCENT\_SILVER)

                screen.blit(help\_text, (content\_x + 50, help\_y))

        elif current\_tab == "websites":

            # Website blocking

            content\_title = subheader\_font.render("Blocked Websites", True, HIGHLIGHT)

            screen.blit(content\_title, (content\_x + 50, content\_y + 30))

            # Instructions

            instr = small\_font.render("Add websites to block during focus sessions:", True, ACCENT\_SILVER)

            screen.blit(instr, (content\_x + 50, content\_y + 70))

            # Input field

            draw\_input\_box(screen, website\_input\_rect.x, website\_input\_rect.y,

                          website\_input\_rect.width, website\_input\_rect.height,

                          website\_input, website\_input\_active, "")

            # Placeholder text only if empty AND not active

            if not website\_input and not website\_input\_active:

                placeholder = text\_font.render("e.g., youtube.com", True, ACCENT\_SILVER)

                screen.blit(placeholder, (website\_input\_rect.x + 15, website\_input\_rect.y + 12))

            # Add button

            add\_website\_button.draw(screen)

            # List of blocked sites

            list\_y = content\_y + 200

            list\_title = text\_font.render("Currently Blocked:", True, TEXT\_COLOR)

            screen.blit(list\_title, (content\_x + 50, list\_y - 40))

            if blocked\_sites:

                for i, site in enumerate(blocked\_sites[:15]):  # Show first 15

                    site\_y = list\_y + i \* 45

                    # Site card

                    site\_rect = pygame.Rect(content\_x + 50, site\_y, content\_width - 150, 40)

                    pygame.draw.rect(screen, INPUT\_BG, site\_rect, border\_radius=8)

                    pygame.draw.rect(screen, BORDER\_COLOR, site\_rect, 1, border\_radius=8)

                    # Site name

                    site\_text = text\_font.render(site, True, TEXT\_COLOR)

                    screen.blit(site\_text, (site\_rect.x + 15, site\_rect.y + 8))

                    # Remove button

                    remove\_btn = pygame.Rect(site\_rect.right + 10, site\_rect.y, 60, 40)

                    btn\_color = BUTTON\_DANGER if remove\_btn.collidepoint(mouse\_pos) else BUTTON\_PRIMARY

                    pygame.draw.rect(screen, btn\_color, remove\_btn, border\_radius=8)

                    remove\_text = tiny\_font.render("✕", True, TEXT\_COLOR)

                    screen.blit(remove\_text, (remove\_btn.centerx - 6, remove\_btn.centery - 8))

            else:

                no\_sites = small\_font.render("No blocked websites yet", True, ACCENT\_SILVER)

                screen.blit(no\_sites, (content\_x + 50, list\_y + 20))

            # Message

            if message:

                msg\_surf = text\_font.render(message, True, message\_color)

                screen.blit(msg\_surf, (content\_x + 50, content\_y + content\_height - 50))

        elif current\_tab == "database":

            # Database info

            content\_title = subheader\_font.render("Database Configuration", True, HIGHLIGHT)

            screen.blit(content\_title, (content\_x + 50, content\_y + 30))

            y\_offset = content\_y + 100

            # Load config

            try:

                with open("mysql\_config.json", 'r') as f:

                    config = json.load(f)

                info\_items = [

                    f"Host: {config.get('host', 'N/A')}",

                    f"Database: {config.get('database', 'N/A')}",

                    f"Port: {config.get('port', 'N/A')}",

                    f"User: {config.get('user', 'N/A')}"

                ]

                for item in info\_items:

                    item\_text = text\_font.render(item, True, TEXT\_COLOR)

                    screen.blit(item\_text, (content\_x + 50, y\_offset))

                    y\_offset += 40

            except:

                error\_text = text\_font.render("Could not load database config", True, ERROR\_RED)

                screen.blit(error\_text, (content\_x + 50, y\_offset))

            y\_offset += 60

            reconfig\_text = small\_font.render("To reconfigure database, edit mysql\_config.json", True, ACCENT\_SILVER)

            screen.blit(reconfig\_text, (content\_x + 50, y\_offset))

        elif current\_tab == "about":

            # About

            content\_title = subheader\_font.render("About Focus Timer", True, HIGHLIGHT)

            screen.blit(content\_title, (content\_x + 50, content\_y + 30))

            y\_offset = content\_y + 100

            about\_lines = [

                "Focus Timer - Productivity Application",

                "",

                "Version: 2.0",

                "Theme: Deep Blue Professional",

                "",

                "Features:",

                "• Pomodoro timer with website blocking",

                "• Task management",

                "• Analytics dashboard",

                "• Multi-user support",

                "• Music player integration",

                "",

                "Created with Python & Pygame",

                "",

                "Developed by:",

                "Amartya Srinivasan",

                "K Shrivathsan",

                "R Vidhyuthram"

            ]

            for line in about\_lines:

                if line.startswith("•"):

                    line\_text = text\_font.render(line, True, ACCENT\_SILVER)

                elif line in ["Amartya Srinivasan", "K Shrivathsan", "R Vidhyuthram"]:

                    line\_text = text\_font.render(line, True, ACCENT\_GOLD)

                else:

                    line\_text = text\_font.render(line, True, TEXT\_COLOR)

                screen.blit(line\_text, (content\_x + 50, y\_offset))

                y\_offset += 35

        pygame.display.flip()

        clock.tick(60)

    pygame.quit()

    sys.exit()

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

    settings\_screen()

## timer.py

A screenshot of a computer

AI-generated content may be incorrect.

import pygame

import sys

import time

import os

import cv2

import numpy as np

import music\_player

import website\_blocker

import subprocess

import platform

import json

import math

import threading

# Database connection function

def get\_database\_connection():

    """Create a database connection using saved config"""

    try:

        config\_file = "mysql\_config.json"

        if os.path.exists(config\_file):

            with open(config\_file, 'r') as f:

                mysql\_config = json.load(f)

        else:

            mysql\_config = {

                'host': 'localhost',

                'user': 'root',

                'password': '',

                'database': 'focus\_app',

                'port': 3306

            }

        from database import Database

        return Database(mysql\_config)

    except Exception as e:

        print(f"Database connection error: {e}")

        return None

# Initialize pygame

pygame.init()

# Screen dimensions - fullscreen

SCREEN\_INFO = pygame.display.Info()

SCREEN\_WIDTH, SCREEN\_HEIGHT = SCREEN\_INFO.current\_w, SCREEN\_INFO.current\_h

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Focus Timer")

# ===== IMPROVED COLOR SCHEME (matches AnalyticsDashboard) =====

BACKGROUND = (25, 25, 50)           # Deep blue background

PANEL\_BG = (40, 40, 80)             # Lighter blue panels

DARK\_PANEL = (30, 30, 60)           # Darker panels

TEXT\_COLOR = (255, 255, 255)        # White text

HIGHLIGHT = (100, 200, 255)         # Light blue highlights

ACCENT\_GOLD = (255, 215, 0)         # Gold accents

ACCENT\_SILVER = (192, 192, 192)     # Silver accents

# Status colors

SUCCESS\_GREEN = (100, 200, 150)     # Softer green

WARNING\_ORANGE = (255, 180, 100)    # Warm orange

PAUSE\_YELLOW = (255, 220, 100)      # Soft yellow

DANGER\_RED = (255, 100, 100)        # Softer red

# UI colors

BUTTON\_PRIMARY = (70, 130, 180)     # Blue buttons

BUTTON\_HOVER = (100, 160, 210)      # Lighter blue on hover

BUTTON\_SUCCESS = (80, 180, 120)     # Green for start

BUTTON\_DANGER = (200, 80, 80)       # Red for stop

INPUT\_BG = (40, 40, 60)             # Input backgrounds

BORDER\_COLOR = (100, 100, 150)      # Subtle borders

SHADOW\_COLOR = (15, 15, 30)         # Shadow color

# Todo list colors

TODO\_BG = (35, 35, 65, 230)         # Semi-transparent

TODO\_ITEM\_BG = (45, 45, 75)         # Individual items

TODO\_HOVER = (60, 60, 90)           # Hover state

TODO\_COMPLETED = (50, 80, 60)       # Completed items

# ===== IMPROVED FONTS (all Times New Roman for consistency) =====

timer\_font = pygame.font.SysFont("timesnewroman", 140, bold=True)

header\_font = pygame.font.SysFont("timesnewroman", 48, bold=True)

text\_font = pygame.font.SysFont("timesnewroman", 24)

small\_font = pygame.font.SysFont("timesnewroman", 18)

tiny\_font = pygame.font.SysFont("timesnewroman", 14)

button\_font = pygame.font.SysFont("timesnewroman", 20, bold=True)

status\_font = pygame.font.SysFont("timesnewroman", 16)

todo\_font = pygame.font.SysFont("timesnewroman", 18)

todo\_header\_font = pygame.font.SysFont("timesnewroman", 22, bold=True)

class VideoBackground:

    def \_\_init\_\_(self, video\_path):

        try:

            self.cap = cv2.VideoCapture(video\_path)

            if not self.cap.isOpened():

                print(f"Warning: Could not open video file {video\_path}")

                self.cap = None

            self.current\_frame = None

        except Exception as e:

            print(f"Error initializing video background: {e}")

            self.cap = None

    def get\_frame(self):

        if self.cap is None:

            return None

        ret, frame = self.cap.read()

        if not ret:

            self.cap.set(cv2.CAP\_PROP\_POS\_FRAMES, 0)

            ret, frame = self.cap.read()

        if ret:

            frame = cv2.resize(frame, (SCREEN\_WIDTH, SCREEN\_HEIGHT))

            frame = cv2.cvtColor(frame, cv2.COLOR\_BGR2RGB)

            # Darker overlay for better readability

            overlay = np.zeros\_like(frame)

            frame = cv2.addWeighted(frame, 0.6, overlay, 0.4, 0)

            frame = np.rot90(frame)

            self.current\_frame = pygame.surfarray.make\_surface(frame)

        return self.current\_frame

    def release(self):

        if self.cap:

            self.cap.release()

class VolumeControls:

    def \_\_init\_\_(self):

        self.visible = False

        self.dragging = False

        self.icon\_rect = pygame.Rect(SCREEN\_WIDTH - 70, 20, 50, 50)

        self.panel\_rect = pygame.Rect(SCREEN\_WIDTH - 200, 70, 180, 120)

        self.slider\_rect = pygame.Rect(SCREEN\_WIDTH - 190, 100, 160, 15)

        self.knob\_radius = 10

        self.knob\_x = self.slider\_rect.x + (music\_player.global\_music.get\_volume() \* self.slider\_rect.width)

        self.knob\_y = self.slider\_rect.y + self.slider\_rect.height // 2

        self.prev\_btn = pygame.Rect(SCREEN\_WIDTH - 190, 130, 40, 30)

        self.play\_pause\_btn = pygame.Rect(SCREEN\_WIDTH - 140, 130, 40, 30)

        self.next\_btn = pygame.Rect(SCREEN\_WIDTH - 90, 130, 40, 30)

        self.load\_icons()

        self.font = pygame.font.SysFont('timesnewroman', 12)

    def load\_icons(self):

        try:

            pause\_img = pygame.image.load("assets/pause.png").convert\_alpha()

            play\_img = pygame.image.load("assets/play.png").convert\_alpha()

            next\_img = pygame.image.load("assets/next.png").convert\_alpha()

            previous\_img = pygame.image.load("assets/previous.png").convert\_alpha()

            self.pause\_icon = pygame.transform.scale(pause\_img, (20, 20))

            self.play\_icon = pygame.transform.scale(play\_img, (20, 20))

            self.next\_icon = pygame.transform.scale(next\_img, (20, 20))

            self.previous\_icon = pygame.transform.scale(previous\_img, (20, 20))

        except Exception as e:

            print(f"Error loading icons: {e}")

            self.pause\_icon = None

            self.play\_icon = None

            self.next\_icon = None

            self.previous\_icon = None

    def handle\_event(self, event):

        mouse\_pos = pygame.mouse.get\_pos()

        if event.type == pygame.MOUSEBUTTONDOWN:

            if self.icon\_rect.collidepoint(mouse\_pos):

                self.visible = not self.visible

                return True

            if self.visible:

                knob\_rect = pygame.Rect(self.knob\_x - self.knob\_radius, self.knob\_y - self.knob\_radius,

                                      self.knob\_radius \* 2, self.knob\_radius \* 2)

                if knob\_rect.collidepoint(mouse\_pos) or self.slider\_rect.collidepoint(mouse\_pos):

                    self.dragging = True

                    self.knob\_x = max(self.slider\_rect.left, min(mouse\_pos[0], self.slider\_rect.right))

                    volume = (self.knob\_x - self.slider\_rect.left) / self.slider\_rect.width

                    music\_player.global\_music.set\_volume(volume)

                    return True

                if self.prev\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.previous\_song()

                    return True

                elif self.play\_pause\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.toggle\_play\_pause()

                    return True

                elif self.next\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.next\_song()

                    return True

                if not self.panel\_rect.collidepoint(mouse\_pos) and not self.icon\_rect.collidepoint(mouse\_pos):

                    self.visible = False

                    return True

        elif event.type == pygame.MOUSEBUTTONUP:

            self.dragging = False

        elif event.type == pygame.MOUSEMOTION and self.dragging:

            self.knob\_x = max(self.slider\_rect.left, min(event.pos[0], self.slider\_rect.right))

            volume = (self.knob\_x - self.slider\_rect.left) / self.slider\_rect.width

            music\_player.global\_music.set\_volume(volume)

            return True

        return False

    def draw(self, screen):

        # Volume icon with improved styling

        pygame.draw.rect(screen, DARK\_PANEL, self.icon\_rect, border\_radius=8)

        pygame.draw.rect(screen, BORDER\_COLOR, self.icon\_rect, 2, border\_radius=8)

        icon\_font = pygame.font.SysFont('timesnewroman', 24)

        icon\_text = icon\_font.render("♪", True, HIGHLIGHT)

        screen.blit(icon\_text, (self.icon\_rect.centerx - 8, self.icon\_rect.centery - 12))

        if self.visible:

            # Shadow

            shadow\_rect = pygame.Rect(self.panel\_rect.x + 4, self.panel\_rect.y + 4,

                                     self.panel\_rect.width, self.panel\_rect.height)

            pygame.draw.rect(screen, SHADOW\_COLOR, shadow\_rect, border\_radius=10)

            # Panel

            pygame.draw.rect(screen, PANEL\_BG, self.panel\_rect, border\_radius=10)

            pygame.draw.rect(screen, BORDER\_COLOR, self.panel\_rect, 2, border\_radius=10)

            # Volume text

            volume\_text = self.font.render("Volume", True, HIGHLIGHT)

            screen.blit(volume\_text, (self.panel\_rect.x + 10, self.panel\_rect.y + 15))

            # Slider track

            pygame.draw.rect(screen, DARK\_PANEL, self.slider\_rect, border\_radius=5)

            # Filled portion

            filled\_width = self.knob\_x - self.slider\_rect.left

            if filled\_width > 0:

                filled\_rect = pygame.Rect(self.slider\_rect.left, self.slider\_rect.top,

                                        filled\_width, self.slider\_rect.height)

                pygame.draw.rect(screen, HIGHLIGHT, filled\_rect, border\_radius=5)

            # Slider knob

            pygame.draw.circle(screen, TEXT\_COLOR, (int(self.knob\_x), int(self.knob\_y)), self.knob\_radius)

            pygame.draw.circle(screen, HIGHLIGHT, (int(self.knob\_x), int(self.knob\_y)), self.knob\_radius - 2)

            # Volume percentage

            volume\_pct = int((self.knob\_x - self.slider\_rect.left) / self.slider\_rect.width \* 100)

            vol\_text = tiny\_font.render(f"{volume\_pct}%", True, TEXT\_COLOR)

            screen.blit(vol\_text, (self.panel\_rect.centerx - 15, self.panel\_rect.y + 35))

            # Control buttons

            for btn in [self.prev\_btn, self.play\_pause\_btn, self.next\_btn]:

                pygame.draw.rect(screen, BUTTON\_PRIMARY, btn, border\_radius=5)

                pygame.draw.rect(screen, BORDER\_COLOR, btn, 1, border\_radius=5)

            # Icons

            if self.previous\_icon:

                screen.blit(self.previous\_icon, (self.prev\_btn.x + 10, self.prev\_btn.y + 5))

            if music\_player.global\_music.is\_playing:

                if self.pause\_icon:

                    screen.blit(self.pause\_icon, (self.play\_pause\_btn.x + 10, self.play\_pause\_btn.y + 5))

            else:

                if self.play\_icon:

                    screen.blit(self.play\_icon, (self.play\_pause\_btn.x + 10, self.play\_pause\_btn.y + 5))

            if self.next\_icon:

                screen.blit(self.next\_icon, (self.next\_btn.x + 10, self.next\_btn.y + 5))

class Button:

    def \_\_init\_\_(self, x, y, width, height, text, color=BUTTON\_PRIMARY,

                 hover\_color=BUTTON\_HOVER, text\_color=TEXT\_COLOR):

        self.rect = pygame.Rect(x, y, width, height)

        self.text = text

        self.color = color

        self.hover\_color = hover\_color

        self.text\_color = text\_color

        self.hovered = False

    def draw(self, screen):

        # Shadow effect

        shadow = pygame.Rect(self.rect.x + 4, self.rect.y + 4,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Button background

        color = self.hover\_color if self.hovered else self.color

        pygame.draw.rect(screen, color, self.rect, border\_radius=12)

        # Border

        border\_color = HIGHLIGHT if self.hovered else BORDER\_COLOR

        pygame.draw.rect(screen, border\_color, self.rect, 2, border\_radius=12)

        # Button text

        text\_surface = button\_font.render(self.text, True, self.text\_color)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.hovered = self.rect.collidepoint(mouse\_pos)

    def is\_clicked(self, mouse\_pos, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            return self.rect.collidepoint(mouse\_pos)

        return False

class TodoItem:

    def \_\_init\_\_(self, text, completed=False):

        self.text = text

        self.completed = completed

class TodoList:

    def \_\_init\_\_(self, x, y, width, height, user\_id, db=None):

        self.rect = pygame.Rect(x, y, width, height)

        self.user\_id = user\_id

        self.db = db  # Reuse database connection

        self.items = []

        self.input\_text = ""

        self.input\_active = False

        self.hovered\_index = -1

        # Load todos from database

        self.load\_todos()

    def load\_todos(self):

        """Load todos from database"""

        try:

            if self.db:

                todos = self.db.get\_todos(self.user\_id)

                self.items = [TodoItem(todo['task'], todo['completed']) for todo in todos]

                print(f"Loaded {len(self.items)} todos")

        except Exception as e:

            print(f"Error loading todos: {e}")

    def save\_todos(self):

        """Save todos to database asynchronously"""

        def \_save\_in\_background():

            try:

                if self.db:

                    self.db.save\_todos(self.user\_id, [(item.text, item.completed) for item in self.items])

                    print("✓ Todos saved")

            except Exception as e:

                print(f"Error saving todos: {e}")

        # Run save in background thread so UI doesn't freeze

        thread = threading.Thread(target=\_save\_in\_background, daemon=True)

        thread.start()

    def handle\_event(self, event):

        mouse\_pos = pygame.mouse.get\_pos()

        if event.type == pygame.MOUSEMOTION:

            # Check clear button hover

            if hasattr(self, 'clear\_button\_rect') and self.clear\_button\_rect:

                self.clear\_hovered = self.clear\_button\_rect.collidepoint(mouse\_pos)

            # Check item hover

            if self.rect.collidepoint(mouse\_pos):

                item\_height = 35

                header\_height = 70

                list\_y = self.rect.y + header\_height + 25

                for i in range(len(self.items)):

                    item\_y = list\_y + i \* item\_height

                    item\_rect = pygame.Rect(self.rect.x + 10, item\_y, self.rect.width - 20, item\_height - 5)

                    if item\_rect.collidepoint(mouse\_pos):

                        self.hovered\_index = i

                        return

                self.hovered\_index = -1

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            # Check clear button click

            if hasattr(self, 'clear\_button\_rect') and self.clear\_button\_rect and self.clear\_button\_rect.collidepoint(mouse\_pos):

                # Remove all completed items

                self.items = [item for item in self.items if not item.completed]

                self.save\_todos()

                return

            if not self.rect.collidepoint(mouse\_pos):

                if self.input\_active:

                    self.input\_active = False

                return

            header\_height = 70

            input\_y = self.rect.y + 55

            input\_rect = pygame.Rect(self.rect.x + 90, input\_y, self.rect.width - 180, 32)

            if input\_rect.collidepoint(mouse\_pos):

                self.input\_active = True

                return

            # Check item clicks

            item\_height = 35

            list\_y = self.rect.y + header\_height + 25

            for i, item in enumerate(self.items):

                item\_y = list\_y + i \* item\_height

                checkbox\_rect = pygame.Rect(self.rect.x + 20, item\_y + 10, 20, 20)

                if checkbox\_rect.collidepoint(mouse\_pos):

                    item.completed = not item.completed

                    self.save\_todos()

                    return

        if event.type == pygame.KEYDOWN and self.input\_active:

            if event.key == pygame.K\_RETURN and self.input\_text.strip():

                self.items.append(TodoItem(self.input\_text.strip()))

                self.save\_todos()

                self.input\_text = ""

                self.input\_active = False

            elif event.key == pygame.K\_BACKSPACE:

                self.input\_text = self.input\_text[:-1]

            elif event.key == pygame.K\_ESCAPE:

                self.input\_active = False

                self.input\_text = ""

            elif len(self.input\_text) < 50:

                self.input\_text += event.unicode

    def draw(self, screen):

        # Shadow

        shadow\_rect = pygame.Rect(self.rect.x + 5, self.rect.y + 5,

                                 self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow\_rect, border\_radius=15)

        # Main panel

        s = pygame.Surface((self.rect.width, self.rect.height), pygame.SRCALPHA)

        s.fill(TODO\_BG)

        screen.blit(s, (self.rect.x, self.rect.y))

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=15)

        # Header background

        header\_height = 70

        # Draw header as rectangle (not using border\_radius to avoid clipping)

        header\_surface = pygame.Surface((self.rect.width - 4, header\_height), pygame.SRCALPHA)

        header\_surface.fill(PANEL\_BG)

        screen.blit(header\_surface, (self.rect.x + 2, self.rect.y + 2))

        # Manually draw rounded top corners

        pygame.draw.rect(screen, PANEL\_BG,

                        (self.rect.x + 2, self.rect.y + 2, self.rect.width - 4, 20),

                        border\_top\_left\_radius=13, border\_top\_right\_radius=13)

        # Title (with more padding from top)

        completed = sum(1 for item in self.items if item.completed)

        title = todo\_header\_font.render("TASKS", True, HIGHLIGHT)

        screen.blit(title, (self.rect.x + 25, self.rect.y + 20))

        # Counter (aligned with title)

        counter\_text = f"{completed}/{len(self.items)}"

        counter = small\_font.render(counter\_text, True, ACCENT\_SILVER)

        screen.blit(counter, (self.rect.right - 240, self.rect.y + 23))

        # Clear Completed button (if there are completed tasks)

        if completed > 0:

            clear\_btn\_width = 150

            clear\_btn\_x = self.rect.right - clear\_btn\_width - 15

            clear\_btn\_y = self.rect.y + 15

            self.clear\_button\_rect = pygame.Rect(clear\_btn\_x, clear\_btn\_y, clear\_btn\_width, 35)

            # Button styling

            btn\_color = BUTTON\_HOVER if hasattr(self, 'clear\_hovered') and self.clear\_hovered else BUTTON\_PRIMARY

            pygame.draw.rect(screen, btn\_color, self.clear\_button\_rect, border\_radius=8)

            pygame.draw.rect(screen, BORDER\_COLOR, self.clear\_button\_rect, 1, border\_radius=8)

            clear\_text = tiny\_font.render("Clear Completed", True, TEXT\_COLOR)

            text\_rect = clear\_text.get\_rect(center=self.clear\_button\_rect.center)

            screen.blit(clear\_text, text\_rect)

        else:

            self.clear\_button\_rect = None

        # Input box (moved down a bit)

        input\_y = self.rect.y + 55

        input\_rect = pygame.Rect(self.rect.x + 90, input\_y, self.rect.width - 180, 32)

        input\_color = HIGHLIGHT if self.input\_active else INPUT\_BG

        pygame.draw.rect(screen, input\_color, input\_rect, border\_radius=8)

        pygame.draw.rect(screen, BORDER\_COLOR, input\_rect, 1, border\_radius=8)

        # Plus icon (aligned with input)

        plus\_text = text\_font.render("+", True, HIGHLIGHT)

        screen.blit(plus\_text, (self.rect.x + 60, input\_y + 2))

        # Input text or placeholder

        if self.input\_active:

            input\_surf = small\_font.render(self.input\_text + "|", True, TEXT\_COLOR)

        else:

            input\_surf = small\_font.render(self.input\_text or "Add new task...", True, ACCENT\_SILVER)

        screen.blit(input\_surf, (input\_rect.x + 10, input\_rect.y + 7))

        # Todo items (adjusted for new header)

        header\_height = 70

        item\_height = 35

        list\_y = self.rect.y + header\_height + 25

        visible\_items = min(len(self.items), (self.rect.height - header\_height - 30) // item\_height)

        for i in range(visible\_items):

            item = self.items[i]

            item\_y = list\_y + i \* item\_height

            # Item background

            item\_rect = pygame.Rect(self.rect.x + 10, item\_y, self.rect.width - 20, item\_height - 5)

            if i == self.hovered\_index:

                pygame.draw.rect(screen, TODO\_HOVER, item\_rect, border\_radius=8)

            elif item.completed:

                pygame.draw.rect(screen, TODO\_COMPLETED, item\_rect, border\_radius=8)

            else:

                pygame.draw.rect(screen, TODO\_ITEM\_BG, item\_rect, border\_radius=8)

            # Checkbox

            checkbox\_rect = pygame.Rect(self.rect.x + 20, item\_y + 10, 20, 20)

            pygame.draw.rect(screen, INPUT\_BG, checkbox\_rect, border\_radius=4)

            pygame.draw.rect(screen, BORDER\_COLOR, checkbox\_rect, 2, border\_radius=4)

            if item.completed:

                # Checkmark

                check\_font = pygame.font.SysFont('timesnewroman', 18, bold=True)

                check\_text = check\_font.render("✓", True, SUCCESS\_GREEN)

                screen.blit(check\_text, (checkbox\_rect.x + 3, checkbox\_rect.y))

            # Text

            text\_color = ACCENT\_SILVER if item.completed else TEXT\_COLOR

            text\_surf = todo\_font.render(item.text, True, text\_color)

            if item.completed:

                # Strikethrough

                line\_y = item\_y + item\_height // 2

                pygame.draw.line(screen, ACCENT\_SILVER,

                               (self.rect.x + 50, line\_y),

                               (self.rect.x + 50 + text\_surf.get\_width(), line\_y), 1)

            screen.blit(text\_surf, (self.rect.x + 50, item\_y + 8))

class TimerApp:

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

        self.user\_id = user\_id

        self.username = username

        self.running = False

        self.paused = False

        self.remaining\_seconds = 1500  # 25 minutes

        self.original\_seconds = 1500

        self.start\_time = 0

        # Database connection

        self.db = get\_database\_connection()

        if not self.db:

            print("⚠️ Database connection failed - running in limited mode")

        # Website blocking

        self.blocking\_active = False

        self.blocked\_sites = []

        # Video background

        self.video\_bg = VideoBackground('snowfall.mp4')

        # Volume controls

        self.volume\_controls = VolumeControls()

        # UI elements - improved styling

        self.back\_button = Button(30, 30, 120, 50, "← BACK", DARK\_PANEL, BUTTON\_HOVER)

        # Timer display area

        self.timer\_rect = pygame.Rect(SCREEN\_WIDTH//2 - 300, SCREEN\_HEIGHT//2 - 140, 600, 280)

        # Start/Stop button - bigger and more prominent

        self.start\_stop\_button = Button(

            SCREEN\_WIDTH//2 - 120,

            SCREEN\_HEIGHT//2 + 50,  # Moved up

            240,

            60,

            "START",

            BUTTON\_SUCCESS,

            BUTTON\_HOVER

        )

        # Todo list (positioned below start button to avoid overlap)

        todo\_width = 700

        todo\_height = 280

        todo\_y = SCREEN\_HEIGHT//2 + 130  # Below start button with gap

        self.todo\_list = TodoList(SCREEN\_WIDTH//2 - todo\_width//2, todo\_y,

                                  todo\_width, todo\_height, user\_id, self.db)

        # Timer input

        self.timer\_input = ""

        self.timer\_input\_active = False

        # Load blocked sites

        self.load\_blocked\_sites()

    def load\_blocked\_sites(self):

        """Load blocked sites from database and write to blocked\_sites.txt"""

        try:

            if self.db:

                self.blocked\_sites = self.db.get\_blocked\_sites(self.user\_id)

                print(f"📋 Loaded {len(self.blocked\_sites)} blocked sites for user {self.username}")

                # Write to blocked\_sites.txt for website\_blocker.py to use

                with open("blocked\_sites.txt", "w") as f:

                    for site in self.blocked\_sites:

                        f.write(site + "\n")

                print(f"✅ Written {len(self.blocked\_sites)} sites to blocked\_sites.txt")

            else:

                self.blocked\_sites = []

                print("⚠️ No database connection - using empty blocked sites list")

        except Exception as e:

            print(f"❌ Error loading blocked sites: {e}")

            self.blocked\_sites = []

    def handle\_event(self, event):

        mouse\_pos = pygame.mouse.get\_pos()

        # Handle volume controls first

        if self.volume\_controls.handle\_event(event):

            return True

        # Handle back button

        self.back\_button.check\_hover(mouse\_pos)

        if self.back\_button.is\_clicked(mouse\_pos, event):

            self.go\_back()

            return True

        # Handle start/stop button

        self.start\_stop\_button.check\_hover(mouse\_pos)

        if self.start\_stop\_button.is\_clicked(mouse\_pos, event):

            if not self.running:

                self.start\_timer()

            else:

                self.stop\_timer()

            return True

        # Handle timer click for edit/pause/resume

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            if self.timer\_rect.collidepoint(mouse\_pos):

                if not self.running:

                    self.timer\_input\_active = True

                    self.timer\_input = f"{self.remaining\_seconds//60:02d}:{self.remaining\_seconds%60:02d}"

                else:

                    if not self.paused:

                        # Pausing - unblock websites

                        self.paused = True

                        self.start\_stop\_button.text = "RESUME"

                        self.start\_stop\_button.color = BUTTON\_SUCCESS

                        self.deactivate\_blocking()  # Unblock when paused

                    else:

                        # Resuming - re-block websites

                        self.paused = False

                        self.start\_stop\_button.text = "STOP"

                        self.start\_stop\_button.color = BUTTON\_DANGER

                        self.start\_time = time.time() - (self.original\_seconds - self.remaining\_seconds)

                        self.activate\_blocking()  # Re-block when resumed

            else:

                if self.timer\_input\_active:

                    self.finish\_editing()

        # Handle timer input in edit mode

        if event.type == pygame.KEYDOWN and self.timer\_input\_active:

            if event.key == pygame.K\_RETURN:

                self.finish\_editing()

            elif event.key == pygame.K\_BACKSPACE:

                self.timer\_input = self.timer\_input[:-1]

            elif event.key in [pygame.K\_0, pygame.K\_1, pygame.K\_2, pygame.K\_3, pygame.K\_4,

                             pygame.K\_5, pygame.K\_6, pygame.K\_7, pygame.K\_8, pygame.K\_9, pygame.K\_COLON]:

                if len(self.timer\_input) < 8:

                    self.timer\_input += event.unicode

        # Handle escape key

        if event.type == pygame.KEYDOWN and event.key == pygame.K\_ESCAPE:

            if self.timer\_input\_active:

                self.timer\_input\_active = False

            else:

                self.go\_back()

            return True

        # Handle todo list

        self.todo\_list.handle\_event(event)

        return False

    def finish\_editing(self):

        try:

            time\_str = self.timer\_input.strip()

            if ':' in time\_str:

                minutes, seconds = map(int, time\_str.split(':'))

                if seconds >= 60:

                    return

            else:

                minutes = int(time\_str)

                seconds = 0

            if minutes < 0 or seconds < 0:

                return

            total\_seconds = minutes \* 60 + seconds

            if total\_seconds == 0:

                return

            self.remaining\_seconds = total\_seconds

            self.original\_seconds = total\_seconds

            self.timer\_input\_active = False

        except ValueError:

            pass

    def activate\_blocking(self):

        """Activate website blocking when timer starts"""

        if not self.blocking\_active and self.blocked\_sites:

            print("🔒 Activating website blocking...")

            success = website\_blocker.block\_websites()

            if success:

                self.blocking\_active = True

                print("✅ Websites blocked successfully")

            else:

                print("⚠️ Website blocking failed - check admin permissions")

    def deactivate\_blocking(self):

        """Deactivate website blocking when timer stops"""

        if self.blocking\_active:

            print("🔓 Deactivating website blocking...")

            success = website\_blocker.unblock\_websites()

            if success:

                self.blocking\_active = False

                print("✅ Websites unblocked successfully")

            else:

                print("⚠️ Website unblocking failed")

    def start\_timer(self):

        if self.timer\_input\_active:

            self.finish\_editing()

        self.running = True

        self.paused = False

        self.start\_stop\_button.text = "STOP"

        self.start\_stop\_button.color = BUTTON\_DANGER

        self.original\_seconds = self.remaining\_seconds

        self.start\_time = time.time()

        self.activate\_blocking()

    def stop\_timer(self):

        self.running = False

        self.paused = False

        self.start\_stop\_button.text = "START"

        self.start\_stop\_button.color = BUTTON\_SUCCESS

        self.deactivate\_blocking()

    def update\_timer(self):

        if self.running and not self.paused:

            current\_time = time.time()

            elapsed = current\_time - self.start\_time

            new\_remaining = max(0, self.original\_seconds - int(elapsed))

            if new\_remaining != self.remaining\_seconds:

                self.remaining\_seconds = new\_remaining

            if self.remaining\_seconds <= 0:

                self.timer\_finished()

    def timer\_finished(self):

        self.running = False

        self.paused = False

        self.start\_stop\_button.text = "START"

        self.start\_stop\_button.color = BUTTON\_SUCCESS

        self.deactivate\_blocking()

        # Record study time

        study\_minutes = self.original\_seconds // 60

        if study\_minutes > 0 and self.db:

            try:

                self.db.record\_study\_session(self.user\_id, study\_minutes)

                print(f"✅ Recorded {study\_minutes} minutes for {self.username}")

            except Exception as e:

                print(f"Error recording session: {e}")

        # Reset timer to original time

        self.remaining\_seconds = 1500

        self.original\_seconds = 1500

    def go\_back(self):

        self.deactivate\_blocking()

        if hasattr(self.video\_bg, 'cap'):

            self.video\_bg.release()

        pygame.quit()

        # Just exit - main.py is still running and will show again

        # Restart main.py (session persists in current\_user.txt)

        import subprocess

        subprocess.run([sys.executable, "main.py"])

        sys.exit()

    def draw\_progress\_ring(self, screen, center\_x, center\_y, radius):

        """Draw a circular progress indicator around the timer"""

        if self.running and self.original\_seconds > 0:

            progress = 1 - (self.remaining\_seconds / self.original\_seconds)

            # Background circle

            pygame.draw.circle(screen, DARK\_PANEL, (center\_x, center\_y), radius, 10)

            # Progress arc

            if progress > 0:

                start\_angle = -math.pi / 2

                end\_angle = start\_angle + (2 \* math.pi \* progress)

                # Draw multiple overlapping arcs for thickness

                for i in range(10):

                    current\_radius = radius - 5 + i

                    rect = pygame.Rect(center\_x - current\_radius, center\_y - current\_radius,

                                     current\_radius \* 2, current\_radius \* 2)

                    pygame.draw.arc(screen, HIGHLIGHT, rect, start\_angle, end\_angle, 2)

    def draw(self, screen):

        # Video background

        bg\_frame = self.video\_bg.get\_frame()

        if bg\_frame:

            screen.blit(bg\_frame, (0, 0))

        else:

            screen.fill(BACKGROUND)

        # Draw back button

        self.back\_button.draw(screen)

        # Timer label (no box behind timer)

        label\_text = "FOCUS TIMER"

        label\_surf = small\_font.render(label\_text, True, HIGHLIGHT)

        label\_rect = label\_surf.get\_rect(center=(SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2 - 180))

        screen.blit(label\_surf, label\_rect)

        # Progress ring (centered on timer)

        self.draw\_progress\_ring(screen, SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2 - 80, 180)

        # Timer display

        if self.timer\_input\_active:

            timer\_surf = timer\_font.render(self.timer\_input, True, HIGHLIGHT)

            timer\_rect = timer\_surf.get\_rect(center=(SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2 - 80))

            screen.blit(timer\_surf, timer\_rect)

            # Blinking cursor

            if int(time.time() \* 2) % 2 == 0:

                cursor\_x = timer\_rect.right + 5

                cursor\_y = timer\_rect.centery - 30

                pygame.draw.line(screen, TEXT\_COLOR, (cursor\_x, cursor\_y), (cursor\_x, cursor\_y + 60), 3)

        else:

            minutes = self.remaining\_seconds // 60

            seconds = self.remaining\_seconds % 60

            timer\_text = f"{minutes:02d}:{seconds:02d}"

            timer\_color = TEXT\_COLOR

            if self.running:

                timer\_color = PAUSE\_YELLOW if self.paused else TEXT\_COLOR

            timer\_surf = timer\_font.render(timer\_text, True, timer\_color)

            timer\_rect = timer\_surf.get\_rect(center=(SCREEN\_WIDTH//2, SCREEN\_HEIGHT//2 - 80))

            screen.blit(timer\_surf, timer\_rect)

        # Percentage text

        if self.running and self.original\_seconds > 0:

            progress\_pct = int((1 - self.remaining\_seconds / self.original\_seconds) \* 100)

            pct\_text = tiny\_font.render(f"{progress\_pct}%", True, ACCENT\_SILVER)

            screen.blit(pct\_text, (SCREEN\_WIDTH//2 - 15, SCREEN\_HEIGHT//2 + 20))

        # Start/stop button (moved up)

        self.start\_stop\_button.draw(screen)

        # Status panel (moved up)

        status\_panel\_width = 700

        status\_panel\_height = 50

        status\_panel\_x = SCREEN\_WIDTH//2 - status\_panel\_width//2

        status\_panel\_y = SCREEN\_HEIGHT//2 + 120

        status\_rect = pygame.Rect(status\_panel\_x, status\_panel\_y, status\_panel\_width, status\_panel\_height)

        pygame.draw.rect(screen, DARK\_PANEL, status\_rect, border\_radius=10)

        pygame.draw.rect(screen, BORDER\_COLOR, status\_rect, 1, border\_radius=10)

        # Status text with improved styling

        status\_text = ""

        status\_color = TEXT\_COLOR

        if self.running:

            if self.paused:

                status\_text = "PAUSED"

                status\_color = PAUSE\_YELLOW

                if self.blocking\_active:

                    status\_text += " • Websites Blocked"

            else:

                status\_text = "FOCUSING"

                status\_color = SUCCESS\_GREEN

                if self.blocking\_active:

                    status\_text += " • Websites Blocked"

        else:

            if self.timer\_input\_active:

                status\_text = "Editing timer • Press ENTER to save"

                status\_color = HIGHLIGHT

            else:

                status\_text = "Click timer to edit • Click START to begin"

                status\_color = ACCENT\_SILVER

        status\_surf = text\_font.render(status\_text, True, status\_color)

        status\_text\_rect = status\_surf.get\_rect(center=status\_rect.center)

        screen.blit(status\_surf, status\_text\_rect)

        # Draw todo list

        self.todo\_list.draw(screen)

        # Draw volume controls

        self.volume\_controls.draw(screen)

    def run(self):

        clock = pygame.time.Clock()

        running = True

        while running:

            for event in pygame.event.get():

                if event.type == pygame.QUIT:

                    running = False

                else:

                    self.handle\_event(event)

            self.update\_timer()

            self.draw(screen)

            pygame.display.flip()

            clock.tick(60)

        self.go\_back()

def main():

    import sys

    # Check if user\_id and username were passed as command line arguments

    if len(sys.argv) >= 3:

        # Called from main.py with user info - use that

        user\_id = int(sys.argv[1])

        username = sys.argv[2]

        print(f"Using provided user: {username} (ID: {user\_id})")

    else:

        # No arguments provided - show login (shouldn't happen normally)

        import login\_screen

        print("No user provided, opening login screen...")

        user\_id, username = login\_screen.login\_screen()

        if user\_id is None:

            print("Login cancelled")

            pygame.quit()

            sys.exit()

            return

        print(f"User logged in: {username} (ID: {user\_id})")

    # Run the timer with the user

    app = TimerApp(user\_id, username)

    app.run()

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

    main()

## user\_management.py

A screenshot of a login form

AI-generated content may be incorrect.

import pygame

import sys

import json

import os

import hashlib

import subprocess

# Initialize pygame

pygame.init()

# Screen dimensions

SCREEN\_INFO = pygame.display.Info()

SCREEN\_WIDTH, SCREEN\_HEIGHT = SCREEN\_INFO.current\_w, SCREEN\_INFO.current\_h

screen = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))

pygame.display.set\_caption("Focus Timer - User Management")

# Colors (matching theme)

BACKGROUND = (25, 25, 50)

PANEL\_BG = (40, 40, 80)

DARK\_PANEL = (30, 30, 60)

TEXT\_COLOR = (255, 255, 255)

HIGHLIGHT = (100, 200, 255)

ACCENT\_SILVER = (192, 192, 192)

BUTTON\_PRIMARY = (70, 130, 180)

BUTTON\_HOVER = (100, 160, 210)

BUTTON\_SUCCESS = (80, 180, 120)

BUTTON\_DANGER = (255, 100, 100)

INPUT\_BG = (40, 40, 60)

BORDER\_COLOR = (100, 100, 150)

SHADOW\_COLOR = (15, 15, 30)

ERROR\_RED = (255, 100, 100)

SUCCESS\_GREEN = (100, 200, 150)

# Fonts

title\_font = pygame.font.SysFont('timesnewroman', 72, bold=True)

header\_font = pygame.font.SysFont('timesnewroman', 36, bold=True)

text\_font = pygame.font.SysFont('timesnewroman', 24)

small\_font = pygame.font.SysFont('timesnewroman', 20)

button\_font = pygame.font.SysFont('timesnewroman', 22, bold=True)

def hash\_password(password):

    '''Hash password using SHA-256'''

    return hashlib.sha256(password.encode()).hexdigest()

def get\_database\_connection():

    """Create a database connection"""

    try:

        config\_file = "mysql\_config.json"

        if os.path.exists(config\_file):

            with open(config\_file, 'r') as f:

                mysql\_config = json.load(f)

        else:

            mysql\_config = {

                'host': 'localhost',

                'user': 'root',

                'password': '',

                'database': 'focus\_app',

                'port': 3306

            }

        from database import Database

        return Database(mysql\_config)

    except Exception as e:

        print(f"Database connection error: {e}")

        return None

def create\_user(username, password):

    """Create a new user in database with hashed password"""

    try:

        if not username or not password:

            return False, "Username and password cannot be empty"

        username = username.strip()

        password = password.strip()

        if len(username) < 3:

            return False, "Username must be at least 3 characters"

        if len(password) < 4:

            return False, "Password must be at least 4 characters"

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            # Check if username already exists

            cursor.execute("SELECT id FROM users WHERE username = %s", (username,))

            if cursor.fetchone():

                cursor.close()

                db.connection.close()

                return False, "Username already exists"

            # Hash password and insert

            try:

                password\_hash = hash\_password(password)

                cursor.execute("INSERT INTO users (username, password\_hash) VALUES (%s, %s)",

                              (username, password\_hash))

                db.connection.commit()

                cursor.close()

                db.connection.close()

                return True, "User created successfully!"

            except Exception as e:

                print(f"Failed to create user: {e}")

                cursor.close()

                db.connection.close()

                return False, f"Failed to create user: {str(e)[:200]}"

    except Exception as e:

        error\_str = str(e)

        print(f"Error creating user: {error\_str}")

        import traceback

        traceback.print\_exc()

        # Provide specific error messages

        if "duplicate" in error\_str.lower() or "1062" in error\_str:

            return False, f"Username '{username}' already exists!"

        elif "connection" in error\_str.lower() or "2003" in error\_str:

            return False, "Cannot connect to MySQL. Is it running?"

        elif "access denied" in error\_str.lower() or "1045" in error\_str:

            return False, "MySQL credentials wrong. Check mysql\_config.json"

        elif "database" in error\_str.lower() and "doesn't exist" in error\_str.lower():

            return False, "Database doesn't exist. Run main.py first to create it."

        else:

            return False, f"Error: {error\_str[:80]}"

    return False, "Could not connect to database"

def get\_all\_users():

    """Get all users from database"""

    try:

        db = get\_database\_connection()

        if db and db.connection:

            cursor = db.connection.cursor(dictionary=True)

            cursor.execute("SELECT id, username FROM users ORDER BY username")

            users = cursor.fetchall()

            cursor.close()

            db.connection.close()

            return users

    except Exception as e:

        print(f"Error loading users: {e}")

    return []

class Button:

    def \_\_init\_\_(self, x, y, width, height, text, color=BUTTON\_PRIMARY, hover\_color=BUTTON\_HOVER):

        self.rect = pygame.Rect(x, y, width, height)

        self.text = text

        self.color = color

        self.hover\_color = hover\_color

        self.hovered = False

    def draw(self, screen):

        # Shadow

        shadow = pygame.Rect(self.rect.x + 4, self.rect.y + 4,

                           self.rect.width, self.rect.height)

        pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=12)

        # Button

        color = self.hover\_color if self.hovered else self.color

        pygame.draw.rect(screen, color, self.rect, border\_radius=12)

        pygame.draw.rect(screen, BORDER\_COLOR, self.rect, 2, border\_radius=12)

        # Text

        text\_surface = button\_font.render(self.text, True, TEXT\_COLOR)

        text\_rect = text\_surface.get\_rect(center=self.rect.center)

        screen.blit(text\_surface, text\_rect)

    def check\_hover(self, mouse\_pos):

        self.hovered = self.rect.collidepoint(mouse\_pos)

    def is\_clicked(self, mouse\_pos, event):

        if event.type == pygame.MOUSEBUTTONDOWN and event.button == 1:

            return self.rect.collidepoint(mouse\_pos)

        return False

def draw\_input\_box(screen, x, y, width, height, text, active, label="", is\_password=False):

    """Draw an input box with label"""

    # Label

    if label:

        label\_surf = text\_font.render(label, True, TEXT\_COLOR)

        screen.blit(label\_surf, (x, y - 35))

    # Shadow

    shadow = pygame.Rect(x + 3, y + 3, width, height)

    pygame.draw.rect(screen, SHADOW\_COLOR, shadow, border\_radius=10)

    # Box

    color = HIGHLIGHT if active else INPUT\_BG

    pygame.draw.rect(screen, color, (x, y, width, height), border\_radius=10)

    pygame.draw.rect(screen, BORDER\_COLOR, (x, y, width, height), 2, border\_radius=10)

    # Text

    if is\_password:

        display\_text = "•" \* len(text)

    else:

        display\_text = text

    if active:

        display\_text += "|"

    text\_surface = text\_font.render(display\_text, True, TEXT\_COLOR)

    screen.blit(text\_surface, (x + 15, y + 12))

def user\_management\_screen(first\_time=False):

    """User management screen for creating/managing users"""

    clock = pygame.time.Clock()

    # State

    username\_input = ""

    password\_input = ""

    username\_active = False

    password\_active = False

    message = ""

    message\_color = TEXT\_COLOR

    # Cache existing users

    existing\_users = get\_all\_users()

    if first\_time:

        message = "Welcome! Please create your first user account."

        message\_color = HIGHLIGHT

    # Input boxes

    input\_width = 400

    input\_height = 50

    input\_x = SCREEN\_WIDTH // 2 - input\_width // 2

    username\_y = 300

    password\_y = 400

    username\_rect = pygame.Rect(input\_x, username\_y, input\_width, input\_height)

    password\_rect = pygame.Rect(input\_x, password\_y, input\_width, input\_height)

    # Buttons

    create\_button = Button(SCREEN\_WIDTH // 2 - 110, 500, 220, 55,

                          "Create User", BUTTON\_SUCCESS, BUTTON\_HOVER)

    # Back button (only if not first time)

    back\_button = None

    if not first\_time:

        back\_button = Button(50, 50, 140, 50, "← Back")

    # Continue button (only if users exist)

    continue\_button = None

    if existing\_users:

        continue\_button = Button(SCREEN\_WIDTH - 200, 50, 150, 50, "Continue →")

    running = True

    while running:

        mouse\_pos = pygame.mouse.get\_pos()

        for event in pygame.event.get():

            if event.type == pygame.QUIT:

                return None

            elif event.type == pygame.MOUSEBUTTONDOWN:

                # Check input clicks

                if username\_rect.collidepoint(mouse\_pos):

                    username\_active = True

                    password\_active = False

                elif password\_rect.collidepoint(mouse\_pos):

                    username\_active = False

                    password\_active = True

                else:

                    username\_active = False

                    password\_active = False

                # Check create button

                if create\_button.is\_clicked(mouse\_pos, event):

                    success, msg = create\_user(username\_input, password\_input)

                    message = msg

                    message\_color = SUCCESS\_GREEN if success else ERROR\_RED

                    if success:

                        username\_input = ""

                        password\_input = ""

                        existing\_users = get\_all\_users()

                        if not continue\_button and existing\_users:

                            continue\_button = Button(SCREEN\_WIDTH - 200, 50, 150, 50, "Continue →")

                # Check back button

                if back\_button and back\_button.is\_clicked(mouse\_pos, event):

                    pygame.quit()

                    return "back"

                # Check continue button

                if continue\_button and continue\_button.is\_clicked(mouse\_pos, event):

                    pygame.quit()

                    subprocess.run([sys.executable, "main.py"])

                    return "continue"

            elif event.type == pygame.KEYDOWN:

                if username\_active:

                    if event.key == pygame.K\_RETURN:

                        username\_active = False

                        password\_active = True

                    elif event.key == pygame.K\_BACKSPACE:

                        username\_input = username\_input[:-1]

                    elif event.key == pygame.K\_TAB:

                        username\_active = False

                        password\_active = True

                    elif len(username\_input) < 30:

                        username\_input += event.unicode

                elif password\_active:

                    if event.key == pygame.K\_RETURN:

                        success, msg = create\_user(username\_input, password\_input)

                        message = msg

                        message\_color = SUCCESS\_GREEN if success else ERROR\_RED

                        if success:

                            username\_input = ""

                            password\_input = ""

                            password\_active = False

                            existing\_users = get\_all\_users()

                            if not continue\_button and existing\_users:

                                continue\_button = Button(SCREEN\_WIDTH - 200, 50, 150, 50, "Continue →")

                    elif event.key == pygame.K\_BACKSPACE:

                        password\_input = password\_input[:-1]

                    elif event.key == pygame.K\_TAB:

                        password\_active = False

                        username\_active = True

                    elif len(password\_input) < 30:

                        password\_input += event.unicode

        # Update hover states

        create\_button.check\_hover(mouse\_pos)

        if back\_button:

            back\_button.check\_hover(mouse\_pos)

        if continue\_button:

            continue\_button.check\_hover(mouse\_pos)

        # Draw everything

        screen.fill(BACKGROUND)

        # Draw buttons in header

        if back\_button:

            back\_button.draw(screen)

        if continue\_button:

            continue\_button.draw(screen)

        # Title

        title\_text = title\_font.render("USER MANAGEMENT", True, HIGHLIGHT)

        title\_rect = title\_text.get\_rect(center=(SCREEN\_WIDTH // 2, 120))

        screen.blit(title\_text, title\_rect)

        # Instruction

        if first\_time:

            instruction = header\_font.render("Create Your First Account", True, ACCENT\_SILVER)

        else:

            instruction = header\_font.render("Add New User", True, ACCENT\_SILVER)

        instruction\_rect = instruction.get\_rect(center=(SCREEN\_WIDTH // 2, 200))

        screen.blit(instruction, instruction\_rect)

        # Input boxes

        draw\_input\_box(screen, input\_x, username\_y, input\_width, input\_height,

                      username\_input, username\_active, "Username:")

        draw\_input\_box(screen, input\_x, password\_y, input\_width, input\_height,

                      password\_input, password\_active, "Password:", is\_password=True)

        # Create button

        create\_button.draw(screen)

        # Message

        if message:

            message\_surf = text\_font.render(message, True, message\_color)

            message\_rect = message\_surf.get\_rect(center=(SCREEN\_WIDTH // 2, 580))

            screen.blit(message\_surf, message\_rect)

        # Existing users list

        if existing\_users:

            list\_y = 640

            list\_title = small\_font.render("Existing Users:", True, ACCENT\_SILVER)

            screen.blit(list\_title, (SCREEN\_WIDTH // 2 - 200, list\_y))

            for i, user in enumerate(existing\_users[:5]):

                user\_text = small\_font.render(f"• {user['username']}", True, TEXT\_COLOR)

                screen.blit(user\_text, (SCREEN\_WIDTH // 2 - 180, list\_y + 30 + i \* 25))

        # Help text

        help\_text = small\_font.render("Press TAB to switch fields • ENTER to create user", True, ACCENT\_SILVER)

        help\_rect = help\_text.get\_rect(center=(SCREEN\_WIDTH // 2, SCREEN\_HEIGHT - 50))

        screen.blit(help\_text, help\_rect)

        pygame.display.flip()

        clock.tick(60)

    return None

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

    result = user\_management\_screen(first\_time=True)

    print(f"Result: {result}")

    pygame.quit()

    sys.exit()

## volume\_controls.py

A screenshot of a music player

AI-generated content may be incorrect.

import pygame

import music\_player

class VolumeControls:

    def \_\_init\_\_(self, screen\_width):

        self.screen\_width = screen\_width

        self.visible = False

        self.dragging = False

        # Volume icon position (top right)

        self.icon\_rect = pygame.Rect(screen\_width - 50, 20, 30, 30)

        # Volume panel (appears when icon clicked)

        self.panel\_rect = pygame.Rect(screen\_width - 200, 60, 180, 120)

        # Slider

        self.slider\_rect = pygame.Rect(screen\_width - 190, 100, 160, 15)

        self.knob\_radius = 10

        self.knob\_x = self.slider\_rect.x + (music\_player.global\_music.get\_volume() \* self.slider\_rect.width)

        self.knob\_y = self.slider\_rect.y + self.slider\_rect.height // 2

        # Buttons

        self.play\_btn = pygame.Rect(screen\_width - 190, 130, 50, 30)

        self.pause\_btn = pygame.Rect(screen\_width - 130, 130, 50, 30)

        self.stop\_btn = pygame.Rect(screen\_width - 70, 130, 50, 30)

        self.font = pygame.font.SysFont('Arial', 12)

    def handle\_event(self, event):

        mouse\_pos = pygame.mouse.get\_pos()

        if event.type == pygame.MOUSEBUTTONDOWN:

            if self.icon\_rect.collidepoint(mouse\_pos):

                self.visible = not self.visible

                return True

            if self.visible:

                # Check slider knob

                knob\_rect = pygame.Rect(self.knob\_x - self.knob\_radius, self.knob\_y - self.knob\_radius,

                                      self.knob\_radius \* 2, self.knob\_radius \* 2)

                if knob\_rect.collidepoint(mouse\_pos):

                    self.dragging = True

                    return True

                # Check buttons

                if self.play\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.play()

                    return True

                elif self.pause\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.pause()

                    return True

                elif self.stop\_btn.collidepoint(mouse\_pos):

                    music\_player.global\_music.stop()

                    return True

                # Click outside panel to close

                if not self.panel\_rect.collidepoint(mouse\_pos):

                    self.visible = False

                    return True

        elif event.type == pygame.MOUSEBUTTONUP:

            self.dragging = False

        elif event.type == pygame.MOUSEMOTION and self.dragging:

            self.knob\_x = max(self.slider\_rect.left, min(event.pos[0], self.slider\_rect.right))

            volume = (self.knob\_x - self.slider\_rect.left) / self.slider\_rect.width

            music\_player.global\_music.set\_volume(volume)

            return True

        return False

    def draw(self, screen):

        # Draw volume icon (always visible)

        pygame.draw.rect(screen, (100, 100, 100), self.icon\_rect, border\_radius=5)

        pygame.draw.rect(screen, (200, 200, 200), self.icon\_rect, 2, border\_radius=5)

        # Volume icon symbol

        icon\_font = pygame.font.SysFont('Arial', 20)

        icon\_text = icon\_font.render("♪", True, (255, 255, 255))

        screen.blit(icon\_text, (self.icon\_rect.centerx - 6, self.icon\_rect.centery - 10))

        # Draw volume panel if visible

        if self.visible:

            # Panel background

            pygame.draw.rect(screen, (50, 50, 50), self.panel\_rect, border\_radius=8)

            pygame.draw.rect(screen, (150, 150, 150), self.panel\_rect, 2, border\_radius=8)

            # Volume text

            volume\_text = self.font.render("Volume", True, (255, 255, 255))

            screen.blit(volume\_text, (self.panel\_rect.x + 10, self.panel\_rect.y + 15))

            # Volume percentage

            volume\_percent = int(music\_player.global\_music.get\_volume() \* 100)

            percent\_text = self.font.render(f"{volume\_percent}%", True, (200, 200, 200))

            screen.blit(percent\_text, (self.panel\_rect.right - 40, self.panel\_rect.y + 15))

            # Slider track

            pygame.draw.rect(screen, (100, 100, 100), self.slider\_rect)

            pygame.draw.rect(screen, (150, 150, 150), self.slider\_rect, 1)

            # Slider knob

            pygame.draw.circle(screen, (100, 200, 200), (self.knob\_x, self.knob\_y), self.knob\_radius)

            pygame.draw.circle(screen, (50, 50, 50), (self.knob\_x, self.knob\_y), self.knob\_radius, 2)

            # Control buttons

            pygame.draw.rect(screen, (0, 150, 0), self.play\_btn, border\_radius=3)

            pygame.draw.rect(screen, (200, 150, 0), self.pause\_btn, border\_radius=3)

            pygame.draw.rect(screen, (150, 0, 0), self.stop\_btn, border\_radius=3)

            # Button text

            play\_text = self.font.render("Play", True, (255, 255, 255))

            pause\_text = self.font.render("Pause", True, (255, 255, 255))

            stop\_text = self.font.render("Stop", True, (255, 255, 255))

            screen.blit(play\_text, (self.play\_btn.x + 10, self.play\_btn.y + 8))

            screen.blit(pause\_text, (self.pause\_btn.x + 5, self.pause\_btn.y + 8))

            screen.blit(stop\_text, (self.stop\_btn.x + 10, self.stop\_btn.y + 8))

## website\_blocker.py

import os

import subprocess

import tempfile

import ctypes

import sys

import time

class WebsiteBlocker:

    def \_\_init\_\_(self):

        self.hosts\_path = r"C:\Windows\System32\drivers\etc\hosts"

        self.blocked\_file = "blocked\_sites.txt"

        self.redirect\_ip = "0.0.0.0"

        self.is\_admin = self.check\_admin()

        print(f"🔧 WebsiteBlocker initialized - Admin: {self.is\_admin}")

    def check\_admin(self):

        """Check if running as administrator"""

        try:

            return ctypes.windll.shell32.IsUserAnAdmin()

        except:

            return False

    def request\_admin(self):

        """Request administrator privileges"""

        if not self.is\_admin:

            print("🛑 Administrator privileges required for website blocking")

            print("💡 Please run the program as Administrator")

            return False

        return True

    def write\_to\_hosts\_file(self, content):

        """Write content to hosts file with admin privileges"""

        try:

            print(f"📝 Attempting to write to hosts file: {self.hosts\_path}")

            # Method 1: Direct write with admin rights

            try:

                with open(self.hosts\_path, 'w', encoding='utf-8') as f:

                    f.write(content)

                print("✅ Successfully wrote to hosts file directly")

                return True

            except PermissionError:

                print("❌ Direct write failed - no permission")

            # Method 2: Use subprocess with admin rights

            try:

                # Create temporary file

                with tempfile.NamedTemporaryFile(mode='w', delete=False, suffix='.txt', encoding='utf-8') as temp\_file:

                    temp\_file.write(content)

                    temp\_path = temp\_file.name

                print(f"📄 Created temp file: {temp\_path}")

                # Use subprocess to copy with admin rights

                copy\_cmd = ['cmd', '/c', 'copy', temp\_path, self.hosts\_path, '/Y']

                result = subprocess.run(copy\_cmd, capture\_output=True, text=True, timeout=10)

                # Clean up temp file

                try:

                    os.unlink(temp\_path)

                except:

                    pass

                if result.returncode == 0:

                    print("✅ Successfully copied to hosts file")

                    return True

                else:

                    print(f"❌ Copy failed: {result.stderr}")

                    return False

            except Exception as e:

                print(f"❌ Subprocess copy failed: {e}")

                return False

        except Exception as e:

            print(f"❌ Error writing to hosts file: {e}")

            return False

    def get\_blocked\_sites(self):

        """Get list of sites to block"""

        print(f"📁 Looking for blocked\_sites.txt in: {os.getcwd()}")

        print(f"📁 File exists: {os.path.exists(self.blocked\_file)}")

        if not os.path.exists(self.blocked\_file):

            print("❌ blocked\_sites.txt not found!")

            return []

        try:

            with open(self.blocked\_file, 'r') as f:

                sites = [line.strip() for line in f.readlines() if line.strip()]

            print(f"📋 Read {len(sites)} sites from blocked\_sites.txt: {sites}")

            return sites

        except Exception as e:

            print(f"❌ Error reading blocked sites: {e}")

            return []

    def get\_comprehensive\_blocks(self, site):

        """Get comprehensive blocking entries for a site"""

        blocks = []

        # Base domains

        base\_domains = [

            site,

            f"www.{site}",

            f"m.{site}",

            f"mobile.{site}",

        ]

        for domain in base\_domains:

            blocks.append(f"{self.redirect\_ip} {domain}")

        # YouTube-specific blocks (most common issue)

        if "youtube" in site.lower():

            youtube\_domains = [

                "youtube.com", "www.youtube.com", "m.youtube.com",

                "youtu.be", "www.youtu.be",

                "googlevideo.com", "www.googlevideo.com",  # Video hosting

                "ytimg.com", "www.ytimg.com",  # Thumbnails

                "google.com", "www.google.com",  # Sometimes needed

                "gstatic.com",  # Google static content

                "ggpht.com",  # Google photos

            ]

            for domain in youtube\_domains:

                blocks.append(f"{self.redirect\_ip} {domain}")

        # Other common sites

        elif "netflix" in site.lower():

            netflix\_domains = ["netflix.com", "www.netflix.com", "nflxso.net", "nflxext.com"]

            for domain in netflix\_domains:

                blocks.append(f"{self.redirect\_ip} {domain}")

        elif "facebook" in site.lower():

            facebook\_domains = ["facebook.com", "www.facebook.com", "fb.com", "fbcdn.net"]

            for domain in facebook\_domains:

                blocks.append(f"{self.redirect\_ip} {domain}")

        return blocks

    def disable\_dns\_over\_https(self):

        """Disable DNS over HTTPS in Windows"""

        try:

            print("🛑 Disabling DNS over HTTPS...")

            # Disable DoH via registry

            reg\_commands = [

                'reg add "HKEY\_LOCAL\_MACHINE\\SYSTEM\\CurrentControlSet\\Services\\Dnscache\\Parameters" /v EnableAutoDoh /t REG\_DWORD /d 0 /f',

                'reg add "HKEY\_LOCAL\_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Windows NT\\DNSClient" /v DoHPolicy /t REG\_DWORD /d 0 /f',

            ]

            for cmd in reg\_commands:

                result = subprocess.run(cmd, shell=True, capture\_output=True, text=True)

                if result.returncode == 0:

                    print(f"✅ Registry command executed: {cmd}")

                else:

                    print(f"⚠️ Registry command failed (may not exist): {cmd}")

        except Exception as e:

            print(f"❌ Error disabling DoH: {e}")

    def flush\_dns\_comprehensive(self):

        """Comprehensive DNS flushing"""

        try:

            print("🔄 Performing comprehensive DNS flush...")

            # Stop DNS client service

            subprocess.run(["net", "stop", "dnscache"], capture\_output=True)

            time.sleep(2)

            # Start DNS client service

            subprocess.run(["net", "start", "dnscache"], capture\_output=True)

            time.sleep(1)

            # Flush DNS

            subprocess.run(["ipconfig", "/flushdns"], capture\_output=True)

            # Additional DNS cache clearing

            subprocess.run(["nbtstat", "-R"], capture\_output=True)

            subprocess.run(["nbtstat", "-RR"], capture\_output=True)

            print("✅ Comprehensive DNS flush completed")

        except Exception as e:

            print(f"❌ DNS flush error: {e}")

    def disable\_browser\_doh(self):

        """Disable DNS over HTTPS in major browsers"""

        print("🛑 Disabling DNS over HTTPS in browsers...")

        try:

            # Chrome/Edge - disable DoH via registry

            chrome\_doh\_disable = [

                'reg add "HKEY\_LOCAL\_MACHINE\\SOFTWARE\\Policies\\Google\\Chrome" /v DnsOverHttpsMode /t REG\_SZ /d "off" /f',

                'reg add "HKEY\_LOCAL\_MACHINE\\SOFTWARE\\Policies\\Microsoft\\Edge" /v DnsOverHttpsMode /t REG\_SZ /d "off" /f',

            ]

            for cmd in chrome\_doh\_disable:

                subprocess.run(cmd, shell=True, capture\_output=True)

            print("✅ Browser DoH disabled")

        except Exception as e:

            print(f"⚠️ Could not disable browser DoH: {e}")

    def test\_blocking(self):

        """Test if blocking is actually working"""

        print("🧪 Testing if blocking is working...")

        # Test DNS resolution

        test\_sites = ["youtube.com", "www.youtube.com"]

        for site in test\_sites:

            try:

                result = subprocess.run(

                    ["nslookup", site],

                    capture\_output=True,

                    text=True,

                    timeout=10

                )

                if "0.0.0.0" in result.stdout:

                    print(f"✅ {site} is correctly blocked (resolves to 0.0.0.0)")

                else:

                    print(f"❌ {site} is NOT blocked - resolves to actual IP")

                    print(f"   Output: {result.stdout}")

            except Exception as e:

                print(f"❌ Error testing {site}: {e}")

    def block\_websites(self):

        """Block all websites in the list"""

        print("🚀 WEBSITE BLOCKER: Starting comprehensive blocking...")

        if not self.request\_admin():

            return False

        sites = self.get\_blocked\_sites()

        if not sites:

            return True

        try:

            # Disable DNS over HTTPS first

            self.disable\_dns\_over\_https()

            # Read current hosts file

            current\_content = ""

            if os.path.exists(self.hosts\_path):

                with open(self.hosts\_path, 'r', encoding='utf-8') as f:

                    current\_content = f.read()

            # Create comprehensive blocking entries

            blocking\_entries = []

            blocking\_entries.append("\n# Focus Timer Blocked Sites - DO NOT EDIT")

            for site in sites:

                # Add comprehensive blocking for each site

                site\_blocks = self.get\_comprehensive\_blocks(site)

                blocking\_entries.extend(site\_blocks)

            blocking\_entries.append("# End Focus Timer Blocked Sites\n")

            # Remove existing block section

            lines = current\_content.splitlines()

            new\_lines = []

            in\_block\_section = False

            for line in lines:

                if "# Focus Timer Blocked Sites - DO NOT EDIT" in line:

                    in\_block\_section = True

                    continue

                if "# End Focus Timer Blocked Sites" in line:

                    in\_block\_section = False

                    continue

                if not in\_block\_section and line.strip():

                    new\_lines.append(line)

            # Build new content

            new\_content = "\n".join(new\_lines).strip()

            if new\_content:

                new\_content += "\n"

            new\_content += "\n".join(blocking\_entries)

            # Write to hosts file

            success = self.write\_to\_hosts\_file(new\_content)

            if success:

                self.flush\_dns\_comprehensive()

                self.disable\_browser\_doh()  # NEW: Disable DoH in browsers

                self.test\_blocking()  # Test if it's working

                print("✅ Comprehensive website blocking activated!")

                return True

            else:

                print("❌ Failed to block websites")

                return False

        except Exception as e:

            print(f"❌ Blocking error: {e}")

            return False

    def unblock\_websites(self):

        """Remove all blocking entries"""

        print("🚀 Starting website unblocking...")

        if not self.request\_admin():

            return False

        try:

            if not os.path.exists(self.hosts\_path):

                print("ℹ️ Hosts file doesn't exist, nothing to unblock")

                return True

            # Read current hosts

            with open(self.hosts\_path, 'r', encoding='utf-8') as f:

                lines = f.readlines()

            # Remove Focus Timer block section

            new\_lines = []

            in\_block\_section = False

            removed\_section = False

            for line in lines:

                if "# Focus Timer Blocked Sites - DO NOT EDIT" in line:

                    in\_block\_section = True

                    removed\_section = True

                    continue

                if "# End Focus Timer Blocked Sites" in line:

                    in\_block\_section = False

                    continue

                if not in\_block\_section:

                    new\_lines.append(line)

            new\_content = "".join(new\_lines).strip()

            print("📄 Writing cleaned hosts file...")

            success = self.write\_to\_hosts\_file(new\_content)

            if success:

                self.flush\_dns\_comprehensive()

                if removed\_section:

                    print("✅ Websites unblocked successfully!")

                else:

                    print("ℹ️ No blocking section found to remove")

                return True

            else:

                print("❌ Failed to unblock websites")

                return False

        except Exception as e:

            print(f"❌ Unblocking error: {e}")

            return False

# Create global instance

blocker = WebsiteBlocker()

# Module-level functions for easy import

def block\_websites():

    return blocker.block\_websites()

def unblock\_websites():

    return blocker.unblock\_websites()

def test\_blocking():

    return blocker.test\_blocking()

# Limitations, References, and Conclusion

**LIMITATIONS AND SHORTCOMINGS**

While the Focus Timer application successfully implements core productivity features, the following limitations and potential enhancements have been identified:

**Current Limitations:**

1. **Platform Dependency**
   * Currently Windows-only due to hosts file modification for website blocking
   * macOS and Linux versions would require different blocking implementations
   * Admin privileges required, which may not be available in all environments
2. **Website Blocking Scope**
   * Only blocks websites at domain level (not specific pages)
   * Cannot block applications (only websites)
   * Requires browser restart for some sites to be blocked
   * No HTTPS inspection (some sites may bypass using HTTPS)
3. **Database Limitations**
   * Requires local MySQL installation
   * No cloud sync between devices
   * Manual backup required for data preservation
   * Single database instance (no multi-device support)
4. **User Interface**
   * Fixed window size (not responsive to different screen resolutions)
   * No dark/light theme toggle
   * Limited customization options for colors and fonts
   * No accessibility features (screen reader support, high contrast mode)
5. **Timer Functionality**
   * No break timer (only focus timer implemented)
   * No automatic Pomodoro cycles (25-5-25-5 pattern)
   * No timer presets for different durations
   * Cannot run multiple timers simultaneously
6. **Task Management**
   * Basic todo list without categories or priorities
   * No due dates or reminders for tasks
   * No task search or filtering
   * Limited to 500 characters per task
7. **Analytics and Reporting**
   * Limited data visualization (basic charts only)
   * No export functionality (PDF, CSV, Excel)
   * Cannot compare productivity across different time periods
   * No goal setting or progress tracking features
8. **Network and Sync**
   * No cloud synchronization
   * Cannot share tasks or compete with remote users
   * Leaderboard limited to local database users
   * No mobile app companion
9. **Session Management**
   * Must login on every app restart
   * No "Remember Me" option
   * No password recovery mechanism
   * Cannot have multiple users logged in simultaneously

**Potential Enhancements (Future Scope):**

1. **Cross-Platform Support**
   * Implement platform-specific blocking mechanisms for macOS/Linux
   * Web-based version accessible from any device
   * Mobile applications (Android/iOS)
2. **Advanced Website Blocking**
   * Application-level blocking (block Discord, games, etc.)
   * Schedule-based blocking (block only during certain hours)
   * Whitelist mode (block everything except allowed sites)
   * Category-based blocking (social media, entertainment, news)
3. **Cloud Features**
   * Cloud database with real-time sync
   * Multi-device support with unified account
   * Online leaderboard with global users
   * Backup and restore from cloud
4. **Enhanced Timer**
   * Full Pomodoro technique with automatic break cycles
   * Multiple timer presets (25-5, 50-10, custom)
   * Timer history with tags and notes
   * Smart scheduling based on productivity patterns
5. **Advanced Task Management**
   * Task categories, priorities, and tags
   * Due dates with notifications
   * Recurring tasks
   * Task dependencies and subtasks
   * Integration with calendar apps
6. **Rich Analytics**
   * Advanced visualizations (heatmaps, trend analysis)
   * Productivity insights and recommendations
   * Goal setting with progress tracking
   * Export reports in multiple formats
   * AI-powered productivity suggestions
7. **Social Features**
   * Study groups with shared leaderboards
   * Challenge friends to focus sessions
   * Share achievements on social media
   * Global tournaments and competitions
8. **Integration Capabilities**
   * Google Calendar integration
   * Notion/Todoist task sync
   * Spotify/YouTube Music integration
   * Slack/Discord notifications
   * Browser extensions
9. **AI and Machine Learning**
   * Smart focus time recommendations
   * Automatic task categorization
   * Predictive productivity analytics
   * Personalized blocking suggestions
10. **Accessibility and Customization**
    * Full theme customization
    * Multiple language support
    * Screen reader compatibility
    * Keyboard shortcuts
    * Customizable UI layout

**REFERENCES AND BIBLIOGRAPHY**

**Documentation and APIs**

1. **Pygame Documentation**
   * Official Documentation: https://www.pygame.org/docs/
   * Pygame Community: https://www.pygame.org/wiki/
   * Used for: GUI development, graphics rendering, event handling
2. **MySQL Documentation**
   * MySQL 8.0 Reference Manual: https://dev.mysql.com/doc/refman/8.0/en/
   * MySQL Connector/Python: https://dev.mysql.com/doc/connector-python/en/
   * Used for: Database design, SQL queries, connection management
3. **Python Documentation**
   * Python 3 Official Documentation: https://docs.python.org/3/
   * Threading Module: https://docs.python.org/3/library/threading.html
   * Subprocess Module: https://docs.python.org/3/library/subprocess.html
   * Used for: Core programming, multithreading, process management
4. **OpenCV Documentation**
   * OpenCV-Python Tutorials: https://docs.opencv.org/4.x/d6/d00/tutorial\_py\_root.html
   * Used for: Video playback for background animations

**Design and Concepts**

1. **Pomodoro Technique**
   * Cirillo, Francesco. "The Pomodoro Technique"
   * https://francescocirillo.com/pages/pomodoro-technique
   * Used for: Timer methodology, productivity principles
2. **UI/UX Design Resources**
   * Material Design Guidelines: https://material.io/design
   * UI Design Principles: https://www.nngroup.com/articles/
   * Used for: Interface design, color schemes, user experience

**Technical References**

1. **Database Design**
   * "Database System Concepts" by Silberschatz, Korth, and Sudarshan
   * Database normalization: https://www.guru99.com/database-normalization.html
   * Used for: Table structure, relationships, foreign keys
2. **Security Practices**
   * Password Hashing: https://www.sha256.org/
   * OWASP Security Guidelines: https://owasp.org/
   * Used for: SHA-256 password hashing, secure authentication
3. **Windows API**
   * ctypes Documentation: https://docs.python.org/3/library/ctypes.html
   * Windows Administrator Privileges: Microsoft Developer Documentation
   * Used for: Admin privilege handling, hosts file modification
4. **Software Architecture**
   * "Clean Code" by Robert C. Martin
   * Python Best Practices: https://docs.python-guide.org/
   * Used for: Code organization, modularity, best practices

**Tools and Libraries**

1. **hashlib (Python)**
   * https://docs.python.org/3/library/hashlib.html
   * Used for: Password hashing (SHA-256)
2. **json (Python)**
   * https://docs.python.org/3/library/json.html
   * Used for: Configuration file management
3. **datetime (Python)**
   * https://docs.python.org/3/library/datetime.html
   * Used for: Timestamp handling, session tracking

**Community Resources**

1. **Stack Overflow**
   * https://stackoverflow.com/
   * Used for: Problem solving, debugging, best practices
2. **GitHub**
   * Various open-source projects for reference
   * Used for: Code patterns, implementation ideas
3. **Reddit Communities**
   * r/learnprogramming
   * r/Python
   * Used for: Learning resources, community support

**Inspiration and Similar Projects**

1. **Forest App**
   * https://www.forestapp.cc/
   * Inspiration for: Gamification, focus tracking
2. **Focus@Will**
   * https://www.focusatwill.com/
   * Inspiration for: Music integration, productivity features
3. **RescueTime**
   * https://www.rescuetime.com/
   * Inspiration for: Analytics dashboard, time tracking
4. **Cold Turkey**
   * https://getcoldturkey.com/
   * Inspiration for: Website blocking implementation

**CONCLUSION**

The project titled **"FOCUS TIMER - Productivity Application with Multi-User Support and Analytics"** done by **Amartya Srinivasan, K Shrivathsan, and R Vidhyuthram** for the academic year **2024-2025**, has been completed, tested, and executed successfully.

**Project Summary**

The Focus Timer application successfully implements a comprehensive productivity solution that combines the proven Pomodoro technique with modern features including website blocking, task management, analytics tracking, and competitive gamification through leaderboards. The application demonstrates proficiency in:

* **Full-Stack Development**: Integration of Python backend with MySQL database and Pygame frontend
* **Security Implementation**: SHA-256 password hashing and secure authentication mechanisms
* **Database Design**: Normalized schema with proper relationships and constraints
* **User Experience**: Intuitive GUI with smooth animations and responsive controls
* **System Integration**: Windows hosts file modification for website blocking
* **Multi-Threading**: Asynchronous operations for responsive user interface
* **Software Architecture**: Modular design with clear separation of concerns

**Key Achievements**

1. **Functional Completeness**: All planned features have been implemented and tested
2. **Data Persistence**: Reliable MySQL database integration with proper error handling
3. **User Management**: Secure multi-user system with session management
4. **Performance**: Optimized database operations and asynchronous loading for smooth experience
5. **Usability**: Clean interface with comprehensive features accessible to all user types

**Technical Competency Demonstrated**

The project showcases understanding of:

* Object-oriented programming principles
* Database management and SQL
* GUI development with event-driven programming
* Process and thread management
* File I/O operations
* Security best practices
* Software testing and debugging
* Documentation and project management

**Practical Application**

The Focus Timer application serves as a practical tool for students and professionals seeking to improve their productivity through structured time management, distraction blocking, and progress tracking. The competitive leaderboard feature adds a social element that encourages consistent usage and healthy competition.

**Learning Outcomes**

Through the development of this project, the team gained valuable experience in:

* Planning and executing a full-scale software project
* Collaborative development and version control
* User-centered design thinking
* Problem-solving and debugging complex systems
* Integration of multiple technologies and libraries
* Project documentation and presentation

**Future Potential**

While the current implementation successfully meets the project objectives, the identified limitations provide clear pathways for future enhancement. The modular architecture allows for easy extension of features, and the foundation is solid enough to support the addition of cloud synchronization, mobile applications, and advanced analytics.

**Final Remarks**

The Focus Timer project demonstrates that effective productivity tools can be developed using open-source technologies and modern software engineering practices. The application not only fulfills its technical requirements but also provides genuine value to users seeking to improve their focus and time management skills.

The successful completion of this project marks a significant milestone in the team's software development journey and serves as a foundation for future projects of greater complexity and scope.

**Project Status**: ✅ COMPLETED AND OPERATIONAL

**Testing Status**: ✅ THOROUGHLY TESTED

**Deployment**: ✅ READY FOR USE

**Documentation**: ✅ COMPREHENSIVE

*Project completed on: December 2024*

*Academic Year: 2024-2025*

**Team Members:**

* Amartya Srinivasan
* K Shrivathsan
* R Vidhyuthram

**Technology Stack:**

* Programming Language: Python 3.13.7
* GUI Framework: Pygame 2.6.1
* Database: MySQL 8.0
* Version Control: Git
* Development Environment: Visual Studio Code

**Total Lines of Code**: ~5000+

For access of code, refer:  
<https://drive.google.com/drive/folders/1c-SVfKJKYBOh5mFrTX-pQCY70K0wpIeG?usp=sharing>

Qr code:

A qr code on a black background

AI-generated content may be incorrect.