

Task Management System

Intro to Programming
CMPT 120L

MOMAP



Marist College
School of Computer Science and Mathematics

Submitted To:
Dr. Reza Sadeghi

Fall 2024

Team Name

MOMAP

Team Members

1. Marissa Ratschki
2. Marlene Santiago-Cuevas
3. Prashna Khadka
4. Anna Chen
5. Olufunke Gando

marissa.ratschki1@marist.edu (Team Lead)
marlene.santiago-cuevas1@marist.edu (Team Member)
prashna.khadka1@marist.edu (Team Member)
anna.chen1@marist.edu (Team Member)
olu.gando1@marist.edu (Team Member)

Description of Team Members**1. Marissa Ratschki**

My name is Marissa Ratschki. I'm 21 years old, and a sophomore majoring in Cybersecurity at Marist College. Although limited, I have experience in HTML, CSS, and Python through collegiate level courses at Champlain College. My choice to work with team MOMAP was based on the group's eagerness to learn within the computer science field, as well as their dedication to academia and willingness to collaborate. Furthermore, I am enthusiastic about the prospect of working with a team of women in the computer science domain—and am confident that through our shared passion for learning, we will put forth a productive learning experience and resulting project.

2. Marlene Santiago-Cuevas

Greetings, my name is Marlene Santiago-Cuevas, and my major is Computer Science. I began my interest in programming in my junior year of high school and have not lost interest since. I began to take classes surrounding the subject such as Programming 101 and 102 and AP Computer Science A. As a first-year student in college, it has been exciting to engage with a group of young adults who share the same passion. I easily connected with my team members, and that is essential when working together to create an exceptional project. Working together, I'm confident we will use our skills to help each other with our certain areas of weaknesses. Our team head was elected based on skill and experience with the required website and coding.

3. Prashna Khadka

My name is Prashna Khadka. I'm a freshman majoring in Computer Science with a concentration in Software Development. I'm enrolled in the Intro to Programming course as it is required for my major. I have a little bit of experience with Python from my high school computer science course but it was primarily focused on pseudocode. I wanted to work with my current team members because they are proactive and committed to creating a successful project. We elected our team head based on her familiarity with Github and her prior experience in coding.

4. Anna Chen

My name is Anna Chen, and I'm a junior studying Business Administration with a concentration in Marketing and a minor in Cybersecurity. I have some experience in coding from my AP course in high school, but otherwise, I don't know much about Python. That being said, I wanted to work with MOMAP because, despite my inexperience, everyone was still very welcoming and willing to work with me on this project. I am very excited to work with a group of welcoming and intelligent women and can't wait to see what we can do together. We decided that the team lead would be the person who knew the most about

GitHub and had the most prior experience in coding. After discussing with the group, it was collectively decided that Marissa was the most experienced, and should be the team lead.

5. Olufunke Gando

My name is Olufunke Gando, and I am 20 years old. I am a freshman taking Intro to Programming since it is a requirement for my major in Computer Science. I have no prior experience with programming so this will be my very first project. I wanted to work with my current team members because we all gravitated toward one another, and they are all women who are confident in themselves and what they are doing. We selected the head of our team based on familiarity with GitHub and different programs.

Table of Contents

Table of Figures	5-6
Project Descriptions	7
GitHub Repository Address.....	8
User Experience Design.....	9-16
Virtual Environment	16
List of Packages.....	17
User Interface Design & Applied Code.....	19-33
Data Storage	34
Improved User Interface and Report.....	35-58
Visually Reported Stored Data.....	59-66
References.....	67

Table of Figures

Figure 1: Login Page Flowchart.....	8
Figure 2: Main Menu Flowchart.....	9
Figure 3: Add Task Flowchart.....	10
Figure 4: Remove Phase Flowchart.....	11
Figure 5: Search Task Flowchart.....	12
Figure 6: Edit Task Flowchart.....	13
Figure 7: Calendar Page Flowchart.....	14
Figure 8: Creating the Virtual Environment MOMAP.....	15
Figure 9: List of Packages Install in Virtual Environment.....	16
Figure 10: Running IDE on a Virtual Environment.....	17
Figure 11: Welcome & Login Page.....	18
Figure 11.1: Welcome Page Code.....	19
Figure 11.2: Login Page Code	19
Figure 12: Main Menu Page.....	20
Figure 12.1: Main Menu Page Code	20
Figure 13: Add Task Menu.....	21
Figure 13.1: Add Task - Completion.....	21
Figure 13.2: Add Task Code.....	22
Figure 14: Task Selection for Editing.....	23
Figure 14.1: Edit Task Page.....	23
Figure 14.2: Edit Task Code.....	24
Figure 15: Remove Task.....	25
Figure 15.1: Removed Task Successfully.....	25
Figure 15.2: Remove Task Code.....	26
Figure 16: Search Task Option.....	27
Figure 16.1: Search Task Not Found.....	27
Figure 16.2: Search Task Code.....	28
Figure 16.3: Search Task Code(Cont.).....	29
Figure 17: Calendar Page.....	30
Figure 17.1: Calendar Code.....	31
Figure 17.2: Calendar Code(Cont.).....	32
Figure 17.3: Calendar Code(Cont.(2)).....	33
Figure 18: Improved Welcome & Login Page.....	35
Figure 18.1: Improved Welcome & Login Page Code.....	36
Figure 19: Admin Login Page	37
Figure 19.1: Admin Login Page Code.....	37
Figure 20: User Login Page.....	38
Figure 20.1: User Login Page Code.....	38
Figure 21: Caesar Cipher Encryption.....	39
Figure 22: Improved Main Menu Page.....	40
Figure 22.1: Improved Main Menu Page Code.....	40
Figure 23: Improved Add Task Page.....	41
Figure 23.1 Improved Add Task Page Code.....	42

Figure 23.2: Improved Add Task Page Code (Cont.).....	43
Figure 24: Improved Remove Task Page.....	44
Figure 24.1 Improved Remove Task Page Code	45
Figure 25: Improved Edit Task Page.....	46
Figure 25.1: Improved Edit Task Page Code.....	47
Figure 25.2: Improved Edit Task Page Code (Cont.).....	48
Figure 25.3: Improved Edit Task Page Code (Cont.(2)).....	49
Figure 26: Improved Search Task Page.....	50
Figure 26.1: Improved Search Task Page Code.....	51
Figure 26.2: Improved Search Task Page Code (Cont.).....	52
Figure 26.3: Improved Search Task Page Code (Cont. (2)).....	53
Figure 27: Improved Calendar Page.....	54
Figure 27.1: Improved Calendar Page Code.....	55
Figure 27.2: Improved Calendar Page Code (Cont.).....	56
Figure 27.3: Improved Calendar Page Code (Cont. (2)).....	57
Figure 27.4: Improved Calendar Page Code (Cont. (3)).....	58
Figure 28: Logging In.....	59
Figure 29: Edit Task Page.....	60
Figure 30: Edit Task Subpage.....	60
Figure 31: Successfully Edit Task Indicator.....	61
Figure 32: Successfully Updated Task.....	61
Figure 33: Figure Task Page and Indicator.....	62
Figure 34: Search Task Page with Details Menu.....	62
Figure 35: Calendar Page.....	63
Figure 36: User Management Page and New Username Field.....	64
Figure 37: Successfully Updated Task Indicator.....	64
Figure 38: Closing Message.....	65

Project Objective/Project Description

Our Task Management System (TMS) presents our users with a calendar for their chosen week, month, or year. It also organizes personal tasks for different users on a specific day. Users can also view and modify their personal calendar information. Our TMS will save data for different user types in separate comma-separated value (CSV) files.

The system will support the following features:

- a. an admin username and password for log-in.
- b. option to change the admin username and admin password.
- c. a normal user can be added to the Task Management System by creating a new username and password.
 - i. They will not be able to define or remove other users.
- d. Users can be removed from the Task Management System by erasing their login credentials.
 - i. This will also erase the corresponding recorded data.

Each user will be able to:

- a. add a task to the Task Management System.
 - i. Each task will contain a title, time, duration, and description.
- b. remove a task.
- c. edit a task's details.
- d. search through the Task Management System by time, title, or duration, then list and view the results on the screen.

Our Task Management System will be a user-friendly software, such that:

- a. It will show a welcome page and provide a menu of all functions available to the user on each page.
- b. It will display a well-organized calendar for every month or year.
- c. It will display a warning whenever a user attempts to enter task information with a name that already exists in the history.
- d. Our Task Management System will have an exit function and thank the user for using our software.

Our Task Management System will protect user information, such that:

- a. The Task Management System user passwords and recorded information will be ciphered using Caesar cipher methodology.

GitHub Repository Address

https://github.com/mratschki/CMPT120L-114_Task-Management-System_MOMAP.git

User Experience Design

Login Page:

Input: The user enters a username and password.

Output: If the username and password are valid, the user gains access to the main menu. If invalid, an error message ("Invalid username or password") is displayed.

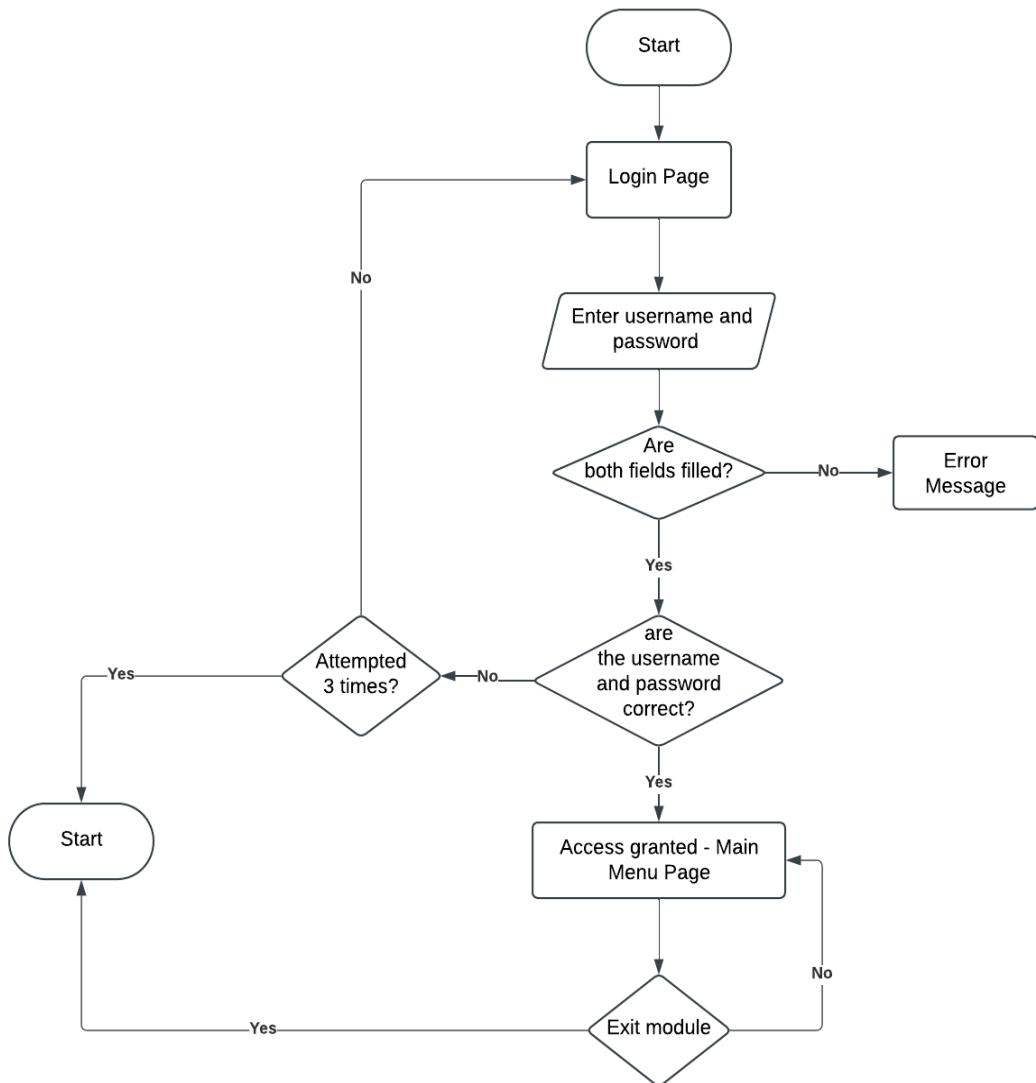


Figure 1: Login Page Flowchart

Main Menu:

Input: The user will select “add,” “remove,” “edit,” “search,” or “calendar.”

Output: This will bring the user to the selected action page screen.

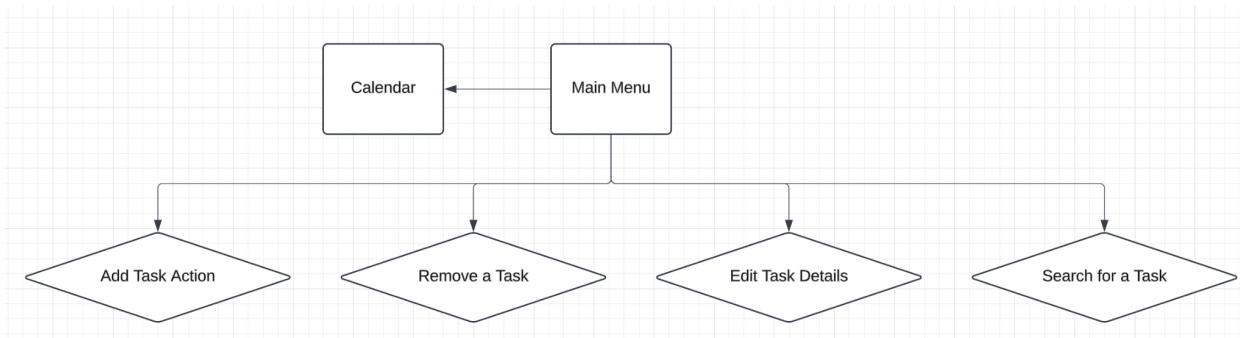


Figure 2: Main Menu Flowchart

Add Task:

Input: The user adds a task.

Output: The task is saved and the user is returned to the main menu screen.

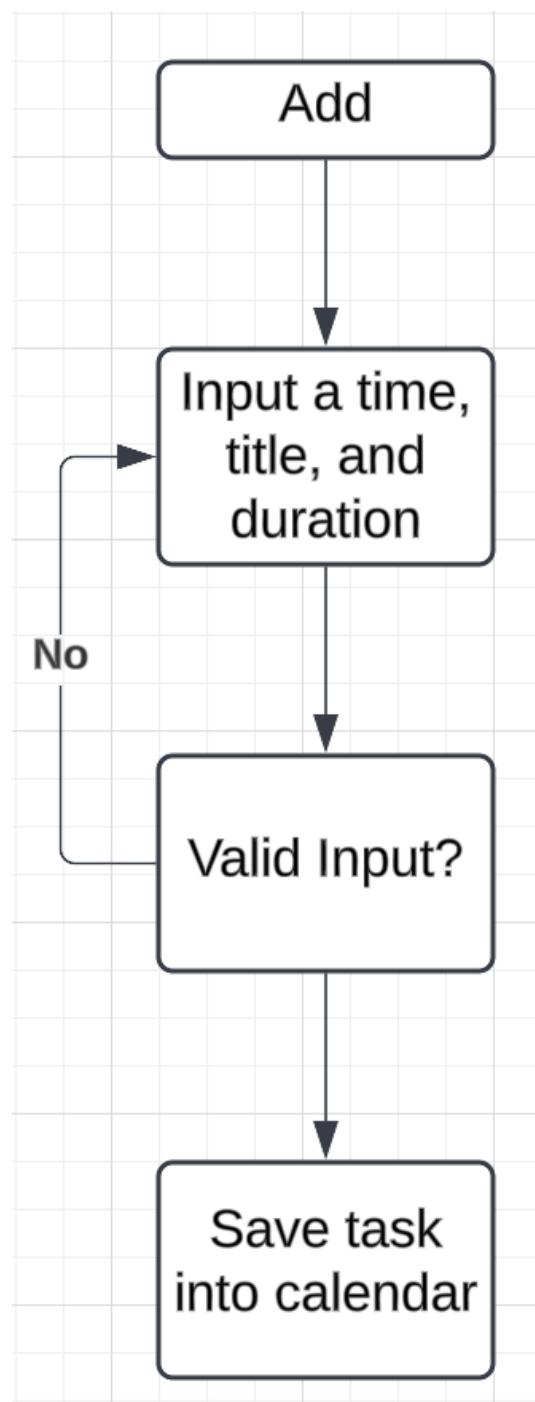


Figure 3: Add Task Flowchart

Remove Phase:

Input: The user selects a task to be deleted.

Output: The user is prompted to confirm if that is the task they want to delete. If the user clicks yes, then the task is deleted. If the user clicks no, they will be returned to the main menu screen.

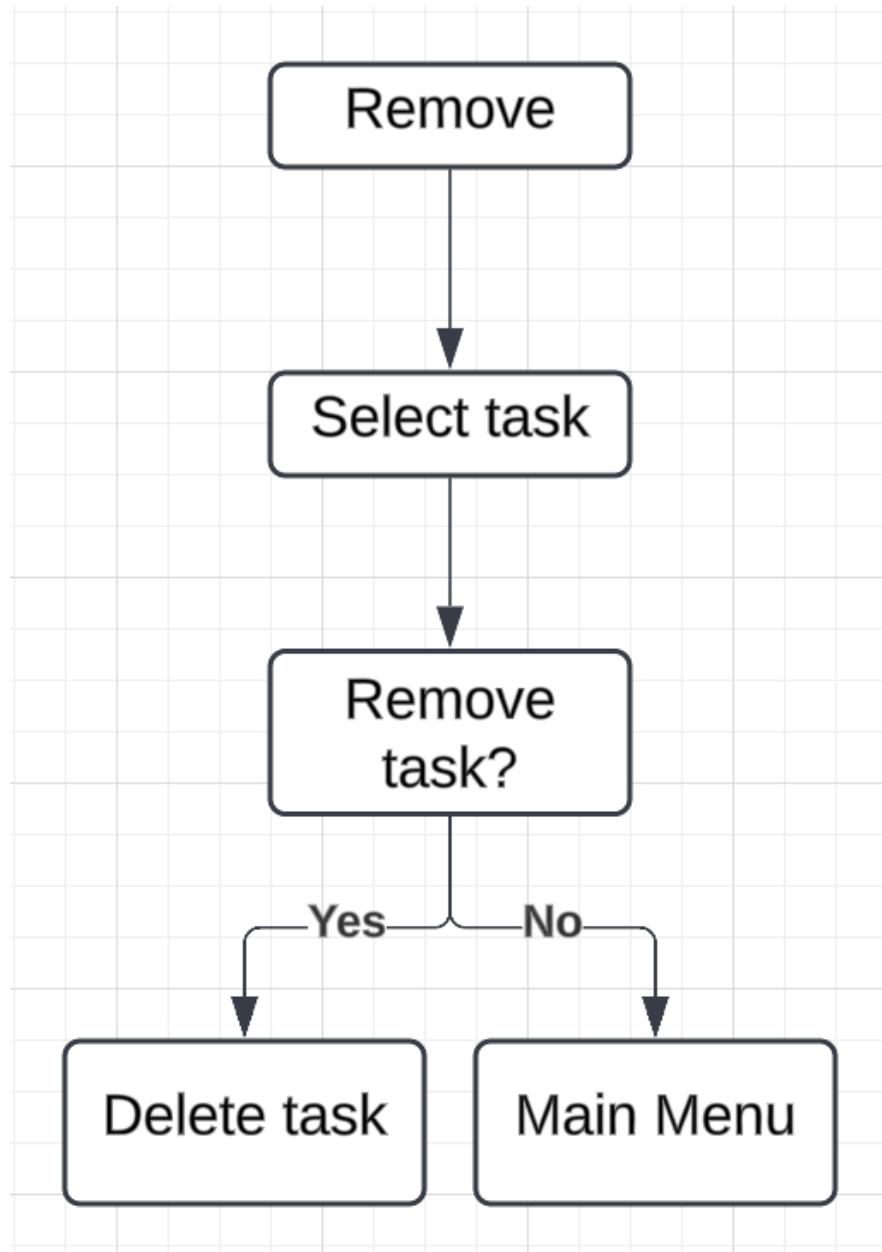


Figure 4: Remove Task Flowchart

Search Task:

Input: The user searches a task by time, title, or duration.

Output: Search tasks from the categories of time, title, or duration and views results.

Results display on the screen, and then proceed to the main menu.

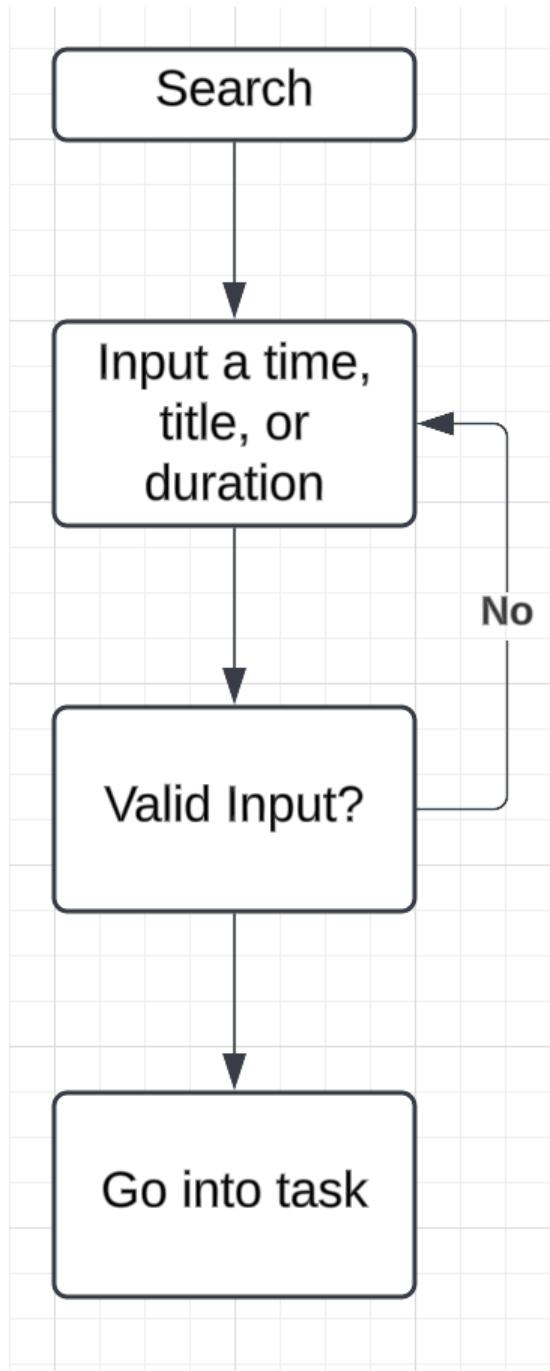


Figure 5: Search Task Flowchart

Edit Task:

Input: The user selects a task to edit.

Output: The user can edit tasks from time, title, or duration. The screen will display if the user wants the changes to be saved. If the user selects yes, the changes will be saved and then they will be brought back to the main menu. If the users select no, they will be brought back to the edit page.

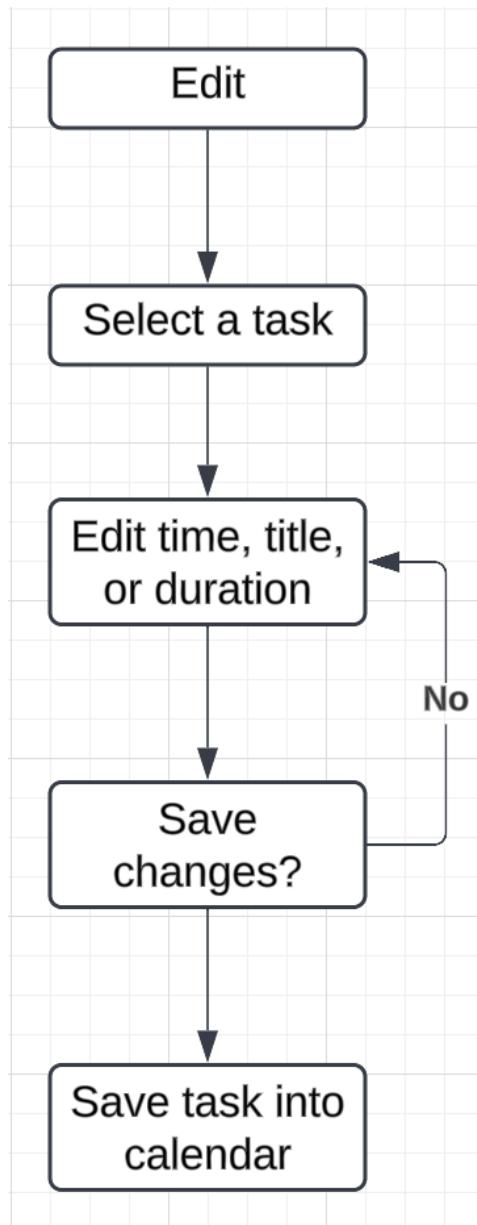


Figure 6: Edit Task Flowchart

Calendar Page:

Input: The user selects to view their calendar for a selected month.

Output: The screen will display all the tasks that the user inputted in the selected month.

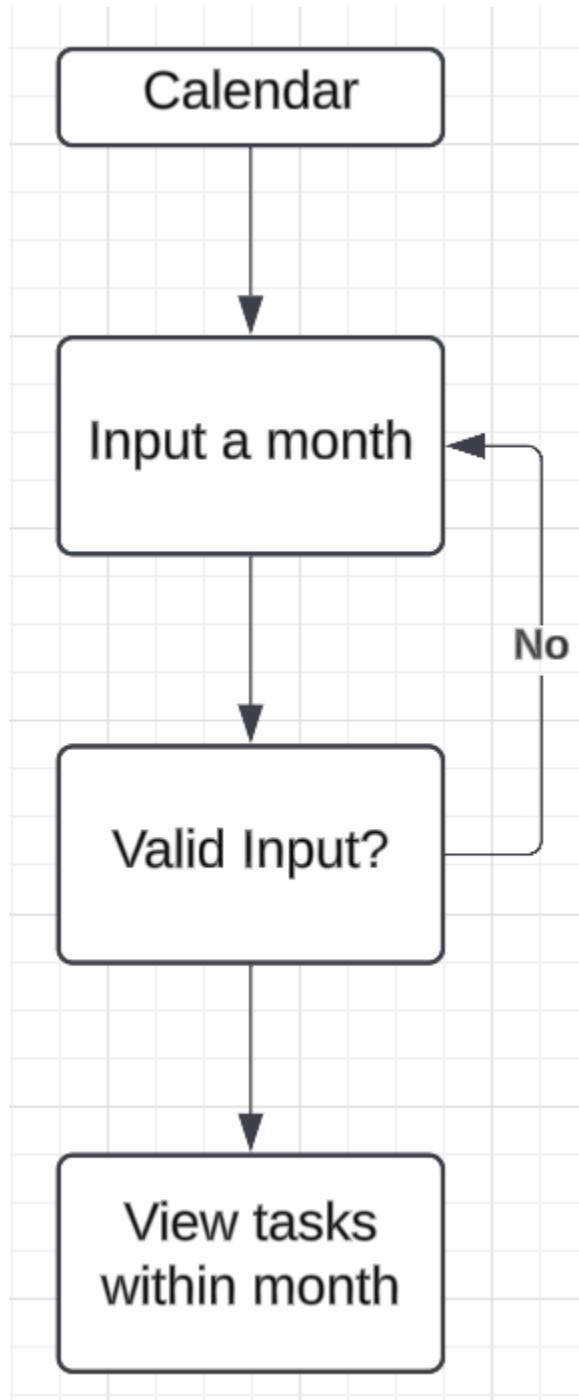


Figure 7: Calendar Page Flowchart

Virtual Environment

Creating the code:

```
Last login: Tue Nov  5 21:42:15 on ttys001
[annachen@148-100-170-111 ~ % sudo pip3 install virtualenv
[Password:
WARNING: The directory '/Users/annachen/Library/Caches/pip' or its parent directory is not owned or is not writable by the current user. The cache has been disabled. Check the permissions and owner of that directory. If executing pip with sudo, you should use sudo's -H flag.
Requirement already satisfied: virtualenv in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (20.27.1)
Requirement already satisfied: distlib<1,>=0.3.7 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from virtualenv) (0.3.9)
Requirement already satisfied: filelock<4,>=3.12.2 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from virtualenv) (3.16.1)
Requirement already satisfied: platformdirs<5,>=3.9.1 in /Library/Frameworks/Python.framework/Versions/3.12/lib/python3.12/site-packages (from virtualenv) (4.3.6)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager, possibly rendering your system unusable. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv. Use the --root-user-action option if you know what you are doing and want to suppress this warning.
annachen@148-100-170-111 ~ % python3 -m venv MOMAP
annachen@148-100-170-111 ~ % source MOMAP/bin/activate
(MOMAP) annachen@148-100-170-111 ~ %
```

Figure 8: Creating the Virtual Environment MOMAP

List of Packages to be used:

Numpy
Matplotlibs
Tkmacos
Tkinter
OS

```
(MOMAP) annachen@148-100-170-111 ~ % python3 -m pip list
Package           Version
-----
colour            0.1.5
contourpy         1.3.0
cycler            0.12.1
fonttools          4.54.1
kiwisolver        1.4.7
matplotlib        3.9.2
numpy              2.1.3
packaging          24.1
pillow             11.0.0
pip                24.3.1
pyparsing          3.2.0
python-dateutil   2.9.0.post0
six                1.16.0
tk                 0.1.0
tkmacosx           1.0.5
(MOMAP) annachen@148-100-170-111 ~ %
```

Figure 9: List of Packages Install in Virtual Environment

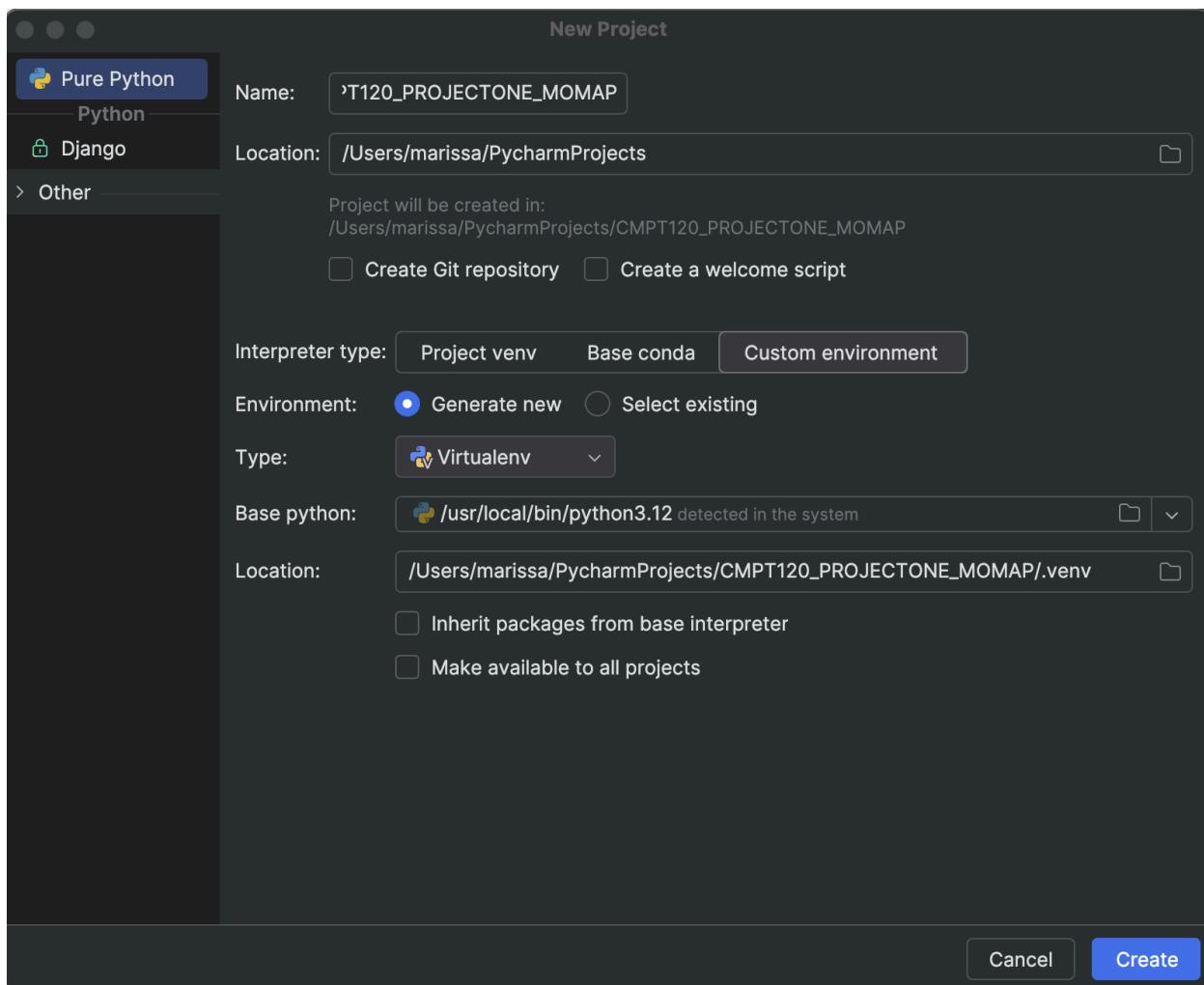
Running IDE on a Virtual Environment:

Figure 10: Running IDE on a Virtual Environment

User Interface Design

Login Page:

The first thing a user will see when accessing the program is the Login Page. This screen has three boxes with which the user can interact: Admin Login, User Login, and Exit. The ‘Admin Login’ allows the admin to enter the program while the ‘User Login’ allows the user to log in. The ‘Exit’ button exits the user out of the program.



Figure 11: Welcome & Login Page

```
class WelcomePage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        tk.Button(self, text="Admin Login", font=("Georgia", 18),
                  command=lambda: parent.show_page(AdminLoginPage)).pack(pady=10)    # pack places widget in parent widget
        tk.Button(self, text="User Login", font=("Georgia", 18), command=lambda: parent.show_page(UserPage)).pack(
            pady=10)
        tk.Button(self, text="Exit", font=("Georgia", 18), command=self.quit_application).pack(pady=10)

    def quit_application(self):
        messagebox.showinfo(title="TMS", message="Thank you for using Task Management System!")
        self.quit()
```

Figure 11.1: Welcome Page Code

```
# ADMIN LOGIN PAGE #
class AdminLoginPage(tk.Frame): 2 usages
    def __init__(self, parent):
        super().__init__(parent)
        ttk.Label(self, text="Admin Login", font=("Georgia", 18)).pack(pady=20)

        ttk.Label(self, text="Username", font=("Georgia", 10)).pack(pady=5)
        self.username_entry = tk.Entry(self)
        self.username_entry.pack()

        ttk.Label(self, text="Password", font=("Georgia", 10)).pack(pady=5)
        self.password_entry = tk.Entry(self, show="*")
        self.password_entry.pack()

        login_button = tk.Button(self, text="Login", font=("Georgia", 10), command=self.login)
        login_button.pack(pady=5)
        back_button = tk.Button(self, text="Back", font =("Georgia", 10), command=lambda: parent.show_page(WelcomePage))
        back_button.pack(pady=5)
    def login(self): 1 usage
        username = self.username_entry.get()
        password = self.password_entry.get()

        if username in users and users[username]["password"] == encrypt(password):
            messagebox.showinfo(title: "Login Successful", message: "Welcome, Admin!")
            self.master.show_page(TaskPage)
        else:
            messagebox.showerror(title: "Login Failed", message: "Invalid admin credentials.")
```

Figure 11.2: Login Page Code

Main Menu:

After the user logs in, they will be brought to a screen with six interactive buttons: Add Task, Edit Task, Remove Task, Search Task, View Calendar, and Back. Each button brings the user to a new screen.



Figure 12: Main Menu Page

```
class TaskPage(tk.Frame): 3 usages
    def __init__(self, parent):
        super().__init__(parent)
        self.master = parent
        tk.Label(self, text="Welcome to Task Management System", font=("Georgia", 24)).pack(pady=20)    ###
        tk.Button(self, text="Add Task", command=self.add_task).pack(pady=10)
        tk.Button(self, text="Edit Task", command=self.edit_task).pack(pady=10)
        tk.Button(self, text="Remove Task", command=self.remove_task).pack(pady=10)
        tk.Button(self, text="Search Tasks", command=self.search_tasks).pack(pady=10)
        tk.Button(self, text="View Calendar", command=lambda: parent.show_page(CalendarPage)).pack(pady=10)
        tk.Button(self, text="Back", command=lambda: parent.show_page(WelcomePage)).pack(pady=10)
```

Figure 12.1: Main Menu Page Code

Action Pages - Add Task:

This page prompts the user to enter four inputs for a task: Title, Time, Duration, and Description. After the user enters all four prompts, the program will then create the task.

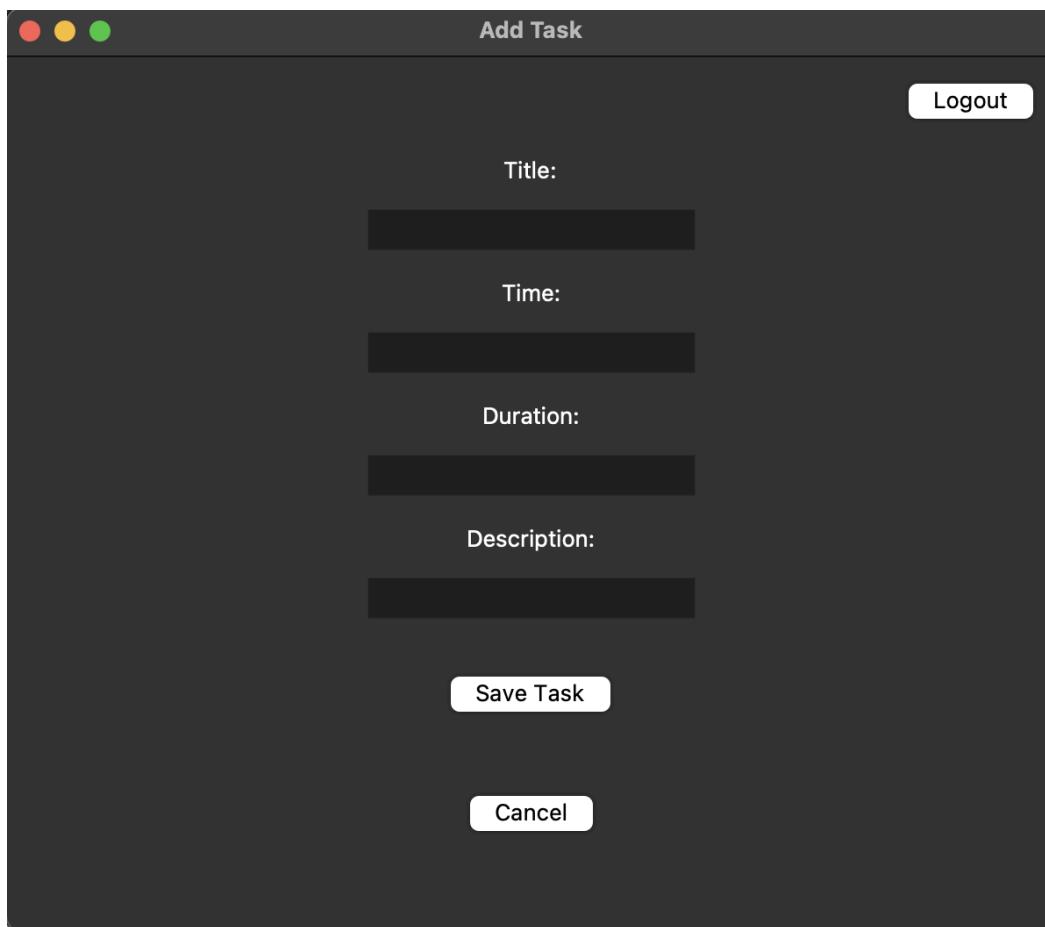


Figure 13: Add Task Menu

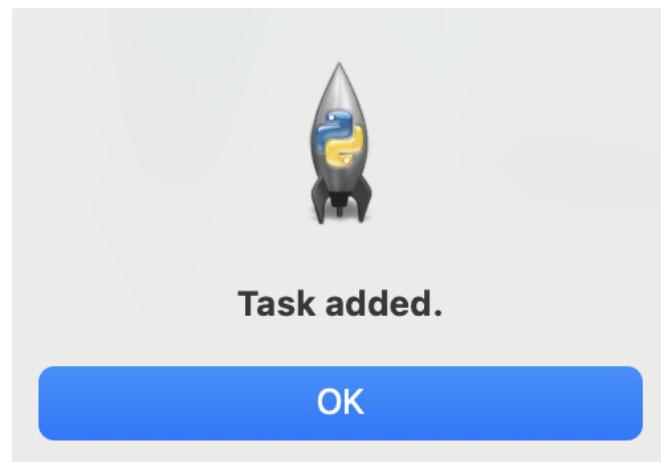


Figure 13.1: Add Task - Completion

```

# ADD TASK PAGE #
def add_task(self): # asks user to input fields. if nothing is entered the add task window closes out

    add_task_window = tk.Toplevel(self)
    add_task_window.title("Add Task")
    add_task_window.geometry("600x500")

    # Logout function
    def logout():
        add_task_window.destroy()
        self.master.show_page(WelcomePage) # Directs to WelcomePage

    tk.Button(add_task_window, text="Logout", command=logout).pack(anchor="ne", padx=10, pady=10)

    # Task entry form
    task = {}

    def save_task():
        task["title"] = title_entry.get().strip()
        task["time"] = time_entry.get().strip()
        task["duration"] = duration_entry.get().strip()
        task["description"] = description_entry.get().strip()

        if not task["title"] or not task["time"] or not task["duration"] or not task["description"]:
            messagebox.showerror(title: "Error", message: "All fields must be filled.")
            return

        username = "admin"
        if username in users:
            users[username]["tasks"].append(task)
            print(f"Task added: {task['title']}")
            self.master.save_task()
            messagebox.showinfo(title: "Success", message: "Task added.")
            add_task_window.destroy()
        else:
            messagebox.showerror(title: "Error", message: "User not found.")

# user enters info in the following fields
tk.Label(add_task_window, text="Title:").pack(pady=5)
title_entry = tk.Entry(add_task_window)
title_entry.pack(pady=5)

tk.Label(add_task_window, text="Time:").pack(pady=5)
time_entry = tk.Entry(add_task_window)
time_entry.pack(pady=5)

tk.Label(add_task_window, text="Duration:").pack(pady=5)
duration_entry = tk.Entry(add_task_window)
duration_entry.pack(pady=5)

tk.Label(add_task_window, text="Description:").pack(pady=5)
description_entry = tk.Entry(add_task_window)
description_entry.pack(pady=5)

# Save button
tk.Button(add_task_window, text="Save Task", command=save_task).pack(pady=20)
tk.Button(add_task_window, text="Cancel", command=add_task_window.destroy).pack(pady=20)

```

Figure 13.2: Add Task Code

Action Pages - Edit Task:

After the user clicks on the Edit Task button in the main menu page, the user will be directed to the task selection page where they will select the task they want to edit from the numbered list of available tasks. After selecting the task, the user will be directed to the edit task page where they can change the task's title, time, duration, and description. The updates can be saved or canceled according to the user's choice.

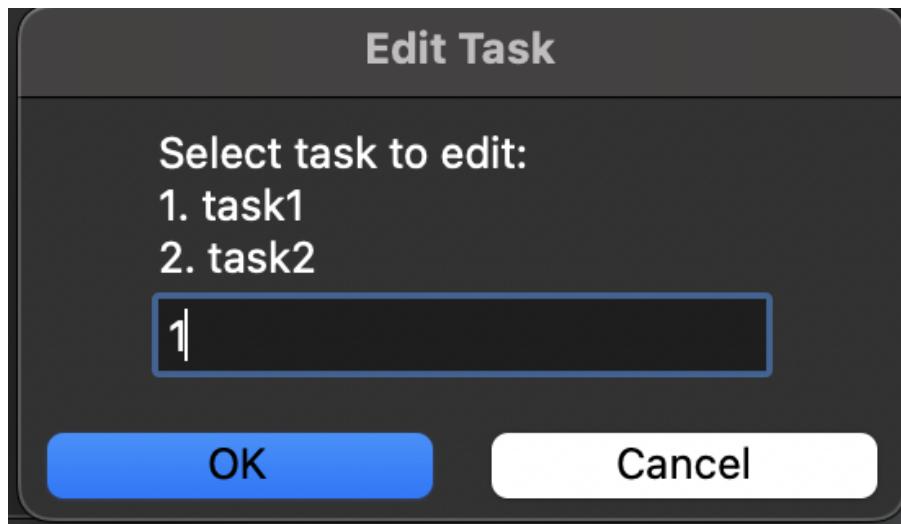


Figure 14: Task Selection for Editing

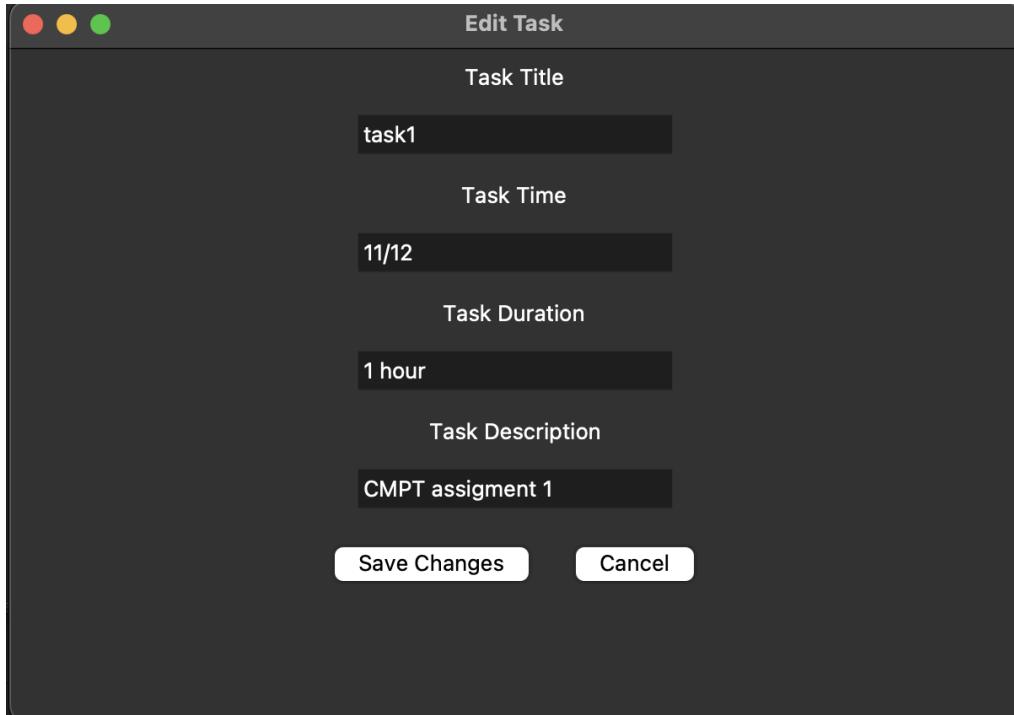


Figure 14.1: Edit Task Page

```

def edit_task(self):
    username = "admin"

    # To check if there are tasks available for this user
    if not users[username]["tasks"]:
        messagebox.showinfo("username: Any = " + username, "available to edit.") # shows this message if there are no tasks available to edit
        return

    task_titles = [task["title"] for task in users[username]["tasks"]] # access the task titles of this user
    task_titles_str = "\n".join(f"{i + 1}. {title}" for i, title in enumerate(task_titles)) # creates a single string with the task titles and numbers them
    task_num = tk.simpledialog.askinteger("Edit Task", "Select task to edit:\n" + task_titles_str) # asks to enter the number of the task user wants to edit

    if task_num is None or not (1 <= task_num <= len(task_titles)): # checks if the number selected is in the task list
        messagebox.showerror("Error", "Invalid selection.") # if number is not in the task list, shows an error message
        return

    task = users[username]["tasks"][task_num - 1] # retrieves the task the user wants to edit

    #window for the editing page
    edit_window = tk.Toplevel(self)
    edit_window.title("Edit Task")
    edit_window.geometry("600x400")

    # Create labels and entry fields for task details
    tk.Label(edit_window, text="Task Title").pack(pady=5)
    title_entry = tk.Entry(edit_window)
    title_entry.insert(0, task["title"]) #default value as current title
    title_entry.pack(pady=5, padx=10)

    tk.Label(edit_window, text="Task Time").pack(pady=5)
    time_entry = tk.Entry(edit_window)
    time_entry.insert(0, task["time"]) #default value as current time
    time_entry.pack(pady=5, padx=10)

    tk.Label(edit_window, text="Task Duration").pack(pady=5)
    duration_entry = tk.Entry(edit_window)
    duration_entry.insert(0, task["duration"]) #default value as current duration
    duration_entry.pack(pady=5, padx=10)

    tk.Label(edit_window, text="Task Description").pack(pady=5)
    description_entry = tk.Entry(edit_window)
    description_entry.insert(0, task["description"]) #default value as current description
    description_entry.pack(pady=5, padx=10)

    #frame for buttons
    button_frame = tk.Frame(edit_window)
    button_frame.pack(pady=10)

    #save button to update and save the task
    def save_changes():
        task["title"] = title_entry.get() or task["title"] #saves the updated one or if field empty keeps the initial value
        task["time"] = time_entry.get() or task["time"]
        task["duration"] = duration_entry.get() or task["duration"]
        task["description"] = description_entry.get() or task["description"]

        self.master.save_task()
        messagebox.showinfo("Success", "Task updated successfully!")
        edit_window.destroy() # Closes the edit window

    save_button = tk.Button(button_frame, text="Save Changes", command=save_changes)
    save_button.pack(side=tk.LEFT, padx=10)

    #a cancel button to close the window without saving
    cancel_button = tk.Button(button_frame, text="Cancel", command=edit_window.destroy)
    cancel_button.pack(side=tk.LEFT, padx=10)

```

Figure 14.2: Edit Task Code

Action Pages - Remove Task:

This page prompts the user to enter the title of the task to be removed. If the task title exists, then the task is removed, otherwise, a screen that says ‘task not found’ appears.

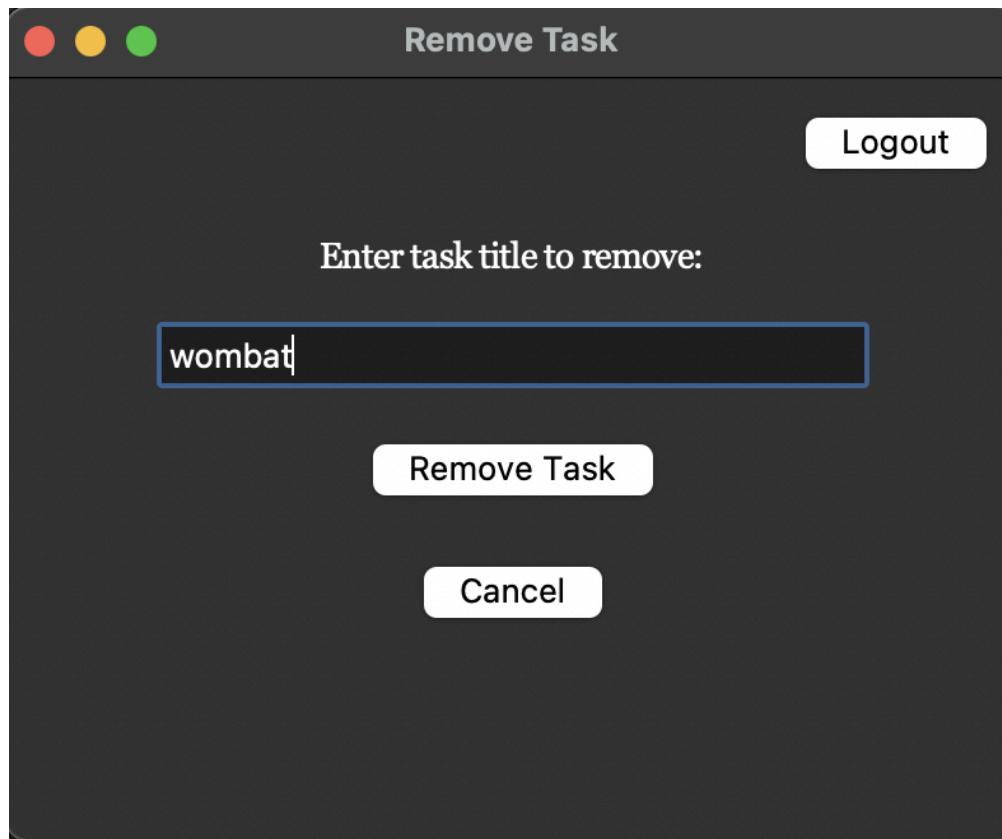


Figure 15: Remove Task

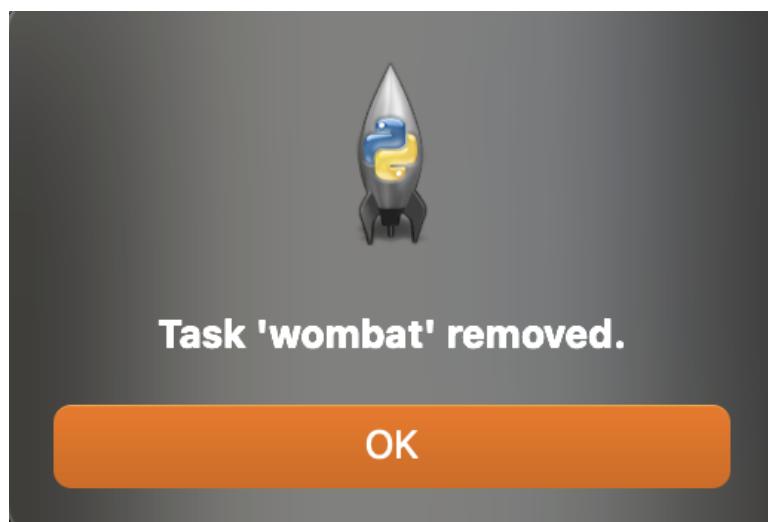


Figure 15.1: Removed Task Successfully

```

def remove_task(self):

    remove_window = tk.Toplevel(self)
    remove_window.title("Remove Task")
    remove_window.geometry("400x300")

    username = "admin"

    # Logout button
    def logout():
        remove_window.destroy()
        self.master.show_page(WelcomePage)

    logout_button = tk.Button(remove_window, text="Logout", command=logout)
    logout_button.pack(anchor="ne", padx=10, pady=10)

    # entry field to enter which task to remove
    tk.Label(remove_window, text="Enter task title to remove:", font=("Georgia", 14)).pack(pady=10)
    task_title_entry = tk.Entry(remove_window, width=30)
    task_title_entry.pack(pady=5)

    # removes the task based on input user provided
    def remove():
        task_title = task_title_entry.get().strip() # Retrieve title from the above entry field

        if not task_title:
            messagebox.showerror(title="Error", message="Please enter a task title")
            return

        if username in users:
            task_to_remove = None
            print(f"Looking for task title: {task_title.strip().lower()}") # Debugging log
            for task in users[username]["tasks"]:
                print(f"Comparing task '{task['title'].strip().lower()}' with stored task '{task['title'].strip().lower()}'")
                if task["title"].strip().lower() == task_title.strip().lower(): # Case-insensitive comparison
                    task_to_remove = task
                    break

            if task_to_remove:
                users[username]["tasks"].remove(task_to_remove)

                self.master.save_task()
                messagebox.showinfo(title="Success", message=f"Task '{task_title}' removed.")
                remove_window.destroy()
            else:
                messagebox.showwarning(title="Not Found", message=f"Task '{task_title}' not found.")

        else:
            messagebox.showerror(title="Error", message="User not found.")

    # Buttons for Remove Task and Cancel
    remove_button = tk.Button(remove_window, text="Remove Task", command=remove)
    remove_button.pack(pady=10)

    cancel_button = tk.Button(remove_window, text="Cancel", command=remove_window.destroy)
    cancel_button.pack(pady=10)

```

Figure 15.2: Remove Task Code

Action Pages - Search Task:

This page prompts the user to enter the title of the task that they wish to find. If the task title exists, then the task is brought up, otherwise, the screen remains blank.

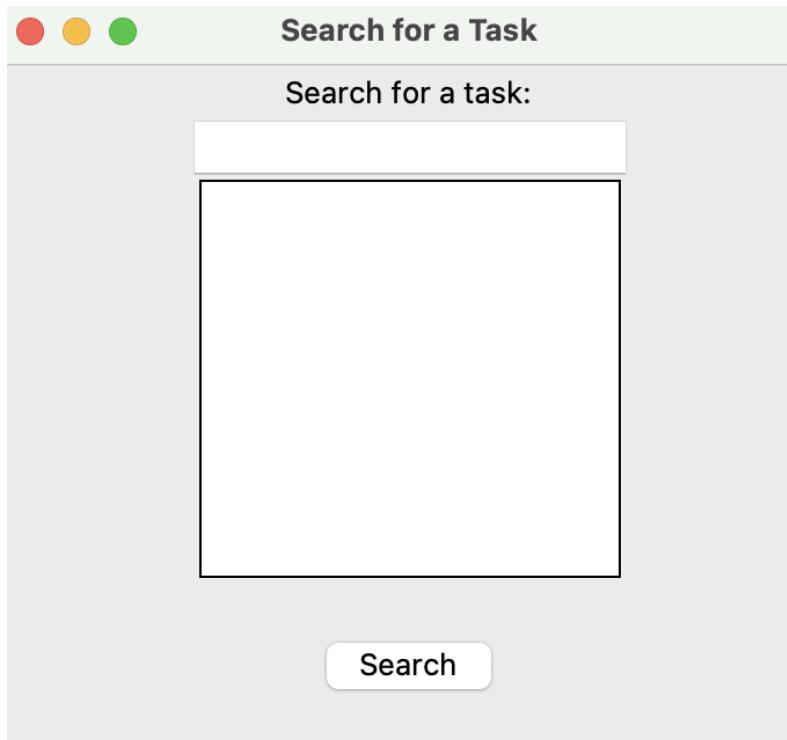


Figure 16: Search Task Option

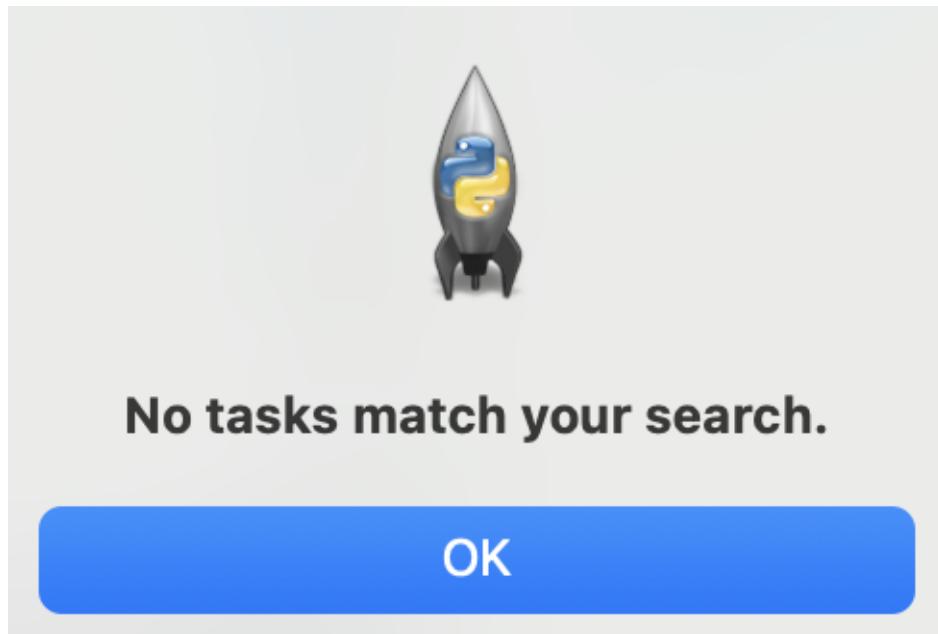


Figure 16.1: Search Task Not Found

Figure 16.2: Search Task Code

```

else:
    messagebox.showerror(title: "Error", message: "User not found")

def show_task_details(event):
    selection = task_listbox.curselection()
    if not selection:
        return
    selected_task_title = task_listbox.get(selection[0]).strip().lower()
    print(f"Debug: Selected task title: {selected_task_title}")

    username = self.master.current_user
    if not username:
        messagebox.showerror(title: "Error", message: "No user logged in")
        return

    task_details = next(
        (task for task in users[username]["tasks"] if task["title"].strip().lower() == selected_task_title),
        None
    )
    print(f"Retrieved task details: {task_details}")

    if task_details:
        detail_window = tk.Toplevel(self)
        detail_window.title("Task Details")
        detail_window.geometry("400x300")
        tk.Label(detail_window, text=f"Title: {task_details['title']}").pack(pady=5)
        tk.Label(detail_window, text=f"Time: {task_details['time']}").pack(pady=5)
        tk.Label(detail_window, text=f"Duration: {task_details['duration']}").pack(pady=5)
        tk.Label(detail_window, text=f"Description: {task_details['description']}").pack(pady=5)
        tk.Button(detail_window, text="Close", command=detail_window.destroy).pack(pady=10)
    else:
        messagebox.showerror(title: "Error", message: "Task not found.")
task_listbox.bind("<<ListboxSelect>>", show_task_details)

# button to trigger the search
search_button = tk.Button(search_window, text="Search", command=perform_search)
search_button.pack(pady=5)

close_button = tk.Button(search_window, text="Close", command=search_window.destroy)
close_button.pack(pady=5)

```

Figure 16.3: Search Task Code (Cont.)

Action Pages - View Calendar:

After the user clicks on this button, they will be brought to a digital calendar. By clicking on dates on the calendar they can view each date's respective tasks.

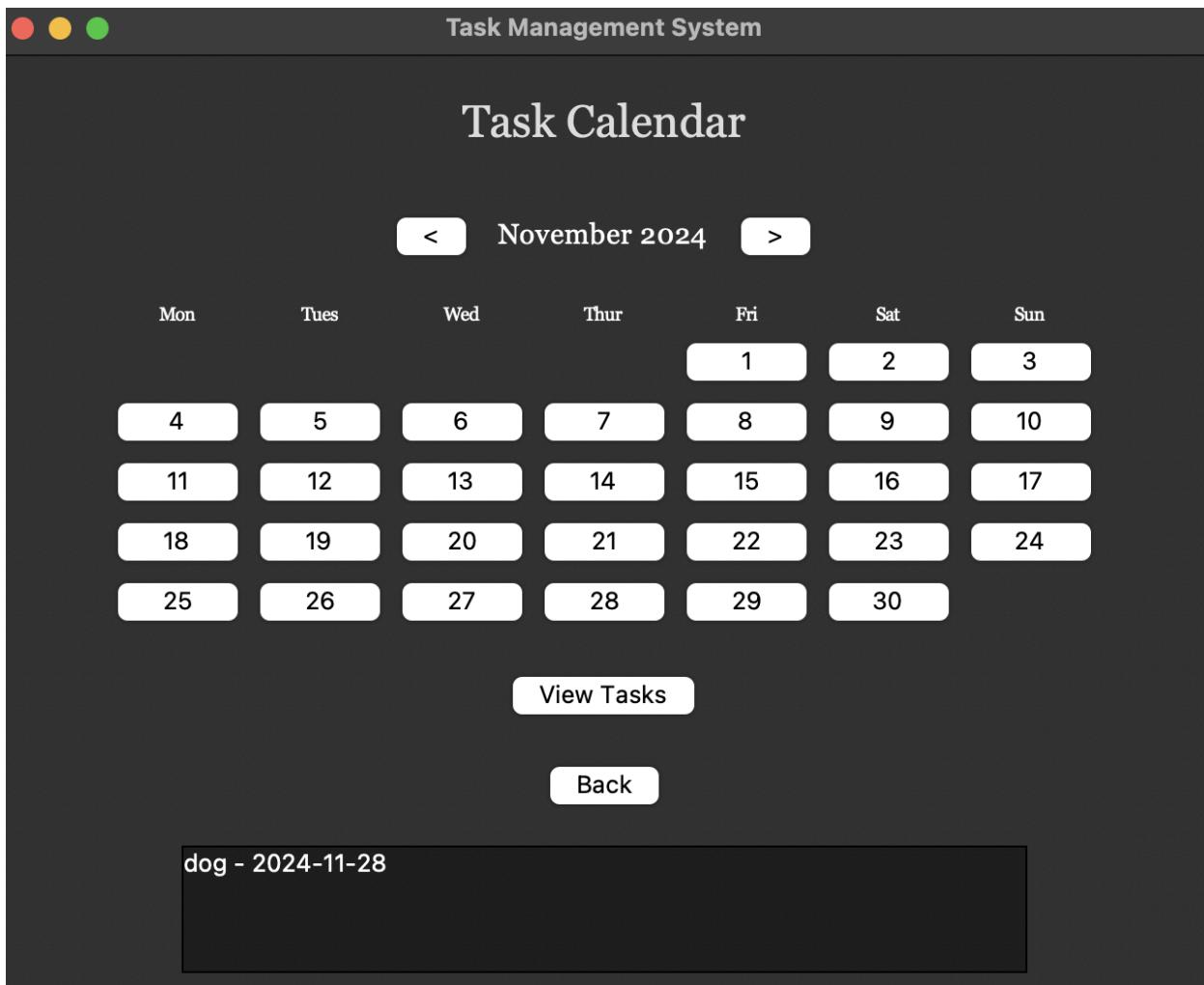


Figure 17: Calendar Page

```
# CALENDAR PAGE #
class CalendarPage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent)
        self.master = parent
        self.current_month = datetime.now().month
        self.current_year = datetime.now().year
        self.selected_date = None

        ttk.Label(self, text="Task Calendar", font=("Georgia", 24)).pack(pady=20)

        nav_frame = tk.Frame(self)
        nav_frame.pack(pady=10)

        tk.Button(nav_frame, text="<", command=self.prev_month).pack(side=tk.LEFT, padx=10)
        self.month_label = tk.Label(nav_frame, text=self.get_month_year_label(), font=("Georgia", 15))
        self.month_label.pack(side=tk.LEFT)
        tk.Button(nav_frame, text=">", command=self.next_month).pack(side=tk.LEFT, padx=10)

        self.calendar_frame = tk.Frame(self)
        self.calendar_frame.pack(pady=10)
        self.create_calendar()

        tk.Button(self, text="View Tasks", command=self.view_tasks).pack(pady=10)
        tk.Button(self, text="Back", command=lambda: parent.show_page(TaskPage)).pack(pady=10)

        self.task_listbox = tk.Listbox(self, height=10, width=50)
        self.task_listbox.pack(pady=10)

        tk.Button(self, text="Logout", command=self.logout).pack(pady=10)

        tk.Button(self, text="Back", command=lambda: parent.show_page(TaskPage)).pack(pady=10)

    def logout(self):
        # logout button
        self.master.current_user = None
        messagebox.showinfo(title="Logout", message="You have been logged out.")
        self.master.show_page(WelcomePage) # Redirect to WelcomePage

    def create_calendar(self):
        for widget in self.calendar_frame.winfo_children():
            widget.destroy()
```

Figure 17.1: Calendar Code

```
days_in_month = calendar.monthrange(self.current_year, self.current_month)[1]
first_weekday = calendar.monthrange(self.current_year, self.current_month)[0]

for day in ["Mon", "Tues", "Wed", "Thur", "Fri", "Sat", "Sun"]:
    tk.Label(self.calendar_frame, text=day, font=("Georgia", 10), width=4).grid(row=0,
                                                                           column=["Mon", "Tues", "Wed",
                                                                           "Thur", "Fri", "Sat",
                                                                           "Sun"].index(day))

row = 1
col = first_weekday
for day in range(1, days_in_month + 1):
    day_button = tk.Button(self.calendar_frame, text=str(day), width=4,
                           command=lambda d=day: self.select_date(d))
    day_button.grid(row=row, column=col, padx=2, pady=2)
    col += 1
    if col > 6:
        col = 0
        row += 1

def get_month_year_label(self):
    return f"{calendar.month_name[self.current_month]} {self.current_year}"

def prev_month(self):
    if self.current_month == 1:
        self.current_month = 12
        self.current_year -= 1
    else:
        self.current_month -= 1
    self.month_label.config(text=self.get_month_year_label())
    self.create_calendar()

def next_month(self):
    if self.current_month == 12:
        self.current_month = 1
        self.current_year += 1
    else:
        self.current_month += 1
    self.month_label.config(text=self.get_month_year_label())
    self.create_calendar()
```

Figure 17.2: Calendar Code (Cont.)

```

def select_date(self, day):
    try:
        self.selected_date = f"{self.current_year}-{self.current_month:02d}-{day:02d}"
        print(f"Date selected: {self.selected_date}")
        self.view_tasks()
    except AttributeError:
        print("Error: Current year or month is not defined. Make sure they are initialized")

def view_tasks(self):
    # shows tasks for selected date
    self.task_listbox.delete(first=0, tk.END)
    print(f"Selected date: {self.selected_date}")

    username = self.master.current_user
    if not username:
        messagebox.showerror(title="Error", message="No user logged in")
        return

    selected_date = self.selected_date
    if not selected_date:
        messagebox.showinfo(title="Info", message="Select a date to view tasks")
        return

    tasks = users.get(username, {}).get("tasks", [])
    tasks_on_date = [task for task in tasks if task.get("time", "") == selected_date]

    print(f"Viewing tasks on {selected_date}: {tasks_on_date}")

    if tasks_on_date:
        for task in tasks_on_date:
            self.task_listbox.insert(tk.END, *elements: f"{task['title']} - {task['time']}")
    else:
        self.task_listbox.insert(tk.END, *elements: "No tasks on this date")

def logout():
    view_tasks_window.destroy()
    self.master.show_page(WelcomePage) # Directs to WelcomePage

```

Figure 17.3: Calendar Code (Cont. (2))

Data Storage:

Our text file for added tasks is named “tms_tasks.txt”. In our application, we have defined the functions `save_task` and `load_task` to save and retrieve the tasks within the file. In this instance, the function `save_task` is opened in the append state to allow users to add tasks without overwriting existing content. `load_task` is opened in a reading state since this action does not entail adding information to the file. We also have defined a function “`add_task`” which presents the interface through which the user will add a task along with its title, time, duration, and description.

We have other files available to help users navigate our system. Through our `add_task` function, a file is created when a user begins adding a task to the system. Then our program asks the user to input time, title, and duration. If their input runs through our code smoothly, the code will save their task and add it to the calendar. If their input is not valid, the user cannot continue. There is also the option to remove a task using the `task_title` function. The user must select an already existing task, then the system will ask if they want it removed. The user is provided with the option of yes or no. If they decide to click yes, the code will delete the selected task. If the user clicks no, the system will return to the main menu. Users also have the ability to search for a task. They have the input of a time, title, or duration. If the input is valid, the function `perform_search` will run and the system will load into the task page. Our code provides users with the ability to edit tasks. The function `edit_task` will check if there are any available tasks. The system asks for user input and then checks if their input is valid; otherwise an error will appear. Then the code will retrieve the task, and the user has the option to change the title, time, duration, and description of the task. All the user’s changes are saved into the file if they select save.

Our program also allows users to set and change their username/password. Their data is stored through our `add_user` function. We use our `remove_user` function to prevent having various users logged in. The data is stored in the file “`users.txt`” once the changes have been approved by the system.

Improved User Interface and Report:

Login/Welcome Menu:

The login and welcome page was improved by adding a logo, including a uniform color scheme, and by adding more visual interactive features for the user. These features include a “About Us” page, buttons for the admin and user to login, a button to register for a user account, and a button for the user to hit that would exit the system. Additionally, the user’s login information is now encrypted behind a Caesar cipher, protecting any information the user would put into the system.



Figure 18: Improved Welcome & Login Page

```

# WELCOME PAGE #
class WelcomePage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent, bg=COMMON_BG_COLOR)
        style = ttk.Style()
        style.theme_use("clam")
        style.configure(
            style: "Rounded.TButton",
            background="grey", # button background color
            foreground="black", # text color
            font=("Montserrat", 18),
            borderwidth=1,
            relief="flat",
            padding=10
        )
        style.map(
            style: "Rounded.TButton",
            background=[("active", "lightgrey")], # changes button color when hovered
            foreground=[("active", "black")]
        )

    # opens the logo and converts transparent areas to the background color
    bg_color = "white"
    logo_image = Image.open("momap_logo-removebg-preview.png").convert("RGBA")

    # creates a solid background with the same size as the logo
    background = Image.new(mode="RGBA", size=logo_image.size, bg_color)
    combined = Image.alpha_composite(background, logo_image)

    # uses the processed image in tkinter
    self.logo_image_tk = ImageTk.PhotoImage(combined)
    logo_label = tk.Label(self, image=self.logo_image_tk, bg="white")
    logo_label.image = self.logo_image_tk # Keep a reference to prevent garbage collection
    logo_label.pack(pady=20)

ttk.Button(self, text="About Us", style="Rounded.TButton",
          command=lambda: parent.show_page(AboutUsPage)).pack(pady=10)
ttk.Button(self, text="Admin Login", style="Rounded.TButton",
          command=lambda: parent.show_page(AdminLoginPage)).pack(pady=10)
ttk.Button(self, text="User Login", style="Rounded.TButton",
          command=lambda: parent.show_page(UserLoginPage)).pack(pady=10)
ttk.Button(self, text="Register", style="Rounded.TButton",
          command=lambda: parent.show_page(RegisterPage)).pack(pady=10)
ttk.Button(self, text="Exit", style="Rounded.TButton",
          command=self.quit_application).pack(pady=10)

def quit_application(self):
    messagebox.showinfo(title: "TMS", message: "Thank you for using Task Management System!")
    self.quit()

```

Figure 18.1: Improved Welcome & Login Page Code

Admin Login

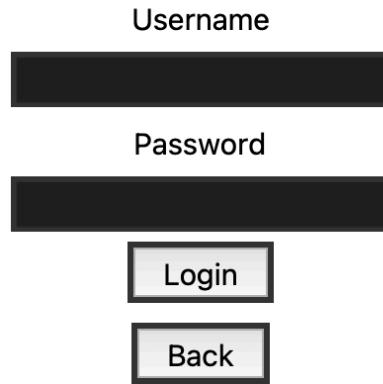


Figure 19: Admin Login Page

```
# ADMIN LOGIN PAGE #
class AdminLoginPage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent, bg=COMMON_BG_COLOR)
        tk.Label(self, text="Admin Login", font=("Montserrat", 24), bg="white", fg="black").pack(pady=20)

        tk.Label(self, text="Username", font=("Montserrat", 15), bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
        self.username_entry = tk.Entry(self)
        self.username_entry.pack()

        tk.Label(self, text="Password", font=("Montserrat", 15), bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
        self.password_entry = tk.Entry(self, show="*")
        self.password_entry.pack()

        login_button = tk.Button(self, text="Login", font=("Montserrat", 15), command=self.login)
        login_button.pack(pady=5)
        back_button = tk.Button(self, text="Back", font=("Montserrat", 15), command=lambda: parent.show_page(WelcomePage))
        back_button.pack(pady=5)

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        if username in users and users[username]["password"] == encrypt(password):
            self.master.current_user = username
            messagebox.showinfo(title="Login Successful", message=f"Welcome, {username}!")
            self.master.show_page(TaskPage)
        else:
            messagebox.showerror(title="Login Failed", message="Invalid admin credentials.")


```

Figure 19.1: Admin Login Page Code

User Login

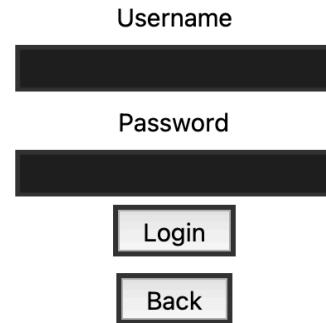


Figure 20: User Login Page

```
# USER EDIT PAGE #
class UserLoginPage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="white")
        tk.Label(self, text="User Login", font=("Montserrat", 24), bg=COMMON_BG_COLOR, fg="black").pack(pady=20)

        tk.Label(self, text="Username", font=("Montserrat", 15), bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
        self.username_entry = tk.Entry(self)
        self.username_entry.pack()

        tk.Label(self, text="Password", font=("Montserrat", 15), bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
        self.password_entry = tk.Entry(self, show="*")
        self.password_entry.pack()

        login_button = tk.Button(self, text="Login", font=("Montserrat", 15), command=self.login)
        login_button.pack(pady=5)
        back_button = tk.Button(self, text="Back", font=("Montserrat", 15),
                               command=lambda: parent.show_page(WelcomePage))
        back_button.pack(pady=5)

    def login(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        if username in users and users[username]["password"] == encrypt(password):
            self.master.current_user = username
            messagebox.showinfo(title="Login Successful", message=f"Welcome, {username}!")
            self.master.show_page(TaskPage)
        else:
            messagebox.showerror(title="Login Failed", message="Invalid user credentials.")


```

Figure 20.1: User Login Page Code

```
Loaded users: {'admin': {'password': 'qt}y~!q$$'}
```

```
# CAESAR CIPHER FUNCTION #
def encrypt(text, shift=encryptionShift):
    return ''.join(chr((ord(char) - 32 + shift) % 95 + 32) for char in
                  text) # ensures that characters stay within the bounds of ASCII 32 - 126 during encryption

def decrypt(text, shift=encryptionShift):
    return ''.join(chr((ord(char) - 32 - shift) % 95 + 32) for char in text)
```

Figure 21: Caesar Cipher Encryption

Main Menu Page:

The main menu was improved by including a more uniform color scheme and by adding a better visual layout. This allows the user to better see what the system has to offer.

Welcome to Task Management System

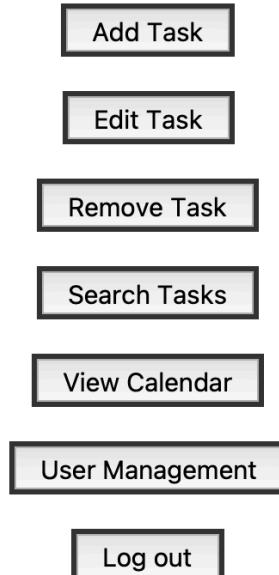


Figure 22: Improved Main Menu Page

```
# TASK MANAGEMENT MENU/WELCOME PAGE #
class TaskPage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="white")
        self.master = parent
        tk.Label(self, text="Welcome to Task Management System", font=("Montserrat", 24), bg="white", fg="black").pack(pady=20) ####
        tk.Button(self, text="Add Task", font=("Montserrat", 15), command=self.add_task).pack(pady=10)
        tk.Button(self, text="Edit Task", font=("Montserrat", 15), command=self.edit_task).pack(pady=10)
        tk.Button(self, text="Remove Task", font=("Montserrat", 15), command=self.remove_task).pack(pady=10)
        tk.Button(self, text="Search Tasks", font=("Montserrat", 15), command=self.search_tasks).pack(pady=10)
        tk.Button(self, text="View Calendar", font=("Montserrat", 15), command=lambda: parent.show_page(CalendarPage)).pack(pady=10)

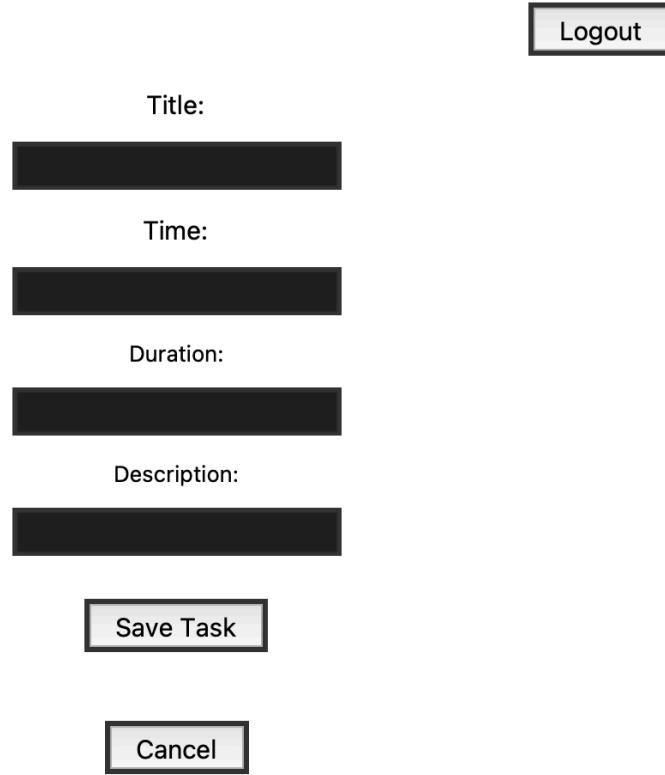
        tk.Button(self, text="User Management", font=("Montserrat", 15), command=self.open_user_management).pack(pady=10)

        tk.Button(self, text="Log out", font=("Montserrat", 15), command=self.logout_and_go_to_welcome).pack(pady=10)
```

Figure 22.1: Improved Main Menu Page Code

Add Task:

The add task page was improved by creating a more uniform color scheme so that it aligns with the other pages to create a consistent and professional appearance throughout the application. Additionally, the font size for the labels and entry fields were adjusted to improve readability, and the button sizes were also increased to improve accessibility and ease of use.



The diagram illustrates the improved 'Add Task' page interface. At the top right is a 'Logout' button. Below it is a 'Title:' label followed by a large, dark rectangular input field. Next is a 'Time:' label followed by another large, dark rectangular input field. Then is a 'Duration:' label followed by a third large, dark rectangular input field. Below these is a 'Description:' label followed by a fourth large, dark rectangular input field. At the bottom center is a 'Save Task' button, and at the very bottom center is a 'Cancel' button.

Figure 23: Improved Add Task Page

```
# ADD TASK PAGE #
def add_task(self): # asks user to input fields. if nothing is entered, the add task window closes
    add_task_window = tk.Toplevel(self, bg="white")
    add_task_window.title("Add Task")
    add_task_window.geometry("600x500")

    # Logout function: Close the add task window and redirect to the welcome page
    def logout():
        add_task_window.destroy()
        self.master.show_page(WelcomePage) # Directs to WelcomePage

    tk.Button(add_task_window, text="Logout", command=logout, font=("Montserrat", 15)).pack(anchor="ne",
                                                                                      padx=10, pady=10)

    # Task entry form and save functionality
    def save_task():
        username = self.master.current_user
        task = {}

        task_title = title_entry.get().strip()

        # Check for duplicate task title
        if username in users and any(
            task["title"].lower() == task_title.lower() for task in users[username]["tasks"]):
            messagebox.showerror(title="Error",
                                 message=f"A task with the title '{task_title}' already exists. Please use a different name.")
            return # Stop execution if a duplicate is found

        task["title"] = task_title
        task["time"] = time_entry.get().strip()
        task["duration"] = duration_entry.get().strip()
        task["description"] = description_entry.get().strip()

        # Validate that all fields are filled
        if not task["title"] or not task["time"] or not task["duration"] or not task["description"]:
            messagebox.showerror(title="Error", message="All fields must be filled.")
            return
```

Figure 23.1: Improved Add Task Page Code

```
# Save the task for the current user
if username in users:
    users[username]["tasks"].append(task)
    print(f"Task added: {task['title']}")  
    self.master.save_task()
    messagebox.showinfo(title: "Success", message: "Task added.")
    add_task_window.destroy()
else:
    messagebox.showerror(title: "Error", message: "User not found.")

# creates input fields for task details
tk.Label(add_task_window, text="Title:", bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
title_entry = tk.Entry(add_task_window)
title_entry.pack(pady=5)

tk.Label(add_task_window, text="Time:", bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
time_entry = tk.Entry(add_task_window)
time_entry.pack(pady=5)

tk.Label(add_task_window, text="Duration:", bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
duration_entry = tk.Entry(add_task_window)
duration_entry.pack(pady=5)

tk.Label(add_task_window, text="Description:", bg=COMMON_BG_COLOR, fg="black").pack(pady=5)
description_entry = tk.Entry(add_task_window)
description_entry.pack(pady=5)

# Save and Cancel buttons
tk.Button(add_task_window, text="Save Task", command=save_task, font=("Montserrat", 15)).pack(pady=20)
tk.Button(add_task_window, text="Cancel", command=add_task_window.destroy, font=("Montserrat", 15)).pack(
    pady=20)
```

Figure 23.2: Improved Add Task Page Code (Cont.)

Remove Task:

The remove task page was improved by using a consistent color scheme to match the other pages so that the application looks more polished and cohesive. The font size for the label and entry field was also adjusted so it's easier for the user to read the text, and the button sizes were also resized to make them more user friendly and accessible.

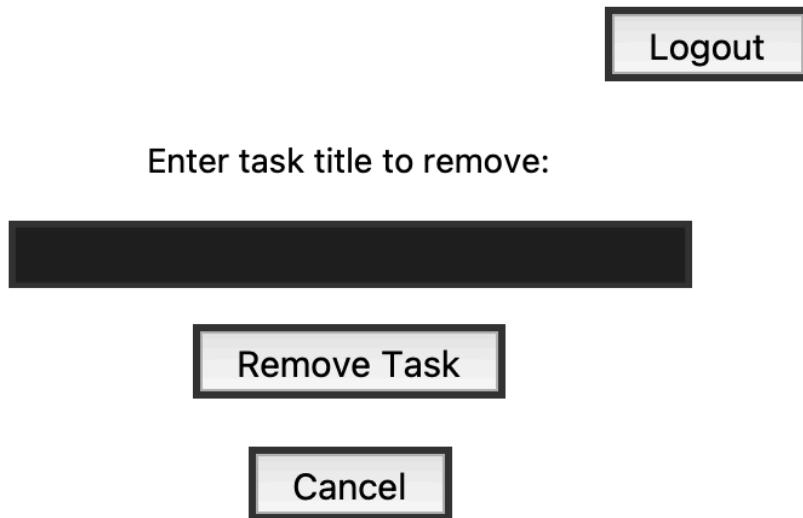


Figure 24: Improved Remove Task Page

```

def remove_task(self):

    remove_window = tk.Toplevel(self, bg="white")
    remove_window.title("Remove Task")
    remove_window.geometry("400x300")

    username = self.master.current_user

    # Logout button
    def logout():
        remove_window.destroy()
        self.master.show_page(WelcomePage)

    logout_button = tk.Button(remove_window, text="Logout", font=("Montserrat", 15), command=logout)
    logout_button.pack(anchor="ne", padx=10, pady=10)

    # entry field to enter which task to remove
    tk.Label(remove_window, text="Enter task title to remove:", font=("Montserrat", 15), bg=COMMON_BG_COLOR,
             fg="black").pack(pady=10)
    task_title_entry = tk.Entry(remove_window, width=30)
    task_title_entry.pack(pady=5)

    # removes the task based on input user provided
    def remove():
        task_title = task_title_entry.get().strip() # Retrieve title from the above entry field

        if not task_title:
            messagebox.showerror(title="Error", message="Please enter a task title")
            return

        if username in users:
            task_to_remove = None
            print(f"Looking for task title: {task_title.strip().lower()}") # Debugging log
            for task in users[username]["tasks"]:
                print(
                    f"Comparing task '{task['title'].strip().lower()}' with stored task '{task['title'].strip().lower()}'")
                if task["title"].strip().lower() == task_title.strip().lower(): # Case-insensitive comparison
                    task_to_remove = task
                    break

        if task_to_remove:
            users[username]["tasks"].remove(task_to_remove)
            self.master.save_task()
            messagebox.showinfo(title="Success", message=f"Task '{task_title}' removed.")
            remove_window.destroy()
        else:
            messagebox.showwarning(title="Not Found", message=f"Task '{task_title}' not found.")

        else:
            messagebox.showerror(title="Error", message="User not found.")

    # buttons for remove task and cancel
    remove_button = tk.Button(remove_window, text="Remove Task", font=("Montserrat", 15), command=remove)
    remove_button.pack(pady=10)

    cancel_button = tk.Button(remove_window, text="Cancel", font=("Montserrat", 15),
                             command=remove_window.destroy)
    cancel_button.pack(pady=10)

```

Figure 24.1: Improved Remove Task Page Code

Edit Task:

The edit page was improved by allowing the user to see an interface that depicted all the tasks available for the user to edit and in an organized manner. Additionally, a distinction was made between the two buttons offered to the user, one that allowed the user to exit out of the program entirely and one that canceled the task.

Task Title	Time	Duration	Description
Walk the Dog	2024-12-01	2 Hours	Walk Rocky

[Logout](#) [Close](#)
[Edit Selected Task](#)

Figure 25: Improved Edit Task Page

```
def edit_task(self):
    username = self.master.current_user

    if not users[username]["tasks"]:
        messagebox.showinfo(title="No Tasks Available", message="No tasks available to edit.")
        return

    # Create the Edit Task window
    edit_window = tk.Toplevel(self)
    edit_window.title("Edit Tasks")
    edit_window.geometry("800x500")
    edit_window.configure(bg="white")

    tree = ttk.Treeview(edit_window, columns=("Title", "Time", "Duration", "Description"), show="headings",
                        height=15)
    tree.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)

    # Define columns
    tree.heading("Title", text="Task Title")
    tree.heading("Time", text="Time")
    tree.heading("Duration", text="Duration")
    tree.heading("Description", text="Description")

    tree.column("Title", anchor="center", width=200)
    tree.column("Time", anchor="center", width=100)
    tree.column("Duration", anchor="center", width=100)
    tree.column("Description", anchor="center", width=300)

    # Populate Treeview with tasks
    for i, task in enumerate(users[username]["tasks"]):
        tree.insert("", index="end", iid=i, values=(task["title"], task["time"], task["duration"], task["description"]))

    # Button Frame
    button_frame = tk.Frame(edit_window, bg="white")
    button_frame.pack(pady=10)

    # Logout Button
    def logout_and_close():
        self.master.current_user = None
        edit_window.destroy() # Close the Edit Task window
        messagebox.showinfo(title="Logout", message="You have been logged out.")

    button_frame.quit_button = tk.Button(button_frame, text="Logout", command=logout_and_close)
```

Figure 25.1: Improved Edit Task Page Code

```
self.master.show_page(WelcomePage) # Redirect to WelcomePage

tk.Button(
    button_frame,
    text="Logout",
    command=logout_and_close,
    font=("Montserrat", 15),
    bg="grey",
    fg="black",
).grid(row=0, column=0, padx=10)

# close button
tk.Button(
    button_frame,
    text="Close",
    command=edit_window.destroy,
    font=("Montserrat", 15),
    bg="grey",
    fg="black"
).grid(row=0, column=2)

# function to edit a selected task
def edit_task_popup():
    selected_item = tree.selection()
    if not selected_item:
        messagebox.showerror(title="Error", message="No task selected.")
        return

    item_id = selected_item[0]
    task = users[username]["tasks"][int(item_id)]

    # Creates a popup window for editing
    popup = tk.Toplevel(edit_window)
    popup.title("Edit Task")
    popup.geometry("400x400")

    # Editable fields
    tk.Label(popup, text="Task Title:").pack(pady=5)
```

Figure 25.2: Improved Edit Task Page Code (Cont.)

```

        title_entry = tk.Entry(popup, font=("Montserrat", 15))
        title_entry.insert( index: 0, task["title"])
        title_entry.pack(pady=5)

        tk.Label(popup, text="Time:").pack(pady=5)
        time_entry = tk.Entry(popup, font=("Montserrat", 15))
        time_entry.insert( index: 0, task["time"])
        time_entry.pack(pady=5)

        tk.Label(popup, text="Duration:").pack(pady=5)
        duration_entry = tk.Entry(popup, font=("Montserrat", 15))
        duration_entry.insert( index: 0, task["duration"])
        duration_entry.pack(pady=5)

        tk.Label(popup, text="Description:").pack(pady=5)
        description_entry = tk.Entry(popup, font=("Montserrat", 15))
        description_entry.insert( index: 0, task["description"])
        description_entry.pack(pady=5)

        def save_changes():
            new_title = title_entry.get().strip()
            new_time = time_entry.get().strip()
            new_duration = duration_entry.get().strip()
            new_description = description_entry.get().strip()

            # Validate input
            if not new_title or not new_time or not new_duration or not new_description:
                messagebox.showerror( title: "Error", message: "All fields must be filled.")
                return

            # Check for duplicate task titles
            if any(t["title"].lower() == new_title.lower() for t in users[username]["tasks"] if t != task):
                messagebox.showerror( title: "Error", message: f"A task with the title '{new_title}' already exists.")
                return

        # Updates task details
        task["title"] = new_title
        task["time"] = new_time
        task["duration"] = new_duration
        task["description"] = new_description

        # Updates treeview
        tree.item(item_id, values=(new_title, new_time, new_duration, new_description))

        self.master.save_task()
        messagebox.showinfo( title: "Success", message: "Task updated successfully!")
        popup.destroy()

        tk.Button(popup, text="Save Changes", command=save_changes, bg="white", fg="black").pack(pady=10)
        tk.Button(popup, text="Cancel", command=popup.destroy, bg="white", fg="black").pack(pady=5)

    # Buttons for edit/delete
    button_frame = tk.Frame(edit_window, bg="#f4f4f9")
    button_frame.pack(pady=10)

    tk.Button(button_frame, text="Edit Selected Task", command=edit_task_popup, font=("Montserrat", 15), bg="#007bff",
              fg="black").grid(row=0, column=0, padx=10)
    logout_button = tk.Button(remove_window, text="Logout", command=logout)
    logout_button.pack(anchor="ne", padx=10, pady=10)
    tk.Button(button_frame, text="Close", command=edit_window.destroy, font=("Montserrat", 15)).grid(row=0, column=2,
                                                padx=10)

```

Figure 25.3: Improved Edit Task Page Code (Cont. (2))

Search Task:

The search page was improved by providing a better visual layout, a button that allows the user to instantly log out of the system, and a button that allows the user to exit out of the action, which wasn't available to the user in the original design.

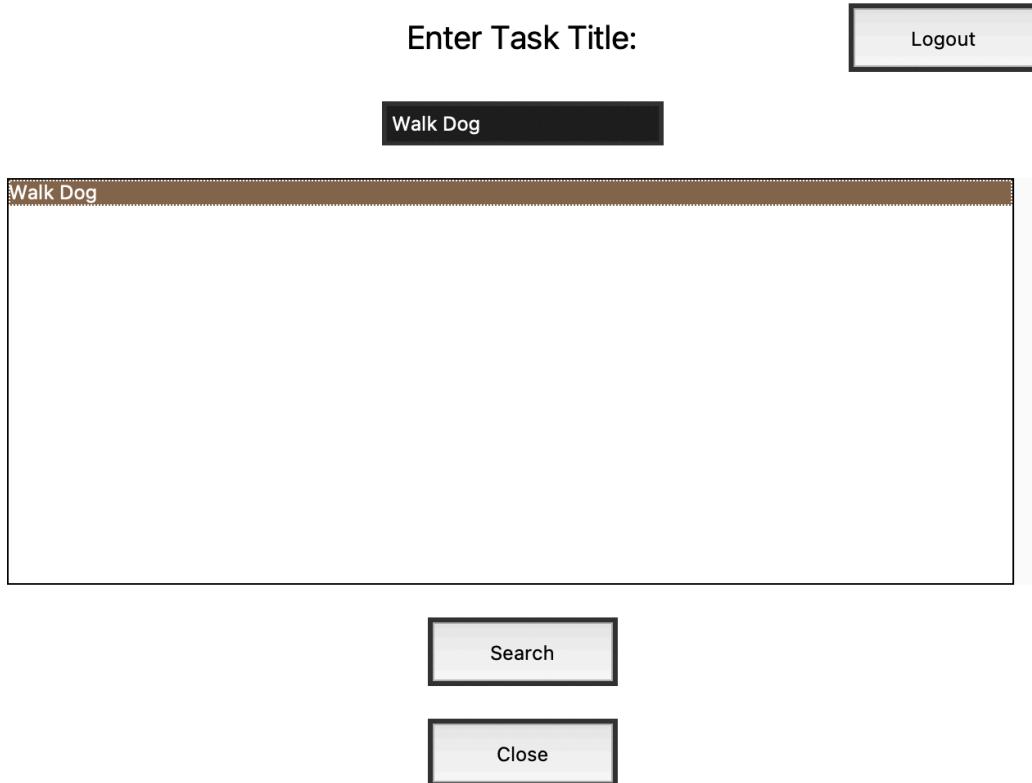


Figure 26: Improved Search Task Page

```
def search_tasks(self):
    # Create the main Search Task window
    search_window = tk.Toplevel(self)
    search_window.title("Search Tasks")
    search_window.geometry("650x500")
    search_window.configure(bg="white")

    # Logout function: Fully log out the user
    def logout():
        self.master.current_user = None # Clear the logged-in user
        search_window.destroy()
        messagebox.showinfo(title: "Logout", message: "You have been logged out.") # notifies the user
        self.master.show_page(WelcomePage) # Redirect to Welcome Page

    # Add the Logout Button
    tk.Button(
        search_window,
        text="Logout",
        command=logout,
        font=("Montserrat", 12),
        bg="white",
        fg="black",
        width=10,
        height=2,
    ).place(relx=1.0, rely=0.0, anchor="ne", x=-10, y=10)

    # Label and Entry for Search Input
    tk.Label(
        search_window,
        text="Enter Task Title:",
        bg="white",
        fg="black",
        font=("Montserrat", 20),
    ).pack(pady=15)
    search_entry = tk.Entry(search_window, font=("Montserrat", 12))
    search_entry.pack(pady=10)

    # Frame for Task Listbox and Scrollbar
    frame = tk.Frame(search_window, bg="white")
    frame.pack(fill=tk.BOTH, expand=True, padx=10, pady=10)
```

Figure 26.1: Improved Search Task Page Code

```
# Scrollbar and listbox
list_scrollbar = tk.Scrollbar(frame, orient=tk.VERTICAL)
task_listbox = tk.Listbox(
    frame,
    yscrollcommand=list_scrollbar.set,
    font=("Montserrat", 12),
    bg="white",
    fg="black",
)
list_scrollbar.config(command=task_listbox.yview)
list_scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
task_listbox.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)

# Function to Perform the Search
def perform_search():
    search_term = search_entry.get().strip().lower() # Get the search term
    task_listbox.delete(first: 0, tk.END) # Clear previous results

    username = self.master.current_user
    if username in users:
        results = [task for task in users[username]["tasks"] if search_term in task["title"].lower()]
        if results:
            for task in results:
                task_listbox.insert(tk.END, *elements: task["title"].strip())
        else:
            messagebox.showinfo(title: "No Results", message: "No tasks match your search.")
    else:
        messagebox.showerror(title: "Error", message: "User not found.")

# Button to Trigger Search
tk.Button(
    search_window,
    text="Search",
    command=perform_search,
    font=("Montserrat", 12),
    bg="white",
    fg="black",
    width=10,
    height=2,
).pack(pady=10)
```

Figure 26.2: Improved Search Task Page Code (Cont.)

```

# Close Button
tk.Button(
    search_window,
    text="Close",
    command=search_window.destroy,
    font=("Montserrat", 12),
    bg="white",
    fg="black",
    width=10,
    height=2,
).pack(pady=10)

# Function to Show Task Details
def show_task_details(event):
    selection = task_listbox.curselection()
    if not selection:
        return
    selected_task_title = task_listbox.get(selection[0]).strip().lower()
    print(f"Debug: Selected task title: {selected_task_title}")

    username = self.master.current_user
    if not username:
        messagebox.showerror(title="Error", message="No user logged in")
        return

    task_details = next(
        (task for task in users[username]["tasks"] if task["title"].strip().lower() == selected_task_title),
        None
    )
    print(f"Retrieved task details: {task_details}")

    if task_details:
        detail_window = tk.Toplevel(self)
        detail_window.title("Task Details")
        detail_window.geometry("400x300")
        tk.Label(detail_window, text=f"Title: {task_details['title']}", font=("Montserrat", 12)).pack(pady=5)
        tk.Label(detail_window, text=f"Time: {task_details['time']}", font=("Montserrat", 12)).pack(pady=5)
        tk.Label(detail_window, text=f"Duration: {task_details['duration']}", font=("Montserrat", 12)).pack(
            pady=5
        )

        tk.Label(detail_window, text=f"Description: {task_details['description']}",
                font=("Montserrat", 12)).pack(pady=5)
        tk.Button(detail_window, text="Close", command=detail_window.destroy, font=("Montserrat", 12)).pack(
            pady=10
        )
    else:
        messagebox.showerror(title="Error", message="Task not found.")

# Bind Listbox Selection to Show Task Details
task_listbox.bind("<>ListboxSelect>>", show_task_details)

```

Figure 26.3: Improved Search Task Page Code (Cont. (2))

Calendar:

The calendar page was improved by adding a logout button which brings the user back to the starting screen and creating a more uniform color scheme.

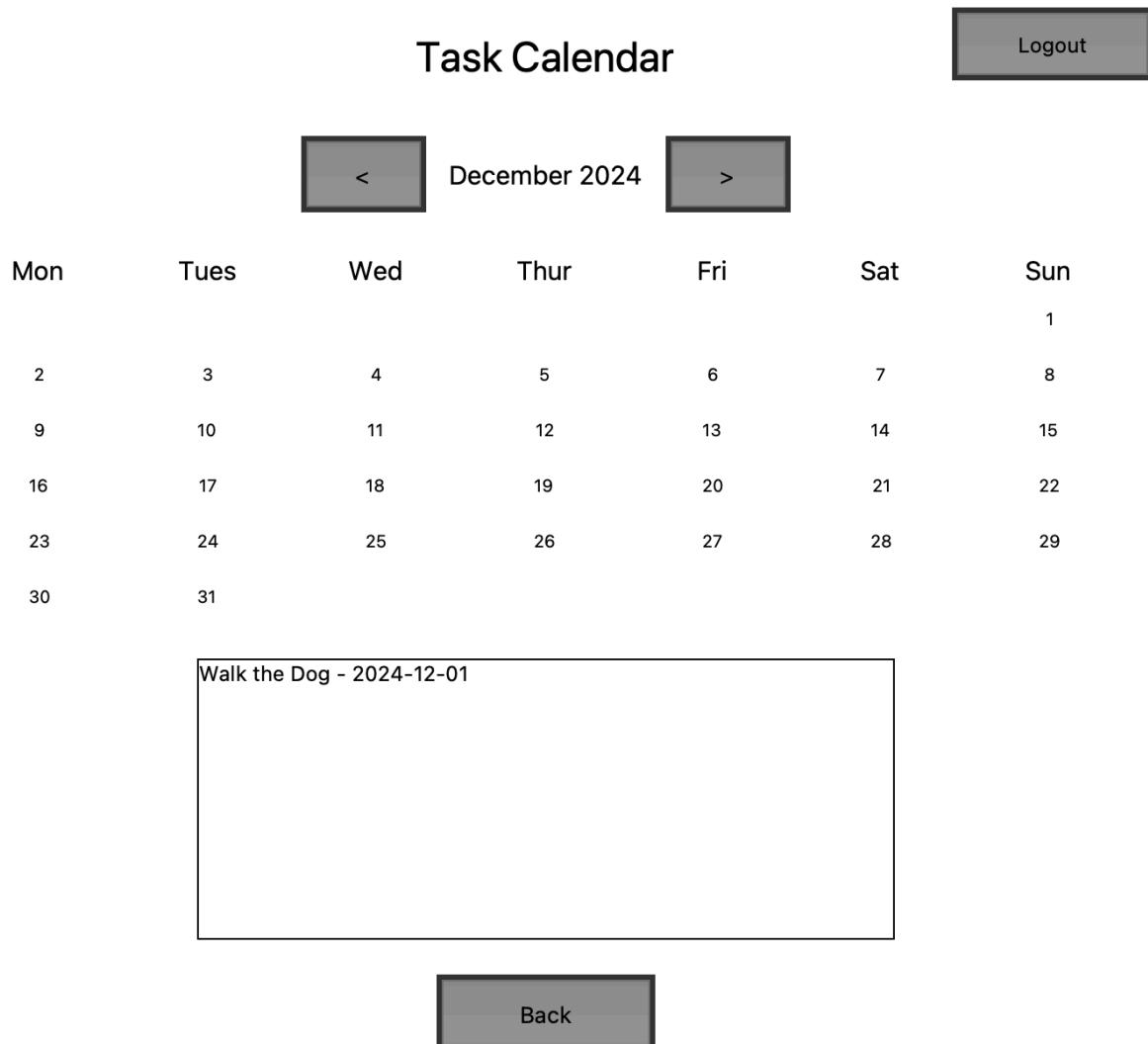
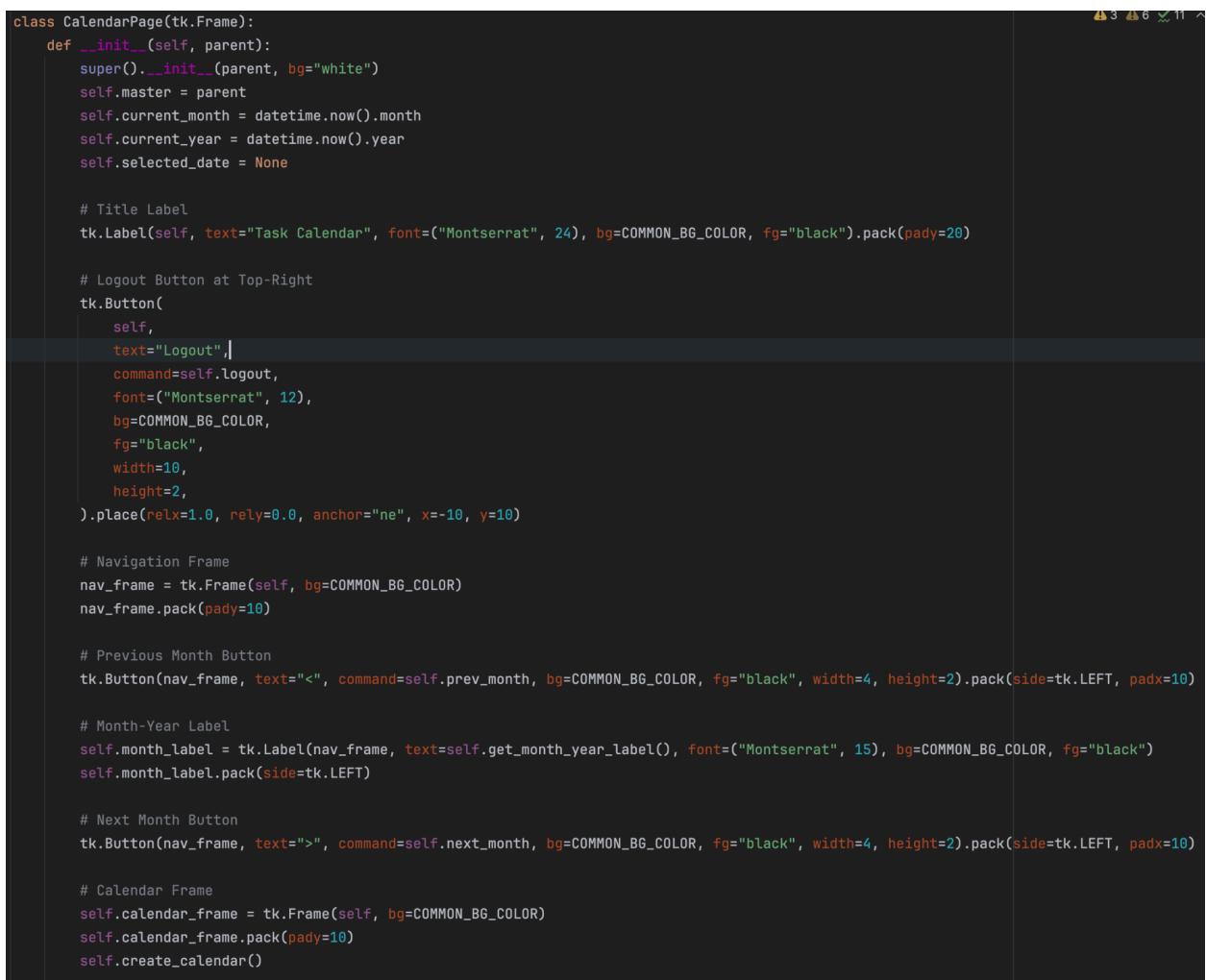


Figure 27: Improved Calendar Page



The image shows a screenshot of a Python code editor with a dark theme. The code is contained within a class named `CalendarPage` that inherits from `tk.Frame`. The code includes methods for initializing the frame, creating a title label, adding a logout button to the top right, setting up a navigation frame with month navigation buttons, and creating a calendar frame. The code uses the `Montserrat` font and standard Tkinter styling.

```
class CalendarPage(tk.Frame):
    def __init__(self, parent):
        super().__init__(parent, bg="white")
        self.master = parent
        self.current_month = datetime.now().month
        self.current_year = datetime.now().year
        self.selected_date = None

        # Title Label
        tk.Label(self, text="Task Calendar", font=("Montserrat", 24), bg=COMMON_BG_COLOR, fg="black").pack(pady=20)

        # Logout Button at Top-Right
        tk.Button(
            self,
            text="Logout",
            command=self.logout,
            font=("Montserrat", 12),
            bg=COMMON_BG_COLOR,
            fg="black",
            width=10,
            height=2,
        ).place(relx=1.0, rely=0.0, anchor="ne", x=-10, y=10)

        # Navigation Frame
        nav_frame = tk.Frame(self, bg=COMMON_BG_COLOR)
        nav_frame.pack(pady=10)

        # Previous Month Button
        tk.Button(nav_frame, text("< ", command=self.prev_month, bg=COMMON_BG_COLOR, fg="black", width=4, height=2).pack(side=tk.LEFT, padx=10)

        # Month-Year Label
        self.month_label = tk.Label(nav_frame, text=self.get_month_year_label(), font=("Montserrat", 15), bg=COMMON_BG_COLOR, fg="black")
        self.month_label.pack(side=tk.LEFT)

        # Next Month Button
        tk.Button(nav_frame, text("> ", command=self.next_month, bg=COMMON_BG_COLOR, fg="black", width=4, height=2).pack(side=tk.LEFT, padx=10)

        # Calendar Frame
        self.calendar_frame = tk.Frame(self, bg=COMMON_BG_COLOR)
        self.calendar_frame.pack(pady=10)
        self.create_calendar()
```

Figure 27.1: Improved Calendar Page Code

```
# Task Listbox
self.task_listbox = tk.Listbox(self, height=10, width=50, bg="white", fg="black", font=("Montserrat", 12))
self.task_listbox.pack(pady=10)

# Back Button
tk.Button(
    self,
    text="Back",
    command=lambda: parent.show_page(TaskPage),
    bg=COMMON_BG_COLOR,
    fg="black",
    width=10,
    height=2,
).pack(pady=10)

def logout(self):
    self.master.current_user = None
    messagebox.showinfo(title: "Logout", message: "You have been logged out.")
    self.master.show_page(WelcomePage)

def logout(self):
    # Logout Logic
    self.master.current_user = None
    messagebox.showinfo(title: "Logout", message: "You have been logged out.")
    self.master.show_page(WelcomePage)

def create_calendar(self):
    # Clear Previous Calendar
    for widget in self.calendar_frame.winfo_children():
        widget.destroy()

    # Style for Calendar Buttons
    style = ttk.Style()
    style.configure(
        style: "Calendar.Rounded.TButton",
        background=COMMON_BG_COLOR,
        foreground="black",
        font=("Montserrat", 10),
```

Figure 27.2: Improved Calendar Page Code (Cont.)

```

        borderwidth=1,
        relief="flat",
        padding=5
    )
    style.map(
        style: "Calendar.Rounded.TButton",
        background=[("active", "#d3d3d3")],
        relief=[("pressed", "groove")]
    )

days_in_month = calendar.monthrange(self.current_year, self.current_month)[1]
first_weekday = calendar.monthrange(self.current_year, self.current_month)[0]

row = 1
col = first_weekday

# Weekday Labels
for day in ["Mon", "Tues", "Wed", "Thur", "Fri", "Sat", "Sun"]:
    tk.Label(self.calendar_frame, text=day, font=("Montserrat", 15), width=4, bg=COMMON_BG_COLOR, fg="black").grid(
        row=0,
        column=[ "Mon", "Tues", "Wed", "Thur", "Fri", "Sat", "Sun"].index(day)
    )

# Calendar Buttons
for day in range(1, days_in_month + 1):
    day_button = ttk.Button(
        self.calendar_frame,
        text=str(day),
        style="Calendar.Rounded.TButton",
        command=lambda d=day: self.select_date(d),
    )
    day_button.grid(row=row, column=col, padx=2, pady=2)
    col += 1
    if col > 6:
        col = 0
        row += 1

def get_month_year_label(self):
    return f"{calendar.month_name[self.current_month]} {self.current_year}"

```

Figure 27.3: Improved Calendar Page Code (Cont. (2))

```

def prev_month(self):
    if self.current_month == 1:
        self.current_month = 12
        self.current_year -= 1
    else:
        self.current_month -= 1
    self.month_label.config(text=self.get_month_year_label())
    self.create_calendar()

def next_month(self):
    if self.current_month == 12:
        self.current_month = 1
        self.current_year += 1
    else:
        self.current_month += 1
    self.month_label.config(text=self.get_month_year_label())
    self.create_calendar()

def select_date(self, day):
    self.selected_date = f"{self.current_year}-{self.current_month:02d}-{day:02d}"
    print(f"Date selected: {self.selected_date}")
    self.view_tasks()

def view_tasks(self):
    self.task_listbox.delete(first=0, tk.END)
    print(f"Selected date: {self.selected_date}")

username = self.master.current_user
if not username:
    messagebox.showerror(title="Error", message="No user logged in")
    return

selected_date = self.selected_date
if not selected_date:
    messagebox.showinfo(title="Info", message="Select a date to view tasks")
    return

tasks = users.get(username, {}).get("tasks", [])
tasks_on_date = [task for task in tasks if task.get("time", "") == selected_date]

if tasks_on_date:
    for task in tasks_on_date:
        self.task_listbox.insert(tk.END, *elements=f"{task['title']} - {task['time']}")
else:
    self.task_listbox.insert(tk.END, *elements="No tasks on this date")

```

Figure 27.4: Improved Calendar Page Code (Cont. (3))

Visually Reported Stored Information:

Admin Login

Username

Password

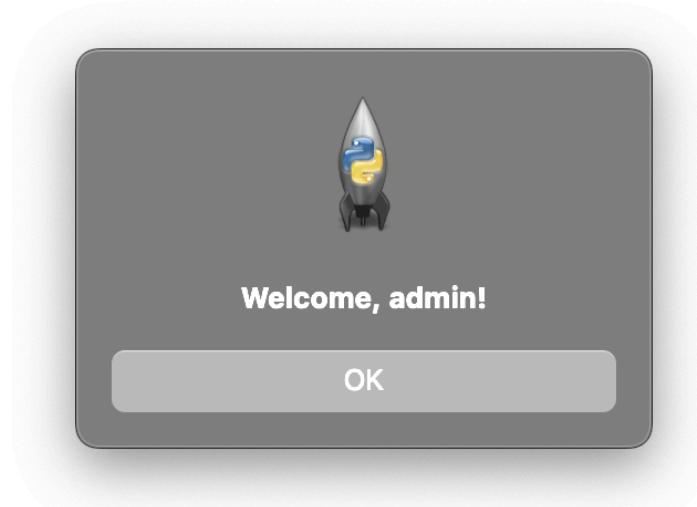


Figure 28: Logging In

Task Title	Time	Duration	Description
Walk the Dog	2024-12-01	2 Hours	Walk Rocky
Feed Cat	2024-12-01	10 mins	Feed Lilith

[Logout](#)
 [Close](#)
 [Edit Selected Task](#)

Figure 29: Edit Task Page

Task Title	Time	Duration	Description
Walk the Dog	2024-12-01	2 Hours	Walk Rocky
Feed Cat	2024-12-01	10 mins	Feed Lilith

Edit Task

Task Title:

Time:

Duration:

Description:

[Save Changes](#)
 [Cancel](#)

Figure 30: Edit Task Subpage

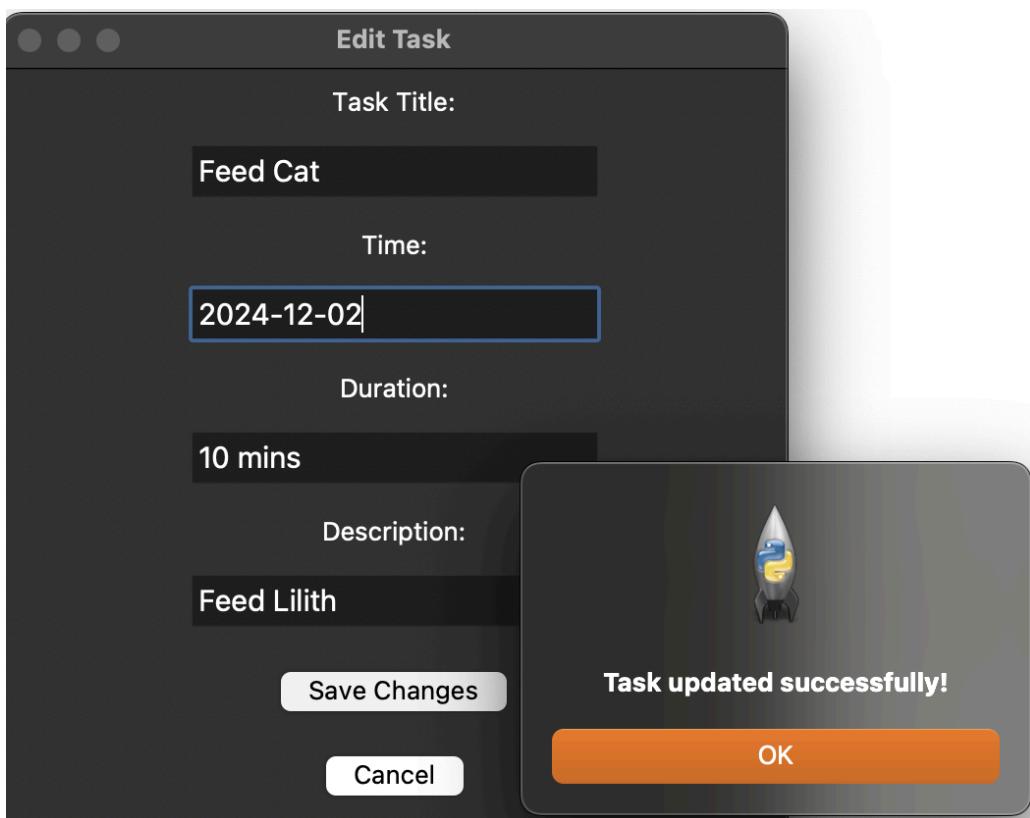


Figure 31: Successfully Edited Task Indicator

Task Title	Time	Duration	Description
Walk the Dog	2024-12-01	2 Hours	Walk Rocky
Feed Cat	2024-12-02	10 mins	Feed Lilith

Logout Close

Edit Selected Task

Figure 32: Successfully Updated Task

Welcome to Task Management System

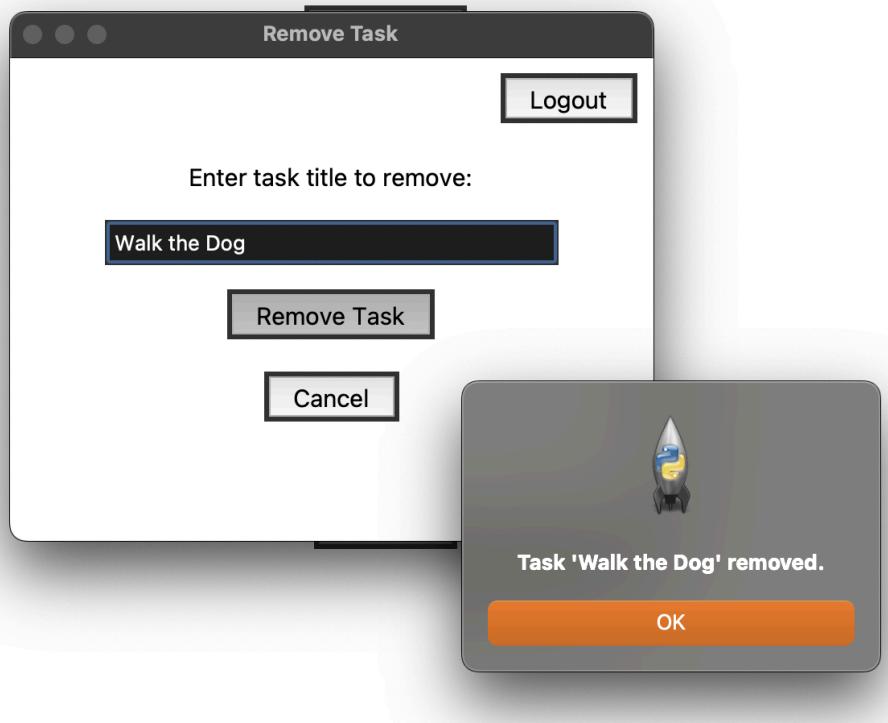


Figure 33: Remove Task Page and Indicator

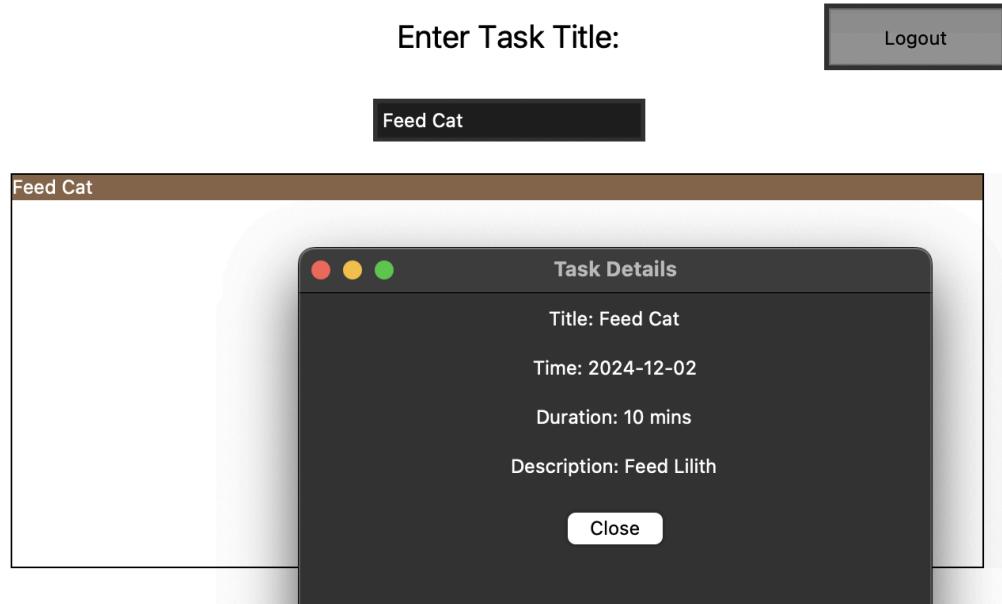


Figure 34: Search Task Page with Details Menu

Task Calendar

[Logout](#)

[!\[\]\(84ce5e30e8643c161862fb10c35964d2_img.jpg\) <](#) December 2024 [!\[\]\(89a19f20c2a1412eb6d886c03be3a972_img.jpg\) >](#)

Mon	Tues	Wed	Thur	Fri	Sat	Sun
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30	31					

Feed Cat - 2024-12-02

[!\[\]\(f59842166a12d90a8a0108ac365de408_img.jpg\) Back](#)

Figure 35: Calendar Page

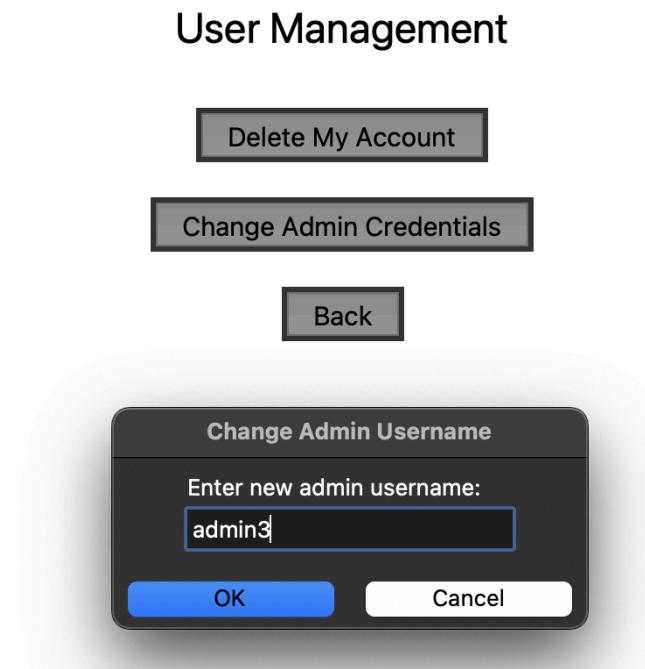


Figure 36: User Management Page and New Username Field

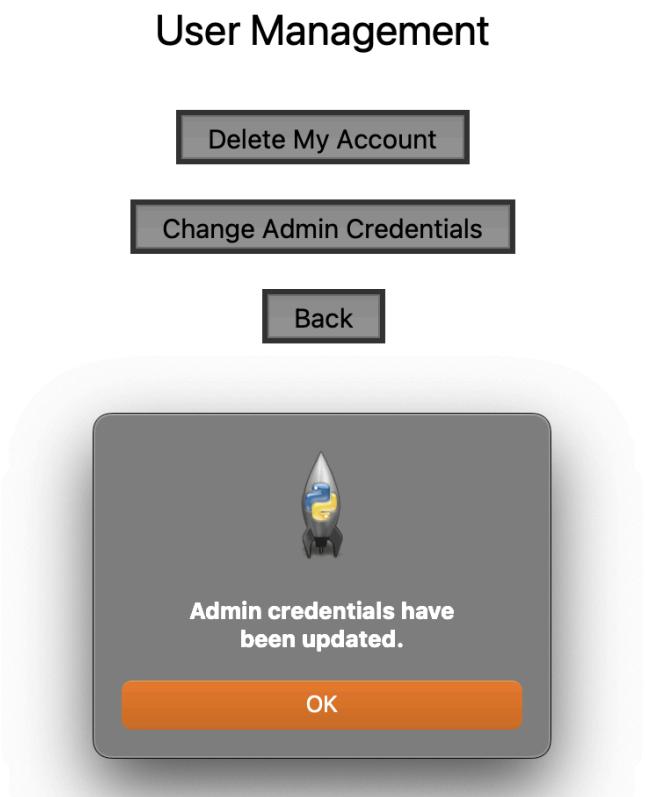


Figure 37: Successfully Updated Task Indicator

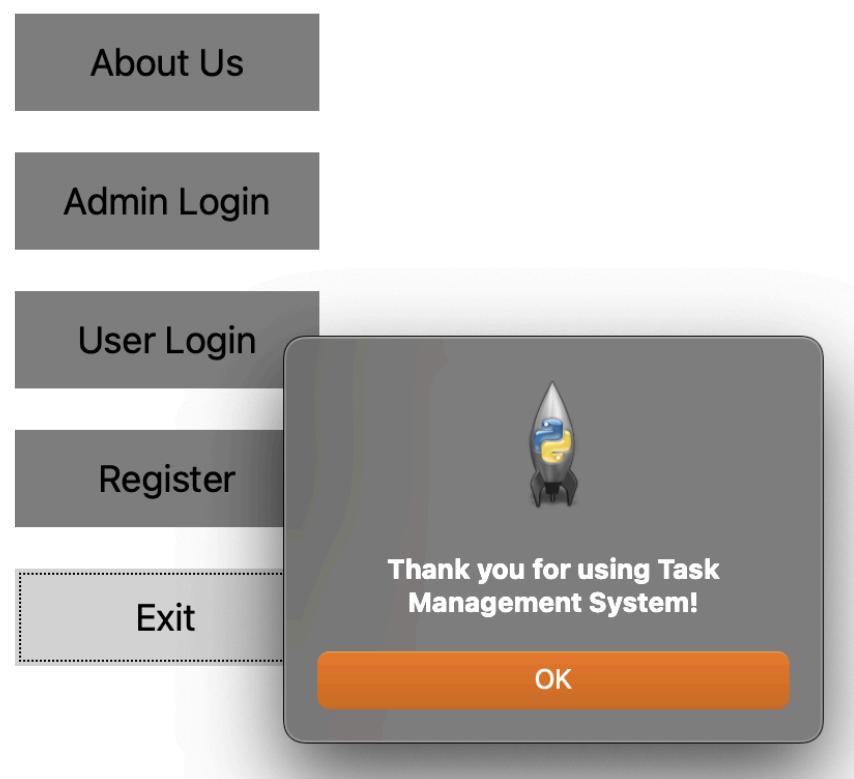


Figure 38: Closing Message

Description:

As previously stated, our project consists of a task management system, where users can login to the software to add, edit, search for, and delete tasks. In order to save relevant data to the system, we referenced a local file directory to access and store information to .txt files for retrieval. In this case, we named the files “users.txt,” and “tms_tasks.txt” to store user login credentials, and user input tasks respectively. As demonstrated above, this implementation works seamlessly to...

- Store new usernames and passwords

- Store new tasks and relevant task details
- Make persistent edits to task details
- Remove tasks
- Represent data in numerous formats (calendars, lists, search menus)

These changes can also be observed directly within the respective local .txt files.

References

- [1]<https://lucidchart.com/blog/flowchart-templates>
- [2]https://github.com/RezaSadeghiWSU/CMPT-120L-112_Task-Management-System_Maddie-Gaby.git