

# Task Management System

Introduction to Programming

CMPT 120L

TLMB

Marist College

School of Computer Science and Mathematics



Submitted to:

Dr. Reza Sadeghi

Fall 2023

# Project Progress Report 1 of Task Management System

## Team Name

Name of the team

TLMB

## Team Members

- |                  |                                     |
|------------------|-------------------------------------|
| 1. Lance Perdue  | Lance.Perdue1@marist.edu (Head)     |
| 2. Timothy Ford  | Timothy.Ford1@Marist.edu (Member)   |
| 3. Matthew Brana | Matthew.Brana1@Marist.edu(Member)   |
| 4.Blaine Chesnut | Blaine.Chesnut1@Marist.edu (Member) |

## Description of Team Members

### 1. Lance Perdue

My name is Lance Perdue, I have approximately a year of java experience through a dual enrollment program my senior year of high school. I have worked with these team members throughout the entire semester and we have been successful in all of our projects thus far. The more we work together, the more efficient and productive our communication has become. I feel that working with these members will continue to be successful in this final project and our experience working together will give our group an advantage. I was chosen to be our team's leader primarily for my experience using GitHub in the past, while my knowledge is limited, it still is useful in allowing our group to work efficiently using this new tool.

## 2. Timothy Ford

My name is Tim Ford and I am a freshman at Marist College pursuing a Bachelor of Science Degree in Computer Science. I have prior experience in both Python and more recently Java through AP Computer Science A during my high school education. I chose to stick with my team members that I have been with through the entire semester as I believe we have worked well together in the past and will only continue to grow as a group throughout this project. We chose our group head to be Lance Perdue because of his experience with Github in the past and also his willingness to step up and lead the group.

## 3. Matthew Brana

My name is Matthew Brana. I am a freshman at Marist and my major is Computer Science. I have an associate's degree from Brookdale Community College in New Jersey in Computer Science. I chose to work with my team members because we have worked together all year and our group assignments have gone smoothly. We decided on Lance Perdue to be our leader because he has the most knowledge of GitHub and set up the repository for our final group assignment.

## 4. Blaine Chesnut

My name is Blaine Chesnut and I am from Morristown, New Jersey. I am a freshman at Marist College and majoring in Data Science and Analytics although I do not truly have any background in working with computers outside of using them for school assignments. I have always had a love for analytics stemming from my love of sports. I wanted to work with this group since we have completed all of the prior group projects together. We work extremely well together and always succeed in our work. We chose Lance as the team head since he is very intelligent and a great leader.

## Table of Contents

1. Table of Figures.....	5
2. Project Objective.....	6
3. Github Repository Link.....	7
4. List of Used Packages.....	7
5. Virtual Environment.....	8
6. User Experience Design.....	8
7. User Interface Design.....	9
8. Link to UX and UI Design program.....	31

## Table of Figures

### Figure 3:

1. User Experience (UX) Flowchart.....	8
--	---

### Figure 4:

1. Login Window.....	9
2. Login failed popup.....	11
3. Admin success popup.....	11
4. Admin Window.....	12
5. Add User Function.....	14
6. Remove User Function.....	16
7. Update Admin Function.....	18
8. User success popup.....	20
9. Main Menu Window.....	20
10. Add Task Function.....	22
11. Remove Task Function.....	24
12. Edit Task Function.....	26
13. Search Task Function.....	29

## **Project Objective**

A task management system (TMS) displays a calendar for the desired week, month, or year. Also, TMS organizes personal tasks of different users on a specific day. The users should be able to see their individual calendar data and update them. Your TMS will store the data of different user types in distinct CSV files. This system should at least support the following items.

1.Admin user is capable of:

- a.Having admin user and password for login (a string of at least 8 characters
- b.Changing the admin user and admin password
- c.Adding a normal user to TMS by creating a new username, password, and corresponding recorded data

2.Each user should be able to:

- a.Add a task to TMS the task contains: title, time, duration, and description
- b.Remove a task
- c.Edit a task's details
- d.Search through TMS based on time, title, or duration and list the results on the screen. For instance, it should be able to list all scheduled works for one day.

3.TMS should be a user-friendly software, such that:

- a. it shows a welcome page and provides a menu of all functions to the user in all pages
- b. It illustrates the reports in a tabular form. For instance, it displays a well-organized calendar of every month, or year.

c. It shows a warning if a user tries to input contact information with a name that exists in the history.

d. TMS should provide an exit function and thank the user for using the software.

4. Optional: TMS should protect the user information, such that:

a. TMS passwords and the recorded information should be ciphered. In the simplest case, you can use the Caesar cipher methodology. The easiest way to understand the Caesar cipher is to think of cycling the position of the letters. in a caesar cipher with a shift of 3, a becomes d, b becomes e, c becomes f, etc. when reaching the end of the alphabet it cycles around, so x becomes a, y becomes b, and z becomes c.

**GitHub Repository**

<https://github.com/Taco956/CMPT-120-Final-Project>

**Used Packages:**

OS, Tkinter, CSV, Pandas

## Used Virtual Environment:

```
(finalProject) C:\Users\jpper>python -m pip install pandas
Collecting pandas
  Obtaining dependency information for pandas from https://files.pythonhosted.org/packages/97/d8/dc2f6bff06a799a5603c414
  afc6de39c6351fe34892d50b6a077df3be6ac/pandas-2.1.3-cp311-cp311-win_amd64.whl.metadata
  Downloading pandas-2.1.3-cp311-cp311-win_amd64.whl.metadata (18 kB)
Collecting numpy<2,>=1.23.2 (from pandas)
  Obtaining dependency information for numpy<2,>=1.23.2 from https://files.pythonhosted.org/packages/da/3c/3ff05c2855eee
  52588f489a4e607e4a61699a0742aa03ccf641c77f9eb0a/numpy-1.26.2-cp311-cp311-win_amd64.whl.metadata
  Downloading numpy-1.26.2-cp311-cp311-win_amd64.whl.metadata (61 kB)
    61.2/61.2 kB 1.6 MB/s eta 0:00:00
Collecting python-dateutil>=2.8.2 (from pandas)
  Using cached python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
Collecting pytz>=2020.1 (from pandas)
  Obtaining dependency information for pytz>=2020.1 from https://files.pythonhosted.org/packages/32/4d/aaf7eff5deb402fd9
  a24a1449a8119f00d74ae9c2efa79f8ef9994261fc2/pytz-2023.3.post1-py2.py3-none-any.whl.metadata
  Downloading pytz-2023.3.post1-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.1 (from pandas)
  Downloading tzdata-2023.3-py2.py3-none-any.whl (341 kB)
    341.8/341.8 kB 4.3 MB/s eta 0:00:00
Collecting six>=1.5 (from python-dateutil>=2.8.2->pandas)
  Using cached six-1.16.0-py2.py3-none-any.whl (11 kB)
Downloading pandas-2.1.3-cp311-cp311-win_amd64.whl (10.6 MB)
    10.6/10.6 MB 4.8 MB/s eta 0:00:00
Downloading numpy-1.26.2-cp311-cp311-win_amd64.whl (15.8 MB)
    15.8/15.8 MB 5.2 MB/s eta 0:00:00
Downloading pytz-2023.3.post1-py2.py3-none-any.whl (502 kB)
    502.5/502.5 kB 6.3 MB/s eta 0:00:00
Installing collected packages: pytz, tzdata, six, numpy, python-dateutil, pandas
```

Our group's virtual environment only contains one external package: pandas. This package is meant to make working in data structures more intuitive and flexible. This package allowed us to manipulate the data within our numerous csv files much easier than with the python tools alone.



## User Experience (UX) Design

When designing the user experience, we decided to start the user on a Login window. Upon entering their information, the system will determine if they have a valid login and direct them to a success popup, if they have a valid login as an admin and directed to an admin success popup, or are using an invalid login which will just route them back to the login screen. A successful admin login will bring the user to an admin window where users can be added or removed and the admin user can be updated, or the main menu can be accessed. A successful regular login will direct the user to the main menu. From the main menu, they are given the option to exit the system entirely or to access one of the four functions of the TMS. Each of these functions, Add, Remove, Edit, and Search, all allow the user to use their specific function, to return to the main menu, or to exit the system in its entirety.

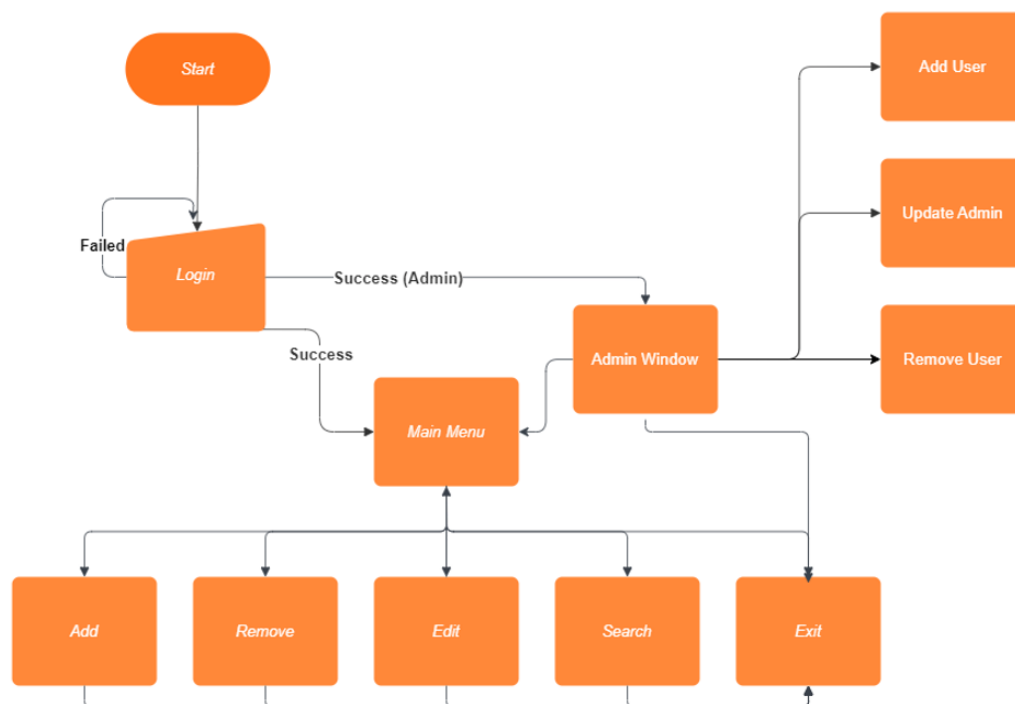
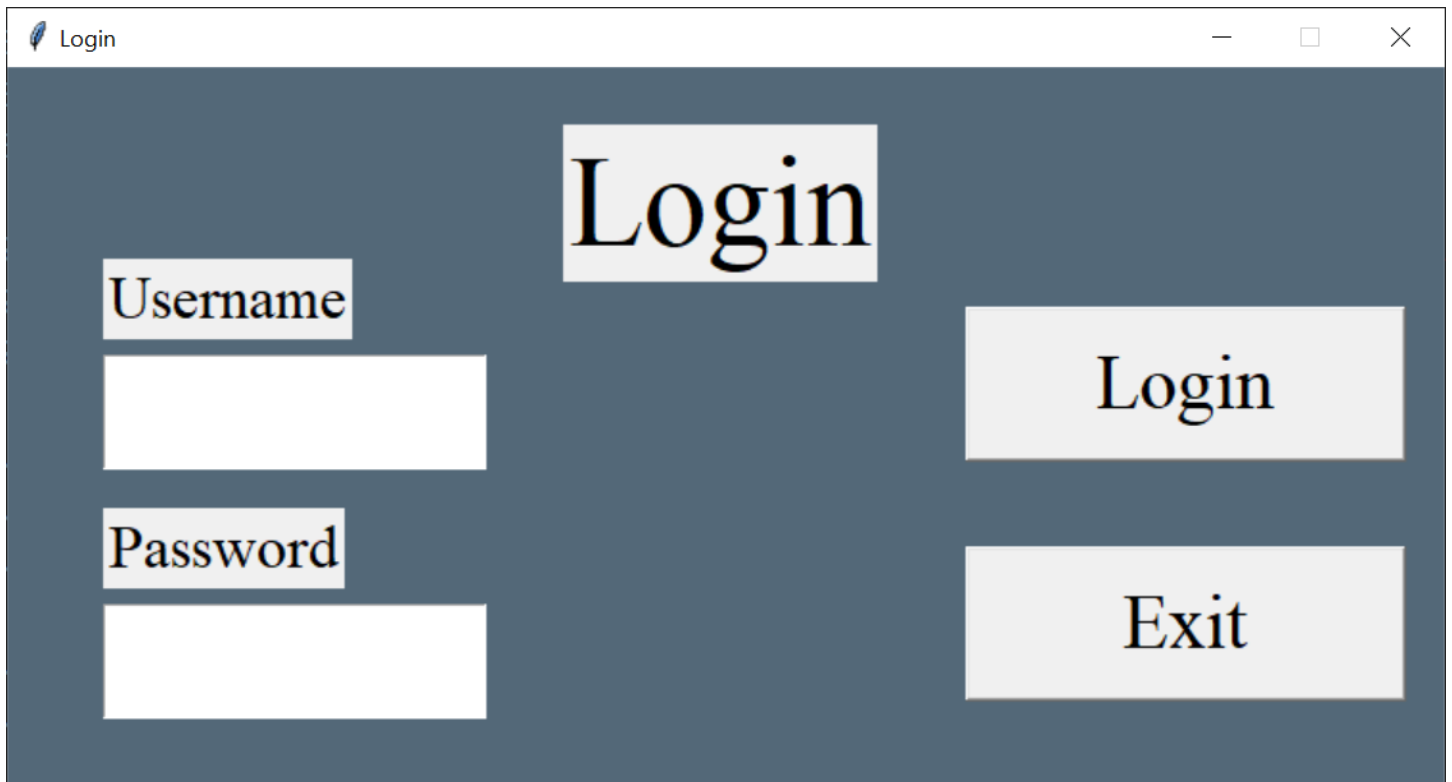


Figure 3.1: Flowchart showing the design of the User's intended experience

## **User Interface (UI) Design**

Figure 4.1: Login Window

Figure 4.1 is the first window that will be shown to the user upon running the program. This login window has two entry boxes, one for the username and one for the password, and two buttons, one to attempt a login and one to exit the program.



The image shows a graphical user interface for a login window. The window is titled "Login" and has a dark blue background. In the center, the word "Login" is displayed in a large, white, serif font. On the left side, there are two white rectangular input boxes. The top box is labeled "Username" and the bottom box is labeled "Password", both in a white, serif font. On the right side, there are two white rectangular buttons. The top button is labeled "Login" and the bottom button is labeled "Exit", both in a black, serif font. The window has a standard title bar with a feather icon, the title "Login", and minimize, maximize, and close buttons.

```
#Login GUI
loginwindow = tk.Tk()
loginwindow.title("Login")
loginwindow.geometry("750x375")
loginwindow.resizable(width=False,height=False)

Body = tk.Frame(loginwindow,bg='#536878',height=375,width=750)
Body.grid(row=0,column=0)

lblUsername = tk.Label(Body,text="Username")
lblUsername.config(font=("Times New Roman",24))
lblUsername.place(x=50,y=100)

entUsername = tk.Entry(Body)
entUsername.config(font=("Times New Roman", 20))
entUsername.place(x=50,y=150,width=200,height=60)

lblPassword = tk.Label(Body,text="Password")
lblPassword.config(font=("Times New Roman",24))
lblPassword.place(x=50,y=230)

entPassword = tk.Entry(Body)
entPassword.config(font=("Times New Roman", 20))
entPassword.place(x=50,y=280,width=200,height=60)

loginTitle = tk.Label(Body, text = "Login")
loginTitle.config(font=("Times New Roman",50))
loginTitle.place(x=290,y=30)

btnLogin = tk.Button(Body,text="Login",command=login,width=10)
btnLogin.place(x=500,y=125)
btnLogin.config(font=("Times New Roman", 30))

btnExit = tk.Button(Body,text="Exit",command=exit,width=10)
btnExit.place(x=500,y=250)
btnExit.config(font=("Times New Roman", 30))

loginwindow.mainloop()
```

This is the code used to first make the window and body of the login, then two entry boxes with labels above them. Then we made a login button and an exit button and titled the window itself. Finally we mainlooped the window to pack it all together and make sure it displays properly.

Figure 4.2: Login failed popup

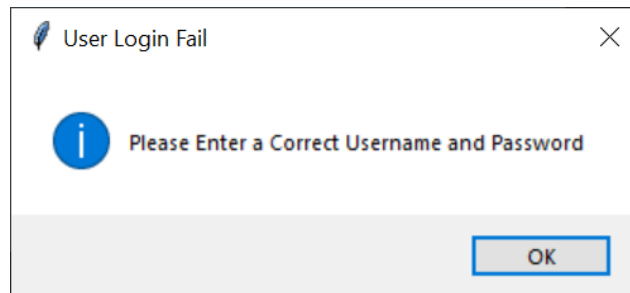


Figure 4.2 is a window that will quickly pop up whenever a user tries to login with unauthorized information, they will then just be directed back to the login window so they can try again.

```
messagebox.showinfo("User Login Fail","Please Enter a Correct Username and Password")
```

This line of code that runs if the user's login info cannot be validated is what flashes this popup box informing the user to use different information.

Figure 4.3: Admin success popup

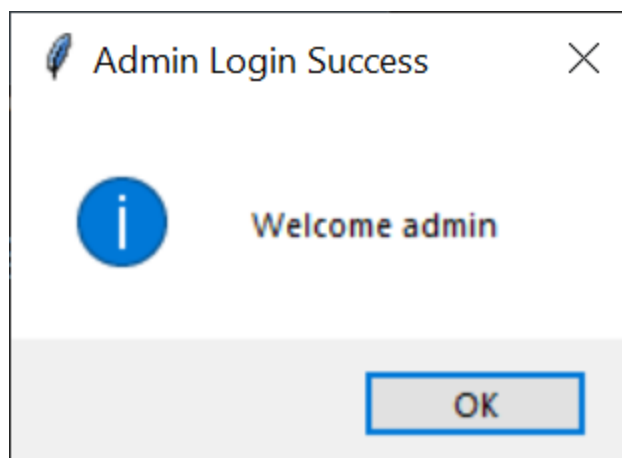


Figure 4.3 is the quick popup that will flash after the correct username and password are used for the admin login. This window will close on its own and then redirect the user to the admin window.

```
messagebox.showinfo("Admin Login Success",f"      Welcome {enteredUser}      ")
loginwindow.destroy()
adminMenu()
```

These lines first display figure 4.3 to show the user that they have logged in as an admin and uses their username in a formatted string to personally welcome them. The next line closes the login window since it is no longer needed and would clutter the user's screen. Finally, the admin window is called since it was a successful admin login

Figure 4.4: Admin Window that appears upon admin login

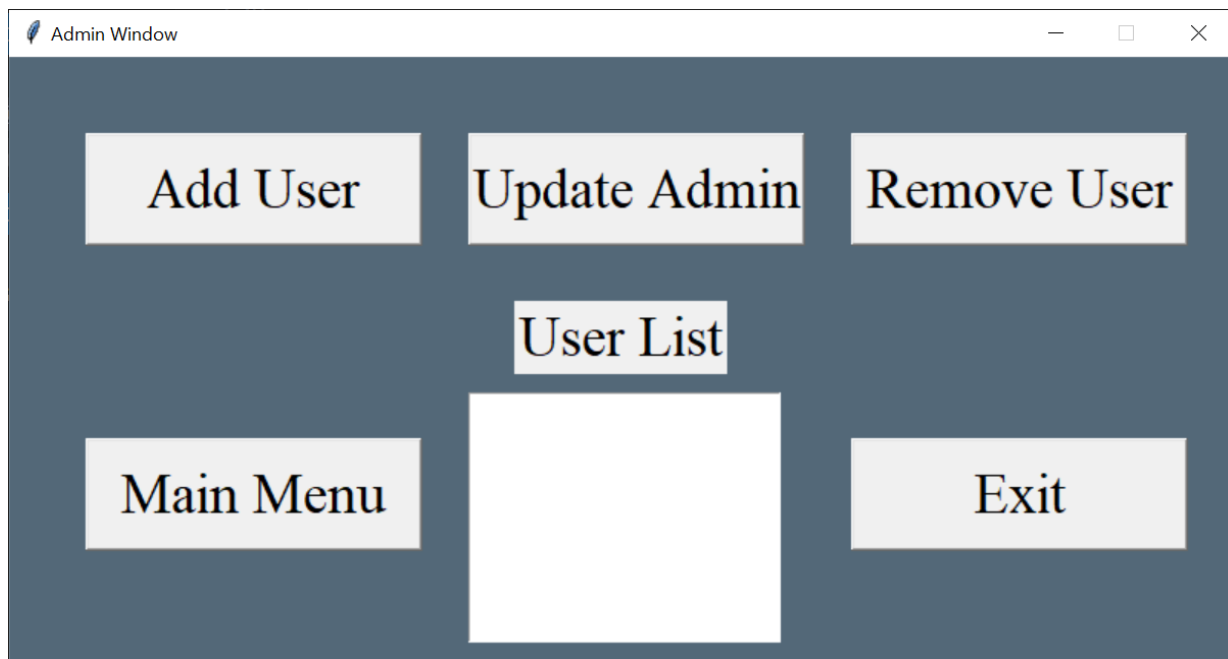


Figure 4.4 is the admin window that will appear after the popup redirects the user. It is made up of 5 buttons that allow the user to access the add user function, the update admin function, and the remove user function as well as to move to the main menu or to exit the program completely. There is also a text box that will show a current list of all the users that have access to the TMS.

```
#Admin Window
def adminMenu():
    adminWindow = tk.Tk()
    adminWindow.title("Admin Window")
    adminWindow.geometry("800x400")
    adminWindow.resizable(width=False,height=False)

    adminBody = tk.Frame(adminWindow,bg='#536878',height=400,width=800)
    adminBody.grid(row=0,column=0)

    btnExit = tk.Button(adminBody,text="Exit",command=exit,width=10)
    btnExit.place(x=550,y=250)
    btnExit.config(font=("Times New Roman", 28))

    btnRemoveUser = tk.Button(adminBody,text="Remove User",command=removeUserWin,width=10)
    btnRemoveUser.place(x=550,y=50)
    btnRemoveUser.config(font=("Times New Roman", 28))

    btnUpdateAdmin = tk.Button(adminBody,text="Update Admin",command=updateAdminWin,width=10)
    btnUpdateAdmin.place(x=300,y=50)
    btnUpdateAdmin.config(font=("Times New Roman", 28))

    btnAddUser = tk.Button(adminBody,text="Add User",command=addUserWin,width=10)
    btnAddUser.place(x=50,y=50)
    btnAddUser.config(font=("Times New Roman", 28))

    btnMainMenu = tk.Button(adminBody,text="Main Menu",command=lambda:[mainMenuWindow()],width=10)
    btnMainMenu.place(x=50,y=250)
    btnMainMenu.config(font=("Times New Roman", 28))

    lblUserList = tk.Label(adminBody, text="User List")
    lblUserList.place(x = 330, y =160)
    lblUserList.config(font=("Times New Roman", 28))

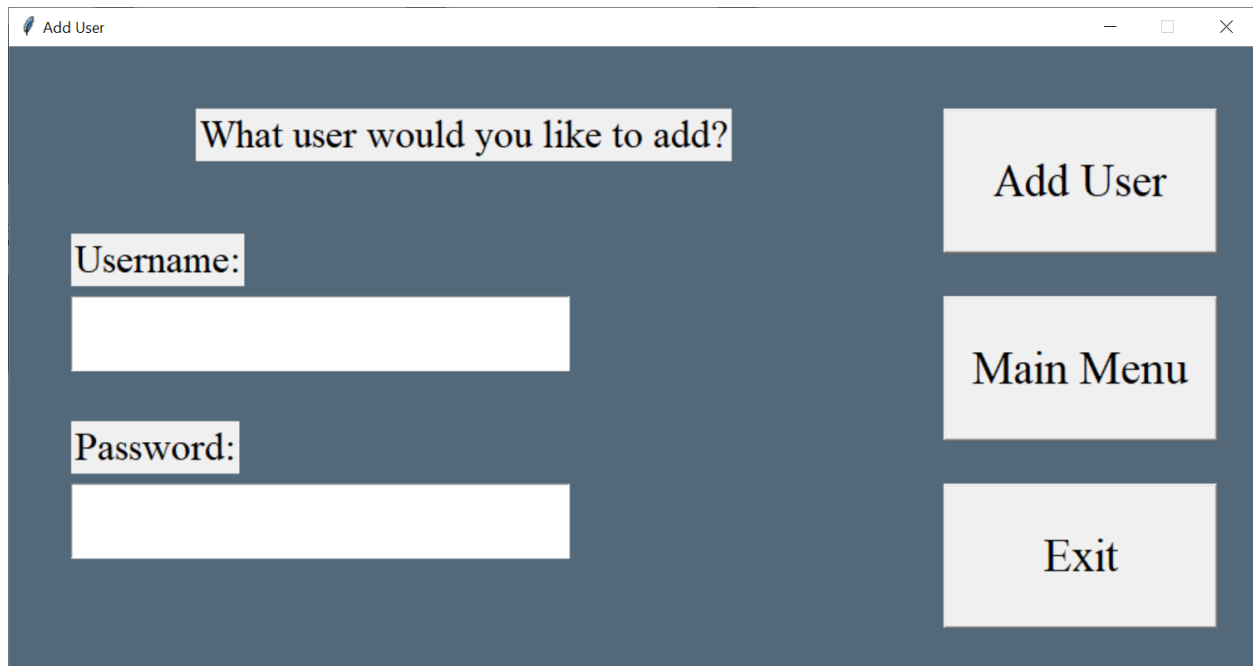
    txtUserList = tk.Text(adminBody, height=10, width=25)
    txtUserList.place(x=300, y = 220)

    adminWindow.mainloop()
```

This section of code starts off by creating a window with TKinter and making sure its restricted in size. After making the body, five buttons and a textbox that will display all of the current users in the system are placed in it to allow the user to

navigate the functions of the admin window, enter the main menu, or to close the program. Again, the code ends with a mainloop to ensure every widget is packed and properly displayed on the user's screen.

Figure 4.5: Add User function from admin window



The image shows a window titled "Add User" with a dark blue background. At the top, a light blue box contains the text "What user would you like to add?". Below this, there are two labels: "Username:" and "Password:", each followed by a white rectangular input field. To the right of the input fields, there are three light gray buttons stacked vertically, labeled "Add User", "Main Menu", and "Exit". The window has a standard title bar with a minimize button, a maximize button, and a close button.

Figure 4.5 is an extension of the previous admin window, once clicked the user will be directed to this window with two entries and three buttons. The admin can add a user by entering the requested username and password and can be fully completed by then pressing the add user button. Once completed a non-admin user can use this newly created information to successfully login and store their own data through the task management system. This window also includes a main menu function which will close this window and redirect the user to the main menu. Lastly, this window contains an exit app button to allow the user to seamlessly close the application entirely.



```
#Add User GUI
def addUserWin():
    addUserWindow = tk.Tk()
    addUserWindow.title("Add User")
    addUserWindow.geometry("1000x500")
    addUserWindow.resizable(width=False,height=False)

    addUserBody = tk.Frame(addUserWindow,bg='#536878',height=500,width=1000)
    addUserBody.grid(row=0,column=0)

    btnMainMenu = tk.Button(addUserBody,text="Admin Menu",command=exit,width=10, height=2)
    btnMainMenu.place(x=750,y=200)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(addUserBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=350)
    btnExit.config(font=("Times New Roman", 28))

    btnAddUser = tk.Button(addUserBody,text="Add User",command=addUser,width=10, height=2)
    btnAddUser.place(x=750,y=50)
    btnAddUser.config(font=("Times New Roman", 28))

    lblAddTitle = tk.Label(addUserBody,text="What user would you like to add?")
    lblAddTitle.config(font=("Times New Roman",24))
    lblAddTitle.place(x=150,y=50)

    lblUsername = tk.Label(addUserBody,text="Username:")
    lblUsername.config(font=("Times New Roman",24))
    lblUsername.place(x=50,y=150)

    global entUsername
    entUsername = tk.Entry(addUserBody)
    entUsername.config(font=("Times New Roman", 20))
    entUsername.place(x=50,y=200,width=400,height=60)

    lblPassword = tk.Label(addUserBody,text="Password:")
    lblPassword.config(font=("Times New Roman",24))
    lblPassword.place(x=50,y=300)

    global entPassword
    entPassword = tk.Entry(addUserBody)
    entPassword.config(font=("Times New Roman", 20))
    entPassword.place(x=50,y=350,width=400,height=60)

    addUserWindow.mainloop()
```

First the window and body are created, and once again the size is restricted. Three buttons are placed, allowing the admin to add a new user, return to the admin menu, or to exit the program. There are also two entry boxes which the admin can use to enter the new username and password of the user that they are adding to the

system. The window is then mainlooped in order to pack the widgets and display them properly when it is called.

Figure 4.6: Remove User function from admin window

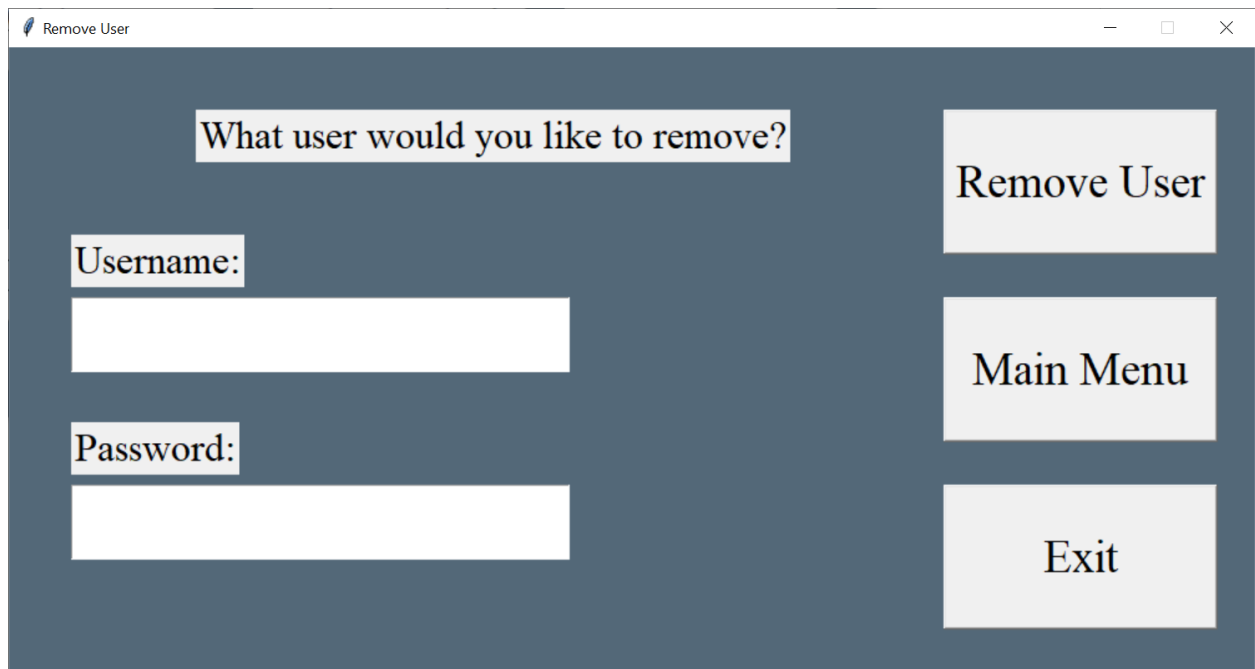


Figure 4.6 is another extension of the admin window which allows for the admin to remove any user other than the admin itself by entering their associated username in the provided entry. For this action to be completed the user will have to press the remove user button to finish the process, after which, that user will no longer be able to login and store information. This window also includes the same main menu and exit app button as the previous figure, allowing the user to navigate the application with little confusion or frustration.

```

#Remove User Below
def removeUserWin():
    removeUserWindow = tk.Tk()
    removeUserWindow.title("Remove User")
    removeUserWindow.geometry("1000x500")
    removeUserWindow.resizable(width=False,height=False)

    removeUserBody = tk.Frame(removeUserWindow,bg='#536878',height=500,width=1000)
    removeUserBody.grid(row=0,column=0)

    btnMainMenu = tk.Button(removeUserBody,text="Admin Menu",command=removeUserWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=200)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(removeUserBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=350)
    btnExit.config(font=("Times New Roman", 28))

    btnRemoveUser = tk.Button(removeUserBody,text="Remove User",command=removeUser,width=10, height=2)
    btnRemoveUser.place(x=750,y=50)
    btnRemoveUser.config(font=("Times New Roman", 28))

    lblAddTitle = tk.Label(removeUserBody,text="What user would you like to remove?")
    lblAddTitle.config(font=("Times New Roman",24))
    lblAddTitle.place(x=150,y=50)

    lblUsername = tk.Label(removeUserBody,text="Username:")
    lblUsername.config(font=("Times New Roman",24))
    lblUsername.place(x=50,y=150)

    global entUsername
    entUsername = tk.Entry(removeUserBody)
    entUsername.config(font=("Times New Roman", 20))
    entUsername.place(x=50,y=200,width=400,height=60)

    lblPassword = tk.Label(removeUserBody,text="Password:")
    lblPassword.config(font=("Times New Roman",24))
    lblPassword.place(x=50,y=300)

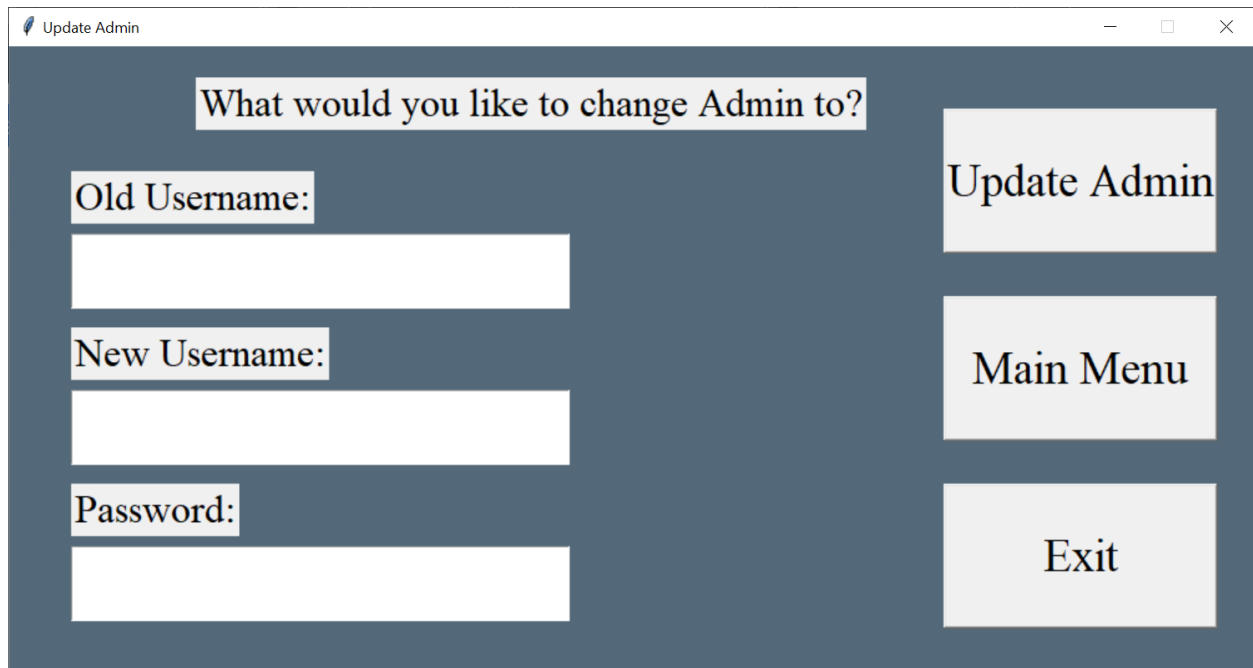
    global entPassword
    entPassword = tk.Entry(removeUserBody)
    entPassword.config(font=("Times New Roman", 20))
    entPassword.place(x=50,y=350,width=400,height=60)

    removeUserWindow.mainloop()

```

This is extremely similar to the add user design, starting with creating the window and body. Then the same buttons are added, except the add user is replaced with a remove user button. Also, the same entry boxes are created since whether you are adding or removing a user, their username and password are what confirms it is the right user you want to be affecting. Again, the last line mainloops the window to ensure everything is correctly displayed and functional in the body.

Figure 4.7: Update Admin function from admin window



Update Admin

What would you like to change Admin to?

Old Username:

New Username:

Password:

Update Admin

Main Menu

Exit

Figure 4.7 is the final window exclusive to the admin user and can be found through the main admin window. This window provides the admin the capability to change who has admin access, still maintaining one primary admin user. The admin must first enter their current username and then the new username and password they would like the admin user to be associated with. After which, the admin must click the update admin button to complete the change. Once completed the current user will no longer be able to login as admin with their current information and will instead be a standard user. The new username and password will be able to login as admin and have all admin capabilities. This window also contains the standard main menu and exit app functions that provide the user with options as to how they would like to proceed after completing their desired function.

```

#Update Admin GUI
def updateAdminWin():
    updateAdminWindow = tk.Tk()
    updateAdminWindow.title("Update Admin")
    updateAdminWindow.geometry("1000x500")
    updateAdminWindow.resizable(width=False,height=False)

    updateAdminBody = tk.Frame(updateAdminWindow,bg='#536878',height=500,width=1000)
    updateAdminBody.grid(row=0,column=0)

    btnMainMenu = tk.Button(updateAdminBody,text="Admin Menu",command=updateAdminWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=200)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(updateAdminBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=350)
    btnExit.config(font=("Times New Roman", 28))

    btnAddUser = tk.Button(updateAdminBody,text="Update Admin",command=updateAdmin,width=10, height=2)
    btnAddUser.place(x=750,y=50)
    btnAddUser.config(font=("Times New Roman", 28))

    lblAddTitle = tk.Label(updateAdminBody,text="What would you like to change Admin to?")
    lblAddTitle.config(font=("Times New Roman",24))
    lblAddTitle.place(x=150,y=25)
    lblUsername = tk.Label(updateAdminBody,text="New Username:")
    lblUsername.config(font=("Times New Roman",24))
    lblUsername.place(x=50,y=225)

    global entUsername
    entUsername = tk.Entry(updateAdminBody)
    entUsername.config(font=("Times New Roman", 20))
    entUsername.place(x=50,y=275,width=400,height=60)

    lblPassword = tk.Label(updateAdminBody,text="Password:")
    lblPassword.config(font=("Times New Roman",24))
    lblPassword.place(x=50,y=350)

    global entPassword
    entPassword = tk.Entry(updateAdminBody)
    entPassword.config(font=("Times New Roman", 20))
    entPassword.place(x=50,y=400,width=400,height=60)

    lblOldUser = tk.Label(updateAdminBody,text="Old Username:")
    lblOldUser.config(font=("Times New Roman",24))
    lblOldUser.place(x=50,y=100)

    entOldUser = tk.Entry(updateAdminBody)
    entOldUser.config(font=("Times New Roman", 20))
    entOldUser.place(x=50,y=150,width=400,height=60)

    updateAdminWindow.mainloop()

```

As the final admin function, the code is once again very similar except it has an extra entry box on top of the three buttons and two entry boxes in the add and remove user functions. This third box is for the old admin's username to ensure the right person is being changed. Other than that, the code remains the same with the window and body creation and the mainloop at the end to pack everything together.

Figure 4.8: User success popup



Figure 4.8 is another quick popup that will show the user that they have successfully logged in as a standard user and then redirects them to the main menu.

```
messagebox.showinfo("User Login Success",f"      Welcome {enteredUser}      ")
loginwindow.destroy()
mainMenuWindow()
```

Similar to the admin login popup, this code first flashes a successful user login popup with the user's name in a formatted string. It then closes the login window since it is no longer necessary and calls the main menu to be opened.

Figure 4.9: Main menu that will appear after successful login

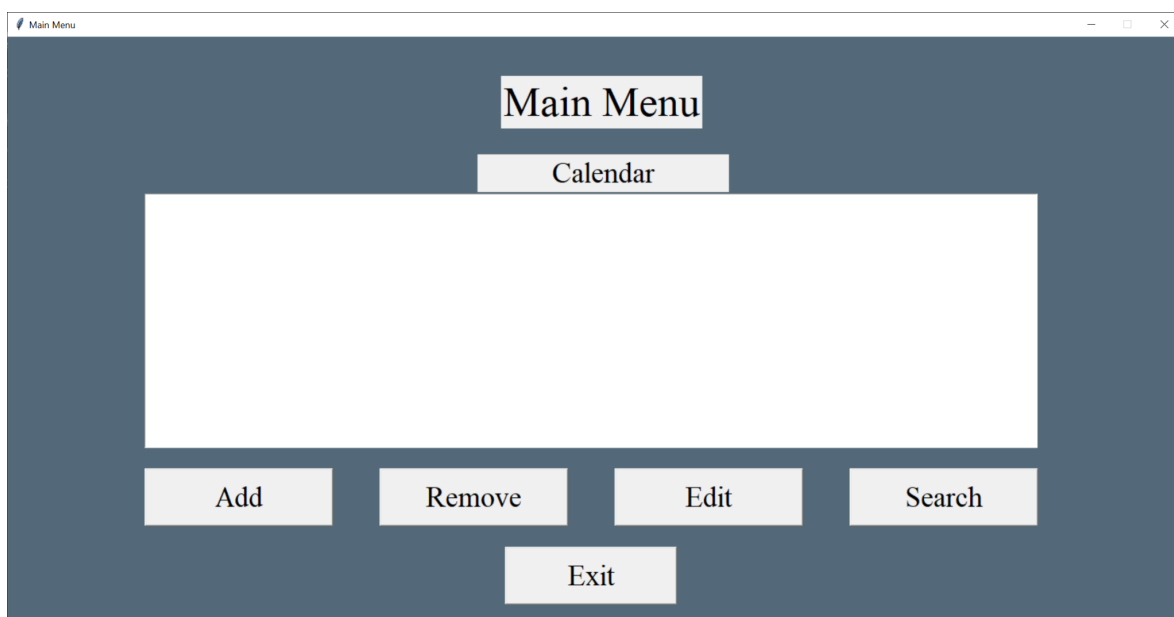


Figure 4.9 is the main menu that the user will interact with either immediately if they are not an admin or through the main menu button on any admin window. This menu allows the user to manipulate and view the task management system along with its contents via the built in buttons which will be further explained in the following Figures. The main menu also will contain a calendar showing the current tasks and their associated traits located in the large central box. Lastly, the main menu has an exit button allowing the user to quit the program entirely when desired.

```
#Main Menu Window
def mainMenuWindow():
    mainWindow = tk.Tk()
    mainWindow.title("Main Menu")
    mainWindow.geometry("1500x750")
    mainWindow.resizable(width=False,height=False)

    mainBody = tk.Frame(mainWindow,bg='#536878',height=750,width=1500)
    mainBody.grid(row=0,column=0)

    lblMainTitle = tk.Label(mainBody,text="Main Menu")
    lblMainTitle.config(font=("Times New Roman",40))
    lblMainTitle.place(x=630,y=50)

    btnExit = tk.Button(mainBody,text="Exit",command=exit,width=10)
    btnExit.place(x=635, y=650)
    btnExit.config(font=("Times New Roman", 28))

    btnAdd = tk.Button(mainBody,text="Add",command=addTaskWin,width=11)
    btnAdd.place(x=175, y=550)
    btnAdd.config(font=("Times New Roman", 28))

    btnRemove = tk.Button(mainBody,text="Remove",command=removeTaskWin,width=11)
    btnRemove.place(x=475, y=550)
    btnRemove.config(font=("Times New Roman", 28))

    btnEdit = tk.Button(mainBody,text="Edit",command=editTaskWin,width=11)
    btnEdit.place(x=775, y=550)
    btnEdit.config(font=("Times New Roman", 28))

    btnSearch = tk.Button(mainBody,text="Search",command=searchTaskWin,width=11)
    btnSearch.place(x=1075, y=550)
    btnSearch.config(font=("Times New Roman", 28))

    lblUserList = tk.Label(mainBody, text="Calendar", width=15)
    lblUserList.place(x = 600, y =150)
    lblUserList.config(font=("Times New Roman", 28))

    txtUserList = tk.Text(mainBody, height=20, width=142)
    txtUserList.place(x=175, y = 200)

    mainWindow.mainloop()
```

The main menu interface is bigger than the rest and starts with creating the window and body while making the sizes unchangeable. A large label is created to show that this window is the main menu, then five buttons are made. The first four bring the user to the functions of the TMS and the last exits the program. Then a large text box is made that will act as a calendar to display tasks. Finally the whole window is mainlooped to pack it all together and ensure the widgets all work properly.

Figure 4.10: Add function from the main menu

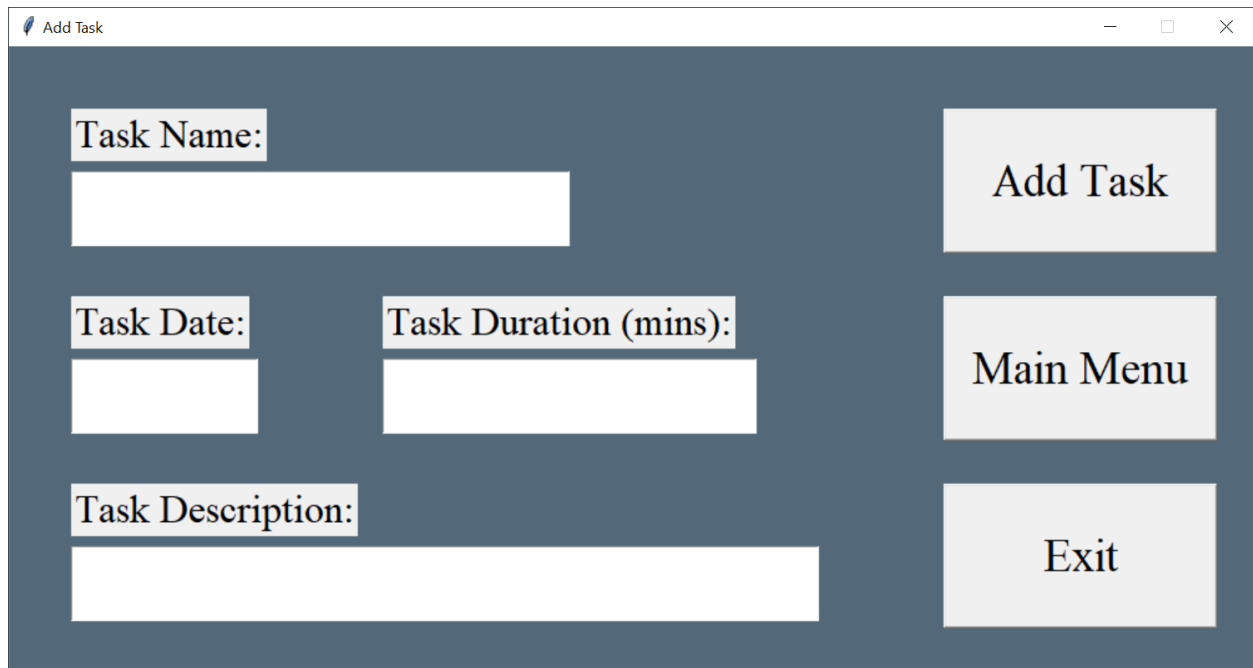
The image shows a window titled "Add Task" with a dark blue background. On the left side, there are four input fields: "Task Name:" with a single-line text box, "Task Date:" with a single-line text box, "Task Duration (mins):" with a single-line text box, and "Task Description:" with a multi-line text box. On the right side, there are three buttons: "Add Task" at the top, "Main Menu" in the middle, and "Exit" at the bottom. The window has standard window controls (minimize, maximize, close) in the top right corner.

Figure 4.10 is the window that will appear when the user wishes to add a task. The text boxes will ask the user to input the task name, date, duration, and description, all of which will be stored when the user presses the Add Task button. The window will close upon the press of any of the three side buttons. Add Task will store the data in the text boxes and then close, Main Menu will return the user to the main menu (Figure 4.9), and Exit App will close everything including the main menu.



```
#Add Task GUI
def addTaskWin():
    addWindow = tk.Tk()
    addWindow.title("Add Task")
    addWindow.geometry("1000x500")
    addWindow.resizable(width=False,height=False)

    addBody = tk.Frame(addWindow,bg='#536878',height=500,width=1000)
    addBody.grid(row=0,column=0)

    btnMainMenu = tk.Button(addBody,text="Main Menu",command=addWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=200)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(addBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=350)
    btnExit.config(font=("Times New Roman", 28))

    btnAddTask = tk.Button(addBody,text="Add Task",command=addTask,width=10, height=2)
    btnAddTask.place(x=750,y=50)
    btnAddTask.config(font=("Times New Roman", 28))

    lblTaskName = tk.Label(addBody,text="Task Name:")
    lblTaskName.config(font=("Times New Roman",24))
    lblTaskName.place(x=50,y=50)

    global entTaskName
    entTaskName = tk.Entry(addBody)
    entTaskName.config(font=("Times New Roman", 20))
    entTaskName.place(x=50,y=100,width=400,height=60)

    lblDate = tk.Label(addBody,text="Task Date:")
    lblDate.config(font=("Times New Roman",24))
    lblDate.place(x=50,y=200)

    global entDate
    entDate = tk.Entry(addBody)
    entDate.config(font=("Times New Roman", 20))
    entDate.place(x=50,y=250,width=150,height=60)

    lblDuration = tk.Label(addBody,text="Task Duration (mins):")
    lblDuration.config(font=("Times New Roman",24))
    lblDuration.place(x=300,y=200)

    global entDuration
    entDuration = tk.Entry(addBody)
    entDuration.config(font=("Times New Roman", 20))
    entDuration.place(x=300,y=250,width=300,height=60)

    lblDesc = tk.Label(addBody,text="Task Description:")
    lblDesc.config(font=("Times New Roman",24))
    lblDesc.place(x=50,y=350)

    global entDesc
    entDesc = tk.Entry(addBody)
    entDesc.config(font=("Times New Roman", 20))
    entDesc.place(x=50,y=400,width=600,height=60)

    addWindow.mainloop()
```

The add function starts off by creating a window and body that is unchangeable in size. Then three buttons are made and placed that allow the user to add the task, return to the main menu, or to exit the program. Four entry boxes and their respective labels are then made to accept the name, date, duration, and description of the task that the user wants to add. These entries are made global for later use and then the whole window is mainlooped to pack the widgets and make them functional in the program.

Figure 4.11: Remove function from the main menu

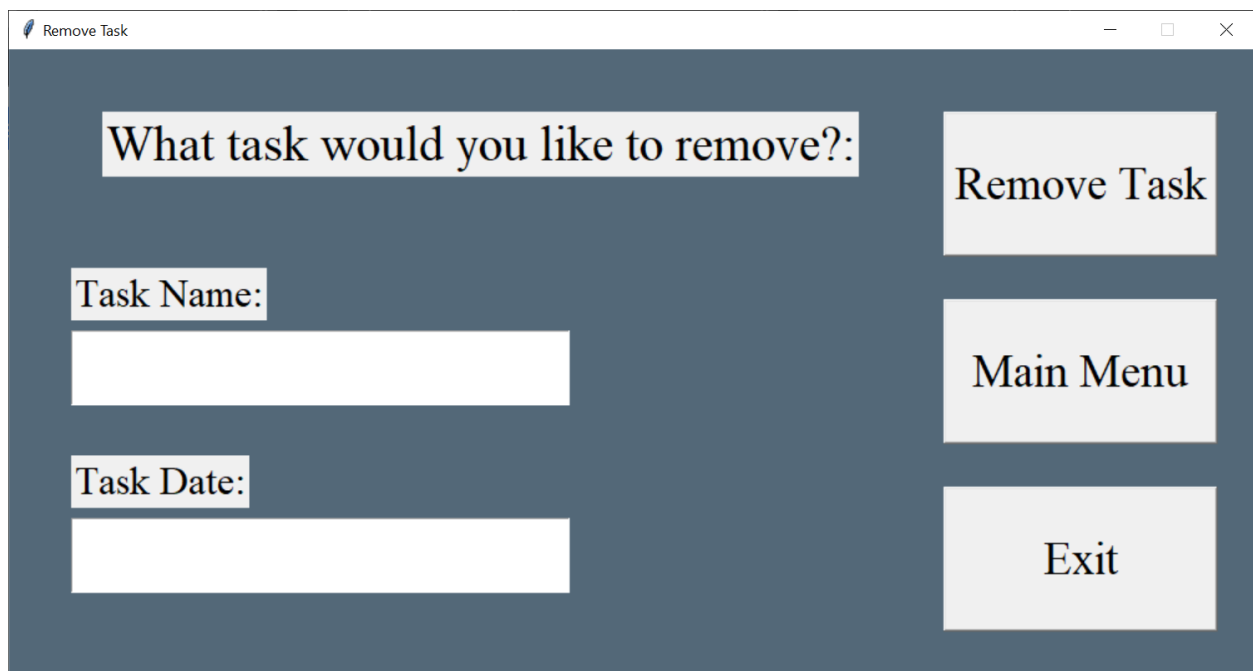


Figure 4.11 is an extension of the main menu window that can be reached by clicking the remove button. This window gives the user the capability to remove any task within the TMS by providing the name and date of the desired task. After which, the user will have to click remove task to finish the operation and then the task will be fully removed from the TMS profile of that user. This window also has a main menu and exit app button allowing the user to either return to the main menu or quit the app after completing their desired function.

```
#Remove Task GUI
def removeTaskWin():
    removeWindow = tk.Tk()
    removeWindow.title("Remove Task")
    removeWindow.geometry("1000x500")
    removeWindow.resizable(width=False,height=False)

    removeBody = tk.Frame(removeWindow,bg='#536878',height=500,width=1000)
    removeBody.grid(row=0,column=0)

    lblRemove = tk.Label(removeBody,text="What task would you like to remove?:")
    lblRemove.config(font=("Times New Roman",30))
    lblRemove.place(x=75,y=50)

    btnMainMenu = tk.Button(removeBody,text="Main Menu",command=removeWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=200)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(removeBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=350)
    btnExit.config(font=("Times New Roman", 28))

    btnAddUser = tk.Button(removeBody,text="Remove Task",command=removeTask,width=10, height=2)
    btnAddUser.place(x=750,y=50)
    btnAddUser.config(font=("Times New Roman", 28))

    lblTaskName = tk.Label(removeBody,text="Task Name:")
    lblTaskName.config(font=("Times New Roman",24))
    lblTaskName.place(x=50,y=175)

    entTaskName = tk.Entry(removeBody)
    entTaskName.config(font=("Times New Roman", 20))
    entTaskName.place(x=50,y=225,width=400,height=60)

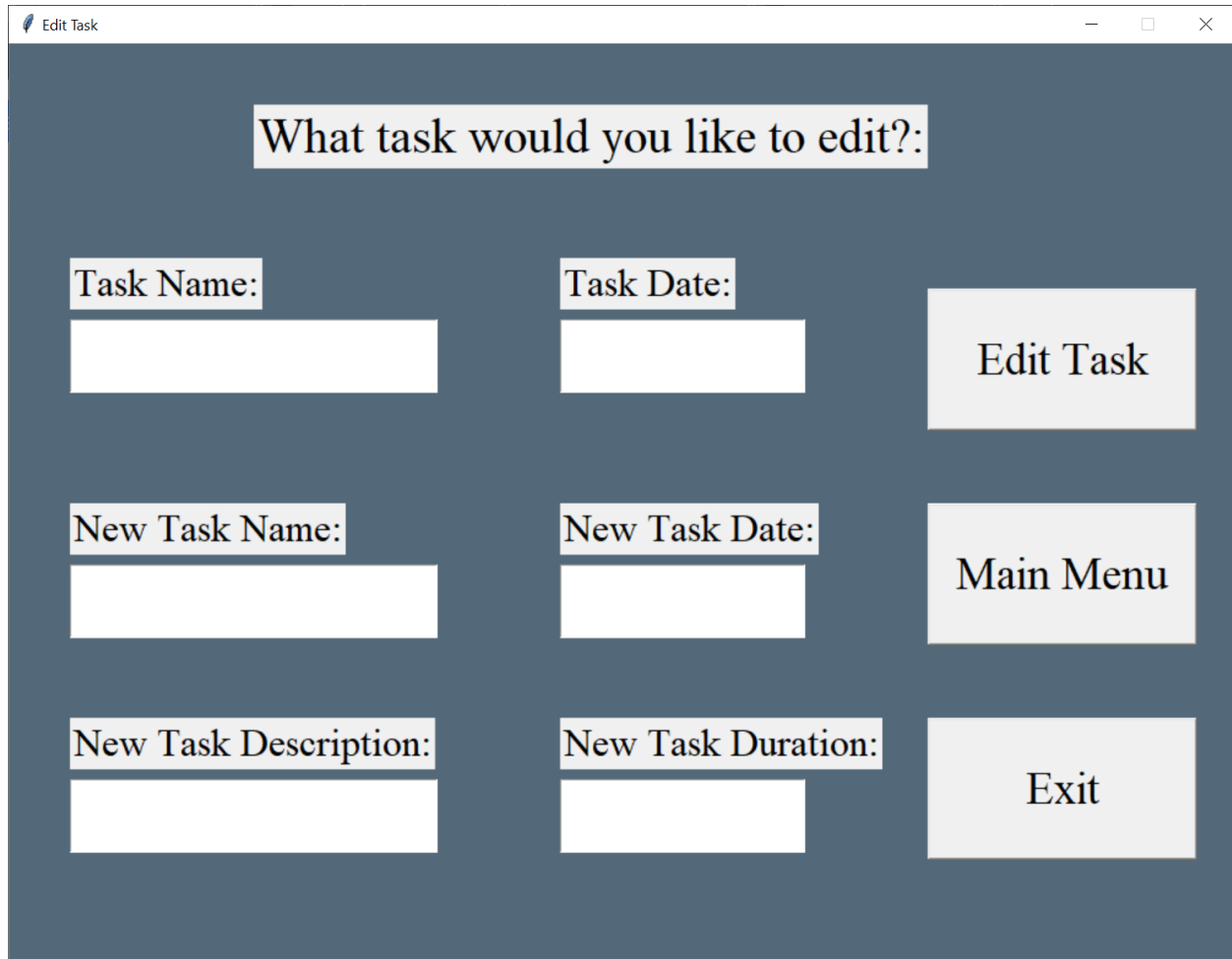
    lblDate = tk.Label(removeBody,text="Task Date:")
    lblDate.config(font=("Times New Roman",24))
    lblDate.place(x=50,y=325)

    entDate = tk.Entry(removeBody)
    entDate.config(font=("Times New Roman", 20))
    entDate.place(x=50,y=375,width=400,height=60)

    removeWindow.mainloop()
```

The remove task window is similar in structure to the add task, however it only has the entry boxes for name and date as opposed to the four boxes in add. It similarly has three buttons, one for removing the task, one for returning to the main menu, and one for exiting the program. It also follows the same formula as all the other windows with the creation of an unchangeable window and body to start and the mainloop to bring it all together at the end.

Figure 4.12: Edit function from the main menu



The screenshot shows a window titled "Edit Task" with a dark blue background. At the top, a white box contains the text "What task would you like to edit?:". Below this, there are three rows of input fields and buttons. The first row has "Task Name:" and "Task Date:" labels, each followed by a white text box, and a button labeled "Edit Task". The second row has "New Task Name:" and "New Task Date:" labels, each followed by a white text box, and a button labeled "Main Menu". The third row has "New Task Description:" and "New Task Duration:" labels, each followed by a white text box, and a button labeled "Exit".

Figure 4.12 is the window that will appear when the user wishes to edit a task that already exists in the system. The text boxes will call for the user to enter the task name and date of the task they want to edit. This will allow us to identify the correct task if it is one that appears multiple times. Next the user will input the new data for the task. The window will close upon the press of any of the three side buttons. Edit Task will update the data in our system and then close, Main Menu will return the user to the main menu (Figure 4.9), and Exit App will close everything including the main menu.

```
#Edit Task GUI
def editTaskWin():
    editWindow = tk.Tk()
    editWindow.title("Edit Task")
    editWindow.geometry("1000x750")
    editWindow.resizable(width=False,height=False)

    editBody = tk.Frame(editWindow,bg='#536878',height=750,width=1000)
    editBody.grid(row=0,column=0)

    lblEdit = tk.Label(editBody,text="What task would you like to edit?:")
    lblEdit.config(font=("Times New Roman",30))
    lblEdit.place(x=200,y=50)

    btnMainMenu = tk.Button(editBody,text="Main Menu",command=editWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=375)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(editBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=550)
    btnExit.config(font=("Times New Roman", 28))

    btnEdit = tk.Button(editBody,text="Edit Task",command=editTask,width=10, height=2)
    btnEdit.place(x=750,y=200)
    btnEdit.config(font=("Times New Roman", 28))

    lblTaskName = tk.Label(editBody,text="Task Name:")
    lblTaskName.config(font=("Times New Roman",24))
    lblTaskName.place(x=50,y=175)

    global entTaskName
    entTaskName = tk.Entry(editBody)
    entTaskName.config(font=("Times New Roman", 20))
    entTaskName.place(x=50,y=225,width=300,height=60)

    lblDate = tk.Label(editBody,text="Task Date:")
    lblDate.config(font=("Times New Roman",24))
    lblDate.place(x=450,y=175)

    global entDate
    entDate = tk.Entry(editBody)
    entDate.config(font=("Times New Roman", 20))
    entDate.place(x=450,y=225,width=200,height=60)
```

```
lblNewName = tk.Label(editBody,text="New Task Name:")
lblNewName.config(font=("Times New Roman",24))
lblNewName.place(x=50,y=375)

global entNewName
entNewName = tk.Entry(editBody)
entNewName.config(font=("Times New Roman", 20))
entNewName.place(x=50,y=425,width=300,height=60)

lblNewDate = tk.Label(editBody,text="New Task Date:")
lblNewDate.config(font=("Times New Roman",24))
lblNewDate.place(x=450,y=375)

global entNewDate
entNewDate = tk.Entry(editBody)
entNewDate.config(font=("Times New Roman", 20))
entNewDate.place(x=450,y=425,width=200,height=60)

lblNewDesc = tk.Label(editBody,text="New Task Description:")
lblNewDesc.config(font=("Times New Roman",24))
lblNewDesc.place(x=50,y=550)

global entNewDesc
entNewDesc = tk.Entry(editBody)
entNewDesc.config(font=("Times New Roman", 20))
entNewDesc.place(x=50,y=600,width=300,height=60)

lblNewDur = tk.Label(editBody,text="New Task Duration:")
lblNewDur.config(font=("Times New Roman",24))
lblNewDur.place(x=450,y=550)

global entNewDur
entNewDur = tk.Entry(editBody)
entNewDur.config(font=("Times New Roman", 20))
entNewDur.place(x=450,y=600,width=200,height=60)

editWindow.mainloop()
```

The edit function starts with the creation of the unchangeable window and body as usual, and also maintains the three button structure on the right side. There's an edit button, main menu button, and an exit program button. This function has six entry boxes for the old name and date, then the new name, date, duration, and description of the task. All of the entries are globalized for later use, and the end of the program mainloops the window to ensure everything is packed together and functional when it is displayed.

Figure 4.13: Search Function from the main menu

The screenshot shows a window titled "Search Task" with a dark blue background. At the top, a white box contains the text "Search for tasks using one or more identifiers". Below this, the window is divided into three main sections. On the left, there are four input fields with labels: "Task Name:", "Task Date:", "Task Duration:", and "Task Description:". Each label is in a white box, and the input fields are white rectangles. In the center, there is a large white rectangular area labeled "Tasks" at the top. On the right side, there are three buttons: "Search", "Main Menu", and "Exit", each in a white box.

Figure 4.13 is the final function a user can perform within the TMS and can be found from the main menu by clicking the search button. This function will allow a user to find tasks with matching attributes to the provided information allowing for both general searches of a single day or a complex search to find an exact task. Once the user provides the desired information in the entries, they can press search to complete the action and find the tasks with matching attributes. This window also includes the standard main menu and exit functions allowing the user to complete more tasks if they desire or quit the app entirely.

```
#Search Task GUI
def searchTaskWin():
    searchWindow = tk.Tk()
    searchWindow.title("Search Task")
    searchWindow.geometry("1000x750")
    searchWindow.resizable(width=False,height=False)

    searchBody = tk.Frame(searchWindow,bg='#536878',height=750,width=1000)
    searchBody.grid(row=0,column=0)

    lblEdit = tk.Label(searchBody,text="Search for tasks using one or more identifiers")
    lblEdit.config(font=("Times New Roman",30))
    lblEdit.place(x=200,y=50)

    btnMainMenu = tk.Button(searchBody,text="Main Menu",command=searchWindow.destroy,width=10, height=2)
    btnMainMenu.place(x=750,y=420)
    btnMainMenu.config(font=("Times New Roman", 28))

    btnExit = tk.Button(searchBody,text="Exit",command=exit,width=10, height=2)
    btnExit.place(x=750,y=620)
    btnExit.config(font=("Times New Roman", 28))

    btnSearch = tk.Button(searchBody,text="Search",command=exit,width=10, height=2)
    btnSearch.place(x=750,y=220)
    btnSearch.config(font=("Times New Roman", 28))

    lblTaskName = tk.Label(searchBody,text="Task Name:")
    lblTaskName.config(font=("Times New Roman",24))
    lblTaskName.place(x=50,y=175)

    entTaskName = tk.Entry(searchBody)
    entTaskName.config(font=("Times New Roman", 20))
    entTaskName.place(x=50,y=225,width=300,height=60)

    lblDate = tk.Label(searchBody,text="Task Date:")
    lblDate.config(font=("Times New Roman",24))
    lblDate.place(x=50,y=325)

    entDate = tk.Entry(searchBody)
    entDate.config(font=("Times New Roman", 20))
    entDate.place(x=50,y=375,width=200,height=60)

    lblNewName = tk.Label(searchBody,text="Task Duration:")
    lblNewName.config(font=("Times New Roman",24))
    lblNewName.place(x=50,y=475)

    entNewName = tk.Entry(searchBody)
    entNewName.config(font=("Times New Roman", 20))
    entNewName.place(x=50,y=525,width=300,height=60)
```



```
lblNewDesc = tk.Label(searchBody, text="Task Description:")
lblNewDesc.config(font=("Times New Roman", 24))
lblNewDesc.place(x=50, y=625)

entNewDesc = tk.Entry(searchBody)
entNewDesc.config(font=("Times New Roman", 20))
entNewDesc.place(x=50, y=675, width=300, height=60)

lblTasks = tk.Label(searchBody, text="Tasks")
lblTasks.place(x = 500, y = 150)
lblTasks.config(font=("Times New Roman", 28))

txtTasks = tk.Text(searchBody, height=32, width=40)
txtTasks.place(x=390, y = 220)

searchWindow.mainloop()
```

Just as the other windows are created, the search window is created and made to be unchangeable with a body the same size. A label is made instructing the users to enter information that will help the TMS search for specific tasks. The same typical three buttons are made allowing the user to search for tasks, return to the main menu, or to exit the program. Four entry boxes are made for the user to enter the name, date, duration, and description of the tasks they are looking for in each labeled box. Then a big text box is made that will display the tasks that match the entered information. Finally, the window is mainlooped to pack all the widgets and ensure they work as intended upon calling the function.

### Full UX and UI Diagram

[https://drive.google.com/file/d/1bygt078\\_N-g\\_0CjdQ1B8VBVUbyjlgudS/view?usp=sharing](https://drive.google.com/file/d/1bygt078_N-g_0CjdQ1B8VBVUbyjlgudS/view?usp=sharing)

### Data Storage

Data from the program is stored in two ways: through a users.csv file and an associated csv file for each user in the database. When a user is created via the admin menu, the users.csv file is updated to create a new row with the person's admin value and their credentials. If a person has a stored admin value of 1, they are granted admin permissions upon login and if they have a 0, they will only be able to access the main menu. Upon user creation a new csv file will be created in their name to better organize the data and keep each user's tasks private only to

them. By having a file associated with each user, the program can only manipulate their associated tasks allowing for multiple users to interact with the program without issues. Similarly, when a user is deleted their credentials are erased from the users.csv file and their associated csv in their name is deleted too so there will only be as many csv files as current users within the program.

For the coding aspect of the storage, we used a combination of built in python functions and the pandas package to receive and manipulate the data provided by the user. In doing so we found the greatest success in creating a sustainable program that can do a combination of functions without losing any data from the users.