

# Goal Journal

Introduction to Programming

CMPT 120L



Marist College

School of Computer Science and Mathematics

Submitted To:

Dr. Reza Sadeghi

Date: April 18, 2024

# Project Progress Report 4 Team Sean

# Team Name

Team Sean

## Team members

1. Sean Bruno                      sean.bruno1@marist.edu (Team Head)

## Description of Team Members

1. My name is Sean Bruno, I am an applied math major with a minor in data science and analytics. I like working by myself and making sure all the work is done. Especially when it comes to a project, so I have decided to pursue a group of just me. I have learned a lot about programming in this course as I have never tried programming before.

# Table of Contents

1. Table of Figures.....	4
2. Project Objective .....	5
3. GitHub Address.....	6
4. User Experience Design .....	7
5. Flowchart.....	8
6. Packages .....	9
7. Virtual Environment.....	9
8. Graphical User Interface Design .....	10
i. Login.....	10
ii. Admin Menu.....	13
iii. Main Menu.....	15
9. Data Storage.....	20
10. Resources.....	21

## Table of Figures

1. Figure 1.0 Flowchart.....	8
2. Figure 1.1 Virtual Environment.....	9
3. Figure 1.2 Login Page.....	10
4. Figure 1.3 Successful Login.....	11
5. Figure 1.4 Failed Login.....	12
6. Figure 1.5 Admin Menu.....	13
7. Figure 1.6 Main Menu.....	15
8. Figure 1.7 Edit Entry.....	18
9. Figure 1.8 Search Results.....	19
10. Figure 1.9 Login Information CSV file .....	20
11. Figure 1.10 Journal txt file.....	20

## Project Objective

**Project Title:** Goal Journal Management System

**Summary:** My project will be a Goal Journal Management System, which will be very similar to the Diary Management System which was given in project sample 2. The Goal Journal will be available for the user to journal an achievement or when they complete something that day. For example, if their overall goal is to go to the gym more, then the user will be able to input when they went to the gym and can leave an entry with it. The data will be stored in a CSV file which stands for Comma Separated Values. The GJMS will be able to provide the following capabilities:

1. Requesting admin user and password for log in
2. Admin user should be able to add a user to GJMS by creating a new username and password for normal users, who are not able to define or remove a user
3. Admin user should be able to remove a user from GJMS by removing his/her username, password, and his/her other corresponding data
4. Each user should be able to:
  - A. Add a Journal record with the following details: time, date, duration, and description
  - B. Remove a Journal record
  - C. Edit a Journal record's details
  - D. Search through Journals based on the date and list the results on the screen.
6. GJMS should be a user-friendly software, such that:
  - A. It shows a warning if all the fields are not filled out properly to add an entry
  - B. GJMS should show a Login page
  - C. GJMS should show a menu of all functions to the user
  - D. GJMS should provide a list of the previous entries visible to the user
  - E. GJMS should provide an exit function
7. Each user will have their own txt file created where there data will be stored
8. All login information will be stored in a CSV file

## **GitHub Address**

[https://github.com/SeanBruno2/111\\_GoalJournalManagmentSystem\\_TeamSean.git](https://github.com/SeanBruno2/111_GoalJournalManagmentSystem_TeamSean.git)

## User Experience Design

In my Goal Journal Management System I plan on having 6 pages. Including a login, main menu, adding a journal entry, removing a journal entry, editing a journal entry, searching for a journal entry, and a set password page.

i. Login:

Input: Username and Password

Output: Correct username and password will be granted access to the system. While an incorrect username and password will be sent a warning message saying incorrect.

ii. Adding a Journal Entry:

Input: Time, date, duration, and description of something you completed to be closer to your goal. For example, Time:1230pm, Date:2/4/2024, Duration:60 minutes, Description: Went to gym for an hour to workout.

Output: Journal entry will be added to the csv value where all entry's will be stored.

iii. Removing a Journal Entry:

Input: Time, date, duration, and description of journal entry you would like to remove.

Output: This journal entry will be removed from storage

iv. Editing Journal Entry:

Input: Input: Time, date, duration, and description with the correction to the journal entry that you would like to edit.

Output: Entry will be edited in the csv file.

v. Search:

Input: Date of entry you are searching for.

Output: The Journal entries that corresponds to the given date.

vi. Add User:

Input: Username and Password in which you would like associated to account.

Output: The password corresponding to the username will be updated.

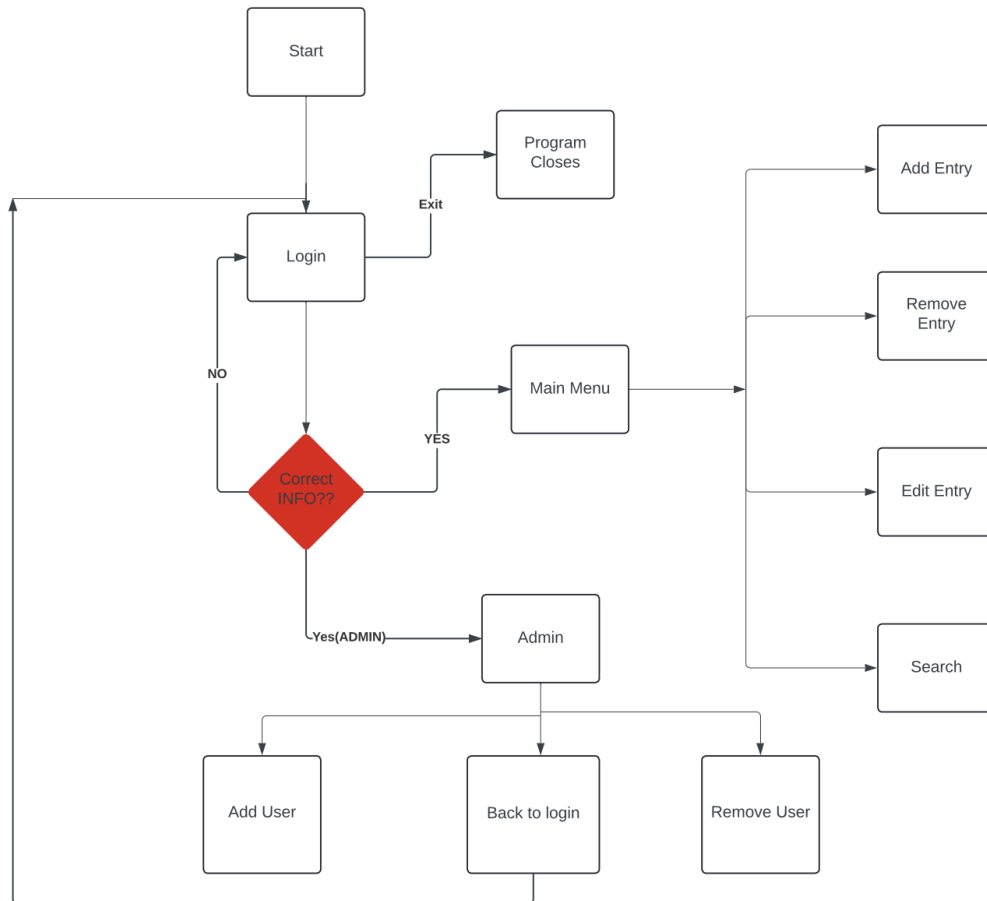
Vii. Administrative:

Input: Adding a User or Removing a User

Output: User will be added or removed

# Flow Chart

Figure 1.0





## Packages to be used

- OS
- Tkinter
- Csv
- Custom Tkinter
- PIL (imaging)

## Virtual Environment

Figure 1.1 Virtual Environment

In terminal I created my virtual environment below.

```
[seanbrunojr@148-100-135-106 ~ % python3 -m venv sample  
[seanbrunojr@148-100-135-106 ~ % source sample/bin/activate  
(sample) seanbrunojr@148-100-135-106 ~ %
```

Here is a list of all the packages installed from terminal.

```
[seanbrunojr@148-100-135-106 ~ % python3 -m pip list  
Package            Version  
-----  
certifi             2023.11.17  
colour              0.1.5  
customtkinter       5.2.2  
darkdetect          0.8.0  
excel               1.0.1  
numpy               1.26.4  
packaging            24.0  
pandas              2.2.1  
pillow              10.3.0  
pip                 24.0  
pymssql             2.2.11  
PyMySQL             1.1.0  
python-dateutil     2.9.0.post0  
pytz                2024.1  
six                 1.16.0  
tkmacosx            1.0.5  
tzdata              2024.1
```

## GUI Design

### i. Login

Figure 1.2 Login Page

The Login Page is the first page a user will see, so I wanted to make it appealing. I added a photo of a mountain which shows success at the top. I made sure to make the background color appealing and will match the vibe of the picture. I Included a title of Goal Journal at the bottom of the page to describe and label the program. There is a space for the username and password to be entered which will take you to admin or main menu based on the login information.



Here is the code that will create the GUI of this login page.

```

root = tk.Tk()
root.title("Login")

root.configure(bg="slategrey")

login_frame = tk.Frame(root,bg="slategrey")
login_frame.pack(padx=20, pady=20)

image_path = "Mountain.jpeg"
image = Image.open(image_path)
tk_image = ImageTk.PhotoImage(image)
label = tk.Label(login_frame, image=tk_image)
label.grid(row=0,column=1,padx=5,pady=5)

username_lbl = tk.Label(login_frame, text="Username:",bg="slategrey",fg="white")
username_lbl.grid(row=1, column=0, padx=5, pady=5)
username_var = tk.StringVar()
username_ent = tk.Entry(login_frame, textvariable=username_var, width=30,bg="slategrey",fg="white")
username_ent.grid(row=1, column=1, padx=5, pady=5)

password_lbl = tk.Label(login_frame, text="Password:",bg="slategrey",fg="white")
password_lbl.grid(row=2, column=0, padx=5, pady=5)
password_var = tk.StringVar()
password_ent = tk.Entry(login_frame, textvariable=password_var, show="*", width=30,bg="slategrey",fg="white")
password_ent.grid(row=2, column=1, padx=5, pady=5)

login_btn = tk.Button(login_frame, text="Login", command=check_login)
login_btn.grid(row=1, column=2, padx=5, pady=5)

exit_btn=tk.Button(login_frame, text="Exit",command=Exit)
exit_btn.grid(row=2, column=2, padx=5, pady=5)

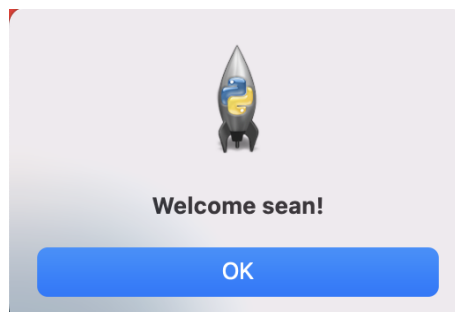
title_lbl=tk.Label(login_frame,text="GOAL JOURNAL",font=("Arial",20),bg="slategrey",fg="white")
title_lbl.grid(row=3,column=1)

root.mainloop()

```

Figure 1.3 Successful Login

This is the message box when login is successful. Which is shown by the code below, using a message box.



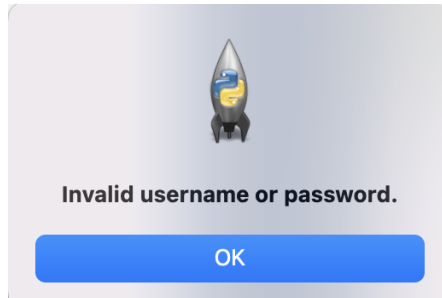
```

~~~~~
messagebox.showinfo("Login Successful", f"Welcome {username}!")
root.withdraw()
user_menu(username)

```

Figure 1.4 Failed Login

This message box appears when either the password given, username, or both are incorrect. Which is programmed by using a message box, shown in the code below.



```
messagebox.showerror("Login Failed", "Invalid username or password.")
```

The Login Page will be reading from a csv file called users. To read from the csv file it must open it in reading mode. Then it must implement the csv reader in order to check if the credentials are correct. In this case I defined Login, where the program will go into the csv file and see if the user provided valid information, this function will return nothing. I then implement that login into a new function that I defined as check\_login, where then the users information is checked if it is an admin or normal user, and if the username and password are correct.

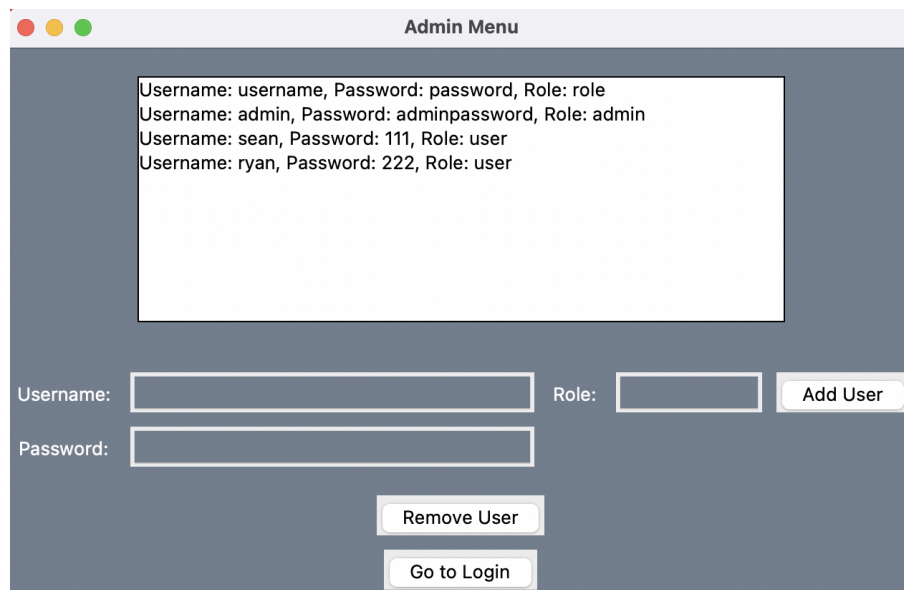
```
def login(username, password):
    with open("users.csv", "r") as file:
        csv_reader = csv.reader(file)
        for row in csv_reader:
            if row[0] == username and row[1] == password:
                return {"username": username, "role": row[2]}
    return None

def check_login():
    username = username_var.get()
    password = password_var.get()
    user = login(username, password)
    if user:
        if user["role"] == "admin":
            messagebox.showinfo("Login Successful", "Welcome Admin!")
            root.withdraw()
            admin_menu()
        else:
            messagebox.showinfo("Login Successful", f"Welcome {username}!")
            root.withdraw()
            user_menu(username)
    else:
        messagebox.showerror("Login Failed", "Invalid username or password.")
```

## ii. Admin Menu

Figure 1.5 Admin Menu

The Admin Menu is shown when an admin logs in successfully. From the admin menu the user will see a list of all the users in the system. Including their role,username, and password. The admin has the option to add or remove a user. They also have a button that will bring them back to the login page and sign them out.



The screenshot shows a web application window titled "Admin Menu". Inside, there is a white box containing a list of users with their details: Username, Password, and Role. Below this list, there are input fields for Username, Password, and Role, followed by an "Add User" button. At the bottom, there are two buttons: "Remove User" and "Go to Login".

Username	Password	Role
username	password	role
admin	adminpassword	admin
sean	111	user
ryan	222	user

Username:  Role:

Password:



```

def admin_menu():
    admin_menu_window = tk.Toplevel(root)
    admin_menu_window.title("Admin Menu")
    admin_menu_window.configure(bg="slategrey")

    users_frame = tk.Frame(admin_menu_window, bg="slategrey")
    users_frame.pack(padx=10, pady=10)

    users_listbox = tk.Listbox(users_frame, width=50)
    users_listbox.pack(padx=10, pady=10)
    populate_users_list()

    add_user_frame = tk.Frame(admin_menu_window, bg="slategrey")
    add_user_frame.pack(pady=10)
    username_lbl = tk.Label(add_user_frame, text="Username:", bg="slategrey", fg="white")
    username_lbl.grid(row=0, column=0, padx=5, pady=5)
    username_var = tk.StringVar()
    username_ent = tk.Entry(add_user_frame, textvariable=username_var, width=30, bg="slategrey", fg="white")
    username_ent.grid(row=0, column=1, padx=5, pady=5)

    password_lbl = tk.Label(add_user_frame, text="Password:", bg="slategrey", fg="white")
    password_lbl.grid(row=1, column=0, padx=5, pady=5)
    password_var = tk.StringVar()
    password_ent = tk.Entry(add_user_frame, textvariable=password_var, width=30, bg="slategrey", fg="white")
    password_ent.grid(row=1, column=1, padx=5, pady=5)

    role_lbl = tk.Label(add_user_frame, text="Role:", bg="slategrey", fg="white")
    role_lbl.grid(row=2, column=0, padx=5, pady=5)
    role_var = tk.StringVar()
    role_ent = tk.Entry(add_user_frame, textvariable=role_var, width=10, bg="slategrey", fg="white")
    role_ent.grid(row=2, column=1, padx=5, pady=5)

    add_user_btn = tk.Button(add_user_frame, text="Add User", command=add_user)
    add_user_btn.grid(row=3, column=0, padx=5, pady=5)

    remove_user_btn = tk.Button(admin_menu_window, text="Remove User", command=remove_user)
    remove_user_btn.pack(pady=5)

```

The Admin menu will also be reading from a csv file, called users as well. But this time the admin has the option to add or remove from this file. There will be a listbox for the Admin to see, this listbox displays the csv file, where it lists all the users in the system. To add a user the csv file will be opened in append mode, where the program will write a new row of the given information from the admin. To remove a user, the admin will select what user they would like to remove in the listbox, the program will create an index for this selection. Then the program will open the csv file in read and be converted to a list called data. Then the data is written back into the csv excluding the selected user to remove.

```

def add_user():
    username = username_var.get()
    password = password_var.get()
    role = role_var.get()
    with open("users.csv", "a") as file:
        csv_writer = csv.writer(file)
        csv_writer.writerow([username, password, role])
    populate_users_list()

def remove_user():
    selection = users_listbox.curselection()
    if selection:
        index = int(selection[0])
        users_listbox.delete(index)
        with open("users.csv", "r") as file:
            data = list(csv.reader(file))
        with open("users.csv", "w", newline='') as file:
            csv_writer = csv.writer(file)
            csv_writer.writerows(data[:index] + data[index+1:])

```

### iii. Main Menu

Figure 1.6 Main Menu

The main menu is shown when a user, who is not given admin permissions, logs in. From here the user will have the following options: add a journal entry, remove a journal entry, edit an entry, and search for an existing entry. Each entry will display: Title, Date(mm/dd/yy), Time(minutes), and a description, these all have places to enter. There is also a listbox displaying all of the entries. I chose the color of the same menu to keep the same theme.

```
def user_menu(username):
    user_menu_window = tk.Toplevel(root)
    user_menu_window.title(f"Welcome {username}")
    user_menu_window.configure(bg="slategrey")

    user_menu_frame = tk.Frame(user_menu_window, bg="slategrey")
    user_menu_frame.pack(padx=10, pady=10)
```

```

users_listbox = tk.Listbox(users_frame, width=50)
users_listbox.pack(padx=10, pady=10)
populate_users_list()

add_user_frame = tk.Frame(admin_menu_window, bg="slategrey")
add_user_frame.pack(pady=10)
username_lbl = tk.Label(add_user_frame, text="Username:", bg="slategrey", fg="white")
username_lbl.grid(row=0, column=0, padx=5, pady=5)
username_var = tk.StringVar()
username_ent = tk.Entry(add_user_frame, textvariable=username_var, width=30, bg="slategrey", fg="white")
username_ent.grid(row=0, column=1, padx=5, pady=5)

password_lbl = tk.Label(add_user_frame, text="Password:", bg="slategrey", fg="white")
password_lbl.grid(row=1, column=0, padx=5, pady=5)
password_var = tk.StringVar()
password_ent = tk.Entry(add_user_frame, textvariable=password_var, width=30, bg="slategrey", fg="white")
password_ent.grid(row=1, column=1, padx=5, pady=5)

role_lbl = tk.Label(add_user_frame, text="Role:", bg="slategrey", fg="white")
role_lbl.grid(row=2, column=0, padx=5, pady=5)
role_var = tk.StringVar()
role_ent = tk.Entry(add_user_frame, textvariable=role_var, width=10, bg="slategrey", fg="white")
role_ent.grid(row=2, column=1, padx=5, pady=5)

add_user_btn = tk.Button(add_user_frame, text="Add User", command=add_user)
add_user_btn.grid(row=3, column=1, padx=5, pady=5)

remove_user_btn = tk.Button(admin_menu_window, text="Remove User", command=remove_user)
remove_user_btn.pack(pady=5)

```

The goal journal entries will be saved into a txt file. Each user will have there own file, stored as {username}\_journal.txt. In order to add a journal entry I must get all the entries that the user entered, the program will get the entries as a string variable. Then stored these into entry. The program will then open the txt file in appended mode and write the entry.

```

def add_journal_entry():
    title = title_var.get()
    date = date_var.get()
    time = time_var.get()
    duration = duration_var.get()
    description = description_var.get()
    entry = f"{title}, {date}, {time}, {duration}, {description}\n"
    journal_file = f"{username}_journal.txt"
    with open(journal_file, "a") as file:
        file.write(entry)
    view_journal()

```



The remove entry function will be deleting an entry from the txt file. This function will have the user select an entry with their cursor that they would like to remove. It will then open the txt file in read, and read the lines. Next the program will open up in write and re write the entries excluding the one selected to remove.

```
def remove_entry():
    selection = journal_listbox.curselection()
    if selection:
        index = int(selection[0])
        journal_listbox.delete(index)
        journal_file = f"{username}_journal.txt"
        with open(journal_file, "r") as file:
            lines = file.readlines()
        with open(journal_file, "w") as file:
            for i, line in enumerate(lines):
                if i != index:
                    file.write(line)
```

The function to search an entry will be able to search the txt file by a certain date and return the result in a pop up window. This function will open up the txt file in read and be able to split the entries apart by commas, then find the date.

```
def search_by_date():
    date_to_search = search_entry.get()
    found_entries = []
    journal_file = f"{username}_journal.txt"
    if os.path.exists(journal_file):
        with open(journal_file, "r") as file:
            for line in file:
                entry = line.strip().split(", ")
                if len(entry) >= 5 and date_to_search in entry[1]:
                    found_entries.append(entry)
```

Figure 1.7 Edit Entry Window

When editing an entry, the user will need to select an entry with their cursor, then press the edit entry page. This will pop open an entry page where the user can enter a new title, date, time and description.

```
users_listbox = tk.Listbox(users_frame, width=50)
users_listbox.pack(padx=10, pady=10)
populate_users_list()

add_user_frame = tk.Frame(admin_menu_window, bg="slategrey")
add_user_frame.pack(pady=10)
username_lbl = tk.Label(add_user_frame, text="Username:", bg="slategrey", fg="white")
username_lbl.grid(row=0, column=0, padx=5, pady=5)
username_var = tk.StringVar()
username_ent = tk.Entry(add_user_frame, textvariable=username_var, width=30, bg="slategrey", fg="white")
username_ent.grid(row=0, column=1, padx=5, pady=5)

password_lbl = tk.Label(add_user_frame, text="Password:", bg="slategrey", fg="white")
password_lbl.grid(row=1, column=0, padx=5, pady=5)
password_var = tk.StringVar()
password_ent = tk.Entry(add_user_frame, textvariable=password_var, width=30, bg="slategrey", fg="white")
password_ent.grid(row=1, column=1, padx=5, pady=5)

role_lbl = tk.Label(add_user_frame, text="Role:", bg="slategrey", fg="white")
role_lbl.grid(row=2, column=0, padx=5, pady=5)
role_var = tk.StringVar()
role_ent = tk.Entry(add_user_frame, textvariable=role_var, width=10, bg="slategrey", fg="white")
role_ent.grid(row=2, column=1, padx=5, pady=5)

add_user_btn = tk.Button(add_user_frame, text="Add User", command=add_user)
add_user_btn.grid(row=3, column=0, padx=5, pady=5)

remove_user_btn = tk.Button(admin_menu_window, text="Remove User", command=remove_user)
remove_user_btn.pack(pady=5)
```

The edit function will be able to delete the old entry from this file and then write the new one in. This program will do this by opening the text file in read and removing the selected entry like the remove function then append the new information into the txt file like the add entry function.

```
def save_changes():
    edited_title = title_var.get()
    edited_date = date_var.get()
    edited_time = time_var.get()
    edited_description = description_var.get()

    if not edited_title or not edited_date or not edited_time or not edited_description:
        messagebox.showerror("Error", "One or more components are missing.")
        return

    edited_entry = ", ".join([edited_title, edited_date, edited_time, edited_description])

    journal_listbox.delete(selected_index[0])

    journal_listbox.insert(selected_index[0], edited_entry)
    journal_file = f'{username}_journal.txt'
    with open(journal_file, "r") as file:
        entries = file.readlines()

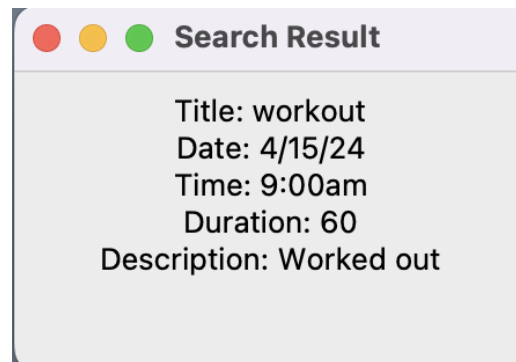
    entries[selected_index[0]] = edited_entry + "\n"

    with open(journal_file, "w") as file:
        file.writelines(entries)

    edit_window.destroy()
```

Figure 1.8 Search Entry Results Window

When searching for an entry, the user will enter the date and press search, doing so will pop up a new window with the entries that were found from that date.



```
if found_entries:
    search_result_window = tk.Toplevel(root)
    search_result_window.title("Search Result")

    for entry in found_entries:
        title, date, time, duration, description = entry
        entry_label = tk.Label(search_result_window, text=f"Title: {title}\nDate: {date}\nTime: {time}\nDuration: {duration}\nDescription: {description}")
        entry_label.pack(pady=5)
```

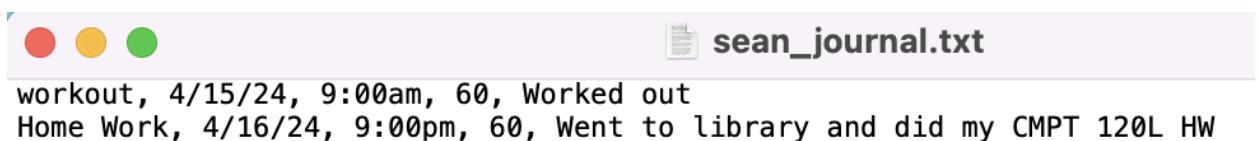
## Data Storage

In this program I chose to use a csv and txt file. The login information, such as, username, password, and role(admin or user) were stored in a csv file. The journal entries information, such as, title, date, time, duration, and description were all saved in a txt file. We stored all the data from the user into these files. As you can see from the previous figures, when adding an entry we were able to append into the txt file. When doing other tasks to the entries we were able to read and write into these txt files. The program creates a txt file for each user, the file is named {their username}\_journal.txt. When logging in the program reads the csv file to make sure the username and password are valid information, and what role is associated with the user. The admin has access to add and remove users from the csv files. The program does this by being able to read the csv file and writing. Using these csv and txt files will store the data that the user inputs and this data will not be lost, the data will be able to be recalled when the program is closed. Csv and txt files are the smartest way to store data of a management system for these reasons.

Figure 1.9 Login Information CSV file

	A	B	C
1	username	password	role
2	admin	adminpassw	admin
3	sean	111	user
4	ryan	222	user

Figure 1.10 Journal txt file



The image shows a screenshot of a text editor window titled "sean\_journal.txt". The window contains two lines of text representing journal entries. The first line is "workout, 4/15/24, 9:00am, 60, Worked out" and the second line is "Home Work, 4/16/24, 9:00pm, 60, Went to library and did my CMPT 120L HW". The text is displayed in a monospaced font.

## Resources

[https://matplotlib.org/stable/gallery/color/named\\_colors.html](https://matplotlib.org/stable/gallery/color/named_colors.html)

<https://www.istockphoto.com/photos/reaching-goals-mountain>

<https://realpython.com/python-gui-tkinter/>

[https://www.tutorialspoint.com/python/tk\\_pack.htm](https://www.tutorialspoint.com/python/tk_pack.htm)

<https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/>

<https://coderslegacy.com/python/tkinter-close-window/>

<https://www.hackerearth.com/practice/python/getting-started/string/tutorial/>

<https://www.lucidchart.com>

[https://www.w3schools.com/python/python\\_strings\\_modify.asp](https://www.w3schools.com/python/python_strings_modify.asp)

<https://www.geeksforgeeks.org/python-list-index/>