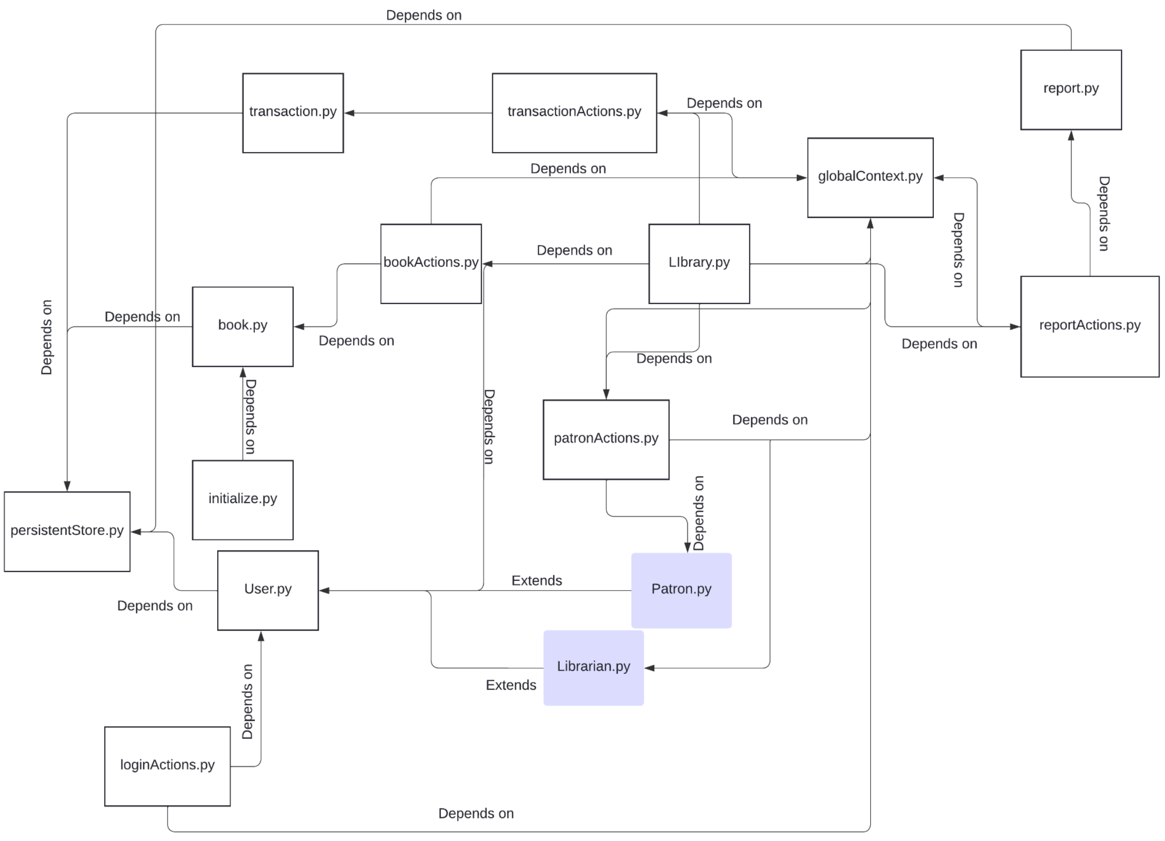
Library Management System (LMS)

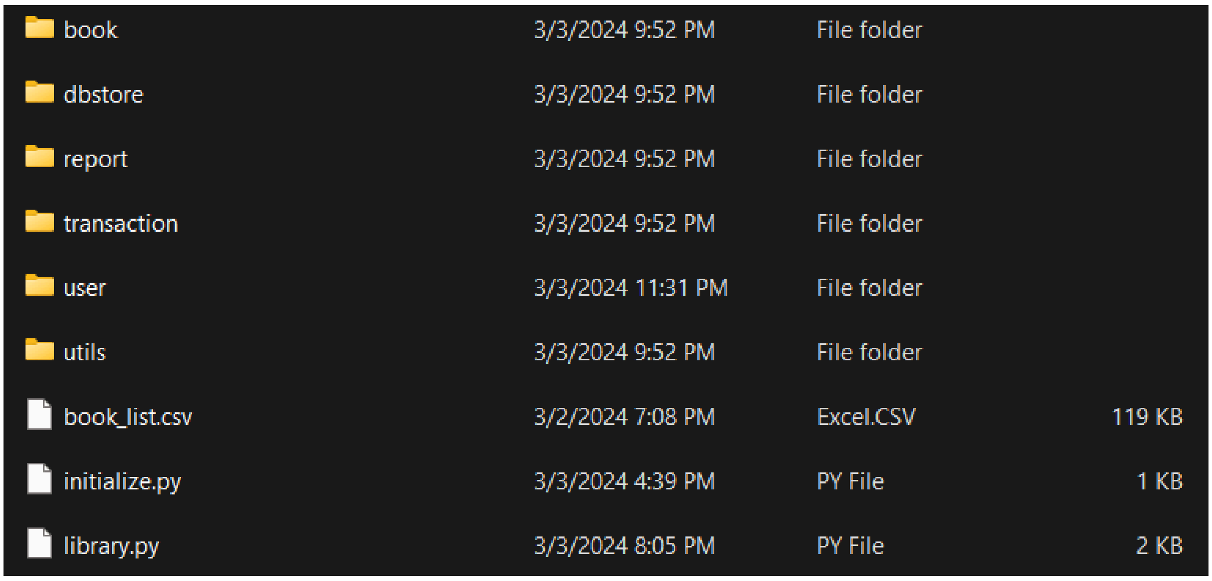
Description of the project

This project aims to create a Library Management System (LMS), as the title suggests, in python. It provides administrators and librarians with a platform to manage their library with just one program. It allows for adding, removing, updating, and searching books, as well as adding, removing, updating, and viewing all patrons and transactions in one place. There is additionally special role based access for patrons and librarians, patrons are allowed to check out and check in books as well as search for books and pay fines. However they are not allowed higher privileges which are only reserved to the librarians like adding and deleting books and patrons and other users as well as generating reports.

Structure of the code with diagram and some comments regarding structure

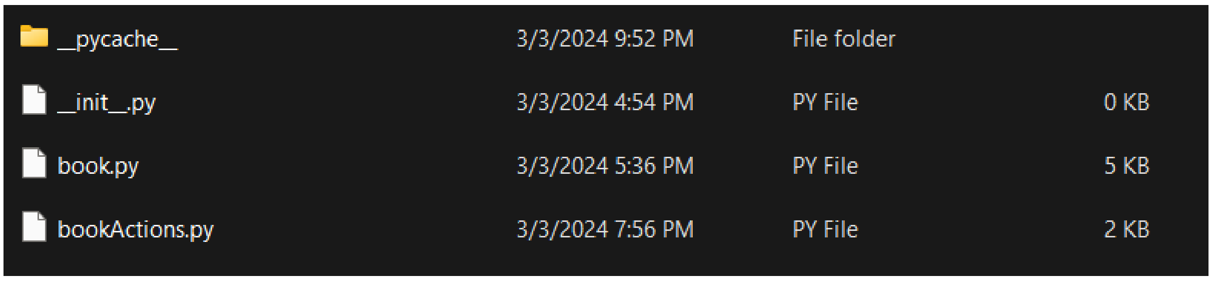


I ran out of free shapes towards the end but if it has an arrow and does not say "Extend" then it means that the class depends on the class that it is pointing to.
The code has a proper folder structure where everything is in its own little folder:



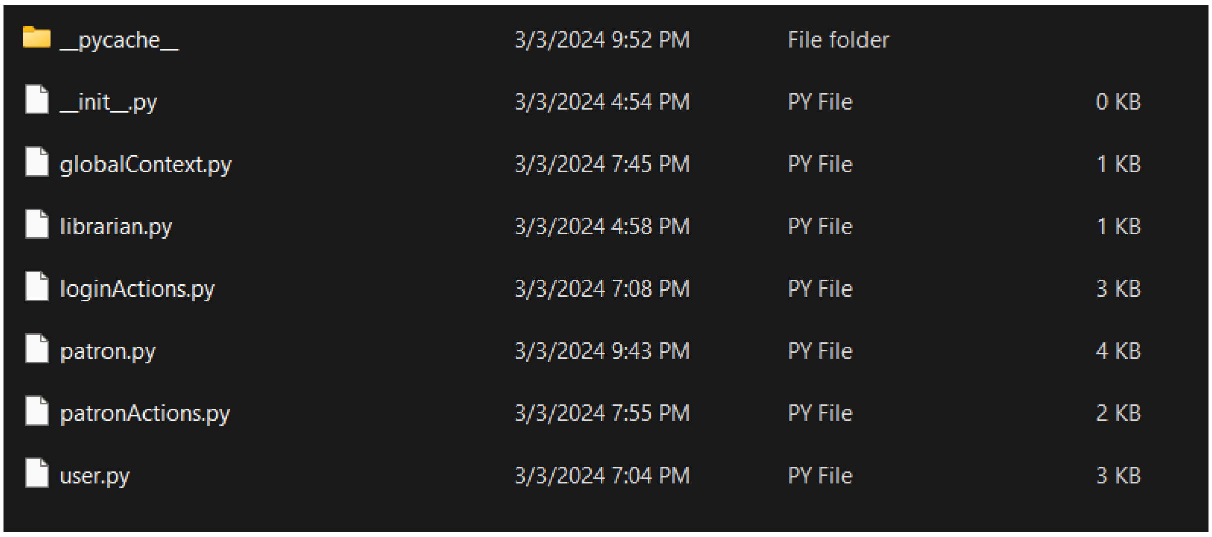
book	3/3/2024 9:52 PM	File folder	
dbstore	3/3/2024 9:52 PM	File folder	
report	3/3/2024 9:52 PM	File folder	
transaction	3/3/2024 9:52 PM	File folder	
user	3/3/2024 11:31 PM	File folder	
utils	3/3/2024 9:52 PM	File folder	
book_list.csv	3/2/2024 7:08 PM	Excel.CSV	119 KB
initialize.py	3/3/2024 4:39 PM	PY File	1 KB
library.py	3/3/2024 8:05 PM	PY File	2 KB

Eg for book:



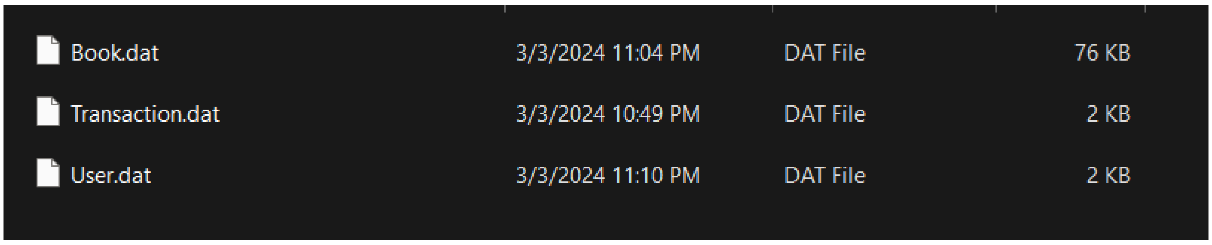
__pycache__	3/3/2024 9:52 PM	File folder	
__init__.py	3/3/2024 4:54 PM	PY File	0 KB
book.py	3/3/2024 5:36 PM	PY File	5 KB
bookActions.py	3/3/2024 7:56 PM	PY File	2 KB

Eg for user:



__pycache__	3/3/2024 9:52 PM	File folder	
__init__.py	3/3/2024 4:54 PM	PY File	0 KB
globalContext.py	3/3/2024 7:45 PM	PY File	1 KB
librarian.py	3/3/2024 4:58 PM	PY File	1 KB
loginActions.py	3/3/2024 7:08 PM	PY File	3 KB
patron.py	3/3/2024 9:43 PM	PY File	4 KB
patronActions.py	3/3/2024 7:55 PM	PY File	2 KB
user.py	3/3/2024 7:04 PM	PY File	3 KB

Eg for dbstore:



Book.dat	3/3/2024 11:04 PM	DAT File	76 KB
Transaction.dat	3/3/2024 10:49 PM	DAT File	2 KB
User.dat	3/3/2024 11:10 PM	DAT File	2 KB

Some comments regarding the structure:

Core Entities:

User.py: Represents a class that embodies a user within the system, encompassing both patrons and librarians.
Patron.py: An extension of User.py, specifically tailored to represent patrons with additional attributes or methods.
Librarian.py: Also extends User.py, delineating librarians with distinct or supplementary functionalities compared to patrons.
book.py: Centralizes the representation of the book class within the library system.
transaction.py: Represents various transactions within the library, such as checkouts, returns, or managing fines.
report.py: Represents the report class, utilized for generating reports within the system.

Action Modules:

loginActions.py: Contains functions dedicated to user authentication and login procedures.
bookActions.py: Prompts the user for book related actions.
patronActions.py: Prompts the user for patron related actions.
transactionActions.py: Prompts the user for book transaction related actions.
reportActions.py: Prompts the user for book report related actions.
Supporting Modules:

initialize.py: Utilized for initializing the system, configuring databases, or preparing the environment upon system startup.
persistentStore.py: Facilitates interaction with a persistent file system, for storing and retrieving and modifying data.
globalContext.py: Manages global context or shared data necessary across different components of the application.

Dependencies:

The diagram illustrates a "Depends on" relationship between various modules, signifying that certain modules require others to function effectively. For instance, bookActions.py depends on globalContext.py, indicating a necessity to access global context data for book-related actions to know if the user can access transaction related actions (only librarians should be able to create or update books whereas patrons should only be able to search or display books)

Inheritance:

The "Extends" relationship denotes inheritance within object-oriented programming paradigms. Patron.py and Librarian.py both inherit from User.py, indicating they inherit common structures or behaviors from the User class while possessing additional specific features.

Instructions on how to use it for full functionalities provided by your code.

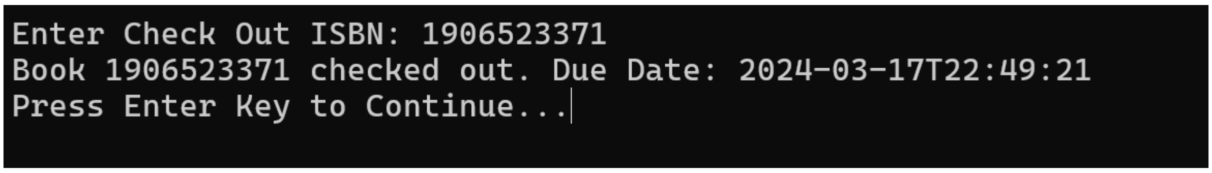
Run initialize.py in the terminal to initialize the database with some sample books. **(Only if you do not have a Book.dat file in your /dbstore folder or if your Book.dat file is empty)**
Run library.py in the terminal to start using the library management system.
You can edit the save files in the dbstore folder to edit Books Users and Transactions in order to change the due date of books to manage fines. I can send a video on a tutorial if you would like. The google doc link below shows every screen in the program and could help with example usage.

Verification of the sanity of the code to check the validity of the implemented functionalities.

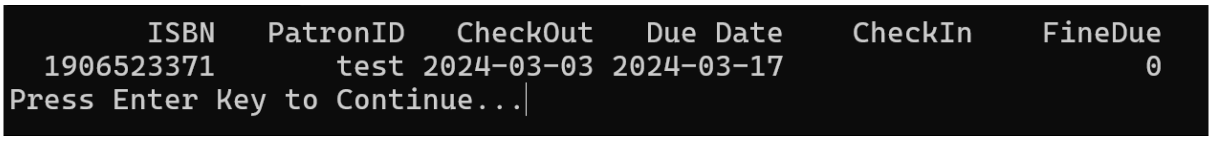
- Include screenshots or examples of code execution.
- Provide sample scenarios demonstrating how to perform common operations such as adding books, checking out books, and generating reports.

EVERY SINGLE SCREENSHOT IN THE PROGRAM:
https://docs.google.com/document/d/10P4WZUO1ihtcfsdMI3fzWkg8-MVFNNUv8QbZLt-_Aqo/edit?usp=sharing

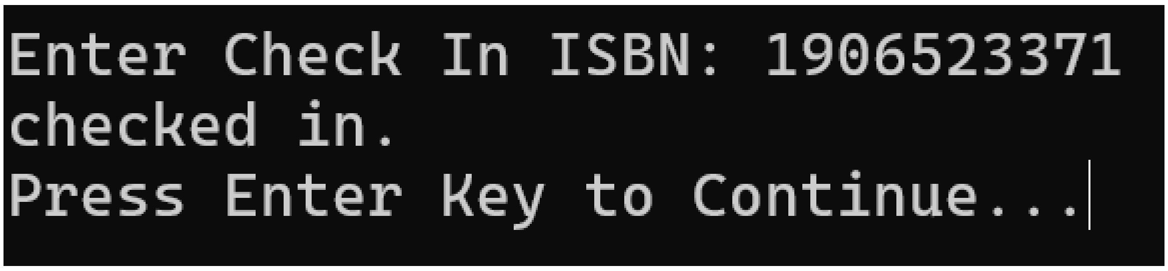
In a different document so this doc is not cluttered.
Here are the screenshots for adding books, checking out books, and generating reports as a sample:



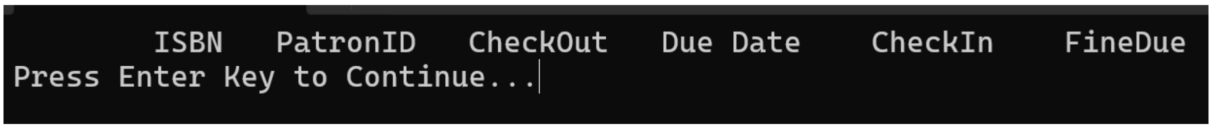
```
Enter Check Out ISBN: 1906523371
Book 1906523371 checked out. Due Date: 2024-03-17T22:49:21
Press Enter Key to Continue...
```



```
        ISBN    PatronID    CheckOut    Due Date     CheckIn     FineDue
  1906523371        test 2024-03-03 2024-03-17                         0
Press Enter Key to Continue...
```

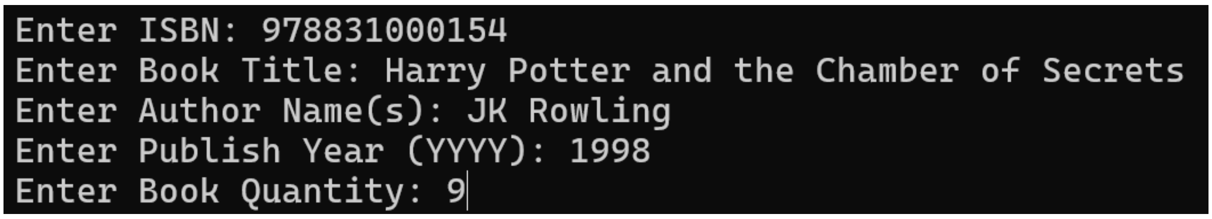


```
Enter Check In ISBN: 1906523371
checked in.
Press Enter Key to Continue...
```

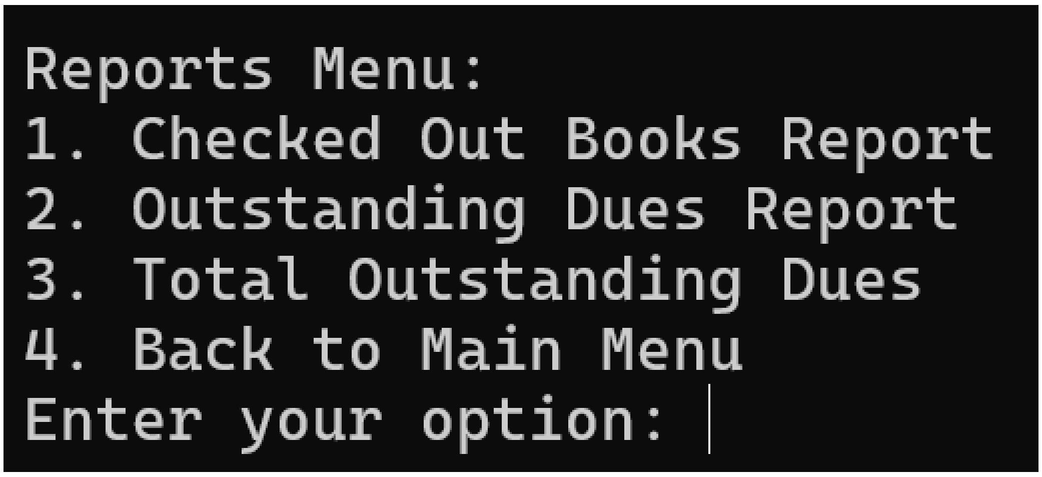

```
        ISBN    PatronID    CheckOut    Due Date     CheckIn     FineDue
Press Enter Key to Continue...
```

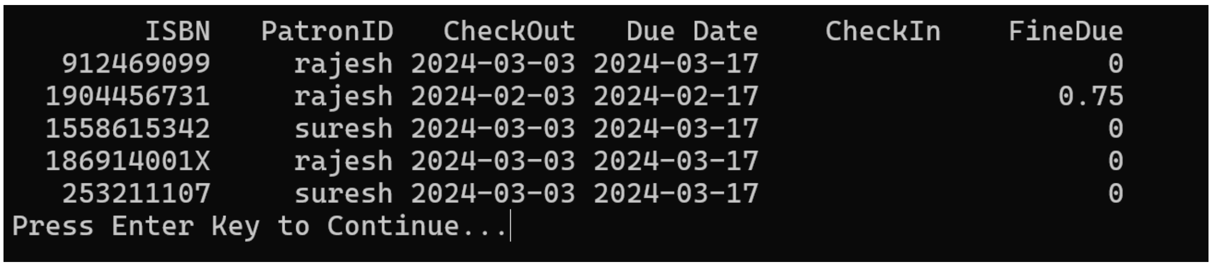
Checking out books ^^

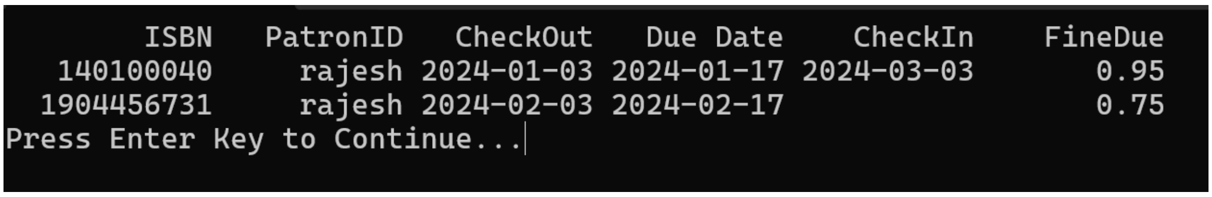
ADDING BOOK:

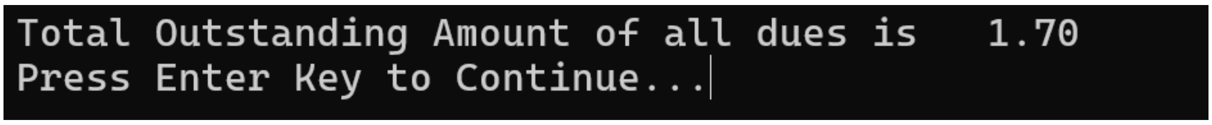

```
Enter ISBN: 978831000154
Enter Book Title: Harry Potter and the Chamber of Secrets
Enter Author Name(s): JK Rowling
Enter Publish Year (YYYY): 1998
Enter Book Quantity: 9
```

REPORTS MENU:


```
Reports Menu:
1. Checked Out Books Report
2. Outstanding Dues Report
3. Total Outstanding Dues
4. Back to Main Menu
Enter your option: 
```


```
        ISBN    PatronID    CheckOut    Due Date     CheckIn     FineDue
   912469099      rajesh 2024-03-03 2024-03-17                         0
  1904456731      rajesh 2024-02-03 2024-02-17                      0.75
  1558615342      suresh 2024-03-03 2024-03-17                         0
  186914001X      rajesh 2024-03-03 2024-03-17                         0
   253211107      suresh 2024-03-03 2024-03-17                         0
Press Enter Key to Continue...
```


```
        ISBN    PatronID    CheckOut    Due Date     CheckIn     FineDue
   140100040      rajesh 2024-01-03 2024-01-17 2024-03-03         0.95
  1904456731      rajesh 2024-02-03 2024-02-17                    0.75
Press Enter Key to Continue...
```


```
Total Outstanding Amount of all dues is   1.70
Press Enter Key to Continue...
```

Discuss your finding through this project, challenges faced during implementation, any limitations or areas for improvement

Through the implementation of this project, it became clear that while adding new books was straightforward by simply appending a stringified JSON object to a DAT file, updating or deleting existing entries posed significant challenges. Unlike the ease of appending, modifying or removing specific entries required file manipulation, as direct modification or deletion of lines wasn't possible. This complexity arose from the necessity to read the entire file, locate the relevant entry, and rewrite the file with the updated data. While I had considered alternative database solutions like SQLite, I made the decision to use a file-based storage system over SQLite. It was driven by reduced dependencies, simplifying setup, and presenting a valuable learning opportunity in file handling and data manipulation which I think is the most important part of the project.

This is also an area of improvement for the project as if this project were to be scaled up and hosted elsewhere with multiple users the constant overwriting and rewriting would increase the risk of data corruption. This adversity truly made me appreciate how modern database technologies let you do these seemingly “simple” things so effortlessly.

An additional hurdle I had to overcome during the making of this project was the portability of the project, as I chose to use a file based database I could not simply assume that the other users of this project would also have their database file in C:/Code for example therefore I made a folder and called it dbstore so I could use that folder as the base location of the database.

Another challenge that I had to overcome was actually about the IDE I was using. I was using VS Code to write this project and found that my file based system that simply dumped a stringified JSON object with the id in the front (think: ID{obj}) was getting flagged as an actual JSON file. This meant that VS Code would end up randomly adding lines here and there in an attempt to format the file to actually look like a JSON file. This caused errors when I tried to update an object in the program it would mess up. The solution to this issue was to add a file extension at the end, I chose .dat so no other program would interfere with the file and mess up the data. Interestingly enough I did some research on file extensions and found that .dat is often used in game development to save game files and progress, along with .sav which makes sense as you would not want the user’s save files to be manipulated by other programs.

I realized part way through that in order to have patrons be able to pay fines and such I would need to add additional attributes to Patron in order to properly facilitate the managing of fines on a per user basis.

When implementing role based access into the program I realized that I needed a way to figure out if the user that is logged in is a patron or a librarian as that would change the menus that I would show to that user. That is why I made a global context file that has the details of the particular user that is in session.

Additionally I used getpass to get the password in a more privacy friendly way as it automatically makes the password that the user is typing invisible. Additionally I also hashed the password so as to not store it in plain text as storing passwords in plain text is bad practice, I was torn between using an external library like bcrypt or using the builtin hashlib but chose to use hashlib due to the ease of access and lack of setup, the process of getting a salt and then hashing the password and comparing hashed passwords with each other to see if the passwords are equal was a quick and easy way to securely store and validate passwords that I learned via this project.

If there is one last improvement I would make or limitation I would chose to remove it is the library employee ID, presently that ID does nothing but in the future of this project I would love to have a file with a list of “approved” librarian IDs so I could validate librarian registration and restrict regular users from becoming librarians and gain access to the LMS that they otherwise should not be able to. I think it would be a cool upgrade to a great project.

Overall I felt as though I was challenged with this project and found the process of making a LMS in python very rewarding.