PROGRAMMING FOR INFORMATION SYSTEMS (B9IS123_2324_TMD1S)

Library Management System using Python Flask framework

Group Member
Pradumnya Kailas Zendphale -20003538
Mohammad Salik Syed - 20020194
Pratik Shivraj Poojari - 20000871

Contents

Introduction

The Library Management System (LMS) is a perfect instance of the way that technology and organizational effectiveness may work together to build a digital environment that is specifically designed to make all aspects of library operations run smoothly. This comprehensive system, created with the Python Flask framework, integrates various technologies, including HTML, CSS, JavaScript, and Bootstrap, to provide a user interface that is responsive and easy to use. The backend expertise makes use of Azure for smooth deployment and PostgreSQL's strength to guarantee solid data management.

The primary objective of this project is to change the standard library system by giving users access to a consolidated digital platform. It is a platform that gives administrators and users access to specific functions on their dashboards. The architecture of the system takes a comprehensive approach, balancing complex backend operations with user-friendly interfaces to provide an effective and efficient library management system.

The primary function of the LMS is effectively handling user credentials, book records, and fines. Users have the ability to access and change their login credentials and to look over their history of books issued and fines accumulated. Users are allowed to borrow up to three books at a time because of the system's enforcement of restrictions. Users need to decide how long they want books to be issued for. Late returns will result in a penalty of 1 EURO per day after the designated return date.

At the same time, the librarian dashboard provides Librarians unheard-of authority, allowing them to effectively manage the library's operations. By the help of the administrative suite, Librarians can supervise the addition and removal of books, manage user accounts, impose late return fines, and do other essential duties that keep the library running smoothly.

Selection of Technology and framework

- **Python Flask :**

  Flask is considered a lightweight and flexible micro framework that allows developers to choose their tool and libraries they need. The key feature of flask such as Routing, HTTP request handling, templating for html pages creation and works with REST APIs.

  We choose Python over Java because of its widespread use and our familiarity with it from our practical classroom experience. The Python web framework Flask was the subject of our class instruction, so it made a reliable and smooth transition into our project. After doing some research, we discovered that Flask works best in smaller projects and situations where sophisticated authentication features are not absolutely necessary.

  We took Flask's because of its quick implementation and ease of learning when making our decisions because they matched the project's goals and scale. This decision was also in line with the general trend of Python's increasing popularity in the software building community.

- **HTML & CSS**

  We used html for our templates as it is the foundation of web pages. Content, which includes headings, paragraphs, lists, links, images, and more, is organized and structured using tags.

  We used CSS for styling and presentation of elements to web pages as CSS enhances HTML. It enhances the overall aesthetics by providing developers control over the visual elements of the content, such as layout, colors, fonts, spacing, and more.

- **Bootstrap :**

  One of the most popular css frameworks, we used its predefined responsive layouts and pre-made components as they are ready to use.
  In our project we use predefined buttons, forms, navigation bars, Tables and Alerts.

- **JQuery:**

  We used JQuery for responsiveness and as it requires less code than plain JavaScript to accomplish common tasks like DOM manipulation, event handling,

  We handled user inputs, clicks, and other types of user actions using an event-driven paradigm and redirected requests accordingly.

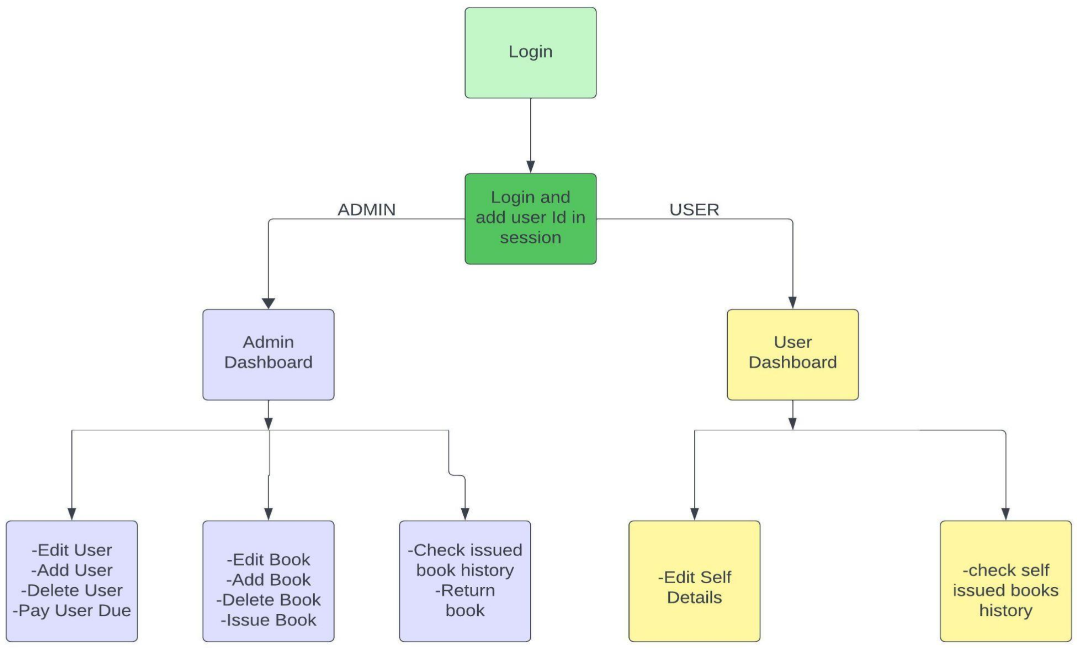
- **Azure :**

  We chose Azure because of its simple interface, which makes it perfect for those who are new to deployment. It is one of the best options for our class projects since it offers free student credits with college personalized email ID . Our database deployment was made simpler by using Azure Database for PostgreSQL flexible server, which complied with our coursework's requirements and the module learning process.

- **Postgresql Database :**

  Since we had used PostgreSQL before and found it to be a reliable and user-friendly database system, we decided to use it for our project. To be more precise, we used PostgreSQL extensively for our project work in our advanced database module, which allowed us to gain practical experience and insight. This familiarity helped us to efficiently utilize PostgreSQL's advanced features while also streamlining our workflow. Our choice as engineering students was influenced by our familiarity with the technology and its track record of dependability when managing challenging data situations. Our practical experience in the module project equipped us with the knowledge necessary to make well-informed decisions and optimize our database design in order to meet the demands of our academic program.

Functional Diagram -



- Any Entity will have to login to see the respective dashboard(ADMIN OR USER). Once login their user_id will be added to the session and redirected to the respective dashboard.
- System does not accept duplicate username and contact and will throw an error.
- It is possible for the admin to search for books using the book ID or title. Additionally, the admin has a search function that allows them to locate users by name or user ID.
- To make the return process easier, the admin can also search for issuers by name or by the title of the book they borrowed.

- **USERS functionality-**

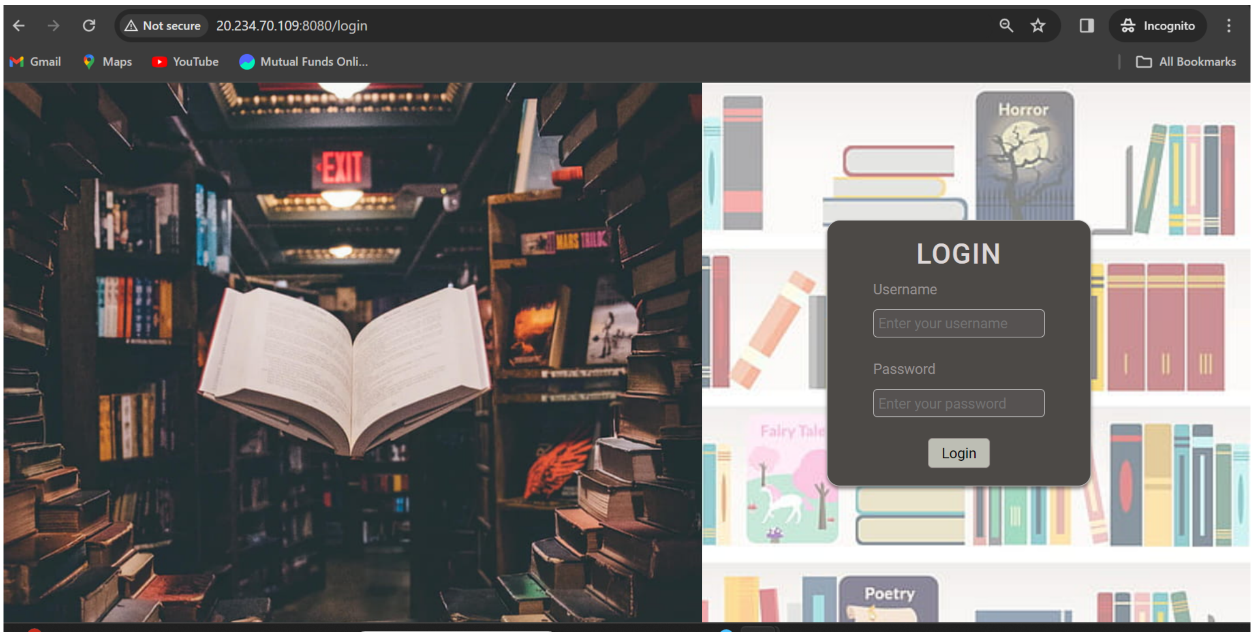
- Users can only check their own issued book history and can change their user details as well. In Addition can check their total due amount.

- **LIBRARIAN (ADMIN) functionality**

- Only ADMIN can add new users and books to the system. Additionally Admin has authority to delete book , editbook, deleteUser, IssueBook, ReturnBook and PayDue for User.
- Users can only check their own issued book history, in addition can change their user details as well.
- Users can issue max 3 books at a time, to issue 4th book users need to return one book from issued 3 books.
- Users can issue1 book at once, if they try to issue the same book again an error occurs.
- Users need to mention the number of days for which they need to issue a book, if the user returns the book after the mentioned days. Users will be fined a due amount of 1 EURO for each day.
- Admin can pay due from the admin dashboard for users.

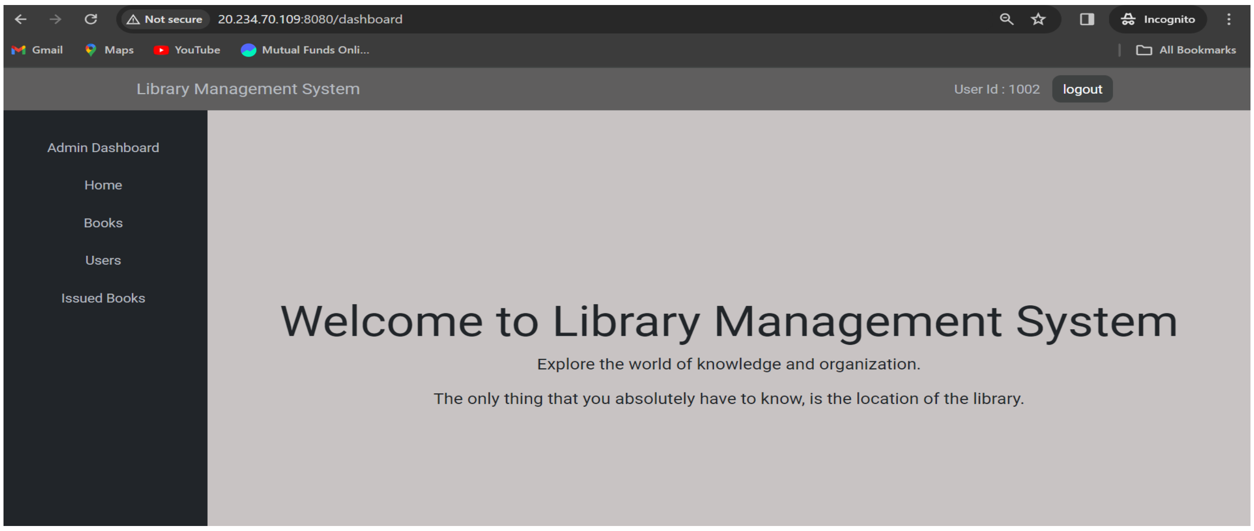
- **UX Design :**

Login form-
Below is our login page for both Librarian(Administrator) and end user as well that authenticates the details and logs in accordingly.
The username and password is provided by the admin. The password is encrypted using Werkzeug library's generate_password_hash function.



- Homepage (Post login screen)

Below is our homepage for Admin(Librarian) login.
Admin dashboard has functionality to manage Books, users and issuance of the books .



Welcome to Library Management System

Explore the world of knowledge and organization.

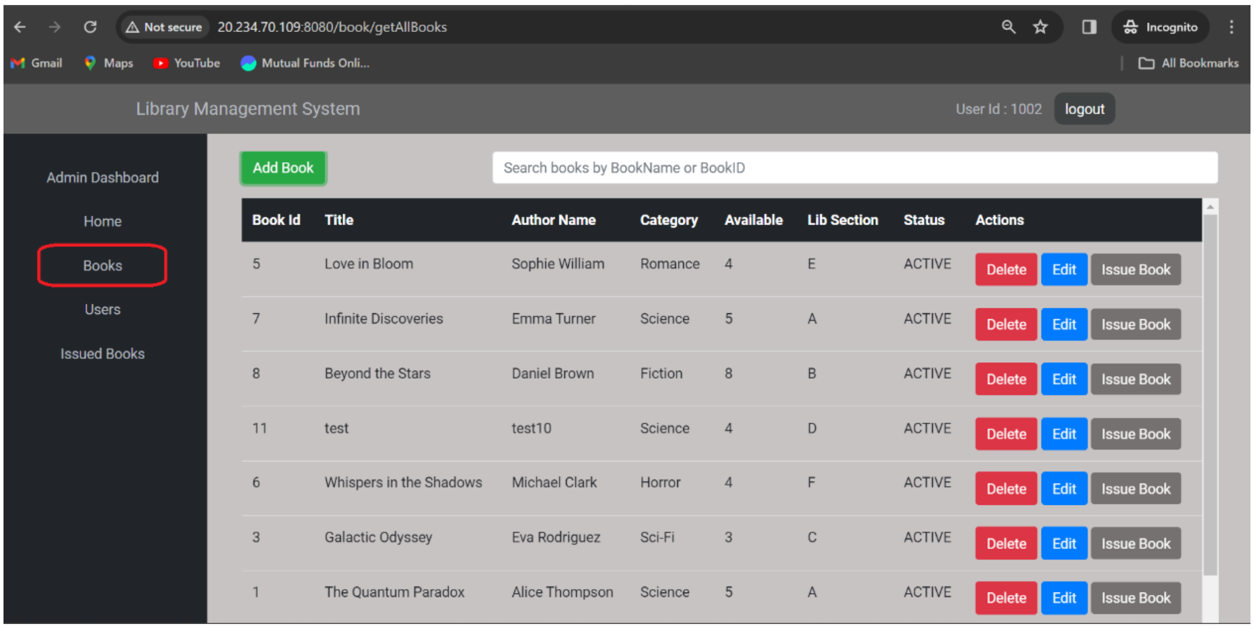
The only thing that you absolutely have to know, is the location of the library.

1. **Books tab-**

   Using this tab admin can perform below task-

   - **Add Book** -

     Add a new Book by entering all the mandatory fields in it as shown in below figures.

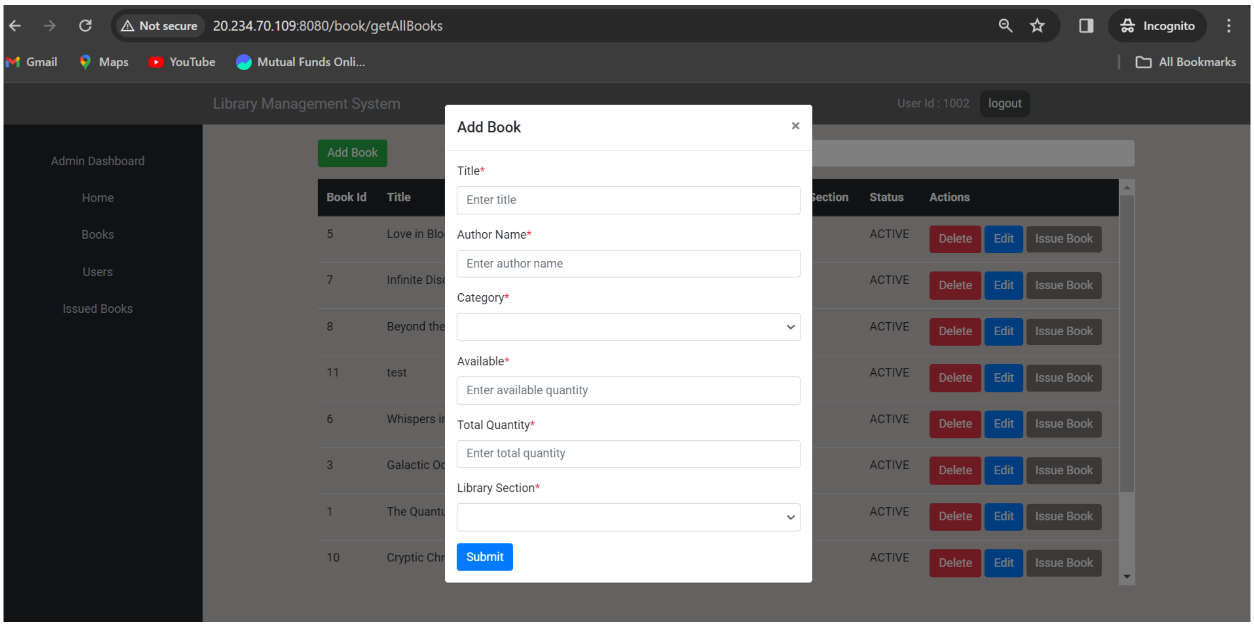

Not secure 20.234.70.109:8080/book/getAllBooks

Library Management System User Id : 1002 logout

Admin Dashboard · Home · Books · Users · Issued Books

Add Book · Search books by BookName or BookID

Book Id	Title	Author Name	Category	Available	Lib Section	Status	Actions
5	Love in Bloom	Sophie William	Romance	4	E	ACTIVE	Delete Edit Issue Book
7	Infinite Discoveries	Emma Turner	Science	5	A	ACTIVE	Delete Edit Issue Book
8	Beyond the Stars	Daniel Brown	Fiction	8	B	ACTIVE	Delete Edit Issue Book
11	test	test10	Science	4	D	ACTIVE	Delete Edit Issue Book
6	Whispers in the Shadows	Michael Clark	Horror	4	F	ACTIVE	Delete Edit Issue Book
3	Galactic Odyssey	Eva Rodriguez	Sci-Fi	3	C	ACTIVE	Delete Edit Issue Book
1	The Quantum Paradox	Alice Thompson	Science	5	A	ACTIVE	Delete Edit Issue Book


Not secure 20.234.70.109:8080/book/getAllBooks

Add Book

Title*
Enter title

Author Name*
Enter author name

Category*

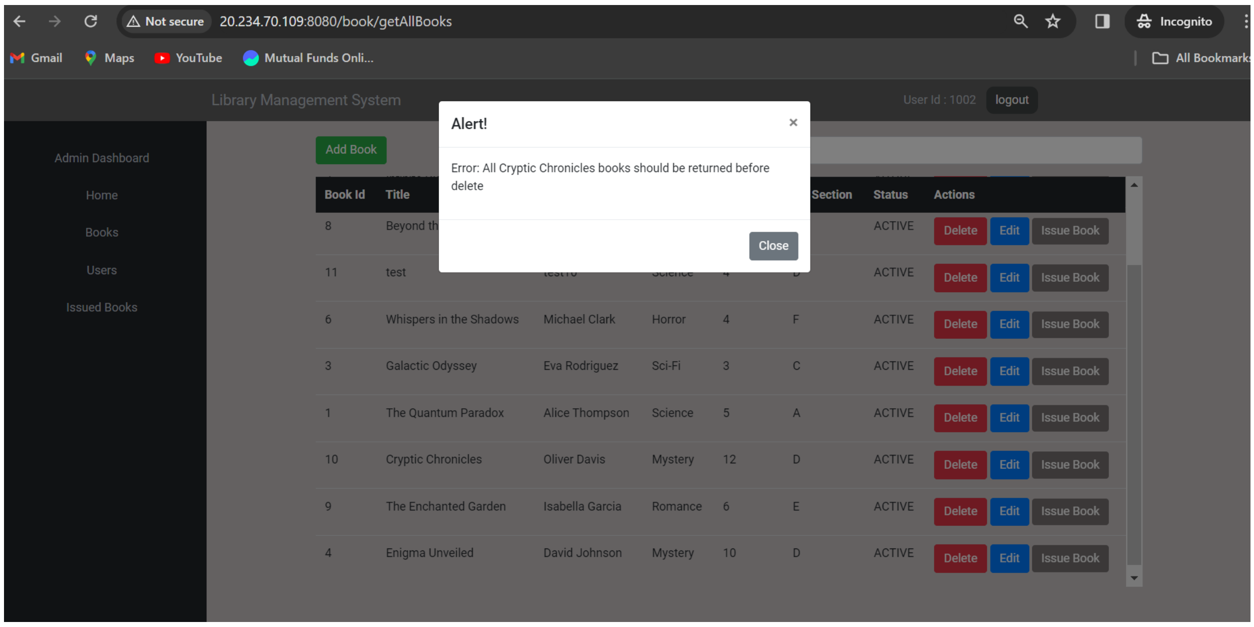
Available*
Enter available quantity

Total Quantity*
Enter total quantity

Library Section*

Submit

- **Delete**- Admin can only Delete the book if the same is not issued to any user or returned (if issued to user) or it will give a prompt as below



Not secure 20.234.70.109:8080/book/getAllBooks

Library Management System

User Id : 1002

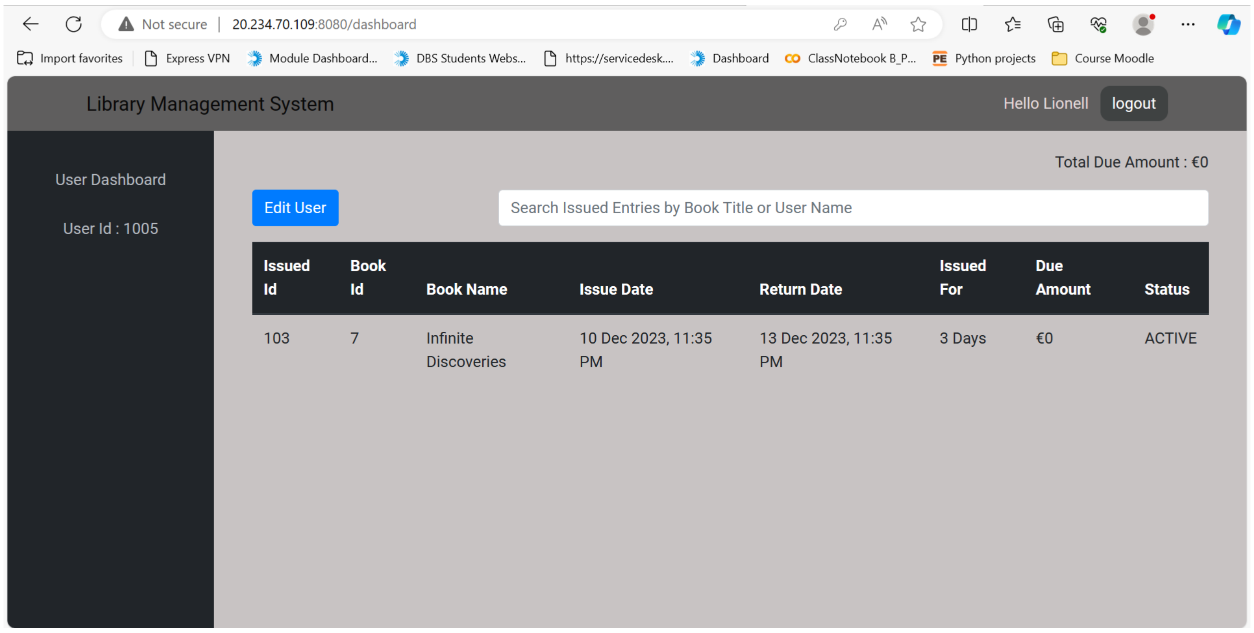
Alert!

Error: All Cryptic Chronicles books should be returned before delete

Book Id	Title				Section	Status	Actions
8	Beyond th					ACTIVE	Delete Edit Issue Book
11	test	test10	Science	4	D	ACTIVE	Delete Edit Issue Book
6	Whispers in the Shadows	Michael Clark	Horror	4	F	ACTIVE	Delete Edit Issue Book
3	Galactic Odyssey	Eva Rodriguez	Sci-Fi	3	C	ACTIVE	Delete Edit Issue Book
1	The Quantum Paradox	Alice Thompson	Science	5	A	ACTIVE	Delete Edit Issue Book
10	Cryptic Chronicles	Oliver Davis	Mystery	12	D	ACTIVE	Delete Edit Issue Book
9	The Enchanted Garden	Isabella Garcia	Romance	6	E	ACTIVE	Delete Edit Issue Book
4	Enigma Unveiled	David Johnson	Mystery	10	D	ACTIVE	Delete Edit Issue Book

- **Edit** - Edit the book details.

- User homepage(Dashboard post logging in using their credentials)



Not secure | 20.234.70.109:8080/dashboard

Library Management System

Hello Lionell

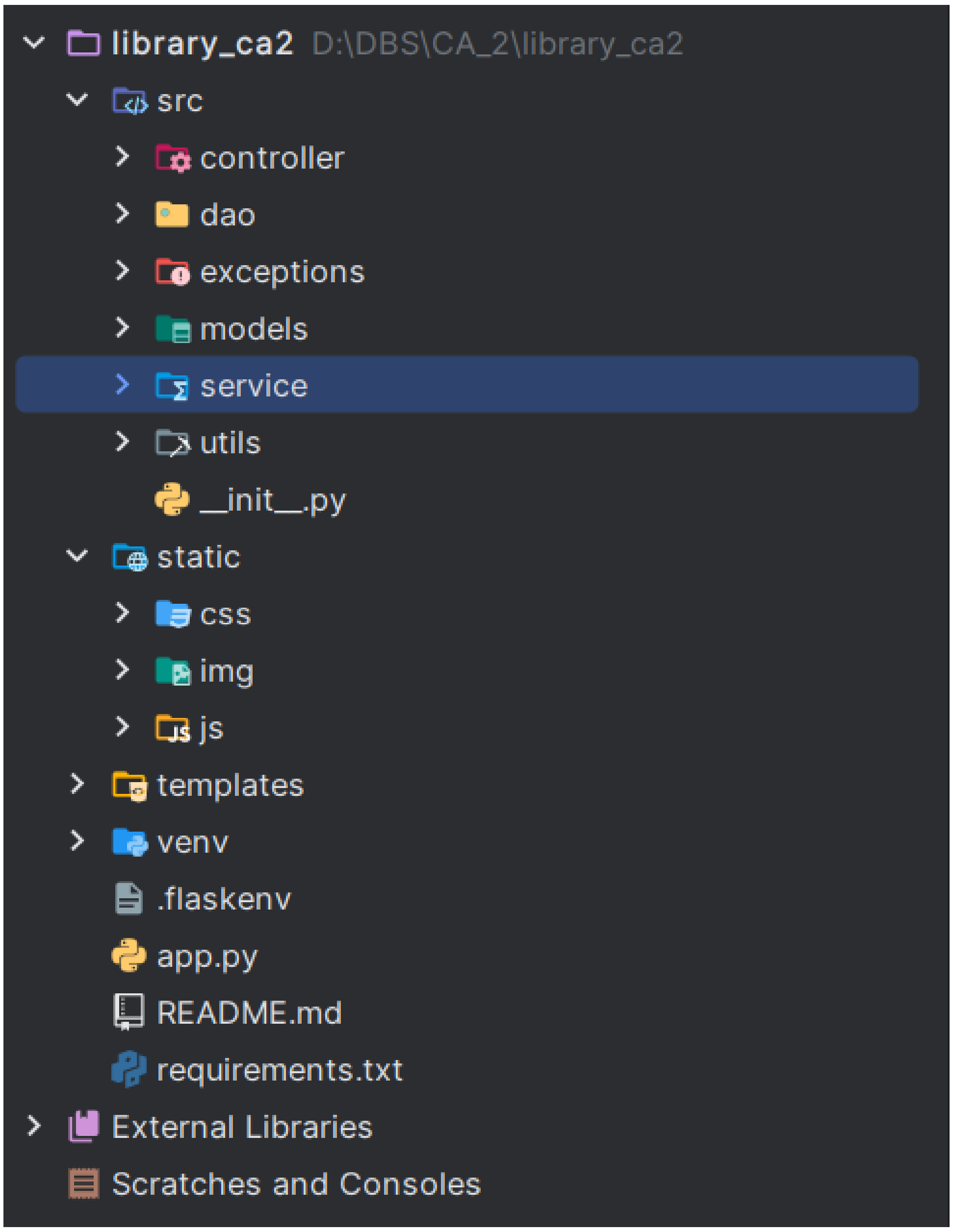
User Id : 1005

Total Due Amount : €0

Issued Id	Book Id	Book Name	Issue Date	Return Date	Issued For	Due Amount	Status
103	7	Infinite Discoveries	10 Dec 2023, 11:35 PM	13 Dec 2023, 11:35 PM	3 Days	€0	ACTIVE

Backend

Folder Structure-


```
library_ca2  D:\DBS\CA_2\library_ca2
  src
    controller
    dao
    exceptions
    models
    service
    utils
    __init__.py
  static
    css
    img
    js
  templates
  venv
  .flaskenv
  app.py
  README.md
  requirements.txt
External Libraries
Scratches and Consoles
```

1- **Controller Layer**:- Controllers are in charge of managing UI (User Interface) user input. This data may originate from button clicks, form submissions, URL parameters, or other sources.
2. **Dao Layer**: - Data Access Object is referred to as DAO. It is used to retrieve and store data in db.
3. **Service Layer: -** Between the controller and the data access layer, this is an important component known as the service layer. It is to manage the business logic and application-specific functionality found in the service layer.
4. **Static** - All static files will be stored here like CSS files, Images and javascript files.

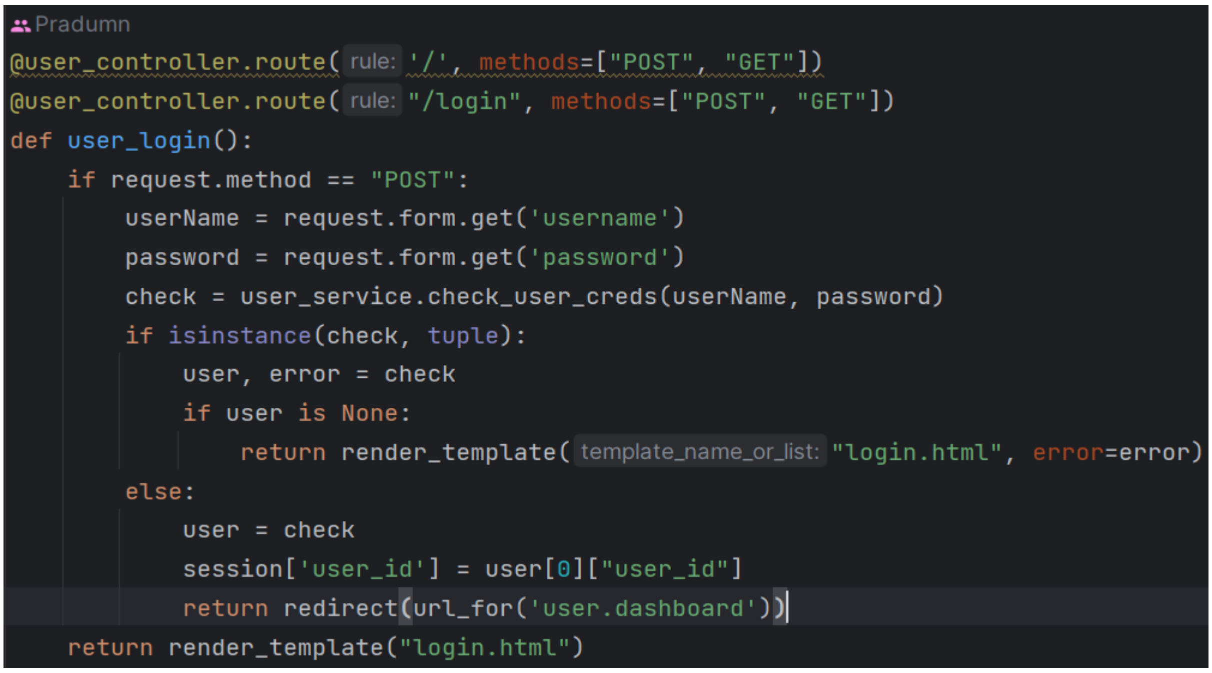
5. **Templates** - Templates like html files will be stored here so that flask can search .

API List

There is a specific route connected to each screen. Let's examine the APIS list.

- **Login page - /login**

  - Every User needs to login with correct credentials, /login api will be used for the login page, once credentials are validated users will be redirected to the respective dashboard based on their role.
  - The application's backend logic prevents the user from accessing this page if they are logged out and try to access the URL "/dashboard."



```
Pradumn
@user_controller.route( rule: '/', methods=["POST", "GET"])
@user_controller.route( rule: "/login", methods=["POST", "GET"])
def user_login():
    if request.method == "POST":
        userName = request.form.get('username')
        password = request.form.get('password')
        check = user_service.check_user_creds(userName, password)
        if isinstance(check, tuple):
            user, error = check
            if user is None:
                return render_template( template_name_or_list: "login.html", error=error)
        else:
            user = check
            session['user_id'] = user[0]["user_id"]
            return redirect(url_for('user.dashboard'))
    return render_template("login.html")
```

- The user must have a valid session in order to access this screen; otherwise, they will be redirected to the login page.
- Session should have user_id which gets added once credentials are validated.

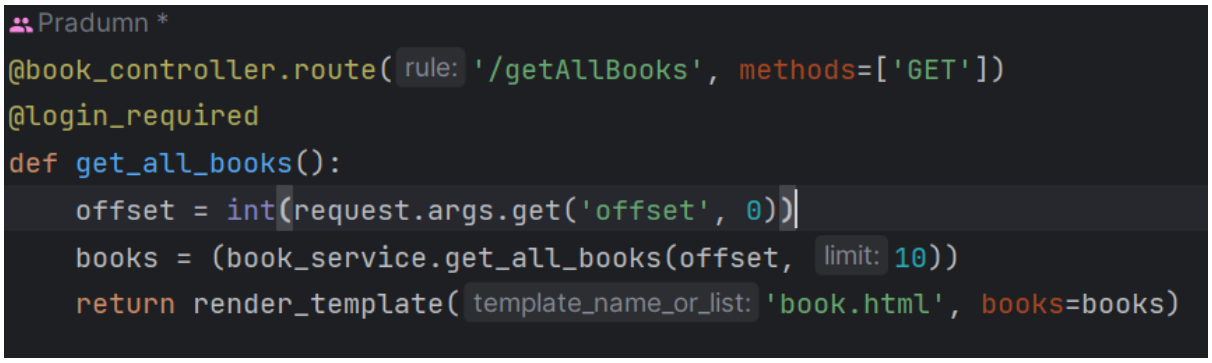
- **Admin Dashboard**
  - We have created a Librarian Dashboard for ‘ADMIN’ role which basically has all rights to add User, delete user and Edit User details.
  - Admin is also responsible for adding, issuing and returning books in the system.

- **User Dashboard**
  - User Dashboard for users with role ‘USER’.
  - Users can edit their details and check book issue history.
  - Users can check their due amount on their dashboard but can’t pay , they need to contact admin to pay the due amount.

1-Books -

- `/getAllBooks`

  This api helps to retrieve all books present in the system, it returns a html file consisting of search functionality, tabular books representations and actions for each book.



```
Pradumn *
@book_controller.route( rule: '/getAllBooks', methods=['GET'])
@login_required
def get_all_books():
    offset = int(request.args.get('offset', 0))
    books = (book_service.get_all_books(offset,  limit: 10))
    return render_template( template_name_or_list: 'book.html', books=books)
```

- `/addBook`

  This api adds a new book to the system**.**

- `/deleteBook`

  This api deletes book from the system. It also validates that all books be returned before book deletion.

- `/updateBook`

  This api updates existing books with title change ,books availability etc.

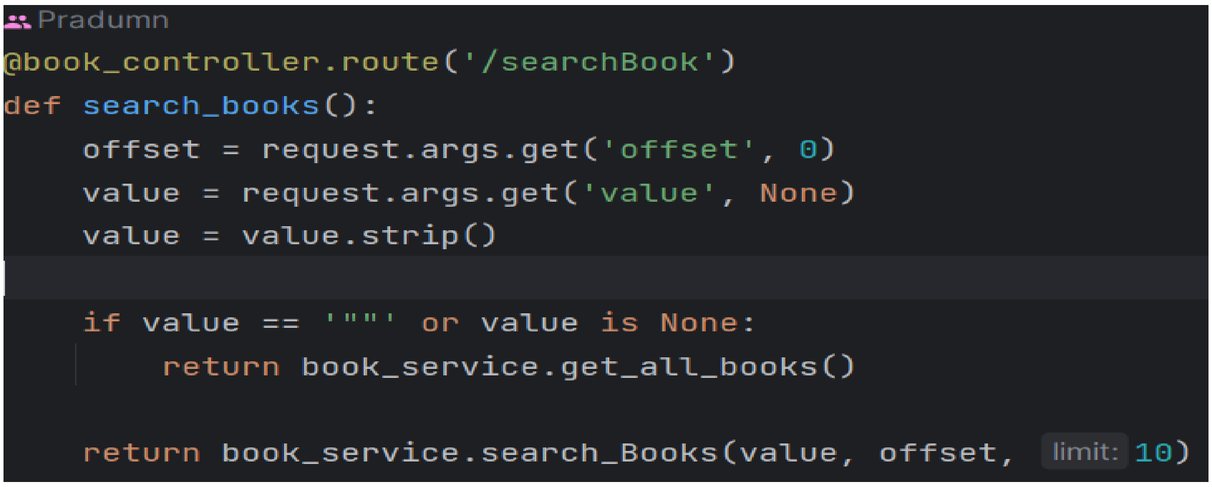
- `/getBookById`

  This api returns book details for bookId passed.

- `/searchBooks?value=''`

  -Api returns all books which match the passed value with the book title.

  -Search uses wildcard search with appending '%' on value.

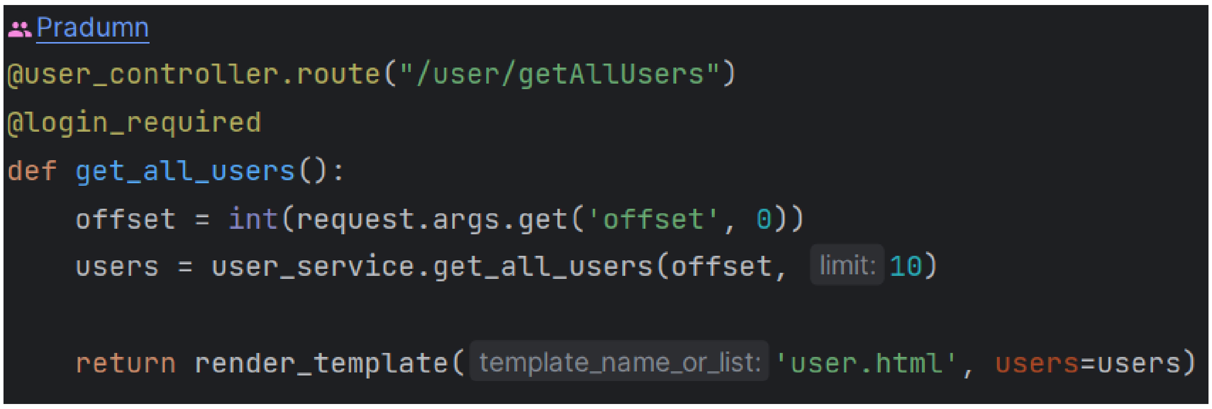


```
Pradumn
@book_controller.route('/searchBook')
def search_books():
    offset = request.args.get('offset', 0)
    value = request.args.get('value', None)
    value = value.strip()

    if value == '""' or value is None:
        return book_service.get_all_books()

    return book_service.search_Books(value, offset,  limit: 10)
```

2.Users

- **/getAllUsers**

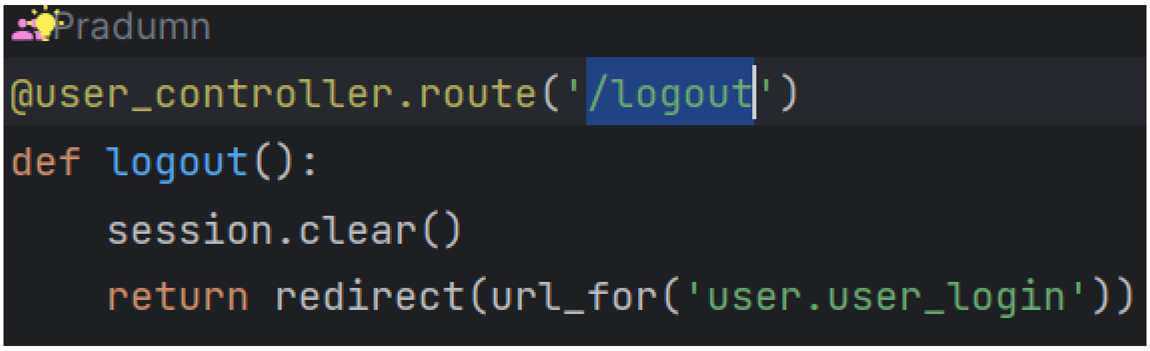
  This api helps to retrieve all Users present in the system, it returns a html file consisting of search functionality, tabular books representations and actions for each User eg- Delete, add,edit User and Pay due Amount.



```
Pradumn
@user_controller.route("/user/getAllUsers")
@login_required
def get_all_users():
    offset = int(request.args.get('offset', 0))
    users = user_service.get_all_users(offset,  limit: 10)

    return render_template( template_name_or_list: 'user.html', users=users)
```

-**/logout**

  This api will clear userId from session, and redirect user to login page.



```
Pradumn
@user_controller.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('user.user_login'))
```

- **/addUser**

  This api adds a new User to the system.

- **/deleteUser**

  This api deletes users from the system.

- **/updateUser**

  This api updates existing books with username, email name change etc.

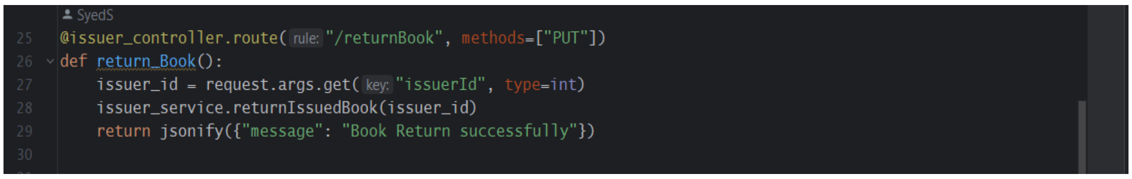
- **/searchUsers?value=''**

  This api returns all Users which match the passed value with the user name or user Id.

3.Issuers

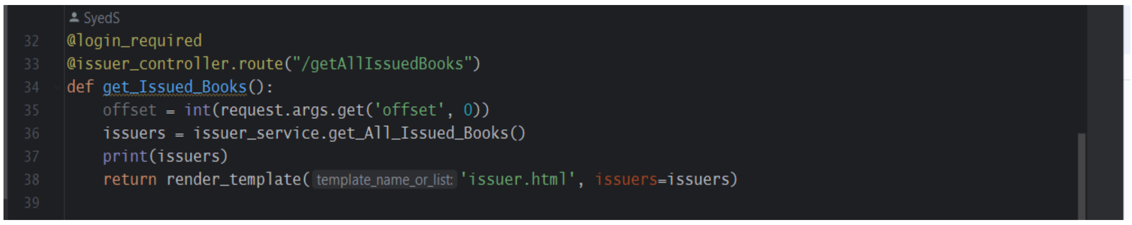
- **/issueBook :**
  This API will help user to issue book and an admin can view the issued

- **/return Book**
  Return Book API Will Help the admin return the book and add the book in the total availability of book.



```
SyedS
@issuer_controller.route( rule: "/returnBook", methods=["PUT"])
def return_Book():
    issuer_id = request.args.get( key: "issuerId", type=int)
    issuer_service.returnIssuedBook(issuer_id)
    return jsonify({"message": "Book Return successfully"})
```

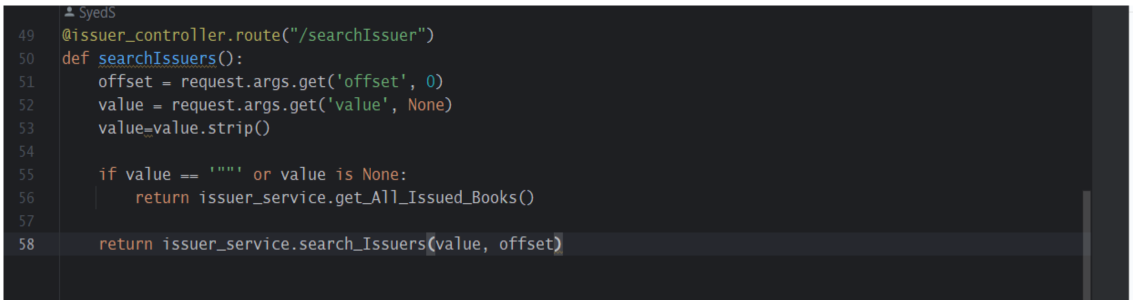
- **/returnDue**
  With the help of return due API an admin can put the due amount of the user



```
SyedS
@login_required
@issuer_controller.route("/getAllIssuedBooks")
def get_Issued_Books():
    offset = int(request.args.get('offset', 0))
    issuers = issuer_service.get_All_Issued_Books()
    print(issuers)
    return render_template( template_name_or_list: 'issuer.html', issuers=issuers)
```

- **/getAllissuedBooks**
  This API will show HTML Consisting of a list of all issued book that have been issued by different user .

- **searchIssuer**
  We can search the issuer history by the User name or By the Book Title that an user has issued .



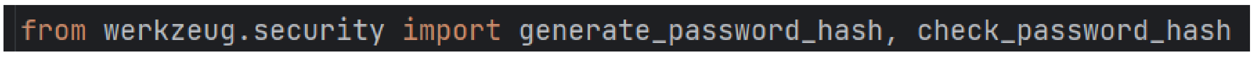
```
SyedS
@issuer_controller.route("/searchIssuer")
def searchIssuers():
    offset = request.args.get('offset', 0)
    value = request.args.get('value', None)
    value=value.strip()

    if value == '""' or value is None:
        return issuer_service.get_All_Issued_Books()

    return issuer_service.search_Issuers(value, offset)
```

Security

Management of sessions

-Pages that require security are session protected, and there are flask libraries that make managing sessions much easier. Therefore, the user will not be able to access the pages if they try to access them while not logged in.
-Once User logout session will be clear, so that to access the page user needs to re login.

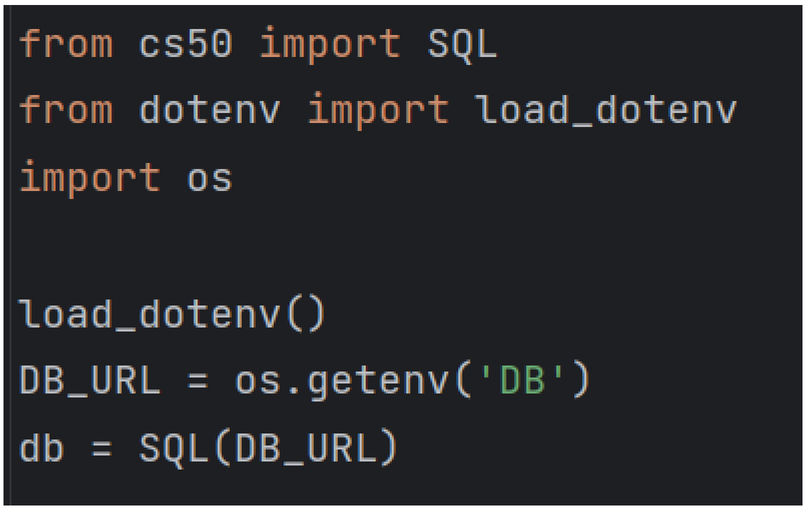
Hashing Password



```
from werkzeug.security import generate_password_hash, check_password_hash
```

Before storing passwords in the database, we securely hash them using the Werkzeug library's generate_password_hash function. By guaranteeing that the stored passwords are not exposed in their original form even in the case of a potential data breach, this practice improves the security of our application. Rather, only their hashed representations are kept in the database, which keeps user credentials private and adds an additional degree of security against unwanted access.

Storing DB details in .ENV file. Instead of In Application



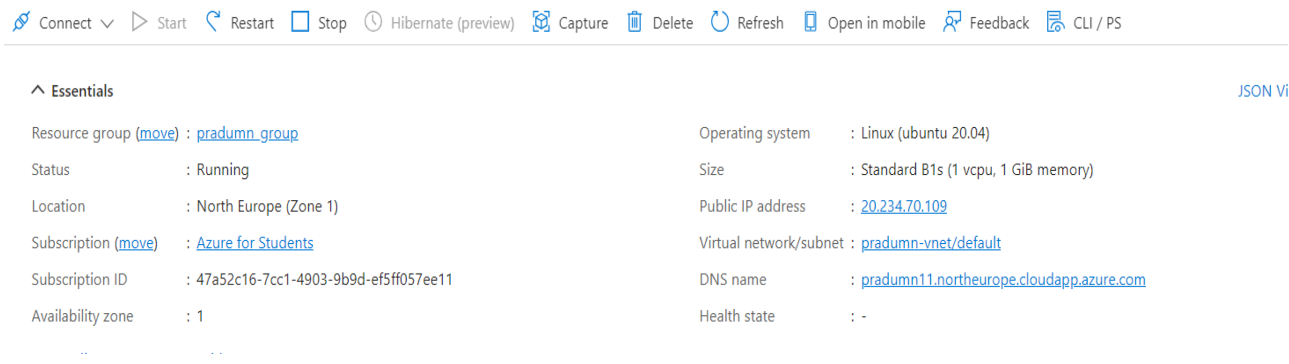
```
from cs50 import SQL
from dotenv import load_dotenv
import os

load_dotenv()
DB_URL = os.getenv('DB')
db = SQL(DB_URL)
```

-Created a string with db details-server, password, port etc, and stored in .env file which will be added in .gitignore file. So that wont be pushed to application for security.
-DB string will be loaded in the application using os library.

Deployment-

1- Application Deployment

-Microsoft offers a full-featured cloud computing platform called Microsoft Azure.
- We deployed our application on Microsoft Azure Virtual Machine.
-Created VM instance which we connected through ssh authentication.
-Configured Vm to listen on 8080 inbound requests. As our application will be running on port 8080 and Cloned the repository in the VM and ran the application.



Resource group (move)	: pradumn_group	Operating system	: Linux (ubuntu 20.04)
Status	: Running	Size	: Standard B1s (1 vcpu, 1 GiB memory)
Location	: North Europe (Zone 1)	Public IP address	: 20.234.70.109
Subscription (move)	: Azure for Students	Virtual network/subnet	: pradumn-vnet/default
Subscription ID	: 47a52c16-7cc1-4903-9b9d-ef5ff057ee11	DNS name	: pradumn11.northeurope.cloudapp.azure.com
Availability zone	: 1	Health state	: -

2- Database Deployment

- Created a Azure Database for PostgreSQL flexible server for database.
- Configured database in such a way that any public ip can connect.
- Added database details like username, password, url of server and port in Application Database config file.

Application Links

Git Link– https://github.com/Pradumn11/library_ca2.git

Postman collections-
https://drive.google.com/file/d/1cNJIZ0eZ_3f4X_VYG-bAT_ntxisEKjxA/view?usp=sharing

Conclusion :

To sum up, a combination of Python, Flask, JavaScript, Bootstrap, CSS, HTML, PostgreSQL, and Microsoft Azure was implemented in the Library Management System to build a dependable, approachable, and expandable platform. A visually appealing user interface, effective backend operations, data integrity, and global accessibility were made possible by this tech stack. All things considered, these technologies combined together to produce a powerful system that puts user experience, data security, and efficient library administration first.

References

Youtube.com - https://www.youtube.com/watch?v=37hHjWF3a0k&t=166s

Youtube.com - CS50 - https://www.youtube.com/watch?v=oVA0fD13NGI&t=6855s

https://www.moesif.com/blog/technical/api-development/Building-RESTful-API-with-Flask/

https://getbootstrap.com/docs/5.3/getting-started/introduction/

https://www.postgresqltutorial.com/postgresql-getting-started/what-is-postgresql/

https://copyprogramming.com/howto/convert-2021-01-18t11-18-10-833876-00-00-to-datetime-python

Individual Contribution

Pradumnya Kailas Zendphale - Student Number- 20003538

- My primary responsibility in the library management system project is to carefully plan and develop the essential features that control user and book management. I created reliable APIs like searchBook/User, add Book/User, Edit Book/User etc with distinct layers, including a Data Access Object (DAO) layer and a Service layer, for both books and users. I used the dao layer to access all data from the database and service to add business logic and validation on each update and add functionality. -Validation Eg- User can't issue more than 3 Books.

- Added Search functionality for Users and Books by username, userID and Book-title, bookId respectively.

- I included password hashing algorithms to secure user password information and prevent potential breaches of sensitive data and put in authentication checks hashed passwords at login. I made steps to stop page caching in order to improve security by making sure that once a user logs out he should not be able to go back to the page he was before logout.

- In addition, I developed HTML and JavaScript files to communicate with the APIs to offer a dynamic and intuitive book and user management interface. Furthermore, I wrote utils methods like login_required, checkUniqueOrThrow etc which are useful throughout the application.

- For DB credentials security included .env file to fetch db details from file instead of storing it in git repo.

- Deployed application on azure using Vm and created database on Azure Database for PostgreSQL flexible server.

Mohammed Salik Syed - Student Number : 20020194

- My role in the part of this assignment was to primarily focus on adding the Issuer Section and the Issuer functionality part of the library management system . Developing this functionality was a prime and important aspect of our assignment

- I added the Issuer ID and Book ID and the time that he or she has issued the book that has been implemented by myself . Return functionality is added so that whenever the user will return the book the admin can opt for the option of return book and the as per the system the availability of that particular book will increase as the user has returned the particular book .

- In the case of the user failing to submit a book as per the given time by him or her, due and fine functionality has been added by me . The time that a user delays to submit will result in the fine and that fine and due payment will be integrated . I have provided the ease of paying the due by adding functionality that if a user wants to pay the due in installments the admin can accept the payment by half or by full as per the user wishes to pay the due .

- Additionally, I created HTML and JavaScript files to connect with the APIs and provide a dynamic and user-friendly book and user management system.

Pratik Shivraj Poojari - Student Number - 20000871

- My contribution to the project is the creation of a User dashboard using HTML, CSS and JavaScript where the user is redirected to their dashboard after validating the role of the user.
- It involves enhancing the user experience and functionality of the web application's user management system.
- I implemented multiple JavaScript functions that facilitate user interaction with the application's user management feature. One of these functions retrieves user details based on their ID, populates the user editing form with the retrieved data, and enables through a modal and provides real-time search results for better data accessibility.
- The form validates to ensure mandatory fields are filled before submitting updates to the server.
- Resolved various bugs with respect to validations and web responsiveness.
- Performed regression testing to ensure the Stability and functions usability correctness.
- Contributed in deployment of the Azure Database for PostgreSQL flexible server.
- I have also performed peer review for the overall functionality code executed by my teammates.