Library Management System using Flask, SQLite and Streamlit UI

Submitted by

Group 7

Tushar Yadav (885174284)
Abhishek Singh (885199042)

Submitted to

CPSC 449 – Web Backend Engineering

Sai Supreeth Kolaparthy

Dept. of Computer Science - California State University Fullerton

April 14, 2024

Table of Contents

Introduction

The Library Management System (LMS) is a web-based application designed to manage the operations of a library, allowing users to register, login, and manage book borrowings and returns through an intuitive interface. This system simplifies the tracking of book inventory and user interactions, making it easier for libraries to maintain up-to-date records of their collections and user activities.

Technologies Used

Backend

- Flask: A lightweight WSGI web application framework in Python, used to build the backend API.
- SQLite: A relational database management system contained in a C library that provides a lightweight disk-based database.
- SQLAlchemy: An SQL toolkit and Object-Relational Mapping (ORM) system for Python, which provides a high-level interface for database operations.
- CORS (Cross-Origin Resource Sharing): Enabled in Flask to allow the frontend hosted on a different domain to interact seamlessly with the backend.

Frontend

- Streamlit: An open-source app framework for Machine Learning and Data Science projects, used here to create a clean, straightforward user interface for the LMS.

Database Schema

- User Model:
  - **id**: Integer, Primary Key
  - **username**: String (50 characters), Unique, Not nullable
  - **password**: String (50 characters), Not nullable

- Book Model:
  - **id**: String (50 characters), Primary Key
  - **title**: String (100 characters)
  - **author**: String (100 characters)
  - **status**: String (20 characters), default "available"

API Endpoints

S. No.	What it Does	Endpoint	Request Type	Request Payload	Request Response
1	**User Registration**	/register	POST	{ "username": "string", "password": "string" }	{ "message": "User registered successfully" }
					{ "message": "User already exists" }
2	**USER LOGIN**	/login	POST	{ "username": "string", "password": "string" }	{ "message": "Login successful" }
					{ "message": "Invalid credentials" }
3	**Add Book**	/books	POST	{ "book_id": "string", "book_info": { "title": "string", "author": "string", "status": "string" } }	{ "message": "Book added successfully" }

4	**Get All Books**	/books	GET		{
					"book_id": {
					"title": "string",
					"author": "string",
					"status": "string"
					},
					"book_id": {
					"title": "string",
					"author": "string",
					"status": "string"
					},
					...
					}

5	**Issue Book**	/issue	POST	{
				"book_id": "string"
				}
					{
					"message": "Book issued successfully"
					}

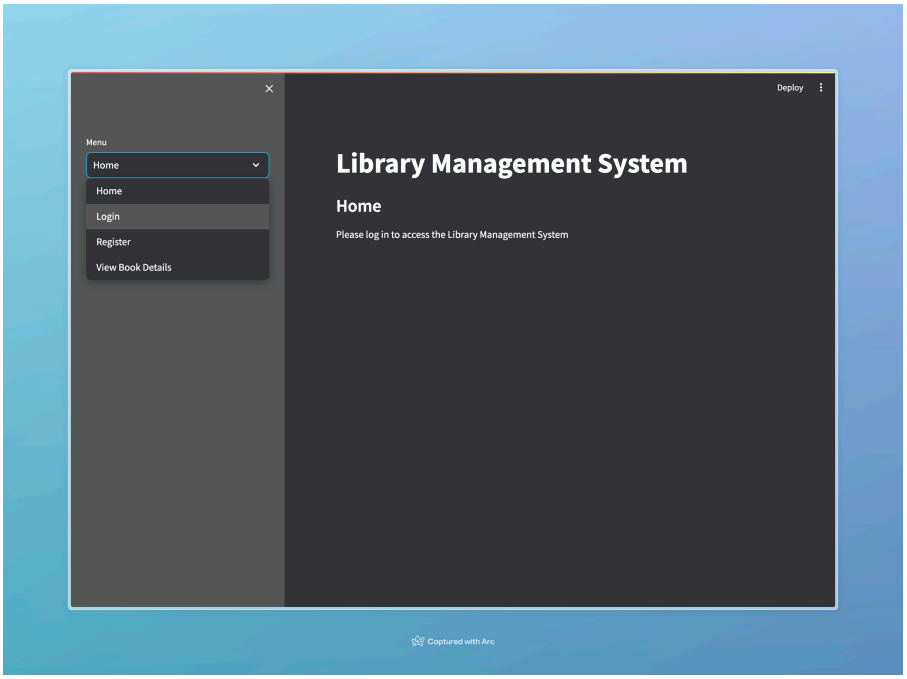
					{
					"message": "Book not available"
					}

6	**Return Book**	/return	POST	{
				"book_id": "string"
				}
					{
					"message": "Book returned successfully"
					}

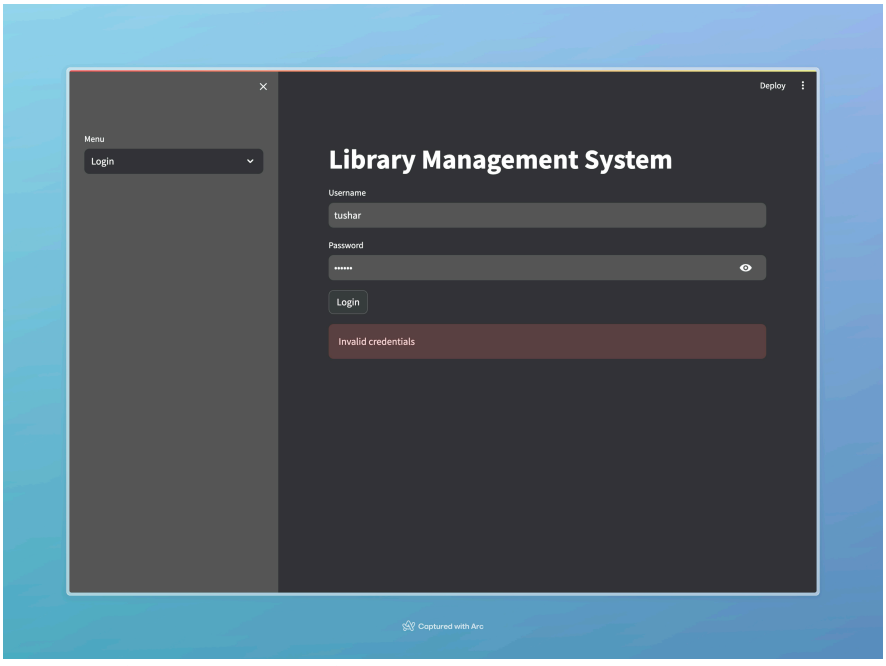
					{
					"message": "Invalid return"
					}

User Interface

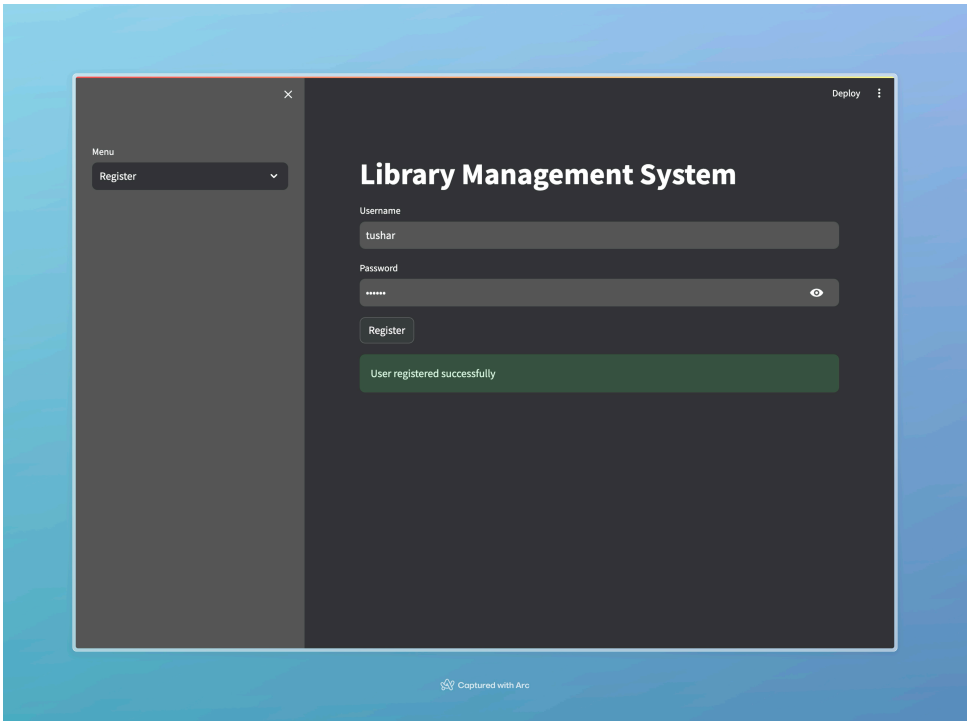
Below are some of the screenshots from the applications depicting some of the major functionalities of the Library Management System.



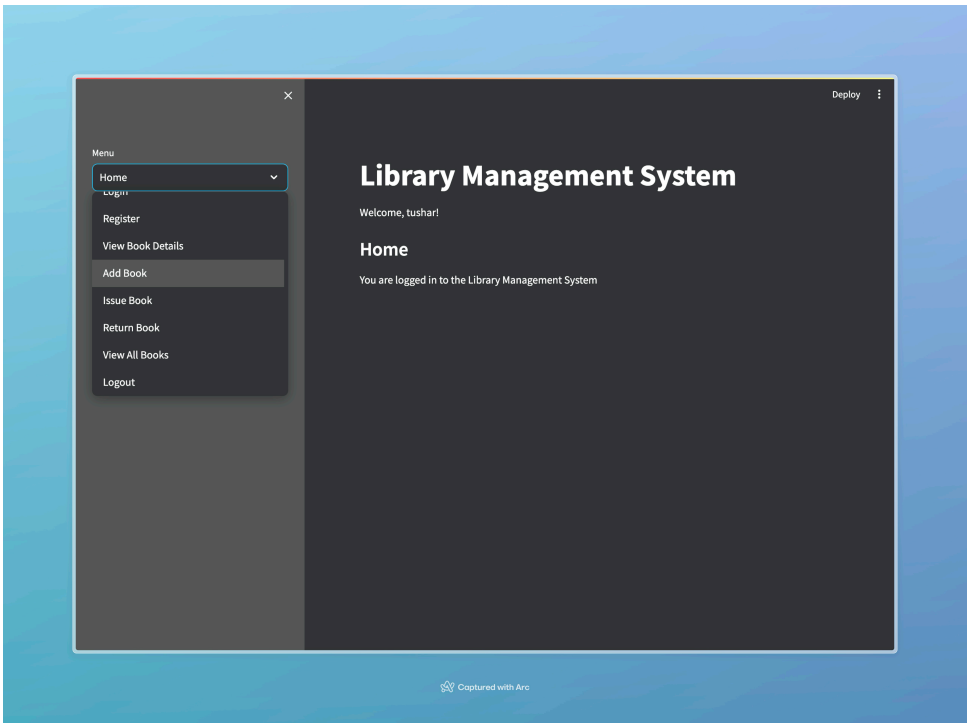
a. Home Screen



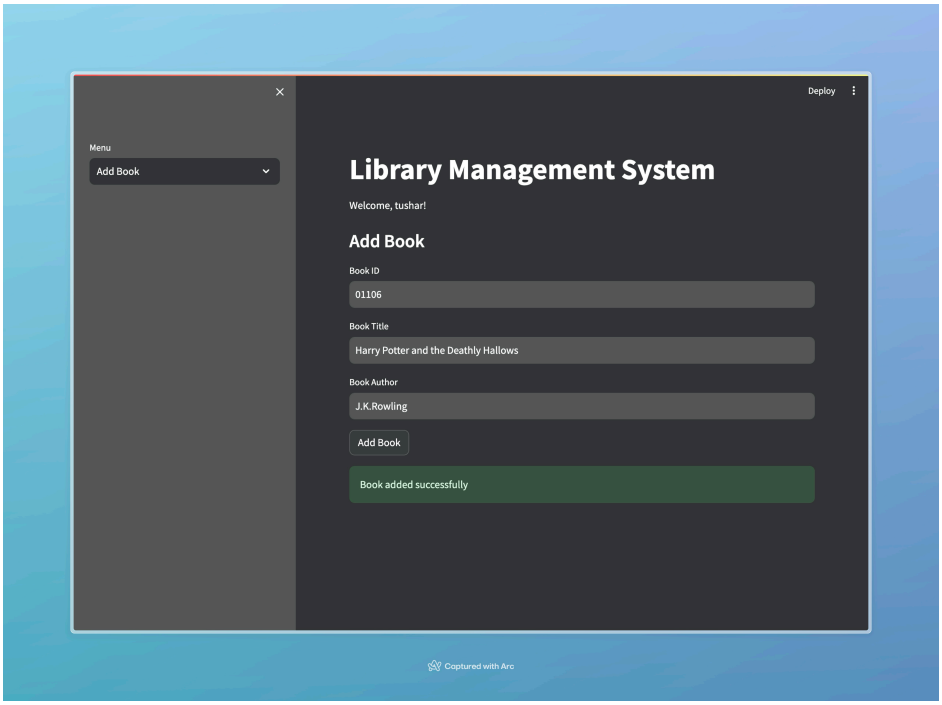
b. Logging in with invalid credentials



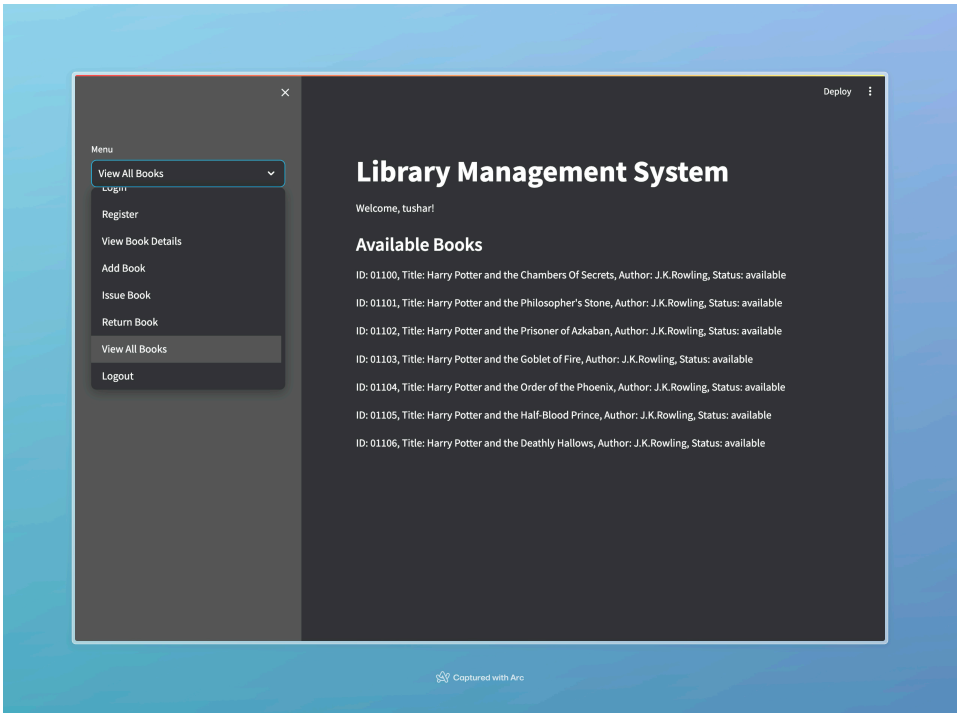
c. Registering a new user



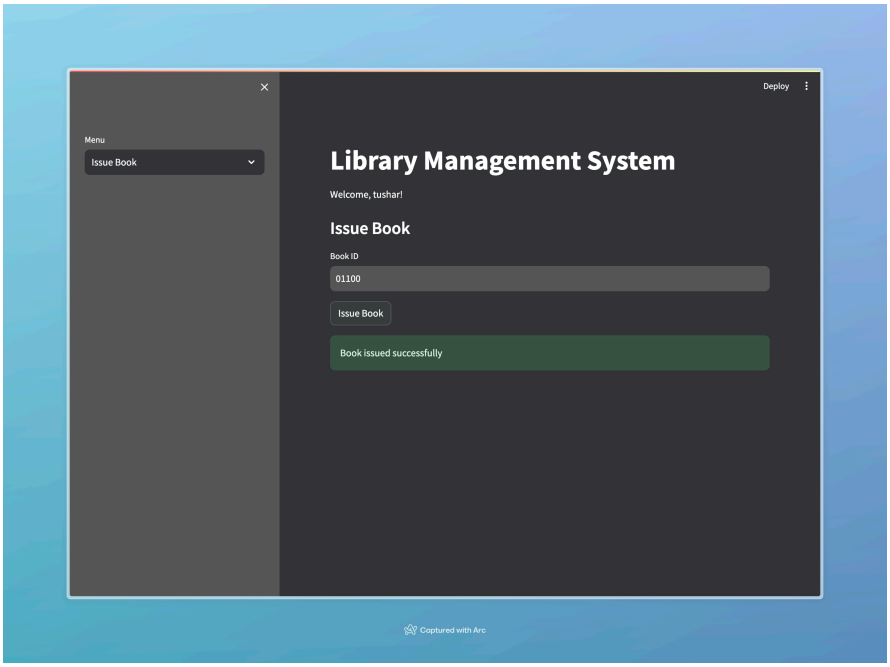
d. Home screen After logging in successfully



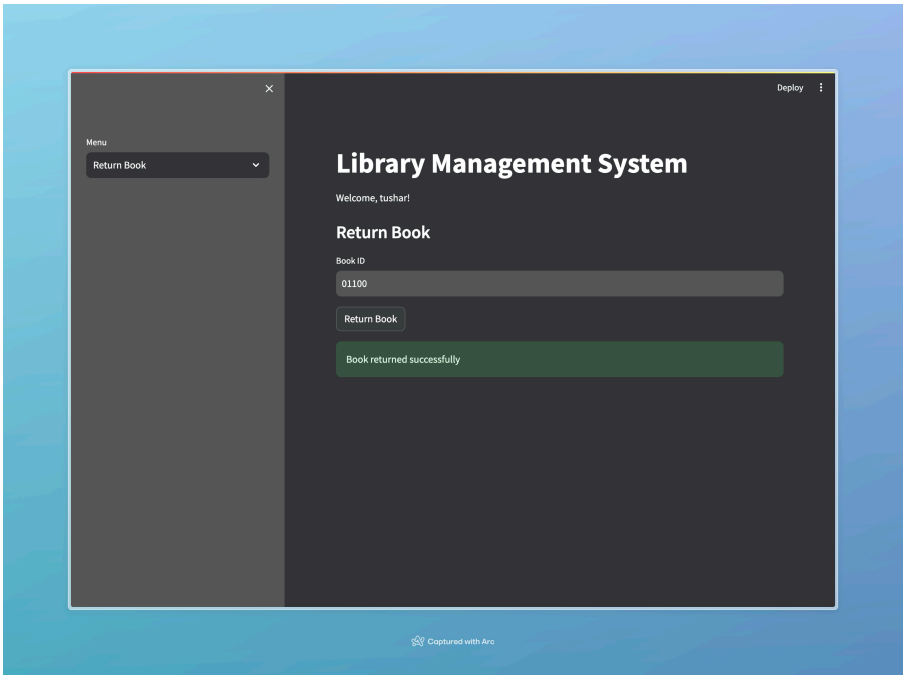
e. Adding a new book to the database



f. Viewing all books in the database



g. Issuing a book to the logged in user



h. Returning the book

Future Enhancements

- ## Frontend Framework Transition:
  - **Transition to React:** Consider replacing Streamlit with React for the frontend. React's robust ecosystem and flexibility would allow for more dynamic user interfaces, improved state management, and better scalability for larger user bases.

- ## Database and Model Enhancements:
  - **Additional Database Models:** Expand the database schema to include models for Reservations, Loans, and Fines, enabling the system to handle more complex library operations such as due dates, overdue penalties, and reservation queues.
  - **Database Upgrade:** Migrate from SQLite to a more robust and scalable database system like PostgreSQL. This would support larger data volumes and more concurrent users effectively.

- ## Feature Additions:
  - **Advanced Search Capabilities:** Implement filters and advanced search options to allow users to find books by genre, publication year, or language.
  - **Notification System:** Develop a notification system to alert users about due dates, book availability, and reservation statuses via email or SMS.
  - **User Reviews and Ratings:** Allow users to rate and review books, which can help others in selecting books and assist in recommending books based on user preferences.
  - **Mobile App:** Develop a mobile app to provide users with the convenience of managing their library activities on the go.

- ## Backend Framework Upgrade:
  - **Migrate to FastAPI:** Transition the backend from Flask to FastAPI. FastAPI provides faster performance with its asynchronous support and is more suited for building APIs with modern features like automatic interactive API documentation using Swagger.

GitHub Link:

https://github.com/YTushar18/web-library-mgmt

References:

- Flask Getting Started Documentation
  - https://flask.palletsprojects.com/en/3.0.x/quickstart/
- Streamlit UI Tutorials
  - https://docs.streamlit.io/get-started/tutorials
- FLASK- CORS Documentation
  - https://flask-cors.readthedocs.io/en/latest/
- SQLite Tutorials
  - https://www.sqlitetutorial.net/sqlite-python/creating-database/