Dublin Business School, Sep 2025

Assignment Cover Sheet

Please fill out and insert as the first page of any essay-style Assignment.

Student Name(s) and Number(s) as per student card (s): Alice Ugbah 20066635 and Etuwe Osakwe 20055960

Programme: MSc in Cybersecurity

Lecturer Name: Swati Dongre

Module/Subject Title: Advanced Programming Techniques

Assignment Title: CA_TWO_

No of Words: 1095

By submitting this assignment, I am/ we are confirming that:

- **This assignment is all my/our own work;**
- **Any sources used have been referenced;**
- **I/we have followed the Generative AI instructions/ scale set out in the Assignment Brief;**
- **I/we have read the College rules regarding academic integrity in the QAH Part B Section 3, and the Generative AI Guidelines, and understand that penalties will be applied accordingly if work is found not to be my/our own.**
- **I/we understand that all work uploaded is submitted via Ouriginal, whereby a text-matching report will show any similarities with other texts.**

Note: Technical support is available to students between **0830- 2000 hrs (Mon-Fri), 0930-1630 (Sat) only**. There is no technical support after 2000 hrs. It is your responsibility to ensure that you allow time to troubleshoot any technical difficulties by uploading early on the due date.

15th December 2025

LIBRARY CATALOGUE SYSTEM REPORT

Alice Ugbah 20066635 | Etuwe Osakwe 20055960

MSc Cybersecurity | Advanced Programming Techniques

Table of Contents

Figure of Contents

Introduction

This report outlines the development of a web-based Library Catalogue System. The aim of the assignment was to construct a program using Python, demonstrating an understanding of key programming skills and concepts. The system produced allows the user to add books, view all books, search for a specific book, borrow books, return books and delete from the catalogue while ensuring all information is stored persistently using JSON files and are safely loaded and updated between application executions. The application follows a front-end and back-end structure where Flask handles the user interaction through a web interface while a separate logic unit manages the core functionalities. This report will discuss this in more detail.

System Requirements

The Library Catalogue System must provide a set of core features that allow the user to interact and manage a collection of books having the ability to add a book, view all, search, delete, borrow and return. All the data must be stored using JSON files, so the library information is preserved between program executions. Try/except blocks must be implemented to handle missing or corrupted files. The interface must operate through a Flask web interface which allows the user to select different operations through HTML forms and pages.

Program Design

The system is built using a simple modular architecture that separates presentation, logic, and data management. The main actors, the users, can view, search, borrow, and return books through a user-friendly web interface implemented with Flask. The application is organized into two main files: library_system.py, which handles all core logic operations such as adding, searching, deleting, and managing books, and app.py, which manages the web interface and routes user requests to the relevant functions. Book and borrowing records are stored in JSON files, eliminating the need for a database while keeping the system lightweight. This clear separation of concerns ensures that each layer works efficiently while maintaining code readability and modularity. The overall structure and user interactions are illustrated in the UML diagram below.

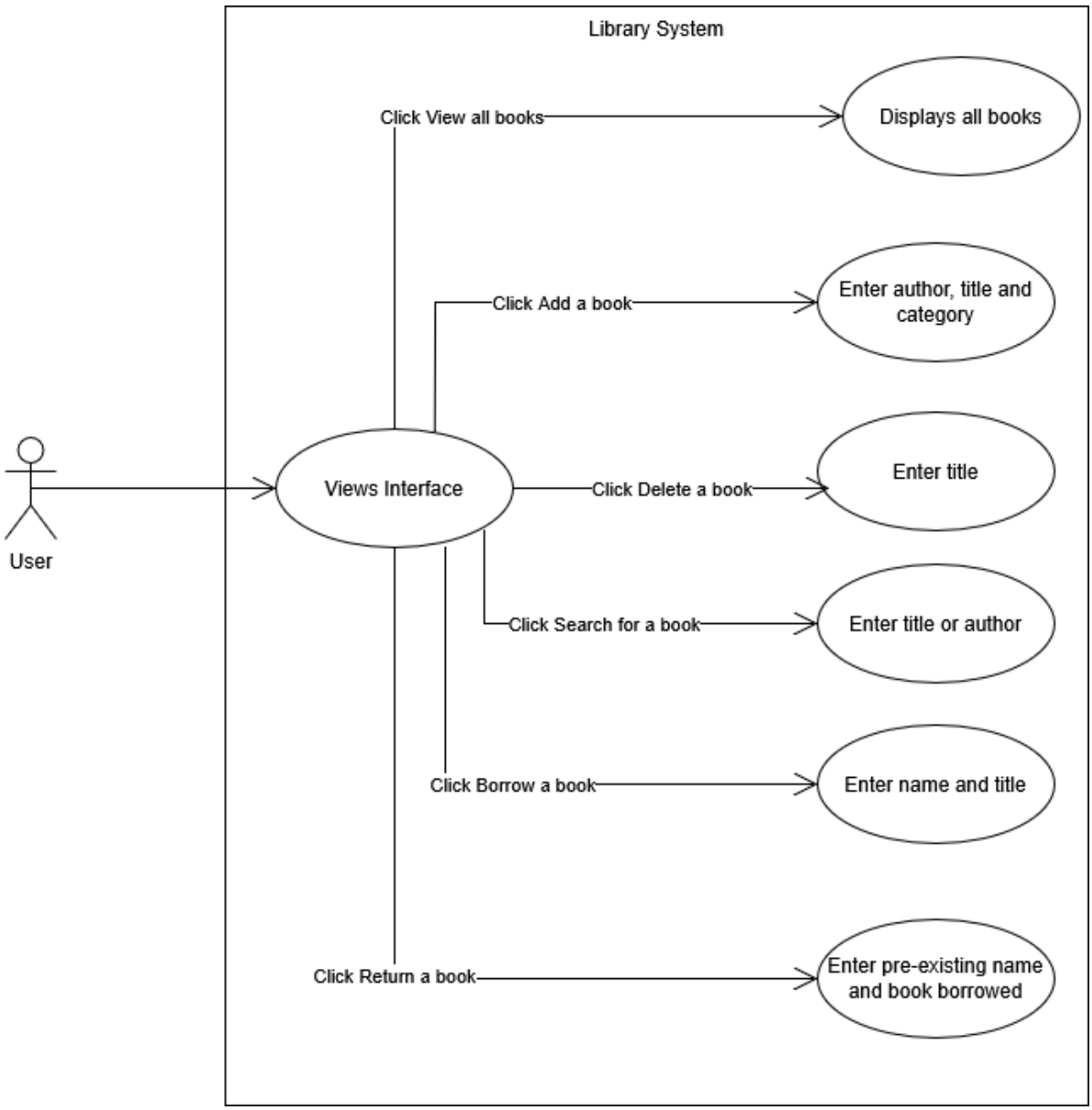

Figure 1 UML: Use Case Diagram

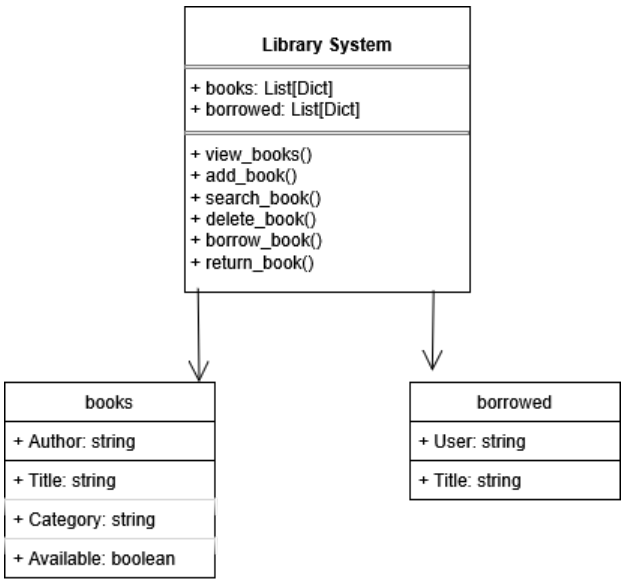

Figure 2 UML: Class Diagram

Data Structure

The system uses lists to store books and borrowed records, and dictionaries to represent each book with its title, author, category, and availability. Linear search and append operations are used to search, add, borrow, and return books efficiently, providing a straightforward and easy-to-understand system. For the web interface, lists and dictionaries were chosen because they remain simple, flexible, and easy to manipulate while supporting dynamic, user-driven interactions on the webpage. Dictionaries clearly store each book's details, while lists allow fast appending, removal, and searching. These data structures keep the system lightweight, readable, and efficient, enabling smooth performance in a web-based environment without the need for complex databases or advanced data models.

Implementation

For this project, a waterfall approach was used for the implementation because the system requirements were clear and unlikely to change. First, the requirements were analysed to determine the system's functionalities, including viewing, adding, deleting, searching, borrowing, and returning books through a web interface. Next, the design phase involved planning the structure of the Flask application, including separate routes for each feature and HTML templates for the presentation layer. During implementation, Python and Flask were used to write the code, while JSON files were used to store books and borrowed records. The system was then tested by verifying each route individually and checking that all features worked together seamlessly. Finally, the system was deployed as a web application, allowing users to interact with the library through forms, buttons, and tables. This software development life cycle approach helped keep the development organized and easy to follow.

Testing

The Library Catalogue System was tested with unit and integration tests. Unit tests ensured books could be added, searched, borrowed, returned, and deleted correctly, with JSON files updating properly. Integration tests simulated full user workflows to verify system reliability, and error handling was checked with invalid inputs to ensure informative messages and stable operation.

Users access a web interface with six main actions which are View All Books, Add Book, Delete Book, Search Book, Borrow Book, and Return Book then further selecting their desired action directly from the page.

The screenshots demonstrate each operation using the example book Berserk by Kentaro Miura, categorized under Horror, showing how the web interface provides a user friendly and visually guided experience.

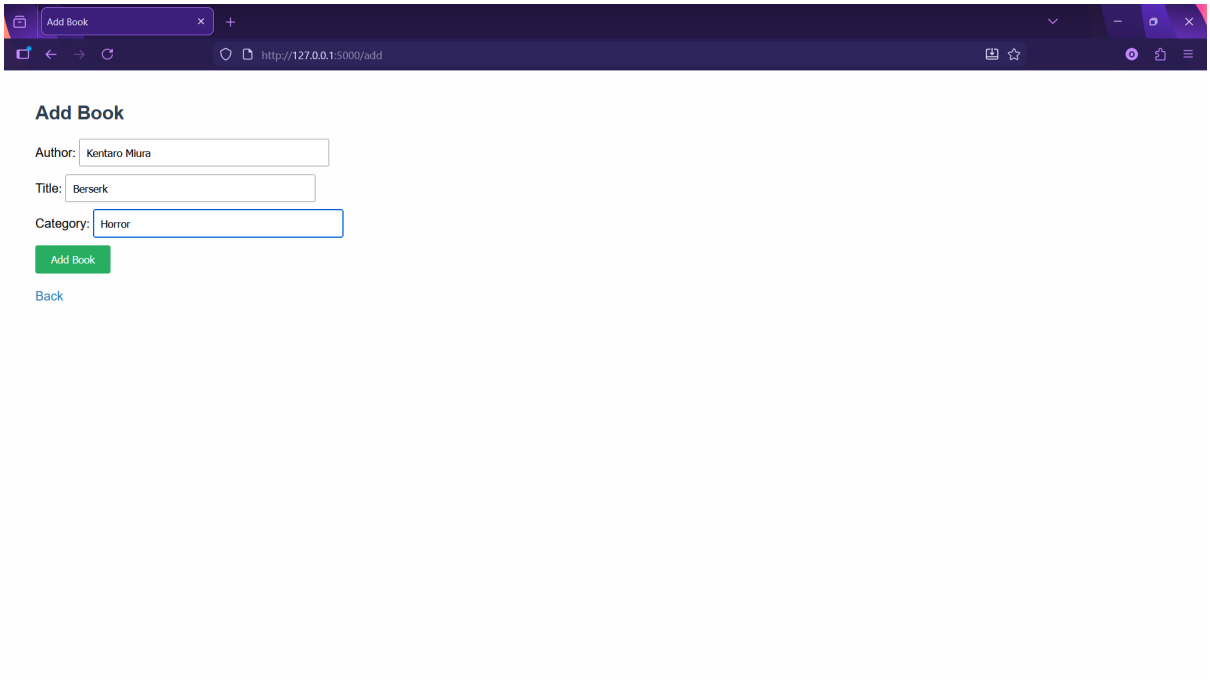

*Figure 3 Click **Add Book** to add a new book by entering the author, title and category.*

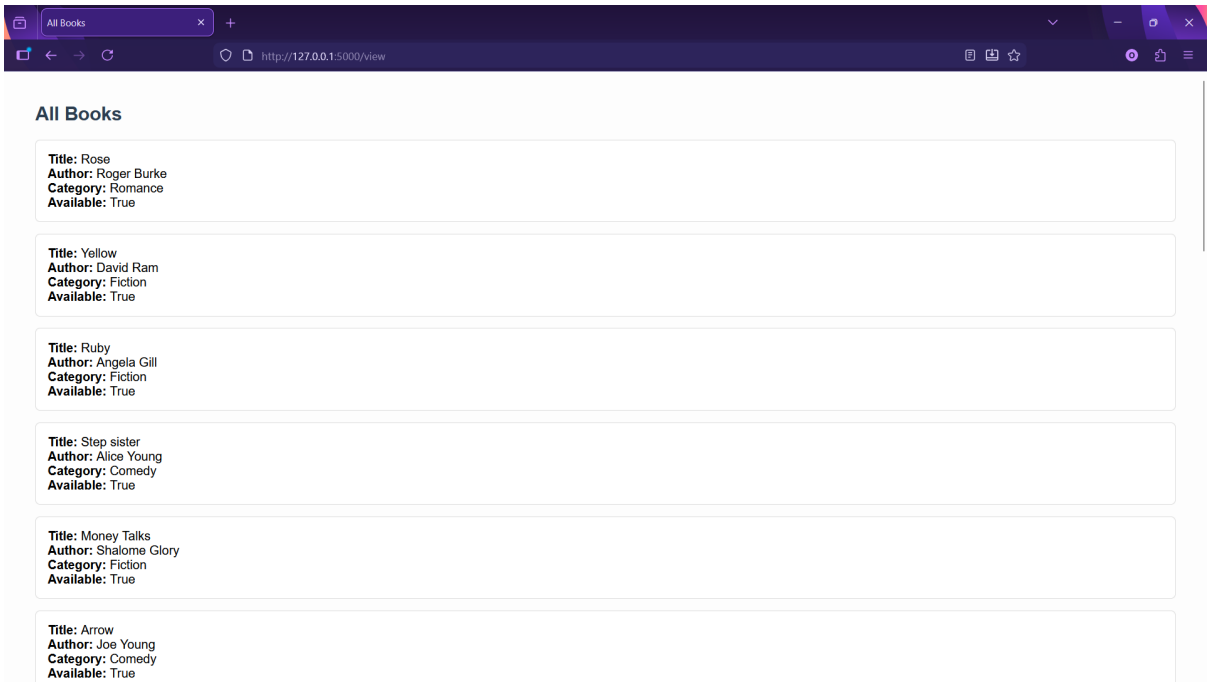

Figure 4 Click View All Books displays the full list of books currently stored in the system

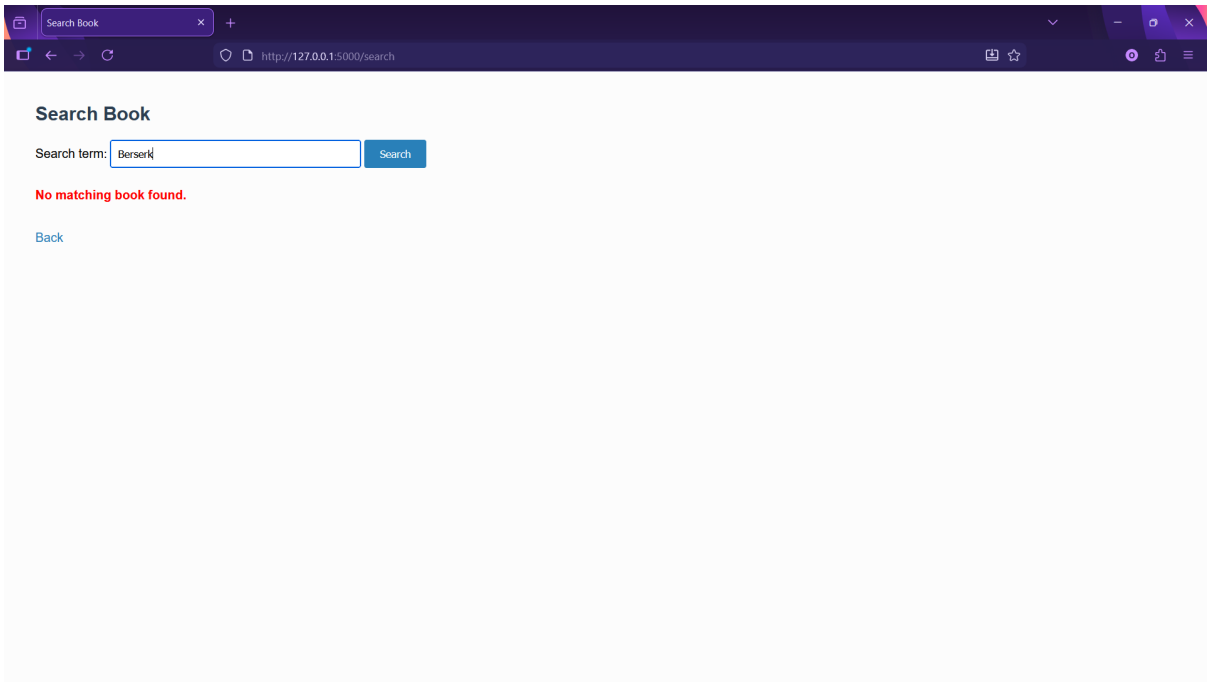

*Figure 5 Clicking **Search Book** enables the user to search for a book by typing the book's title or author*

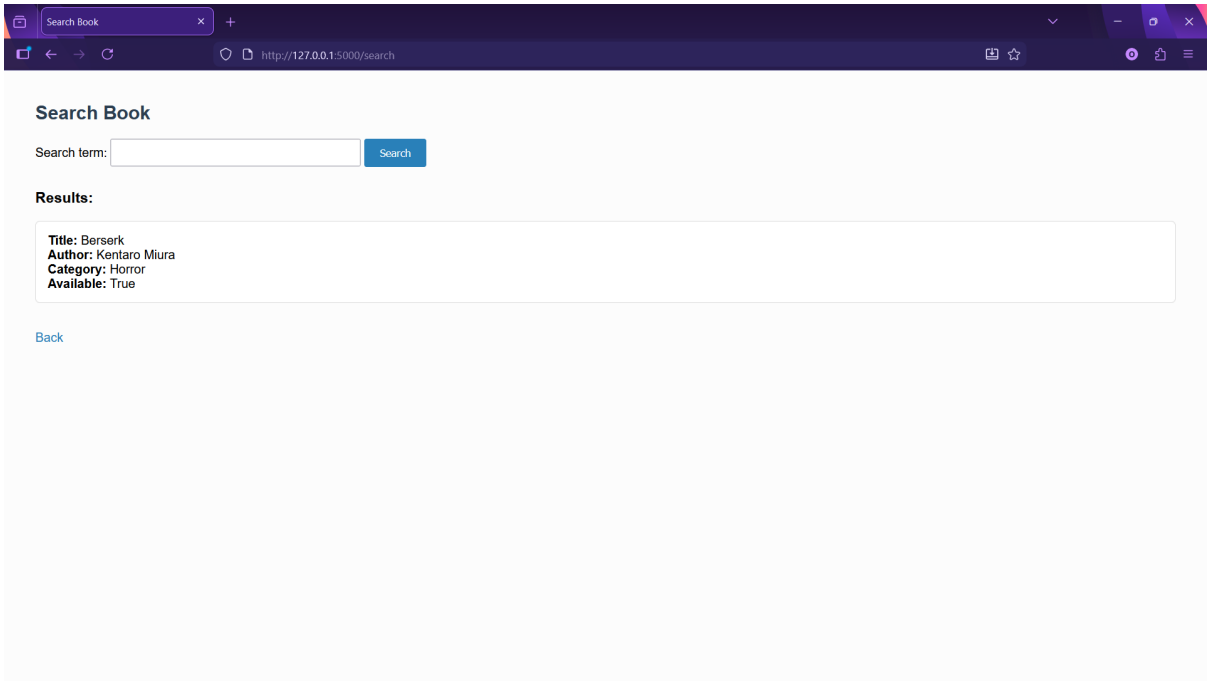

Figure 6 Results

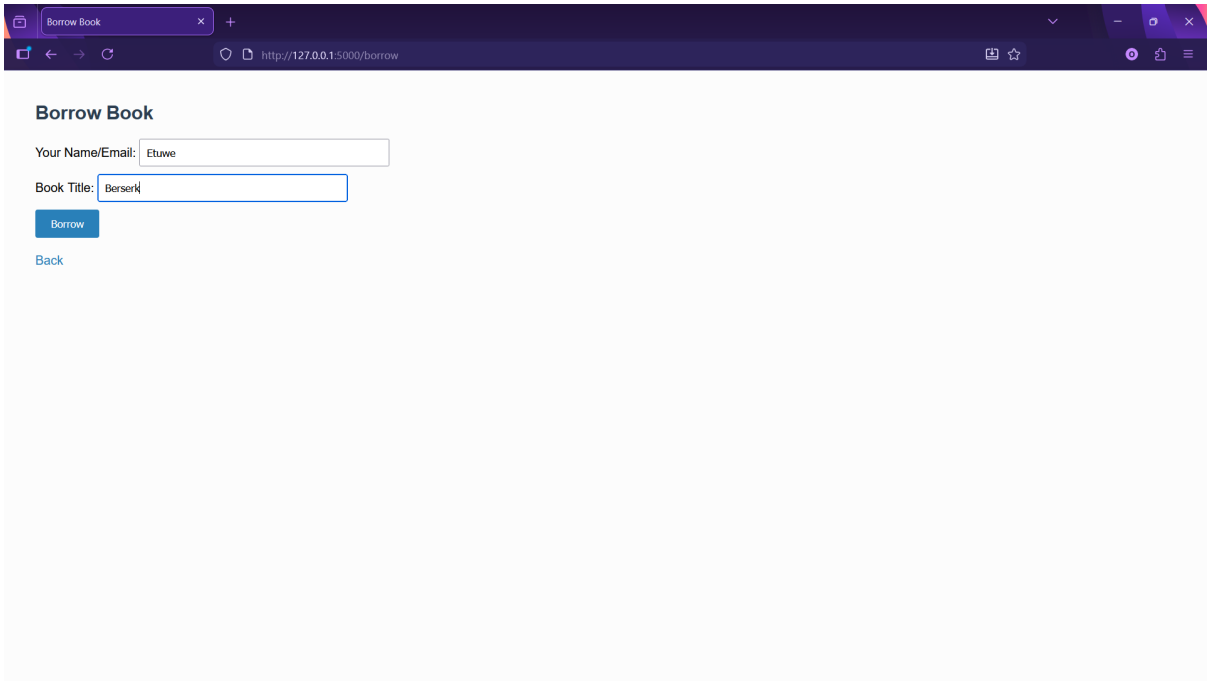

Figure 7 Clicking Borrow Book initiates the borrowing process requiring the user's name and the book's title.

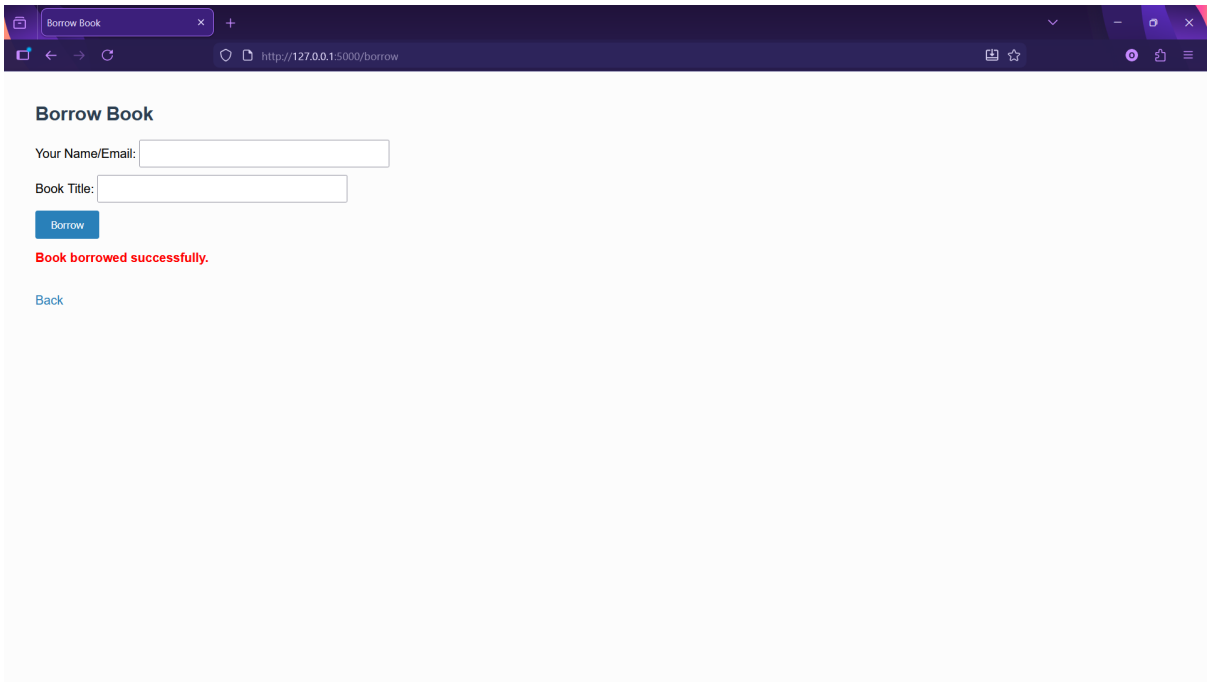

Figure 8 Results

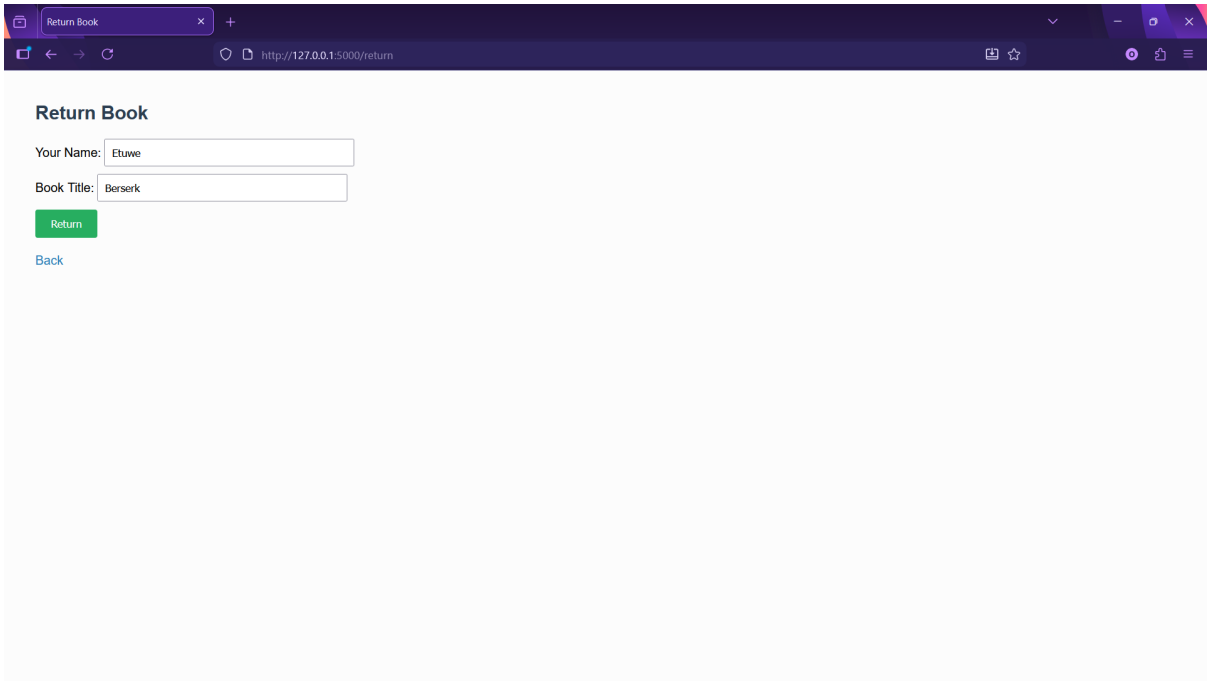

*Figure 9 Clicking **Return Book** facilitates returning a book using the same identifying information*

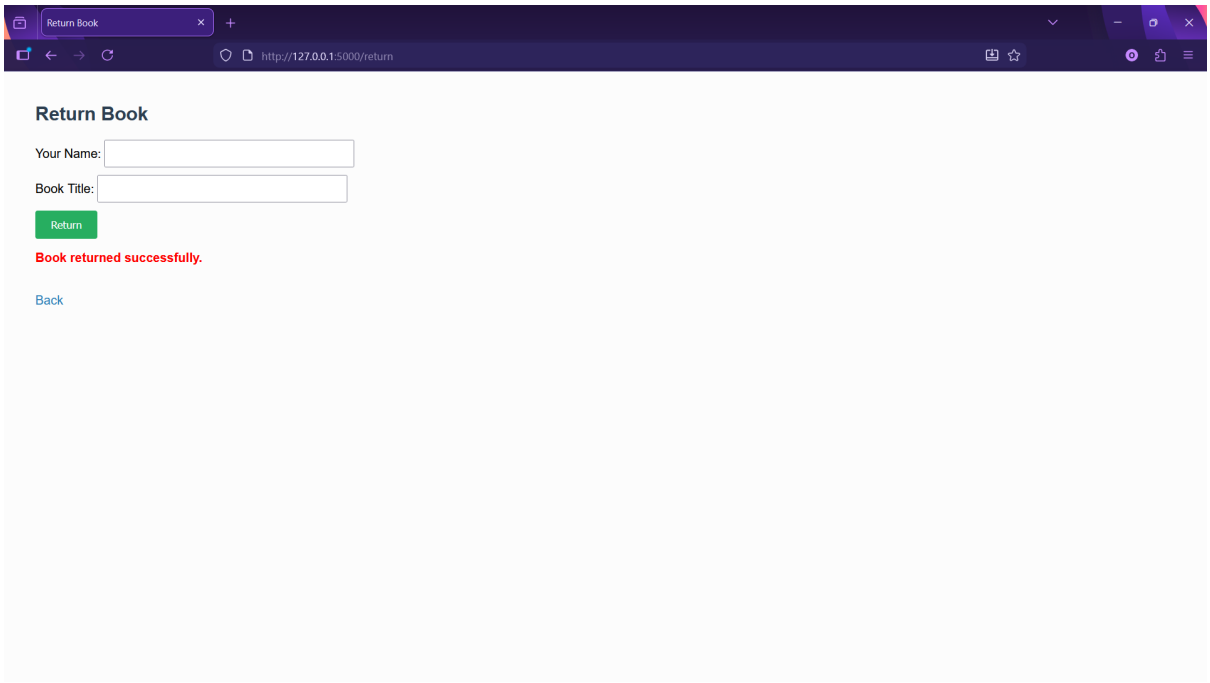

Figure 10 Results

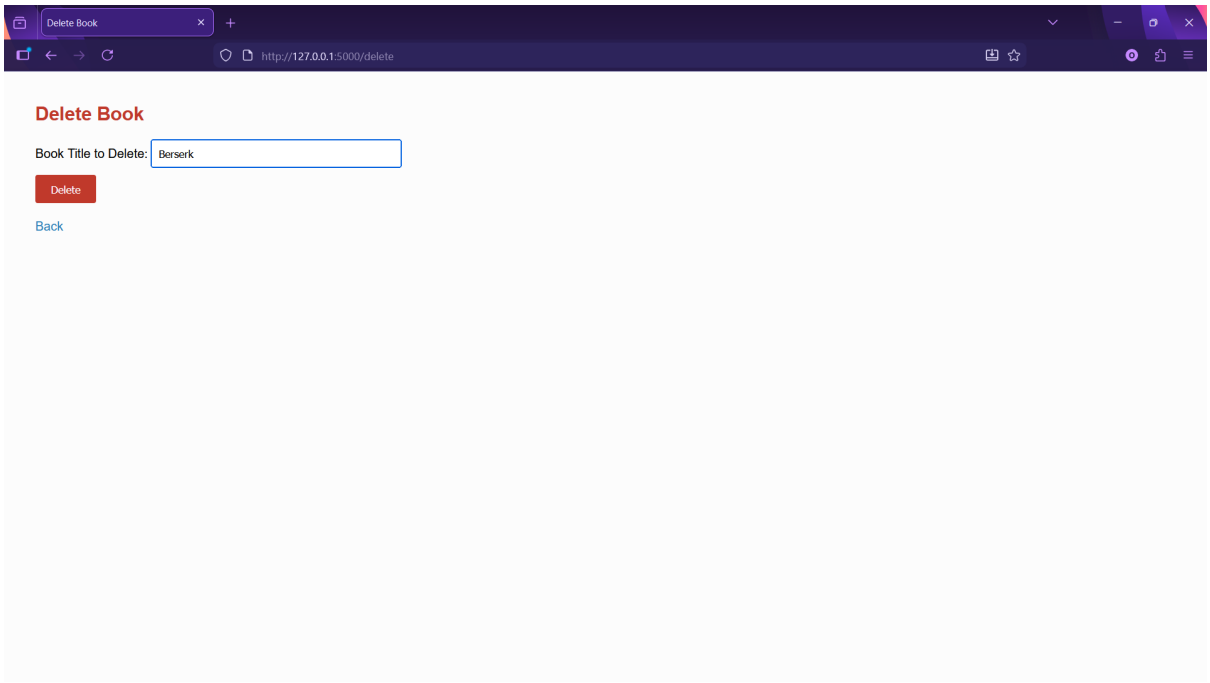

*Figure 11 Clicking **Delete Book** removes a book by providing the book's title.*

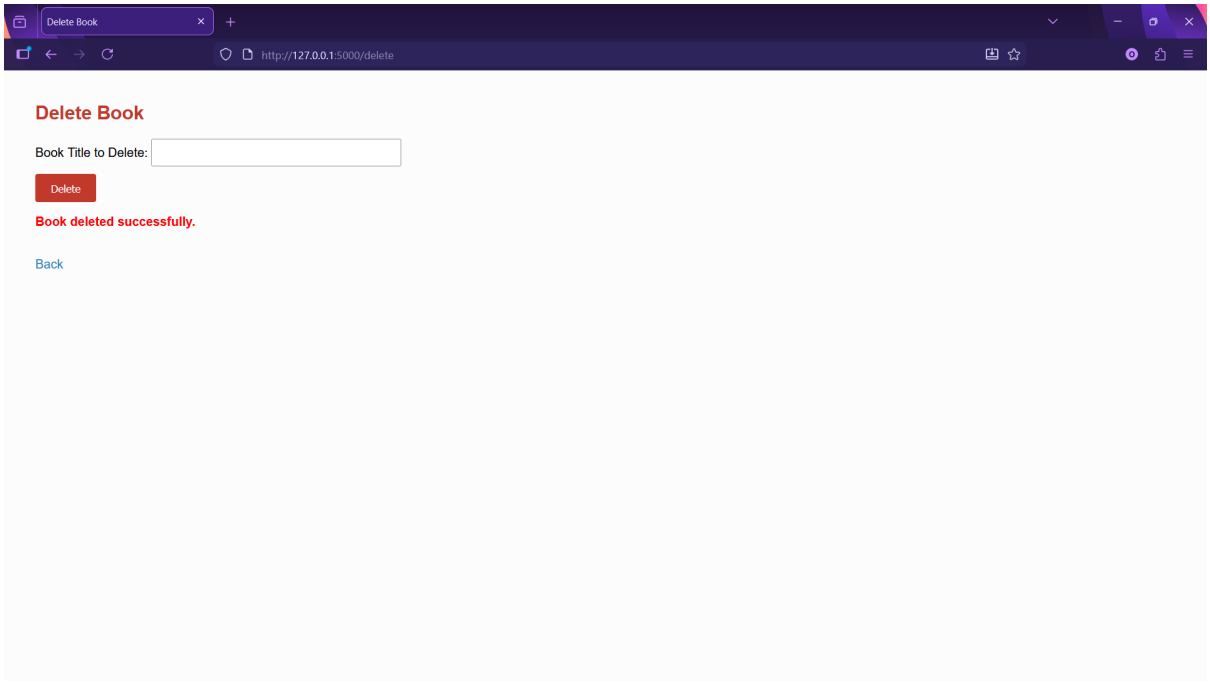

Figure 12 Results

API Usage

No external APIs were required for this project, as all data is stored locally in JSON. This decision simplified the system and removed dependency on external services.

Security Measures

Basic security is provided by checking all user input, using try-except blocks to prevent crashes, and safely handling JSON file updates. These simple measures help protect data from accidental errors and keep the system running smoothly. Advanced security such as authentication and encryption were considered but due to the absence of personal data they were excluded, however the layered architecture ensures security features could be integrated in future.

Evaluation

The system successfully meets the functional requirements and performed well once tested. The implementation of JSON ensured persistent data storage allowing the system to maintain data integrity when executing. Lists and dictionaries were sufficient to manage book records. Error handling was employed through a try/except block to improve the system's durability reducing the risk of unexpected errors. In total the system demonstrates core information system concepts and is intentionally simple yet reliable, easy to expand and build upon in future to add features such as authentication and relational database support.

Conclusion

In conclusion the Library Catalogue System achieved its intended aims delivering a clear solution generating a maintainable and coherent information system. The fusion of persistent data storage, error handling, program design and testing the system exhibits Python programming principles. The web based interface provides a user-friendly way to interact with the catalogue while the logic behind it remains organised but also maintainable. Overall, the system achieved its objectives reflecting a strong application of software development practices.

References

Welcome to Flask — Flask Documentation. Available at: https://flask.palletsprojects.com/en/stable/

Python Software Foundation (2024) json — JSON encoder and decoder. Available at: https://docs.python.org/3/library/json.html

Repository Link: https://github.com/20055960/Advanced_Programming_CA2.git

Video Presentation Link: https://mydbs-my.sharepoint.com/:v:/g/personal/20066635_mydbs_ie/IQCPc7-qrzLoRJ3I2td0PByQAf9RFOhCbzhTbSySvgJC864