Object-Oriented Programming Assignment: Library Book Management App

This comprehensive assignment guides Python developers through creating a Library Book Management application using object-oriented programming principles. The project involves building both a command-line interface and a Streamlit web application that will allow users to add books to the library, issue books to borrowers, process returns, and view the current status of all books. Through this practical exercise, developers will implement and solidify their understanding of key OOP concepts including classes, instance variables, class variables, and methods that modify object state.

by mukesh kumar

Project Overview and Requirements

The Library Book Management App is designed to simulate a real-world library system where books can be tracked, issued to borrowers, and returned. This project requires implementing object-oriented programming concepts to create a functional and user-friendly application with both command-line and web interfaces.

Core Problem Statement

Your task is to develop a Python program that manages library books through object-oriented programming. The application must handle basic library operations including adding new books to the inventory, issuing books to users, processing returns, and displaying the current status of all books in the collection. The solution must implement a Book class with appropriate instance variables to track individual book details and class variables to maintain library-wide statistics.

Required Interfaces

Command Line Interface (CLI)

A menu-driven system allowing users to perform all library operations interactively through the terminal. This interface must provide options for adding books, issuing books to borrowers, returning books, and viewing book status information.

Streamlit Web Application

A graphical user interface built with Streamlit that provides input fields and buttons for all library operations. The web application must dynamically display book statuses and update library statistics in real-time as operations are performed.

OOP Concepts to Implement

Concept	Application in Project
Instance Variables	Store individual book data (title, author, issued_to status)
Class Variables	Track library-wide statistics (total books, currently issued books)
Methods	Implement functionality for adding, issuing, returning, and displaying books
Encapsulation	Group related data and behaviors within the Book class

This project serves as a practical application of OOP principles in a scenario that developers might encounter in real-world software development. By implementing both a CLI and web interface, developers will gain experience in creating different types of user interfaces while maintaining the same core functionality through object-oriented design.

Book Class Design and Implementation

The core of the Library Book Management App is the Book class, which encapsulates all the data and behaviors needed to represent and manage books in the library system. This section details the structure and implementation requirements for this class.

Class Structure

Instance Variables

- **title**: Stores the book's title as a string
- **author**: Stores the book's author as a string
- **issued_to**: Tracks who has borrowed the book; None if available

Class Variables

- **total_books**: Counter for all books in the library
- **total_issued_books**: Counter for currently borrowed books

Required Methods

__init__(self, title, author)

Constructor method that initializes a new Book object with the provided title and author. It sets the initial issued_to status to None (indicating the book is available) and increments the total_books class variable to maintain an accurate count of all books in the library.

issue(self, borrower_name)

Handles the process of issuing a book to a borrower. This method checks if the book is currently available (issued_to is None). If available, it assigns the borrower's name to issued_to, increments the total_issued_books counter, and returns a success indicator. If the book is already issued, it returns a failure indicator.

return_book(self)

Processes a book return operation. It checks if the book is currently issued (issued_to is not None). If issued, it resets issued_to to None, decrements the total_issued_books counter, and returns a success indicator. If the book is not currently issued, it returns a failure indicator.

show_status(self)

Displays comprehensive information about the book, including its title, author, and current availability status. If the book is issued, it shows which user has borrowed it. This method helps library staff and users quickly determine whether a specific book is available for borrowing.

When implementing the Book class, it's important to maintain proper encapsulation and ensure that class variables are accessed and modified correctly. The methods should include appropriate error handling and return clear success/failure indicators to facilitate integration with both the command-line and web interfaces. Remember that class variables are shared among all instances, so operations that modify these variables must be carefully implemented to maintain data consistency.

Building the User Interfaces

Your Library Book Management application requires two distinct user interfaces: a Command Line Interface (CLI) and a Streamlit web application. Each interface will provide access to the same core functionality but through different interaction methods. This section provides guidance on implementing both interfaces.

Command Line Interface (CLI)

The CLI should provide a text-based, menu-driven system that allows users to interact with the library management system through the terminal. This interface should be intuitive and user-friendly, with clear instructions and feedback.

1 Main Menu

Implement a main menu that displays all available operations and prompts the user to select an option. The menu should loop continuously until the user chooses to exit the application.

2 Input Handling

Create functions to handle user input for each operation, including validation to prevent errors. For example, when adding a new book, prompt for title and author separately with appropriate error messages for invalid inputs.

3 Output Formatting

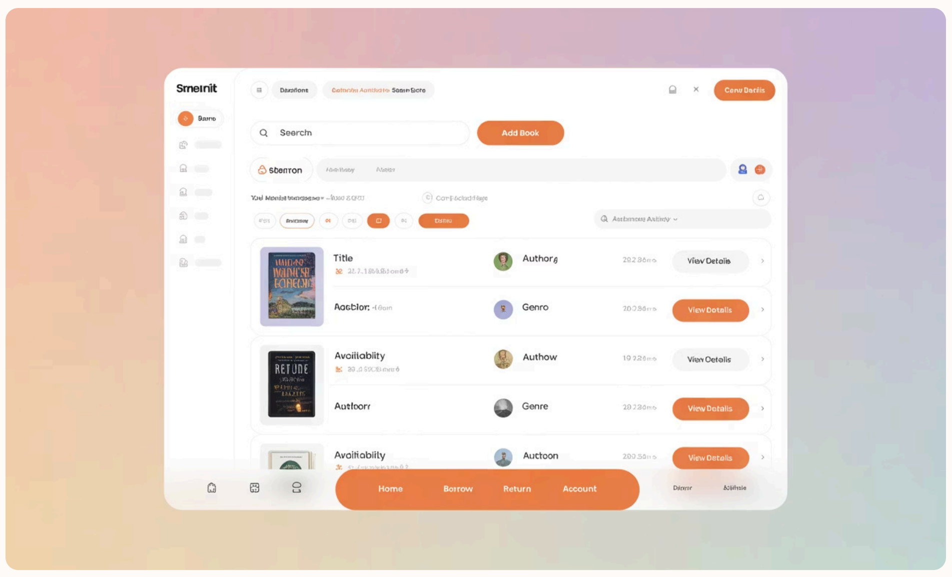
Design clear, well-formatted output displays for showing book status, success/failure messages, and library statistics. Consider using tabular formats for displaying multiple books.

4 Error Handling

Implement robust error handling to manage invalid inputs, book not found scenarios, and other potential issues that might arise during operation.

Streamlit Web Application

The Streamlit web application should provide a graphical interface with input fields, buttons, and dynamic displays. This interface should leverage Streamlit's reactive programming model to create an interactive and responsive user experience.



A sample Streamlit interface for the Library Book Management App showing input fields for adding books and a display of current book status.

Key Components to Implement:

- **Input Fields**: Create text inputs for book titles, authors, and borrower names
- **Action Buttons**: Implement buttons for adding books, issuing books, returning books, and refreshing displays
- **Status Displays**: Create dynamic displays that show current book status and update in real-time
- **Statistics Section**: Include a section that displays library-wide statistics such as total books and issued books
- **Feedback Messages**: Implement success/error messages to provide feedback on operations
- **Book Listings**: Create a tabular or card-based display of all books in the library

Integration Considerations

Both interfaces should use the same underlying Book class implementation to ensure consistent behavior. Consider creating a Library class that manages a collection of Book objects and provides methods that both interfaces can use. This approach follows the Model-View-Controller (MVC) pattern, where:

- The Book and Library classes serve as the Model (data and business logic)
- The CLI and Streamlit interfaces serve as Views (presentation layers)
- Interface-specific code handles the Controller role (input processing and response generation)

This separation of concerns will make your code more maintainable and testable, while also demonstrating a deeper understanding of object-oriented design principles. Remember to thoroughly test both interfaces to ensure they properly reflect the state of the underlying Book objects and correctly update class variables as operations are performed.