REPORT ON OOP MINI PROJECT

TOPIC: Bookstore Management System

Course: Object-oriented Programming 152626 – 2024.1

Dr. Tran The Hung

Group 13

Nguyễn Hoàng Thái - 20235557

Lương Minh Hiếu - 20230083

Nguyễn Ngọc Linh - 20235520

Đoàn Trường Giang - 20235494

Trần Quang Trọng - 20235565

Đặng Xuân Đạt - 20235485

1. Introduction:

This Java-based bookstore management system is a comprehensive point-of-sale and inventory management application developed using JavaFX for the graphical user interface and MySQL for database management. The application implements object-oriented programming principles to create a robust and maintainable system for bookstore operations.

This system is specifically designed to assist bookstore cashiers in managing critical tasks such as tracking revenue, monitoring book availability, and processing customer purchases efficiently. Additionally, the system automates the purchase process and revenue calculation, significantly reducing manual workload and minimizing errors..

The application is designed primarily for:

- Bookstore administrators and managers
- Sales staff and cashiers
- Inventory managers
- Small to medium-sized bookstore operations

2. Mini project description:

- Login System: The application begins with a secure login system that allows authorized administrators to access the system using their credentials. This ensures that only authenticated users can access sensitive business data and perform operations.
- Dashboard Interface: The system provides a comprehensive dashboard that displays:
    - Total number of available books
    - Total income generated
    - Total customer count
    - Visual analytics through income and customer charts
    - Real-time business performance metrics
- Book Management: The system offers complete inventory control capabilities:
    - Add new books with detailed information (ID, title, author, genre, publication date, price)
    - Upload and store book cover images

    - Update existing book information
    - Remove books from inventory
    - Search and filter book inventory
    - Real-time inventory tracking
- Purchase Management: The purchase module enables:
    - Book selection through ID or title
    - Quantity specification
    - Automatic price calculation
    - Multiple book purchase handling
    - Transaction recording
    - Payment processing
    - Customer purchase history tracking

- Use case diagram

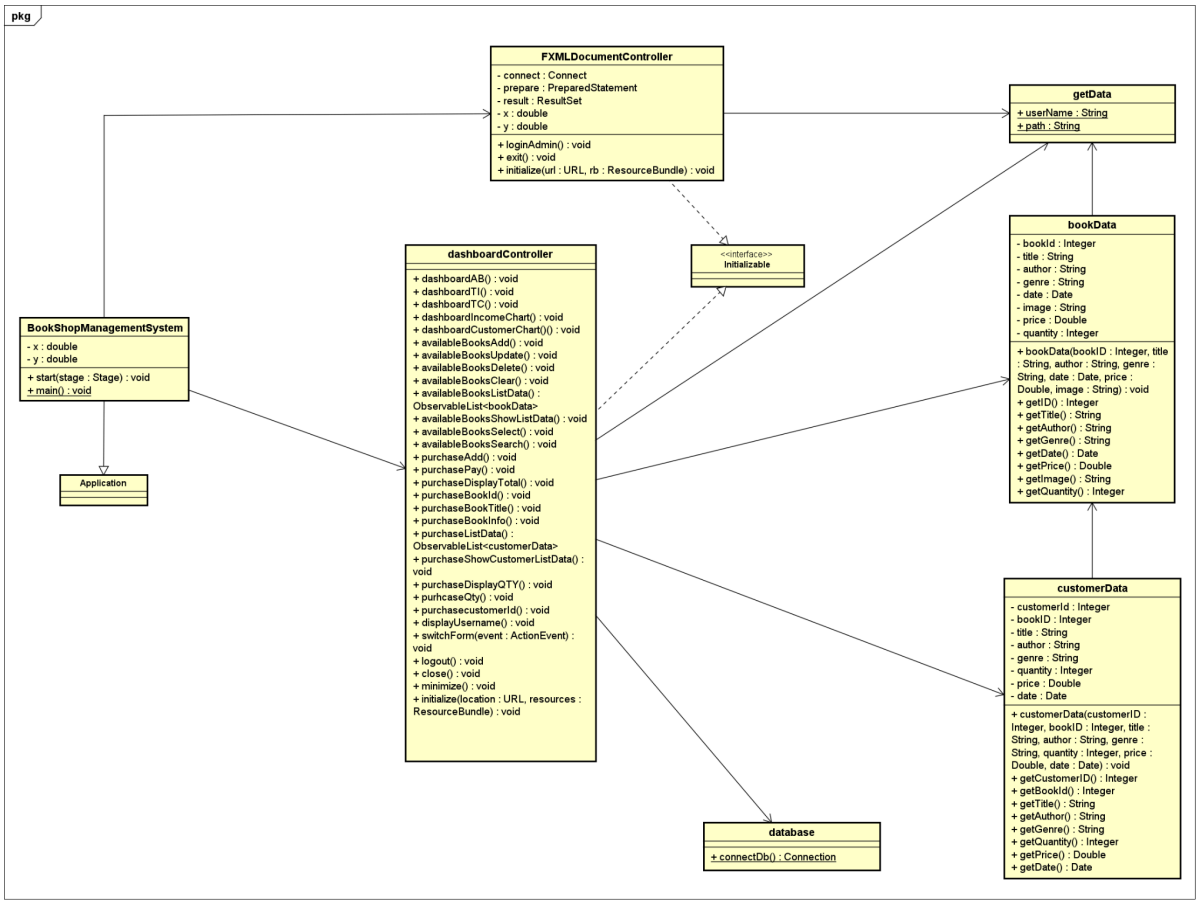
Based on all the requirements we decided to develop three use cases (as shown in the figure for our application.)

To be more specific, after login into the system, admin can:

- View the dashboard: check number of available books, customers and revenue, view analytic chart
- Manage the bookstore: add/delete book, update book details, search book
- Sales Management: process purchase, view sales history

3. Design:

3.1. General class diagram:



In the current implementation of the Bookstore Management System, all classes are placed within a single package named `bookshopmanagementsystem`. This includes:

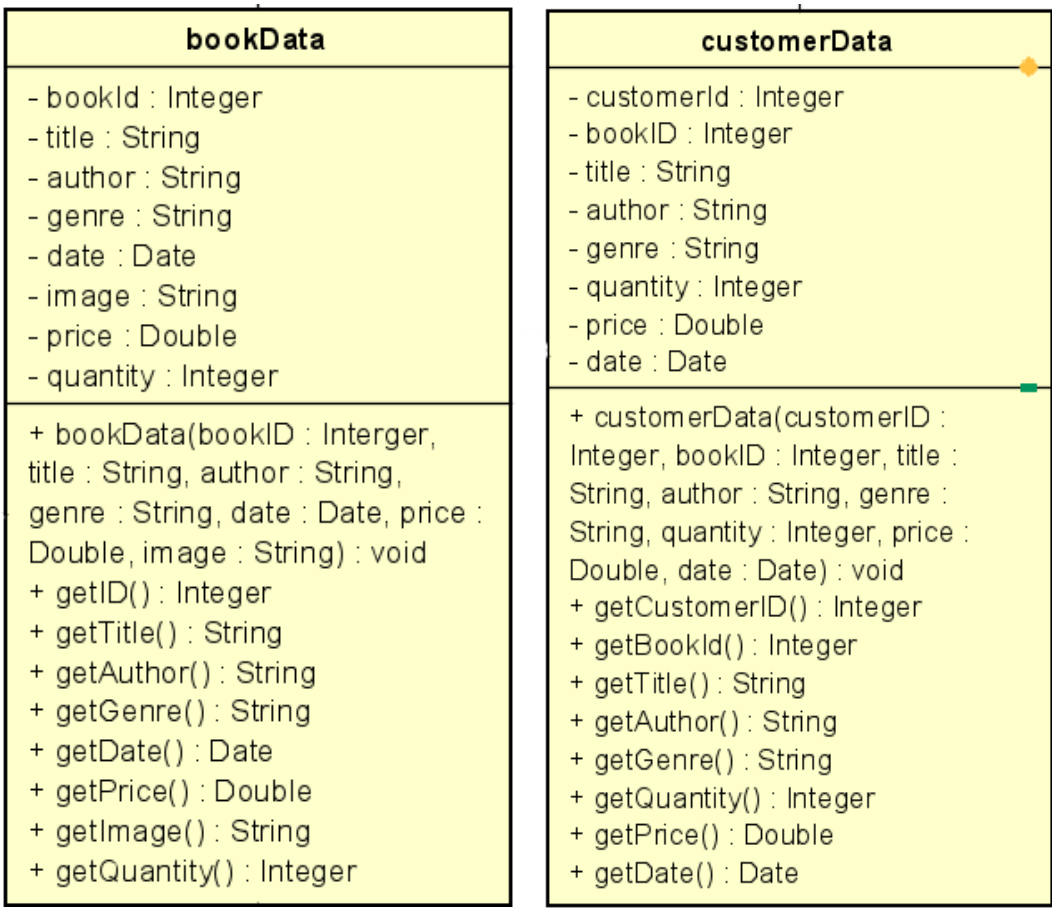
- `bookData.java`: Represents book information.
- `customerData.java`: Manages customer purchase data.
- `database.java`: Handles database connectivity.
- `getData.java`: Utility class for data retrieval.
- `dashboardController.java`: Handles dashboard operations.
- `FXMLDocumentController.java`: Manages the UI components and user interactions.

- `BookShopManagementSystem.java`: The main application class.

Relationships

- **bookData** and **customerData** are entity classes representing data models in the system.
- **dashboardController** uses **bookData** and **customerData** to manage and display data in the UI.
- **database** provides the database connection used by **dashboardController** and **FXMLDocumentController**.
- **FXMLDocumentController** handles user authentication and navigation to the dashboard, interacting with **database** for login validation.
- **getData** is used to share data between **FXMLDocumentController** and **dashboardController**.

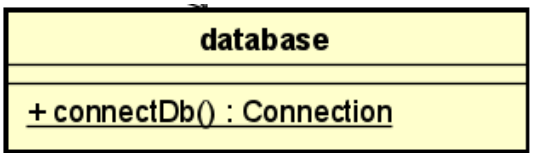
3.2. Book data and customer data class diagram



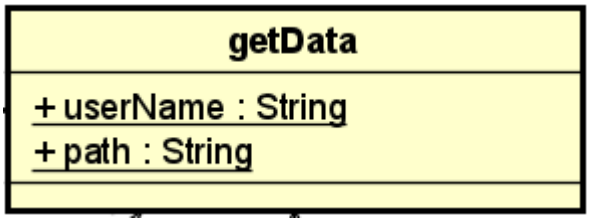
Purpose: These 2 classes are data model classes, designed to get data of books and customer's purchase.

- **bookData**: Represents book information including ID, title, author, genre, publish date, price, image and quantity.
- **customerData**: Represents a customer's purchase record, containing customer ID, book details, quantity, price, and purchase date.

3.3. database and getData class diagram



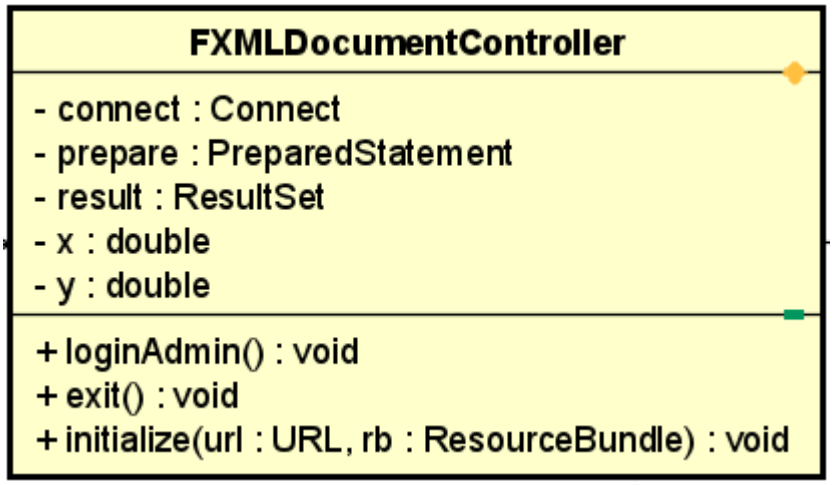
- **Purpose**: Utility class for establishing a database connection.
- **Methods**: **connectDb()**: Establishes and returns a connection to the MySQL database.



- **Purpose**: Utility class for sharing data between controllers.
- **Fields**:
  - **username**: Stores the logged-in username.
  - **path**: Stores the image path for books.

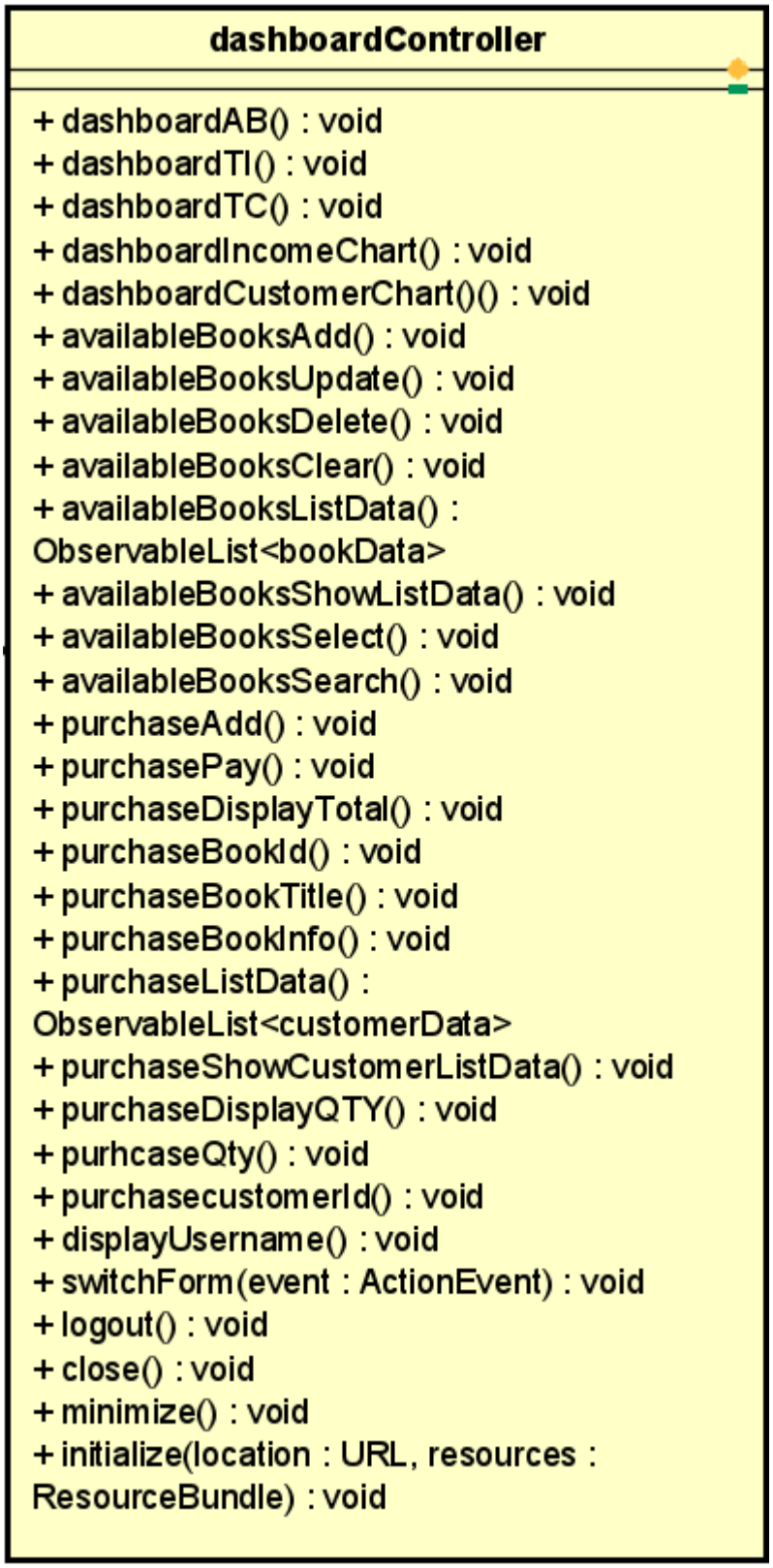
3.4. Controller class diagram

- FXMLController



- **Purpose**: Controller for the login form, handling user authentication.
- **Methods**:
  - **loginAdmin()**: Authenticates the user and navigates to the dashboard.
  - **close()**: Exits the application.
  - **initialize()**: Initializes the login form.

- dashboardController



- **Purpose**: Controller for the dashboard interface, managing UI components and business logic.
- **Method:**

- **Dashboard Operations** (**dashboardAB(), dashboardTI(), dashboardTC(), …**): These methods display and interact with specific sections of the dashboard like analytics data, or display charts related to income and customer data.
- **Available Books Management:** These methods manage a collection of available books. Some operations can be performed:
  - CRUD operations (Add, Update, Delete, and Clear).
  - **availableBooksShowListData()** displays the list in the dashboard.
  - **availableBooksSelect()** and **availableBooksSearch()** help in interacting with the books.
- **Purchase Management:** These methods handle purchasing functionality, including:
  - **purchaseAdd():** Adding purchases.
  - **purchasePay():** Payment processing.
  - **purchaseDisplayTotal(), purchaseDisplayQty():** Displaying purchase totals and quantities.
  - **purchaseShowCustomerListData():** Managing customer-related purchase data.
  - **purchaseBookInfo():** Fetching book-related purchase info.
- **initialize()**: Initializes the controller once its root element has been loaded.
- **switchForm():** handles navigation between forms.
- **logout(), close(), minimize():** manage session and window lifecycle.

4. Technical implementation:

4.1. GUI Design with JavaFX

This project is a Bookstore Management System built using JavaFX for the graphical user interface (GUI). The system allows administrators to manage available books, track purchases, and view analytics through a dashboard. The provided files include:

- FXML Files: dashboard.fxml and FXMLDocument.fxml for defining the UI layout.
- CSS Files: dashboardDesign.css and loginDesign.css for styling the UI components.
- Controllers: dashboardController.java and FXMLDocumentController.java for handling UI logic.

- **Login Screen (`FXMLDocument.fxml`)**
- **Purpose**: Allows admins to log in using a username and password.
- **UI Elements**:
    - `TextField` for username input.
    - `PasswordField` for password input.
    - `Button` for login submission.
    - `FontAwesomeIcon` for user icon.

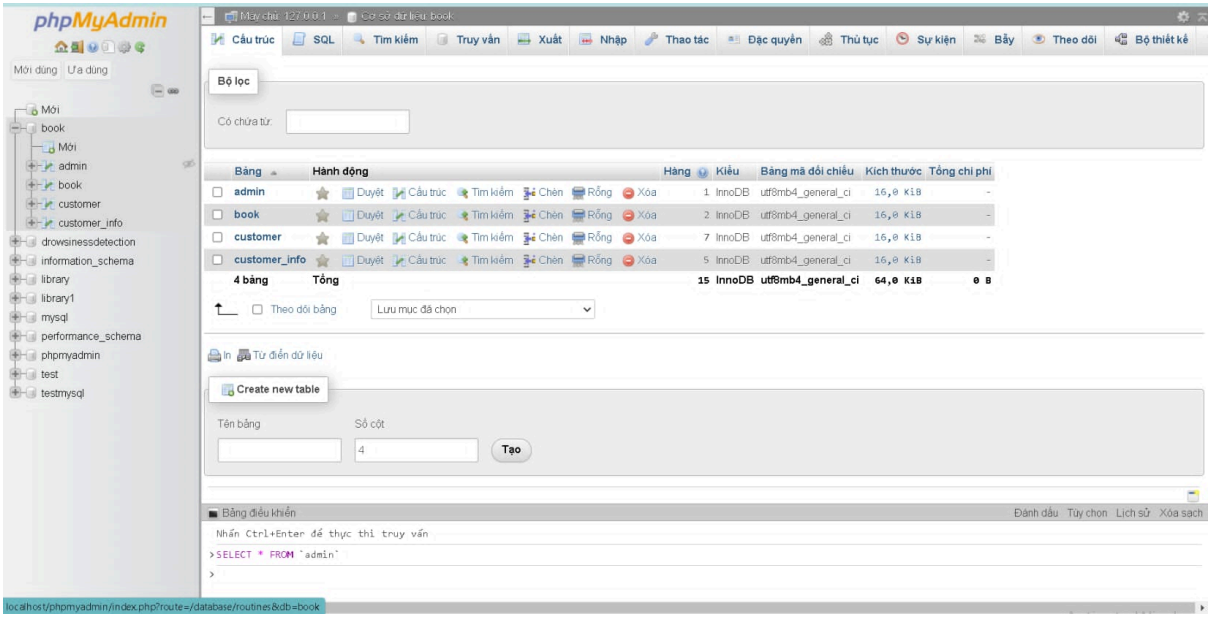
- **Dashboard Screen (`dashboard.fxml`)**
- **Purpose**: Serves as the main interface post-login, offering navigation to different management sections.
- **Layout**:
    - **BorderPane** organizes content into top, left, and center regions.
    - **Top Region**: Contains the header with the app title and control buttons (minimize, close).
    - **Left Region**: Navigation pane with buttons to switch between Dashboard, Available Books, Purchase, and Logout.
    - **Center Region**: Displays forms based on the selected navigation button.
- **UI Elements**:
    - **Navigation Buttons**: Dashboard, Available Books, Purchase, Logout.
    - **Labels and Icons**: Display user welcome message, section titles, and icons.
    - **Charts**: AreaChart and BarChart for income and customer analytics.
    - **Forms**: Separate AnchorPanes for Dashboard, Available Books, and Purchase forms, each containing relevant controls (e.g., text fields, buttons, table views).

- **Styling with CSS**
- **Purpose**: Enhances the visual appeal and consistency of the UI.
- **CSS Styles**:
    - Defines styles for forms, buttons, text fields, tables, and icons.
    - Uses classes like `.textfield`, `.login-btn`, `.card`, and `.shadow` to apply consistent styling.
    - Incorporates gradients and effects for backgrounds and buttons.

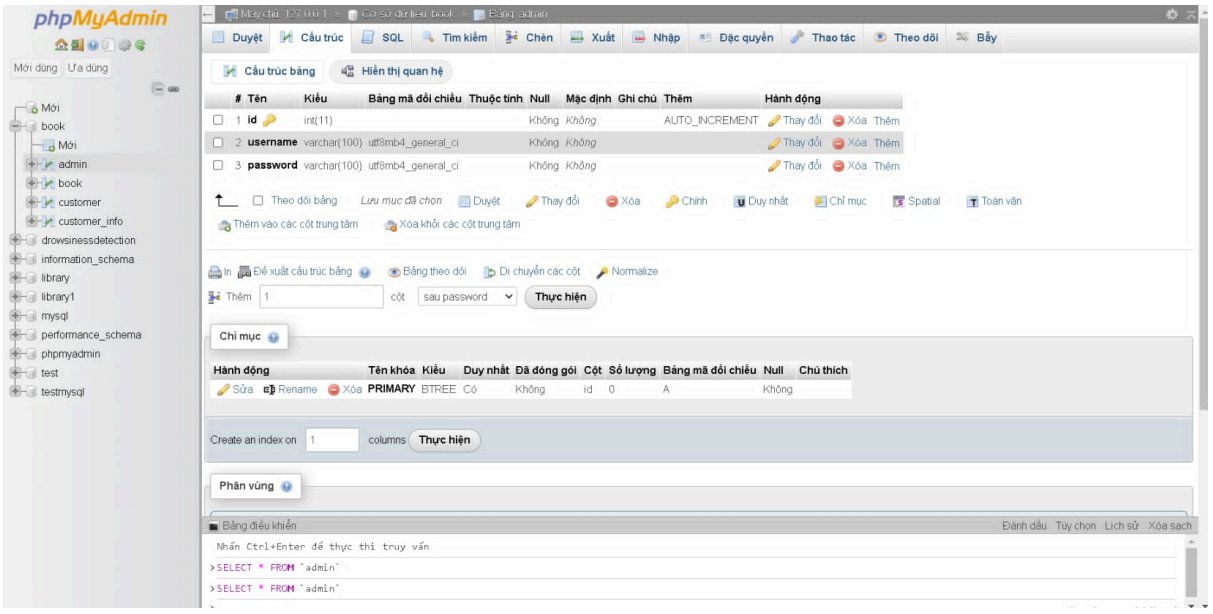
4.2. **Database**

- Query language: MySQL managed in Xampp

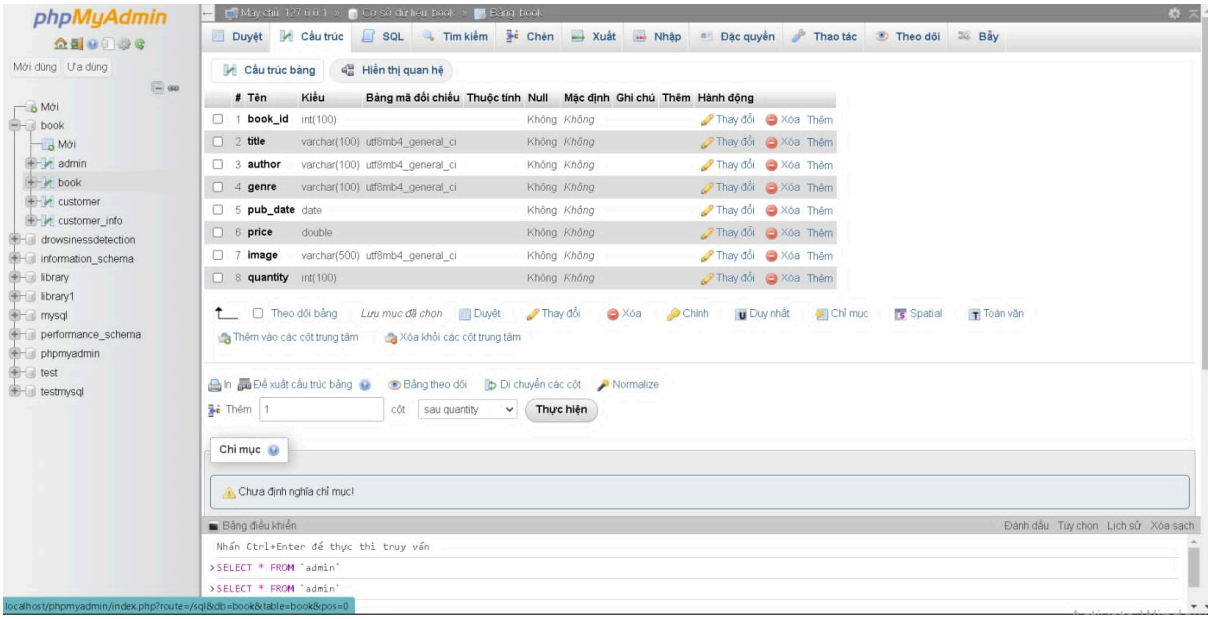
- Structure: book database containing 4 tables



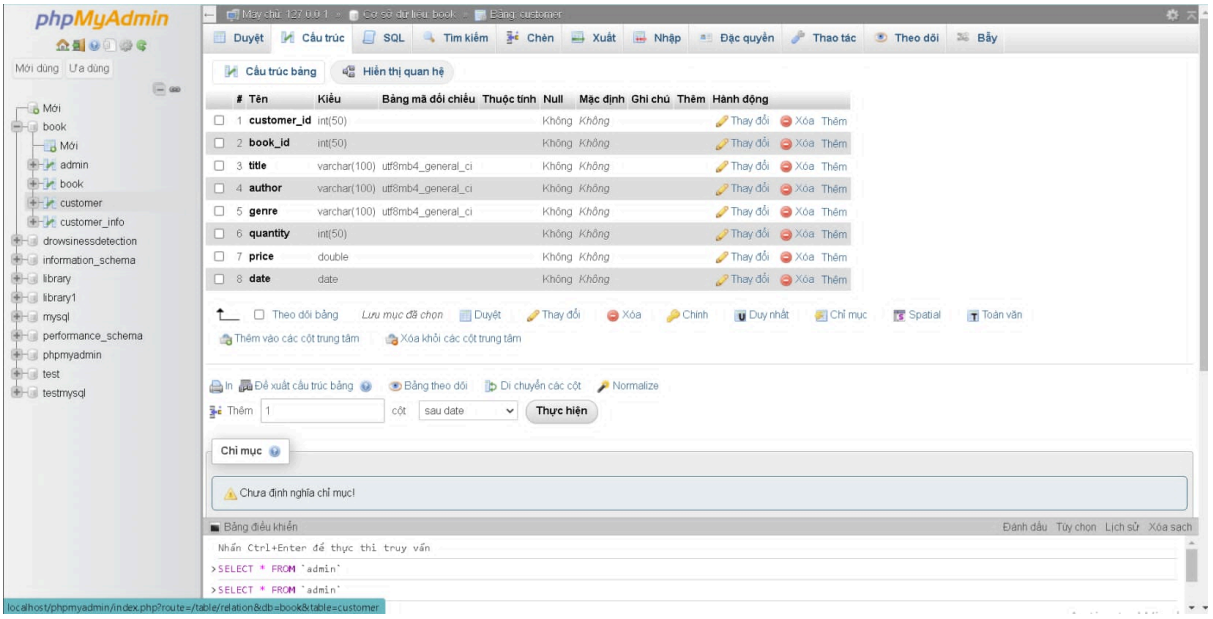
+ Table 1: admin



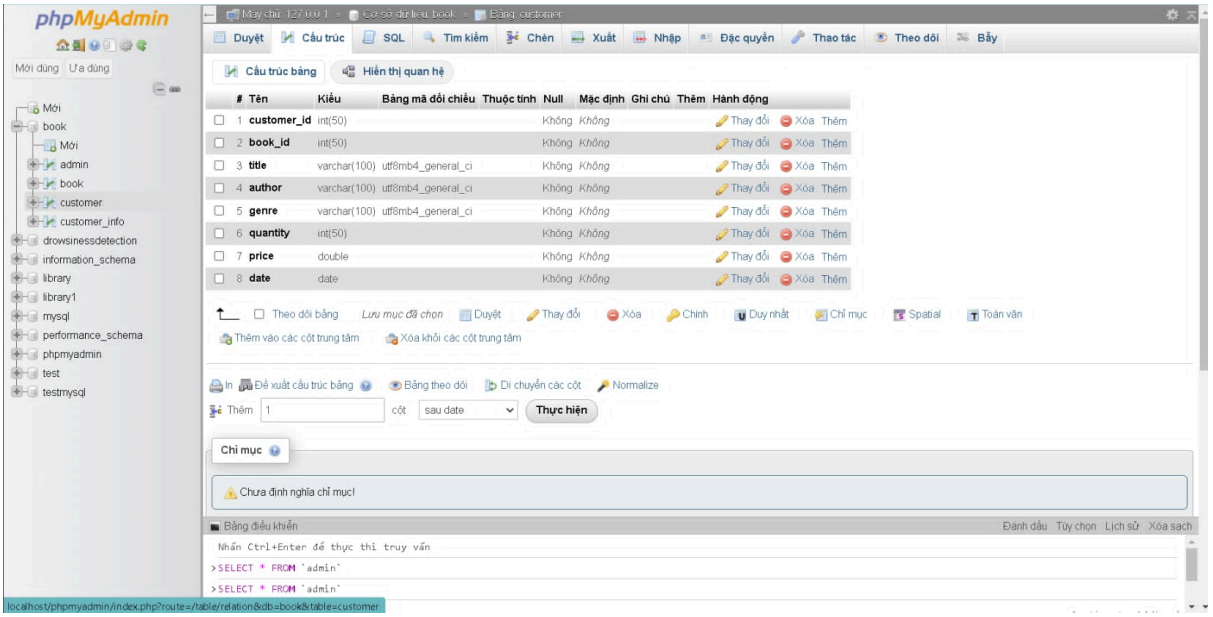
+ Table 2: book



+ Table 3: customer



+ Table 4: customer_info



5. Suggestion for future developments and improvements

5.1. Developing app for customers:

The idea of developing a customer app for a bookstore management system is a forward-thinking solution aimed at streamlining the shopping experience for customers while providing valuable insights for the bookstore's operations. This app would serve as a bridge between customers and the bookstore, offering a seamless interface for browsing, purchasing, and managing interactions with the store. Customers can also order their favourite book online through this app.

5.2. Functionality expansion:

- Recommendation system: Introduce a recommendation engine that suggests books based on users' browsing and purchase history, enhancing the user experience and potentially increasing sales.
- Advanced Analytics: Expand the admin dashboard to include more detailed analytics, such as sales trends, best-selling books, and customer demographics, to support better decision-making.

6. Conclusion

In conclusion, the Book Shop Management System effectively manages book inventory and customer purchases, offering functionalities such as CRUD operations for books, purchase management, and data visualization through charts. However, it currently lacks advanced features like user role management and integration with external services, and its coding could benefit from better organization and enhanced database organization. Future development should focus on implementing user roles, improving error handling, optimizing database interactions, and enhancing the user interface for a more seamless experience. These enhancements will not only strengthen the system's capabilities but also ensure its scalability and security for broader applications.