Tanta University

Faculty of Computers and Informatics

Library Management System

Graduation Project

Student's names

Mohammed Mahmoud Abd El-Ghani

Abd El-Halim Mahmoud Abd El-Halim Gadwal

Moamen Mohamed Abd El-Rahim Abu Al-Gheit

Huda Khedr Mohamed El-Gahawy

Mariam Mohamed Saeed Eltras

Doaa Gamal Mustafa Ali Saeed

Basmala Yahya Ibrahim Ali Nawar

Supervisor

Dr. Moustafa El-Ashry

Academic Year 2025

Abstract

A Library Management System (LMS) is a digital solution designed to streamline and manage library operations, enhancing the efficiency of traditional manual processes. This system allows for the centralized management of library resources, including the acquisition, cataloging, circulation, and tracking of books, journals, and digital media. The LMS provides tools for managing user accounts, lending services, and reservation requests. It simplifies the task of maintaining an up-to-date record of resources and automates many aspects of library management. By providing user-friendly search capabilities, a library management system also enables patrons to easily locate and access resources, making library services more accessible and promoting resource utilization. With integrated reporting and analytics, it aids librarians in decision-making and inventory management. Overall, the Library Management System improves library services by reducing administrative tasks and enabling a seamless user experience.

Acknowledgment

All praise to Allah (S.W.T), the Almighty, who granted us the strength and patience to work on this project. Peace and blessing of Allah be upon his last prophet Muhammad (PBUH), his family and his companions.

Our deep appreciation and gratitude go to our thesis advisor Dr. Moustafa El-Ashry for his guidance, consistent help and continuous support with creative input throughout our work. His valuable suggestions and useful discussion made this work interesting to us.

Our full appreciation and gratitude go to our families, a pillar support for us. Thanks to our parents who were- after Allah- the source of success in our lives. Words fall short in conveying our gratitude towards them. May Allah (S.W.T) gives them good health and gives us ample opportunities to serve them through our lives.

Finally, thanks are due to all the group members for their moral and support during project work time, difficult and happy moments, good wishes and the memorable days shared together.

Table of Contents

List of Figures

List of Tables

Chapter 1: Introduction

1.1 Introduction

The Library Management System (LMS) project aims to develop a comprehensive web-based platform to facilitate the effective management of library resources and services. Traditional library management, which relies heavily on manual processes, often leads to inefficiencies in tracking inventory, managing checkouts, and handling overdue notices. This project replaces these cumbersome procedures with a streamlined digital platform that automates routine tasks, enhances user experience, and ensures accurate records of all library activities.

The LMS manages diverse operations, including cataloging, acquisition, circulation, and resource tracking, all from a central database.

It is designed to serve two main user groups:

- **Users (Patrons)**: Library members who can explore services like reading, borrowing, and reserving books, managing their profiles, and making payments online or offline.
- **Admins (Library Staff)**: Administrators who can manage books, authors, categories, and user activities, view reports, and oversee overall system operation.

Through this project, we aim to provide a scalable, efficient, and user-friendly solution that improves accessibility to resources and enhances library services.

1.2 System Features

The Library Management System **contains:**

User Features (Patron Side)

- **Home Page**
  - Explore available services: *Read Book*, *Reserve Book*, *Borrow Book*.

- View team information: names, professional titles, and roles of team leaders.

- **Authentication**
  - Login using email and password.
  - Sign up by providing first name, last name, phone number, address, email, password, and password confirmation.
- **Contact Us Page**
  - Submit inquiries by providing name, email, and message.
  - View library contact details, including address, phone number, and email.
- **Book Categories**
  - Browse books by name, price (for borrowing or purchase), and cover image, search for specific books.
  - View detailed book information, including cover, ratings, publisher, language, author, ISBN, publication date, and overview.
- **Profile Management**
  - Update or delete profile picture.
  - Edit personal details, including first name, last name, email, mobile number, and address.
  - Change password (enter new password and confirm).
- **Your Book Page**
  - View list of reserved books or current reading selections.
- **Cart Management**
  - Review all books added to the cart, along with pricing details, item count, subtotal, delivery fee, and total price.
  - Remove books from the cart as needed.
- **Payment (Online and Offline)**
  - Pay using Visa, credit card, or cash on delivery.
  - Provide or update shipping address and card details (card number, expiration date, security code).
  - Review order summary, including subtotal, delivery fee, and total cost.

- **Review Page**
  - Confirm final order details before completing payment, including book list, prices, and quantity adjustments.
  - Display successful order confirmation upon completion of payment.

Admin Features

- **Admin Authentication**
  - Admin login using email and password.
  - Dedicated sign-up page for new admin accounts.
- **Dashboard**
  - Overview of all books and authors.
  - Overview of user data.
- **Review Management**
  - View and manage user reviews.
- **Member Management**
  - Oversee member activities and account details.

1.3 Motivation

The motivation behind developing a Library Management System (LMS) stem from the need to modernize and streamline library operations, improving the efficiency and accessibility of resources for both staff and patrons. Traditional libraries often face challenges such as time-consuming manual cataloging, inefficient tracking of inventory, and difficulties in managing user borrowing and returns. These issues not only burden librarians but also lead to a suboptimal experience for library users.

An LMS provides a digital solution that automates repetitive tasks, organizes resources, and simplifies access to information, making libraries more adaptable to the digital age. As libraries expand to include digital materials like e-books, journals, and multimedia, there is an increased need for a system that can handle both physical and electronic resources effectively. Additionally, today's users expect quick and easy access to

information, self-service options, and real-time notifications, all of which can be achieved through an LMS.

The motivation for this system also lies in its potential to offer analytics and insights into resource usage, which can help librarians make data-driven decisions about inventory and acquisitions. With an LMS, libraries can provide a modern, user-friendly experience while maintaining the integrity and organization of their collections, thereby maximizing the library's impact on its community**.**

The survey results are as follows:

80% of library users favor web-based LMS for its flexibility and accessibility as shown in Figure 1. Additionally, 75% of users express satisfaction with current library services as shown in Figure 2.

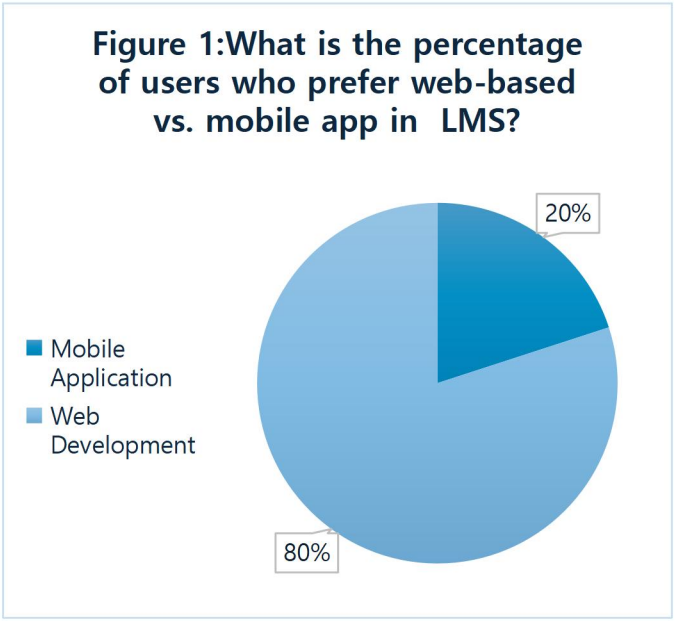


Figure 1

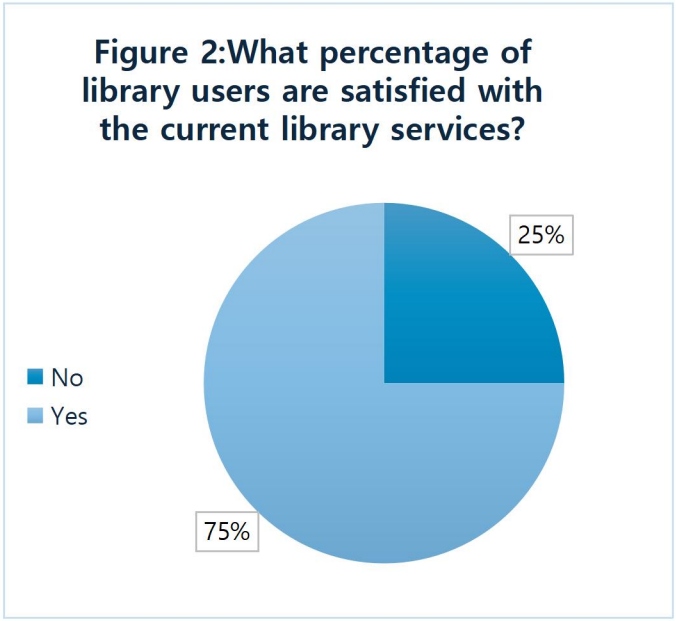


Figure2

Based on the survey results highlighted a need for a system that facilitates efficient and timely assistance. Users desire a platform where they can easily seek help and receive prompt responses.

1.4 Project Objectives

- **Automate Library Operations**

  Streamline core library tasks, including cataloging, circulation, and inventory management, to minimize manual effort and reduce human errors.

- **Improve Resource Accessibility**

  Enable users to easily search for, locate, and access both physical and digital resources, making library materials more accessible and improving user satisfaction.

- **Efficient User and Membership Management**

  Simplify the process of registering users, managing memberships, and tracking borrowing history, while ensuring that personal information is securely maintained.

- **Enhance Borrowing and Return Processes**

  Provide a clear, user-friendly borrowing and return process with automated due-date reminders and overdue notifications, ensuring effective circulation of resources.

- **Facilitate Reservations and Holds**

  Allow users to reserve items or place holds on high-demand resources, ensuring equitable access and reducing waiting times for popular materials.

- **Enable Comprehensive Inventory Tracking**

  Keep an accurate, real-time record of all resources, including location and condition status, to help staff manage resources efficiently and reduce loss.

- **Offer Detailed Reporting and Analytics**

  Generate data-driven reports on resource usage, borrowing patterns, popular items, and overdue instances, aiding in decision-making for acquisitions and library policies.

- **Ensure Data Security and User Privacy**

  Protect user information, borrowing history, and library records through secure access control, data encryption, and regular system backups.

- **Increase User Engagement and Satisfaction**

  Provide personalized notifications, seamless access, and intuitive navigation, ensuring a positive and engaging experience for all users.

- **Scalability and Flexibility**

  Design the LMS to be adaptable for different library sizes and types, ensuring that the system can grow with the library's needs.

1.5 Project Scope

The **Library Management System (LMS)** project covers the development, implementation, and management of a full-featured digital platform that supports both library patrons (users) and administrators (admins), ensuring smooth operations, modern digital services, and enhanced user experience.

Key Scope Areas:

- **User Management**
- **Patron Management:**

  ▪ User registration, login, profile management, password changes, and membership tracking.

  ▪ Managing personal details (first name, last name, mobile number, address, email) and profile pictures.

- ***Admi*n Management:**

  ▪Admin login and signup.

  ▪ Access to a full-featured dashboard for book and author management, category handling, member oversight, and review monitoring.

- **Resource and Catalog Management**
  - Comprehensive cataloging of books (physical and digital), e-books, magazines, and offline materials.
  - Metadata management: book cover image, rates, publisher, language, author, ISBN, publication date, overview.

- Categories section with book browsing, search tools, price listings (borrow and buy price), and detailed book views.

- **Circulation, Reservations, and Borrowing**
  - Book reservation system, borrowing history tracking, current reading lists, and profile-linked reservations.
  - Management of reserved and borrowed items, with the ability to cancel or delete items.

- **Cart and Payment System**
  - Full cart management, including subtotal, delivery fee, and total price.
  - Payment methods supporting both online (credit/debit card, with card info, expiration date, security code) and offline (cash on delivery) payments.
  - Editable shipping address and real-time order summary.

- **Communication and Support**
  - Contact us page allowing users to send messages, along with visible library address, phone number, and email.

- **Reviews and Feedback**
  - User-generated review pages with post-payment feedback, displaying all relevant order details (excluding sensitive card info).

- **Search and Discovery Tools**
  - Advanced search functionalities with filters by book name, author, category, etc., and personalized recommendations.

- **Reporting and Analytics (Admin Side)**
  - Custom reports on system use, book popularity, reviews, member activity, borrowing patterns, and overdue instances.

- **System Integration and Interoperability**
    - Integration with external databases and APIs for enhanced cataloging, such as Google Books or Library Thing.

Additional Considerations:

• **User Experience**: Prioritize intuitive, clean interfaces for both users and admins.
• **Security**: Apply robust encryption, secure authentication, and permissions management.
• **Scalability**: Ensure the system supports library growth, expanding collections, and increasing user bases.
• **Maintenance and Support**: Provide regular updates, backups, and technical assistance.

1.6 Literature Review

1.6.1 Introduction

This literature review explores the development and implementation of Library Management Systems, highlighting key technologies, features, and trends that have improved library operations. It aims to provide an overview of existing research to support the design of more efficient and user-friendly library systems.

1.6.2 Related Work

Several studies have examined the evolution and effectiveness of Library Management Systems (LMS) in automating library functions. Researchers such as Sharma et al. (2019) emphasized the role of LMS in improving cataloging and circulation processes through digitization. Ahmad and Raza (2020) discussed the integration of RFID and barcode technologies to enhance book tracking and reduce manual errors. Cloud-based systems, as highlighted by Kumar and Singh (2021), offer scalability and remote access, making them suitable for academic and public libraries alike. Other works have focused on user interface design and mobile access, with

studies noting increased user engagement and satisfaction. Despite advancements, challenges remain in system interoperability, data migration, and user training, suggesting the need for more adaptive and user-centered solutions.

- **Proposed LMS Architecture**

  Our LMS is designed as a **web-based system** using **ASP.NET Core** and **C#** as the primary language and framework. It leverages:

  - **Entity Framework Core** for database interactions,
  - **Firebase** for push notifications,
  - **Redis** for caching and performance improvements,
  - **SQL Server** is the main relational database.

  It addresses the shortcomings of previous systems by providing:

  - A scalable, component-based architecture using modern C# practices.
  - Real-time notifications for overdue books and reservations.
  - Enhanced performance using Redis for frequently accessed data.
  - Administrative tools for librarians to manage resources efficiently.

The use of **React** (for frontend) further enhances the interactivity and responsiveness of the user interface, making the system user-friendly for both students and librarians.

1.7 Document Organization

This project consists of five chapters. These chapters are organized to reflect the scientific steps toward our main objective. Chapter 1 introduces the project objectives, the motivation of the project, the scope of the work, and project layout.

Chapter 2 will provide system analysis, including functional and non-functional requirements, use case diagram with its description cards, sequence diagram, and the approach used in the project. In chapter 3 we will provide architectural design, class diagram, and user interface samples. Chapter 4 will present the implementation side of the project such as: the tools and languages used to develop the system. Chapter 5 will present the conclusion, future work.

Chapter 2: System Analysis

2.1 Introduction

This chapter presents a comprehensive analysis of the Library Management System (LMS). It outlines the system's functional and non-functional requirements, the interactions between users and the system, and the sequence of events in key processes. By establishing a clear understanding of these elements, we lay a solid foundation for the subsequent design, implementation, and testing phases of the project. This systematic analysis ensures the development of a robust, scalable, and user-centric LMS.

2.2 Requirements Specification

The Requirements Specification defines the set of functionalities and constraints that the LMS must satisfy. It acts as a formal agreement between the stakeholders and the development team, ensuring alignment with the library's operational needs and user expectations.

2.2.1 Functional Requirements

Functional requirements describe the specific operations, tasks, and behaviors the system must perform. For the LMS, these include:

- **User Management**
  - **User Registration:**
    - Creation of unique user accounts with identifiers (username, password).
    - Role assignment based on privileges (e.g., member, librarian, admin).
  - **User Login:**
    - Credential authentication for system access.

- Password recovery and reset mechanisms.

- **User Profile Management:**
  - Updating personal details (first name, last name, mobile number, email, address, profile picture).
  - Managing user preferences, notifications, and password changes.

- **Item and Resource Management**
  - **Cataloging:**
    - Maintenance of a comprehensive catalog including books, e-books, magazines, and offline materials.
    - Assignment of metadata such as title, author, publisher, publication date, ISBN, and descriptive overview.
  - **Check-in/Check-out:**
    - Recording borrowing and returning of items.
    - Managing reservations, holds, and current readings.
  - **Inventory Management:**
    - Tracking the physical location, status, and condition of resources.
    - Handling damaged, lost, or missing items and conducting periodic audits.

- **Circulation and Transaction Management**
  - **Loans and Returns:**
    - Processing loan requests, renewals, returns, and generating receipts.
    - Automated notifications for due dates, renewals, and overdue fines.
  - **Reservations and Holds:**
    - Allowing users to reserve or place holds on high-demand resources.

    - Automated system handling reservations based on availability.

- **Search and Discovery**
    - **Advanced Search:**
        - Search by title, author, category, keyword, or other criteria.
        - Filtering, sorting, and faceted navigation for refined results.
    - **Recommendation System:**
        - Personalized and collaborative filtering recommendations based on user activity and preferences.
- **Cart and Payment Management**
    - **Cart Features:**
        - Adding books to the cart, viewing order summaries, and editing shipping details.
    - **Payment Processing:**
        - Supporting online (card-based) and offline (cash on delivery) transactions.
        - Handling card information securely and excluding sensitive data from post-payment views.
- **Feedback and Review System**
    - Allowing users to submit reviews and feedback post-transaction.
    - Admin-side monitoring of reviews for quality control.
- **Admin-Specific Functionalities**
    - **Dashboard Management:**
        - Admin tools for managing books, users, reviews, orders, and system reports.
        - Role-based access controls ensuring separation of user and admin interfaces.

- **Reporting and Analytics**
    - Generating detailed system reports on borrowing patterns, member activity, popular items, and overdue instances.
    - Visualizing data trends through charts, graphs, and dashboards.

2.2.2 Non-Functional Requirements

Non-functional requirements define the system's overall qualities and constraints, ensuring it operates efficiently, securely, and reliably.

1. Performance Requirements

- **Response Time:** The system should respond to user interactions within a minimal, specified timeframe.
- **Throughput:** It should handle a defined number of simultaneous transactions without degradation.
- **Scalability:** The system must accommodate growth in the user base, transactions, and resources.

2. Usability Requirements

- **User Interface:** Intuitive, clean, and user-friendly interfaces for both users and admins.
- **User Experience:** Providing engaging, seamless experience across devices.

3. Security Requirements

- **Confidentiality:** Protecting sensitive user information and resource data.
- **Integrity:** Maintaining data accuracy and consistency.
- **Availability:** Ensuring system uptime and access when needed, supported by backups.

4. Maintainability Requirements

- **Modularity:** Structured in modular components for easy updates and fixes.
- **Testability:** Designed for systematic testing and validation.

5. Portability Requirements

- **Platform Independence:** Ability to run across different hardware and software environments with minimal adjustments.

2.2.3 Use case diagram

The Use Case Diagram offers a visual summary of the key interactions between system actors (users, librarians, admins) and the system itself. It helps identify system functionality from the user's perspective. Figure 3 shows the system's use case diagram.

Figure3: Use case diagram

1. Login

- **Actors:** User, Librarian
- **Description:** This case details the process by which users and librarians authenticate their access to the system.
- **Flow of Events:**
    1. The actor (User or Librarian) enters their username and password.
    2. The system validates the provided credentials against stored records.
    3. Upon successful validation, the system grants access to the appropriate user interface based on the actor's role.
    4. If the credentials are invalid, the system displays an error message and prompts the actor to re-enter their information.
- **Excludes:** Incorrect Email or Password

2. Register New User

- **Actor:** User
- **Description:** This case outlines the steps for new users to create an account within the system.
- **Flow of Events:**
    1. The user provides their personal information, including name, contact details, and address.
    2. The system validates the submitted user details for completeness and correctness.
    3. The system generates a unique user identifier and initial password for the new account.

4. The system sends a confirmation email containing an activation link to the user's provided email address.
5. The user clicks the activation link to verify their email address and activate their account.

- **Includes:** Verify Register Details

3. Search for Books

- **Actor:** User
- **Description:** This use case describes how users can locate books within the library catalog.
- **Flow of Events:**
    1. The user inputs search keywords based on criteria such as title, author, or subject.
    2. The system queries the library database for records matching the provided keywords.
    3. The system displays the search results to the user.
    4. The user can select a specific book from the results to view its detailed information.
- **Includes:** Book Information

4. Add to Cart

- **Actor:** User
- **Description:** This use case allows users to select books they intend to borrow or potentially purchase.
- **Flow of Events:**

1. The user selects one or more books to add to their virtual cart.
2. The system records the selected book numbers.
3. The system calculates the total price for the items in the cart (if purchasing).

- **Includes:** Book Numbers
- **Excludes:** Add Book Records

5. Buy Books

- **Actor:** User
- **Description:** This case details the process for users to purchase selected books.
- **Flow of Events:**
    1. The user proceeds to finalize the purchase of the items in their cart.
    2. The system prompts the user to provide payment details.
    3. The system processes the payment.
    4. The system generates confirmation of the payment.
- **Includes:** Payment Details, Confirmation Payment

6. Borrow Books

- **Actors:** User, Librarian
- **Description:** This case outlines the procedure for users to borrow physical books from the library.
- **Flow of Events:**
    1. The user presents their library card and the books they wish to borrow to the librarian.
    2. The librarian verifies the user's membership and the availability of the books.

3. The librarian checks out the books to the user.
4. The system records the borrowing transaction and updates the library's inventory.

7. Return Books

- **Actors:** User, Librarian, System
- **Description:** This use case describes the process for users to return borrowed books.
- **Flow of Events:**
    1. The user returns the borrowed book(s) to the librarian.
    2. The librarian inspects the condition of the returned book(s).
    3. The system updates the return record in the database.
    4. If the book(s) are overdue, the system calculates any applicable fines.
    5. The librarian collects the fines, or the system processes online payment.
    6. The system sends confirmation of the return to the user.
- **Includes:** Calculate Borrowing Time

8. Contact Us

- **Actor:** User
- **Description:** This case allows users to provide feedback or contact the library administration.
- **Flow of Events:**
    1. The user accesses and fills out a feedback form within the system.
    2. The system records the submitted feedback.
    3. The librarian reviews the feedback and may take appropriate actions.

- **Includes:** Filling Up Feedback Form

9. Manage Books

- **Actor:** Librarian
- **Description:** This case outlines the administrative tasks performed by the librarian to manage the library's book catalog.
- **Flow of Events:**
    1. The librarian adds records for new books to the catalog.
    2. The librarian updates details of existing book records (e.g., title, author, edition).
    3. The librarian deletes records of outdated or lost books from the catalog.
    4. The librarian searches and retrieves book records as needed.
    5. The librarian reviews and adjusts the inventory status of books.
- **Includes:** Add Book Records, Update Book Records, Delete Book Records

10. Prepare Library Database

- **Actors:** Librarian, System
- **Description:** This use case describes the process of maintaining and updating the central library database.
- **Flow of Events:**
    1. The librarian updates the database with new user registrations, book acquisitions, or system configuration changes.
    2. The system synchronizes data across different modules to ensure consistency.
- **Includes:** Send Emails.

11. Pay Fines

- **Actors:** User, System
- **Description:** This use details the process for users to pay any outstanding fines.
- **Flow of Events:**

1. The system identifies and displays any outstanding fines associated with the user's account.
2. The user initiates the payment process.
3. The system processes the payment of fines.
4. The system updates the user's account status to reflect the payment.

- **Includes:** Calculate Borrowing Time

2.3 Sequence diagram

A sequence diagram is a type of UML diagram that illustrates the interactions between objects in a system over time. In the context of an LMS, a sequence diagram can visualize the flow of actions and messages between different components of the system, such as users, the LMS system itself, and database.

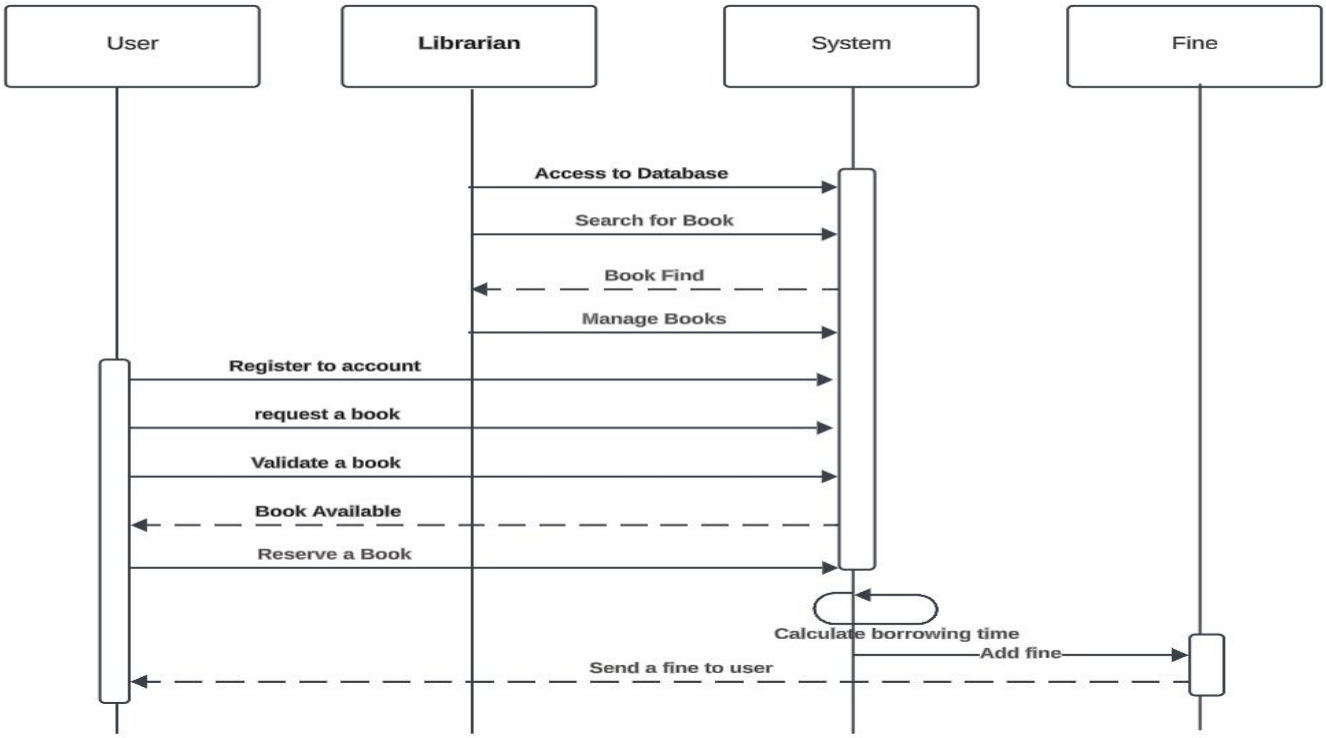


Figure4: Sequence diagram for LMS use case

2.4 Development Methodology

The LMS development adopts the Agile methodology, which focuses on iterative, incremental progress and continuous user feedback. By delivering small, functional modules in sprints, the team can respond flexibly to changing requirements and improve the system throughout its development lifecycle. Agile ensures that the LMS remains aligned with the needs of both librarians and library members, resulting in a practical, user-friendly system.

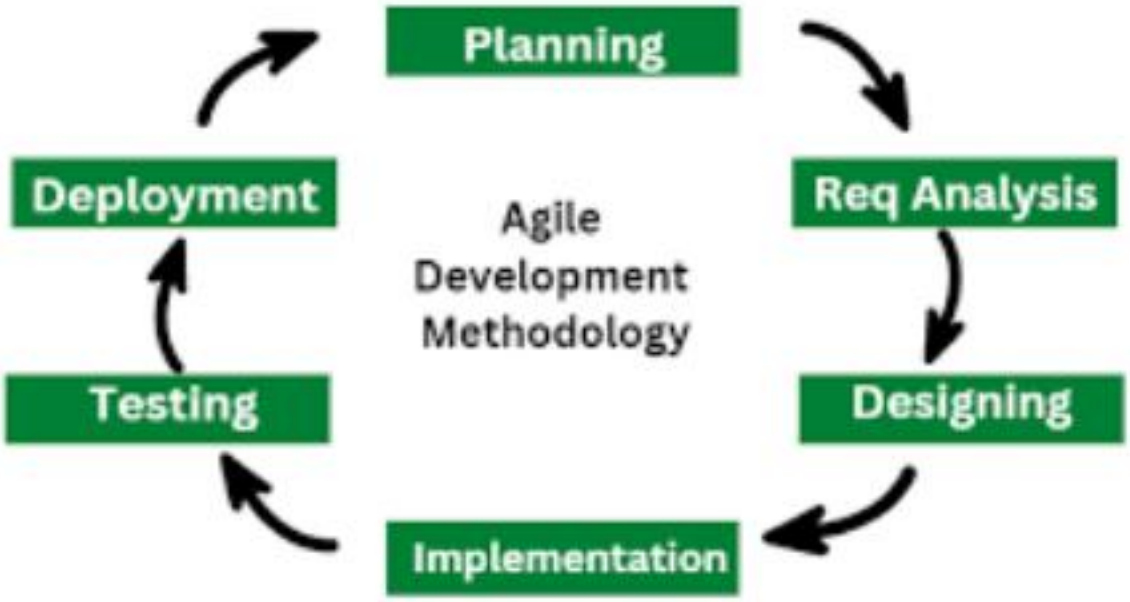


Figure5: Agile Development Methodology

Chapter 3: System Design

3.1 Introduction

System Design is the process of defining the architecture, components, modules, interfaces, and data for a system to satisfy specified requirements. It involves translating user needs and functional requirements into a detailed blueprint that guides the implementation phase.

3.2 Architectural design

Architectural Design defines how software and hardware components interact to deliver a complete system. For the LMS, we adopt a **client-server architecture** with a **microservices approach** to maximize modularity, scalability, and maintainability.

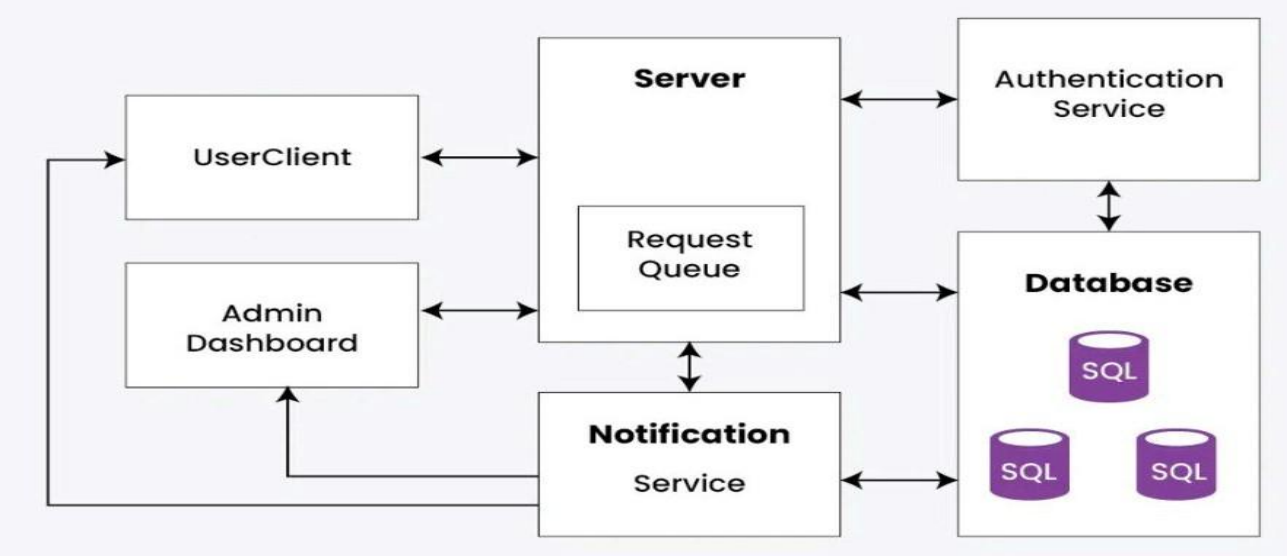


Figure6: Architecture Design

3.2.1 Client-Side Components

The client-side of the LMS is responsible for interacting with the users, presenting data, and gathering input. The key client-side components include:

- **User Interface (UI)**

  The user interface provides the visual elements and interactions necessary for user engagement. It is designed to be:

  - **Intuitive**: Easy to navigate and understand.
  - **Responsive**: Adapts to various screen sizes (desktop, tablet, mobile).

The components of the UI include:

- **Login/Registration**: Authenticates users and creates new accounts securely.
- **Search Bar**: Enables users to search for books, articles, and other library resources.
- **Catalog**: Displays a list of available resources, including metadata (e.g., title, author, publication date).
- **Item Details**: Provides detailed information about a selected resource.
- **Checkout/Return**: Allows users to check out and return books and resources.
- **Hold Requests**: Enables users to place holds on unavailable items.
- **Account Management**: Manages user profiles, preferences, and history.
- **Feedback Form**: Allows users to submit feedback on the library services.

- ## Client-Side Logic:

  The client-side logic manages user interactions, validates input, and facilitates communication with the server:

- **JavaScript Framework/Library**: Utilizes frameworks such as React, Angular, or Vue.js for structuring the app's logic.
- **Data Binding**: Ensures synchronization between UI elements and data models.

- **Error Handling**: Captures errors and provides clear user feedback.
- **Form Validation**: Ensures the accuracy and integrity of user input.
- **RESTful API Interactions**: Facilitates communication with server-side services via HTTP requests.

- **Request Manager**:

  The Request Manager handles communication between the client-side logic and the server:

- **Initiating Requests**: Sends HTTP requests for various user actions such as searching for books or placing holds.
- **Asynchronous Operations**: Manages asynchronous requests to maintain UI responsiveness.
- **Error Handling**: Detects and reports errors in user requests.
- **Data Validation**: Ensures that all data sent to the server is valid and consistent.

- **Response Manager:**

  The Response Manager processes the server's response:

- **Processing Responses**: Receives and interprets responses from the server.
- **Updating the UI**: Modifies the interface based on received data (e.g., displaying search results, updating user profiles).
- **Error Handling**: Handles server-side errors and displays appropriate messages.
- **Caching Data**: Stores frequently access data to improve performance.

- **User Profile Manager**:

  This component allows users to manage their profile details:

- **Profile Information**: Enables users to view and edit personal details such as name, email, and contact information.
- **Password Management**: Supports secure password changes.
- **Account Settings**: Allows users to configure preferences like notifications and privacy settings.
- **Feedback Manager**:

  The Feedback Manager enables users to submit feedback on the system and library services:

- **Submitting Feedback**: Allows users to provide feedback on specific system components like the search process or checkout experience.
- **Viewing Feedback History**: Displays past feedback submissions.
- **Rating Library Services**: Lets users rate library services on a scale of 1 to 5.

3.2.2 Server-Side Components

The server-side components are responsible for processing requests, managing data, and offering services to the client-side components. These components include:

- **Authentication Service:**
  - Verifies user credentials and manages user sessions.
  - Manages user sessions and access control based on roles (e.g., librarian, member).

- **Catalog Service:**
    - Stores and manages metadata about library resources.
    - Handles search, retrieval, and item availability.
- **Circulation Service:**
    - Manages the checkout and return of library items.
    - Calculates fines for overdue items and generates notices.
- **User Management Service:**
    - Handles user registrations, password resets, and account updates.
    - Maintains user profiles and preferences.
- **Reporting Service:**
    - Generates various reports, such as usage statistics, overdue reports, and inventory reports.
    - Provides insights into library usage patterns and trends.
- **Notification Service:**
    - Send notifications to users regarding reminders, alerts, and updates via email or SMS.
- **Database:**
    - Stores all critical library data, ensuring data consistency, integrity, and security.
- **Data Manager:**
    - Manages interactions with the database, including retrieving, updating, and deleting data as needed.

Data Flow

The data flow in the system proceeds through the following steps:

- **User Interaction:** Users interact with the client-side interface via a web browser.
- **Request Processing:** The client-side logic processes the user's request, which is then formatted into an API call.
- **API Request:** The client sends a request for the appropriate server-side service.
- **Server-Side Processing:** The server processes the request, accesses the database, and performs necessary operations.
- **Response Generation:** The server generates a response containing the requested data.
- **Client-Side Rendering:** The client-side logic processes the server's response and updates the UI accordingly.

3.2.3 Object Design

The object design phase focuses on identifying key objects, their attributes, methods, and interactions. A class diagram is utilized to depict the system's structure, representing classes, their operations, and relationships.

3.2.1 Class Diagram

A class diagram is a graphical representation of the system's structure via its classes and their operations, attributes and the relationships between them.

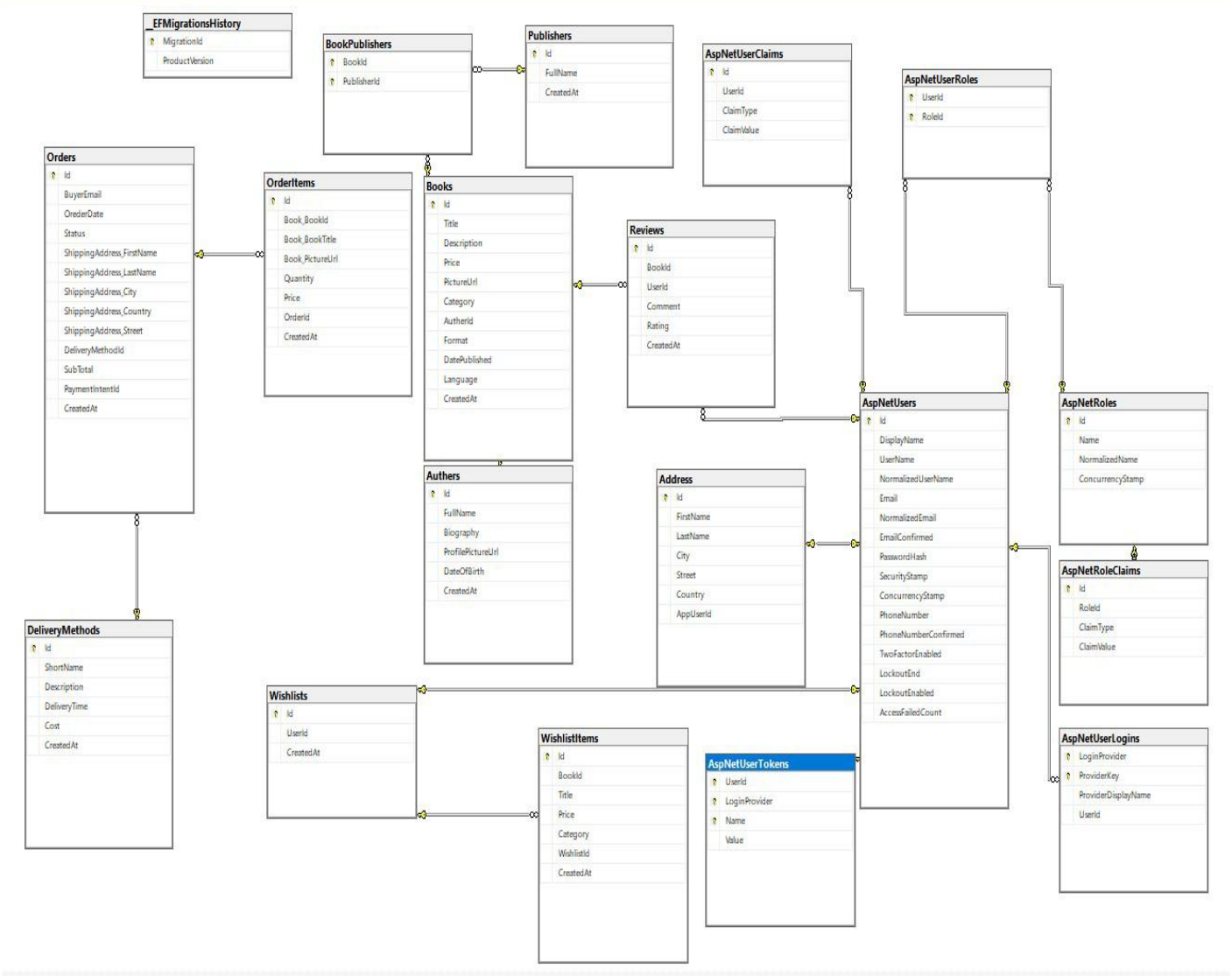


Figure7: System's class diagram

3.2.4 User Interface Design (UI)

User Interface refers to the visual elements and interactions that enable users to interact with a software application. In the context of a Library Management System (LMS), the UI plays a crucial role in providing a seamless and user-friendly experience for both librarians and patrons.

Key Components of LMS UI:

- **For user Interface:**

1. **Login Page:**

The login page provides a secure entry point for all users to access the system by providing their registered email address and password.

- Features clear fields for "Email" and "Password".
- Provides a direct link for new users to create an account "Sign up".
- Ensures the protection of user login information.
- Offers a convenient Google Sign-In option (indicated by a "G" icon).

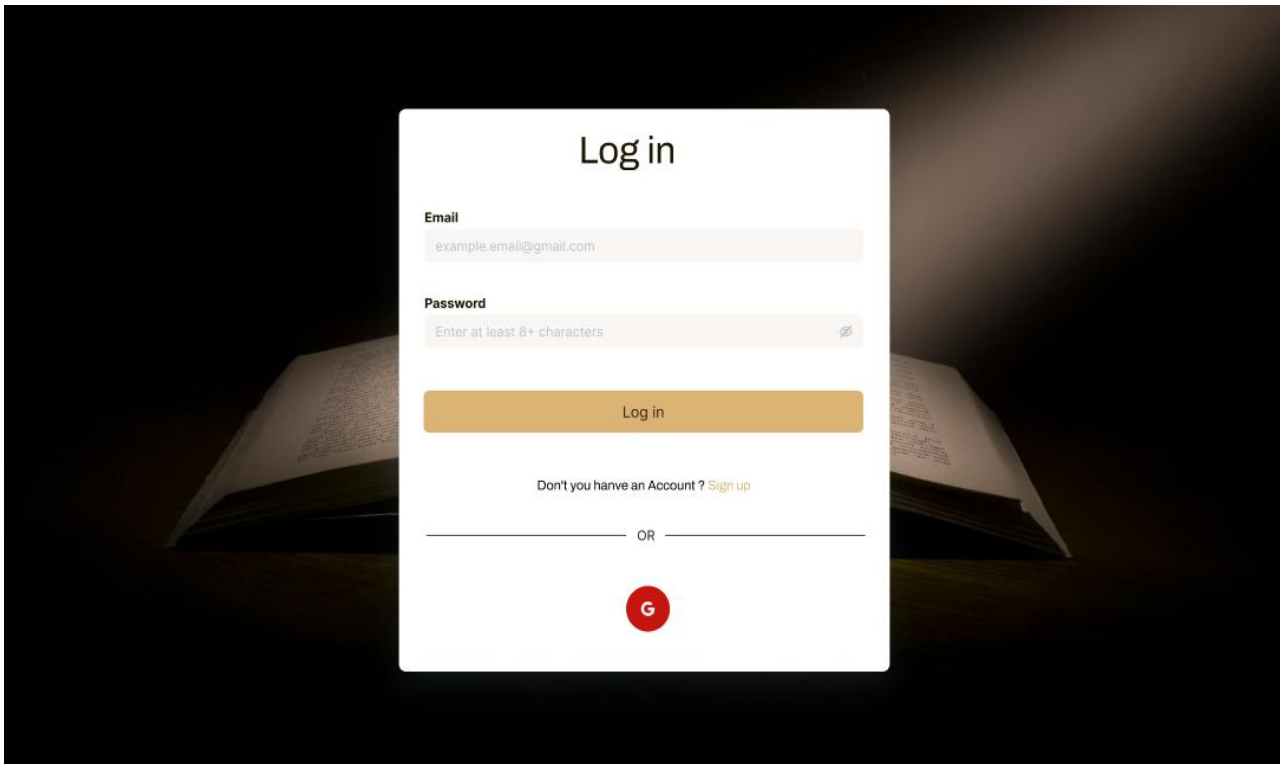


Figure8: Login UI page

2. Signup Page:

The Signup page facilitates the creation of new user accounts.

- Allow new users to create accounts.
- Collects necessary user details, including first name, last name, phone number, email address, and password (with password confirmation)
- Provides a link for existing users to "Log in".
- Send verification emails to confirm accounts.

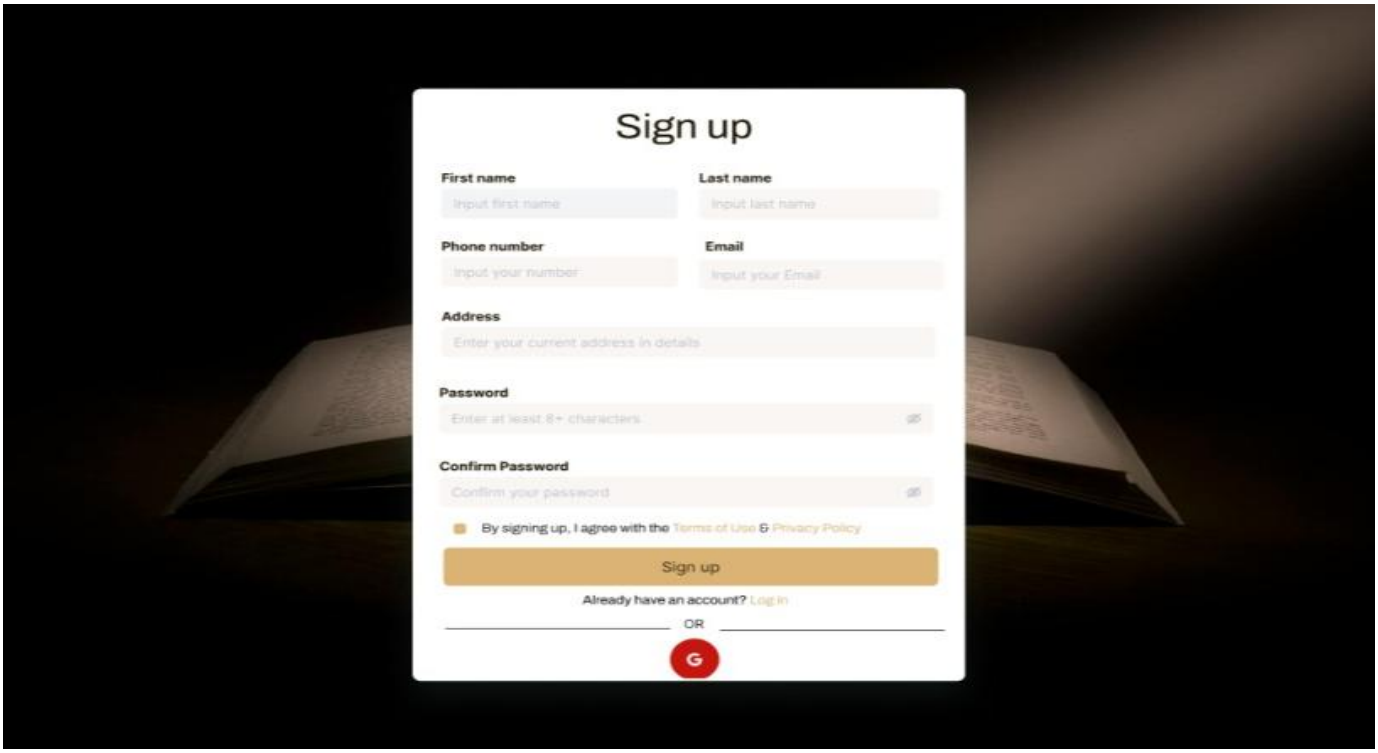


Figure9: Signup UI page

3. Categories Page:

- Displays books within a specific genre (e.g., "Mystery").
- Showcases multiple book entries within the selected category.
- Displays the book cover image to provide a visual representation.

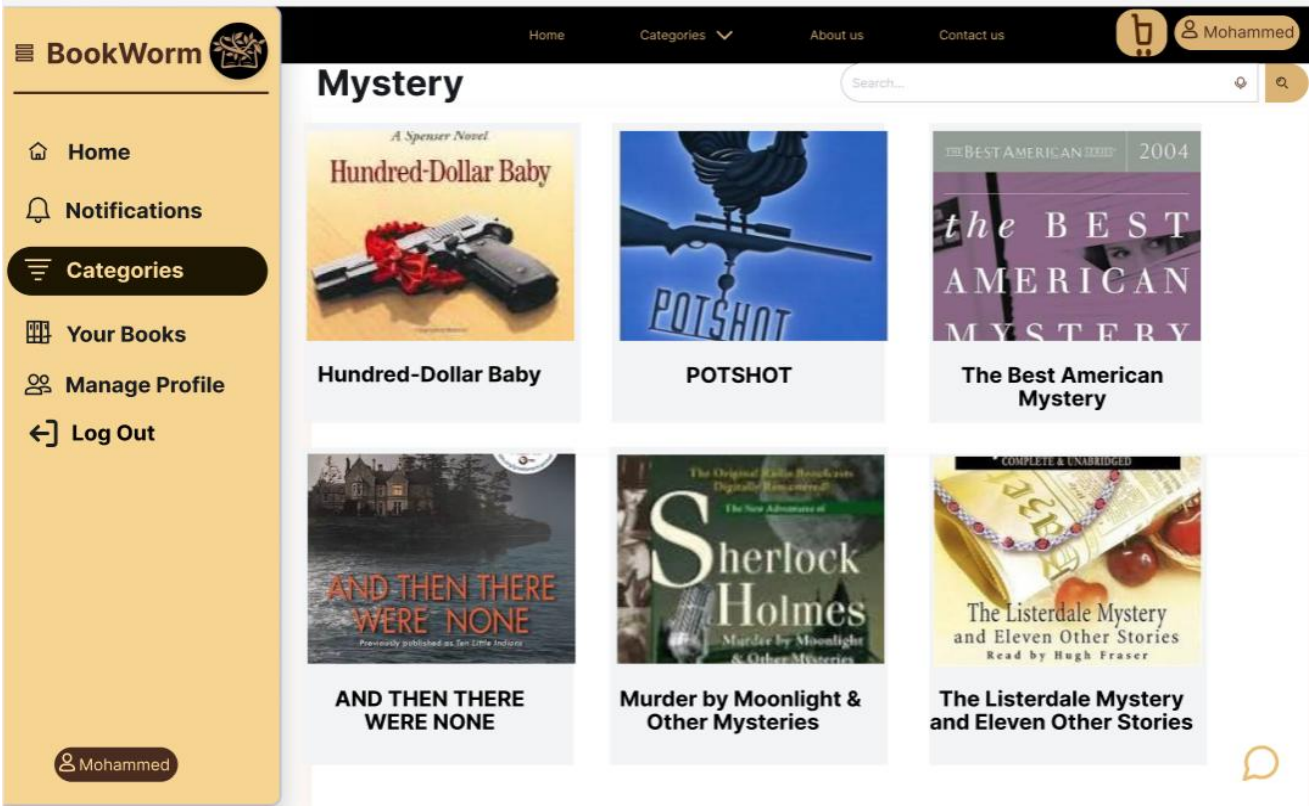


Figure10: Category UI page

4. Manage Profile page:

- Displays the user's name, email, and mobile number.
- Provides options to edit the user's first name, last name, email, and mobile number.
- Allows users to change their password by entering the old password, a new password, and confirming the new password.
- The "Save" button confirms changes made to the profile, while the "Cancel" button discards any changes and reverts to the previous information.

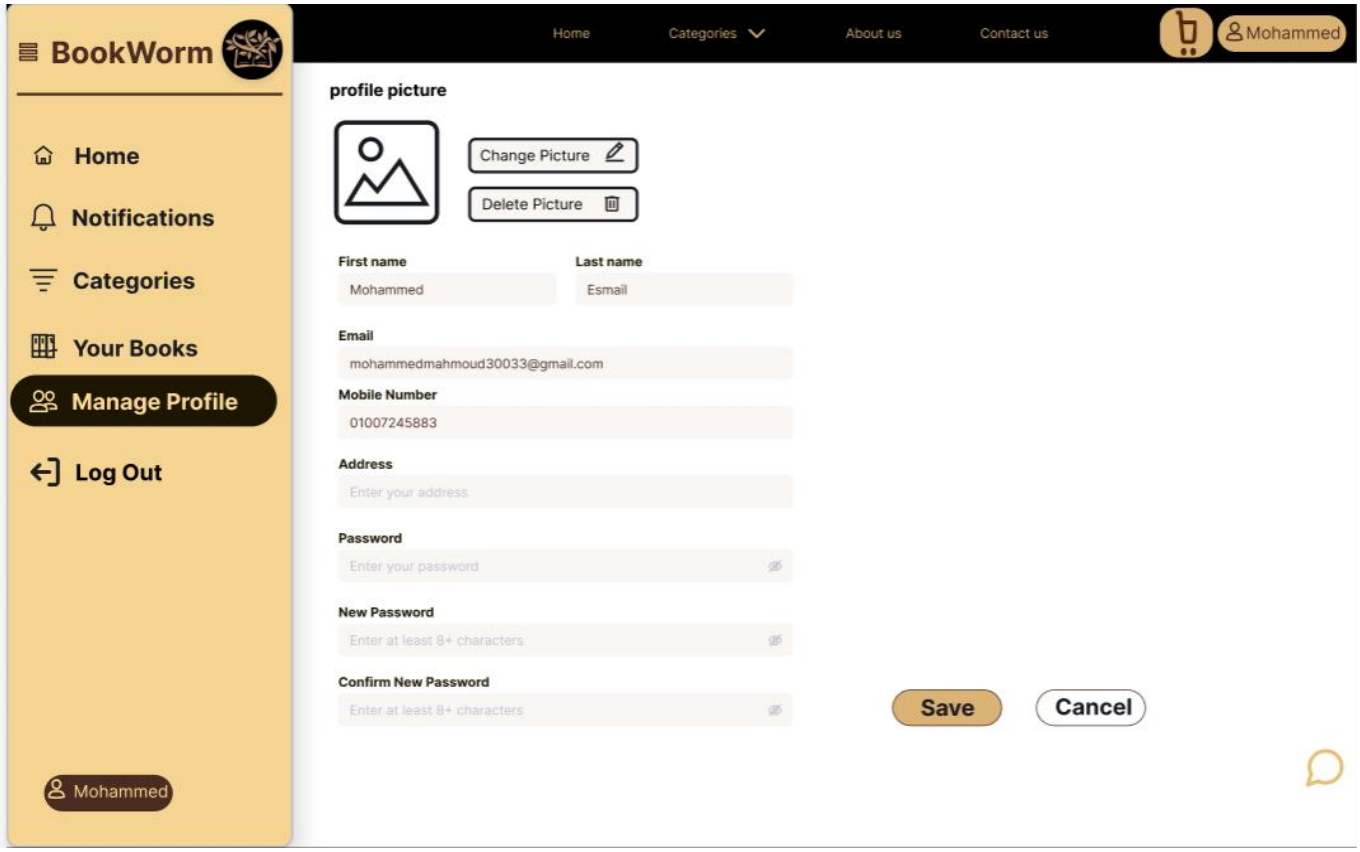


Figure11: Manage Profile UI page

5. **Cart Page:**

- Allows users to select individual items for action or select all items in the cart.
- Each item presents with its title, cover image, rating, and borrowing price.
- Provides "+" and "-" buttons to adjust the quantity of each item and offers a trash icon to remove items from the cart.
- Displays the "Subtotal" of the selected items, the "Delivery Fee," and the "Total" cost.
- Features a "Payment Page" button to navigate to the checkout process.

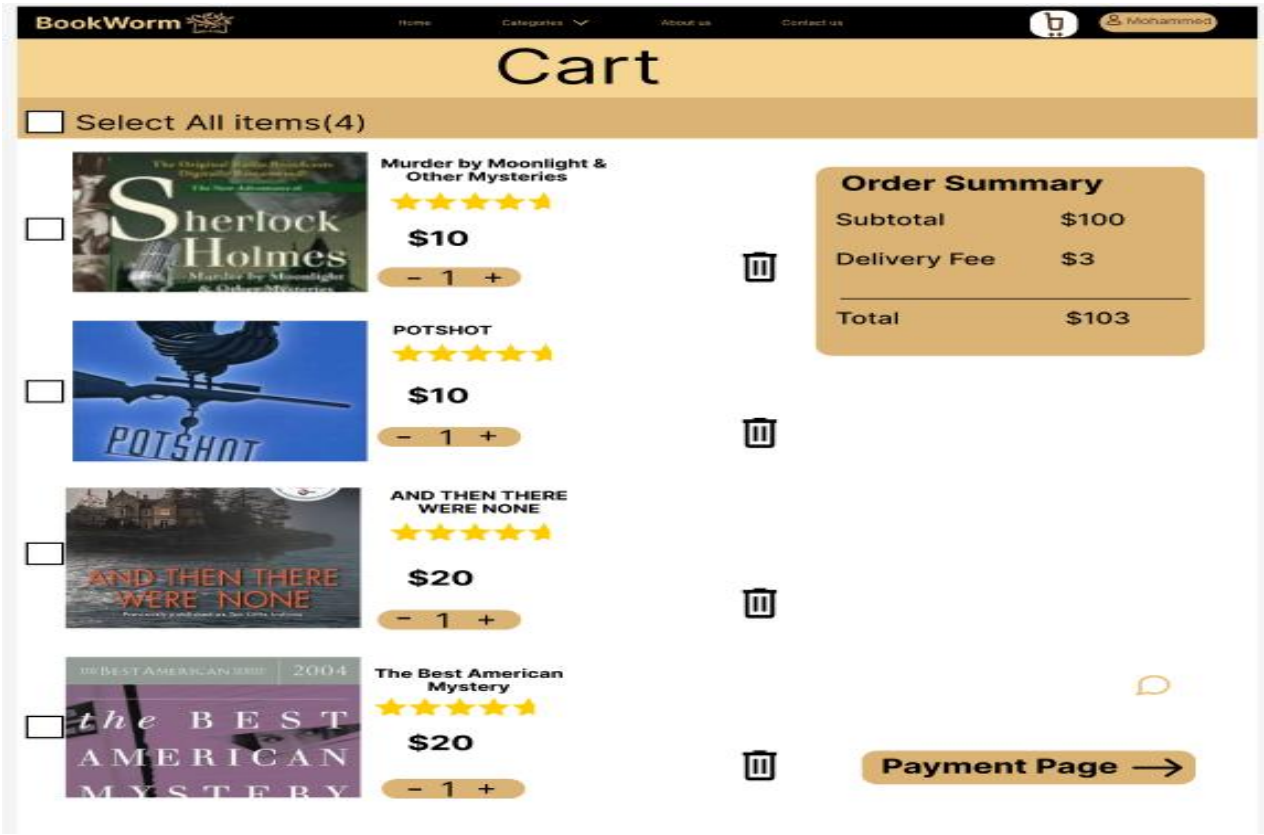


Figure12: Cart UI Page

6. Payment Pages:

- Shows the user's saved shipping address with an option to "Change" it.
- Provides fields for entering credit card details: "Card No.", "Expiration Date," and "Security Code."
- Offers a checkbox for "Cash on Delivery" as an alternative payment method.
- Presents a breakdown of the order cost: "Subtotal," "Delivery Fee," and "Total."
- Features a "Review Page" button to navigate to the order review before final confirmation.

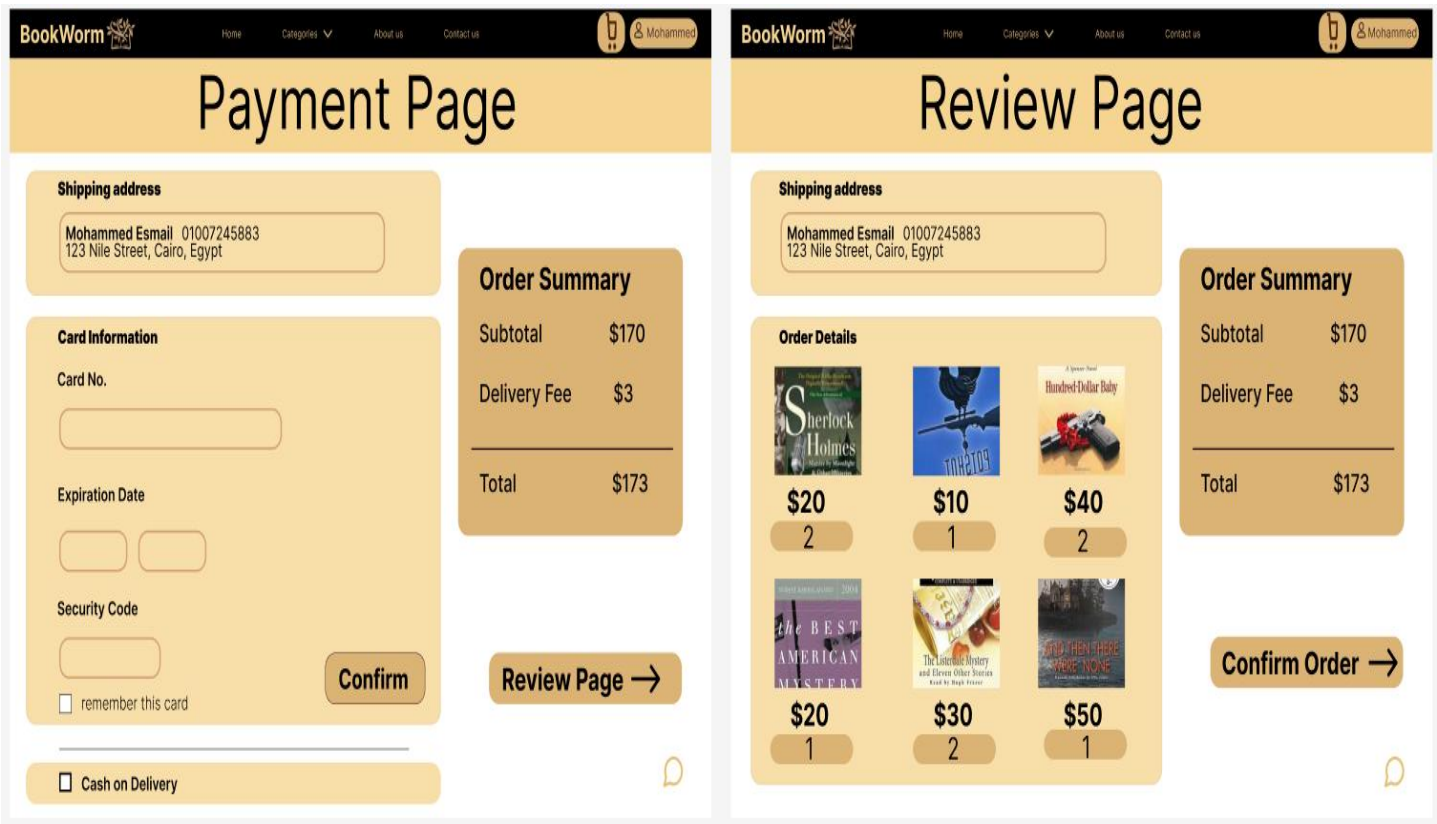


Figure13: Payment UI Pages

- **For Admin Interface:**

1. **Members Page:**
    - Displays a comprehensive list of registered users within the system.
    - Presents key information for each member, including:
        - Register Date: The date when the user registered.
        - Name: The full name of the user.
        - Email: The user's email address.
        - Address: The user's physical address.
        - Phone: The user's phone number.
    - Shows the ID of the book currently borrowed by the user.

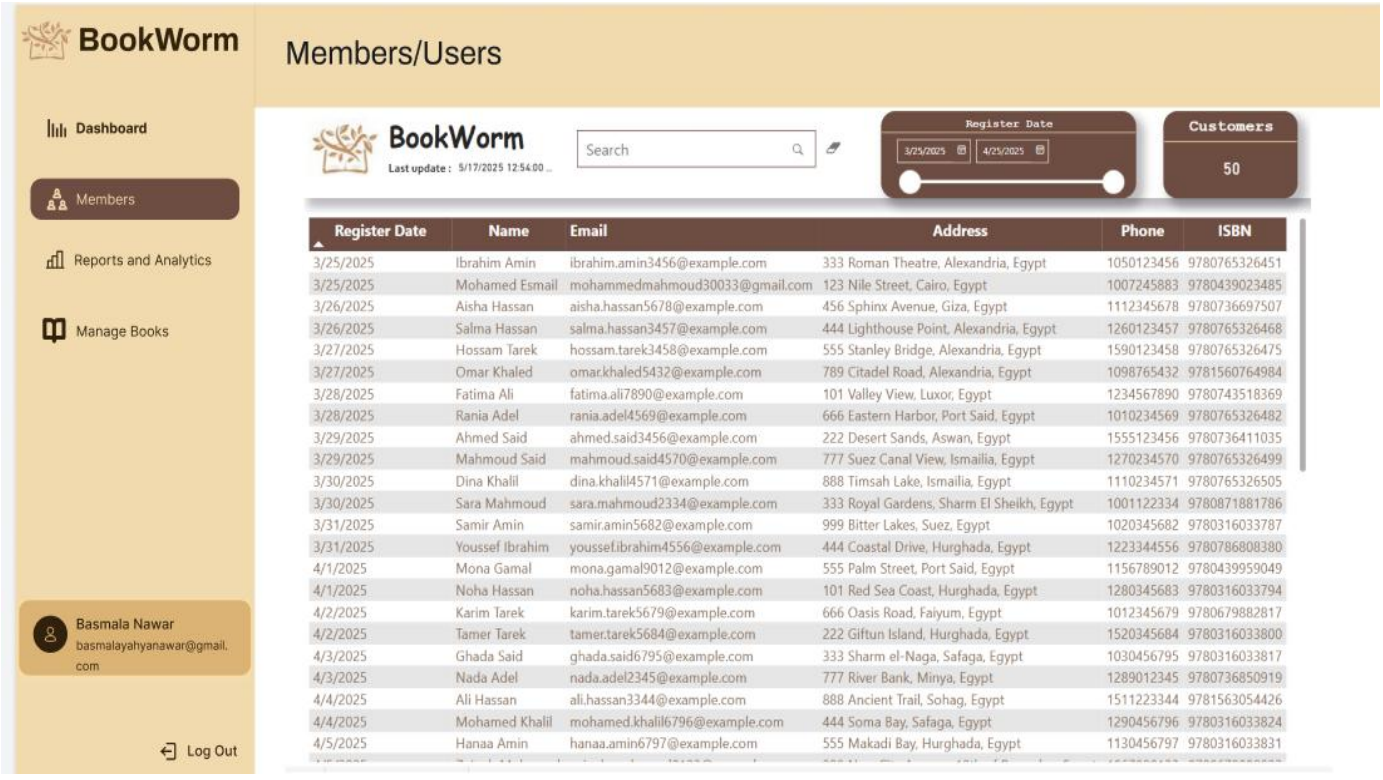


Figure14: Member UI Page

2. Dashboard Pages:

- **For Overall Book Data**
    - Displays the total count, average price of books currently available in the library's catalog.
    - Presents a visual representation showing the trend of book publications over time.
    - Displays a visual breakdown illustrating the distribution of book statuses.
    - Showcases a list or chart of the most expensive books in the library's collection, potentially with their prices.
    - Presents a list or chart of books with the highest page counts.

Figure15: Dashboard for Books

- **For Author Performance Metrices**
    - Displays the average rating and total number of authors within the library's system.
    - Indicates the number of different languages in which the library's books are written
    - Shows the total number of publishers whose books are included in the library's collection
    - resents a visual representation showing the relationship between authors.
    - Displays a bar chart showing the number of books written by specific authors.

Figure16: Dashboard for Authors

Chapter 4: System Implementation

4.1 Introduction

This chapter will cover the implementation side of the project such as: the tools and languages used to develop the system, most important codes, results of the project and system testing including unit tests, integration tests and usability tests.

4.2 Tools and Languages

The LMS was developed using a blend of modern and reliable tools and technologies, chosen for their suitability in building a scalable and maintainable library management solution.

- **Frontend Development:**
  - **Language:** HTML, CSS, and JavaScript - the fundamental building blocks of web development, used to create the user interface and provide basic interactivity.
  - **Framework:** A lightweight framework like **React** will be used as the main framework, offering component-based architecture, efficient state management, and strong support for building a responsive, interactive user interface.
- **Backend Development:**
  - **Language:** C# — a powerful, object-oriented language well-suited for building robust backend logic and managing data effectively.
  - **Framework:** .NET — a comprehensive framework that provides essential tools for web development, including routing, request handling, dependency injection, and rendering, enabling scalable and maintainable applications.
- **Database:** SQL Server - A powerful and reliable relational database management system (RDBMS) for storing and managing application data, offering scalability, security, and advanced features.

- **Development Tools:**
  - **IDE:** VS Code or any text editor with syntax highlighting - for writing and editing code.
  - **Version Control:** Git - for tracking changes and collaborating on the codebase.

4.3 Main/Most Important Codes

The most important functionalities in this LMS (Library Management System) project include account creation, book borrowing/returning, and user management. In this section, we will highlight a crucial piece of backend code responsible for configuring the application's services. The remaining code samples are provided in Appendix A.

First: Application Configuration and Service Registration (Program.cs)

This section of the code is responsible for initializing and configuring services at application startup using the .NET framework.

Code Description:

This code is responsible for setting up the core infrastructure of the LMS web application. It registers essential services such as database connectivity, Firebase, Redis, and other configurations required for the system to function properly.

1. Environment Setup:

- This code initializes the **web application builder** and detects the current environment (e.g., Development, Staging, Production).
- It ensures the app loads the correct settings based on environment variables.

```
13  var builder = WebApplication.CreateBuilder(new WebApplicationOptions
14  {
15      Args = args,
16      EnvironmentName = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT") ?? "Development"
17  });
18
```

Figure17: Environment Setup Code

2. Configuration File Loading:

- Dynamically loads the configuration file (e.g., appsettings.Development.json) that matches the current environment.
- This allows the system to use different settings (such as database connection strings) for development or production.

```
18
19    builder.Configuration.AddJsonFile($"appsettings.{builder.Environment.EnvironmentName}.json", optional: true);
20
```

Figure18: Configuration File Code

3. Database Context Registration:

- Registers the **Entity Framework Core DBContext** (LibraryDbContext) with dependency injection.
- Connects the LMS to a **SQL Server database**, using the connection string defined in the app settings file.
- Enables data operations for features like:
    - User registration and login
    - Book catalog management
    - Borrowing and returning books
    - Transaction logs

```
22  builder.Services.AddDbContext<LibraryDbContext>(options =>
23  {
24      options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
25  });
```

Figure19: Database Registration Code

4. Firebase Configuration:

- Initializes the **Firebase Admin SDK**.
- Used for:
    - Sending **notifications** (e.g., overdue reminders)
    - Possibly integrating **authentication** (if Firebase Auth is used)
    - Managing asynchronous communication (if needed)

```
26    // Get Redis configuration from appsettings.json
27    FirebaseApp.Create(new AppOptions
28    {
29        Credential = GoogleCredential.FromFile("wwwroot/config/firebase-admin.json")
30    });
```

Figure20: Firebase Configuration Code

5. Redis Setup:

- Configures **Redis** as a singleton service.
- Uses Redis for:
    - **Caching frequently used data** like book lists to improve performance
    - **Session management**
    - **Pub/Sub messaging** for real-time updates (if implemented)

```
32    // Register Redis ConnectionMultiplexer as a singleton
33
34    builder.Services.AddSingleton<IConnectionMultiplexer>(options =>
35    {
36        var connection = builder.Configuration.GetConnectionString("RedisConnection");
37        return ConnectionMultiplexer.Connect(connection);
38    });
39
```

Figure21: Redis Setup Code

4.4 System Testing

System testing ensures that all components of the Library Management System (LMS) work

Test Case	Input Value	Expected Output	Pass / Fail
Enter valid details	Correct user details (name, email, password, etc.)	A new user profile is created and stored in the database	Pass

Test Case	Input Value	Expected Output	Pass / Fail
Correct credentials	Valid email and password	User is logged in and redirected to home page	Pass
Incorrect credentials	Wrong email or password	The system displays error messages and blocks access	Pass

together as expected, covering both functional and non-functional requirements. Below is a structured breakdown of the planned system tests.

4.4.1 Unit Testing

We focus on verifying that individual components behave correctly before integrating them into the full system.

Table 1: Signup Test Plan

Enter invalid details	Missing fields, invalid email format	System rejects submission and prompts user to correct inputs	Pass

Table 2: Login Test Plan

Table 3: Search Book Test Plan

Table 4: Borrow Book Test Plan

Test Case	Input Value	Expected Output	Pass / Fail

Test Case	Input Value	Expected Output	Pass / Fail
Valid search query	Book title or author name	System displays matching book records	Pass
Invalid search query	Gibberish or empty query	System shows “No results found” or similar feedback	Pass

Request available book	Select book, click borrow	System updates book status to borrowed and assigns due date	Pass
Request unavailable book	Select book already borrowed	System notifies users that the book is currently unavailable	Pass

Table 5: Return Book Test Plan

Test Case	Input Value	Expected Output	Pass / Fail
Return within deadline	Select borrowed book, return	System marks book as available, no fine applied	Pass
Return after deadline	Select overdue book, return	System calculates fine and notifies user	Pass

Table 6: Notification Test Plan

Test Case	Input Value	Expected Output	Pass / Fail
Overdue notification	Due date passed	System sends automatic notifications to users about overdue status	Pass

4.4.2 Integration Testing

Table 7: Integration Test Plan

Scenario	Steps	Expected Result
Signup → Login → Borrow Book	User signs up → logs in → searches book → borrows it	System tracks user session, allows book borrowing, updates status
Borrow → Return → Fine	User borrows book → keeps it past due date → returns it	System calculates fine, updates user account, resets book status
Admin Add Book → User Search	Librarian adds new book → user searches using title or author	Book appears in search results, available for reservation or borrowing
Notification on Due Date	User borrows book → system tracks borrowing period → sends notification on due date or overdue	User receives timely reminder to return book

4.4.3 System Testing (Full System)

Table 8: System Test Plan

Test Area	Description
Performance	Test system under load (e.g., multiple users searching, borrowing, and returning simultaneously)
Security	Ensure password encryption, restrict admin functions to librarians only
Usability	Verify UI responsiveness on desktop and mobile, check ease of navigation
Error Handling	Simulate system errors (e.g., database offline) and confirm graceful failure messages
Compatibility	Test across multiple browsers (Chrome, Firefox, Edge) and devices

4.4.4 Acceptance Testing

Table 9: Acceptance Test Plan

Test Case	Acceptance Criteria	Result
Users can sign up and log in	User successfully creates account and accesses system	Pass
Users can search, borrow, and return books	System correctly processes each action and updates records	Pass
User receives notifications	Users are alerted when books are due or overdue	Pass
Admin can manage books	Admin can add, edit, and delete book records	Pass
System handles peak load	System remains stable with multiple simultaneous users	Pass

4.5 System Scenarios

We are going to present two of our main scenarios in the Library Management System (LMS). Both scenarios require the user to have an account and to log in to the system.

Scenario 1: User Borrowing and Returning Books

A registered user logs into the LMS, searches for books, and checks availability. If available, they borrow the book, and the system updates the status and assigns a due date. If unavailable, the user can reserve it. Users can track borrowed books and due dates in their profile. If overdue, the system calculates a fine and notifies the user. Returns can be made online or at the library, and fines can be paid through the system.

Scenario 2: Librarian Managing Library Resources

The librarian logs into the LMS to manage the book collection by adding, updating, or removing records. They validate borrow and reservation requests, monitor overdue books, and manage user notifications. The librarian can also generate reports on borrowing activities, fines, and system usage to support library operations and decision-making.

Chapter 5: Conclusion and Future Work

5.1 Future Work

The developed Library Management System (LMS) currently offers a solid set of core functionalities, efficiently supporting book search, reservations, borrowing, returns, account management, and automated notifications to alert users when borrowing periods are nearing expiration or have ended. Looking ahead, several key enhancements are planned to improve the system's security, performance, and user experience. These improvements include strengthening data security by implementing robust encryption for sensitive information such as user passwords, improving system performance by applying caching strategies to reduce server load and speed up repeated tasks, and expanding platform availability by developing dedicated mobile applications for systems such as Ios and potentially Android. These enhancements aim to make the LMS more secure, efficient, and better suited to meet the future needs of both users and librarians.

5.2 Conclusion

The developed Library Management System (LMS) successfully automates key library functions, including book search, reservations, borrowing, returns, user account management, and sending automated notifications for due and overdue items. The system improves resource management, enhances user experience, and streamlines librarian tasks. Despite time and resource limitations, it provides a strong, adaptable foundation with clear potential for future upgrades and long-term success.

References

Book Resources:

- **Amazon Books**

  https://www.amazon.com/books

- **Google Books**

  https://books.google.com

- **Internet Archive – Books Collection**

  https://archive.org/details/books

- **Project Gutenberg – Free eBooks**

  https://www.gutenberg.org

- **Many Books**

  https://manybooks.net

- **Open Library**

  https://openlibrary.org

AI Resources:

- **ChatGPT (OpenAI)**

  https://chat.openai.com

- **Google Gemini**

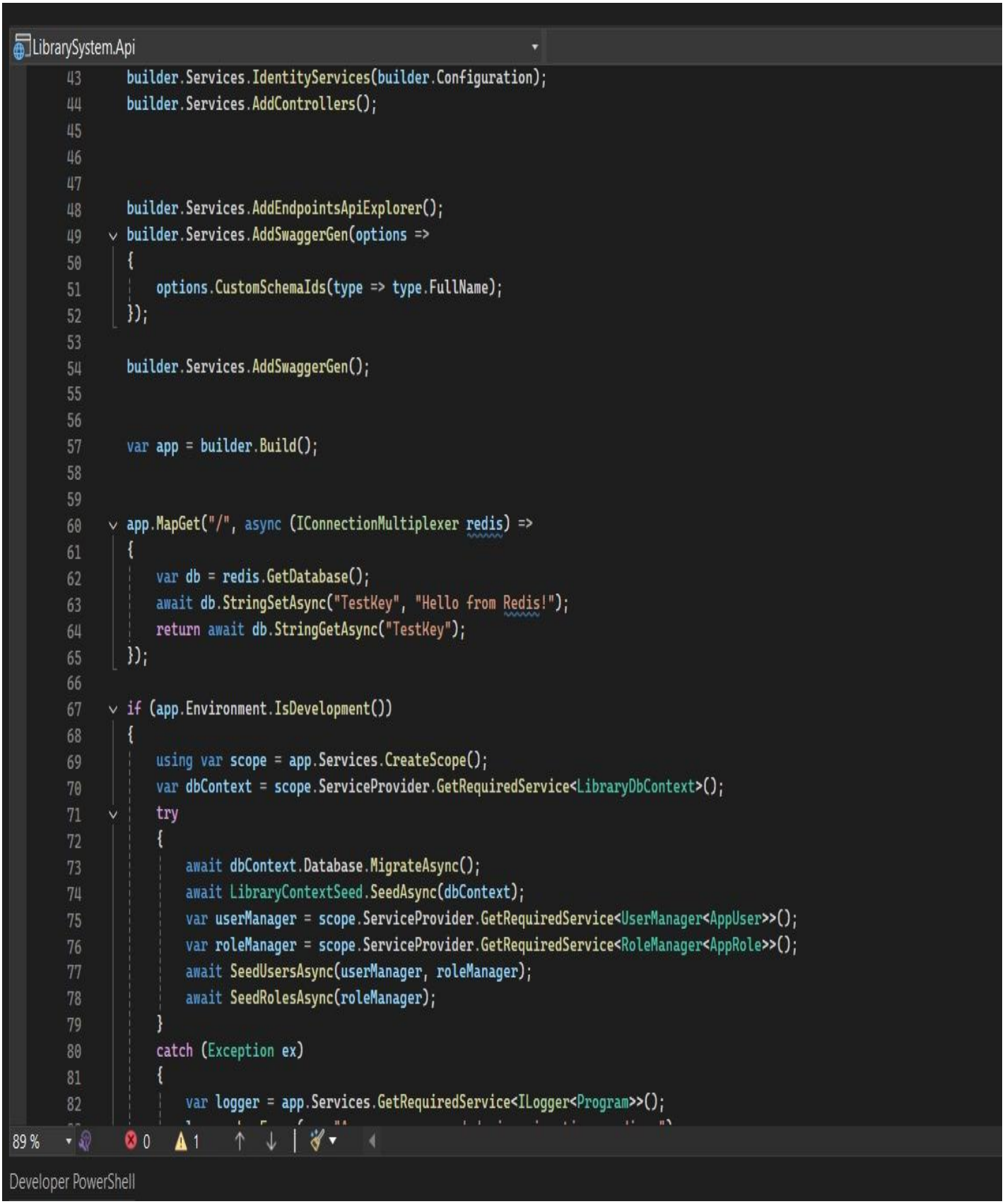
  https://gemini.google.com

Appendix A

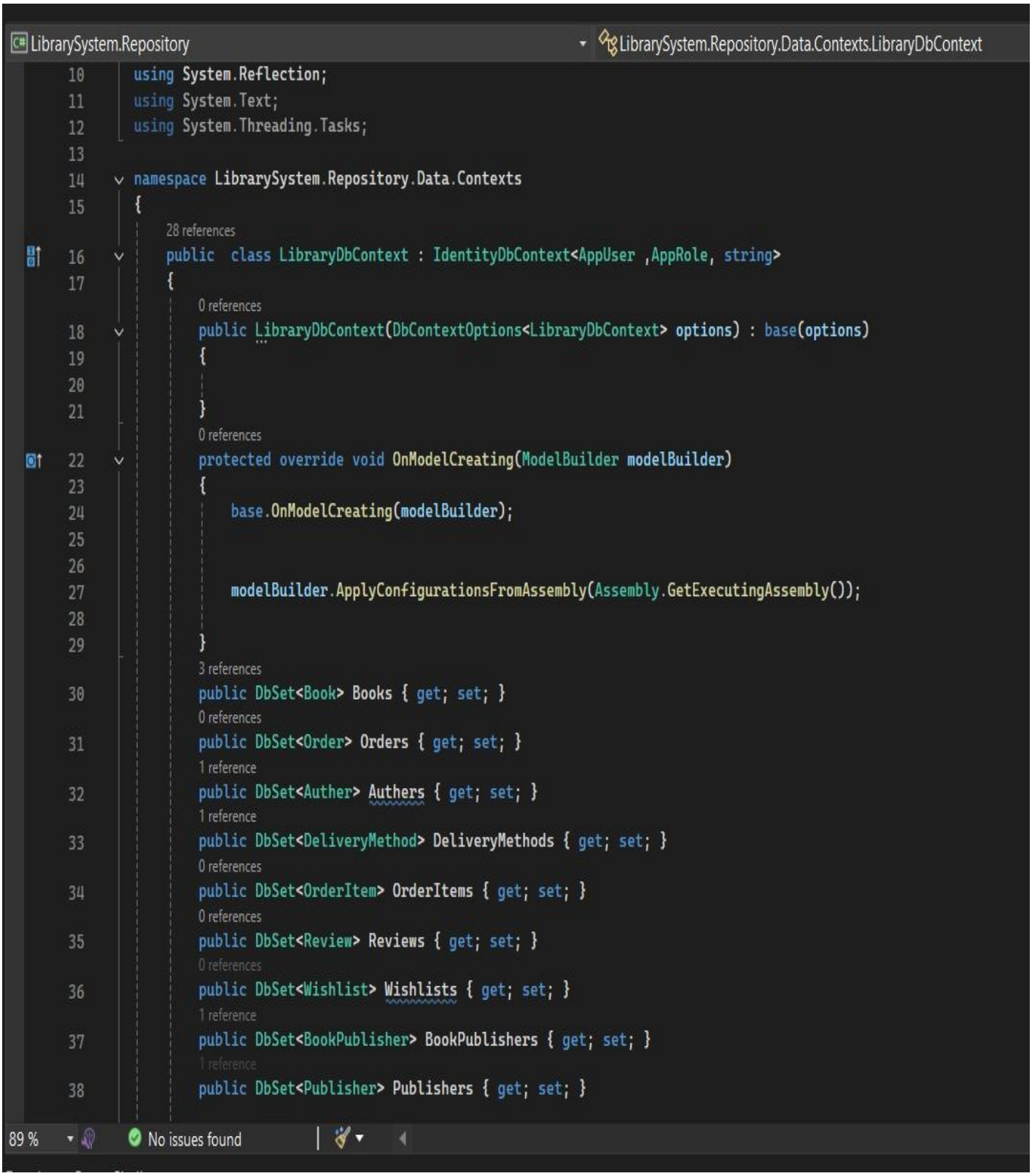
LibrarySystem API Startup Configuration:

LibrarySystem.Api

```
using FirebaseAdmin;
using Google.Apis.Auth.OAuth2;
using LibrarySystem.Api.Extensions;
using LibrarySystem.Api.Middlewares;
using LibrarySystem.Core.Entitties.Identity;
using LibrarySystem.Repository.Data.Contexts;
using Microsoft.AspNetCore.Authentication.Cookies;
using Microsoft.AspNetCore.Identity;
using Microsoft.EntityFrameworkCore;
using StackExchange.Redis;
using static LibrarySystem.Repository.Data.Contexts.IdentityContextSeed;

var builder = WebApplication.CreateBuilder(new WebApplicationOptions
{
    Args = args,
    EnvironmentName = Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT") ?? "Development"
});

builder.Configuration.AddJsonFile($"appsettings.{builder.Environment.EnvironmentName}.json", optional: true);


builder.Services.AddDbContext<LibraryDbContext>(options =>
{
    options.UseSqlServer(builder.Configuration.GetConnectionString("DefaultConnection"));
});
// Get Redis configuration from appsettings.json
FirebaseApp.Create(new AppOptions
{
    Credential = GoogleCredential.FromFile("wwwroot/config/firebase-admin.json")
});

// Register Redis ConnectionMultiplexer as a singleton

builder.Services.AddSingleton<IConnectionMultiplexer>(options =>
{
    var connection = builder.Configuration.GetConnectionString("RedisConnection");
    return ConnectionMultiplexer.Connect(connection);
});

// Add authentication services
```


LibrarySystem.Api
builder.Services.IdentityServices(builder.Configuration);
builder.Services.AddControllers();
builder.Services.AddEndpointsApiExplorer();
builder.Services.AddSwaggerGen(options =>
{
options.CustomSchemaIds(type => type.FullName);
});
builder.Services.AddSwaggerGen();
var app = builder.Build();
app.MapGet("/", async (IConnectionMultiplexer redis) =>
{
var db = redis.GetDatabase();
await db.StringSetAsync("TestKey", "Hello from Redis!");
return await db.StringGetAsync("TestKey");
});
if (app.Environment.IsDevelopment())
{
using var scope = app.Services.CreateScope();
var dbContext = scope.ServiceProvider.GetRequiredService<LibraryDbContext>();
try
{
await dbContext.Database.MigrateAsync();
await LibraryContextSeed.SeedAsync(dbContext);
var userManager = scope.ServiceProvider.GetRequiredService<UserManager<AppUser>>();
var roleManager = scope.ServiceProvider.GetRequiredService<RoleManager<AppRole>>();
await SeedUsersAsync(userManager, roleManager);
await SeedRolesAsync(roleManager);
}
catch (Exception ex)
{
var logger = app.Services.GetRequiredService<ILogger<Program>>();
89 %
0
1
Developer PowerShell

LibraryDBContext Class Definition



BookController Class Definition

```
namespace LibrarySystem.Api.Controllers
{
    1 reference
    public class BookController : ApiControllerBase
    {
        private readonly IUnitOfWork _unitOfWork;
        private readonly IBookService _bookService;
        private readonly IMapper _mapper;

        0 references
        public BookController(IUnitOfWork unitOfWork , IBookService bookService , IMapper mapper)
        {
            _unitOfWork = unitOfWork;
            _bookService = bookService;
            _mapper = mapper;
        }

       // [Authorize(Roles = "Admin,Librarian")]
        [HttpPost]

        0 references
        public async Task<ActionResult<BookDTO>> AddBookAsync(BookDTO bookDTO)
        {
            if (bookDTO == null)
                return BadRequest(new ApiResponse(400, "Invalid book data"));

            var existingBook = await _bookService.ExistsBookAsync(bookDTO.Title);

            if (existingBook != null)
                return BadRequest(new ApiResponse(400, "Book already exists"));


            var book = _mapper.Map<BookDTO, Book>(bookDTO);

            await _unitOfWork.Repository<Book>().AddAsync(book);
            var result = await _unitOfWork.CompleteAsync();

            var isBookAdded = await _bookService.AddBookWithPublishersAsync(book, bookDTO.Publishers);

            if (!isBookAdded)
```

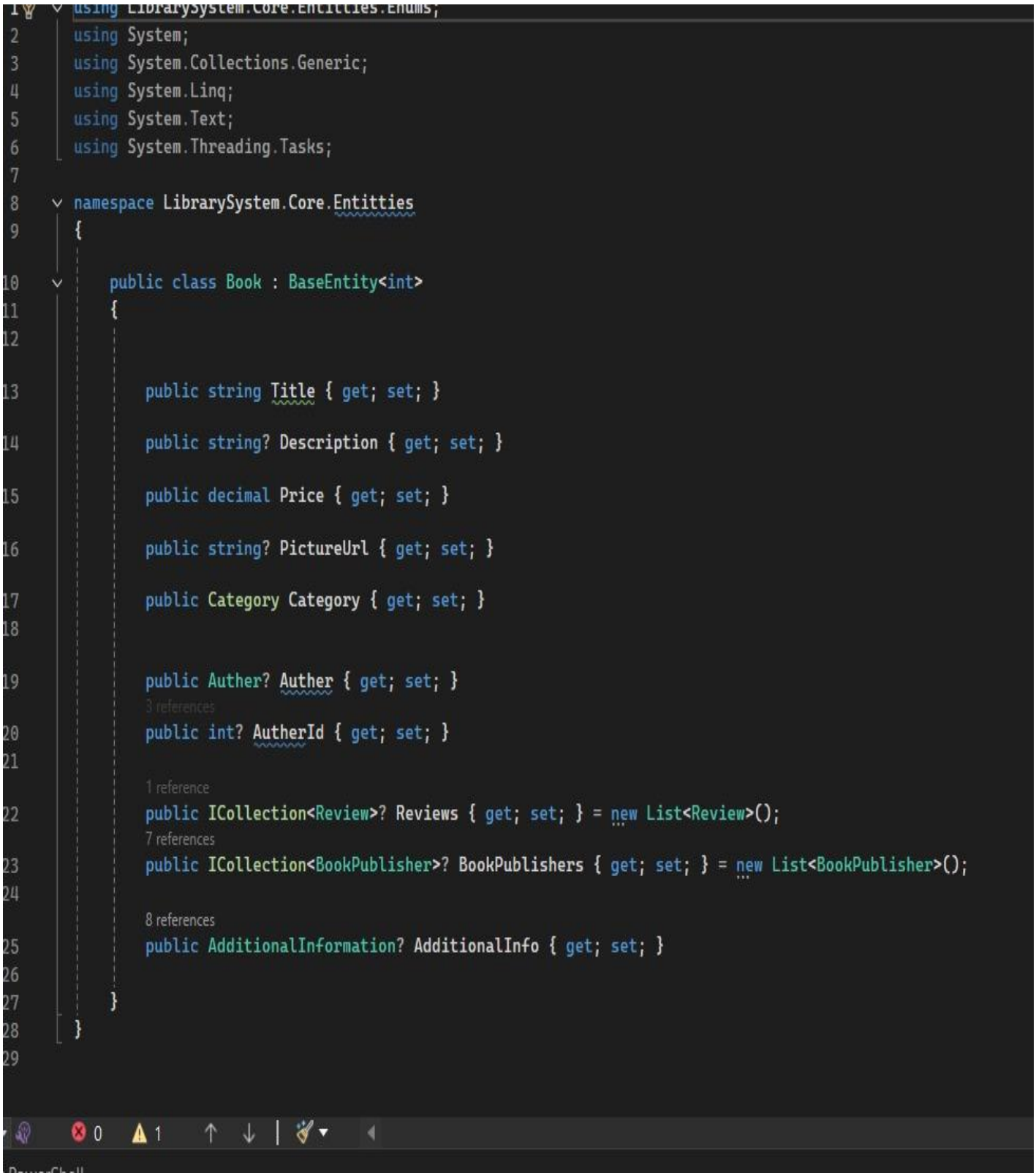
PaymentController Class Definition

```
using LibrarySystem.Api.DTOs;
using LibrarySystem.Api.Errors;
using LibrarySystem.Core.Services.Contract;
using Microsoft.AspNetCore.Mvc;
using Stripe;

namespace LibrarySystem.Api.Controllers
{
    1 reference
    public class PaymentController : ApiControllerBase
    {
        private readonly IPaymentService _paymentService;
        private readonly IConfiguration _configuration;

        0 references
        public PaymentController(IPaymentService paymentService , IConfiguration configuration)
        {
            _paymentService = paymentService;
            _configuration = configuration;
        }

        [HttpPost("{BasketId}")]
        [ProducesResponseType(typeof(CustomerBasketDto), StatusCodes.Status200OK)]
        [ProducesResponseType(typeof(ApiResponse), StatusCodes.Status400BadRequest)]
        0 references
        public async Task<ActionResult<CustomerBasketDto>> CreateOrUpdatePaymentIntent(string BasketId)
        {
            var Basket =await _paymentService.CreateOrUpdatePaymentIntentId(BasketId);
            if (Basket is null)
                return BadRequest(new ApiResponse(400, "There is Problem With Your Basket !"));
            return Ok(Basket);
        }

        [HttpPost("Webhook")] //stripe listen --forward-to https://localhost:7194/api/payment/Webhook
        0 references
        public async Task<ActionResult> Stripewebhook()
        {
            var json = await new StreamReader(HttpContext.Request.Body).ReadToEndAsync();
            var stripeSignature = Request.Headers["Stripe-Signature"];
            if (string.IsNullOrEmpty(stripeSignature))
            {
```

No issues found

werShell

Book Entity Class Definition



```
using LibrarySystem.Core.Entitties.Enums;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace LibrarySystem.Core.Entitties
{

    public class Book : BaseEntity<int>
    {



        public string Title { get; set; }

        public string? Description { get; set; }

        public decimal Price { get; set; }

        public string? PictureUrl { get; set; }

        public Category Category { get; set; }


        public Auther? Auther { get; set; }
        3 references
        public int? AutherId { get; set; }

        1 reference
        public ICollection<Review>? Reviews { get; set; } = new List<Review>();
        7 references
        public ICollection<BookPublisher>? BookPublishers { get; set; } = new List<BookPublisher>();

        8 references
        public AdditionalInformation? AdditionalInfo { get; set; }


    }
}
```