

|  |  |
| --- | --- |
| Project title | Project Report and Presentation |
| Course Code | DSE211/03 |
| Course Name | Application Development Capstone |
| Project Start date | 24-09-2024 |
| Project Submission Date | 24-11-2024 |

# Executive Summary

### **Purpose of the Project**

The primary purpose of the Library Management System (LMS) project is to modernize and streamline library operations through the development of a robust digital solution. This project aims to replace manual record-keeping processes, reduce administrative workload, and enhance the overall experience for both library staff and users. The significance of this project lies in its potential to improve accessibility to library resources, ensure efficient management of inventory, and foster a seamless interaction between users and the library.

**Summary of the Application Developed**

The Library Management System (LMS) is a web-based application designed to address key challenges faced by traditional libraries. It serves as a centralized platform to facilitate core library operations, making it accessible to users across devices with an internet connection.

**Primary Functionality**

1. **User Management:**

* Enables students and staff to register, log in, and manage their profiles.
* Differentiates roles for users (e.g., students, librarians, administrators) with appropriate access controls.

1. **Catalog Management:**

* Allows users to search, filter, and browse through an extensive digital catalog of books and resources.
* Supports inventory updates, book categorization, and real-time availability tracking.

1. **Borrowing and Returns:**

* Automates the process of issuing and returning books, with real-time updates.
* Sends reminders and notifications for due dates and overdue books.

1. **Reporting and Analytics:**

* Provides administrators with dashboards to generate reports on user activity, inventory status, and system usage metrics.

1. **Notifications and Communication:**

* Integrates SMS/email notifications for overdue alerts, announcements, and user interactions.

**Target Audience**

* **Students:** To search, borrow, and reserve library materials seamlessly.
* **Library Staff:** To manage daily library operations, including inventory tracking and transaction processing.
* **Administrators:** To monitor system performance, generate analytics, and ensure efficient resource allocation.

**Key Achievements**

1. Functional Application: Successfully developed and deployed a web-based LMS with all core features operational, including real-time book tracking and automated reminders.
2. User-Centric Design: Created an intuitive interface that ensures a smooth user experience for all stakeholders.
3. Efficiency Gains: Reduced manual work for library staff by automating processes, such as issuing and tracking returns.
4. Data Insights: Introduced reporting tools that provide actionable insights into library usage patterns and inventory needs.

**Lessons Learned**

1. Effective Planning is Crucial: Proper requirement gathering and planning phases significantly reduced the scope for rework.
2. Team Collaboration: Maintaining clear communication within the team and leveraging project management tools ensured timely delivery of tasks.
3. User Feedback is Valuable: Iterative testing and incorporating feedback from potential users helped refine the application’s functionality and usability.
4. Scalability Matters: Designing the system with scalability in mind ensures it can adapt to future requirements, such as expanding to multiple libraries or integrating with external systems.
5. **Introduction**

**Project Background**

Libraries have customarily served as significant center points for learning and asset get to. Be that as it may, the quick pace of innovative headway has highlighted the wasteful aspects of conventional, manual library administration frameworks. These frameworks regularly battle with challenges such as wrong record-keeping, restricted openness to assets, and tall authoritative workloads. For understudies and staff, the need of an proficient look and borrowing handle can lead to disappointment and misplaced efficiency.

Recognizing these challenges, the advancement of a computerized Library Administration Framework (LMS) was started to modernize library operations and improve client encounters. This venture points to address these issues by giving an robotized, user-friendly arrangement that bolsters real-time information administration and streamlines intelligent between library staff and clients.

### **Project Objectives**

1. **Enhancing Efficiency**: Automate repetitive tasks such as book borrowing, returns, and overdue notifications, reducing manual workload and errors.
2. **Improving Accessibility**: Provide a digital platform accessible via web browsers, enabling users to search and reserve books from anywhere.
3. **Streamlining Resource Management**: Facilitate real-time tracking of book inventory and ensure accurate status updates for borrowed, reserved, or available items.
4. **Empowering Administrators with Insights**: Offer reporting tools for analyzing library usage trends and optimizing resource allocation.
5. **Promoting User Engagement**: Deliver personalized notifications and updates to users, improving communication and satisfaction.

**Scope**

1. **User Account Management**: Role-based user access, including students, library staff, and administrators.
2. **Book Catalog**: An interactive, searchable catalog with filters for title, author, genre, and availability status.
3. **Transaction Automation**: Features for borrowing, returning, and reserving books, supported by real-time updates and notifications.
4. **Data and Reporting Tools**: Dashboards for administrators to monitor library activity and generate reports on usage, inventory, and user behavior.
5. **Notification System**: Automated email and SMS alerts for due dates, overdue fines, and library announcements.

**Limitations and Constraints**

1. **Limited Development Time**: The timeline constrained the scope, limiting the integration of advanced features like multimedia resource management or AI-based recommendations.
2. **Budgetary Restrictions**: Reliance on open-source tools and technologies led to compromises in design flexibility and scalability.
3. **Single Library Focus**: Testing was limited to a single library setup, leaving multi-branch operations for future iterations.
4. **System Architecture**

## ****3.1 System Architecture****

## **The Library Management System (LMS) utilizes a modern tech stack designed for scalability, efficiency, and ease of use. Below are the key technologies, frameworks, and tools used**

## ****Frontend:****

## ****HTML (Hypertext Markup Language):** The backbone of our webpage, defining the structure of our content.**

## ****CSS (Cascading Style Sheets):** Used for styling our HTML elements to make our web pages look appealing and user-friendly.**

## ****JavaScript** : Adds interactivity and dynamic behavior to the user interface, such as form validation and interactive elements.**

## ****Bootstrap** : A front-end framework that simplifies responsive web design and provides pre-styled components like modals, buttons, and navigation bars.**

## ****Backend:****

## ****PHP (Hypertext Preprocessor):** A widely-used server-side scripting language that's especially suited for web development. It allows you to create dynamic content and interact with databases.**

## ****Database:****

## ****MySQL:** An open-source relational database management system. It's known for its reliability and efficiency in handling large amounts of data.**

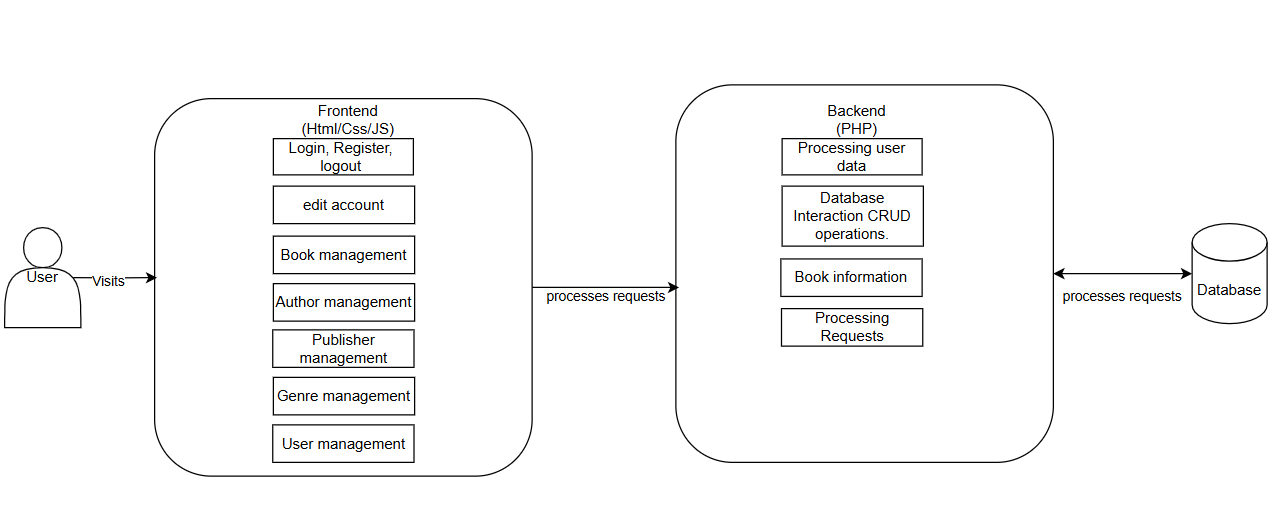
## ****phpMyAdmin:** A free tool written in PHP, intended to handle the administration of MySQL over the web. It supports a wide range of operations on MySQL and MariaDB.**

## ****Development Environment:****

## ****Visual Studio Code (VS Code):** A highly popular, free source-code editor developed by Microsoft. It includes support for debugging, embedded Git control, syntax highlighting, intelligent code completion**

### **3.2. System Architecture Diagram**

The architecture follows a **three-tier architecture**, comprising the presentation layer (frontend), application layer (backend), and database layer. Below is the diagram illustrating the system architecture



**Front end Layer**

**Technologies:** HTML, CSS, JavaScript

**Role:** User interface and interaction handling.

**Endpoints:**

**Public endpoints:** index, login, register

**User endpoints:** edit\_account, log\_out

**Book management:** add\_book, all\_book, book\_details, edit\_book, borrow

**Author management:** add\_author, all\_authors, author\_details, edit\_author

**Publisher management:** add\_Publisher, all\_publisher, publisher\_details, edit\_publisher

**Genre management:** add\_genre

**User management:** add\_users, all\_users, edit\_user, delete

**Back end Layer**

**Processing user data:**login, register, logout

Database Interaction CRUD operations (Create, Read, Update, Delete).

**Book information :** add\_book, all\_book, book\_details, edit\_book

**Processing Requests:**Receives HTTP requests from the frontend and Handles actions like user authentication (login), data submission (add\_book), and edits (edit\_user).

**3.3 Database Design**

The database is structured to manage a library system, with information about users, books, authors, publishers, genres, transactions, and permissions. Each table serves a specific purpose and is connected to other tables through primary keys and foreign keys, ensuring data integrity and smooth interactions.

1. **Table users**

* Stores details about the users of the library system.
* Fields like FirstName, LastName, Email, and Phone represent user details.
* MembershipDate records when a user joined the library.
* Permission links to the table permission table to define user roles (e.g., admin, member).

1. **Table books**

* Contains information about the books available in the library.
* Fields like Title, PublishedYear, and Quantity provide basic book details.
* AuthorID, PublisherID, and GenreID are foreign keys linking books to their respective authors, publishers, and genres.

1. **Table authors**

* Stores details about book authors.
* Includes fields like FirstName, LastName, Description, and Image for author information.

1. **Table publishers**

* Manages information about book publishers.
* Fields like PublisherName, Address, and Phone describe the publisher's details.

1. **Table genres**

* Represents different genres or categories of books (e.g., Fiction, Science, History).
* Includes fields like GenreID and GenreName.

1. **Table transactions**

* Tracks the borrowing and returning of books.
* Links to table users via UserID and table books via BookID.
* Includes fields like BorrowDate, ReturnDate, and DueDate to manage borrowing activity.

1. **Table permission**

* Defines the roles and permissions of users (e.g., Admin, Librarian, Member).
* Links to table users via the Permission field.

**Relationships Between Tables**

**Users and Transactions**

* A user can borrow multiple books, so there is a One-to-Many relationship between table users and table transactions.

**Books and Transactions**

* A book can be borrowed multiple times, so there is a One-to-Many relationship between table books and table transactions.

**Books and Authors**

* Each book has one author, but an author can write many books, creating a Many-to-One relationship between table books and table authors.

**Books and Publishers**

* Each book is linked to a publisher, but a publisher can publish many books, resulting in a Many-to-One relationship between table books and table publishers.

**Books and Genres**

* Each book belongs to one genre, but a genre can have many books, leading to a Many-to-One relationship between table books and table genres.

**Users and Permissions**

* Each user has a specific role or permission, creating a Many-to-One relationship between table users and table permission.

How It All Works Together

**User Management:**

* Users are stored in the users table.
* Their permissions or roles are determined by the permission table.

**Book Information:**

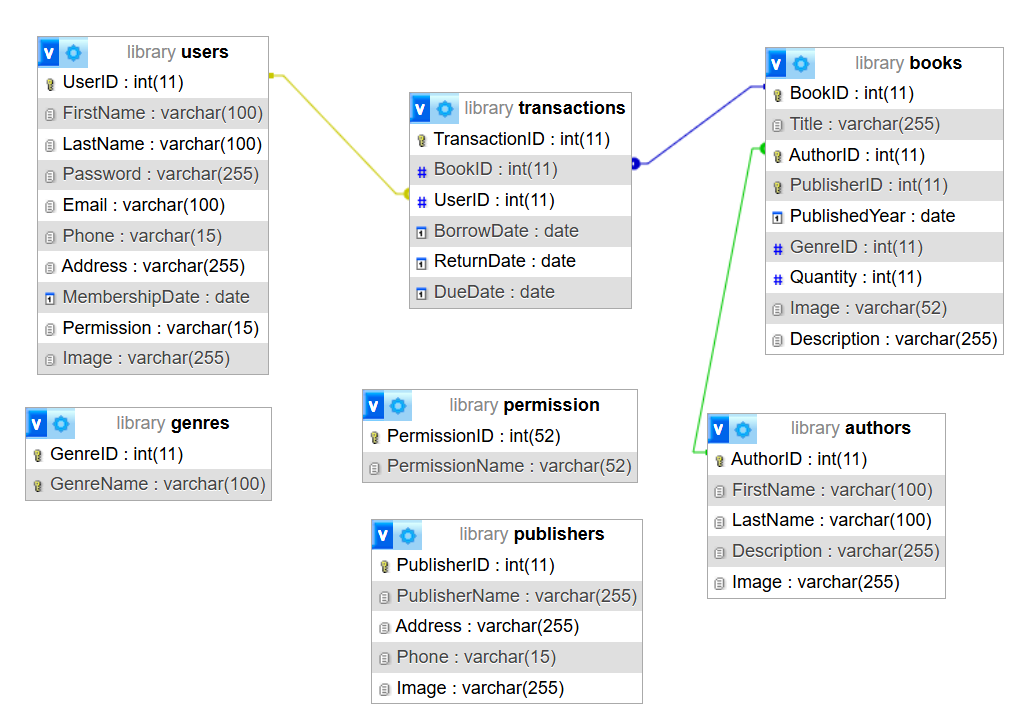
* The books table holds all details about books.
* Each book is connected to its author (table authors), publisher (table publishers), and genre (table genres).

**Borrowing System:**

* The library\_transactions table manages borrowing activities.
* It links users (via UserID) and books (via BookID) with timestamps for when the book was borrowed, returned, or due.

**Genres and Publishers:**

* Genres and publishers are separately managed in their respective tables (genres and publishers) and linked to books.



1. **Application Feature**

The Library Management System (LMS) integrates essential features aimed at addressing the operational inefficiencies in traditional library systems while meeting the diverse needs of students, staff, and administrators.

4.1.1. User Management

This module ensures secure and role-specific access to the system:

Registration and Login:

* Users can create accounts using unique credentials, verified via email confirmation.
* Login functionality supports both manual credentials entry and password recovery options.

Role-Based Access Control:

* Three roles are defined: Students, Library Staff, and Administrators.
* Students can search and borrow books, while staff manages transactions, and administrators handle system analytics and configuration.

Profile Management:

* Users can view and update their personal details, track borrowing history, and check reservations.

4.1.2. Book Catalog Management

The catalog offers an intuitive and real-time view of library resources:

Search Functionality:

* Users can search for books by title, author, genre, ISBN, or keywords.
* Advanced filters allow sorting by category, availability, and publication year.

Book Details:

* Each book listing provides detailed information, including the title, author, genre, summary, ISBN, and availability status.

Dynamic Availability Updates:

* Real-time updates ensure users can see whether a book is currently available, borrowed, or reserved.

4.1.3. Borrowing and Return Transactions

Streamlined processes make borrowing and returning books more efficient:

Automated Borrowing:

* Users can select books to borrow, and the system generates a borrowing record with a due date.
* Digital confirmation is sent via email or SMS.

Return Management:

* When books are returned, staff scan the book's barcode, and the system updates the inventory automatically.

Reservation System:

* Users can place reservations on currently unavailable books, with priority-based queuing.

4.1.4. Notifications System

The system enhances communication with users by automating notifications:

Overdue Reminders:

* Automatic alerts are sent for overdue books, including details of penalties, if applicable.

Reservation Alerts:

* Users receive notifications when reserved books are available for pickup.

Library Updates:

* Administrators can send announcements about new book arrivals, events, or holiday schedules.

4.1.5. Reporting and Analytics

Provides actionable insights into library usage:

Usage Reports:

* Tracks key metrics, such as the most borrowed books, peak borrowing periods, and user demographics.

Inventory Management:

* Allows administrators to monitor stock levels and identify high-demand books.

Fine Collection Tracking:

* Displays records of overdue fines collected and pending payments.

**4.2 User Interface**

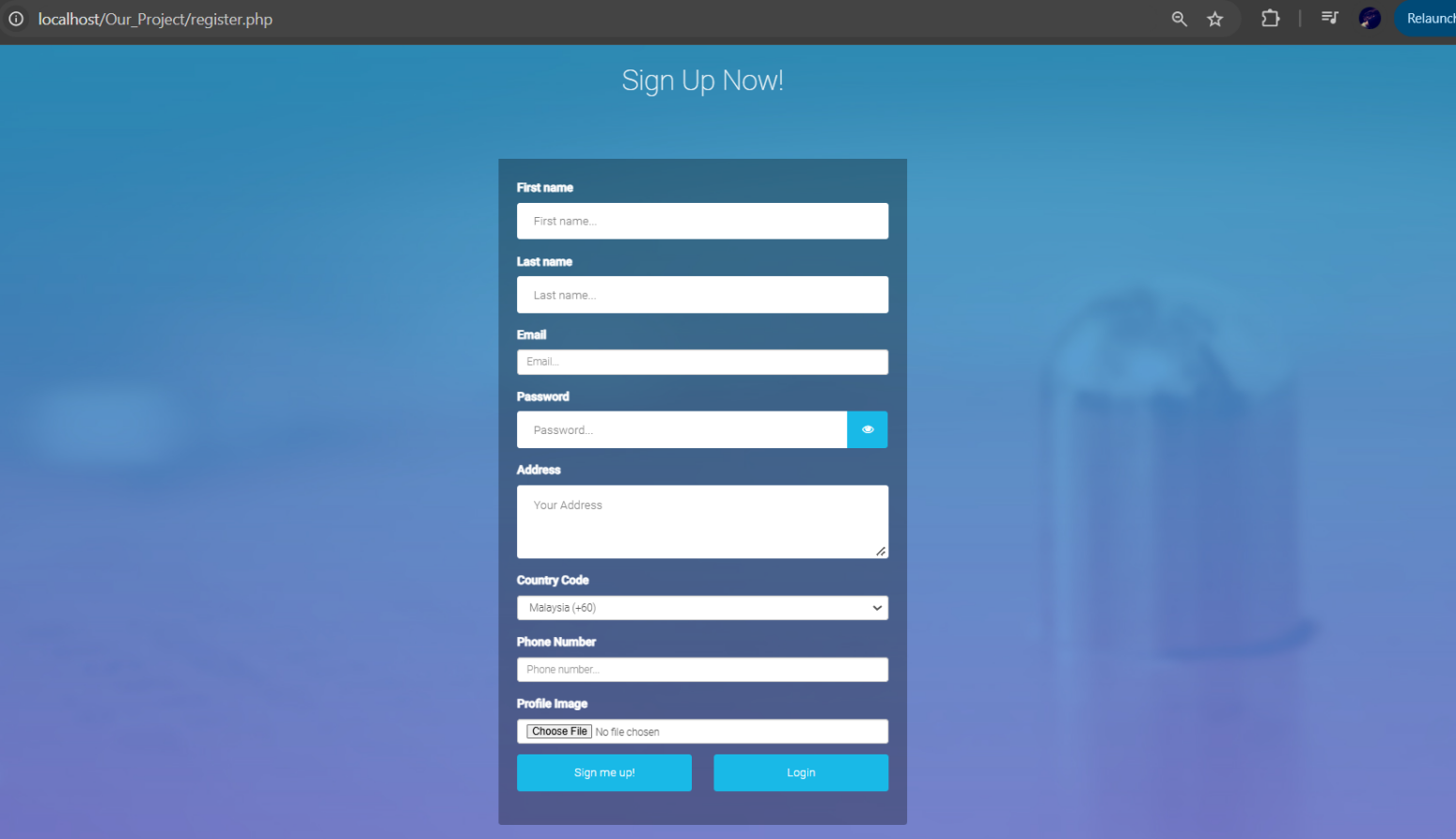
Screenshots or Wireframes

4.2.1 User Management

**Pages/Sections to Include:**

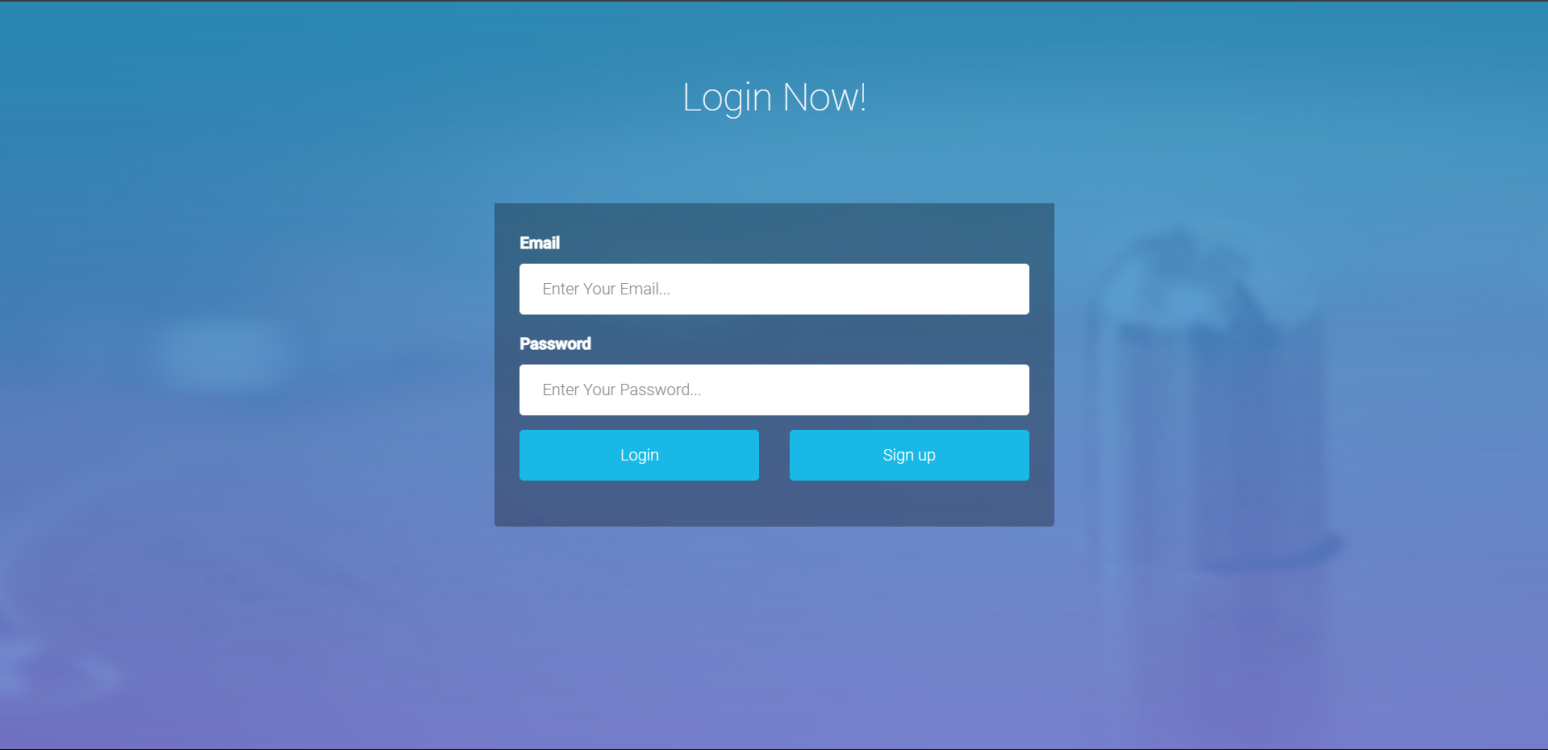
Register Page: A form where new users input their details (e.g., name, email, password).

* Fields: Name, Email, Password, Confirm Password.
* Action: Submit button.



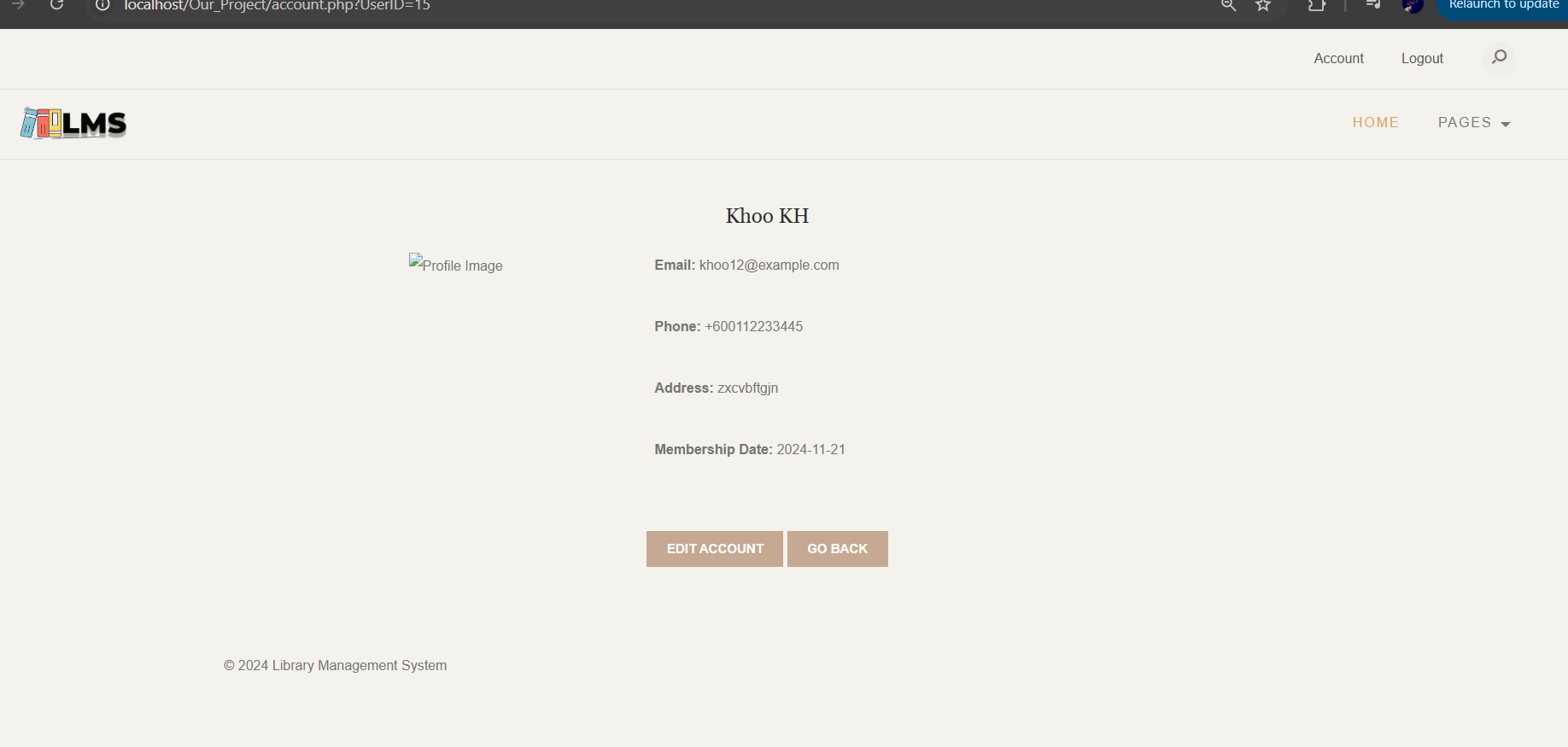
Login Page: A simple form with fields for email and password.

* Fields: Email, Password.
* Actions: Login button.



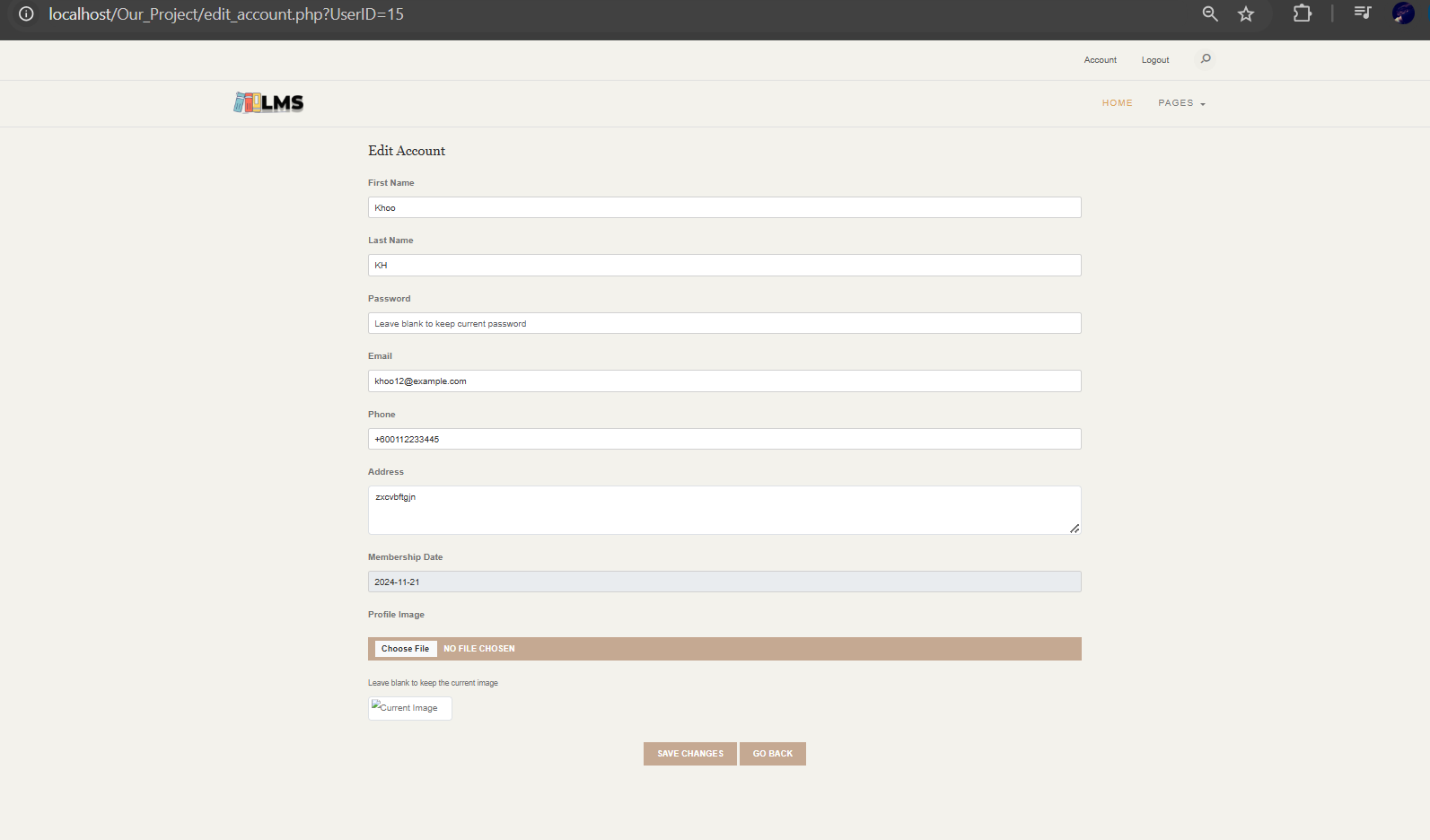
User Dashboard: Displays user information and options.

* Components: User profile card, account settings.



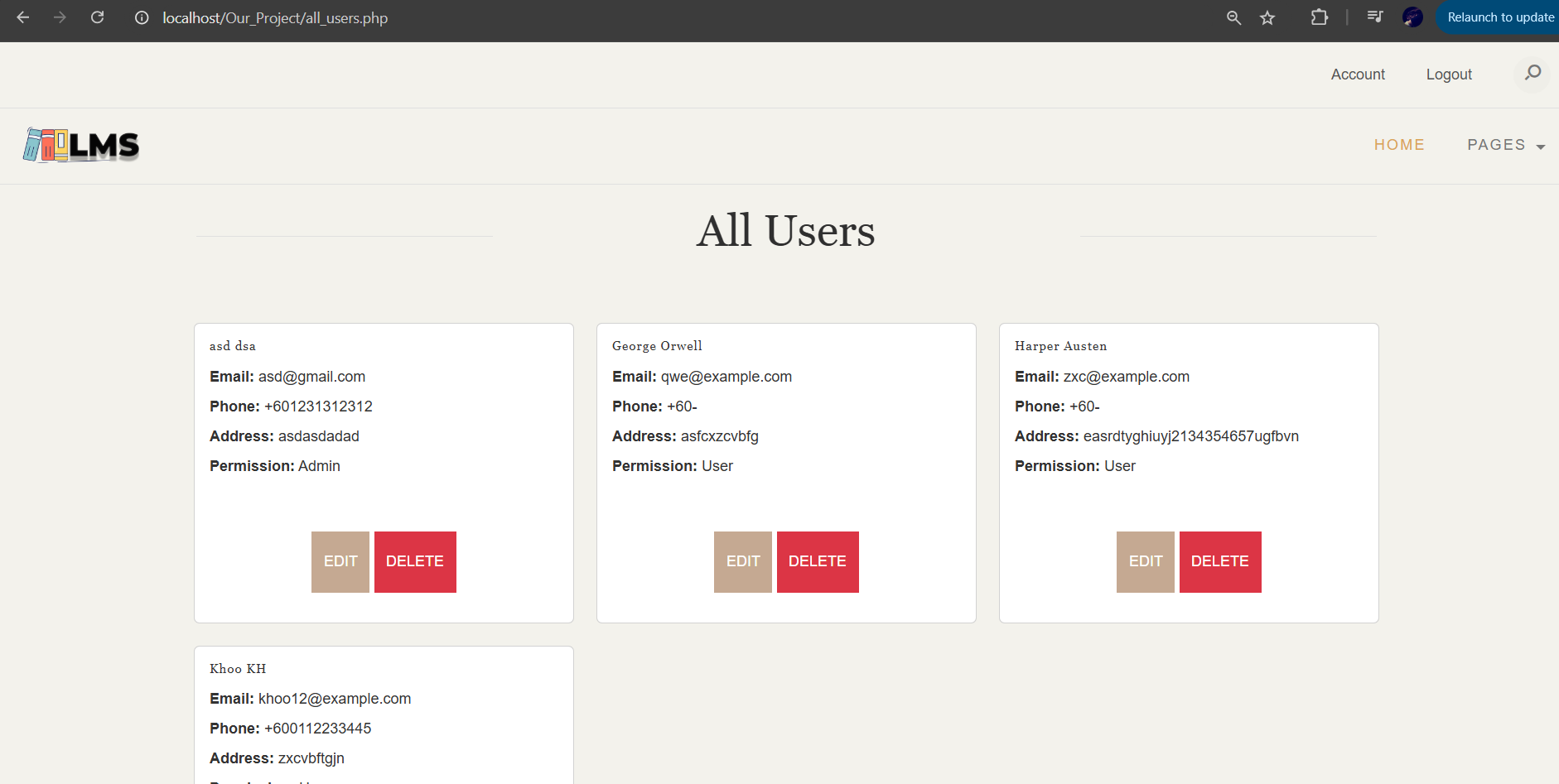
Edit Account Page: Form for updating user details.

* Fields: Name, Email, Password (optional), Profile Picture (optional).



Admin View (All Users): A table showing all registered users.

* Columns: User ID, Name, Email, Role, Actions (Edit, Delete).

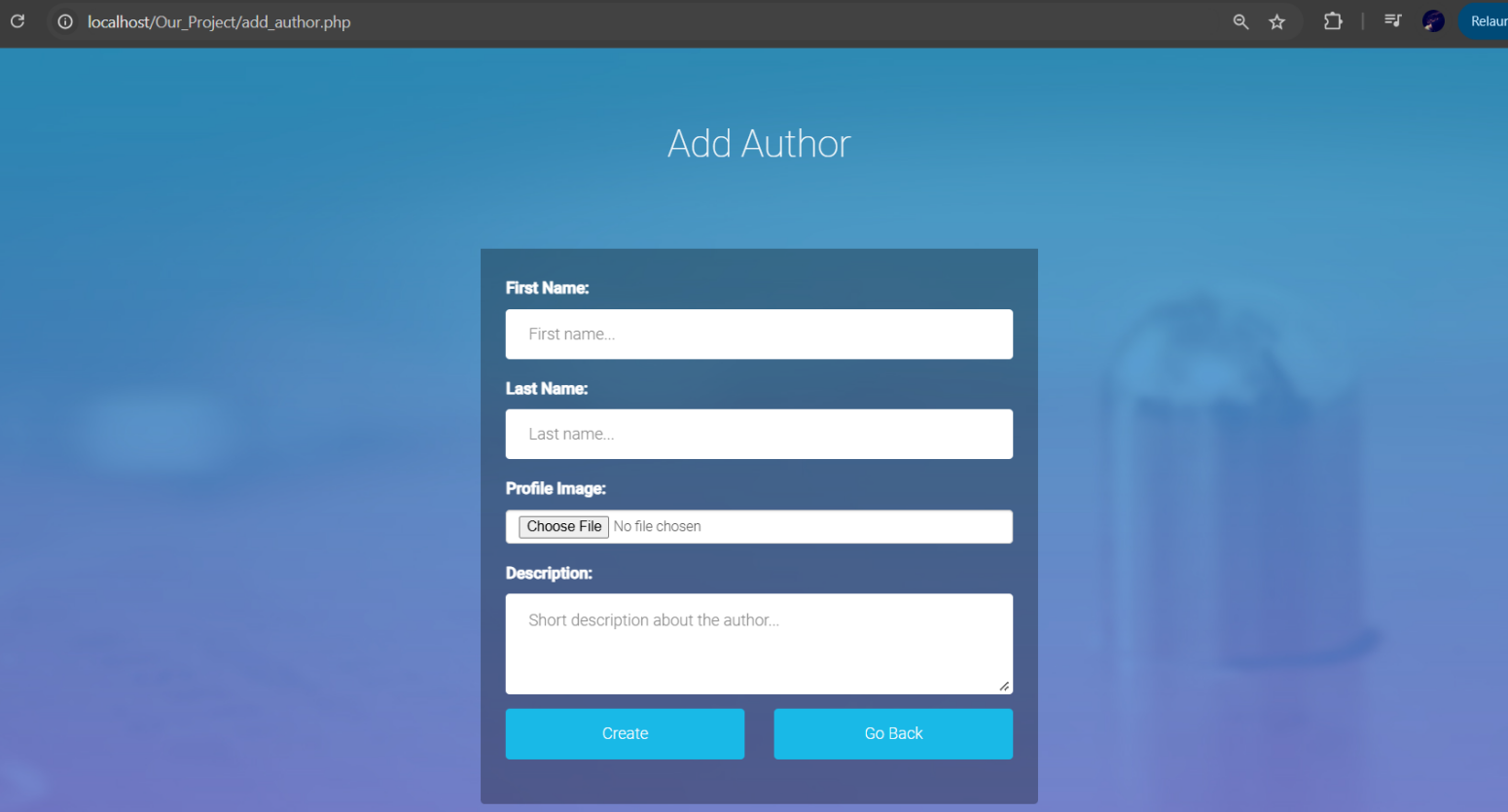


**4.2.2 Author Management**

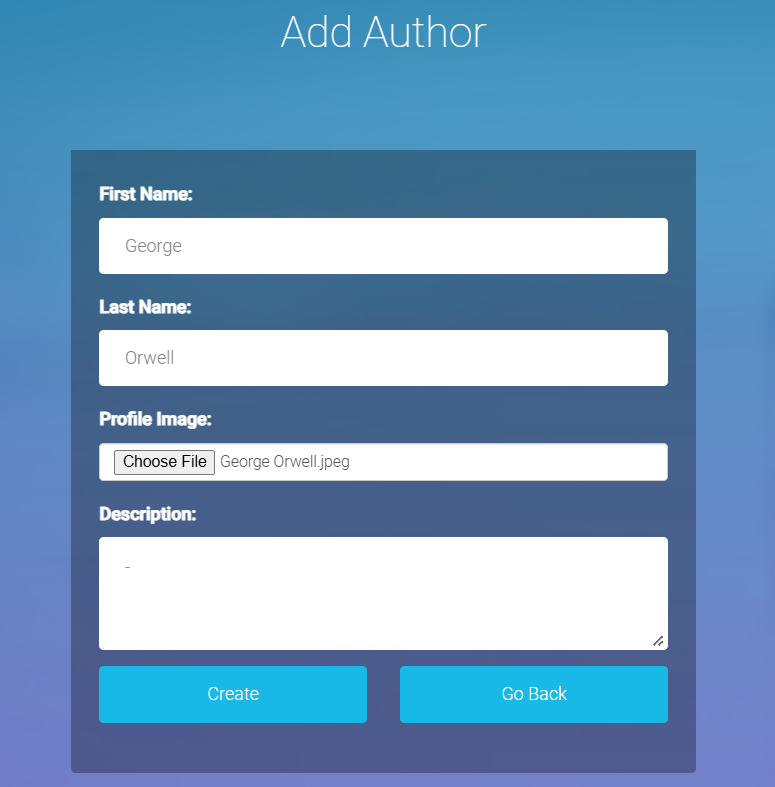
Pages/Sections to Include:

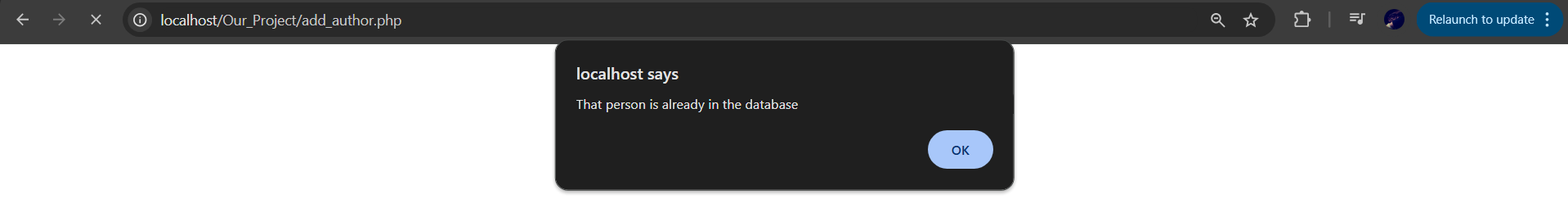
Add Author Page: A form to add a new author.

* Fields: Author Name, Biography, Nationality.



Edit Author Page: Similar to the "Add Author" page but pre-filled with existing details.

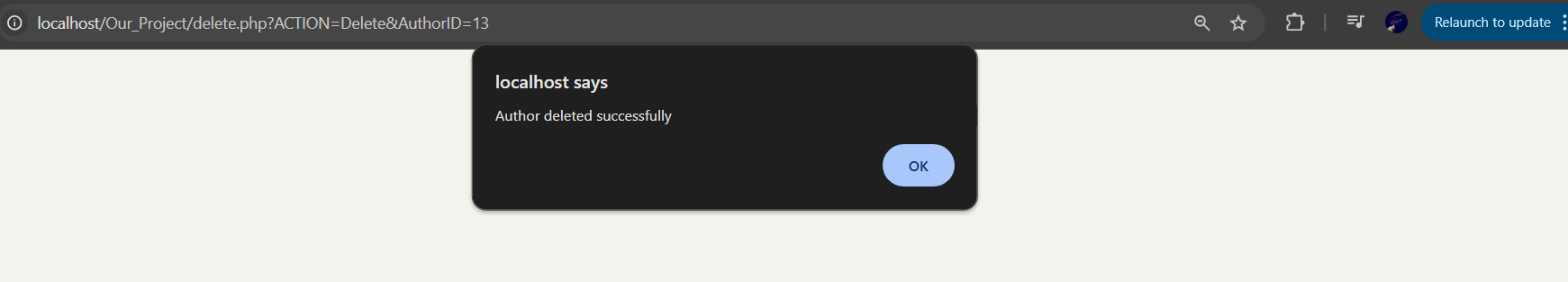




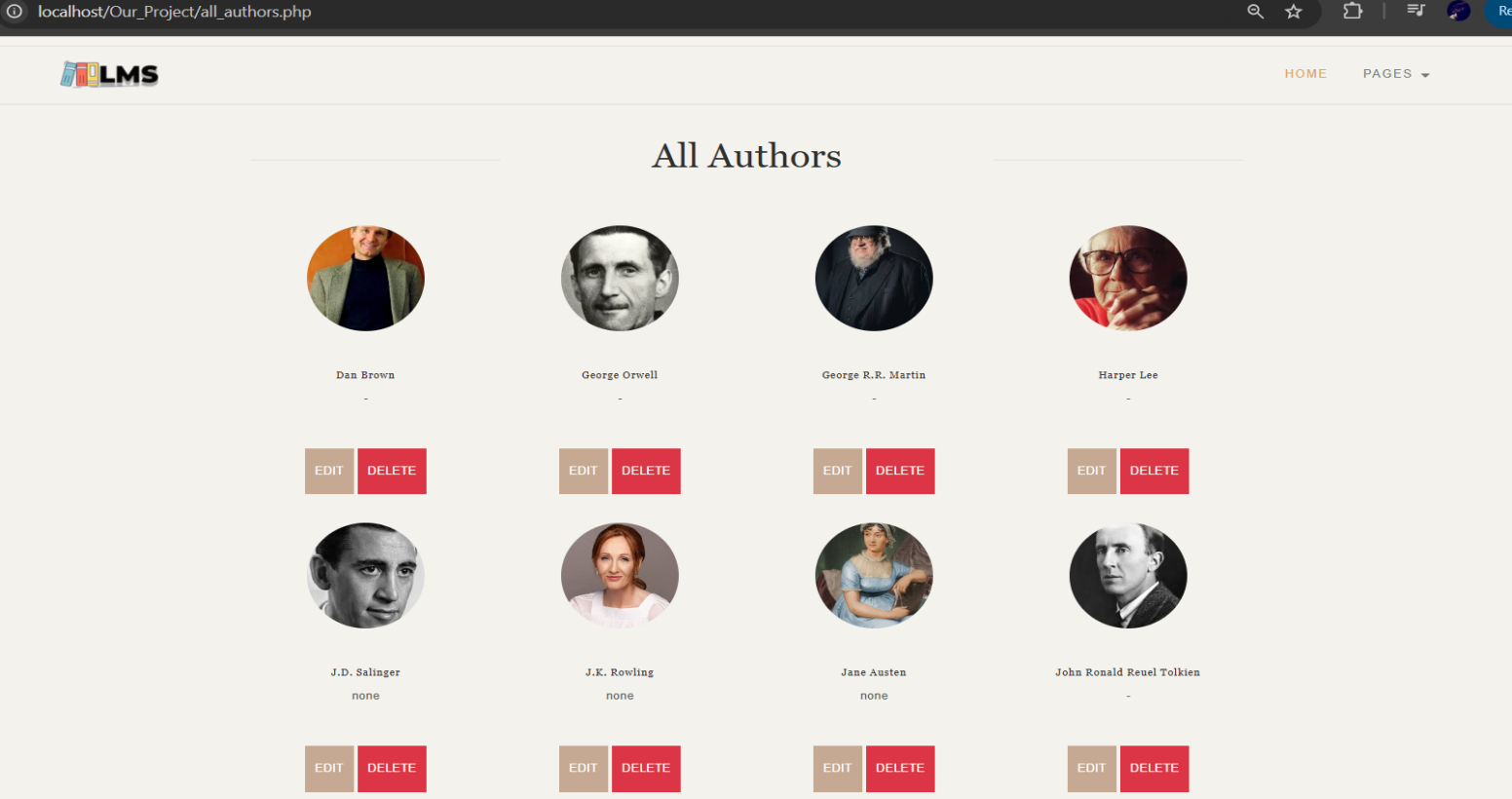
Author List Page: A table displaying all authors.

* Columns: Author ID, Name, Nationality, Number of Books, Actions (View, Edit, Delete).

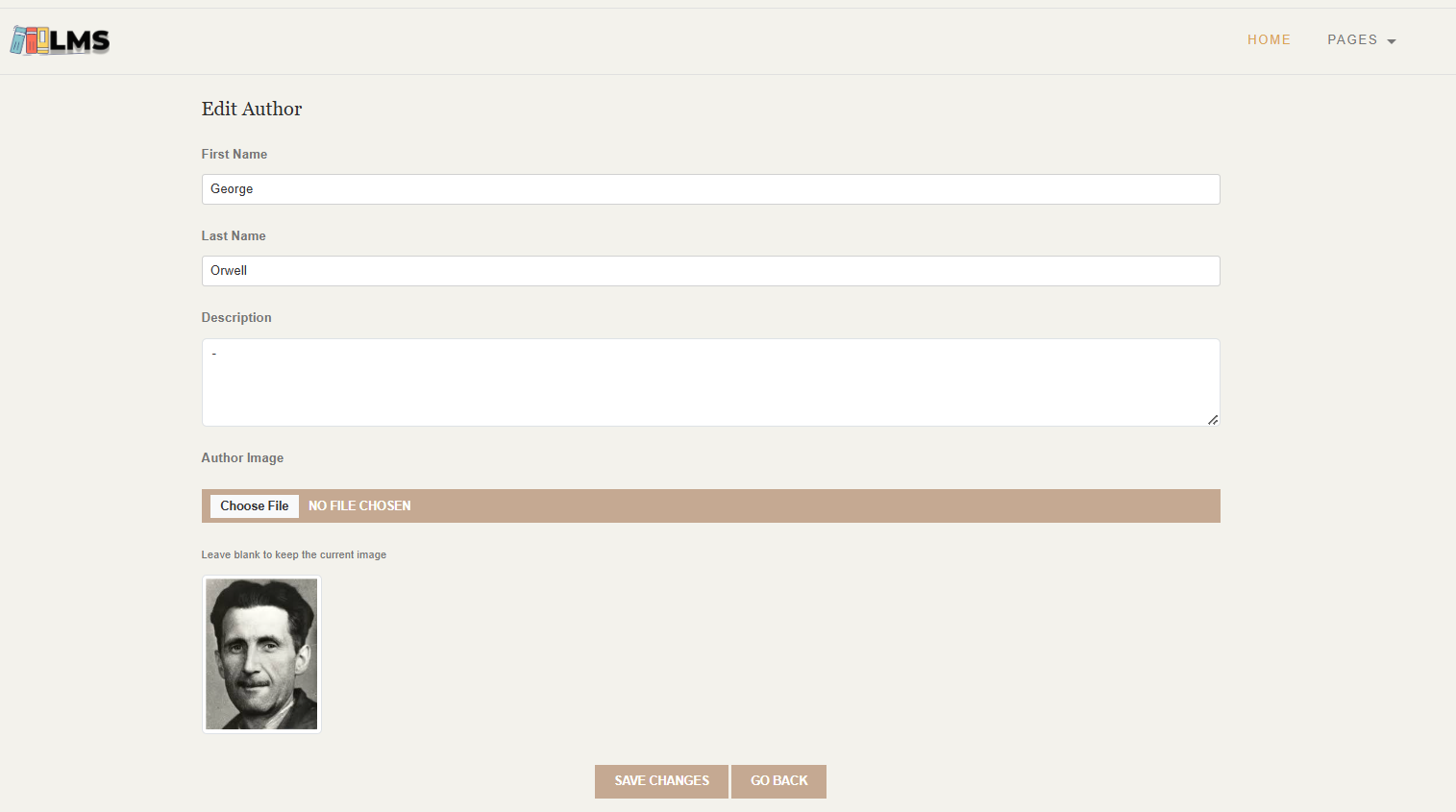
Delete



Author list



Edit

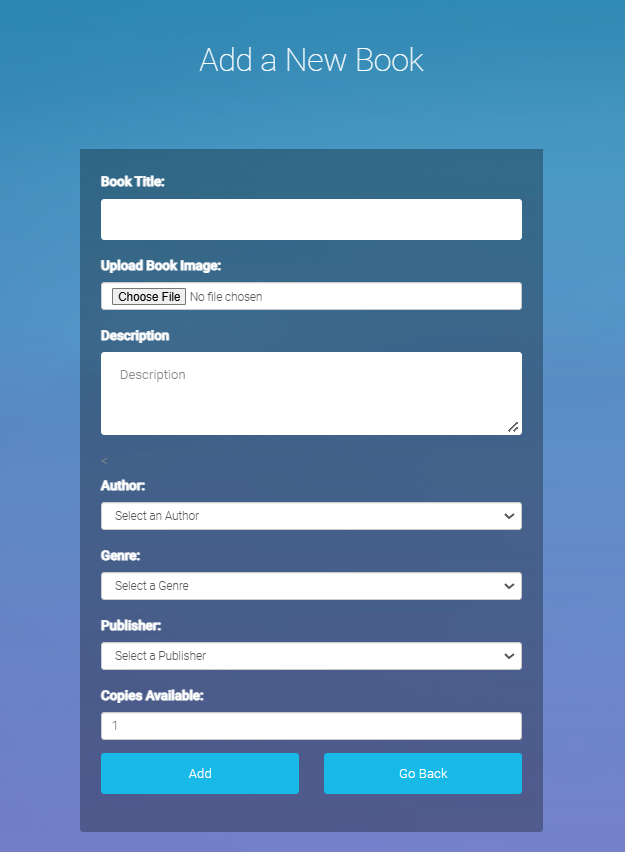


**4.2.3 Book Management**

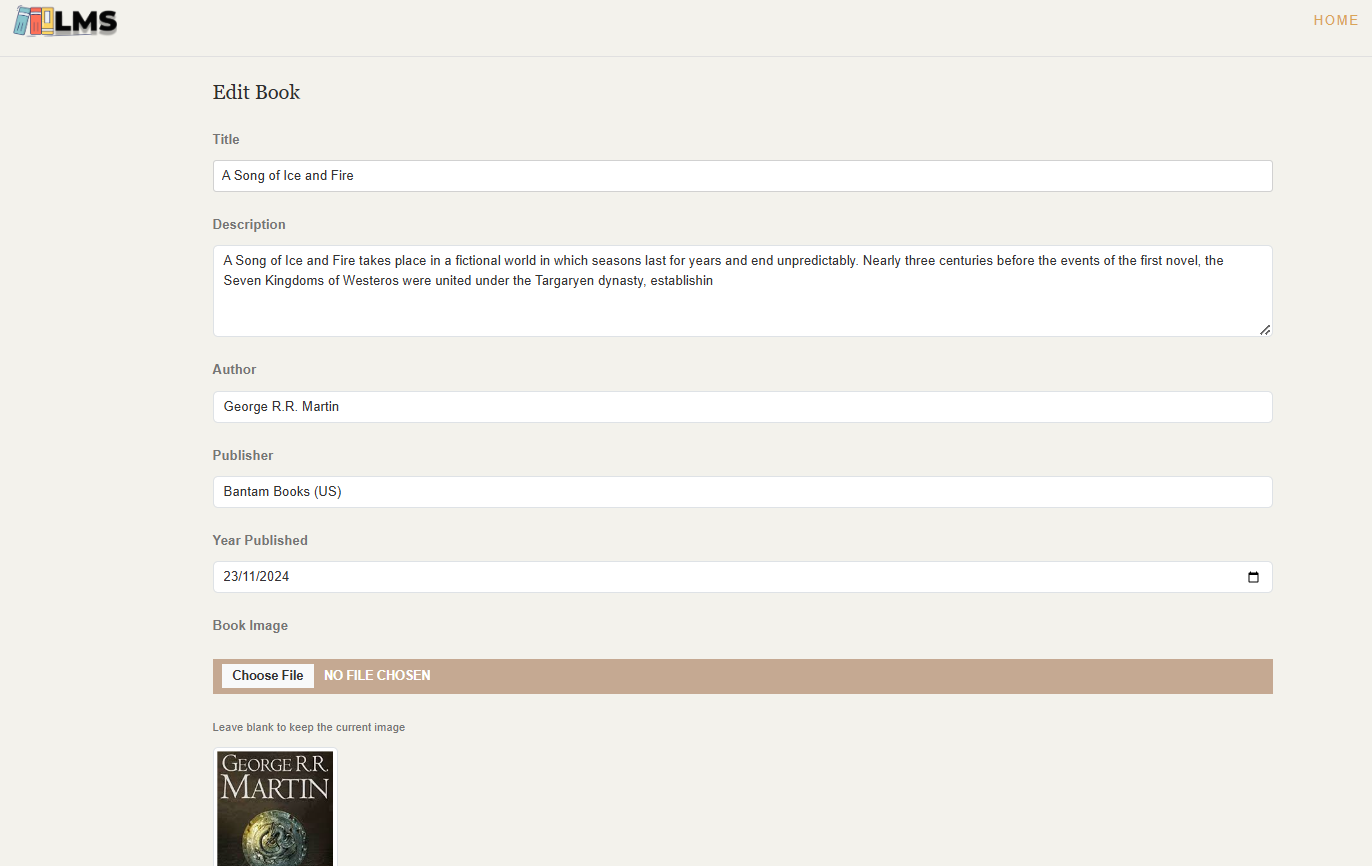
**Pages/Sections to Include:**

**Add Book Page:** A form to add a new book.

* Fields: Title, Author , Genre , Publisher , ISBN, Year Published, Copies Available.

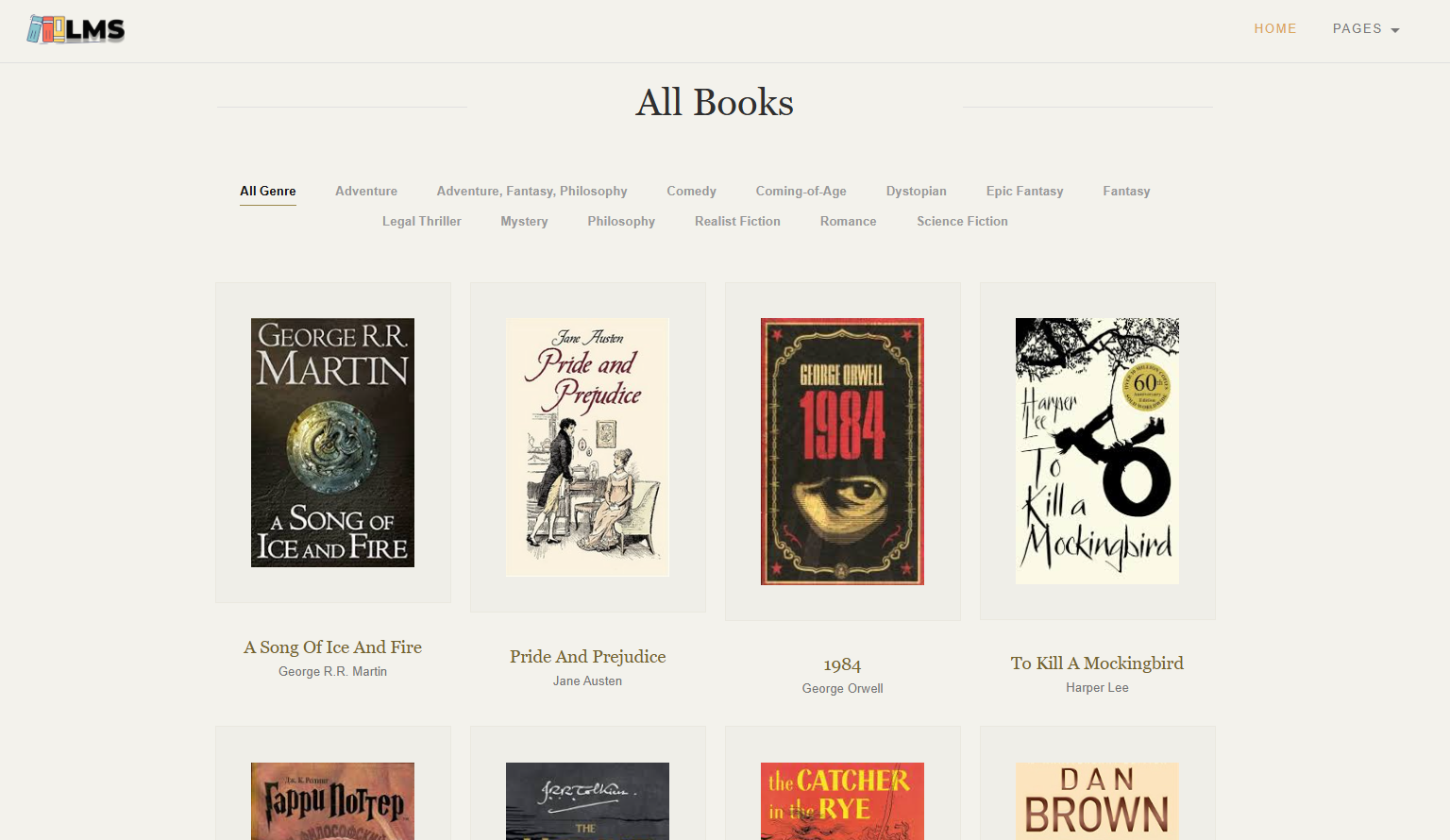


Edit Book Page: Pre-filled form similar to the "Add Book" page.



Book List Page: A searchable table of books.

* Columns: Book ID, Title, Author, Genre, Copies Available, Actions (View, Edit, Delete).

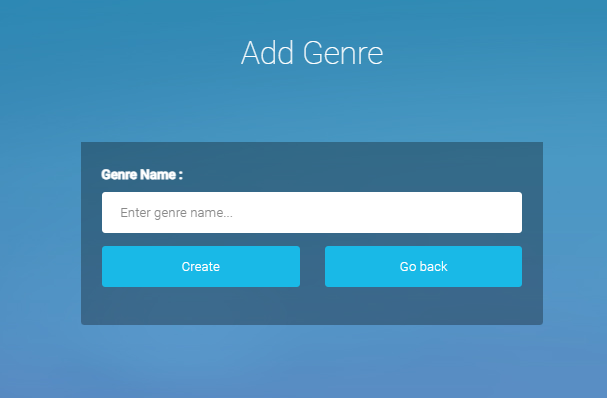


4.2.4 Genre Management

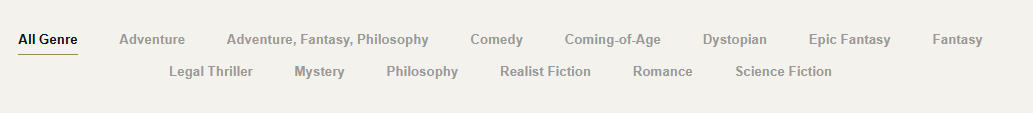
Pages/Sections to Include:

Add Genre Page: A form to add new genres.

* Fields: Genre Name.



Genre List Page: Displays all available genres.

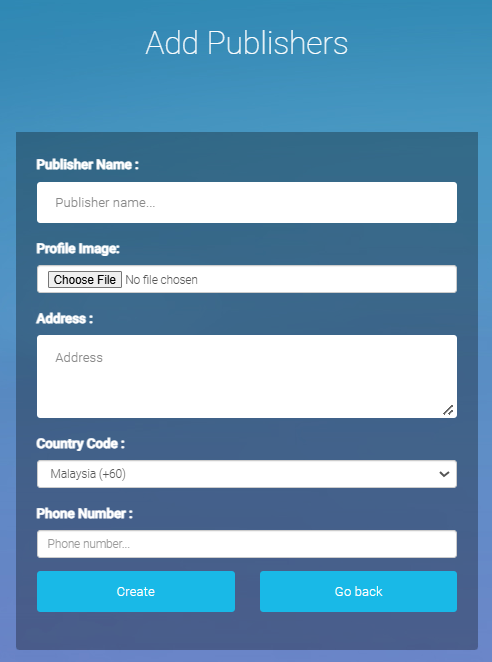


4.2.5 Publisher Management

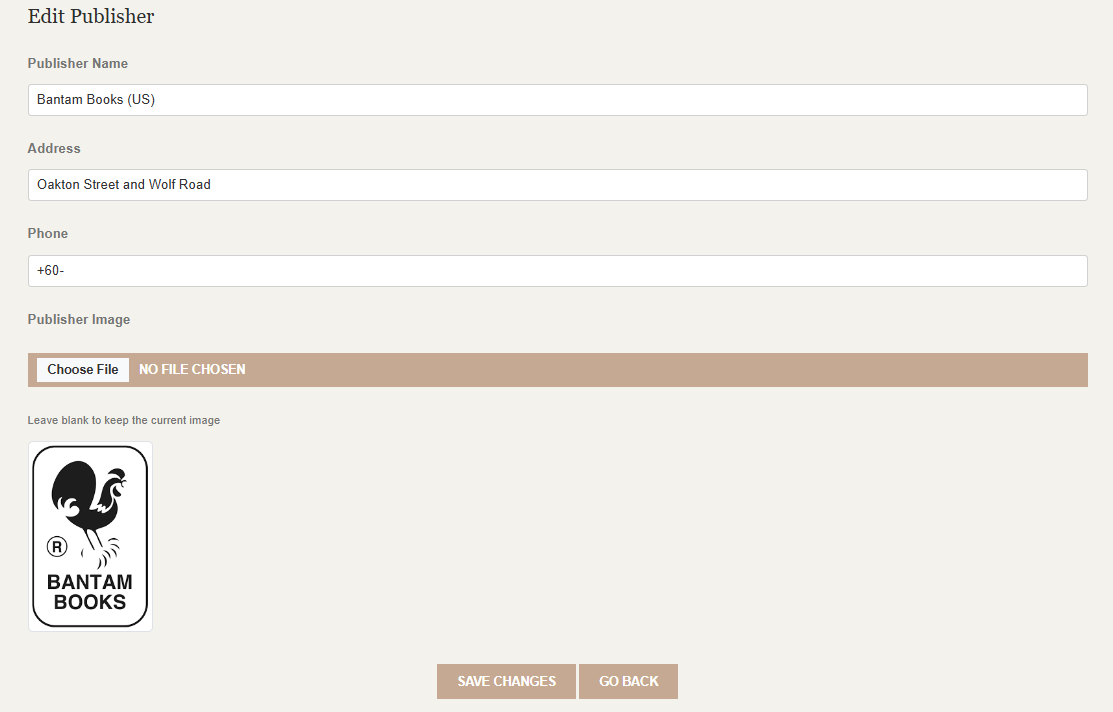
Pages/Sections to Include:

Add Publisher Page: A form to add a new publisher.

* Fields: Publisher Name, Address, Contact Information.

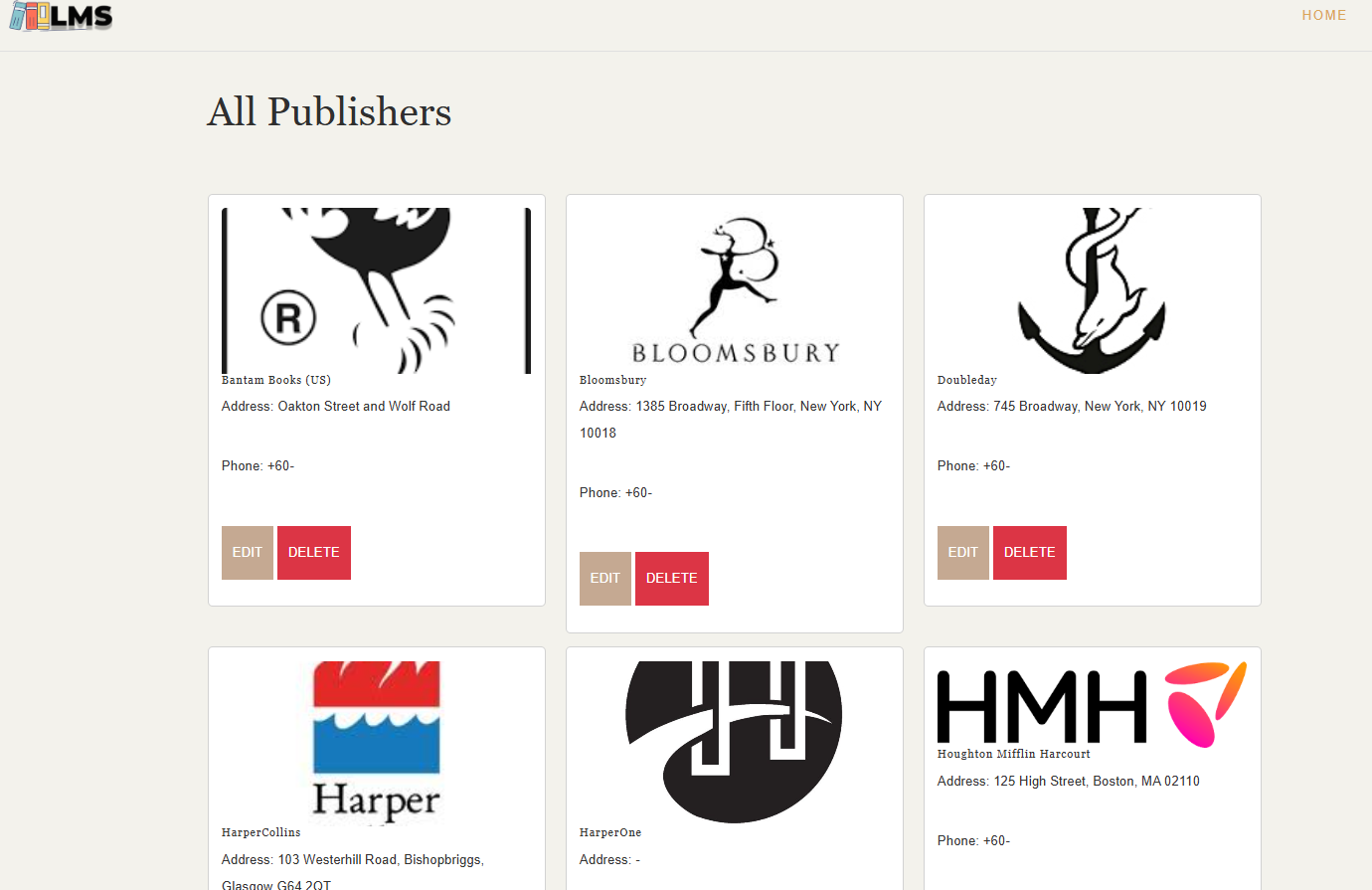


Edit Publisher Page: Pre-filled form similar to "Add Publisher."



Publisher List Page: A table displaying all publishers.

* Columns: Publisher ID, Name, Contact, Actions (View, Edit, Delete).



1. **Testing and Quality Assurance**

**Purpose**

The purpose of this section is to outline the testing processes and strategies applied to ensure that the application functions as expected, is user-friendly, and is free of critical bugs. The testing process covers functional, performance, and security aspects, ensuring a robust and reliable system.

Testing Strategies

Unit Testing

Objective: Test individual PHP scripts or functions to ensure they work correctly.

Tool: Use simple PHP test scripts (or a lightweight framework like PHPUnit).

Example: Validate a PHP function that processes form submissions (e.g., user registration).

Integration Testing

Objective: Test the interaction between PHP back-end and HTML/CSS front-end.

Method: Manually navigate through forms/pages and check if data flows as expected.

Example: Add a new book and ensure it appears in the book list on the front-end.

System Testing

Objective: Ensure that the entire application works end-to-end.

Approach: Create scenarios (e.g., user registration) and test them manually.

Tool: Browser developer tools to monitor performance and behavior.

### **Test Cases**

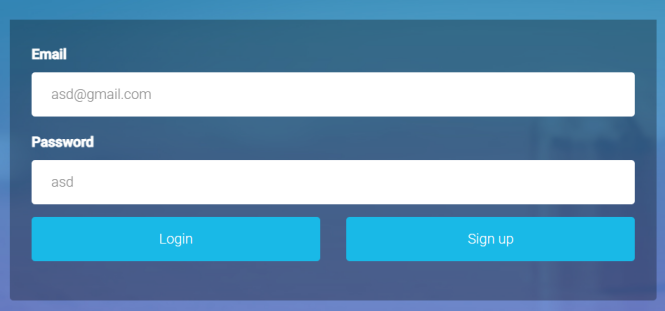
|  |  |  |  |
| --- | --- | --- | --- |
| **Test Scenario** | **Expected Result** | **Actual Result** | **Status** |
| User Registration | New user is successfully added to the database with hashed password. | Passed | Success |
| Login with Invalid Credentials | User receives an error message for incorrect email or password. | Passed | Success |
| Search for Books | Displays books matching the entered title, author, or genre. | Passed | Success |
| Borrow a Book | Book availability decreases by 1 and transaction is recorded with the due date. | Passed | Success |
| Return a Book | Transaction is updated as "returned," and book quantity increases. | Passed | Success |
| Edit Book Details | Updates the book information (e.g., title, author, genre) and reflects in searches. | Passed | Success |

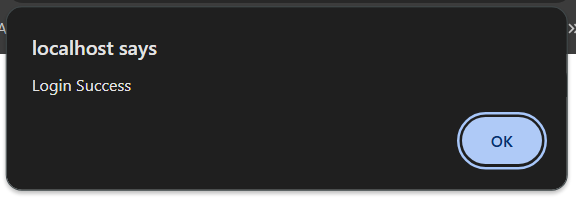
#### **Example Test Case 1: User Registration**

* **Test Scenario**: Verify user login functionality.
* **Steps**:

1. **Open the login page.**
2. **Enter valid credentials (e.g., Email and Password).**
3. **Click the "Login" button.**

* **Expected Result**: User is redirected to the login page with a success message.
* **Actual Result**: Pass. Works as expected.





#### **Example Test Case 2: Adding a Book**

* **Test Scenario**: Admin adds a book.
* **Steps**:
  1. Log in as an admin.
  2. Navigate to the "Add Book" page.
  3. Fill in book details and submit.
* **Expected Result**: Book appears in the catalog.
* **Actual Result**: Fail. Incorrect validation for ISBN field.
* **Resolution**: Updated PHP validation logic.

### **Performance Testing**

#### **Approach**

* Conducted load testing to simulate multiple users accessing the system simultaneously.
* Tools Used: Apache JMeter, Locust.

#### **Results**

**Login Page**:

* **Simulated Load**: 100 users in 10 seconds.
* **Response Time**: Average of 1.2 seconds.
* **Optimization**: Reduced server-side queries by adding caching for session data.

**Book Addition Functionality**:

* **Simulated Load**: 50 simultaneous submissions of the "Add Book" form within 1 minute.
* **Response Time**: Average response time of 1.8 seconds.
* **Optimization**: Optimized database insert queries and reduced validation overhead in the PHP script.

**Security Testing**

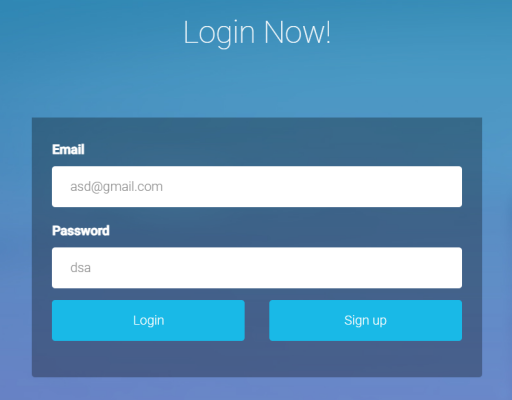
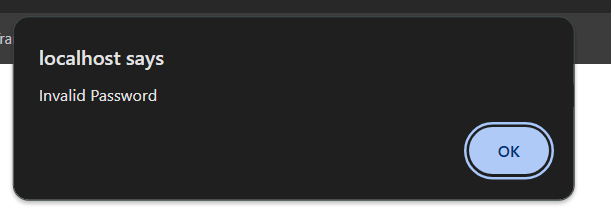
Test Scenario: Validate user login and access control.

Steps: Attempt login with incorrect credentials (wrong password or email).

Expected Result:

Users can log in only with valid credentials.

Admin features are accessible only by users with admin roles.

1. **Deployment and Optimization**

**Purpose**

This section outlines the process of deploying the application to a live environment and the optimization efforts implemented to enhance performance, scalability, and security.

**Hosting Environment**

For the deployment process, the hosting environment includes using a local web server like XAMPP or WAMP for testing, and a cPanel-based shared hosting for live deployment. The server specifications required are PHP 7.4, MySQL 5.7, Apache Web Server, and an SSL certificate for HTTPS.

Deployment Steps

1. Development Preparation:

* Ensure all files are organized and properly structured.
* Test the application locally to confirm functionality.

1. Database Setup:

* Export the local database using phpMyAdmin.
* Import the database to the live server using the hosting provider’s phpMyAdmin interface.

1. File Transfer:

* Use GitHub or the hosting provider’s file manager to upload all project files to the /public\_html directory.
* Update database connection settings in the PHP configuration file to reflect the live server’s database credentials.

1. Domain Configuration:

* Link the hosting server to a custom domain name.
* Ensure DNS settings are correctly configured for domain propagation.

1. SSL Configuration:

* Enable SSL through the hosting provider’s control panel to ensure secure communication.

1. Testing and Verification:

* Test all critical functionalities (e.g., login, registration, book management) in the live environment.
* Verify page load times, responsiveness, and cross-browser compatibility.

Challenges Encountered

* Database Connection Errors: Incorrect database credentials initially caused connection issues, resolved by verifying live server settings.
* File Permission Issues: Some PHP scripts required updated file permissions (e.g., 755 for directories, 644 for files) to function correctly.

### **Optimization Techniques**

#### **Performance Optimizations**

**Code Optimization**:

* Minimized redundant PHP code and reused functions to improve maintainability.
* Optimized SQL queries by adding indexes to frequently searched fields.

**Caching**:

* Implemented server-side caching for session data and database queries to reduce load times.

**Asset Optimization**:

* Minimized CSS and JavaScript files using online tools (e.g., CSS Minifier, JS Compress).
* Enabled Gzip compression on the server for faster file transfers.

#### **Scalability Optimizations**

**Database Optimization**:

* Added indexing to large database tables (e.g., books, users).
* Regularly performed database cleanup to remove unused or outdated entries.

**Load Balancing** (Future Ready):

* Prepared for potential scaling by ensuring the application can be moved to a hosting service with load balancing capabilities.

### **Continuous Integration/Continuous Deployment (CI/CD)**

Since the application uses PHP without APIs, a basic CI/CD pipeline was implemented manually:

#### **Manual Pipeline**

1. **Version Control**:

* Git was used to maintain versions of the application code.

1. **Testing**:

* Each change was tested locally on XAMPP before being deployed to the live server.

1. **Deployment**:

* File changes were uploaded via GitHub or directly through cPanel’s File Manager.

#### **Future Enhancements**

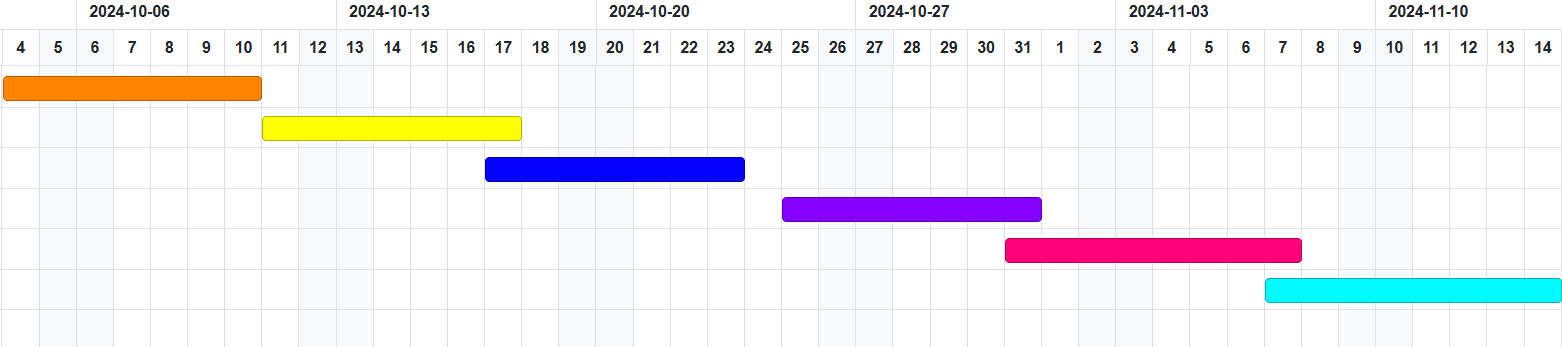
* Setting up automated testing and deployment using tools like Jenkins or GitHub Actions for more consistent updates.
* Integrating unit tests into the CI/CD pipeline to detect potential bugs early.

1. **Project Management and Collaboration Documentation**

**Purpose**

To detail how the project was managed, the tools and techniques used for task organization and communication, and the roles and responsibilities of each team member to ensure successful completion.

|  |  |  |  |
| --- | --- | --- | --- |
| **Phase** | **Duration** | **Key Activities** | **Deliverables** |
| **Planning** | Week 1 | Requirements gathering, project scoping, and system architecture design | Project proposal, System Architecture Diagram |
| **Design** | Weeks 2 | Database design, wireframe creation, and UI/UX design | Wireframes, Database schema |
| **Development** | Weeks 3 | Frontend and backend implementation, integration of features | Functional application, Core features implemented |
| **Testing** | Weeks 4 | Unit, integration, and system testing | Bug-free application ready for deployment |
| **Deployment** | Week 5 | Hosting setup, application deployment to a live environment | Deployed application |
| **Documentation & UAT** | Week 6-7 | Final user acceptance testing, project documentation | Final report, User manual |



#### **Tools and Techniques**

**Project Management Tools**

**Free Online Gantt Chart Software**

* 1. Used for task scheduling, tracking progress, and visualizing the timeline.
  2. Helped ensure all team members were aware of deadlines and deliverables.

**GitHub**

* 1. Centralized code repository for version control and collaboration.
  2. Branching strategy ensured separate environments for development, staging, and production.
  3. Used GitHub Issues and Project Boards to track bugs and features.

**Discord**

* 1. Main communication platform for instant messaging, group calls, and file sharing.
  2. Weekly meetings were held to review progress and address roadblocks.
  3. Channels were created for specific topics (e.g., development, testing, documentation).

**Visual Studio Code**

* 1. Primary code editor for the team, with extensions for PHP, HTML, and collaborative coding.
  2. Integrated with Git for seamless version control.

**Team Roles and Responsibilities**

|  |  |  |
| --- | --- | --- |
| **Role** | **Team Member(s)** | **Responsibilities** |
| **Project Manager** | Wong Park Kit | - Oversaw project timelines, milestones, and resource allocation. |
|  |  | - Facilitated team meetings and maintained project management tools. |
| **Backend Developer** | Wong Park Kit, Tan Ee Fun | - Developed server-side logic using PHP. |
|  |  | - Designed and implemented database schema and queries. |
|  |  | - Integrated backend with frontend features. |
| **Frontend Developer** | Wong Park Kit, Tan Ee Fun, Khoo Kah Heng | - Developed the user interface using HTML, CSS, and JavaScript. |
|  |  | - Ensured responsiveness and usability of the UI across devices. |
| **Tester** | Wong Park Kit, Tan Ee Fun, Khoo Kah Heng | - Conducted unit, integration, and system testing. |
|  |  | - Logged and tracked bugs and verified fixes. |
| **Documentation Lead** | Khoo Kah Heng | - Prepared all project documentation, including user manuals and final reports. |
|  |  | - Maintained records of user stories, use cases, and design choices. |
| **DevOps Engineer** | Wong Park Kit, Tan Ee Fun, Khoo Kah Heng | - Set up the hosting environment and deployment pipelines. |
|  |  | - Implemented optimization techniques and managed application security. |

1. **Challenges and Lessons Learned**

### **Challenges**

#### **8.1.1. Technical Difficulties**

* **Integration Issues**:  
  Merging the frontend and backend components initially caused discrepancies due to mismatched API response formats and inconsistent endpoint naming.
* **Database Optimization**:  
  Inefficient queries led to slower response times, especially for search and filtering functionalities with large datasets.
* **Deployment Challenges**:  
  Deploying the application to the live environment encountered firewall and configuration issues, leading to downtime during the deployment process.

#### **8.1.2. Time Management Issues**

* **Unrealistic Estimates**:  
  Some tasks, such as database schema design and UI testing, took longer than anticipated, affecting the overall timeline.
* **Conflicting Schedules**:  
  Coordinating among team members with varying academic and personal schedules posed difficulties in setting consistent work sessions.

### **Resolutions**

#### **8.2.1. Time Management Solutions**

**Task Re-prioritization**:  
Focused on delivering core features first and deferred non-essential features to post-deployment phases.

**Improved Scheduling**:  
Used shared calendars and Discord reminders to schedule collaborative work sessions, maximizing overlap in team availability.

#### **8.2.2. Handling Unexpected Obstacles**

**Feature Management**:  
Implemented a clear change request process, assessing the feasibility and impact of new features before acceptance.

**Learning and Adaptation**:  
Team members allocated time to research and implement best practices for optimization, improving the application’s performance.

### **Lessons Learned**

#### **8.3.1. Collaboration and Communication**

**Shared Knowledge Base**:  
Maintaining centralized documentation, such as API references and design guidelines, significantly improved collaboration and efficiency.

**Frequent Check-ins**:  
Regular daily stand-ups kept everyone aligned and allowed quick identification of blockers.

#### **8.3.2. Technical Insights**

**Performance Optimization**:  
Early focus on database design and efficient query writing reduces downstream performance issues.

**Testing**:  
Comprehensive and continuous testing, including unit tests and integration tests, is essential for maintaining application stability and performance.

### **Summary of Outcomes**

#### **Final Application Delivered**

* The project successfully delivered a fully functional library management system built using HTML, CSS, and PHP.
* Key features include:
  + User authentication (login and registration).
  + Book management (add, edit, delete books).
  + Borrowing and returning functionality.
  + Admin and user roles with distinct privileges.

#### **Impact on Users**

* The application significantly simplifies library operations by automating routine tasks such as tracking borrowed books and managing inventory.
* Users benefit from an intuitive interface and secure login system, making the library experience seamless.
* Administrators can efficiently monitor library activities and ensure data consistency.

### **Future Enhancements**

While the application meets its current objectives, there are several potential areas for improvement and expansion:

#### **New Features**

* **Mobile Application**:  
  Develop a dedicated mobile app for iOS and Android to reach a broader audience and improve accessibility.
* **Advanced Analytics Dashboard**:  
  Introduce detailed insights for librarians, such as reading trends, most borrowed books, and user activity heatmaps.
* **Digital Library**:  
  Enable e-book borrowing and integration with digital content providers for users to read books online.
* **Multilingual Support**:  
  Add support for multiple languages to cater to a diverse user base.

### **Final Thoughts**

The project has been a rewarding journey, resulting in a practical and user-friendly application that addresses the primary needs of a library system. Through effective collaboration, the team overcame technical and time-related challenges to deliver a robust solution.

The lessons learned and the foundation laid in this project will serve as a stepping stone for future developments, enabling us to build even more feature-rich and scalable applications.

The team takes pride in the accomplishments and looks forward to expanding the system’s capabilities in the future, ensuring it continues to meet the evolving needs of its users.