

|  |  |
| --- | --- |
| **Project title** | **Proposal and design** |
| **Course Code** | **DSE211/03** |
| **Course Name** | **Application Development Capstone** |
| **Project Start date** | **24-09-2024** |
| **Project Submission Date** | **27-11-2024** |

## ****1. Project Scope and Objectives****

### **1.1 Understanding Project Scope**

**Definition**:  
The Library Management System (LMS) aims to streamline library operations by providing an efficient, user-friendly platform to manage books, users, and transactions. This project will include specific goals, deliverables, and deadlines to ensure the successful deployment of the system.

**Scope Statement**:  
This project focuses on developing a **web-based Library Management System** tailored to streamline library operations and enhance user experience. The system will be accessible to library administrators, staff, and members, ensuring efficient management of library resources and user accounts.

**Goals**:

* Simplify the management of library resources, including books, authors, genres, and publishers.
* Provide seamless user account management, including registration and updates.
* Enable borrowing and returning of books with real-time status tracking.

**Deliverables**:

1. A functional web application with the following modules:
   * User management (add, edit, delete users).
   * Book management (add, edit, delete books).
   * Genre, publisher, and author management.
   * Detailed views for books, authors, and publishers.
   * Borrowing and returning functionality.
   * User login and logout.
2. A centralized database using phpMyAdmin to store and manage data.

**Technologies**:

* Frontend: HTML, CSS.
* Backend: PHP.
* Database: phpMyAdmin.
* Version Control: GitHub.

**Constraints**:

* Limited to using only HTML, CSS, and PHP without external APIs.
* Adhering to a strict 7-week project timeline.

### **1.2 Defining Project Objectives**

**SMART Objectives**:

1. Develop a fully functional Library Management System that can manage at least 100 users and 500 books by the end of Week 5.
2. Implement a borrowing system that processes transactions in under 2 minutes with 95% accuracy.
3. Design a user-friendly interface with an 80% or higher usability score during testing.

## ****2. User Requirements and Expectations****

### **2.1 Gathering User Requirements**

**Methods Used**:

1. **Interviews**: Conducted with library staff to understand their daily challenges.
2. **Surveys**: Distributed to potential library members to gather expectations for an ideal library system.

**Key Findings**:

* Users need a simple and intuitive interface for book search and borrowing.
* Staff require robust management tools for tracking book availability and user activities

### **2.2 Documenting User Requirements**

**Use Cases**:

**Borrow Book**:

* **Actor**: Library member.
* **Scenario**: A member logs in, searches for a book, and borrows it.
* **Outcome**: The system updates the book’s availability and records the transaction.

**Add New Book**:

* **Actor**: Library staff.
* **Scenario**: Staff adds details of a newly purchased book into the system.
* **Outcome**: The book is added to the catalog and made available for borrowing.

**Register User**:

* **Actor**: Library member.
* **Scenario**: A new user registers and creates an account in the system.
* **Outcome**: The user gains access to borrow books.

**User Stories**:

1. As a library member, I want to borrow books so that I can read them at home.
2. As library staff, I want to manage books efficiently so that the catalog is always up-to-date.
3. As an administrator, I want to generate reports on book borrowings so that I can analyze trends.

### **2.3 Managing Expectations**

**Communication Plan**:

* Weekly progress updates through Discord.
* Code updates shared via GitHub.
* A final review meeting after testing and deployment.

## ****3. Project Timeline and Milestones****

### **3.1 Project Timeline**

|  |  |  |
| --- | --- | --- |
| **Week** | **Tasks** | **Deliverables** |
| Week 1 | Requirements gathering, scoping, and architecture | Project proposal, system architecture diagram |
| Week 2 | Database design and wireframe creation | Database schema, wireframes |
| Week 3 | Frontend and backend implementation | Core features, functional modules |
| Week 4 | Testing and debugging | Bug-free application |
| Week 5 | Deployment to live environment | Deployed application |
| Weeks 6-7 | Documentation and user acceptance testing | Final report, user manual |

### 

### **3.2 Milestones**

1. Complete requirements gathering and scoping by **Week 1**.
2. Finalize database schema and UI design by **Week 2**.
3. Implement core functionalities by **Week 3**.
4. Finish testing and debugging by **Week 4**.
5. Deploy the application by **Week 5**.
6. Complete UAT and documentation by **Week 7**.

## ****4. Collaborative Development Tools and Environments****

### **4.1 Collaborative Tools**

* **Version Control**: GitHub for managing code versions and collaboration.
* **Communication**: Discord for real-time discussions and updates.
* **Task Management**: Use a simple task board on Discord to track progress.

**Rationale**:  
GitHub ensures version control and prevents conflicts during development. Discord enables quick communication and status updates, ensuring smooth teamwork.

### **4.2 Development Environment**

* **Languages**: HTML, CSS, PHP.
* **Database**: phpMyAdmin for structured data management.
* **IDE**: Visual Studio for efficient coding and debugging.
* **Coding Standards**: Follow PHP best practices and HTML/CSS conventions for readability and maintainability.

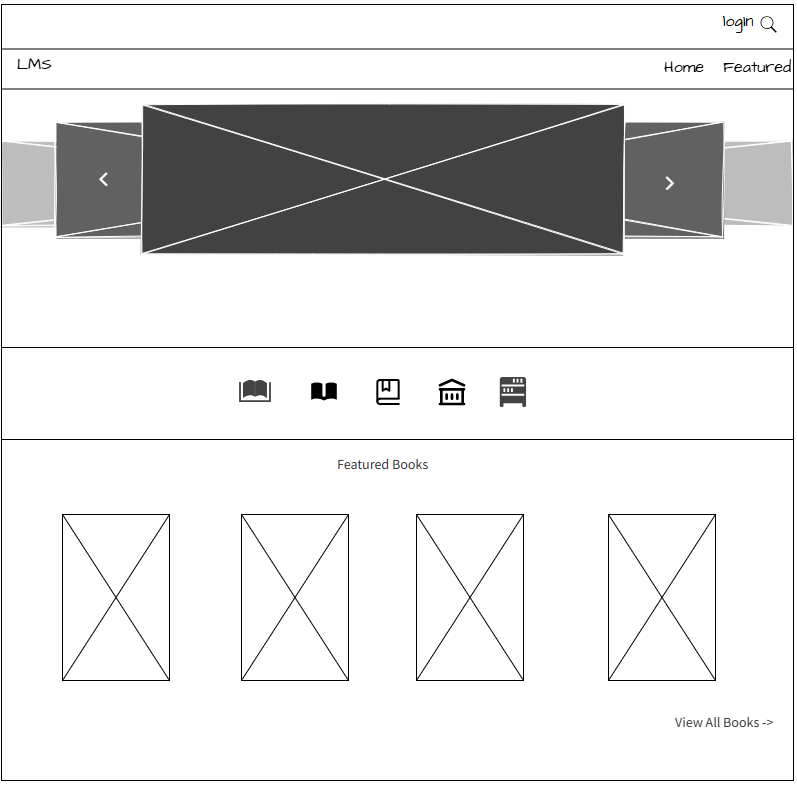
# Project Design

### **1. User Interface (UI) Design**

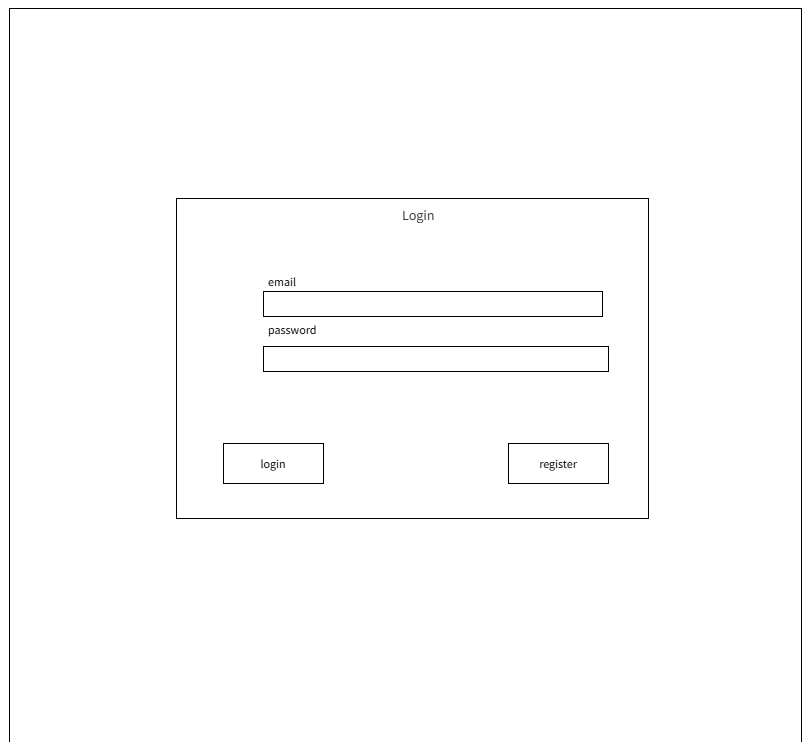
#### **1.1 UI Frameworks Selection**

* **Overview**: For this project, you’ve selected **Bootstrap** and **CSS** to ensure a responsive, user-friendly interface. Bootstrap offers a solid, flexible grid system that helps in building responsive pages easily, while CSS allows custom styling for unique elements.
* **Activity**:
  + **Bootstrap**: Chosen for its ready-made components like navigation bars, buttons, forms, and modals, speeding up development and ensuring consistency across pages.
  + **CSS**: Used for fine-tuning visual styles such as colors, fonts, and layout adjustments.

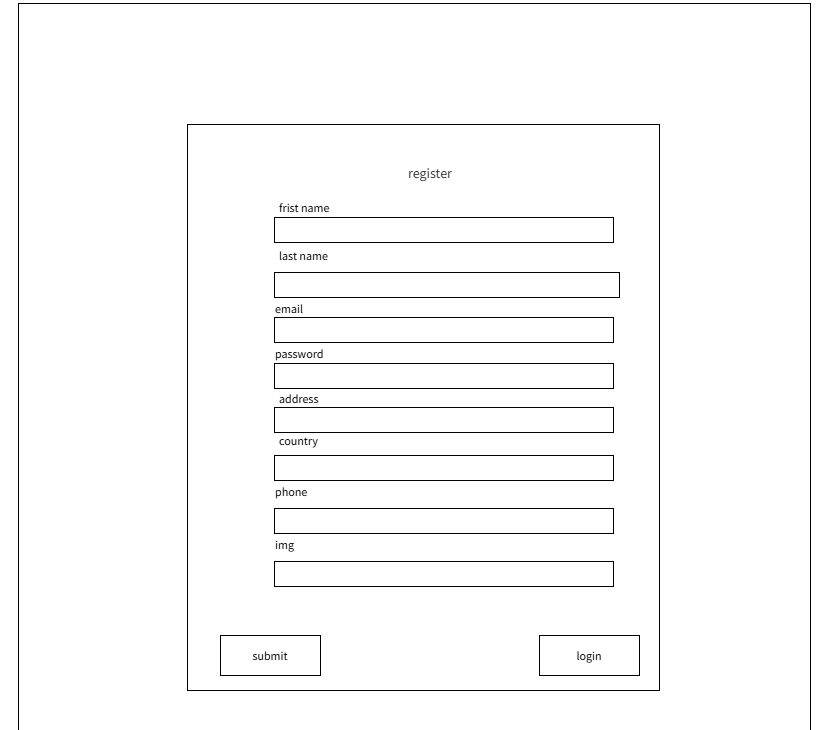
Home



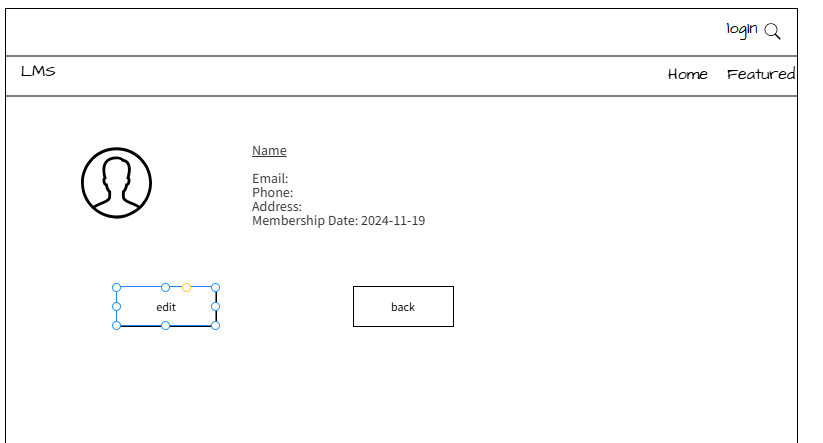
Login



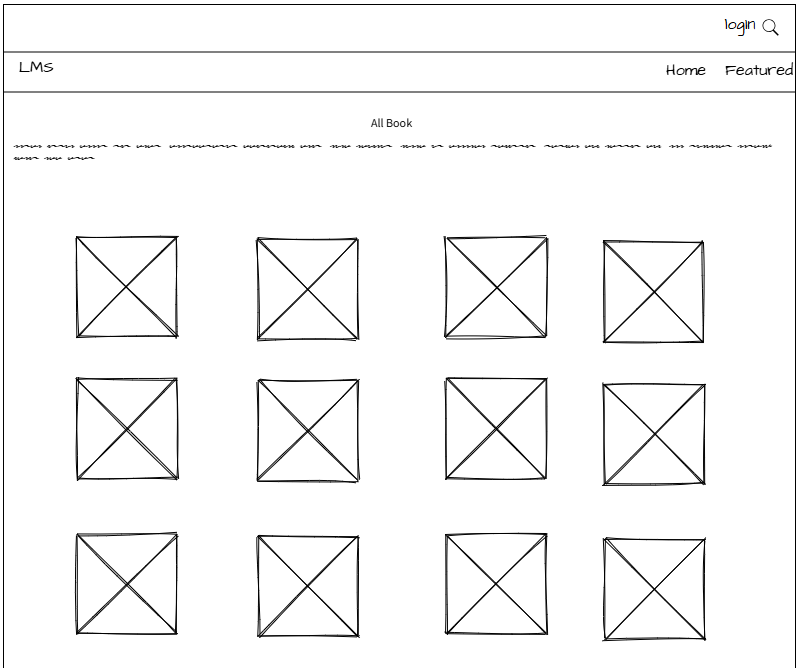
Register



Profile



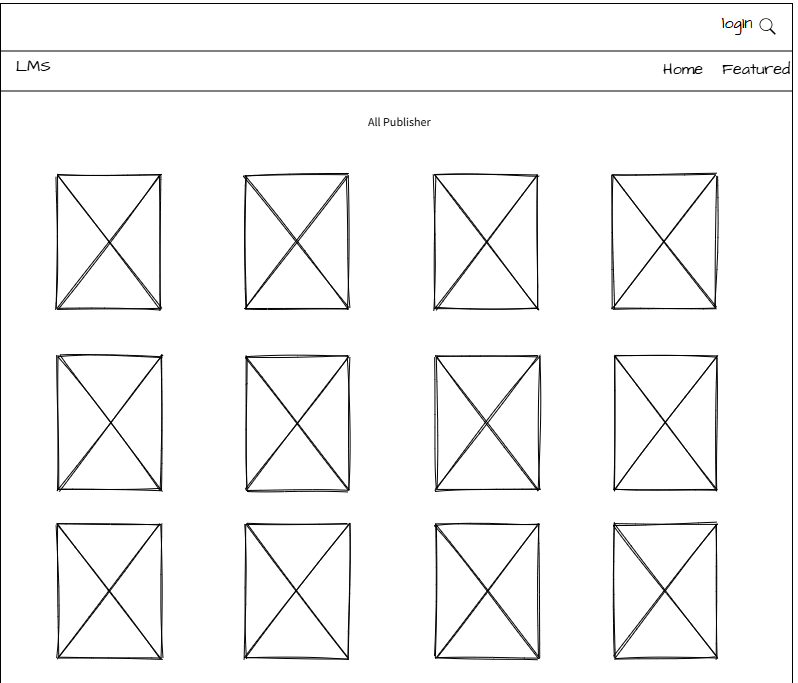
Book list



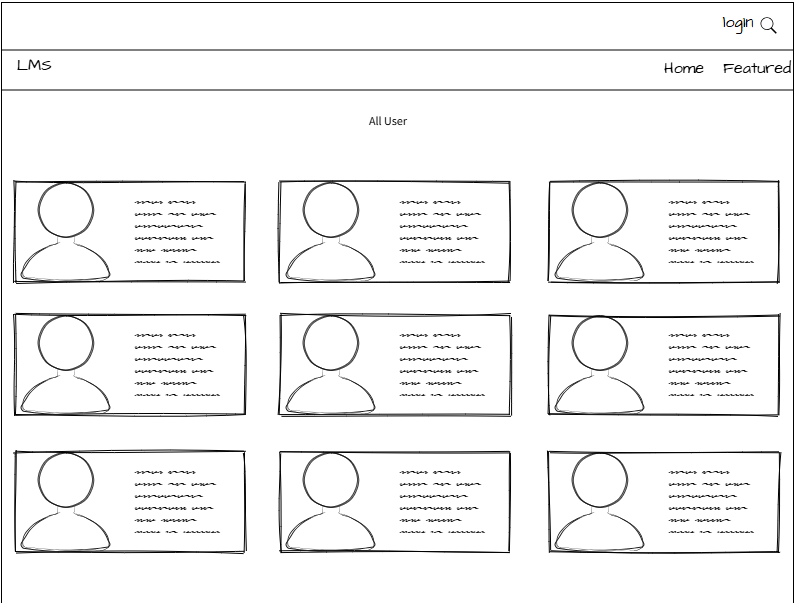
Authors list



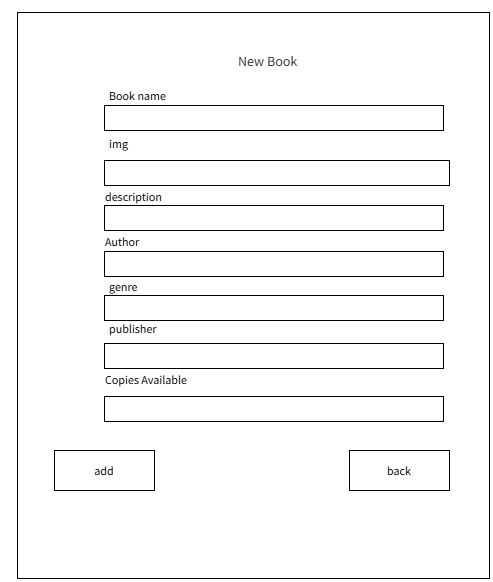
Publishers list



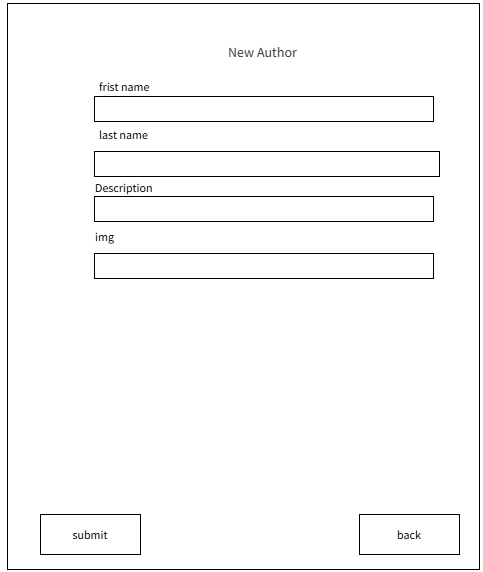
Users list



Add Book



Add Authors



Add publishers



#### **1.3 Responsive Design**

* **Implementation**: Since Bootstrap is already responsive, we will focus on ensuring content is adaptable to various screen sizes.
* **Activity**:
  + **Media Queries**: Further enhance responsiveness with CSS media queries for specific tweaks on mobile devices, such as hiding sidebars or adjusting column layouts.

### **2. Backend Architecture**

#### **2.1 Server-Side Frameworks**

* **Overview**: PHP has been chosen as the backend framework for this project. PHP is suitable due to its widespread use, ease of integration with MySQL, and ability to handle dynamic web pages effectively.
* **Activity**:
  + **PHP**: Facilitates the handling of user requests, server-side processing (such as user login and form submissions), and database interaction.
  + **phpMyAdmin**: Used to manage MySQL databases for storing data like books, authors, publishers, and users.

#### **2.2 RESTful API Design**

* **Definition**: Even though you’re not using APIs, you will design endpoints that reflect a RESTful structure for internal use within the PHP backend to handle various tasks like adding, editing, and fetching data.
* **Activity**:
  + **GET** requests:
    - GET /books: Fetch all books.
    - GET /authors: Retrieve all authors.
    - GET /users: Get all users.
    - GET /book/{id}: Fetch detailed information about a single book.
    - GET /author/{id}: Fetch detailed information about a single author.
  + **POST** requests:
    - POST /add\_book: Submit a new book to the database.
    - POST /add\_author: Add a new author to the database.
    - POST /register: Create a new user account.

#### **2.3 Database Design and Integration**

* **Database Choice**: **MySQL** is chosen due to its support for relational data and ease of management through **phpMyAdmin**.
* **Schema Design**:
  + **Users Table**: Fields include id, username, password, email.
  + **Books Table**: Fields include id, title, author\_id, genre, publisher\_id.
  + **Authors Table**: Fields include id, name, bio.
  + **Publishers Table**: Fields include id, name, address.
  + **Genres Table**: Fields include id, name.
* **Integration**:
  + **Entity-Relationship Diagram (ERD)**: The tables are linked by relationships (e.g., Books are linked to Authors and Publishers through foreign keys).
  + **Interaction**: The PHP backend will handle data manipulation and fetch records from MySQL using SQL queries.

### **3. Framework Integration**

#### **3.1 Framework Integration Strategy**

* **Overview**: PHP and Bootstrap (along with custom CSS) will work together to deliver a seamless user experience.
* **Activity**:
  + **Frontend** (HTML + Bootstrap + CSS): The user interacts with the interface (e.g., viewing books, registering accounts), which sends requests to the backend.
  + **Backend** (PHP): Handles the request logic (e.g., adding books, updating user details), interacts with the MySQL database, and sends a response back to the frontend.

#### **3.2 Code Optimization for Performance**

* **Techniques**:
  + **Lazy Loading**: Implement lazy loading for large book images or resources that aren’t immediately visible.
  + **Minification**: Use tools to minimize CSS and JavaScript file sizes.
  + **Caching**: Cache frequently accessed data like book lists and author details to reduce load times.
* **Activity**: Optimize PHP code for database queries to ensure fast and efficient data retrieval.

#### **3.3 Security Considerations**

* **Security Measures**:
  + **SQL Injection Prevention**: Use prepared statements and parameterized queries in PHP to prevent SQL injection attacks.
  + **Cross-Site Scripting (XSS)**: Sanitize user inputs to prevent XSS vulnerabilities.
  + **Password Protection**: Hash passwords before storing them in the database (e.g., using **password\_hash()** in PHP).
* **Activity**: Implement secure authentication and authorization mechanisms for users (e.g., session-based login for authorized access to features like editing or borrowing).