Library Management System - Software Development Document

---

1. Project Overview

The Library Management System (LMS) is a software application designed to manage a library's daily activities such as tracking books, managing members, issuing books, and collecting fines. The system will be built using the .NET MVC framework, and SQL Server will be used as the database backend.

---

2. Architecture Overview

The application will follow the MVC (Model-View-Controller) architectural pattern. It will use the following components:

- **Model**: Represents the application data (Entities like Books, Members, and Transactions).
- **View**: The user interface that presents the data to the user (Web pages displaying library data).
- **Controller**: Handles user requests and maps them to the appropriate business logic.

3. Technologies

- **Backend**: .NET 6 (or latest version) using ASP.NET MVC.
- **Database**: SQL Server (latest version).
- **Frontend**: HTML5, CSS, Bootstrap, JavaScript, jQuery.
- **DevOps**: Azure DevOps for CI/CD, testing, and deployment.
- **Source Control**: Git.
- **Dependencies**: Entity Framework Core, Automapper (for mapping between DTOs and models), Serilog (for logging).

---

4. Requirements

4.1 Functional Requirements

1. **User Management**:
    - Register/Login/Logout functionality.
    - Role-based access control for Librarian and Member.
    - CRUD operations for user profiles.
2. **Book Management**:
    - CRUD operations for books (add, update, delete, and search).
    - Track book availability and location.
3. **Member Management**:

    - Manage member details (add, edit, delete).
    - Track active and inactive memberships.
4. **Book Issue/Return**:
    - Issue books to members with automatic due date calculation.
    - Track return dates and calculate fines.
5. **Fine Calculation**:
    - Automatic fine calculation based on return date.
    - Record and track fine payments.
6. **Search and Reports**:
    - Search for books, members, and transactions.
    - Generate reports (monthly book issues, overdue books, etc.).

4.2 Non-functional Requirements

1. **Security**: Use ASP.NET Identity for secure authentication and authorization.
2. **Scalability**: The system should be able to handle an increasing number of users and transactions.
3. **Performance**: Ensure minimal response time for queries and reports.
4. **Usability**: The system should have an intuitive and easy-to-navigate user interface.
5. **Availability**: The system should be available 24/7 with minimal downtime.

---

5. System Design

5.1 Database Design A normalized relational database will be used. The primary entities and relationships include:

- **Books**: BookId, Title, Author, ISBN, Category, PublishedDate, CopiesAvailable.
- **Members**: MemberId, Name, Email, Address, MembershipDate, Status.
- **Transactions**: TransactionId, BookId, MemberId, IssueDate, ReturnDate, Fine.
- **Librarians**: LibrarianId, Name, Email, Role, DateJoined.

ER Diagram:

- Book (1

) Transaction.

- Member (1

) Transaction.

5.2 MVC Design

- **Model Layer**:
    - Entities: Book, Member, Transaction.
    - Data Access: Entity Framework Core will be used to map these models to the SQL Server database.
- **Controller Layer**:
    - HomeController: For general application pages (home, login, etc.).
    - BookController: For managing books.
    - MemberController: For managing members.
    - TransactionController: For issuing and returning books.
- **View Layer**:
    - Razor views will be used to render dynamic web pages. Bootstrap will ensure responsive design.

---

6. Implementation Plan

6.1 Development Phases

1. **Phase 1**: User Management Module
    - Create Login, Registration, and Role Management functionalities.
2. **Phase 2**: Book Management Module
    - Implement CRUD operations for books, along with search functionality.
3. **Phase 3**: Member Management Module
    - Implement member management functionality.
4. **Phase 4**: Book Issue/Return and Fines
    - Create logic for book transactions, tracking, and fine calculation.
5. **Phase 5**: Reporting Module
    - Generate reports and implement advanced search features.

6.2 CI/CD Pipeline

- Use Azure DevOps for the CI/CD pipeline:
    - **Build Pipeline**: Automatically trigger builds upon code commits.
    - **Testing**: Unit and integration tests to ensure code quality.
    - **Release Pipeline**: Deploy the system to an Azure-hosted environment.

---

7. Security and Authentication

- **Authentication**: ASP.NET Identity will be used for user management, ensuring secure login, password hashing, and role management.
- **Authorization**: Implement role-based access control (RBAC) to restrict functionalities to certain user roles (e.g., Librarians vs. Members).
- **Data Encryption**: Sensitive data such as passwords will be encrypted using hashing mechanisms.

---

8. Error Handling and Logging

- **Logging**: Use Serilog for logging errors and system events. Logs will be written to a file or a database for analysis and monitoring.
- **Error Handling**: Implement global error handling using try-catch blocks and custom error pages to handle system errors gracefully.

---

9. Testing Strategy

- **Unit Testing**: Test individual components using xUnit or NUnit.
- **Integration Testing**: Test the integration of the application with the SQL Server database and external services.
- **End-to-End Testing**: Use Selenium to test the entire flow of the application (user login, book issue, fine calculation, etc.).

---

10. Deployment Strategy

- **Environment**: The system will be deployed on Azure. Staging and production environments will be set up for testing and deployment.
- **Database**: SQL Server will be hosted on Azure with automated backups configured.
- **Scaling**: Set up Azure Autoscale to handle traffic spikes.

---

11. Maintenance and Support

1. **Documentation**:
    - All development-related documents, including API references and database schemas, will be updated.
    - Maintenance logs will be kept for troubleshooting and system updates.
2. **Monitoring**:
    - Application Insights will be used for real-time monitoring, performance metrics, and error tracking.

This document outlines the approach for developing a Library Management System. Following the MVC pattern ensures maintainability, while .NET and SQL Server provide a robust platform for development.