Library Management

Problem statement:

The objective is to design and implement a comprehensive Library Management System to streamline library operations, enhance user experience, and automate key processes for a library.

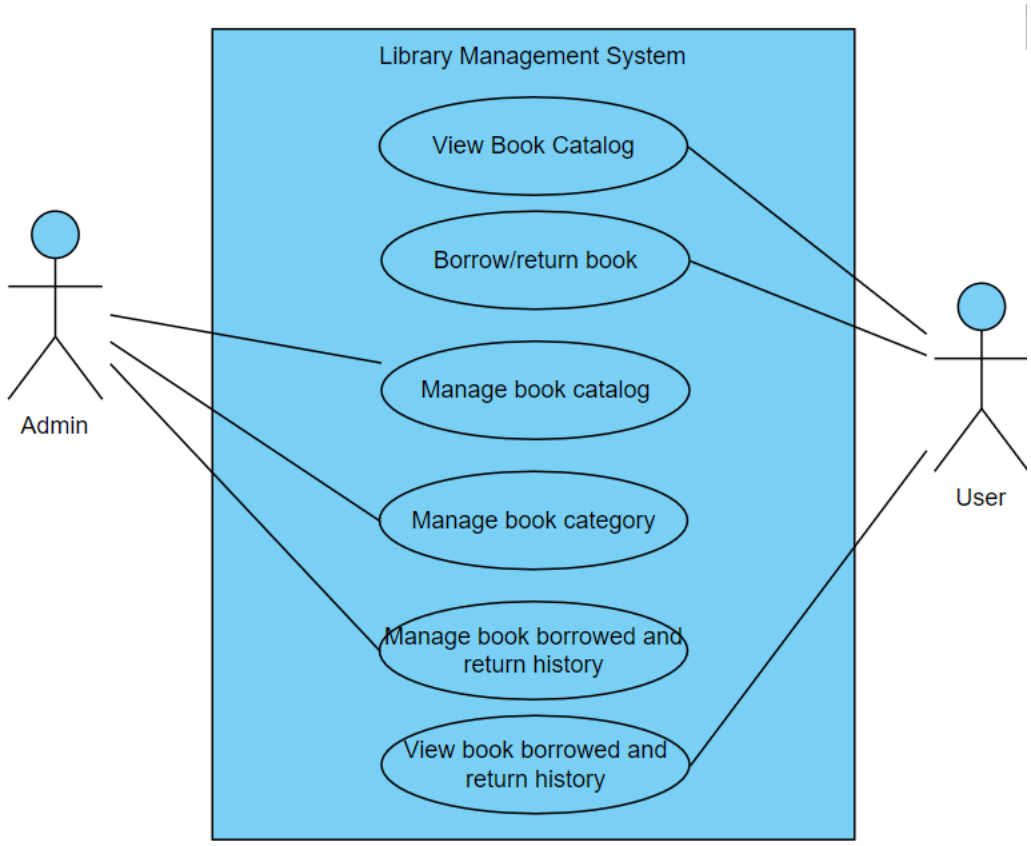
Scope

1. **Employee Registration and Authentication:** Allow users to register, log in, and securely manage their accounts.
2. **Book Catalogue and Inventory:** Manage a digital catalogue of all available books and automatically update the inventory as books are borrowed or returned.
3. **Borrowing and Returning Books:** Allow users to borrow and return books through a user-friendly interface and set borrowing limits and due dates.
4. **Reservation System:** Enable users to reserve books that are currently on loan.
5. **Late Fee Calculation:** Automatically calculate and notify users about late fees for overdue books.
6. **Search and Filter Functionality:** Implement a robust search and filter system for users to easily find books based on various criteria.
7. **Email Notifications:** Send automated email notifications for overdue books, reservation confirmations, and other relevant updates.

Technologies:

- Frontend: React.js / Angular Js.
- Backend: Java, Spring Boot/C#, .Net / Python Djnago for API development.
- Database: MySql / Sql Server.
- Authentication: JSON Web Tokens (JWT) for secure user authentication.

Use case Diagram:



Use Cases:

Actor: User/Employee

- Use Case: User/Employee Registration and login.
- Use Case: View book Catalogue.
- Use Case: Borrow/Return Books.
- Use Case: view history of borrowed books.
- Use Case: view fine for late return.
- Use Case: view and reserve the book in loan.

Actor: Administrator

- Use Case: Log In
- Use Case: Manage Book Catalogue
- Use Case: Manage Book Categories
- Use Case: Manage users.
- Use Case: View all borrowed books; total fine, individual user fine.

System: Security and Authentication

- Use Case: Authenticate User

System: Database Management

- Use Case: Store Book Information
- Use Case: Store Book genre Information.
- Use Case: Store Borrowed Book with fine amount details Information.
- Use Case: Store Book Reservation Information
- Description: Manages the storage information related to book, including details such as

Development Process:

1. **User Registration and Admin Login:**
    a. User and Admin can create accounts, providing personal details (name, gender, contact number, address, etc.)
    b. The system validates the information and creates user profiles.
    c. Employee and Admin log in using their credentials (username/email and password).
2. **User Dashboard:**
    a. Users can browse Book available, view detailed descriptions, images, book name, author name, publication and with published date, edition and language.
    b. Users navigate through book categories (such as Fiction, Nonfiction, Fantasy, Thriller, Crime, Technical) search for specific items.
        i. Include filters for category, language, author.
        ii. Implement a search feature with autosuggestions and predictive text.
        iii. Clicking on book from list of books to view more information about a book with availability detail, borrow/return option or reserve option.
    c. User can raise the reserve book request for the books already borrowed by some other user.
    d. User can borrow maximum of five books. Once the user reaches the limit they can't borrow or reserve request for a book. User must return the book in 10 days from the borrowed date.
    e. User can view their history of borrowed books with borrowed date, return date, fine amount for a book which is not returned within return date. Fine amount is 5 Rs/- per day.
    f. User can set the status of the book as lost when user received the return book request.
3. **Admin Dashboard:**
    a. Admin can add new book by providing following information Title, Author, ISBN (International Standard Book Number), Publisher Name, Publication Date, Edition, Genre/Category, Description/Summary, Language, Number of Pages, cost Of Book.
    b. Admin can delete and update the book.
    c. Admin can add and delete the new category of book list.
    d. Admin can send return request book to the user who not returned the book for long period of time.
    e. Admin can view the history of book borrowed and returned book, also they can view history of book reservation.
4. **Security and Compliance**
    a. User authentication and authorization are enforced to ensure data privacy.

1. JWT Authentication:

JWT authentication involves generating a token upon successful user login and sending it to the client. The client includes this token in subsequent requests to authenticate the user.

a. User Login: Upon successful login (using valid credentials), generate a JWT token on the server.
b. Token Payload: The token typically contains user-related information (e.g., user ID, roles, expiration time).
c. Token Signing: Sign the token using a secret key known only to the server. This ensures that the token hasn't been tampered with.

d. Token Transmission: Send the signed token back to the client as a response to the login request.
e. Client Storage: Store the token securely on the client side (e.g., in browser storage or cookies).

2. JWT Authorization:

JWT authorization involves checking the token on protected routes to ensure that the user has the required permissions.

a. Protected Routes: Define routes that require authentication and authorization.
b. Token Verification:
   i. Extract the token from the request header.
   ii. Verify the token's signature using the server's secret key.
c. Payload Verification:
   i. Decode the token and extract user information.
   ii. Check user roles or permissions to determine access rights.
d. Access Control: Grant or deny access based on the user's roles and permissions.

Logout:

- Logging out involves invalidating the JWT token on both the client and the server to prevent further unauthorized requests.

5. User Profile:

- Each actor of the application must have a user profile page.
- Profile page must allow the user to upload picture, edit/delete information such as name, phone number.
- It is recommended to have change password feature in profile page.

Project Development Guidelines

The project to be developed based on the below design considerations.

1	**Backend Development**	• Use Rest APIs (Spring boot/ASP. Net Core Web API to develop the services. • Use Java/C# latest features. • Use ORM with database. • perform backend data validation. • Use Swagger to invoke APIs. • Implement API Versioning. • Implement security to allow/disallow CRUD operations. • Message input/output format should be in JSON (Read the values from the property/input files, wherever applicable). Input/output format can be designed as per the discretion of the participant. • Any error message or exception should be logged and should be user-readable (not technical). • Database connections and web service URLs should be configurable. • Implement JWT for Security. • Follow Coding Standards with proper project structure.

2	**Error Handling and Exception Management**	• Ensure errors and exceptions are handled with the status code. • Ensure that there are appropriate error-handling mechanisms in place to handle unexpected situations gracefully. Implement custom exceptions. • Ensure robust error logging and reporting mechanisms.

3	**Testing and Debugging:**	• Implement Unit Test Project for testing the API.
		• Establish what each unit test aims to verify request mappings, status codes, returned views or response bodies.
		• Focus on business logic and typically mock away data access layers.
		• Ensure the interaction with the database behaves as expected.
		  • Test Structure
		  • Arrange: Set up the test data and mocks.
		  • Act: Execute the method being tested.
		  • Assert: Verify the output or behaviour with assertions.

4	**Logging**	• Define different logging levels (e.g., DEBUG, INFO, WARN, ERROR) based on the severity of the logged message. • Ensure that log messages are descriptive and informative, providing enough context to understand the event or condition being logged. • Include relevant information such as timestamps, user IDs, and error codes in log messages. • Must be logged in a separate file under the source folder of the application.
5	**Email notification**	• send email notification to the user when there is user action performed. • notify the user when user logged in into the profile and when database interaction is performed the notification are send to corresponding email.
6	**File Management**	• PDF formats should be used for bills/invoices and such kinds. • Ensure to format the data properly in the PDF and make it downloadable and sharable via email. • If the application (like quiz tool) has a way to interact with excel/csv (like bulk upload/download) implementation is a must.

Frontend Constraints

1.	**Layout and Structure**	Create a clean and organized layout for your registration and login pages. You can use a responsive grid system (e.g., Bootstrap or Flexbox) to ensure your design looks good on various screen sizes.
2	**Visual Elements**	**Logo:** Place your application's logo at the top of the page to establish brand identity.

		Form Fields: Include input fields for email/username and password for both registration and login. For registration, include additional fields like name and possibly a password confirmation field.
		Buttons: Design attractive and easily distinguishable buttons for "Register," "Login," and "Forgot Password" (if applicable).
		Error Messages: Provide clear error messages for incorrect login attempts or registration errors.
		Background Image: Consider using a relevant background image to add visual appeal.
		Hover Effects: Change the appearance of buttons and links when users hover over them.
		Focus Styles: Apply focus styles to form fields when they are selected
3.	**Colour Scheme and Typography**	Choose a colour scheme that reflects your brand and creates a visually pleasing experience. Ensure good contrast between text and background colours for readability. Select a legible and consistent typography for headings and body text.
4.	**Registration Page, add product page by seller, add shipping address page by user**	**Form Fields:** Include fields for users to enter their name, email, password, and any other relevant information. Use placeholders and labels to guide users.
		Validation: Implement real-time validation for fields (e.g., check email format) and provide immediate feedback for any errors. **Form Validation:** Implement client-side form validation to ensure required fields are filled out correctly before submission.
	Registration Page	**Password Strength:** Provide real-time feedback on password strength using indicators or text. **Password Requirements**: Clearly indicate password requirements (e.g., minimum length, special characters) to help users create strong passwords.
		Registration Success: Upon successful registration, redirect users to the login page.
5.	**Login Page**	**Form Fields:** Provide fields for users to enter their email and password.
		Password Recovery: Include a "Forgot Password?" link that allows users to reset their password.

6.	**Common to React/Angular**	• Use Angular/React to develop the UI. • Implement Forms, databinding, validations, error message in required pages. • Implement Routing and navigations. • Use JavaScript to enhance functionalities. • Implement External and Custom JavaScript files. • Implement Typescript for Functions Operators. • Any error message or exception should be logged and should be user-readable (and not technical). • Follow coding standards. • Follow Standard project structure. • Design your pages to be responsive so they adapt well to different screen sizes, including mobile devices and tablets.

Good to have implementation features

1. Generate a SonarQube report and fix the required vulnerability.
2. Use the Moq framework as applicable.
3. Create a Docker image for the frontend and backend of the application.
4. Implement OAuth Security.
5. Implement design patterns.
6. Deploy the docker image in AWS EC2 or Azure VM.
7. Build the application using the AWS/Azure CI/CD pipeline. Trigger a CI/CD pipeline when code is checked-in to GIT. The check-in process should trigger unit tests with mocked dependencies.
8. Use AWS RDS or Azure SQL DB to store the data.