جامعة 6 اكتوبر

October 6 University

كلية نظم المعلومات وعلوم الحاسب

Faculty of Information System and Computer Science

Case study: Library Management System Project Overview

Objective

The goal of this project is for student teams to apply software engineering principles to design, implement, test, and document a Library Management System. The system will allow users (library members) to search for, borrow, and return books, while administrators (librarians) manage book inventory, user accounts, and borrowing rules.

Stakeholders

1. Library Members (Users): Individuals who use the system to browse, borrow, and return books.
2. Librarians (Administrators): Personnel responsible for managing the library's book inventory, user accounts, and borrowing rules.
3. Developers: responsible for developing the Library Management System.
4. Client: providing requirements and feedback.

Functional requirements:

1. User Registration and Authentication

- Users (library members) can register with an email and password.
- Implement secure password storage (e.g., hashing with bcrypt).
- Users can log in and log out securely.
- Librarians have a separate login interface.

2. User Profile Management

- Users can view and edit their profile information.
- Users can change their password and manage contact information.

3. Book Catalog and Search

- Display a catalog of available books with categories and genres.
- Users can search for books by title, author, ISBN, or keyword.
- Display detailed information for each book (author, publication date, description, availability).

4. Borrow Management

- **Borrowing Books:**
  - Users can browse available books and borrow them for a set period (e.g., 14 days).
  - Users can see the number of available copies for each book.
  - The system will track the borrowing duration and set a due date for the return.
- **Return Books:**
  - Users can return books via the system, and the system will update the availability status.
- **Borrow History:**
  - Users can view their borrowed history, including current and past borrowed books, with return statuses.
- **Notifications:**

    - Automated email notifications will remind users of upcoming due dates (e.g., 3 days before and on the due date).

- **Extensions/Renewals:**
    - Users can request an extension for borrowed books if they are not reserved by other users.

5. **Admin Dashboard (Librarian)**
    - **Manage Book Inventory:**
        - Librarians can add, edit, or remove books from the library's catalog.
        - Track the availability of books and manage copies available for borrowing.
    - **Manage Borrowing Rules:**
        - Librarians can set borrowing limits (e.g., maximum number of books a user can borrow) and borrowing periods (e.g., 14 days).
        - They can extend borrow periods for users upon request and track overdue books.
    - **Manage Users:**
        - Librarians can view, edit, or deactivate user accounts.
        - View user borrowing history and overdue books.
6. **Notifications and Alerts**
    - **Users will receive notifications via email for:**
        - Book due date reminders (3 days before due date and on the due date).

- Overdue book alerts.

- **Librarians will receive alerts for overdue books and system errors.**

7. **Reservation System**
   - Users can place reservations for books that are currently unavailable (all copies are borrowed).
   - Notify users when a reserved book becomes available.

Non-Functional Requirements

1. **Performance**
   - The system should support at least 100 concurrent users without performance degradation.
   - Pages should be loaded within 2 seconds under normal load conditions.

Capacity estimation:

- It involves determining the necessary resources to handle expected user loads and data volumes efficiently. The goal is to ensure that the system performs well under peak-load conditions and can be scaled as needed.
- **Concurrent Users:** Assumes that up to 1000 users (library staff and patrons) might be using the system simultaneously for searching, borrowing, returning books, etc.

- **Transactions per Second (TPS)**: Estimate the number of transactions the system can handle per second. Assume **TPS is** 10, Given 1000 daily transactions spread over 10 hours of operation, the peak TPS is around 10 transactions per second.

- **Data Storage**: Estimate the amount of data storage needed to store the number of books, users, transactions, etc.
  - **Books Data Storage:**
  - Number of Books: 1,000
  - Size per Book: 720 bytes
  - Total Storage for Books: 1,000×720 bytes = 720,000 bytes = 720 KB
- **Members Data Storage:**
  - **Number of Members:** 5000
  - **Size per Member:** 250 bytes
  - **Total Storage for Members:** 5000×250 bytes = 12,50,000 bytes = 1250 KB
- **Transactions Data Storage:**
- **Number of Transactions per Year:** 100 transactions/day × 365 days/year = 36,500 transactions
- **Size per Transaction:** 80 bytes
- **Total Storage for Transactions**: 36,500 × 80 bytes = 29,20,000 bytes = 2920 KB
- **Total Data Storage**
  - **Books:** 720 KB
  - **Members:** 25 KB
  - **Transactions:** 292 KB

        - **Total**: 720 KB + 25 KB + 292 KB = 1,037 KB ≈ 1 MB

- **Network Bandwidth:** Assess the required bandwidth to support data transfer and user interactions without latency.
    - **Daily Data Transfer**
        - Assuming each read/write operation involves 1 KB of data
        - **Daily Transactions:** 100
        - **Daily Reads (search, views, etc.):** 1000
        - **Total Daily Data Transfer:** (100+1000) × 1 KB = 1100 KB

2. Security

- Use HTTPS for all sensitive data transmissions.
- Protect against common web vulnerabilities (e.g., SQL injection, XSS).
- Encrypt sensitive data stored in the database.

3. Usability

- Intuitive and user-friendly interface.
- Consistent design and navigation across all pages.
- Responsive design to support various devices (desktop, tablet, mobile).

4. Scalability

- Design the system to accommodate future growth (more users, books, and additional features).

5. Reliability and Availability

- The system should have an uptime of 99.9% during operational hours.
- Implement error handling to manage and log exceptions gracefully.

6. **Maintainability**

    - Code should be modular, well-documented, and follow coding standards.
    - Use version control to manage code changes effectively.

Technical Requirements

1. **Programming Language and Framework**

    - **Choose one of the following:**
        - Java with Spring Boot
        - Python with Django
        - C# with ASP.NET Core

2. **Database**

    - Use a relational database (MySQL or PostgreSQL).
    - Design an efficient database scheme with proper indexing.

3. **Version Control (optional)**

    - Use Git for version control.
    - Host the repository on GitHub, GitLab, or Bitbucket.

4. **Development Environment**

    - Teams can choose any suitable IDE (IntelliJ IDEA, Visual Studio Code, Eclipse).

- Ensure consistent development environments across the team.

5. **Testing Frameworks**
    - **Unit Testing:**
        - Java: JUnit
        - Python: PyTest
        - C#: NUnit
    - **Integration Testing:**
        - Use framework-specific tools (Spring Test, Django Test).
6. **Deployment** (Optional)
    - Deploy the application using free-tier cloud services (Heroku, AWS Free Tier).

Prepared by:

Eng. Asmaa AbdulQawy