Implementation and Testing

for

Library Management System V2

Version 1.0 approved

Prepared by José E. Plaud Ortiz

Universidad de Puerto Rico, Rio Piedras

December 15, 2024

Revision History

Name	Date	Reason For Changes	Version
José Plaud Ortiz	12/16/24	Final Application Update	1.0

Table of Contents

1. Introduction

1.1 Purpose

This document serves to outline the implementation details and testing procedures for the **Library Management System (LMS)**. It provides developers and testers with clear guidelines on how to set up the application, the technical specifications of the system, and the methods for validating its functionality.

1.2 Document Conventions

- **Bold Text**: Used for section headers and important notes.
- *Italic Text*: Used for warnings, optional steps, or tips.
- Technical terminology:
    - **Frontend** refers to the React-based user interface of the LMS.
    - **Backend** refers to the Django-powered REST API.
    - **Database** refers to the MySQL database used to store persistent data

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers**: Focus on Section 2 (Implementation Details) for setup and configuration instructions.
- **Testers**: Focus on Section 3 (Test Cases) to validate application functionality.
- **Stakeholders**: Review the document as a whole for a high-level understanding of the system.

1.4 References

- React Documentation: https://reactjs.org/docs/
- Django Documentation: https://docs.djangoproject.com/
- MySQL Documentation: https://dev.mysql.com/doc/

- JWT Authentication Guide: https://jwt.io/introduction/

2. Implementation Details

*This section provides the technical specifications and setup instructions required to run the **Library Management System (LMS)** locally. It includes details about the frontend (React), backend (Django), and the database (MySQL), along with step-by-step guidance on initializing the system and mock data. Credentials for different user roles (User and Staff) are also provided.*

2.1 Technical Specifications

- **Frontend**: React and Node.js, running locally. The application is not hosted externally.
- **Backend**: Django and Python 3.10, running locally.
- **Database**: MySQL Community Edition for persistent data storage.
- **Other Tools**: JWT is used for user authentication.

2.2 How to Run the Application

Frontend Setup

1. Clone the repository and navigate to the `lms_frontend` directory.
2. Install dependencies using `npm install`.
3. Start the React development server with `npm run dev`.
4. Access the frontend at: http://localhost:3000.

Backend Setup

1. Navigate to the `lms_backend` directory.
2. Set up a Python virtual environment and activate it.
3. Install required packages using `pip install -r requirements.txt`.
4. Configure the MySQL database:
   - Create the database using the appropriate SQL commands.
   - Apply migrations using `python manage.py migrate`.
5. Load mock data with `python manage.py load_data`.

6. Start the Django development server with `python manage.py runserver`.
7. Access the backend at: http://127.0.0.1:8000.

2.3 Credentials

Role	Email	Password
Librarian	librarian1@example.com	password1
User	user1@example.com	password3

3. Test Cases

This section outlines the key functionalities of the LMS and specifies the test cases required to validate its behavior. It includes user authentication, routing and role-based access control, book catalog access, CRUD operations for staff users, and borrowing/returning workflows for regular users. Each test case includes the steps to follow and the expected outcomes to ensure system reliability.

3.1 Test Case – User Login

Steps:

1. Navigate to the login page.
2. Enter valid credentials and click "Login."
3. Enter invalid credentials.
4. Leave fields blank and attempt to log in.
5. Enter credentials with leading/trailing spaces.
6. Attempt SQL injection in email/password fields.

Expected Result:

- Successful login redirects to the respective user dashboard (User or Staff).
- Error messages are displayed for invalid credentials, blank fields, or invalid formats.

3.2 Test Case – Routing and Access

Steps:

1. Access the home route without logging in.
2. Log in as a **User** and access staff-only routes.
3. Log in as **Staff** and attempt to access user-specific routes.
4. Attempt to access invalid routes.

Expected Result:

- Unauthorized routes redirect to a 403 error page.
- Invalid routes show a 404 error.

3.3 Test Case – Book Catalog Access

Steps:

1. Navigate to the book catalog as a logged-in user.
2. Search for books by title, author, or genre.
3. Attempt partial title searches.
4. Filter books by availability.
5. Access detailed book information.

Expected Result:

- Book catalog loads with accurate data.
- Search and filter return relevant results.

3.4 Test Case – CRUD Operations for Books (Staff Role)

Steps:

1. Log in as **Staff**.
2. Add a new book with valid details.
3. Add a book with an existing ISBN.

4. Update book details (e.g., title, author).
5. Delete a book that is currently borrowed.
6. Delete an available book.

Expected Result:

- Book addition, update, and deletion behave as expected.
- Borrowed books cannot be deleted.

3.5 Test Case – Borrow and Return Books (User Role)

Steps:

1. Log in as a **User**.
2. Borrow an available book.
3. Attempt to borrow a book with no copies available.
4. Return a book on time.
5. Return a book late.
6. Verify waiting list behavior for unavailable books.

Expected Result:

- Borrow and return operations work as expected.
- Users are added to waiting lists when books are unavailable.
- Overdue returns display appropriate notifications and fees.

4. Bugs and Fixes

Identified Bugs

1. **Issue**: React local setup was failing due to lack of a `utils.ts` file that was not being tracked by git.
   - **Fix**: Removed the file from `.gitignore` and committed it into the repository.

2. **Issue**: Mock Data was being marked invalid due to a conflict with the primary keys.
    - **Fix**: Removed the IDs from the `.sql` file, allowing MySQL to create them automatically via auto-increment.
3. **Issue**: Passwords in the mock data were not hashed, causing authentication to fail.
    - **Fix**: Updated the `load_data` management command to hash passwords using Django's authentication system.
4. **Issue**: CORS errors when connecting frontend to backend during initial setup.
    - **Fix**: Add the frontend URL to `CORS_ALLOWED_ORIGINS` in Django settings.
5. **Issue**: Static files not served correctly in production.
    - **Fix**: Run `python manage.py collectstatic` and ensure static files are handled correctly in local development.
6. **Issue**: Mock data not loading during initial backend setup.
    - **Fix**: Ensure the `load_data` management command is executed after migrations.