Software Verification and Validation Report
Library Management System

Team: Fantastic Two

July 20, 2025

Team Introduction and Report Overview

Team Members

Team Member	Responsibilities
Sifat Sabrina Rahman	Project Manager
Priyanka Saha	Senior Developer

Report Overview

This document presents the complete Verification and Validation (V&V) process for the Library Management System (LMS) developed by our team. The report includes:

- Comprehensive evaluation of testing methodologies applied
- Detailed evidence from unit, system, and exploratory testing
- Analysis of testing outcomes and identified limitations
- Conclusions about system quality and recommendations

Key Features Covered:

- User authentication and role management
- Book catalog and inventory management
- Book borrowing and return workflows
- Loan tracking and fine calculation

"Quality is not an act, it is a habit."
– Aristotle

Contents

Chapter 1

Introduction

1.1 Project Background

The Library Management System (LMS) is a web-based application developed to digitize and manage everyday library tasks, such as managing books, tracking loans, and handling user roles. The system distinguishes between two primary user roles: Library Admin, who manages the entire system, and Member, who can browse, borrow, and return books. The system is built with modular components to ensure maintainability, security, and scalability.

1.2 Importance of Verification and Validation

In software engineering, Verification involves checking that the product is being built correctly according to its specifications, while Validation ensures the final product meets user needs and expectations. These two pillars of quality assurance are critical for detecting faults early and delivering a dependable software system [1].

1.3 Purpose of the Report

The purpose of this report is to document the Verification and Validation (V&V) efforts undertaken during the development of the LMS. It aims to assess

whether the system meets its design and functional requirements, maintains quality standards, and delivers a satisfactory user experience.

1.4 Scope of the Report

This report includes:
- Review of related V&V practices
- Testing methodologies applied to evaluate the LMS
- Key findings and limitations observed during testing
- Assessment of overall system quality

1.5 Definitions, Acronyms and Abbreviations

LMS	Library Management System
Admin	Library Administrator
Member	Registered User of the Library
ISBN	International Standard Book Number
V&V	Verification and Validation

Chapter 2

Literature Review

2.1 Role of Software Verification and Validation

Verification and Validation (V&V) are critical components of software engineering that ensure software correctness, completeness, and fitness for use. According to [1], verification is "the process of evaluating work-products...to ensure that they meet specified requirements," while validation is "the process of evaluating the final product to determine whether it satisfies the intended use and user needs."

2.2 V&V Techniques and Practices

Software V&V encompasses a range of techniques, including but not limited to static analysis, unit testing, integration testing, system testing, and user acceptance testing. [2] emphasizes the importance of incorporating V&V early in the development lifecycle to reduce downstream defects and maintenance costs.

2.3 Agile V&V Context

With the rise of agile methodologies, V&V practices have evolved to fit fast-paced, incremental workflows. According to [3], effective V&V in agile projects hinges on test automation, team collaboration, and rapid feedback loops.

2.4 V&V in Web-Based Systems

For web-based systems like a Library Management System, testing must consider functional correctness, security, usability, and performance. Research suggests that combining functional and non-functional testing ensures a more robust user experience [4].

Chapter 3

Methodology

To ensure the correctness, reliability, and usability of the Library Management System (LMS), a three-pronged verification and validation (V&V) strategy was applied.

3.1 Unit Testing — API-Level Validation with Supertest

Tool and Scope

- **Tool Used:** Supertest (Node.js)
- **Scope:** Isolated backend route testing (Express.js)

Description

Unit testing focused on verifying the correctness of individual backend routes and logic modules in isolation from the rest of the system. These included tests for:

- Borrowing books
- Returning books with fine calculation
- User login validation
- Book CRUD operations

Objectives

The unit testing aimed to:
- Ensure core business logic works as expected
- Detect regressions at the component level
- Provide fast feedback to developers

Test Examples

Key test cases included:
- **Borrowing a book:** Verifies availability updates
- **Returning after due date:** Validates fine calculation
- **Unauthorized access:** Confirms admin route blocking

3.2 System Testing — End-to-End Testing with Cypress

Tool and Scope

- **Tool Used:** Cypress
- **Scope:** Full user journey across frontend + backend

Description

System testing validated complete workflows from a user's perspective in a browser-based environment. It ensured that all parts of the LMS — user interface, API integration, and database interactions — worked as expected together.

Objectives

- Simulate realistic usage scenarios.
- Verify UI elements and navigation workflows.
- Ensure role-based access control across components.

Tested Flows

- Member logs in → browses books → borrows → returns.
- Admin adds a new book → book appears for all users.
- Access control: Members cannot visit /admin routes.

3.3 Manual Testing for UI

Scope

Manual testing covered the following key areas of the Library Management System:

- **User & Role Management:**
  - Creating, updating, and deleting users
  - Enforcing role-based access for Admin and Member roles
- **Book Management:**
  - Adding, editing, and deleting books
  - Handling optional fields and file uploads
- **Borrow & Return Workflow:**
  - Borrowing available books
  - Returning books
  - Tracking loan history and fines
- **Access Control:**
  - Ensuring users can only access features permitted by their roles
- **System Feedback & Validation:**
  - Verifying appropriate messages for valid actions
  - Validating error messages for invalid actions

3.4 Exploratory Testing — Real-World Simulation

Approach

- Unscripted user exploration of the live app

Tools

- Manual browser testing with UI checklist and notes

Testing Metaphors

- **Business District Tour:**
  - Focused testing of high-traffic, essential components
  - Includes: borrow/return flow, login, admin dashboard
- **Guidebook Tour:**
  - Broader exploratory walkthrough of secondary features
  - Includes: uploading images, pagination, form validations

Objectives

- Identify issues missed by automated tests
- Validate app behavior under unexpected inputs or usage patterns
- Evaluate UX, usability, and resilience

Chapter 4

Evidence of Verification and Validation

This section provides practical evidence to demonstrate how the applied testing methodologies—Unit Testing, System Testing, and Exploratory Testing—were effectively implemented and supported by the project. It includes screenshots, code snippets, and links to video recordings for clarity and verification.

4.1 Unit Testing Evidence

Table 4.1: Unit Test Modules

File	**Module Tested**
auth.test.js	Authentication
books.test.js	Book management
lending.test.js	Lending transactions
users.test.js	Admin user management

```
await request(app).post('/api/auth/register').send({
  username: 'alice',
  password: 'pass123',
}).expect(201);

const res = await request(app).post('/api/auth/login').send({
  username: 'alice',
  password: 'pass123',
}).expect(200);

expect(res.body.token).toBeDefined();
```

Figure 4.1: Authentication test implementation

This test confirms successful user registration and login, and asserts that a valid JWT token is returned.

```
const resCreate = await request(app)
  .post('/api/books')
  .set('Authorization', `Bearer ${token}`)
  .field('bookId', 'B1')
  .field('title', '1984')
  .field('isbn', '9780451524935')
  .expect(201);

expect(resCreate.body.title).toBe('1984');
expect(resCreate.body.availableCopies).toBe(2);
```

Figure 4.2: Book management test implementation

This unit test ensures that an admin can create a book, and the response matches the input data.

```
const borrow = await request(app)
  .post('/api/loans/borrow')
  .set('Authorization', `Bearer ${memberToken}`)
  .send({ bookId, dueDate: dueDate.toISOString() })
  .expect(201);

await request(app)
  .put(`/api/loans/return/${borrow.body._id}`)
  .set('Authorization', `Bearer ${memberToken}`)
  .expect(200);
```

Figure 4.3: Loan processing test implementation

Tests borrowing and returning a book using the member's credentials. Both endpoints are verified.

```
const res_1 = await request(app)
  .post('/api/users/create')
  .set('Authorization', `Bearer ${token}`)
  .send({ username: 'newUser', password: 'abc123', role: 'user' })
  .expect(201);

expect(res_1.body.message).toBe('User created successfully');
```

Figure 4.4: Admin User Creation test implementation

Verifies that an admin user can create new users and retrieve them from the list.

Summary of Tools & Practices

Item	Description
Testing Tool	Supertest
Test Runner	Jest
Database	MongoDB
Best Practices	Idempotent setup, test isolation, assertions

4.2 System Testing Evidence

Testing Framework

For system testing of the Library Management System, the Cypress framework was employed to validate the full user workflow across the frontend and backend, ensuring proper integration between modules.

Component	Details
Framework:	Cypress
Purpose:	End-to-end testing to simulate real-world user behavior
Test File:	`borrow_return.cy.js`
Scope:	Login → Browse Books → Borrow → View Loans → Return

Test Scenario Overview

The test verifies the core functionality that enables a member to:
- Log into the system via UI
- Borrow a book listed in the library catalog
- View their current loans
- Return the borrowed book successfully

Setup Operations

- Logs in or creates an admin user
- Creates a sample test book (*Cypress Book*)
- Creates a standard user (`testuser`) to perform the borrowing actions

Test Execution Steps

1. User logs into the UI
2. Navigates to the Books section
3. Locates the test book and borrows it
4. Receives a UI alert confirming success: `"Book borrowed!"`

5. Navigates to Loans
6. Returns the book
7. Receives return confirmation: `"Book returned successfully"`

```
cy.visit('http://localhost:3000/login');
cy.get('input[placeholder="Username"]').type(username);
cy.get('input[placeholder="Password"]').type(password);
cy.get('button[type="submit"]').click();

cy.contains('Books').click();
cy.contains(testBook.title)
  .parents('.book-card')
  .within(() => {
    cy.contains('Borrow').click();
  });

cy.on('window:alert', (text) => {
  expect(text).to.equal('Book borrowed!');
});

cy.contains('Loans').click();
cy.contains(testBook.title)
  .parents('.loan-card')
  .within(() => {
    cy.contains('Return').click();
  });

cy.get('@alert').should('have.been.calledWith', 'Book returned successfully');
```

Figure 4.5: Sample Code Snippet

Test Results

Successful Verifications:

- Book borrowing correctly updated backend systems
- UI properly reflected borrowing status
- Book return process completed successfully
- Return confirmation alerts displayed correctly
- All critical path scenarios passed verification

Video Demonstration

- Full system test walkthrough available at:
  `https://www.loom.com/share/d5b8f7090e8f4120a4427d91e1acc9d7?sid=f9eaab5e-178f-42b0-b9b4-d05bbf7905aa`

or with clickable link:
blue Watch Cypress System Test Demonstration

4.3 Functional Test Scenarios

Test Case 1: As an Admin, create a new user with valid data

Steps:

- Log in as admin ¿ Navigate to the Manage Users tab
- For the username field, add a username
- For the password field, add a password
- Click on the role dropdown to select the user role
- Hit the Submit button
- Verify that a new user has been added

Test Case 2: As an admin, modify the newly created users

Steps:

- Log in as admin ¿ Navigate to the Manage Users tab
- Click on the "Make Member" button
- Verify that the user has the member role
- Click on the Reset password button
- Verify that the password has been reset from the system automatically
- Click on the Delete button
- Verify that the user has been deleted successfully
- Verify that all the modifications are updated on the UI as expected

Test Case 3: As an Admin, verify user management page displays all users

Steps:
- Log in as admin ¿ Navigate to the Manage Users tab
- Verify that the List of all users is displayed on the user management page

Test Case 4: As an Admin, add a new book with all fields

Steps:
- Log in as admin
- Navigate to Book Management ¿ Add Book ¿ Fill details ¿ Upload files ¿ Save
- Verify that the Book is added and listed

Test Case 5: As an Admin, verify book can be added without optional fields

Steps:
- Log in as admin
- Navigate to Book Management ¿ Add Book ¿ Fill required fields ¿ Save
- Verify that the book is added successfully

Test Case 6: As an admin, verify book list can be edited

Steps:

- Log in as admin ¿ Navigate to Book Management
- Click on the Edit button for any book list
- Update the information and hit the Save button
- Verify that book information has been updated
- Click on the Cancel button
- Verify that the updated information is not saved

Test Case 7: As a member, verify borrow functionality works

Steps:
- Log in as a member and navigate to the Books tab
- Verify that the available book list is displayed
- Click on the borrow button
- Verify that the member can borrow the book successfully

Test Case 8: Verify Member can return a book

Steps:
- Log in as a member and navigate to the Loans tab
- Verify that the borrowed book is visible on the loans tab
- Click on the Return button
- Verify that the user can return the book

Test Case 9: Verify Member can view loan history

Steps:
- Log in as a member and navigate to the Loans tab
- Verify that the book list shows all the details of loan and return for the card
- Click on the filter dropdown
- Verify that the filter returns based on the selection of Returned and Not Returned
- Verify that the loan history can be searched using the Date
- Verify that all filters work according to the selection and return the exact result

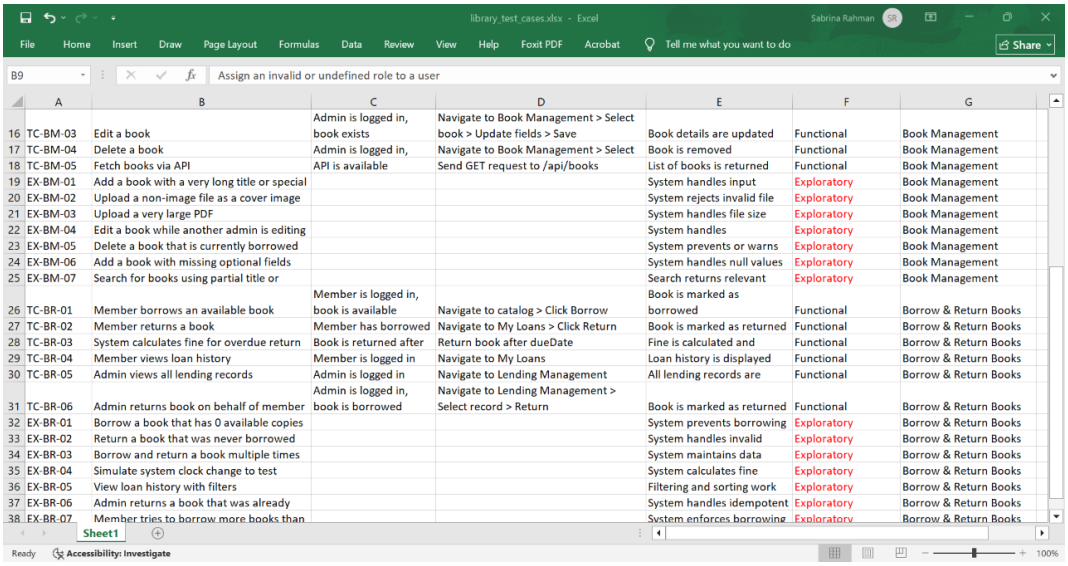
Test Case 10: Verify Admin can view loan history

Steps:

- Log in as admin and navigate to the Loans tab
- Verify that the admin can view the loan book, which has been returned or taken as a loan

	A	B	C	D	E	F	G
1	Test Case ID	Description	Preconditions	Test Steps	Expected Result	Type	Feature
2	TC-URM-01	Create a new user with valid data	Admin is logged in	Navigate to User Management > Add	New user is created and	Functional	User & Role Management
3	TC-URM-02	View all users	Admin is logged in	Navigate to User Management	List of all users is displayed	Functional	User & Role Management
4	TC-URM-03	Update a user's role	Admin is logged in, user exists	Navigate to User Management > Select user > Change role > Update	User role is updated	Functional	User & Role Management
5	TC-URM-04	Delete a user	Admin is logged in, user	Navigate to User Management > Select	User is removed from the	Functional	User & Role Management
6	TC-URM-05	Enforce role-based access	Non-admin user is	Attempt to access User Management	Access is denied with	Functional	User & Role Management
7	EX-URM-01	Attempt to create a user with an existing			System shows duplicate	Exploratory	User & Role Management
8	EX-URM-02	Create a user with missing required fields			System shows validation	Exploratory	User & Role Management
9	EX-URM-03	Assign an invalid or undefined role to a			System handles unknown	Exploratory	User & Role Management
10	EX-URM-04	Try accessing user management via direct URL as a non-admin			Access is denied	Exploratory	User & Role Management
11	EX-URM-05	Update a user's role while they are logged			Role change takes effect	Exploratory	User & Role Management
12	EX-URM-06	Delete a user who has active book loans			System prevents deletion or	Exploratory	User & Role Management
13	EX-URM-07	View user list with a large number of users			Pagination or performance	Exploratory	User & Role Management
14	TC-BM-01	Add a new book with all fields	Admin is logged in	Navigate to Book Management > Add Book > Fill details > Upload files > Save	Book is added and listed	Functional	Book Management
15	TC-BM-02	Add a book with optional fields left blank	Admin is logged in	Repeat TC-BM-01 but leave image/file	Book is added successfully	Functional	Book Management
16	TC-BM-03	Edit a book	Admin is logged in, book exists	Navigate to Book Management > Select book > Update fields > Save	Book details are updated	Functional	Book Management
17	TC-BM-04	Delete a book	Admin is logged in,	Navigate to Book Management > Select	Book is removed	Functional	Book Management

Figure 4.6: Test Case



	A	B	C	D	E	F	G
16	TC-BM-03	Edit a book	Admin is logged in, book exists	Navigate to Book Management > Select book > Update fields > Save	Book details are updated	Functional	Book Management
17	TC-BM-04	Delete a book	Admin is logged in,	Navigate to Book Management > Select	Book is removed	Functional	Book Management
18	TC-BM-05	Fetch books via API	API is available	Send GET request to /api/books	List of books is returned	Functional	Book Management
19	EX-BM-01	Add a book with a very long title or special			System handles input	Exploratory	Book Management
20	EX-BM-02	Upload a non-image file as a cover image			System rejects invalid file	Exploratory	Book Management
21	EX-BM-03	Upload a very large PDF			System handles file size	Exploratory	Book Management
22	EX-BM-04	Edit a book while another admin is editing			System handles	Exploratory	Book Management
23	EX-BM-05	Delete a book that is currently borrowed			System prevents or warns	Exploratory	Book Management
24	EX-BM-06	Add a book with missing optional fields			System handles null values	Exploratory	Book Management
25	EX-BM-07	Search for books using partial title or			Search returns relevant	Exploratory	Book Management
26	TC-BR-01	Member borrows an available book	Member is logged in, book is available	Navigate to catalog > Click Borrow	Book is marked as borrowed	Functional	Borrow & Return Books
27	TC-BR-02	Member returns a book	Member has borrowed	Navigate to My Loans > Click Return	Book is marked as returned	Functional	Borrow & Return Books
28	TC-BR-03	System calculates fine for overdue return	Book is returned after	Return book after dueDate	Fine is calculated and	Functional	Borrow & Return Books
29	TC-BR-04	Member views loan history	Member is logged in	Navigate to My Loans	Loan history is displayed	Functional	Borrow & Return Books
30	TC-BR-05	Admin views all lending records	Admin is logged in	Navigate to Lending Management	All lending records are	Functional	Borrow & Return Books
31	TC-BR-06	Admin returns book on behalf of member	Admin is logged in, book is borrowed	Navigate to Lending Management > Select record > Return	Book is marked as returned	Functional	Borrow & Return Books
32	EX-BR-01	Borrow a book that has 0 available copies			System prevents borrowing	Exploratory	Borrow & Return Books
33	EX-BR-02	Return a book that was never borrowed			System handles invalid	Exploratory	Borrow & Return Books
34	EX-BR-03	Borrow and return a book multiple times			System maintains data	Exploratory	Borrow & Return Books
35	EX-BR-04	Simulate system clock change to test			System calculates fine	Exploratory	Borrow & Return Books
36	EX-BR-05	View loan history with filters			Filtering and sorting work	Exploratory	Borrow & Return Books
37	EX-BR-06	Admin returns a book that was already			System handles idempotent	Exploratory	Borrow & Return Books
38	EX-BR-07	Member tries to borrow more books than			System enforces borrowing	Exploratory	Borrow & Return Books

Figure 4.7: Test Case

4.4 Exploratory Testing

Overview

Exploratory testing was conducted to simulate real-world user interactions beyond scripted test cases. This approach allowed for the discovery of potential edge cases and usability issues by interacting with the Library Management System freely and contextually.

Approach

Two structured techniques were used for exploratory testing:

- **Business District Tour**
  - Explored core features that receive the most user traffic
  - **Focus:** Authentication, Book Listings, Borrow/Return Flow, User Dashboard
  - **Goal:** Validate the most "used" user flows under real-life usage simulation
- **Guidebook Tour**
  - Followed predefined documentation or feature descriptions
  - **Focus:** Feature instructions in user documentation were followed and verified
  - **Goal:** Identify mismatches between documentation and implementation

Testing Areas & Observations

- **Mission:** Explore the robustness of the core loan process of a library management system
- **Charter:**
  - **Login System**
    - Log in as an existing user with username and password
    - Log in as existing user with Google account
    - Log in as existing user with Facebook account
    - Enter incorrect username and password to verify the validation message

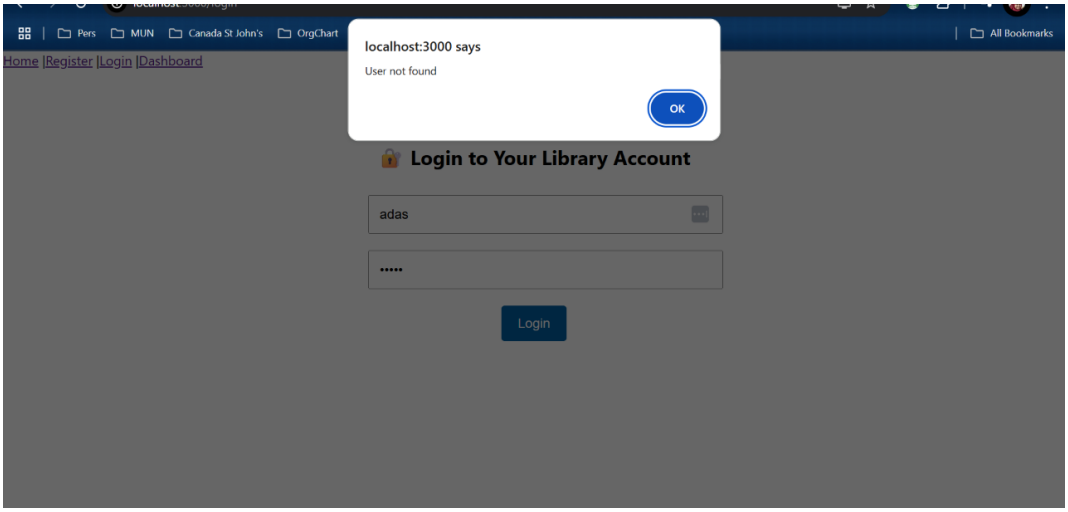


Figure 4.8: Login failed message

Loan System Testing

- **Loan System**
  - Borrow a book that has 0 available copies
  - Return a book that was never borrowed
  - Borrow and return a book multiple times quickly
  - Admin returns a book that was already returned
  - Member tries to borrow more books than allowed

Key Exploratory Testing Tours for User Login

1. **Business District Tour**
   - Enter valid username and password; verify successful login
   - Try an invalid password or username; check that:
     - Error messages appear
     - No access is granted
2. **Guidebook Tour**
   - Welcome to the Login Portal
   - See input fields for Username or Email and Password

- Optionally find buttons for Login
- Users enter their login credentials:
  - Username/Email: Must be valid and registered
  - Password: Should match the stored credentials
  - Typo in email or username
  - CAPS LOCK on when typing password
- **Authentication Process**
  - Once users click "Login":
    - The app verifies credentials with the server
    - If valid: creates secure session (with tokens/cookies)
    - If invalid: shows friendly error message ("Incorrect password. Try again!")
  - Use HTTPS to secure communication
  - Implement rate-limiting to prevent brute-force attacks
- **Successful Login**
  - Upon successful authentication:
    - Users are redirected to their dashboard or homepage

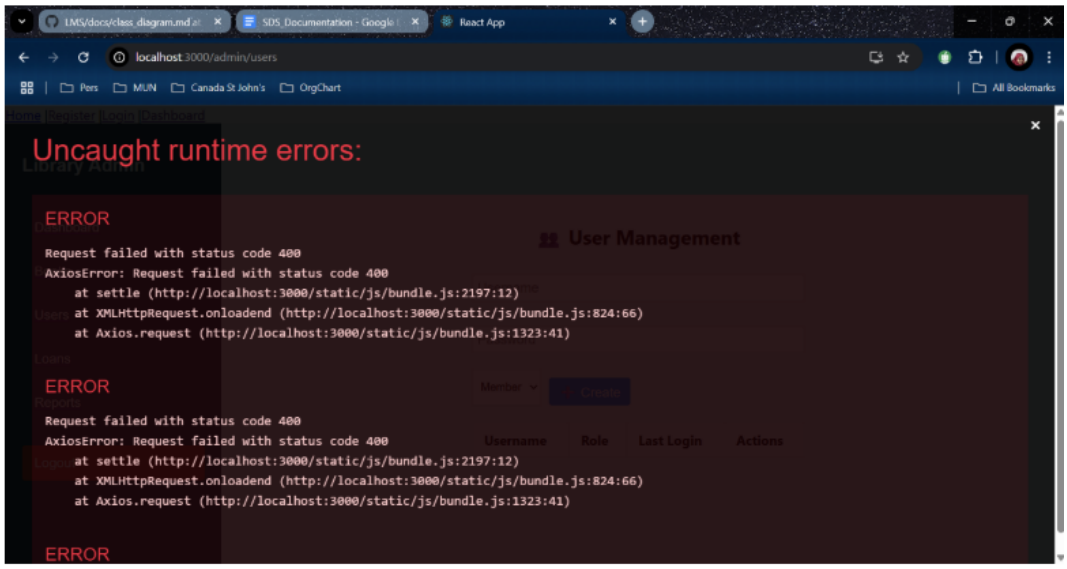


Figure 4.9: Bug report 1 : Session timeout out error occurred while attempting to log in

Chapter 5

Limitations

5.1 Project-Specific Limitations

1. **Limited Scalability Testing**
   - The system was tested in a controlled environment with limited data and users
   - Load testing (e.g., with concurrent users or large datasets) was not conducted
   - As a result, performance under real-world traffic or institutional-scale deployment is unverified

2. **Authentication Security Simplification**
   - Basic JWT-based authentication was implemented
   - Advanced security practices like rate limiting, brute-force protection, and OAuth2 were not integrated
   - This may limit the system's resilience in a production environment

3. **No Offline Functionality**
   - The system assumes continuous server and network availability
   - Users cannot access any cached or offline features

4. **Lack of Automated Reservation Handling Logic**
   - Although the reservation system is outlined in the design, its full backend implementation (e.g., queue management, auto-notifications) remains

pending

5.2 Research and Documentation Limitations

1. **Limited Access to Paid Journals**
   - Several academic resources on formal verification (e.g., Z notation, Alloy modeling) and advanced test case generation (e.g., model-based testing) were behind paywalls or required institutional access
   - This may have affected the depth of literature-based validation
   - **Reference:** [4]
2. **Tooling Constraints**
   - UML diagrams were created using PlantUML instead of advanced tools like Enterprise Architect
   - End-to-end tests were done manually and with Cypress, but load/performance testing tools (e.g., JMeter, Gatling) were not used due to time constraints
   - **Reference:** [5]
3. **Human Factors in Exploratory Testing**
   - Exploratory testing depends heavily on tester experience and biases
   - Not all possible flows or rare edge cases may have been explored
   - There is a risk of inconsistent coverage compared to structured test cases
   - **Reference:** [6]

Chapter 6

Conclusion

The Library Management System (LMS) project demonstrates a practical implementation of software verification and validation (V&V) methodologies through a combination of automated unit tests, system-level tests, and exploratory functional testing. By applying tools such as Supertest for backend API testing and Cypress for full-stack end-to-end testing, the project ensured that both individual components and the integrated system behaved as expected.

Exploratory testing further validated the usability and responsiveness of the application in real-time scenarios, capturing interactions that scripted tests may not cover. This multi-layered approach aligns with industry best practices for maintaining software reliability, usability, and correctness.

Despite limitations such as the absence of scalability testing, limited use of advanced security mechanisms, and constrained access to formal verification tools or academic research, the LMS system satisfies its intended functional requirements in a controlled environment. The integration of verification practices at various stages of development strengthens the system's trustworthiness and highlights the importance of V&V in real-world software projects.

This report reinforces the critical role of structured and exploratory testing in software development and provides a foundation for further enhancements—such as stress testing, advanced formal specification, and integration of CI/CD pipelines to support ongoing validation.

Bibliography

[1] *Ieee standard for system and software verification and validation*, Institute of Electrical and Electronics Engineers, 2016.

[2] I. Sommerville, *Software Engineering*, 10th ed. Pearson, 2016, Contains chapters on performance testing and non-functional requirements.

[3] L. Crispin and J. Gregory, *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley, 2009.

[4] A. Bertolino, "Software testing research: Achievements, challenges, dreams," in *Future of Software Engineering*, IEEE, 2007, pp. 57–72. DOI: 10.1109/FOSE.2007.25. [Online]. Available: https://doi.org/10.1109/FOSE.2007.25.

[5] M. Utting, A. Pretschner, and B. Legeard, "A taxonomy of model-based testing approaches," *Software Testing, Verification and Reliability*, vol. 22, no. 5, pp. 297–312, 2012.

[6] J. Itkonen, M. V. Mäntylä, and C. Lassenius, "Defect detection efficiency: Test case based vs. exploratory testing," *Empirical Software Engineering*, vol. 11, no. 4, pp. 467–489, 2007.