An Academic Report on
Design by Contract in a Layered Library Management System

Submitted To:

Dr. Raja Manzar Abbas

Submitted By:

Anil Manyam (Student Id:202483873)

Rama Satya Narayana (Student Id: 202480704)

TABLE OF CONTENTS:

1. ABSTRACT

This report details the design and implementation of a modern Library Management System (LMS) built using a layered architectural pattern and guided by Design by Contract (DbC) principles. The system addresses key challenges often found in traditional LMS solutions, such as inconsistencies, poor maintainability, and lack of formal verification. Users can register, search for books, borrow and return them, leave feedback, and manage fines, while librarians can handle book catalog management, monitor loans and borrow rules. The architecture includes a React.js frontend, a Python Flask microservice backend, and a cloud-hosted MongoDB Atlas database. To ensure correctness and support future scalability, the system applies formal specification techniques for defining critical operations and uses UML diagrams to validate the system's structure and behavior. Altogether, the approach enhances software reliability, simplifies debugging, and lays a solid foundation for future development.

Keywords: *Library Management System (LMS), Design by Contract (DbC), MongoDB Atlas, Flask API, React.js, Book Loan Management, User Feedback System, Borrowing Constraints, Fine Calculation Logic, User Authentication, UML Design*.

2. INTRODUCTION

Library Management Systems (LMS) have become essential tools in academic and public libraries, helping manage critical operations such as lending and returning books, user registration, fine calculation, and maintaining borrowing histories. As library usage grows and user expectations shift toward seamless digital interactions, modern LMS solutions must also handle complex policies like borrowing limits, varied loan durations, and role-based access. While these needs may seem straightforward, implementing them consistently and accurately presents significant technical challenges.

Many traditional LMS implementations rely on informal development approaches, where business rules are scattered and enforced inconsistently across different modules. This often leads to policy violations, logic errors, and maintenance difficulties particularly in large systems handled by multiple developers. Consequences such as incorrect fine calculations, unauthorized borrowing, or data integrity issues can seriously impact the reliability of the system, which is unacceptable in library environments that depend on accuracy and trust.

To address these limitations, this project proposes a modern LMS built around Design by Contract (DbC) principles and a layered software architecture. DbC ensures correctness by defining precise preconditions, postconditions, and invariants for all critical operations, making the system easier to test, maintain, and extend. The layered architecture separates the presentation, service, and data layers, which enhances modularity and simplifies future development.

The system is implemented using a modern technology stack: React.js for an intuitive frontend, Flask for a lightweight backend API, and MongoDB Atlas as a cloud-hosted NoSQL database for scalable and persistent data storage. To reinforce software correctness and clarity, formal specification techniques using Z notation are employed to rigorously define key operations. UML diagrams are also used to visualize system architecture and workflows. Altogether, this approach provides a reliable, scalable, and future-ready LMS that addresses the shortcomings of traditional systems and supports long-term adaptability.

3. PROBLEM STATEMENT

Traditional Library Management Systems (LMS) often struggle with enforcing complex borrowing rules, maintaining data consistency, and ensuring long-term maintainability. Users may have varying loan limits, due dates, and fine structures based on membership types, all of which must be accurately enforced. However, in many systems, business rules are implemented informally or inconsistently, leading to issues such as users exceeding borrowing limits, incorrect fine calculations, or double loaning of books. These problems are compounded by weak validation mechanisms that fail to enforce rules uniformly across components, allowing for bypasses, policy violations, or data integrity issues particularly in areas like book returns or user feedback handling.

The root cause of these issues lies in the lack of structured design and formal behavior contracts in traditional LMS architectures. Without clearly defined preconditions, postconditions, and invariants, software components operate with uncertain guarantees, making the system difficult to test, debug, or evolve. As a result, maintaining and scaling such systems becomes increasingly error-prone over time. To address these challenges, there is a clear need for a more systematic approach one that promotes correctness, clarity, and maintainability through formal methods and well-defined contracts, while still supporting flexibility for future enhancements.

4. LITERATURE REVIEW

Design by Contract (DbC) has emerged as a powerful methodology in software engineering, offering formal guarantees about program behavior through well-defined contracts. This literature review examines key research contributions that explore contract-based development, practical implementation techniques, and alternative architectural models for Library Management Systems (LMS). The review highlights the relevance of DbC in enhancing modularity, correctness, and maintainability, while also contrasting it with other formal and semantic approaches to system design. The following three studies provide a strong foundation for understanding how this project's architectural and methodological choices align with current research trends.

4.1 A Systematic Mapping Study on Contract-Based Software Design (Okumuş et al., 2025)

Okumuş et al. conducted an extensive systematic mapping study, analyzing 288 research papers on contract-based software design. Their work focused on the use of preconditions, postconditions, and invariants to improve software correctness and modular boundaries. The study revealed that formal contracts play a significant role in reducing defect rates and improving code clarity, particularly in enterprise and embedded systems. However, the authors identified a notable research gap in applying DbC principles to modern, cloud-native systems that use NoSQL databases, precisely the architectural context of this LMS project, which integrates Flask with MongoDB Atlas [1].

Key insights from their review emphasize that contract-based development enhances component reusability and promotes safe interchangeability between modules. Additionally, the presence of tool support such as icontract or Eiffel's native support was strongly associated with fewer runtime errors and better enforcement of logic constraints [1]. The combination of DbC with automated verification tools was found to bridge the gap between informal design and formally correct implementation. This study validates the decision to apply icontract in the LMS backend, confirming that such tools not only help enforce business logic but also position the system for future scalability, including microservice migration, without compromising correctness [1].

4.2 Applying Design by Contract: Insights from Engineering Practice (Meyer, 1992)

Bertrand Meyer's seminal work on Design by Contract laid the foundation for understanding contracts as both formal specifications and executable components in software development. In his exploration of Eiffel a programming language built around DbC Meyer demonstrates how preconditions, postconditions, and invariants act as self-verifying contracts between software components. These contracts serve as executable documentation, making assumptions and guarantees explicit, thereby acting as both unit tests and detailed design specifications [2].

Meyer further discusses how Eiffel's environment enforces contracts at runtime, enabling early detection of violations and improving software reliability. While contracts may be disabled in production environments for performance optimization, their presence during development and testing phases proves critical in catching defects early [2]. A key takeaway from Meyer's work is the ability of contracts to clearly define the source of errors in distributed or layered systems, improving traceability and debugging. These principles are directly reflected in this LMS project, where backend service methods are decorated using icontract decorators such as @require and @ensure to enforce constraints like "a user cannot borrow more than three books." Meyer's methodology strongly supports the structured, contract-enforced backend design adopted here [2].

4.3 Ontology-Based Library Management: Building a Semantic LMS (Mohd et al., 2020)

While many systems rely on procedural or contract-based logic, Mohd et al. propose an entirely different approach to library management using semantic web technologies and ontologies. Their work introduces LMSO, an ontology-driven architecture modeled using OWL (Web Ontology Language), designed to represent entities such as books, users, and loans in a formal, machine-readable format. This model organizes domain entities into modular, class-based structures with logical relationships and constraints. For example, a loan must reference both a book and a user, and inheritance is used to generalize shared properties among entities [3].

One of the key strengths of this ontology-based approach is reasoning. Through formal axioms and inference engines like HermiT, the system can automatically detect inconsistencies, such as overdue items or invalid user actions. The use of semantic standards like RDF and OWL also makes the system highly interoperable with external catalogs and linked data sources. Mohd et al. validated their model by implementing over 200 instances and running consistency checks using

Protégé. While this semantic approach does not follow the Design by Contract paradigm, it offers a powerful alternative for ensuring correctness through logic-based reasoning. It also presents opportunities for integration with broader knowledge systems, making it a valuable contrast to the DbC-based architecture implemented in this LMS project [3].

5. SYSTEM ARCHITECTURE

This Library Management System (LMS) is designed using a three-tier layered architecture, separating the system into the presentation, service, and data layers. This separation improves clarity, maintainability, and flexibility for future updates.

5.1 Layered Architectural Model:

Presentation Layer (React.js): Provides a responsive, component-based user interface. It handles user interactions like searching for books or initiating loans.

Service Layer (Flask): Acts as the middleware, exposing a REST API. It applies business logic, validates inputs using Design by Contract (icontract), and handles communication between the frontend and database.

Data Layer (MongoDB Atlas): A cloud-based NoSQL database that stores structured collections for users, books, loans, loan history, and feedback. Its schema-less design allows easy data model evolution.

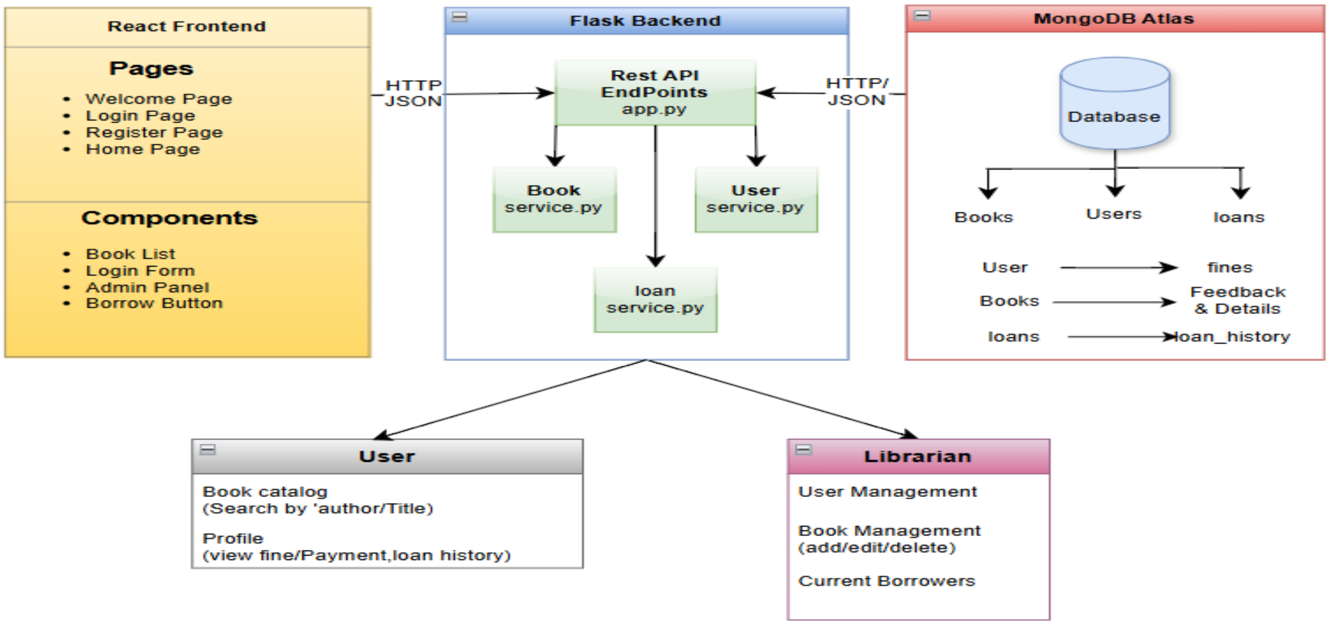
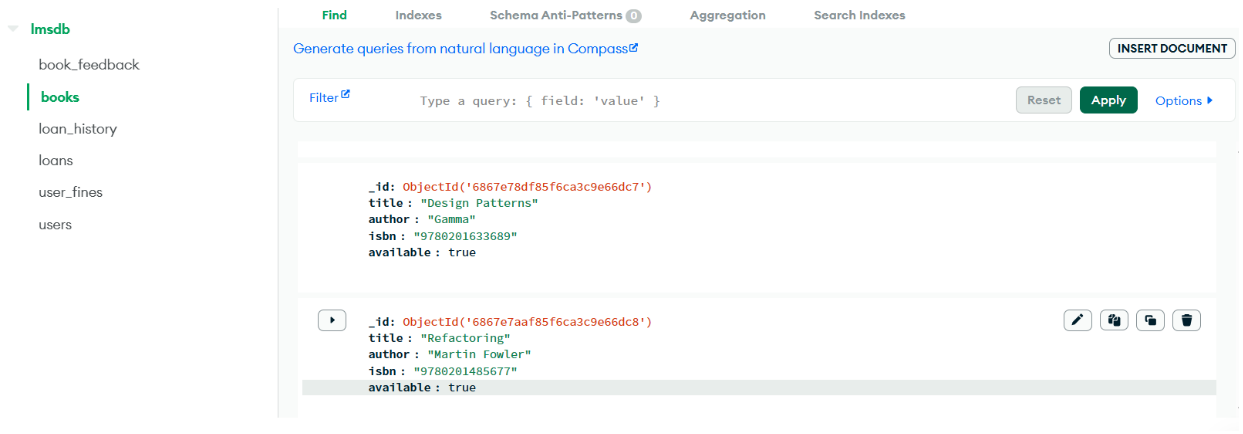


Fig-1: Layered Architecture Diagram of LMS

5.2 Data Persistence Strategy: MongoDB Atlas offers flexibility through its document-oriented storage and global accessibility. Key collections include users, books, loans, loan_history, and book_feedback. Indexes on fields like username and book_id ensure fast data retrieval, while replica sets enhance fault tolerance and availability.



```
_id: ObjectId('6867e78df85f6ca3c9e66dc7')
title : "Design Patterns"
author : "Gamma"
isbn : "9780201633689"
available : true

_id: ObjectId('6867e7aaf85f6ca3c9e66dc8')
title : "Refactoring"
author : "Martin Fowler"
isbn : "9780201485677"
available : true
```

Fig-2: The above figure showcases all the collections in our database.

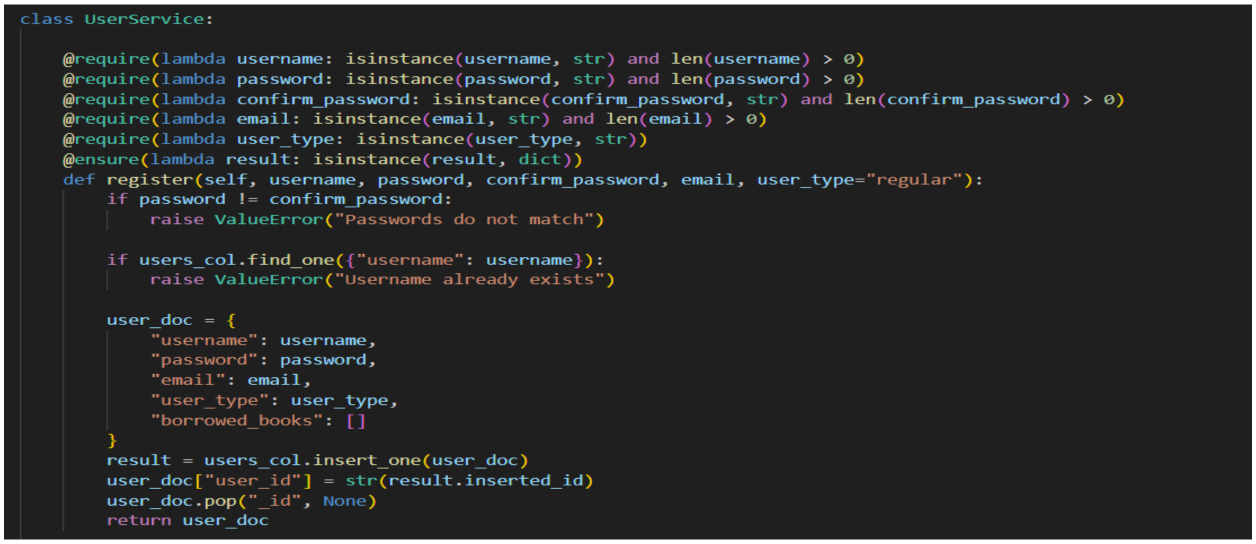
5.3 Communication Flows: The React frontend interacts with Flask through RESTful APIs and retrieve the necessary information from the database based on user requests and functions.

6. METHODOLOGY & SYSTEM DESIGN

6.1 Requirements Analysis: The requirements for this Library Management System were based on traditional library workflows and typical user interactions. Core features include user registration, book search, borrowing, returns, fine management, and librarian functions. To guide development, the requirements were divided into two categories: functional requirements (e.g., borrowing, returning, viewing history, managing books, and submitting feedback) and non-functional requirements, which focus on maintainability, scalability, security, cloud availability, and formal correctness. This structure ensures the system remains reliable and adaptable over time.

6.2 Design by Contract Implementation: In this project, Design by Contract (DbC) is implemented using the icontract library in Python. This approach ensures that each critical function within the system such as user registration, book borrowing, and returns strictly follows predefined rules. These rules are enforced using decorators that define both preconditions (what must be true before the function runs) and postconditions (what must be true after the function completes).

For instance, in the register function, preconditions ensure that the provided username, password are non-empty strings, and postconditions confirm that the function returns a dictionary representing a valid user record. If any of these conditions are not met such as if the input is invalid or the function fails to return the expected result icontract raises an exception immediately. This stops the error from spreading further into the system and makes debugging much easier. Overall, this approach adds a layer of reliability by making the system's logic self-validating and easier to maintain. Here is the example:



```python
class UserService:

    @require(lambda username: isinstance(username, str) and len(username) > 0)
    @require(lambda password: isinstance(password, str) and len(password) > 0)
    @require(lambda confirm_password: isinstance(confirm_password, str) and len(confirm_password) > 0)
    @require(lambda email: isinstance(email, str) and len(email) > 0)
    @require(lambda user_type: isinstance(user_type, str))
    @ensure(lambda result: isinstance(result, dict))
    def register(self, username, password, confirm_password, email, user_type="regular"):
        if password != confirm_password:
            raise ValueError("Passwords do not match")

        if users_col.find_one({"username": username}):
            raise ValueError("Username already exists")

        user_doc = {
            "username": username,
            "password": password,
            "email": email,
            "user_type": user_type,
            "borrowed_books": []
        }
        result = users_col.insert_one(user_doc)
        user_doc["user_id"] = str(result.inserted_id)
        user_doc.pop("_id", None)
        return user_doc
```

Fig-3: Registration Function implemented by using DBC through icontracts in python

6.3 System Modeling with UML: To effectively visualize and plan the system's structure and behavior, Unified Modeling Language (UML) was used throughout the design phase. Four key diagrams were created to support different aspects of the system. A class diagram illustrates the relationships between core entities such as User, Book, Loan, and Feedback, providing a clear view of the system's data model. A sequence diagram captures the flow of operations over time, particularly focusing on dynamic processes like borrowing and returning books. A use case diagram maps out how various users such as members and librarians interact with different system functionalities. Additionally, an activity diagram which showcases the flow of activities within the system.

Class Diagram:

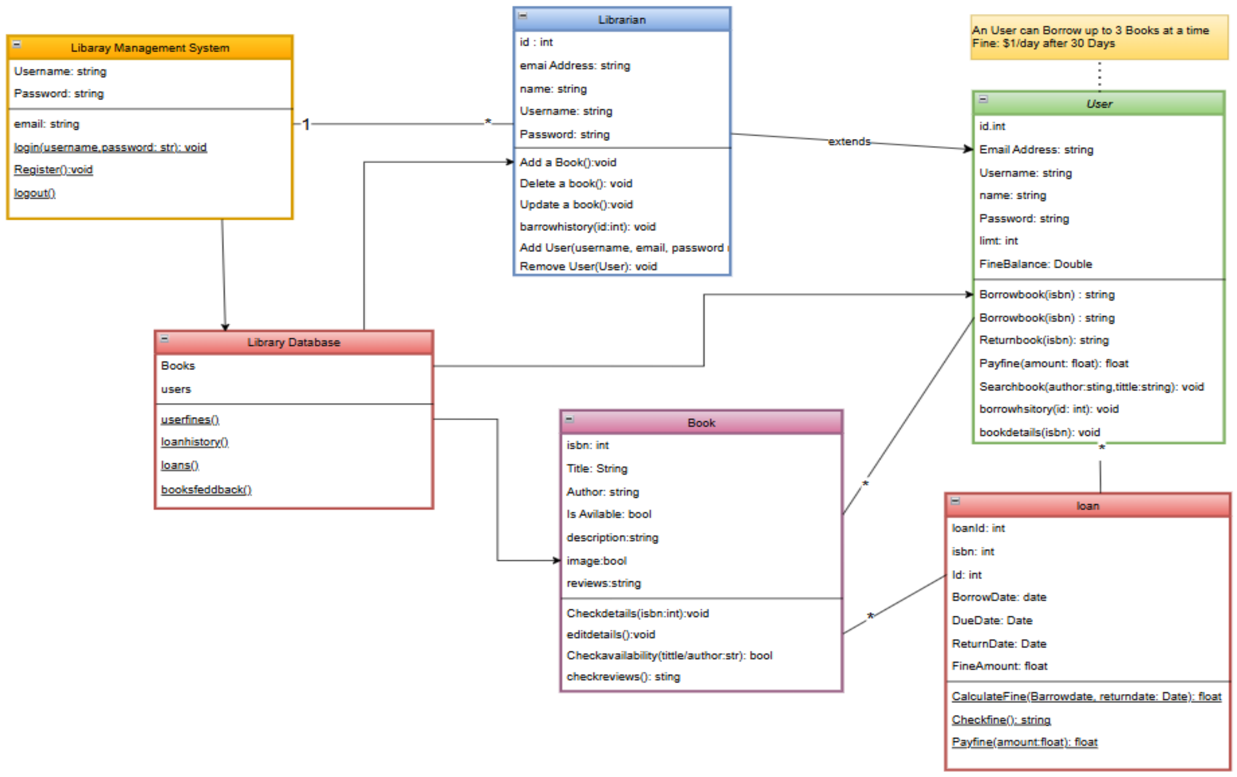


Fig-4: Class Diagram of overall Library Management System

The class diagram above presents a comprehensive object-oriented design of the Library Management System, capturing the entities, behaviors, and relationships that define the system's architecture. The system comprises several core components: Library Management System, Librarian, User, Book, Loan, and a centralized Library Database. At the interface level, the Library Management System provides methods for users to login, register, and logout. The system differentiates users into two roles: User and Librarian, where the Librarian class inherits from User, enabling a hierarchical role-based access model.

The User class encapsulates all typical functionalities available to regular patrons of the library. It includes attributes such as id, email address, username, password, a limit to restrict borrowing to a maximum of 3 books at a time, and a fineBalance to track dues. The behaviors defined in this class allow a user to borrow and return books, pay fines, search for books using keywords, check borrowing history, and view feedback or reviews related to a specific book.

On the other hand, the Librarian class, which extends the User class, is equipped with administrative methods. These include adding new books, deleting or updating existing book records, viewing any user's borrowing history, and managing users specifically, creating new users and removing them from the system. These functionalities make the librarian the central authority responsible for catalog and user management. Books are represented by the Book class, which includes attributes such as isbn, title, author, description, availability status, and image or reviews. This class provides methods to check availability by ISBN or author/title, retrieve book details, edit book metadata, and view reviews. The Loan class models individual borrowing transactions, associating users with borrowed books and tracking relevant data like borrow date, due date, return date, and any applicable fine. It includes methods for calculating fines based on the return delay and for processing fine payments.

At the backend, all data is centrally managed by the Library Database class, which contains collections of books and users, and provides interfaces for operations such as userfines(), loanhistory(), loans(), and booksfeedback(). These backend utilities ensure the frontend can retrieve and manage dynamic content related to user activity, book availability, and transactional records. Additionally, the diagram highlights important constraints and rules. For example, users are limited to 3 books at any one time, and late returns incur a $1 fine per day after a 30-day borrowing period. These constraints are enforced through the business logic within the loan calculation methods. The association and inheritance relationships between classes are well defined such as one-to-many relationships between User and Loan, Book and Loan, and the specialization of Librarian from User. This structured design enhances system scalability, security, and modularity, making it suitable for real-world deployment in a small to medium-sized library.

Use case Diagrams:

The use case diagrams provide a clear understanding of the functional interactions between various actors users, librarians/admins, and the database and the Library Management System. In the below, the system identifies three main actors: the User, the Librarian/Admin, and the Database. Each of these actors performs a set of specific operations that define their roles within the system. The User interacts with the system by performing multiple essential tasks. These include logging in for authentication, searching for books, viewing book details and reviews, borrowing and returning books, paying fines, checking their borrowing history, providing feedback, and registering as a new user. The feedback process includes an additional step filling up feedback which is marked with an <<include>> relationship to show it is a required sub-function of providing feedback. The user functionalities are fundamental to ensuring a seamless reading and borrowing experience for library members and are designed to be intuitive and self-service oriented.

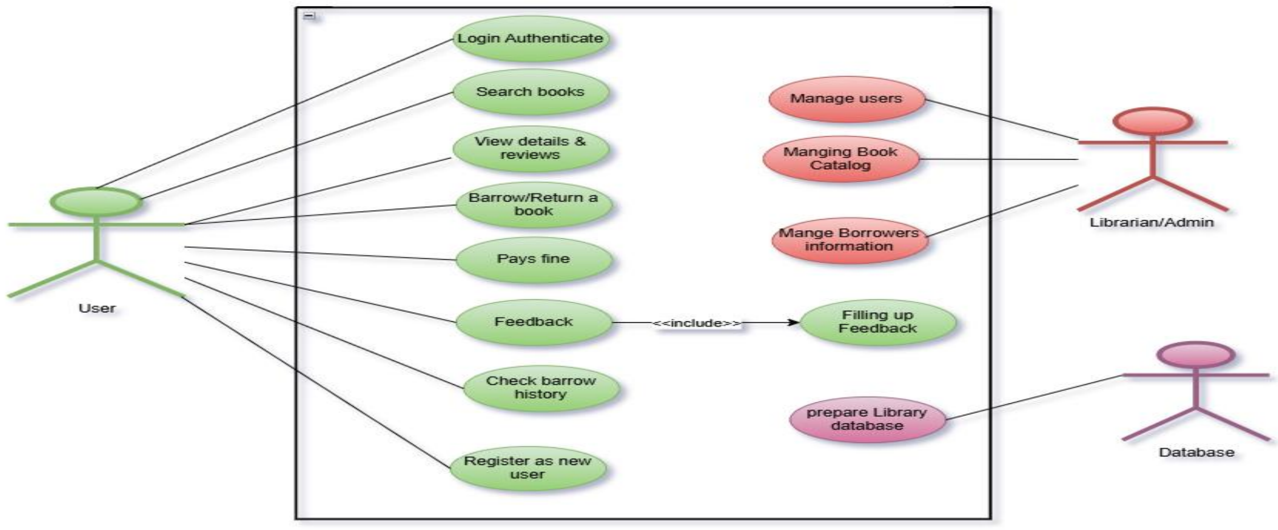


Fig-5: Use case diagram with all 3 actors of LMS

On the other hand, the Librarian/Admin is responsible for overseeing the backend operations of the system. Their tasks include managing user accounts, handling the book catalog, and maintaining borrower information. These operations support the smooth functioning of the system by ensuring data consistency, keeping the catalog up-to-date, and monitoring user activities. The Database is depicted as a system actor responsible for storing and managing all relevant data. It supports operations such as preparing the library database, which underpins all transactional and query-based functionalities in the system.

The next use case diagram focuses specifically on Book Catalog Management, which is an extension of the librarian's responsibilities from the first diagram. In this extended view, the Librarian performs the overarching use case of Managing Books. This includes Book Catalog Updates, which further breaks down into two more specific cases: Deleting Books and Checking and Saving New Book Information. The use case Deleting Books is linked with an <<include>> relationship, indicating it is a necessary part of the update process, while Checking and Saving

New Book Information is shown as an <<extends>> relationship, signifying that it is an optional or conditionally triggered task based on whether the librarian is adding or editing a book's data.

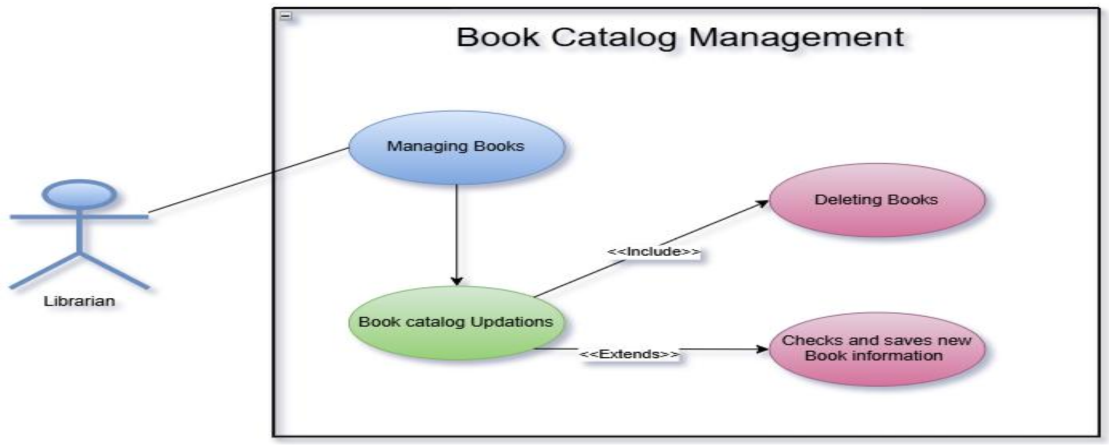


Fig-6: Book Catalog Management Use case diagram from librarian Use cases

Similarly, the there are other use case diagrams for user management, Borrowers information management from librarian side and Book search from user side.

Sequence Diagrams:

The first sequence diagram in Fig-7 illustrates the key interactions between the Librarian (Admin), the Library Management System, and the Database. The process begins when the librarian logs in, and the system verifies their credentials. Upon successful authentication, the admin dashboard is displayed, providing access to both book and user management features. In book management, the librarian can add a new book by submitting details such as the title, author, description, and cover image. The system prepares the entry and stores it in the database. Books can also be deleted or edited, with corresponding updates or removals made in the backend.

Under user management, the librarian can create new user accounts by entering relevant information like name, email, username, and password. These records are saved in the user database. The librarian also has the ability to delete users when needed. Additionally, the librarian can request borrower reports, which the system retrieves from the database. These reports provide insights into current loans, including book titles and associated user details. Overall, the sequence diagram captures the flow of administrative tasks and highlights how the system supports efficient, structured management of library resources and users.

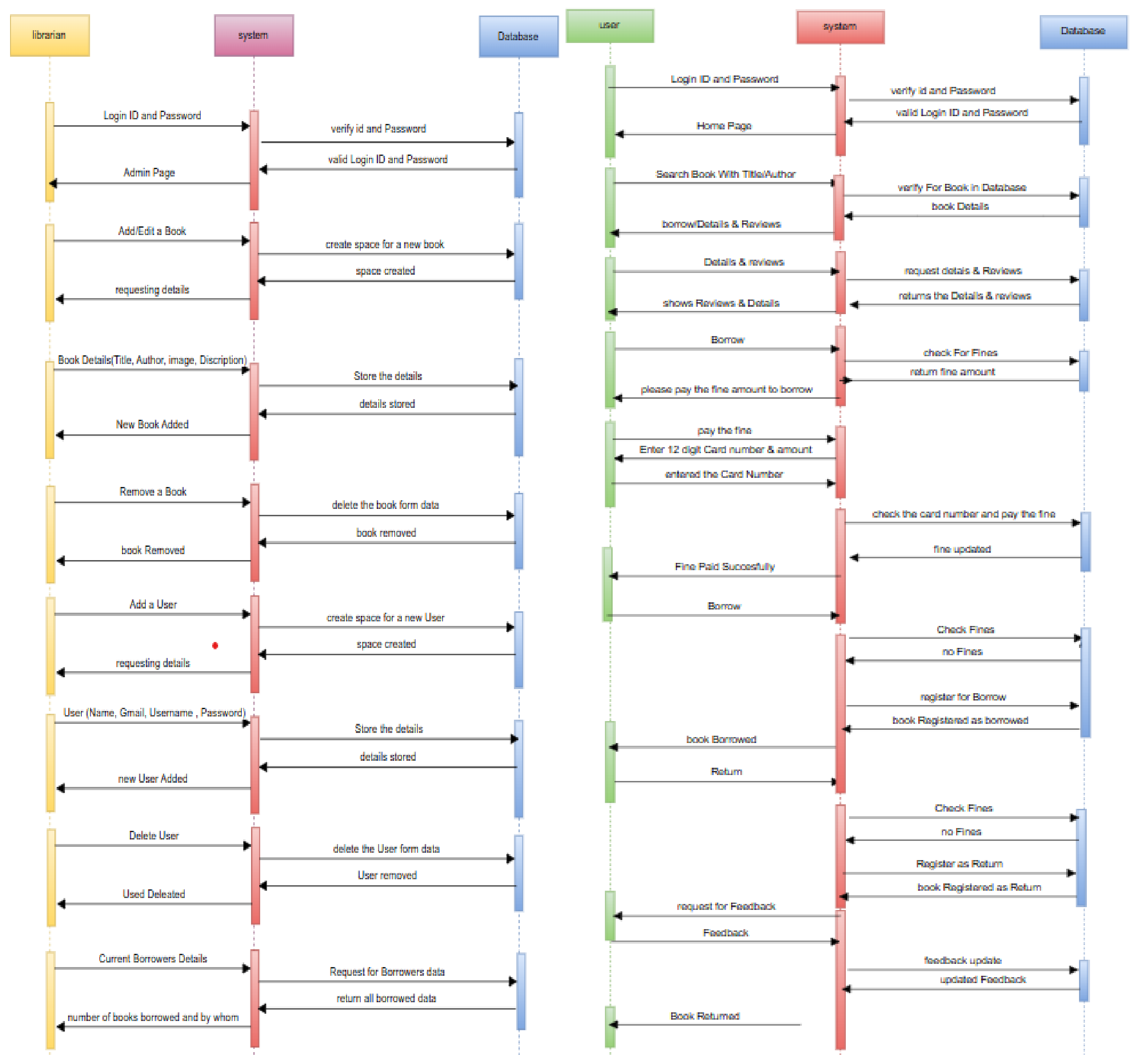


Fig-7: Sequence Diagrams for both librarian and user system interactions

This second sequence diagram in figure 7 illustrates the interaction between a User, the Library Management System, and the Database during the complete book borrowing and feedback process. It begins with the user logging in using valid credentials, which the system verifies against stored records. Once authenticated, the user accesses the homepage and searches for a book by title or author. The system checks the database for the book's availability, retrieves the relevant details and reviews, and displays them to the user. If the user decides to borrow the book, the system first checks for any outstanding fines. If fines exist, the user is prompted to make a payment by entering card details. Upon successful payment, the fine record is updated in the database, and the user is allowed to proceed with borrowing. The system then marks the book as borrowed.

Later, when the user returns the book, the system checks for any new fines related to overdue returns. If no fines are due, the return is registered, and the book's status is updated in the database. The user is also prompted to provide feedback, which is then stored alongside the book's data. The process concludes with confirmation of both the return and feedback submission. This sequence effectively captures the user's full interaction cycle with the system covering search, borrow, fine handling, return, and feedback ensuring a smooth and rule-compliant user experience.

Activity Diagrams:

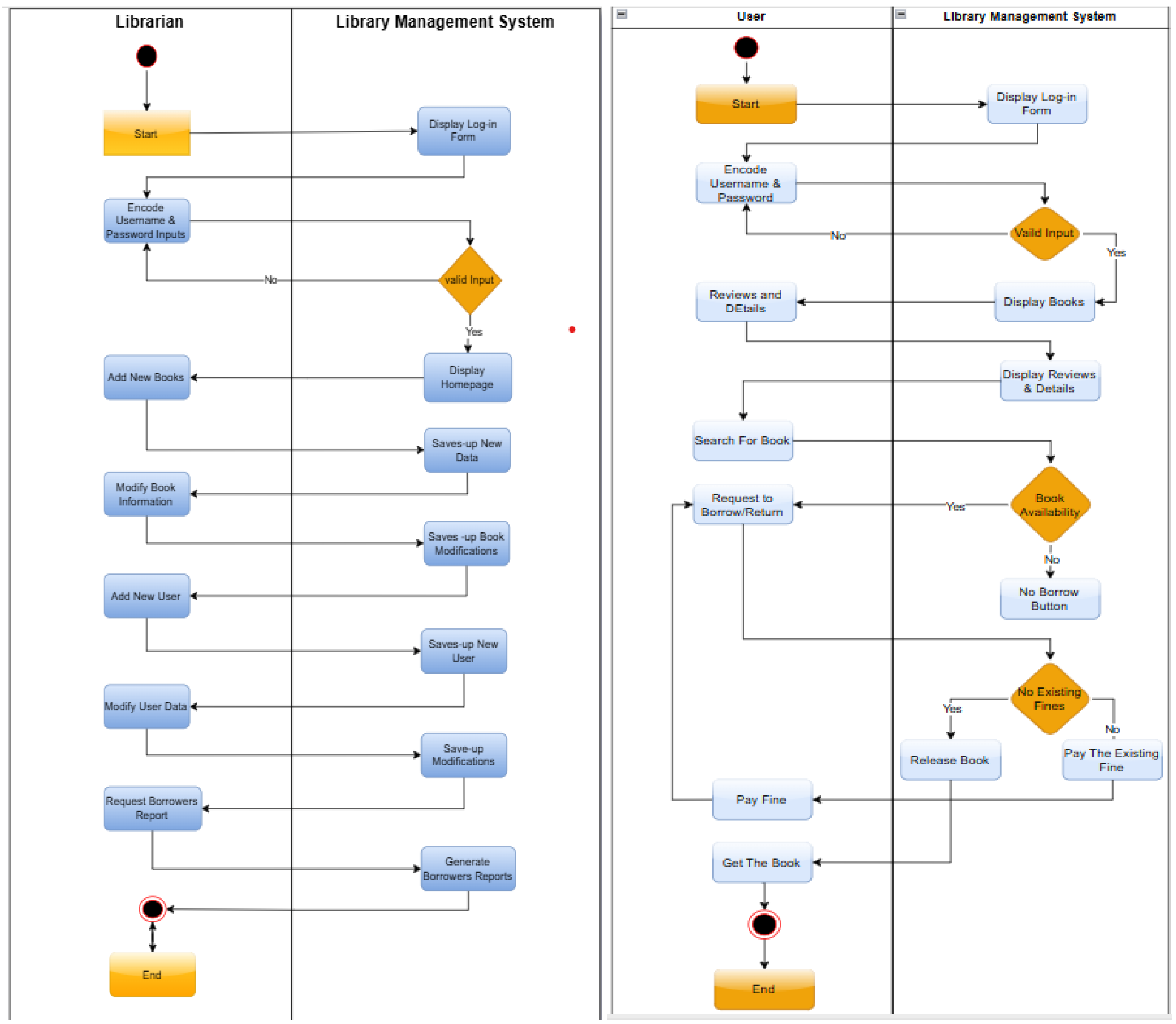


Fig-8: Activity Diagrams - Librarian & User Workflow in the Library Management System

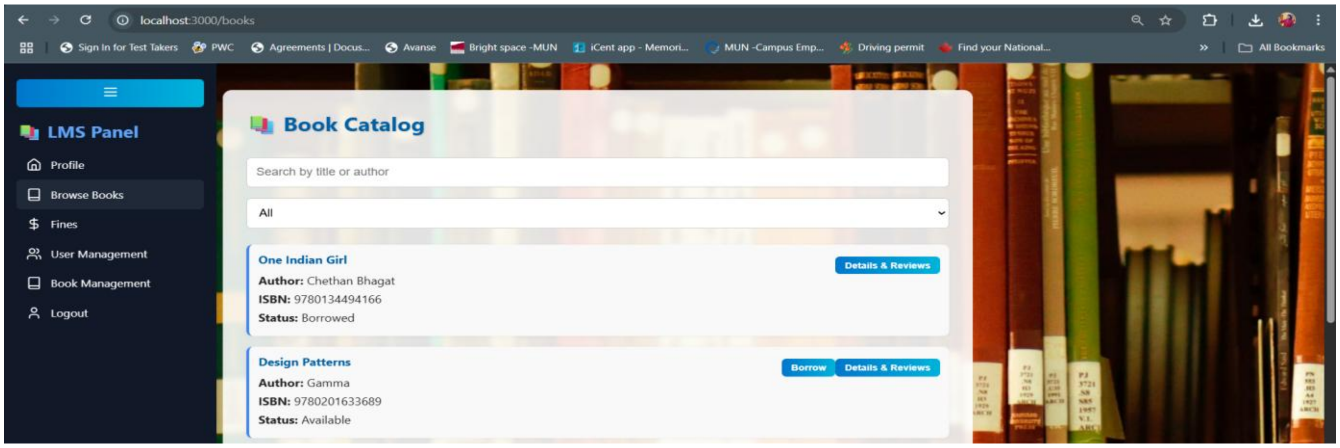
This first activity diagram outlines the workflow of a Librarian using the Library Management System. The process begins with the librarian logging in using valid credentials. Upon successful authentication, the system redirects to the dashboard, where the librarian can perform various administrative tasks. Key operations include adding new books by entering their details, which are then stored in the database, and updating existing book information such as title, author, or availability status. Librarians can also manage users by creating new accounts, updating details, or assigning roles. Another important feature is generating a borrower's report, which lists all currently borrowed books, their due dates, and any associated fines. This helps monitor borrowing activity and enforce library policies. The session concludes when the librarian logs out. This workflow ensures full administrative control over the system's resources and data integrity. This second activity diagram represents how a User (student or staff) interacts with the LMS. The session starts with the user logging in. After successful authentication, the user is taken to the homepage, where they can browse or search for books and view book details. Users can request to borrow or return books. The system first checks if the book is available. If it is, it then verifies whether the user has any outstanding fines. If no issues exist, the system processes the borrowing request. If the book is unavailable or fines exist, the user is notified, and borrowing is restricted until resolved. This workflow enforces borrowing rules, fine management, and system security, ensuring that users interact with the LMS in a fair and structured manner.

Together, these two diagrams offer a complete view of librarian and user interactions within the LMS. They highlight how responsibilities are divided, how the system ensures policy compliance, and how front-end actions trigger secure backend operations. This design supports both administrative control and a smooth user experience in a reliable, rule-based environment.

7. RESULTS & DISSCUSSION

The Library Management System project delivered a fully functional, role-based web application that meets the needs of both users and librarians. Users can register, log in, search for books, borrow and return items, pay fines, and leave feedback—all through an intuitive, user-friendly interface. Librarians have access to administrative tools for managing books, users, and borrowing records, including the ability to generate borrower reports. During development, improvements were made to enforce role-based access, such as moving borrower data to librarian-only views and streamlining the user management interface. System design was supported by various UML diagrams, including class, use case, activity, and sequence diagrams, which helped define system structure, user workflows, and backend interactions.

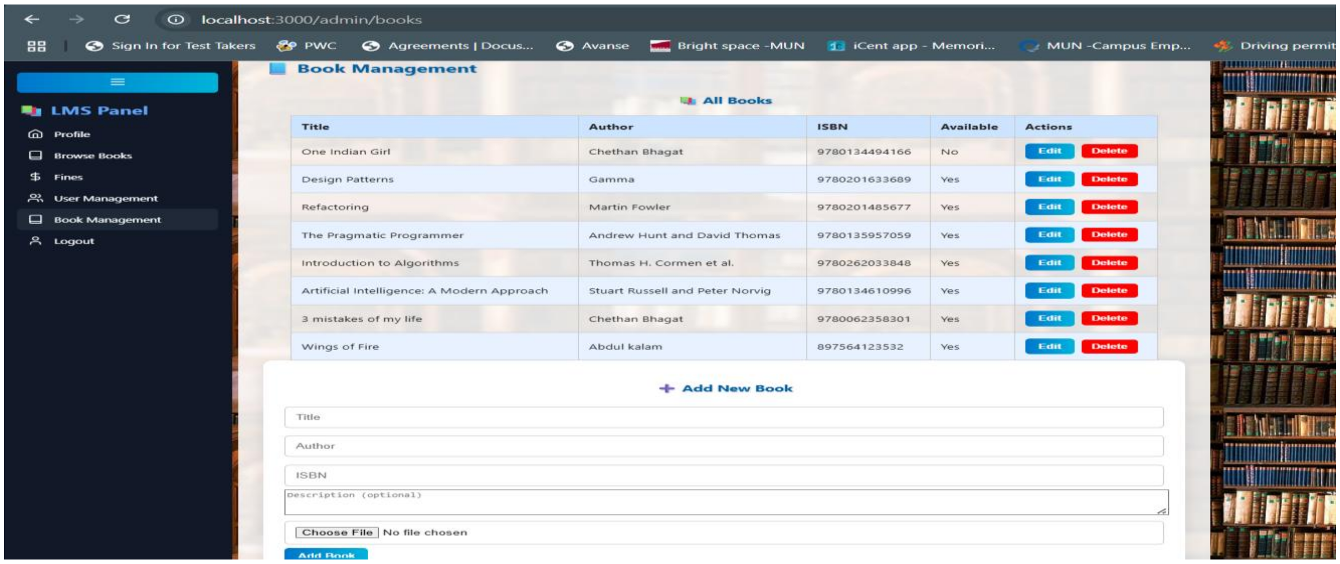
Throughout the project, software engineering best practices were followed using the **SOLID principles**. The system maintained a modular, scalable structure by assigning distinct responsibilities across components (SRP), allowing extension without modifying existing logic (OCP), and supporting polymorphic behavior between users and librarians (LSP). Interfaces and APIs were designed with specificity (ISP), and the frontend communicated with the backend via abstract REST APIs rather than direct database access (DIP), ensuring clean separation between layers.



One Indian Girl
Author: Chethan Bhagat
ISBN: 9780134494166
Status: Borrowed

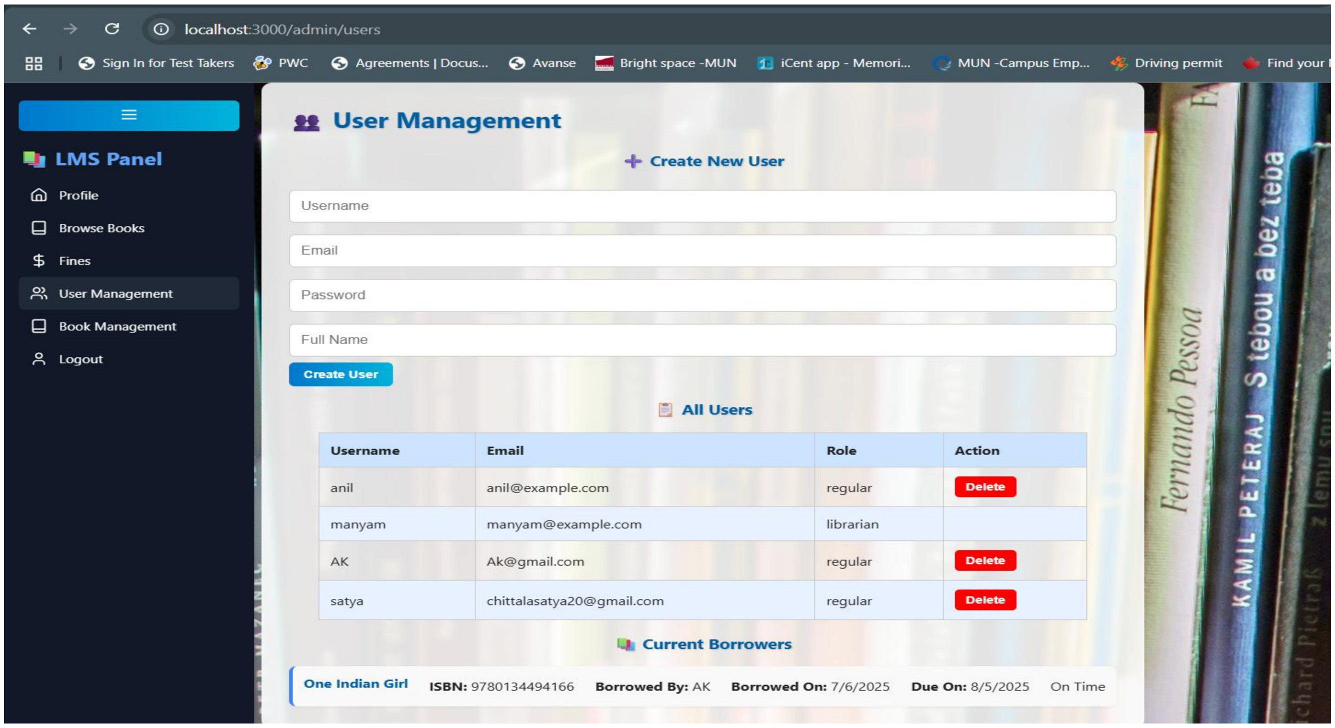
Design Patterns
Author: Gamma
ISBN: 9780201633689
Status: Available

Fig-9: The User Interface of a regular user having with Book CatLog



Title	Author	ISBN	Available	Actions
One Indian Girl	Chethan Bhagat	9780134494166	No	Edit Delete
Design Patterns	Gamma	9780201633689	Yes	Edit Delete
Refactoring	Martin Fowler	9780201485677	Yes	Edit Delete
The Pragmatic Programmer	Andrew Hunt and David Thomas	9780135957059	Yes	Edit Delete
Introduction to Algorithms	Thomas H. Cormen et al.	9780262033848	Yes	Edit Delete
Artificial Intelligence: A Modern Approach	Stuart Russell and Peter Norvig	9780134610996	Yes	Edit Delete
3 mistakes of my life	Chethan Bhagat	9780062358301	Yes	Edit Delete
Wings of Fire	Abdul kalam	897564123532	Yes	Edit Delete

Fig:10 – The User Interface of an Admin/Librarian with options to add & edit books



Username	Email	Role	Action
anil	anil@example.com	regular	Delete
manyam	manyam@example.com	librarian	
AK	Ak@gmail.com	regular	Delete
satya	chittalasatya20@gmail.com	regular	Delete

One Indian Girl **ISBN:** 9780134494166 **Borrowed By:** AK **Borrowed On:** 7/6/2025 **Due On:** 8/5/2025 On Time

Fig-11: Admin Page with User Management & Current Borrowers View

In summary, The UI of the Library Management System has been designed with clear user roles and functionalities. As shown in the figure 10 and 11, Manyam, logged in as a librarian, has access to administrative features such as adding books, managing users, and viewing current borrowers, all from a centralized dashboard. In contrast, in figure 9 Anil, a regular user, is presented with a clean and intuitive interface that allows him to browse the catalog, borrow available books, and view details and reviews. The interface adapts based on user roles, providing only relevant features and ensuring ease of use. The design emphasizes functionality with minimal clutter, supporting seamless interaction with the system. Similarly, we will be having welcome, Login, Register, user profile page etc. from our UI design.

8. CONCLUSION & FUTURE WORK

The Library Management System project successfully delivered a functional, role-based web application to manage essential library tasks. Users can register, log in, search for books, borrow and return items, pay fines, and submit feedback. Librarians have access to tools for managing users, updating the catalog, and generating borrower reports. A RESTful backend and a responsive React frontend ensure smooth interaction, while UML diagrams guided system structure and design validation. A key strength of the project is its use of SOLID principles. Responsibilities were clearly separated (SRP), allowing the system to be easily extended without altering core features (OCP). User and librarian roles were structured hierarchically (LSP), and APIs were tailored to serve only necessary data for each role (ISP). The frontend communicates through abstract APIs, maintaining a clean separation from backend logic (DIP), ensuring long-term maintainability and scalability.

Despite its strengths, the system has a few limitations. Notably, it currently lacks session management. Future improvements should include session management using JWT. Also, the fine payment can be integrated with real payment gateways like Stripe or PayPal. In summary, this project establishes a strong foundation for efficient library management and demonstrates effective application of software engineering principles. With targeted enhancements in user experience, the system holds strong potential for deployment in academic, institutional, or public libraries.

9. REFERENCES

[1] F. F. Okumuş, A. Ramic, and S. Kugele, "A Systematic Mapping Study on Contract-based Software Design for Dependable Systems," *arXiv preprint* arXiv:2505.07542, 2025. [Online]. Available: https://arxiv.org/abs/2505.07542.

[2] B. Meyer, "Applying 'Design by Contract'," *Computer*, vol. 25, no. 10, pp. 40–51, Oct. 1992. https://doi.org/10.1109/2.161279.

[3] W. N. A. W. Mohd, R. Ahmad, M. F. M. Mohsin, and A. H. Abdullah, "Ontology-Based Library Management: Building a Semantic LMS," in *Proc. 2020 6th Int. Conf. on Information Management (ICIM)*, London, UK, 2020. https://doi.org/10.1109/ICIM49319.2020.244689.

10. GIT REPOSITORY DETAILS:

The complete source code for this project is maintained on GitHub and is publicly accessible. The repository contains all frontend and backend code, along with necessary configuration and testing files. To run and check the code base please go through the Readme.md file.

- **GitHub Repo Link**: https://github.com/Anil-Manyam/Spring25
- **Branch Name**: main