University of Rajshahi

Department of Information and Communication Engineering
Faculty of Engineering

Automated Library Management System

with Admin and Client Interfaces

A Project Report Submitted in Partial Fulfillment of the Requirements for the Course

Information System Analysis and Software Engineering Lab

Submitted by: Group 9

1. Sirajus Salekin Sami ID: 2111177113
2. Jubayer Khan Hridoy ID: 2110277114
3. Shakila Sharmin Jui ID: 2112477126
4. Most. Taskin Khatun ID: 2112277151

Lab Supervisors:

1. Dr. Dipankar Das

Professor, Dept. of Information and Communication Engineering

2. Dr. Md. Emdadul Haque

Professor, Dept. of Information and Communication Engineering

3. Dr. Md. Golam Rashed

Associate Professor, Dept. of Information and Communication Engineering

Contents

List of Figures

Chapter 1 : Introduction

1.1 Definition

A Library Management System (LMS) is software designed to manage and automate the daily operations of a library. It is a digital system used to streamline all library functions, including book tracking, borrowing management, and fine calculations, ensuring efficiency and accuracy in library operations.

1.2 Problem Statement

Traditional library systems rely heavily on manual processes, leading to inefficiencies and errors. Below are the key challenges faced in such systems:

- Manual book tracking causes errors and confusion.
- Difficulty in managing large volumes of books and student records.
- Lack of an easy way to check book availability or borrowing status.
- Students lack clear information about due dates or fines.
- Manual fine calculations are time-consuming and error-prone.
- Absence of real-time data or a proper reporting system.
- No secure login or role-based access for different users.
- Inability to easily generate reports or view library statistics.

1.3 Objectives

Objectives are specific goals that guide the development of a Library Management System to meet the needs of librarians and students. The key objectives include:

- Automate the entire process of book issuance and return.
- Maintain accurate and up-to-date records of books, students, and borrowing history.

- Reduce manual work, save time, and minimize human errors.
- Provide easy access to information like book availability, due dates, and fines.
- Ensure secure login and role-based access for different user types.
- Offer a modern, user-friendly interface for efficient library operations.
- Facilitate report generation and library statistics for better decision-making.

1.4 Model Used & Rationale

1.4.1 Agile Model Overview

The Agile Model is an iterative and incremental software development approach that emphasizes collaboration, flexibility, and continuous user feedback. The project is divided into small units called sprints, each delivering a functional part of the software.

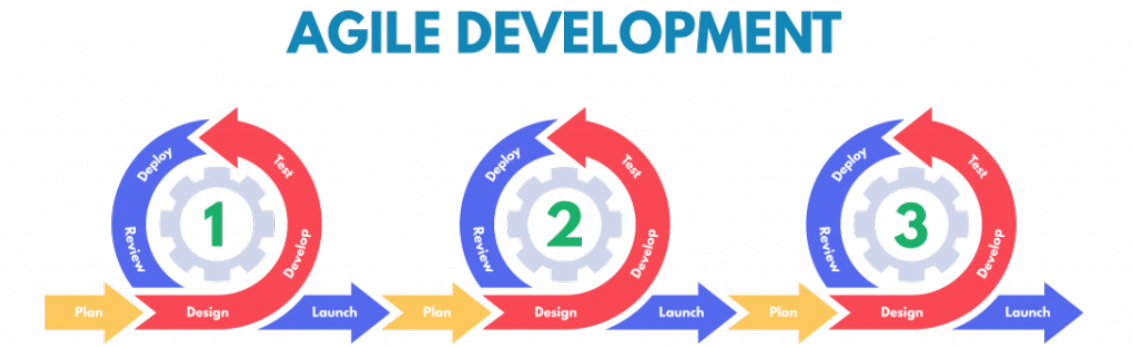

Figure 1.1: Agile Model in the Software Development Life Cycle (SDLC)

1.4.2 Why We Chose the Agile Model

The Agile Model was selected for the Library Management System due to its flexibility and ability to accommodate evolving requirements. Key reasons include:

- ▷ **Fast Delivery of Features**: Important components like the login system, book management, and borrowing module were developed and tested incrementally.
- ▷ **Flexibility to Add New Features**: Features such as fine calculation, student dashboard, and book request system were seamlessly integrated during development.
- ▷ **Improved Team Communication**: Daily progress updates and short sprints enabled quick problem resolution and enhanced collaboration.

- ▷ **Helpful User Feedback**: Feedback from librarians and users during sprints improved search features, user interface, and reporting functions.
- ▷ **Better Quality with Fewer Errors**: Testing after each sprint ensured early error detection, resulting in a stable and reliable system.
- ▷ **Easy to Handle Changes**: Modifications, such as updates to fine rules or UI layout, were implemented without disrupting the entire system.
- ▷ **Visible Progress**: Each sprint delivered working modules, providing clear evidence of continuous improvement.

1.4.3 Limitations Managed

While the Agile Model proved effective, some challenges were addressed:

- Regular user feedback was sometimes delayed.
- Estimating time and total cost was challenging due to evolving features.
- Early stages prioritized development over documentation.
- Active team communication was essential for Agile's success.

Chapter 2 : Requirement Analysis

This chapter outlines the requirements for the Library Management System (LMS), a desktop application designed to streamline library operations such as book management, user administration, and transaction tracking. The requirements are categorized into functional and non-functional requirements, ensuring the system meets the needs of librarians and students while maintaining performance, security, and scalability. These requirements were gathered through stakeholder consultations and aligned with the system's layered architecture and technical specifications.

2.1 Functional Requirements

The functional requirements define the core functionalities of the LMS, enabling efficient management of books, users, and transactions. These are divided into three key modules: Book Management, User Management, and Borrow/Return System.

2.1.1 Book Management

The Book Management module allows librarians to manage the library's book inventory, supporting operations on the `books` table in the database.

- **Add New Books**: Register new books with details such as title, author, ISBN, publication year, and category.
- **Update Book Details**: Modify existing book information, including availability status and location.
- **Delete Books**: Remove books from the inventory, ensuring no active transactions exist.
- **Search for Books**: Search books by title, author, or ISBN with partial and exact match capabilities.
- **View All Books**: Display the complete book catalog with availability status.
- **Show Availability**: Indicate whether a book is available, borrowed, or reserved.

2.1.2 User Management

The User Management module handles student and admin accounts, interacting with the `users` and `student_profiles` tables.

- **Add New Users**: Create accounts for students and admins with role-based access (admin or student).
- **Update Details**: Modify user information, including email, phone number, and borrowing limits.
- **Delete Users**: Remove user accounts, ensuring no active transactions or requests.
- **Search for Users**: Locate users by username, email, or student ID.
- **View All Users**: Display a list of all users with their roles and borrowing status.

2.1.3 Borrow/Return System

The Borrow/Return System manages book lending and returning, utilizing the `transactions` and `book_requests` tables.

- **View Available Books**: Display books with `available = TRUE` for borrowing.
- **Search Books to Issue**: Search books for issuing using title, author, or ISBN.
- **Issue Books**: Record borrowing transactions with due dates and update book availability.
- **Return Books**: Process book returns, update availability, and log transactions.
- **View Issued Books**: Show a user's currently borrowed books with due dates.

2.2 Non-Functional Requirements

The non-functional requirements ensure the system's performance, security, and usability meet industry standards for a robust library management solution.

Scalability

The system supports growth to handle up to 10,000 books and 1,000 users without performance degradation, facilitated by MySQL's indexing and Hibernate's caching.

Performance

Query response times (e.g., book searches, transaction updates) are under 2 seconds, achieved through database indexes on `books.title`, `books.author`, and `books.isbn`.

Security

User data is protected with BCrypt password hashing, role-based access control (admin vs. student), and secure database connections to prevent unauthorized access.

Usability

The JavaFX-based interface is intuitive, with clear navigation and responsive design for librarians and students, supporting CSS-styled layouts.

Backup and Recovery

Regular automated backups of the MySQL database prevent data loss, with recovery mechanisms to restore data in case of failures.

Compatibility

The system is accessible on desktop platforms, with plans for future mobile and web interfaces to enhance accessibility.

2.3 Proposed Solution

To address the limitations of manual library systems and meet the defined objectives, we propose a **Desktop-based Library Management System** built using modern Java technologies. The system introduces automation, role-based access, and a responsive interface for both administrators and students.

The key components of the proposed solution are:

- **Authentication & Security:** Secure login system using BCrypt password hashing for both admin and student roles.
- **Book Management:** Admins can add, update, delete, and track books using an intuitive interface backed by Hibernate for ORM (Object-Relational Mapping).
- **Search Functionality:** Students can search for books using optimized search queries for better performance and partial matches.
- **Borrowing & Returning:** A transaction service ensures books are issued and returned with proper validation. Atomic operations are used to maintain consistency between book status and borrowing records.
- **Fine Calculation:** Automatic fine tracking based on due dates and return status, removing the need for manual calculations.
- **Reports & Statistics:** Admin dashboard offers overdue book reports, top borrowed books, and student activity logs via SQL aggregations and filtering.
- **User Interface:** The application features a clean, responsive JavaFX/Java-based frontend that improves usability and access across devices.
- **Scalability:** The system is modular, making it easy to extend features such as book reservation, notifications, and multi-library support.

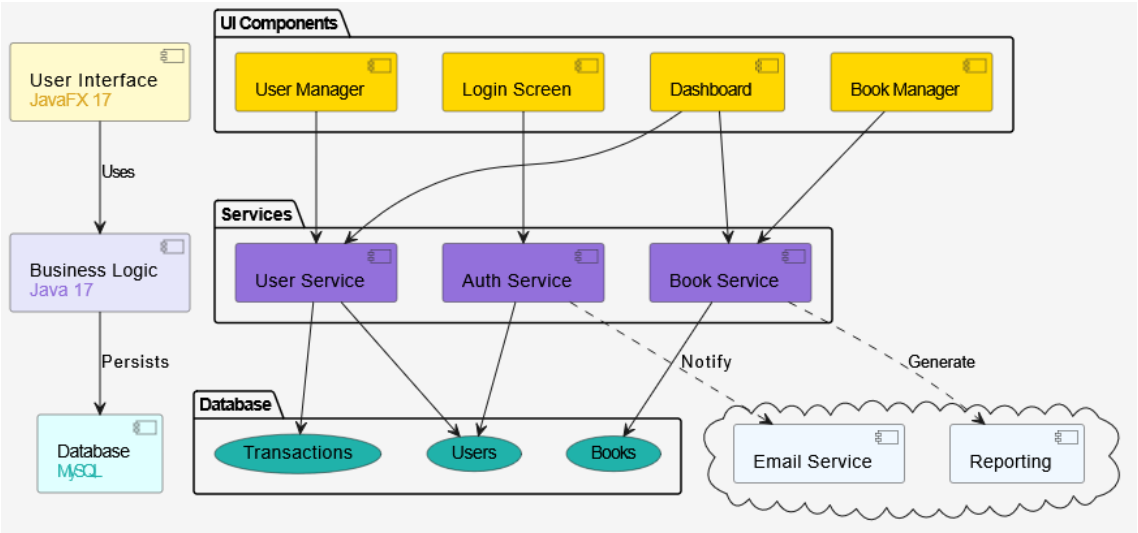

Figure 2.1: Proposed System Architecture for the Library Management System

Chapter 3 : System Architecture and Technology Stack

The proposed Library Management System (LMS) follows a **layered architectural design** to ensure modularity, maintainability, and scalability. It is a desktop-based application developed with Java technologies, leveraging JavaFX for the user interface and Hibernate ORM for database interactions.

3.0.1 Architecture Layers

- **Presentation Layer:** Implements the user interface using JavaFX, providing an intuitive and responsive environment for librarians and students. This layer handles user input, data validation, and displays outputs such as book lists, user details, and transaction histories.
- **Business Logic Layer:** Contains core services managing system workflows including authentication, book management, user management, borrowing and returning processes, and fine calculations. This layer enforces business rules, validates transactions, and ensures consistency.
- **Data Access Layer:** Uses Hibernate ORM to interact with the MySQL database, abstracting direct SQL queries into object-oriented operations. This layer manages CRUD operations on key entities such as books, users, and transactions, enabling efficient data persistence and retrieval.
- **Database Layer:** A MySQL relational database storing all system data. It includes normalized tables for books, users, transactions, and book requests. Proper indexing is used to optimize search and transaction performance.

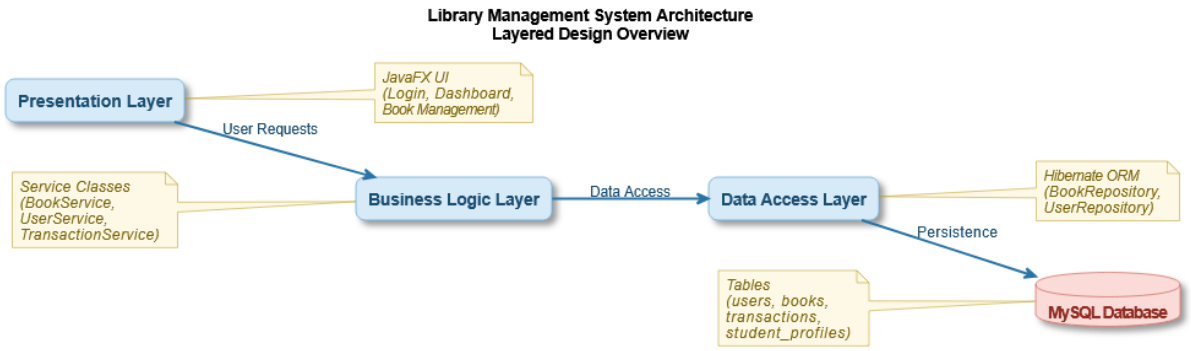

Figure 3.1: Layered Architecture of the Library Management System illustrating the Presentation Layer (JavaFX UI), Business Logic Layer (core services and workflows), Data Access Layer (Hibernate ORM), and Database Layer (MySQL with normalized tables and indexing).

3.1 Technology Stack

This project adopts a layered architecture where each technology serves a distinct role within the system:

3.1.1 Presentation Layer

- **Java 17**
  Used across all layers, Java serves as the core programming language. In the UI layer, it drives the event logic, input validation, and controller interactions.

- **JavaFX 17.0.2**
  Powers the user interface, providing an interactive and modern GUI for students and librarians. It uses scene graphs, event-driven patterns, and CSS styling to create responsive views.

3.1.2 Business Logic Layer

- **Java 17**
  Implements service classes like `BookService`, `UserService`, and `TransactionService`, where business rules and validation are enforced.

- **JUnit 5 & Mockito**
  Used to test business logic methods, ensuring reliability of services such as book issuance, user authentication, and fine calculation. Mockito mocks repository or DAO layers during tests to isolate logic.

3.1.3 Data Access Layer

- **Hibernate 6.2.13**
  Acts as the bridge between business logic and the database. It maps Java entities to database tables and handles persistence operations through repositories like `BookRepository` and `UserRepository`.

- **Maven**
  Manages dependencies for Hibernate and other libraries required by the data access layer. Ensures consistent builds and integration of database connectors.

3.1.4 Database Layer

- **MySQL**
  Stores all system data including users, books, transactions, and profiles. It is accessed via Hibernate for CRUD operations, with indexes and constraints to ensure performance and consistency.

3.1.5 Security and Communication (Cross-Cutting Layers)

- **BCrypt**
  Used in the business logic layer for secure password hashing and validation during login. Ensures encrypted password storage in the database.

- **JavaMail API (Planned)**
  To be integrated into the business layer for email notifications such as password resets or overdue alerts.

- **OkHttp (Planned)**
  Reserved for possible future REST integrations or external data fetching needs, operating alongside the service layer.

3.1.6 Version Control and Collaboration

- **Git & GitHub**
  Used throughout the development process for version tracking, collaborative development, and deployment history. Each commit reflects changes in one or more architectural layers.

Chapter 4 : Design

This chapter outlines the system design using three core models: the AIR (Architecture–Interaction–Responsibility) Diagram, the Class Diagram, and the Database Schema. Together, they define the system's structure, behavior, and data organization.

4.1 AIR Diagram Concept

The AIR (Architecture-Interaction-Responsibility) diagram outlines the high-level structure of the system. It defines:

- **Architecture:** Logical layering and major components, such as the UI layer, service layer, and data layer.
- **Interaction:** Flow of data and control between components. For example, how user inputs trigger backend operations or database queries.
- **Responsibility:** The role of each component or class in the system, such as handling business logic, user interaction, or data access.

4.2 Class Diagram Concept

The Class Diagram represents the object-oriented design of the system. It shows:

- The system's classes, such as `Book`, `Member`, `Admin`, and `Transaction`.
- Attributes and methods of each class.
- Relationships like inheritance, association, aggregation, or composition.

This diagram helps developers understand the structure and behavior of individual components and their relationships in the system.

4.3 Database Schema Concept

The Database Schema illustrates the relational structure of the backend MySQL database. It includes:

- Tables such as `users`, `books`, `borrow_records`, `categories`, etc.
- Columns, data types, primary and foreign keys.
- Relationships (one-to-many, many-to-many) between tables.

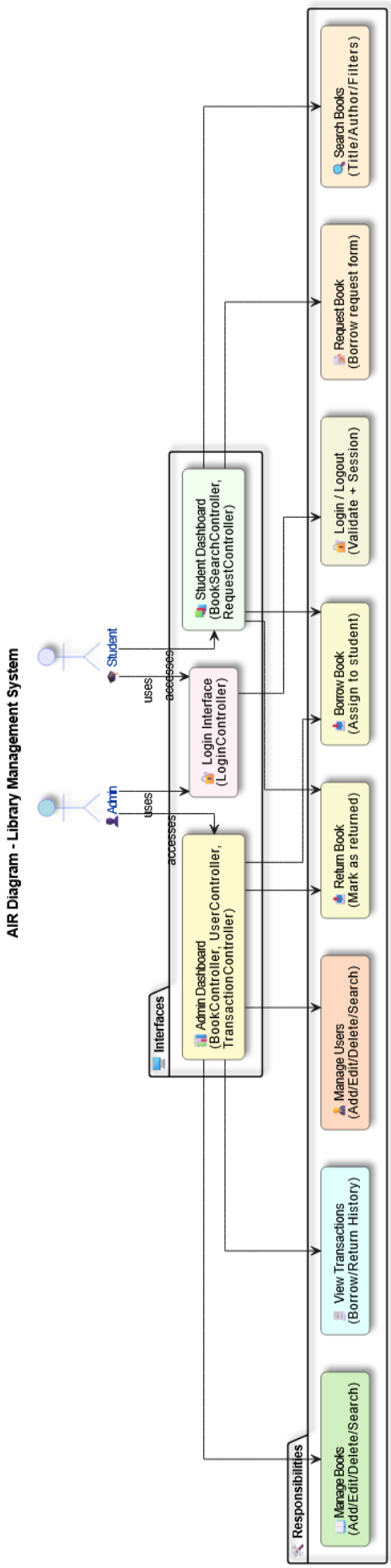

Figure 4.1: AIR Diagram showing the high-level component interactions and responsibilities

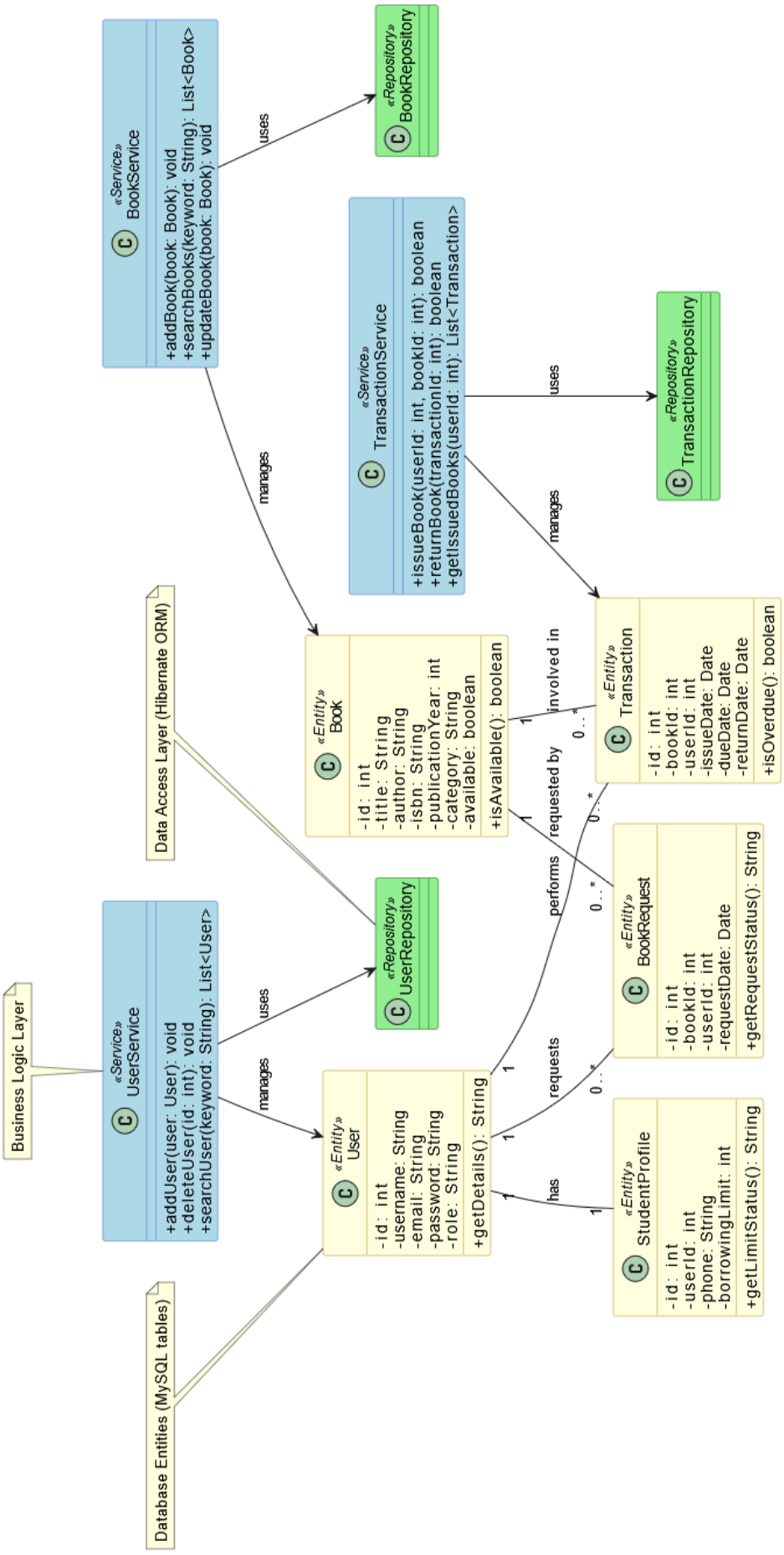

Figure 4.2: UML Class Diagram showing classes, relationships, and methods used in the system

Figure 4.3: ER Diagram representing the normalized relational schema of the MySQL database

Chapter 5 : Testing

A robust testing strategy is crucial for ensuring the quality, reliability, and correctness of any software system. For the Library Management System, testing was integrated throughout the development lifecycle to identify and resolve defects early, validate functionality, and enhance the overall stability of the application.

Our testing strategy primarily focused on **Unit Testing** and **Integration Testing**, complemented by manual testing efforts. The goal was to ensure that individual components perform correctly in isolation and that these components interact as expected when combined.

5.1 Unit Testing

Unit Testing was the foundational level of testing in our project. This involved testing the smallest testable parts of the application, referred to as "units," in isolation. In our layered architecture, the primary units tested were individual methods within the Service Classes (`AuthService`, `BookService`, `UserService`, `TransactionService`) and Utility Classes (`ValidationUtil`, `SecurityUtil`).

- **Tool Used:** JUnit 5 was the chosen framework for writing and executing unit tests. JUnit 5 provides a flexible and powerful platform for structuring tests using annotations such as `@Test`, `@BeforeEach`, `@AfterEach`, etc., and offers a wide range of assertion methods (`assertEquals`, `assertTrue`, `assertNull`, etc.) to verify the expected outcomes of tested units.
- **Objectives:** The objectives of unit testing were to verify the correctness of the business logic, ensure that utility functions perform as intended, and confirm that individual methods handle various inputs (valid, invalid, edge cases) appropriately.

5.2 Mocking for Isolation

To test units (especially Service classes) in isolation from their dependencies, such as the database access layer, we utilized a mocking framework.

- **Tool Used:** Mockito was employed to create mock objects. Mockito allowed us to simulate the behavior of dependencies (like `UserRepository`, `BookRepository`) and control the outcomes of method calls to these dependencies during unit tests.

- **Role of Mocking:** By mocking repositories, we could test the logic within a Service method without requiring an actual database connection or performing real database operations. This made the unit tests faster, more reliable, and truly focused on the logic of the unit under test, isolating it from external factors. We could also verify that the Service class interacted with its dependencies as expected (e.g., called a specific repository method a certain number of times).

5.3 Conclusion of Testing Strategy

The chosen testing strategy, heavily leveraging JUnit 5 for unit testing and Mockito for effective isolation, allowed us to maintain a high level of code quality and confidence in the core business logic. By identifying and fixing issues early in the development cycle, we aimed to deliver a more stable, reliable, and robust Library Management System. Future efforts would involve increasing automated test coverage, particularly at the integration and system levels.

Chapter 6 : Project Screenshots and Discussion

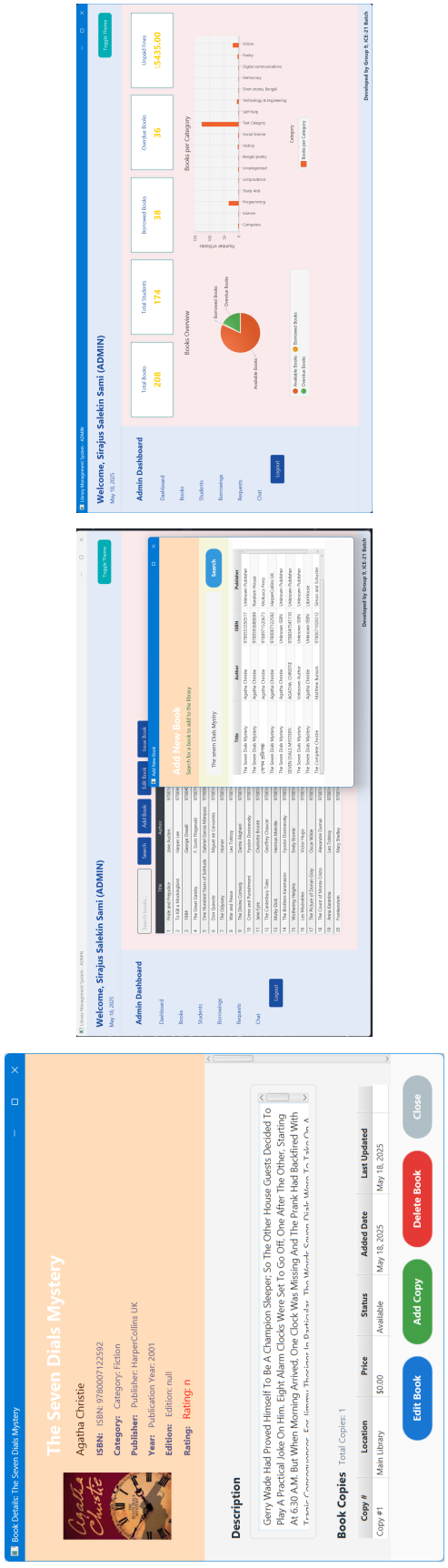

Figure 6.1: Project screenshots – part 1

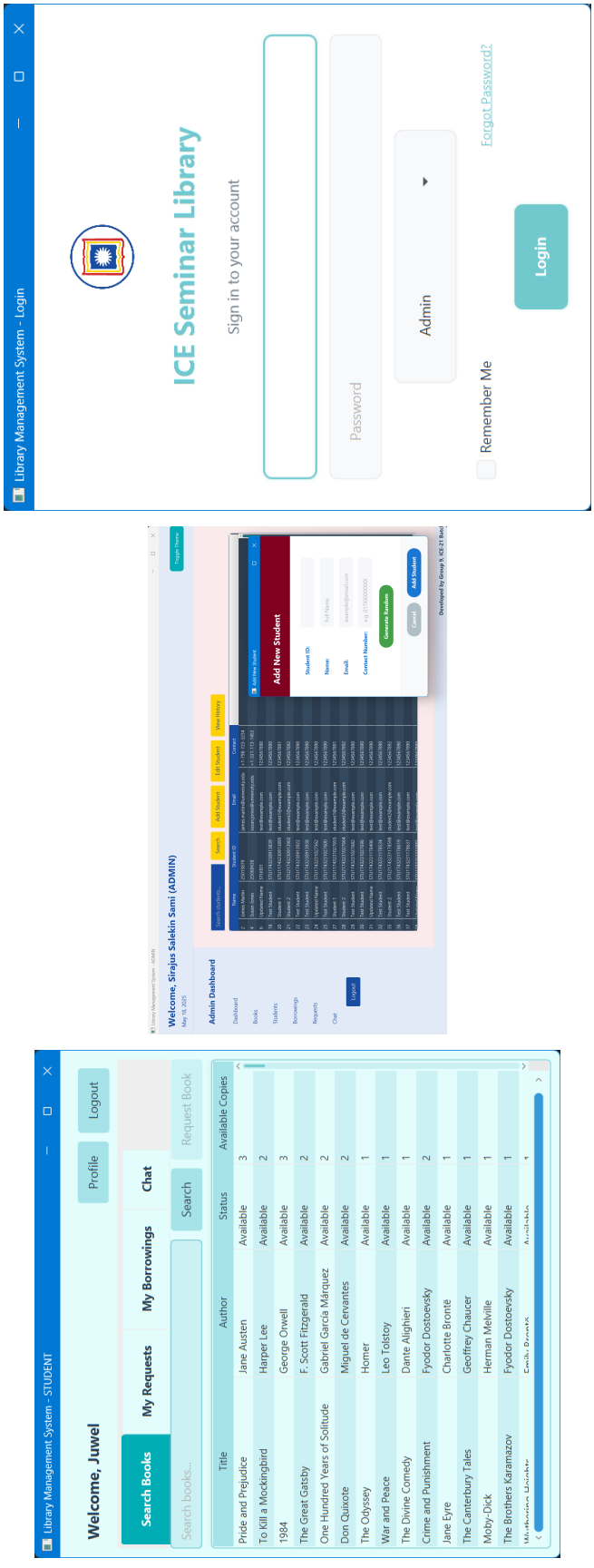

Figure 6.2: Project screenshots – part 2

6.1 User Interface and Functionality

The User Interface (UI), developed using JavaFX, serves as the Presentation Layer of the system. It ensures an intuitive and role-based interaction model for both librarians and students. Key UI components are organized using FXML, with dedicated controller classes handling logic and event management.

6.1.1 Login Screen

- Authenticates users and determines their role (Admin or Student).
- UI elements include email/username input, password, role selector, and login button.
- The `LoginController` interacts with `AuthService` to:
  - Verify credentials using BCrypt password hashing.
  - Confirm user-role match and load role-specific dashboard.
  - Display error messages for invalid login attempts.

6.1.2 Admin Dashboard - Books Tab

- Central interface for managing books.
- Features: welcome header, navigation tabs, book search, add book/copy buttons, and a table of books.
- Functionalities:
  - **Search**: Controller triggers a query via `BookService`.
  - **Add Book/Copy**: Opens input form, saves via service layer, updates database.
  - Table supports future extension (e.g., Edit/Delete).

6.1.3 Admin Dashboard - Students Tab

- Interface for managing student accounts.
- Elements: student search bar, add student button, and student data table.
- Functionalities:
  - **Search Students**: Uses `UserService` and repositories.

- **Add Student**: Saves records into both `users` and `student_profiles` tables with hashed passwords.

6.1.4 Admin Dashboard - Borrowings Tab

- Displays borrowing transactions and allows return processing.
- Elements: search bar, filter dropdown, return button, transaction table (color-coded).
- Functionalities:
  - **Search/Filter**: Retrieves filtered transactions from `TransactionService`.
  - **Return Book**: Updates status, records return, adjusts student data, and calculates fines if applicable.

6.1.5 Admin Dashboard - Book Requests Tab

- Handles book requests submitted by students.
- Elements: request table, approve/reject buttons.
- Functionalities:
  - **Approve Request**: Updates request status, may initiate borrowing.
  - **Reject Request**: Marks request as rejected.

6.1.6 Admin Dashboard - Statistics Tab

- Visual and numerical summary of system activity.
- Elements: key metrics, pie chart (book overview), and recent activity feed.
- Functionalities:
  - Aggregates data from multiple services (books, users, transactions).
  - Offers visual insights via charts and summaries.

6.1.7 Student Dashboard

- Role-specific UI for students.
- Tabs: Search Books, My Requests, My Borrowings.
- Header includes student name, profile, and logout.

Search Books Tab

- Allows students to search the catalog.
- Elements: search bar, request button, and book table.
- Functionality:
  - Students can request a book from the displayed results.

Chapter 7 : Conclusion and Contribution

Conclusion

The Library Management System is a robust and scalable solution designed using Java 17, JavaFX, MySQL, and Hibernate. It offers a user-friendly interface and ensures a secure, efficient way to manage library resources. Security features like `BCrypt` reinforce password protection, while the use of the `JavaMail API` opens up potential for future communication features such as overdue notifications and system alerts.

The system is flexible and modular, making it easy to maintain and extend. It can be tailored to meet the specific needs of educational institutions or public libraries, thereby serving as a long-term solution for digital library automation.

Contribution Table

Component	**Sami**	**Hridoy**	**Taskin**	**Shakila**
Core System Architecture	100%	–	–	–
Core Backend Development	70%	10%	10%	10%
Frontend Implementation (Client Side)	–	80%	10%	10%
Frontend Implementation (Admin Side)	40%	20%	20%	20%
UI/UX Design	–	20%	40%	40%
Testing & Quality Assurance	30%	30%	20%	20%
Project Management	80%	10%	5%	5%
Documentation	–	20%	40%	40%
Presentation Materials	25%	25%	25%	25%
Technical Diagrams	50%	30%	10%	10%
Project Report	10%	40%	25%	25%

Table 7.1: Team contribution distribution across project components