# Software Requirements Specification (SRS) for Library Management System

## Chapter 1: Introduction

### 1.1 Purpose

This document explains what the Library Management System (LMS) will do. The system will help libraries manage books, borrowing, returns, and fines more efficiently.

### 1.2 Scope

The Library Management System will have the following features:

• User Registration and Login: Users can sign up, log in, and reset their passwords.

• Book Management: Librarians can add, update, or remove books from the system.

• Book Search: Users can search for books by title, author, or category.

• Borrowing and Returning Books: Users can borrow books for a limited time and return them before the due date.

• Fine Management: The system will automatically calculate fines for overdue books.

• Reports and Analytics: Librarians can create reports about book usage and overdue books.

• Notifications: Users will get reminders for due dates and overdue books.

### 1.3 Technologies Used

• Frontend: React.js (for the user interface)

• Backend: Node.js with Express.js (for the server)

• Database: MySQL (to store data)

• Authentication: JWT (JSON Web Token) (for secure login)

• Hosting: AWS or Firebase (to host the system online)

### 1.4 Intended Audience

• Developers: To understand what the system needs to do.

• Library Staff (Librarians/Admins): To manage books and track borrowed/returned books.

• Students: To search, borrow, and return books easily.

• Project Managers: To make sure the system is built correctly.

### 1.5 Overview of the Document

• Chapter 1: Introduction - Explains the purpose, scope, technologies, and who will use the system.

• Chapter 2: Overall Description - Gives a high-level view of the system and its features.

• Chapter 3: Functional and Non-Functional Requirements - Lists the system's features and performance needs.

## Chapter 2: Overall Description

### 2.1 Overall Description of the System

The Library Management System (LMS) is a website that helps libraries manage book borrowing, returning, and tracking. Students can search for books, borrow them, and return them on time. Librarians can manage the book inventory, track borrowed books, and handle overdue fines.

### Key Features:

• User Roles: There are two types of users: Admin (Librarian) and Students.

• Book Inventory Management: Admins can add, update, or delete books.

• Borrowing System: Users can borrow books for up to 14 days.

• Return System: Users must return books before the due date to avoid fines.

• Fine Management: The system will calculate fines for overdue books and notify users.

• Reporting and Analytics: Librarians can create reports about book usage and overdue fines.

### System Constraints:

• The system must be available 24/7.

• Users must log in with valid credentials.

• Books can only be borrowed for a maximum of 14 days.

• The system should support at least 500 users at the same time.

## Chapter 3: Functional and Non-Functional Requirements

### 3.1 Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Description |
| FR-1 | User Authentication | Users can register, log in, and reset passwords securely. |
| FR-2 | Book Search | Users can search for books by title, author, or category. |
| FR-3 | Borrow Books | Users can borrow books for up to 14 days. |
| FR-4 | Return Books | Users can return borrowed books before the due date. |
| FR-5 | Fine Calculation | The system will calculate fines for overdue books ($0.50 per day). |
| FR-6 | Admin Management | Admins can add, edit, or remove books. |
| FR-7 | Reports & Analytics | Admins can create reports about book usage and overdue fines. |

### 3.2 Non-Functional Requirements

|  |  |  |
| --- | --- | --- |
| ID | Requirement | Measurement Criteria |
| NFR-1 | Performance | The system should respond within 2 seconds for 95% of user requests. |
| NFR-2 | Security | User passwords must be stored securely, and data must be sent over HTTPS. |
| NFR-3 | Availability | The system should be available 99.5% of the time. |
| NFR-4 | Usability | The user interface should be easy to use and follow design guidelines. |
| NFR-5 | Scalability | The system should support 500+ users at the same time without slowing down. |
| NFR-6 | Data Integrity | All book borrowing and returning transactions should be secure and accurate. |
| NFR-7 | Backup & Recovery | The system should automatically back up data every 24 hours to prevent loss. |

Diagrams

sequence diagram1\_AdminLogIN\_BookMangment:

**A screenshot of a computer

AI-generated content may be incorrect.**

**SequenceDiagram\_UserLogin\_BookIssuance**

**A screenshot of a computer

AI-generated content may be incorrect.**

**ClassDiagram**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. Use Case Diagram

**Description**:

The Use Case Diagram offers a comprehensive overview of the Library Management System’s functional requirements, delineating the interactions between external actors—namely the Admin and User—and the system’s key use cases. This diagram serves to outline the primary operations available to each actor, emphasizing the foundational role of the Login use case as a gateway to all other functionalities.

**Components**

Actors:

Admin: Represents the system administrator tasked with overseeing book inventory, user accounts, and report generation.

User: Represents library patrons authorized to perform actions such as borrowing and returning books and submitting feedback.

Use Cases:

Login: The initial authentication process required for both Admin and User to access system features.

Manage Books: Enables the Admin to perform operations such as adding, removing, or reviewing book records.

Issue Book: Allows the User to check out an available book from the library.

Return Book: Permits the User to return a previously borrowed book.

Submit Feedback: Provides the User with the ability to submit ratings and comments regarding their experience.

View Reports: Grants the Admin access to generate and analyze usage statistics and overdue fine reports.

**Relationships**:

The Admin and User actors are directly linked to the Login use case, indicating it as a mandatory step.

Dashed arrows from Manage Books, Issue Book, Return Book, Submit Feedback, and View Reports to Login signify a dependency, accompanied by a note affirming Login as a prerequisite for accessing these functionalities.

**Explanation**

This diagram underscores the critical dependency of all system operations on successful authentication via the Login use case, reinforcing the system’s security framework. The Admin’s exclusive access to Manage Books and View Reports highlights their administrative responsibilities, while the User’s interactions are confined to Issue Book, Return Book, and Submit Feedback, aligning with typical patron activities. The dependency notation ensures clarity on access control, a key non-functional requirement, and provides a high-level blueprint for the system’s intended user interactions.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. Sequence Diagram: Admin Login and Book Management

**Description**

The Sequence Diagram for Admin Login and Book Management illustrates the temporal sequence of interactions within the Library Management System when an Admin authenticates and subsequently adds a book. It details the message exchanges among the Admin actor, the LibraryApp class, and the database components (mysql.connector.cursor and mysql.connector.connection), providing insight into the authentication and book management workflows.

**Components**

Actor: Admin, who initiates the login process and performs book management tasks.

Objects:

LibraryApp: The core application class responsible for managing the user interface and operational logic.

mysql.connector.cursor: Facilitates the execution of SQL queries against the database.

mysql.connector.connection: Handles the database connection and commits transaction changes.

Messages:

The Admin triggers the login process by selecting "Admin Login," invoking the admin\_login() method in LibraryApp.

The Admin inputs credentials, prompting a database query to verify the password.

Upon successful authentication, the Admin navigates to the "Book Management" menu and adds a book, involving user input via a UI, a database insertion command, and a commit operation.

The diagram includes an error-handling path for invalid login attempts, limiting the user to three tries before the application exits.

Control Flow:

An alt fragment delineates the paths for successful login (proceeding to book addition) and failed login (displaying an error message).

**Explanation**

This sequence diagram maps the Admin’s workflow from authentication to adding a book, showcasing the system’s security validation through database interaction and its data management capabilities. The process highlights the integration of user interface elements with database operations, ensuring data consistency via the commit mechanism. The inclusion of error handling for login failures underscores the system’s reliability, aligning with security and usability requirements. This diagram is essential for understanding the Admin’s operational flow and the system’s database dependency.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Sequence Diagram: User Login and Book Issuance

**Description**

The Sequence Diagram for User Login and Book Issuance outlines the dynamic interplay between the User actor, the LibraryApp class, and database components (mysql.connector.cursor and mysql.connector.connection) during the user authentication and book issuance procedures. It captures the step-by-step communication, including checks for existing loans and book availability, with book options presented via a tk.Listbox widget.

**Components**

Actor: User, who initiates the login and book issuance actions.

**Objects**:

LibraryApp: Oversees the user interface and core business logic.

mysql.connector.cursor: Executes SQL queries to validate user status and update records.

mysql.connector.connection: Manages database transaction commits.

**Messages**:

The User selects "User Login," triggering the user\_login() method in LibraryApp.

The User enters credentials, initiating a database query for authentication.

Upon success, the User accesses the "Issue Book" feature, selecting a book from a tk.Listbox display, followed by system checks for existing loans and availability.

If criteria are satisfied, the book is issued with a recorded borrow date; otherwise, relevant error messages are displayed.

Failed login attempts are restricted to three before the application terminates.

**Control Flow:**

Nested alt fragments manage the flow for successful login, loan eligibility checks, and error scenarios.

**Explanation**

This sequence diagram traces the User’s path from authentication to issuing a book, emphasizing the system’s loan management logic using a tk.Listbox for book selection. It details database interactions for credential validation and loan status verification, as well as the update process for issuing a book, including due date assignment. The nested conditional logic reflects critical system rules (e.g., one book per user, availability constraints), fulfilling functional requirements. The diagram supports verification of the system’s workflow integrity and error management, ensuring a reliable user experience, though the tk.Listbox restricts display to a single-column format.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**4. Class Diagram**

Description

The Class Diagram presents the static structure of the Library Management System, detailing the classes, their attributes, methods, and interrelationships within a package-based framework. It reflects the object-oriented design, organizing classes under the tkinter module for graphical user interface components and the mysql.connector module for database connectivity, with tk.Listbox substituted for the previously considered ttk.Treeview to facilitate simpler list displays.

**Components**

Packages:

tkinter: Encompasses the foundational GUI framework classes.

tk.Tk: The primary window class, offering methods for window configuration and event looping.

tk.Frame: A layout container class for organizing UI elements.

tk.Entry: A widget for text input capture.

tk.Listbox: A class for rendering a basic list of items, serving as a replacement for more complex table displays.

tk.Button: A widget for initiating actions through user commands.

mysql.connector: Includes classes for database interaction.

connection: Manages the database link with a commit() method.

cursor: Executes database queries with methods such as execute() and fetchone().

LibraryApp: The central class, inheriting from tk.Tk, which coordinates application logic and interfaces with GUI and database components.

**Relationships**:

LibraryApp extends tk.Tk to leverage Tkinter’s windowing capabilities.

LibraryApp maintains associations with tk.Frame, tk.Entry, tk.Listbox, tk.Button, connection, and cursor to support UI rendering and database operations.

Attributes and Methods:

LibraryApp features attributes mydb and mycursor for database connectivity, alongside an extensive set of methods covering login, menu navigation, book and user management, and reporting functions.

**Notes**:

A note beside tk.Listbox indicates its role as a replacement for ttk.Treeview, noting its limitation to single-column data display.

Explanation

The Class Diagram organizes the system’s architecture by grouping classes under their respective tkinter and mysql.connector packages, eliminating the ttk submodule and adopting tk.Listbox for a streamlined GUI approach. This adjustment aligns with a reliance on standard tkinter widgets, though it sacrifices the multi-column table functionality once provided by ttk.Treeview. The inheritance and association relationships illustrate how LibraryApp integrates these components to meet the system’s requirements, offering a clear structural guide for development and maintenance. The accompanying note highlights the trade-off in display complexity, which developers should address during implementation.

**INTERFACE:**

This section documents the user interface (UI) and user experience (UX) design of

"The Library Management System," a desktop application developed using

Python and Tkinter. The system provides a streamlined interface for both admin and

user roles to manage library operations such as book issuance, returns, feedback sub

mission, and report generation. The UI is designed to be minimalistic and functional,

prioritizing ease of navigation and accessibility. The UX focuses on ensuring efficiency

workflows for users, with clear feedback mechanisms and intuitive controls.

**Design Principles:**

The UI design adheres to the following principles:

• **Simplicity**: A clean, uncluttered layout with minimal visual distractions ensures

users can focus on their tasks.

• **Consistency**: Uniform button placement (e.g., "Back" button at the bottom of

each screen) and typography enhance predictability.

• **Functionality**: Each screen is purpose-driven, presenting only the necessary ele

ments to complete a task.

• Feedback: The system provides immediate feedback (e.g., error messages for failed

logins, success messages for book issuance) to guide users.

The UX design emphasizes:

• **Efficiency**: Streamlined workflows reduce the number of clicks needed to perform

actions (e.g., issuing a book directly from a list).

• **Accessibility**: Large, readable fonts and high-contrast buttons cater to a wide

range of users.

• **Error Prevention**: Input validation (e.g., limiting login attempts) and clear

prompts reduce user errors.

**Interface Screenshots and Descriptions:**

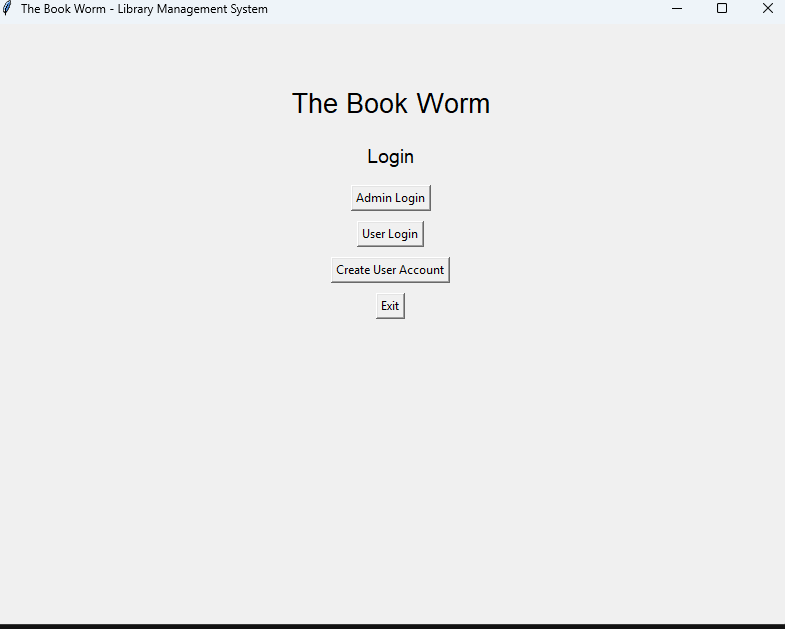
The following subsections present the key interface screens of the system, along with

detailed descriptions of their UI elements and UX considerations. Each screenshot is

referenced by its title, and placeholders indicate where images would be embedded in a

typical document setup

**Login Screen:**

****

**Description**: The Login Screen serves as the entry point to the system, displayed

upon application launch. It features the title "The Book Worm" followed by a "Login"

subtitles, with three buttons: "Admin Login," "User Login," and "Create User Account,"

and an "Exit" button at the bottom. The layout is centered, with buttons vertically

stacked for clarity.

**UI Elements:**

• Title and Subtitle: "The Book Worm" and "Login" are prominently displayed at

The top, using a bold font to establish the applications identity.

• Buttons: Four buttons are presented in a vertical arrangement, each with a reaction

angular outline and clear labels. The "Exit" button is separated at the bottom to

indicate its distinct function.

**UX Considerations:**

• The minimalistic design ensures users can quickly identify their role (Admin or

User) and proceed accordingly.

• The "Create User Account" option supports new users, enhancing inclusivity.

• The “Exit" button provides a clear escape mechanism, reducing frustration for users

who wish to close the application

**Admin Menu:** A screenshot of a computer

AI-generated content may be incorrect.

**Description**: The Admin Menu is displayed after a successful admin login, pro

viding access to administrative functions. It includes buttons for "Book Management,"

"User Management," "Admin Management," "View Feedback," "Generate Reports," and

"Logout," arranged vertically.

**UI Elements:**

• Title: "Admin Menu" is centered at the top, indicating the user’s current context.

• Buttons: Six buttons are listed vertically, each with a consistent rectangular out

line and label. The "Logout" button is positioned at the bottom to signify the end

of the session.

UX Considerations:

• The vertical arrangement of buttons aligns with the systems consistent navigation

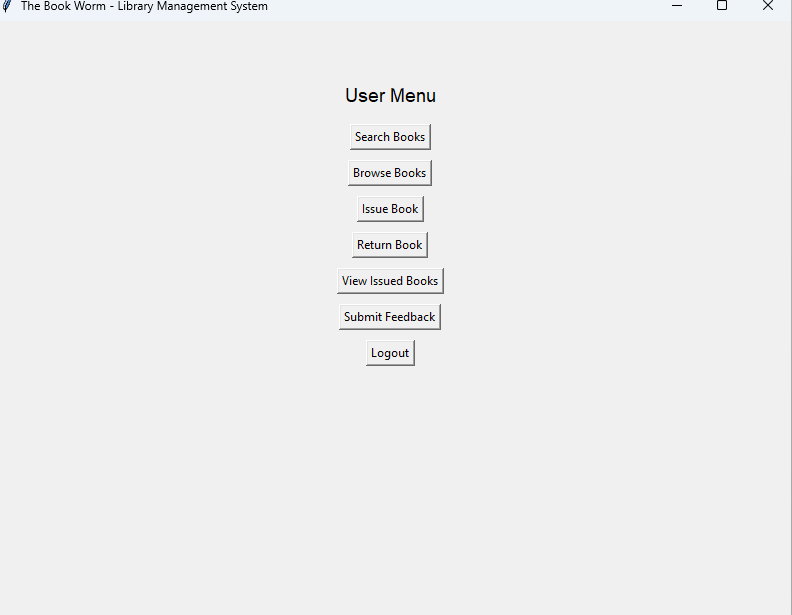
pattern, making it easy for admins to locate desired functions.

• The "Logout" button ensures admins can securely exit their session, addressing

security requirements.

• The menu consolidates all admin functionalities into a single screen, reducing navigation

Complexity

**User Menu:**

**Description**: The User Menu appears after a successful user login, offering options

tailored for library patrons. It includes buttons for "Search Books," "Browse Books,"

"Issue Book," "Return Book," "View Issued Books," "Submit Feedback," and "Logout,"

arranged vertically.

**UI Elements:**

• Title: "User Menu" is displayed at the top, clearly indicating the users context.

• Buttons: Seven buttons are presented in a vertical stack, each with a consistent

design. The "Logout" button is at the bottom.

**UX Considerations:**

• The menu provides a comprehensive set of user actions in a single view, minimizing

navigation steps.

• The consistent button placement and design enhance usability by maintaining familiarity across screens.

• The "Logout" option ensures users can end their session securely, aligning with

Privacy requirements

**Generate Reports Screen:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Description**: The Generate Reports Screen, accessible from the Admin Menu, allows

admins to generate analytical reports. It features two buttons: "Book Usage Report" and

"Overdue Fines Report," with a "Back" button at the bottom.

**UI Elements:**

• Title: "Generate Reports" is centered at the top, defining the screens purpose.

• Buttons: Three buttons are vertically aligned, with "Back" separated at the bot

tom for navigation consistency.

**UX Considerations:**

• Thelimited options (two report types) simplify the admins decision-making process,

reducing cognitive load.

• The "Back" button ensures easy navigation to the Admin Menu, maintaining a

seamless workflow.

• Thescreens minimal design focuses the admins attention on report generation tasks.

**Issue Book Screen:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Description**: The Issue Book Screen enables users to borrow books by selecting from

a list. It includes a text entry field for "User ID," a table listing available books (Book

ID, Book Name, Author, Publisher), and buttons for "Issue" and "Back."

**UI Elements:**

• Title: "Issue Book" is displayed at the top, indicating the screens function.

• Text Entry: A labeled "User ID" field allows users to input their identifier.

• Table: A table displays book details in a single-column format (simulated as a list

due to the use of tk.Listbox), with columns for Book ID, Book Name, Author,

and Publisher.

• Buttons: "Issue" and "Back" buttons are placed at the bottom, with "Issue" initiating the borrowing process and "Back" returning to the User Menu.

**UX Considerations:**

• The table format provides a clear overview of available books, though the tk.Listbox

limitation restricts it to a concatenated single-column display, which may reduce

readability for complex data.

• The “User ID" field ensures accurate user identification, supporting system security.

• The "Issue" buttons placement next to the table allows for a straightforward selection and action process, enhancing efficiency

**Book Records Screen:**

A screenshot of a computer

AI-generated content may be incorrect.

**Description**: The Book Records Screen, accessible to admins, displays a list of all

books in the system. It features a table with columns for Book ID, Book Name, Author,

and Publisher, and a "Back" button at the bottom.

**UI Elements:**

• Title: "Book Records" is centered at the top.

6

• Table: A table lists book details, implemented as a single-column display using

tk.Listbox, simulating columns for Book ID, Book Name, Author, and Publisher.

• Button: A "Back" button allows navigation to the previous screen.

**UX Considerations:**

• The table provides a comprehensive view of book records, though the tk.Listbox

limitation may reduce readability due to its single-column format.

• The "Back" button ensures admins can easily return to the previous menu, maintaining navigation flow.

• The screens focus on data display aligns with the admins need for quick access to

book information.

**Browse Books Screen:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Description**: The Browse Books Screen allows users to view all available books in

the library. It mirrors the Book Records Screen in structure, with a table listing Book

ID, Book Name, Author, and Publisher, and a "Back" button.

**UI Elements:**

• Title: "Browse Books" indicates the screens purpose.

• Table: A tk.Listbox-based display lists book details in a single-column format,

simulating a table with columns.

• Button: A "Back" button facilitates navigation.

**UX Considerations:**

• The screen provides users with a straightforward way to explore the librarys col

lection, though the single-column display may require additional scrolling.

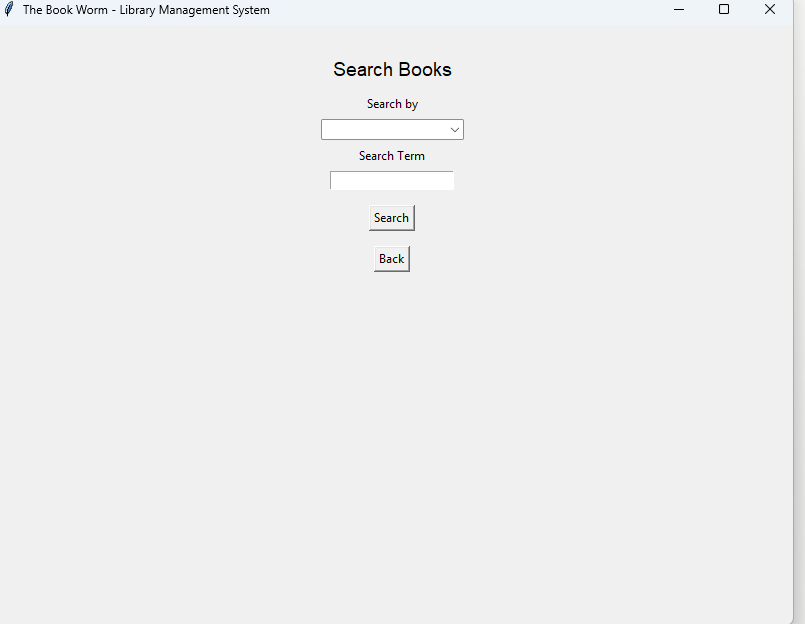
• The consistent "Back" button placement ensures ease of navigation, aligning with

the systems design standards.

• The design prioritizes functionality, ensuring users can quickly identify books of

interest.

**Search Books Screen:**



**Description**: The Search Books Screen enables users to search for books by specific

criteria. It includes a dropdown menu for selecting the search category (e.g., by title,

author), a text entry field for the search term, and "Search" and "Back" buttons.

**UI Elements:**

• Title: "Search Books" is displayed at the top.

• Dropdown Menu: A dropdown labeled "Search by" allows users to choose the

search criterion.

• Text Entry: A "Search Term" field accepts user input for the search query.

• Buttons: "Search" initiates the search, and "Back" returns to the User Menu.

**UX Considerations:**

• The dropdown menu simplifies the search process by providing predefined categories, reducing user errors.

• The clear labeling of the search term field ensures users understand the input requirement.

• The "Search" buttons placement after the input fields follows a logical workflow,

while the "Back" button maintains navigation consistency

The Library Management Systems UI/UX design effectively balances simplicity and functionality, ensuring that both admins and users can perform their tasks efficiently. The use of tk.Listbox for displaying tabular data, while limiting the interface to single column lists, maintains a minimalistic approach but may impact readability for complex data sets. The consistent placement of navigation buttons (e.g., "Back," "Logout") across screens enhances usability, and the system’s feedback mechanisms (e.g., success messages, error prompts) support a positive user experience. Future improvements could include adopting a more advanced widget for multi-column displays or enhancing visual design elements to improve aesthetics without sacrificing functionality.

**System Interface: Library Management System with MySQL Integration**

The Library Management System (LMS) is a desktop application designed to manage library operations, including book inventory, user accounts, book borrowing/returning, and administrative tasks. The system integrates a Tkinter-based graphical user interface (GUI) with a MySQL database backend to store and manage data persistently. This section describes the system interface, emphasizing the integration with MySQL, as implemented in the provided code.

**1. System Overview**

The LMS provides two primary user roles: Admin and User, each with distinct functionalities. The system uses a client-server architecture where the Tkinter GUI acts as the client, and the MySQL database serves as the backend for data storage and retrieval. The integration ensures that all user interactions (e.g., adding books, issuing books, submitting feedback) are reflected in the database, maintaining data consistency and integrity.

**2. System Components**

The system comprises the following key components:

Frontend (GUI): Built using Python's Tkinter library, providing an interactive interface for admins and users.

Backend (Database): MySQL database named "Library" storing tables for books, users, admins, and feedback.

Database Connector: The mysql-connector-python library facilitates communication between the Tkinter frontend and the MySQL backend.

Business Logic: Python functions within the LibraryApp class handle user inputs, validate data, and execute database queries.

3. MySQL Database Structure

The MySQL database ("Library") is initialized and managed through the init\_db() function. It consists of four tables:

**BookRecord**:

Purpose: Stores information about books in the library.

Columns:

BookID (varchar(10), Primary Key): Unique identifier for each book.

BookName (varchar(50)): Title of the book.

Author (varchar(30)): Author of the book.

Publisher (varchar(30)): Publisher of the book.

Category (varchar(20)): Genre or category of the book.

Price (DECIMAL(10,2)): Price of the book.

Initialization: If empty, the table is populated with 30 predefined books, each with a unique BookID, name, author, publisher, category, and price.

UserRecord:

Purpose: Manages user account details and tracks borrowed books.

Columns:

UserID (varchar(10), Primary Key): Unique identifier for each user.

UserName (varchar(20)): Name of the user.

Password (varchar(20)): User's password (stored in plaintext for simplicity).

BookID (varchar(10), Foreign Key): ID of the book currently borrowed (nullable).

BorrowDate (DATE): Date when the book was borrowed (nullable).

Fine (DECIMAL(10,2)): Accumulated fine for overdue books (default 0.00).

Initialization: Pre-populated with three users (101, 102, 103) with no initial borrowed books or fines.

AdminRecord:

Purpose: Stores admin credentials for system access.

Columns:

AdminID (varchar(10), Primary Key): Unique identifier for each admin.

Password (varchar(20)): Admin's password.

Initialization: Pre-populated with three admins (Kunal1020, Siddesh510, Vishal305).

Feedback:

Purpose: Stores user feedback and ratings.

**Columns**:

Feedback (varchar(100), Primary Key): User feedback text.

Rating (varchar(10)): Rating provided by the user (e.g., out of 10).

Initialization: Created empty, populated as users submit feedback.

The init\_db() function ensures the database and tables are created if they do not exist, verifies table structures, and adds necessary columns (Category, Price, BorrowDate, Fine) if missing. It also modifies the BookName column to varchar(50) if needed, ensuring compatibility with longer titles.

4. MySQL Integration with Tkinter

The integration between the Tkinter GUI and MySQL is achieved through the mysql-connector-python library, which enables the application to execute SQL queries and manage database transactions. The key aspects of this integration are:

**1)Database Connection:**

The init\_db() function establishes a connection to the MySQL server using: mydb = mysql.connector.connect(host="localhost", user="root", passwd="@Mamdouh2")

The connection is reused throughout the application via the global mydb and mycursor objects, ensuring efficient database access.

Error handling catches connection failures or unexpected errors, displaying appropriate messages and exiting the application if initialization fails.

**2)SQL Query Execution:**

The application uses parameterized queries to prevent SQL injection and ensure data security. For example: query = "INSERT INTO BookRecord (BookID, BookName, Author, Publisher, Category, Price) VALUES (%s, %s, %s, %s, %s, %s)"

mycursor.execute(query, (book\_id.get(), book\_name.get(), author.get(), publisher.get(), category.get(), float(price.get())))

Queries are executed for CRUD operations (Create, Read, Update, Delete) across all tables, such as inserting books, updating user fines, or retrieving feedback.

**3)Transaction Management:**

Database changes (e.g., inserting a book, issuing a book) are committed using mydb.commit() to ensure data persistence.

Rollbacks are not explicitly implemented, but MySQL's default behavior ensures data integrity in case of errors.

**4)Performance Monitoring:**

The application monitors query execution time using time.time(). If a database operation takes longer than 2 seconds, a warning is displayed: if time.time() - start\_time > 2:

messagebox.showwarning("Performance", "Response took longer than 2 seconds.")

**5. Interface Interactions with MySQL**

The Tkinter GUI provides various screens for admins and users, each interacting with the MySQL database to perform specific functions. Below are the key interactions:

**.Login Screen:**

Admin Login: Validates AdminID and Password against the AdminRecord table:

mycursor.execute("SELECT Password FROM AdminRecord WHERE AdminID=%s", (admin\_id.get(),))

**User** **Login**: Validates UserID and Password against the UserRecord table and checks for overdue books:( mycursor.execute("SELECT Password, UserID FROM UserRecord WHERE UserID=%s", (user\_id.get(),))

**Admin Menu:**

**.Book Management:**

Add Book: Inserts a new record into BookRecord with user-provided BookID, BookName, Author, Publisher, Category, and Price.

Delete Book: Removes a book from BookRecord based on BookID.

View Books: Retrieves all books from BookRecord, joining with UserRecord to show issued status:

python

mycursor.execute("SELECT BookRecord.BookID, BookRecord.BookName, BookRecord.Author, " +

"BookRecord.Publisher, BookRecord.Category, BookRecord.Price, " +

"UserRecord.UserName, UserRecord.UserID " +

"FROM BookRecord LEFT JOIN UserRecord ON BookRecord.BookID=UserRecord.BookID")

**.User Management:**

**Add Use**r: Inserts a new record into UserRecord.

Remove User: Deletes a user from UserRecord based on UserID.

View Users: Retrieves user details, including borrowed books, from UserRecord and BookRecord.

**Apply Penalty**: Updates the Fine column in UserRecord for a specified UserID.

**.Admin Management:**

Add Admin: Inserts a new record into AdminRecord.

View Admins: Retrieves all admin records from AdminRecord.

View Feedback: Displays all feedback from the Feedback table.

**Generate Reports:**

Book Usage Report: Counts the number of times each book has been issued:

python

mycursor.execute("SELECT BookRecord.BookID, BookRecord.BookName, " +

"COUNT(UserRecord.BookID) as TimesIssued " +

"FROM BookRecord LEFT JOIN UserRecord " +

"ON BookRecord.BookID=UserRecord.BookID " +

GROUP BY BookRecord.BookID, BookRecord.BookName")

**Overdue Fines Report**: Identifies overdue books, calculates days overdue, and computes fines:

mycursor.execute("SELECT UserRecord.UserID, UserRecord.UserName, UserRecord.BookID, " +

"BookRecord.BookName, UserRecord.BorrowDate " +

"FROM UserRecord INNER JOIN BookRecord " +

"ON UserRecord.BookID=BookRecord.BookID")

**User Menu:**

Search Books: Allows searching by Title or Category using a LIKE query:

query = "SELECT BookID, BookName, Author, Publisher, Category, Price FROM BookRecord WHERE " + search\_field + " LIKE %s"

mycursor.execute(query, ("%" + search\_term.get() + "%",))

**Browse Books**: Displays all books from BookRecord.

Issue Book: Updates UserRecord with the selected BookID and BorrowDate for the specified UserID, ensuring the book is not already issued and the user has no other borrowed books.

Return Book: Clears BookID and BorrowDate from UserRecord, calculates and applies any overdue fines:

mycursor.execute("SELECT BookID, BorrowDate, Fine FROM UserRecord WHERE UserID=%s AND BookID IS NOT NULL", (user\_id.get(),))

**View Issued Books**: Shows details of books borrowed by a specific user, joining UserRecord and BookRecord.

**Submit Feedback**: Inserts user feedback and rating into the Feedback table.

**Error Handling and Data Validation**

Database Errors: All database operations are wrapped in try-except blocks to catch mysql.connector.Error exceptions, displaying user-friendly error messages via messagebox.showerror.

Input Validation: Basic validation ensures required fields (e.g., BookID, UserID) are provided. For example, price inputs are converted to float to catch invalid formats.

Foreign Key Constraints: The BookID in UserRecord references BookRecord, ensuring only valid books can be issued.

Performance Checks: Slow queries trigger warnings, aiding in performance optimization.

**ScreenShot from mysql server connected with python**

A screenshot of a computer

AI-generated content may be incorrect.

**Chapter 6: Testing**

This chapter documents the test cases used to verify the functionality of the Library Management System. A total of 10 test cases were run, covering both database and GUI-level operations. Below are 5 representative test cases, including a negative test case.

**Database Connection**

Description: Verifies whether the database connection is active.

Expected Result: Should return a result from SELECT 1.

Actual Result: Connection was active; test passed.

Status: Pass

**Insert Book Record**

Description: Tests inserting a new book into the BookRecord table.

Expected Result: The inserted book should be retrievable from the database.

Actual Result: Book was successfully inserted and verified.

Status: Pass

**Issue Book via GUI (Positive Test)**

Description: Tests if a book can be issued successfully to a user through the GUI.

Expected Result: Book should be assigned to the user with correct borrow date, and a confirmation message should be shown.

Actual Result: Book issued successfully.

Status: Pass

**Issue Book When One Already Issued (Negative Test)**

Description: Attempts to issue another book to a user who already has one issued.

Expected Result: System should reject the request and show an error message.

Actual Result: Proper error message displayed: 'You already have a book issued. Return it first.'

Status: Pass

**Overdue Fine Calculation**

Description: Verifies fine calculation when a book is returned late (more than 14 days).

Expected Result: Fine should be calculated at $0.50 per day of delay.

Actual Result: Fine calculated correctly for 5 days overdue = $2.50.

Status: Pass

GitHub Update: The latest features and tests were committed and pushed to the GitHub repository. These include backend logic for issuing and returning books and GUI validations for overdue fines.

GitHub: <https://github.com/MamdouhKoritm/library-management-system>

**Chapter 7: Risk and Security Requirements**

This chapter outlines potential risks to the Library Management System and how to address them.

**Risk Identification**

|  |  |
| --- | --- |
| Risk Description | Category |
| SQL Injection in login or query forms | Security |
| Data loss due to uncommitted transactions or crashes | Technical |
| Unauthorized book issue/return by bypassing GUI | Operational |
| User data (e.g., passwords) stored in plaintext | Confidentiality |
| GUI crashing due to unexpected input or unhandled exceptions | Usability |

**Risk Assessment**

|  |  |  |  |
| --- | --- | --- | --- |
| Risk # | Likelihood | Impact | Classification |
| 1 | High | Very High | Critical Security Risk |
| 2 | Medium | High | Technical Risk |
| 3 | Medium | Medium | Operational Risk |
| 4 | High | Very High | Confidentiality Risk |
| 5 | Medium | Medium | Usability Risk |

**Mitigation Techniques**

|  |  |
| --- | --- |
| Risk # | Mitigation Strategy |
| 1 | Sanitize all user inputs; use parameterized SQL queries |
| 2 | Use transaction blocks with commit/rollback and regular backups |
| 3 | Restrict access using role-based authentication |
| 4 | Hash passwords using bcrypt or SHA-256 with salt |
| 5 | Implement try/except blocks and validate all user input types |

**Requirement Refinement**

Based on the identified risks, the following functional and non-functional requirements are refined or added:

**Functional Requirements**

- FR12: The system shall reject any login attempt with unsanitized inputs (Risk #1).

- FR13: Only authenticated admins can issue/return books via GUI or backend (Risk #3).

**Non-Functional Requirements**

- NFR6: Passwords shall be stored in hashed format using SHA-256 (Risk #4).

- NFR7: System shall display user-friendly error messages and prevent crashes from invalid inputs (Risk #5).

- NFR8: All SQL queries shall use prepared statements to prevent SQL injection (Risk #1).

- NFR9: The system shall ensure data consistency using transactions and commit operations (Risk #2).

**Additional Test Cases:**

**Return Book (Database Test)**

Description: Tests if a user can return a book successfully by clearing BookID and BorrowDate.

Expected Result: BookID and BorrowDate should be set to NULL; Fine remains unchanged.

Actual Result: Return successful with BookID and BorrowDate cleared.

Status: Pass

**Return Book via GUI (On Time)**

Description: Tests returning a book on time via the GUI without overdue fine.

Expected Result: System clears book assignment and confirms return with $0.00 fine.

Actual Result: Return completed with fine confirmation.

Status: Pass

**Return Book via GUI (Overdue)**

Description: Tests returning an overdue book via the GUI and calculating fine.

Expected Result: System calculates fine based on overdue days and updates user record.

Actual Result: Fine calculated and updated correctly.

Status: Pass

**View Issued Books (GUI)**

Description: Tests if issued books are correctly retrieved and displayed in the GUI.

Expected Result: Treeview should display current user's issued book record.

Actual Result: Correct data shown in the interface.

Status: Pass

**Fixed Mitigation Techniques Table:**

|  |  |
| --- | --- |
| Risk # | Mitigation Strategy |
| 1 | Sanitize all user inputs; use parameterized SQL queries |
| 2 | Use transaction blocks with commit/rollback and regular backups |
| 3 | Restrict access using role-based authentication |
| 4 | Hash passwords using bcrypt or SHA-256 with salt |
| 5 | Implement try/except blocks and validate all user input types |