Library Management System - Detailed Project Report

# 1. Introduction

The Library Management System (LMS) is designed to automate and streamline the core functions of a library. It allows librarians and users to manage the book catalog, issue and return books, and keep records of transactions digitally.

## 1.1 Problem Statement

Manual record-keeping in libraries often leads to misplaced books, untracked borrowings, and time-consuming searches. The lack of automation increases staff workload and reduces service efficiency.

## 1.2 Objectives

- Automate book lending and return processes.  
- Provide search functionality for available books.  
- Manage user accounts and permissions.  
- Generate transaction and inventory reports.

## 1.3 Scope

The system is designed for educational institutes and small public libraries. It will manage up to 10,000 book entries and 1,000 users. Features like book reservation, fine calculation, and overdue alerts are included.

## 1.4 Methodology

The LMS is developed using the waterfall model. Requirements are collected, followed by system design, implementation, testing, and deployment. Python and SQLite are chosen for simplicity and flexibility.

# 2. Literature Survey / Existing System

Current library systems vary from manual registers to full-fledged ERP software. Most advanced systems are not suitable for small libraries due to high cost and maintenance complexity.

## 2.1 Existing Systems

- Koha: Open-source, widely used in universities.  
- NewGenLib: Enterprise-level but resource-intensive.  
- Manual systems: prone to human error and duplication.

## 2.2 Gap Analysis

Existing systems either offer too many features or require significant infrastructure. The proposed LMS balances usability and efficiency, making it ideal for smaller institutions.

# 3. Software Requirement Analysis

The LMS has both functional and non-functional requirements, ensuring the system performs effectively and meets user expectations.

## 3.1 Functional Requirements

- User authentication and role management.  
- Add, edit, delete book entries.  
- Track issue and return dates.  
- View user history and overdue books.  
- Generate inventory reports.

## 3.2 Nonfunctional Requirements

- System should respond within 2 seconds.  
- Secure password storage using hashing.  
- User-friendly interface with tooltips and help.  
- Compatibility with Windows and Linux.

# 4. Software Design

The design of the LMS emphasizes modularity, allowing each component to function independently while interacting through defined interfaces.

## 4.1 System Architecture

Follows a three-tier model:  
- Presentation Layer (GUI)  
- Logic Layer (Backend code)  
- Data Layer (SQLite database)

## 4.2 Control Flow

User logs in → Chooses operation → Backend processes request → Data is fetched or updated → Result is shown.

## 4.3 Data Flow Diagrams

Includes Level 0 and Level 1 DFDs for processes like book issuing and returning. These show how data moves between modules and database.

# 5. Proposed System

The LMS will be a desktop-based or web-based application, tailored for simplicity and reliability. It will allow real-time inventory updates and user-based actions.

## 5.1 Module Description

- Login Module: Differentiates admin and user.  
- Book Inventory Module: Manages books.  
- Issue/Return Module: Tracks transactions.  
- Reporting Module: Generates logs and analytics.

## 5.2 Features

- Search by title, author, genre.  
- Auto-due date calculation.  
- Alerts for overdue books.  
- Backup and restore functionality.

# 6. Coding

The project uses Python with Tkinter for GUI and SQLite for the backend. Code is structured with classes and reusable functions.

## 6.1 Technologies Used

- Frontend: Python Tkinter  
- Backend: SQLite  
- Language: Python 3.x

## 6.2 Sample Code

def return\_book(user\_id, book\_id):  
 if is\_book\_issued(user\_id, book\_id):  
 update\_book\_status(book\_id, 'available')  
 log\_transaction(user\_id, book\_id, 'return')  
 return 'Return successful'  
 return 'Book not issued'

# 7. Testing

Both manual and automated tests were conducted to ensure data accuracy and functional completeness.

## 7.1 Black Box Testing

Focuses on system behavior. Example: Test login with wrong credentials → Expected: 'Login Failed' → Actual: 'Login Failed' → Pass.

## 7.2 White Box Testing

Involves code-level validation. Example: Function for book availability is tested for multiple user roles and edge cases.

# 8. Output Screens / Results

The system's outputs include visual screens for:  
- Login  
- Dashboard  
- Book List  
- Issue and Return Page  
- Report Generation  
These outputs ensure that all modules are integrated and operational.

# 9. Conclusion and Further Work

The LMS project successfully digitizes library processes, increasing efficiency and accuracy. It is scalable and can be adapted for different institutions.

## 9.1 Limitations

- Not accessible via internet unless hosted.  
- Limited reporting formats.  
- No integration with external devices like barcode readers.

## 9.2 Future Enhancements

- Add cloud synchronization.  
- Mobile app version.  
- Barcode/QR code integration.  
- Fine and fee collection module.

# 10. References

- Library Management System – Wikipedia  
- Software Engineering – Pressman  
- Python SQLite3 Documentation  
- Tkinter GUI Tutorial – GeeksforGeeks  
- Open-source LMS platforms for comparison