Library Management System – Project Report

Author: Abdul Mateen
Degree Program: Bachelor of Science in Information Security Engineering Technology (BS-ISET)
University: Government College University Lahore
Date: December 2025
Version: 1.0

Table of Contents

1. Introduction

The **Library Management System (LMS)** is a desktop-based application developed to automate and simplify the management of library resources. Traditional manual systems are time-consuming, error-prone, and difficult to maintain. This system provides an efficient and reliable solution for managing book records, tracking available copies, and handling borrowing and returning operations.

The application is implemented in **C++** with a **Graphical User Interface (GUI)** using the **Fast Light Toolkit (FLTK)**. It focuses on efficiency, data persistence, modular design, and ease of use, making it suitable for small to medium-sized libraries.

2. Objectives of the System

The main objectives of the Library Management System are:

- To provide an easy-to-use interface for managing library records.
- To automate book addition, deletion, borrowing, and returning processes.
- To ensure persistent storage of data using file handling.
- To reduce manual errors and improve efficiency.
- To implement core programming concepts such as classes, dynamic memory allocation, and file I/O.

3. System Requirements

Software Requirements

- **Operating System:** Windows 10/11, Linux (Ubuntu 20.04+), or macOS
- **Programming Language:** C++ (C++17 standard)
- **Compiler:** G++ or Clang (version 7.0 or later)
- **GUI Library:** FLTK (Fast Light Toolkit) version 1.3.x or 1.4.x
- **Development Environment:** VS Code or any C++ supported IDE

Hardware Requirements

- **Processor:** Any modern 64-bit processor
- **RAM:** Minimum 512 MB (2 GB recommended)
- **Storage:** At least 50 MB of free disk space

4. System Design & Architecture

4.1 High-Level Architecture

The system follows a **three-tier architecture**:

1. **Presentation Layer (GUI):**
   Developed using FLTK, this layer handles user interaction and displays data. All interface-related code is separated from the core logic to ensure modularity.
2. **Business Logic Layer:**
   This layer contains the `Library` and `Book` classes. It enforces rules such as checking book availability before borrowing and ensuring valid operations.
3. **Data Layer:**
   Responsible for data persistence using text files. All records are saved and loaded from files to maintain consistency across sessions.

4.2 Data Structures

The `Library` class uses a **dynamic array** to store objects of the `Book` class.

- **Initial Capacity:** 10 books
- **Dynamic Resizing:** When the array is full, its capacity is doubled
- **Advantages:**

    - Efficient memory usage
    - Fast access to elements
    - Demonstrates core C++ memory management concepts

4.3 File Handling Mechanism

The system uses **text file handling** for data storage.

- **File Name:** `books.txt`

Storage Format:
ID|Title|Author|Category|AvailableCopies

- **Loading Data:** Data is read from the file at application startup.
- **Saving Data:** After every update operation (add, delete, borrow, return), data is immediately written back to the file.

This approach ensures data persistence and prevents data loss.

5. Features & Functionality

- **Add Book:** Add new books with title, author, category, and number of copies.
- **Borrow Book:** Decrease the available copies when a book is borrowed.
- **Return Book:** Increase the available copies when a book is returned.
- **Delete Book:** Permanently remove a book record using its unique ID.
- **View Books:** Display a complete list of all books with relevant details.

6. Implementation Details

6.1 Technologies Used

- **Programming Language:** C++17
- **GUI Framework:** FLTK
- **Libraries Used:**
    - `<iostream>`
    - `<fstream>`
    - `<string>`
    - `<filesystem>`

6.2 Core Logic: Dynamic Array Resizing

To handle an increasing number of books efficiently, the system implements dynamic array resizing:

```
if (size == capacity) {
    capacity *= 2;
```

```
    Book* temp = new Book[capacity];
    for (int i = 0; i < size; i++) {
        temp[i] = books[i];
    }
    delete[] books;
    books = temp;
}
```

This ensures smooth performance while maintaining memory efficiency.

7. Testing & Validation

The system was tested using multiple test cases to ensure reliability:

- Valid book addition and deletion
- Borrowing a book with available copies
- Preventing borrowing when no copies are available
- Returning books correctly
- File loading and saving accuracy

All test cases passed successfully, confirming correct system behavior.

8. Limitations

- No user authentication or role management
- Uses text files instead of a database
- Designed for single-user access only
- Limited search and filtering options

9. Conclusion & Future Enhancements

The Library Management System is a reliable and efficient desktop application that fulfills its intended objectives. It demonstrates strong use of object-oriented programming, dynamic memory allocation, file handling, and GUI development in C++.

Future Enhancements

- Implement user login and role-based access
- Add advanced search and filtering features
- Integrate a database such as SQLite
- Generate analytical reports for library usage
- Improve UI design and usability