**Library MANAGEMENT SYSTEM**

**A PROJECT REPORT**

Submitted by : - vishwas

Date :-20-8-2024

**TABLE OF CONTENT**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPIC** | **PAGE NO** |
| 1 | Introduction | **3-4** |
| 2 | Objective | **5** |
| 3 | Tools/Environment | **8** |
| 4 | Program code | **9-20** |
| 5 | Input and output screen | **21-23** |
| 6 | Limitations of the project | **24-25** |
| 7 | Future applications of this project | **26-28** |
| 8 | biblography | **29** |

**INTRODUCTION**

The Library Management System (LMS) is a software application designed to help libraries manage their books and the various transactions associated with them. This system provides a simple yet effective way to handle the everyday tasks of a library, such as adding new books, keeping track of book availability, and managing borrowing and returning processes.

Developed using C++, this console-based application is tailored for smaller libraries or educational institutions that need a straightforward tool for managing their collections without the need for complex and expensive software solutions. It utilizes basic file handling to store data persistently, ensuring that all records are accurately maintained even after the application is closed.

****

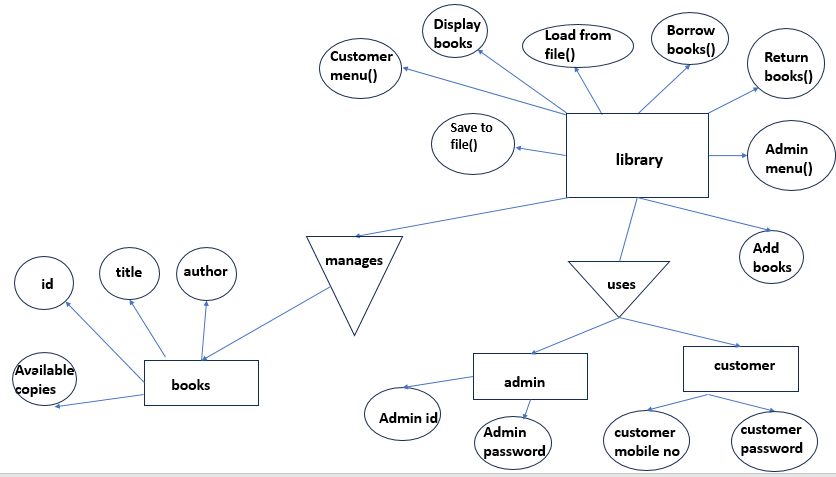
**Key Features**

1. Book Management:
   * Allows the addition of new books to the library's collection by storing essential details such as the book ID, title, author, and the number of available copies.
   * Supports updating the inventory when books are borrowed or returned, ensuring real-time accuracy in book availability.
2. User-Friendly Interface:
   * Offers a clear and intuitive menu-driven interface for both librarians (administrators) and customers (users), making it easy to navigate through the system.
   * Provides separate menus for administrators and customers, catering to the specific needs of each user type.
3. Borrowing and Returning Books:
   * Enables users to borrow books by checking their availability and decrementing the available copies count.
   * Allows users to return books, automatically updating the inventory by incrementing the available copies.
4. Persistent Data Storage:
   * Utilizes simple file handling to save and load book data, ensuring that all changes are saved between sessions.
   * Data is stored in a plain text file format, making it easy to view and edit outside the application if necessary.
5. Basic Security Features:
   * Implements basic authentication for both customers and administrators to protect the system from unauthorized access.
   * Uses hardcoded credentials for demonstration purposes, which can be modified to meet more secure needs.

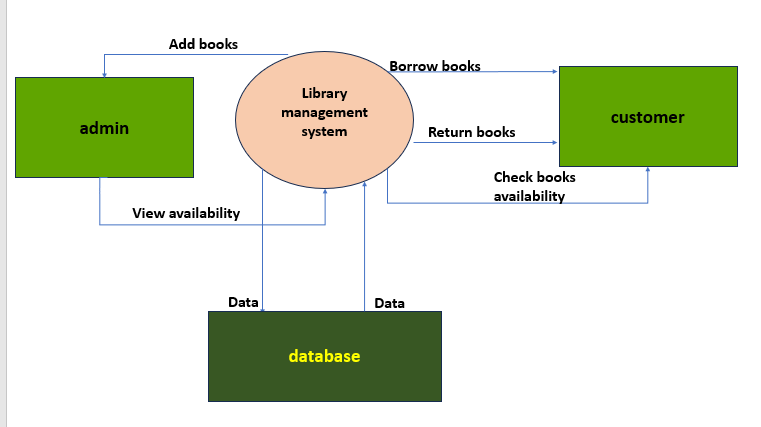
**OBJECTIVE**

* **Efficient Book Management**:
* To provide a system for adding, updating, and managing book details, including the book ID, title, author, and the number of available copies. This objective ensures the library’s inventory is always accurate and up-to-date.
* **Streamline Borrowing and Returning Processes**:
* To automate the borrowing and returning of books, making these processes quicker and more reliable. The system keeps track of all borrowed and returned books, ensuring that book availability information is always current.
* **Enhance Accessibility for Users**:
* To offer a user-friendly interface that allows both administrators and customers to easily navigate and perform their tasks. This objective aims to reduce the learning curve and improve user experience, regardless of the user's technical expertise.
* **Maintain Accurate Records with Data Persistence**:
* To implement simple file handling techniques to save and retrieve book data, ensuring that all information is stored persistently and is not lost between sessions. This objective supports consistent data management and retrieval, contributing to the overall reliability of the system.
* **Implement Basic Security Measures**:
* To ensure basic security by requiring authentication for access to the system's functions, protecting sensitive data from unauthorized users. This objective aims to provide a secure environment for managing library operations.

**ER DIAGRAM**



**DATA FLOW DIAGRAM**



**TOOLS AND ENVIORNMENT**

HARDWARE REQUIREMENTS

Processor: Minimum Pentium IV 2.4 GHZ

RAM: At Least 100 MB

Disk Space: At Least 500 MB

SOFTWARE REQUIREMENTS

Operating System: Windows,IOS,LINUX,Etc.

Code Compiler : Visual Code Studio / Dev C++/ Turbo C++/Etc.

**PROGRAM CODE**

#include <iostream>

#include <string>

#include <fstream>

using namespace std;

class Book {

public:

int id;

string title;

string author;

int availableCopies;

void addBook(int bookId, string bookTitle, string bookAuthor, int copies) {

id = bookId;

title = bookTitle;

author = bookAuthor;

availableCopies = copies;

}

void displayBook() {

cout << "Book ID: " << id << endl;

cout << "Title: " << title << endl;

cout << "Author: " << author << endl;

if (availableCopies > 0) {

cout << "Status: Available (" << availableCopies << " copies)" << endl;

} else {

cout << "Status: Not Available" << endl;

}

}

};

class Library {

public:

Book books[100];

int bookCount;

Library() {

bookCount = 0;

loadFromFile();

}

void saveToFile() {

ofstream outFile("library\_data.txt");

if (outFile.is\_open()) {

for (int i = 0; i < bookCount; i++) {

outFile << books[i].id << endl;

outFile << books[i].title << endl;

outFile << books[i].author << endl;

outFile << books[i].availableCopies << endl;

}

outFile.close();

} else {

cout << "Unable to open file for writing." << endl;

}

}

void loadFromFile() {

ifstream inFile("library\_data.txt");

if (inFile.is\_open()) {

while (inFile >> books[bookCount].id && bookCount < 100) {

inFile.ignore(); // Ignore newline character after id

getline(inFile, books[bookCount].title);

getline(inFile, books[bookCount].author);

inFile >> books[bookCount].availableCopies;

inFile.ignore(); // Ignore newline character after availableCopies

bookCount++;

}

inFile.close();

}

}

void addBook() {

if (bookCount < 100) {

int id, copies;

string title;

string author;

cout << "Enter Book ID: ";

cin >> id;

cin.ignore(); // Ignore the newline character after entering the ID

cout << "Enter Book Title: ";

getline(cin, title);

cout << "Enter Book Author: ";

getline(cin, author);

cout << "Enter number of copies: ";

cin >> copies;

books[bookCount].addBook(id, title, author, copies);

bookCount++;

saveToFile();

cout << "Book added successfully!" << endl;

} else {

cout << "Library is full. Cannot add more books." << endl;

}

}

void displayBooks() {

if (bookCount == 0) {

cout << "No books available in the library." << endl;

} else {

for (int i = 0; i < bookCount; i++) {

books[i].displayBook();

cout << "----------------------" << endl;

}

}

}

void borrowBook() {

int id;

cout << "Enter Book ID to borrow: ";

cin >> id;

for (int i = 0; i < bookCount; i++) {

if (books[i].id == id) {

if (books[i].availableCopies > 0) {

books[i].availableCopies--;

saveToFile(); // Save changes to file

cout << "Book borrowed successfully!" << endl;

} else {

cout << "Book is not available." << endl;

}

return;

}

}

cout << "Book not found." << endl;

}

void returnBook() {

int id;

cout << "Enter Book ID to return: ";

cin >> id;

for (int i = 0; i < bookCount; i++) {

if (books[i].id == id) {

books[i].availableCopies++;

saveToFile(); // Save changes to file

cout << "Book returned successfully!" << endl;

return;

}

}

cout << "Book not found." << endl;

}

void adminMenu() {

int choice;

do {

cout << endl;

cout << "\*\*\*\*\*\*\*Admin Menu\*\*\*\*\*\*\*" << endl;

cout << "1. Add Book" << endl;

cout << "2. Display All Books" << endl;

cout << "3. Exit Admin Menu" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

addBook();

break;

case 2:

displayBooks();

break;

case 3:

cout << "Exiting Admin Menu..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 3);

}

void customerMenu() {

int choice;

do {

cout << endl;

cout << "\*\*\*\*\*\*\*Customer Menu\*\*\*\*\*\*\*" << endl;

cout << "1. Borrow Book" << endl;

cout << "2. Return Book" << endl;

cout << "3. Check Book Availability" << endl;

cout << "4. Exit Customer Menu" << endl;

cout << "Enter your choice: ";

cin >> choice;

switch (choice) {

case 1:

borrowBook();

break;

case 2:

returnBook();

break;

case 3:

displayBooks();

break;

case 4:

cout << "Exiting Customer Menu..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

} while (choice != 4);

}

};

int main() {

Library lib;

int userchoice;

string admin\_password;

string admin\_id;

string customer\_mobile\_no;

string customer\_password;

do {

cout << endl;

cout << "\*\*\*\*\*\*\*Library Management System\*\*\*\*\*\*\*" << endl;

cout << "1. Customer Menu" << endl;

cout << "2. Admin Menu" << endl;

cout << "3. Exit" << endl;

cout << "Enter your choice: ";

cin >> userchoice;

switch (userchoice) {

case 1: {

do {

cout << endl << "Enter your mobile number: ";

cin >> customer\_mobile\_no;

if (customer\_mobile\_no.length() != 10) {

cout << "Mobile number must be exactly 10 digits long. Please try again." << endl;

}

} while (customer\_mobile\_no.length() != 10);

cout << "Enter your password: ";

cin >> customer\_password;

if (customer\_password == "223") {

lib.customerMenu();

} else {

cout << "Invalid mobile number or password!" << endl;

}

break;

}

case 2: {

cout << endl << "Enter admin ID: ";

cin >> admin\_id;

cout << "Enter admin password: ";

cin >> admin\_password;

if (admin\_id == "vishwas" && admin\_password == "223") {

lib.adminMenu();

} else {

cout << "Invalid admin ID or password!" << endl;

}

break;

}

case 3:

cout << "Exiting the program..." << endl;

break;

default:

cout << "Invalid choice. Please try again." << endl;

}

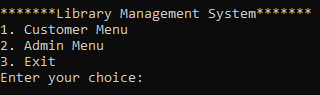
} while (userchoice != 3);

return 0;

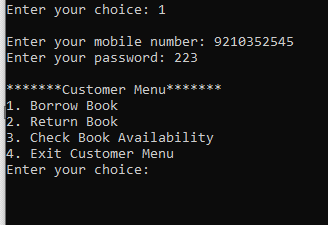
}

**INPUT/OUTPUT SCREEN**

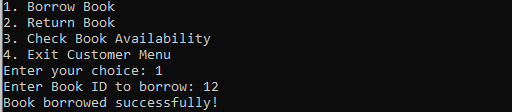
**1** Main menu



**2** Customer menu



**3** borrow book



4 return book



**5** admin login

****

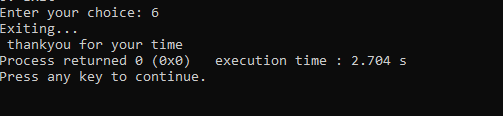
**6** add book



7 display books



7 exit



**Limitations of this project**

* **Limited Scalability**:
* The current system is designed to handle a maximum of 100 books. For larger libraries with more extensive collections, the system may need to be modified or restructured to accommodate a larger inventory.
* **Basic Security Measures**:
* The system uses simple password authentication without advanced security features, such as encryption or role-based access control. This could be insufficient for environments that require a higher level of data security and user privacy.
* **No Support for Multiple Users**:
* The system is designed for single-user operation at a time, meaning it does not support multiple concurrent users or have a networked or web-based interface. This could limit usability in a multi-user environment.
* **Manual Data Management**:
* Data for books and transactions are managed using basic file handling without a database. This approach limits the ability to efficiently search, sort, or filter records and may lead to slower performance as the data size grows.
* **No Advanced Reporting or Analytics**:
* The system lacks features for generating advanced reports or analytics, such as popular books, borrowing trends, or user activity. This could be a drawback for libraries that require detailed insights into their operations.
* **Limited User Interface**:
* The system features a basic console-based interface that may not be intuitive or visually appealing for all users, particularly those who are accustomed to graphical user interfaces (GUIs) with more advanced features.
* **No Fine or Overdue Management**:
* The system does not include features for managing fines for overdue books or reminding users of due dates, which could be important for maintaining proper library discipline and ensuring book availability.
* **No Backup and Recovery Mechanisms**:
* The current design lacks built-in features for automatic data backup and recovery. In case of data corruption or accidental deletion, there may be a risk of losing important library information.
* **Limited Customization and Flexibility**:
* The system is designed with predefined functionalities and may require significant code changes to add new features or adapt to specific library needs, making it less flexible for different use cases.
* **Lack of Advanced Book Search and Filtering**:
* Users can only browse books in the order they are stored without advanced search capabilities, such as searching by keyword, genre, or publication date, limiting the ease of finding specific books quickly.

**Future applications of this project**

* **Integration with Digital Libraries**:
* Extend the system to integrate with digital libraries and online resources, allowing users to access e-books, audiobooks, and other digital materials directly through the system. This integration could also include access to digital subscriptions, journals, and research databases.
* **Web-Based Interface**:
* Develop a web-based version of the LMS that allows users to access the library catalog, borrow and return books, and manage their accounts online. A web-based system would support multiple concurrent users and provide remote access from any device with internet connectivity.
* **Mobile Application Development**:
* Create a mobile app for Android and iOS devices to enhance user convenience. This app could allow users to search the library catalog, reserve books, receive notifications, and manage their accounts directly from their smartphones or tablets.
* **Advanced Search and Filtering Options**:
* Implement advanced search capabilities, such as searching by keyword, genre, publication date, or author, and filtering options to help users quickly find specific books or resources.
* **Automated Notifications and Reminders**:
* Introduce automated email or SMS notifications to remind users of due dates, overdue books, or reserved books that are available for pickup. This feature could help reduce overdue fines and improve user engagement.
* **Integration with RFID and Barcode Systems**:
* Integrate RFID (Radio-Frequency Identification) and barcode scanning technology to automate the process of checking in and out books. This would speed up transactions, reduce errors, and provide real-time updates to the inventory system.
* **Data Analytics and Reporting Tools**:
* Develop advanced data analytics and reporting tools to provide insights into library usage patterns, popular books, borrowing trends, and user demographics. This information could help librarians make data-driven decisions regarding inventory management and user engagement strategies.
* **Enhanced Security and User Authentication**:
* Implement stronger security measures, such as multi-factor authentication (MFA), encryption, and role-based access control, to protect sensitive user data and ensure that only authorized users have access to certain features or data.
* **Cloud-Based Library Management**:
* Move the system to a cloud-based platform, allowing for easier scalability, centralized data management, and improved access from various locations. A cloud-based system could also offer better data backup and disaster recovery options.
* **Support for Multimedia and Special Collections**:
* Expand the system to support multimedia resources (such as videos, images, and music) and special collections (like rare books and archival materials). This feature could help libraries manage a broader range of materials and provide enhanced access to unique resources.

**BIBLOGRAPHY**

Books

Let Us C by Yashavant Kanetkar.

Let us C++ by Yashavant Kanetkar.

C in Depth by S.K Srivastava.

The C++ Programming Language By Bjarne Stroustrup.

Websites

[www.google.com](http://www.google.com)

[www.youtube.com](http://www.youtube.com)

[www.w3schools.com](http://www.w3schools.com)

[www.geeksforgeeks.com](http://www.geeksforgeeks.com)