Library Management System

TECHNICAL REPORT

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
Bachelor of Technology
in Computer Science and Engineering

Submitted by
Balkrishan Singh
URN 2302492
Diljot Singh
URN 2302509
Ayush Mishra
URN 2302490

Submitted To
ER.Shailja

Department of Computer Science and Engineering
Guru Nanak Dev Engineering College
Ludhiana, 141006

Chapter 1

main.cpp

```cpp
#include <iostream>
#include "library.h"

int main()
{

    library lib;
    std::cout << "Welcome to Library management System! ";
    char choice;
    std::cout << "Registered Already? (y/n)";
    std::cin >> choice;
    toupper(choice);
    if (choice == 'y')
    {
        Menu::Login(lib);
    }
    else if (choice == 'n')
    {
        Menu::Registration(lib);
        Menu::Login(lib);
    }
    return 0;
}
```

Chapter 2

Library.h

```

#ifndef LIBRARY_H
#define LIBRARY_H
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include <memory>
class Book;
class User;
class library;

class Book {
    bool isIssued;
    std::string bookName;
    std::string author;
    int bookID;

public:
    std::shared_ptr<User> bookBorrower;
    Book(int bookID, std::string bookName, std::string author);

    Book(std::ifstream &inFile);

    void Save(std::ofstream &outFile);

    void BookInformation();

    int getBookID() {
        return bookID;
    }

    bool isBookIssued() const { return isIssued; }
    void setIssued(bool status) { isIssued = status; }
};
```

```cpp

class User {
protected:
    std::string userName;
    int password;

    User(std::string userName, int password);

public:
    virtual ~User() = default;

    std::string getUserName() const { return userName; }
    int getPassword() const { return password; }
};

class Student : public User {
    int userID; //Roll no
    //TODO Save which book has been borrowed.
    std::shared_ptr<Book> borrowedBook;
    int borrowedBookID;

public:
    void SaveStudent(std::ofstream &outFile);

    Student(int userID, std::string userName, int password);

    void DisplayIssuedBook();

    std::shared_ptr<Book> getBorrowedBook() const { return borrowedBook; }

    void setBorrowedBook(std::shared_ptr<Book> borrowedBook) {
        this->borrowedBook = borrowedBook;
    }

    bool hasIssuedBook() {
        return borrowedBook != nullptr;
    }

    Student(std::ifstream &inFile);

    void studentInformation();
};

class FileManager {
public:
    static void SaveBooks(const std::vector<std::shared_ptr<Book> > &books, const std

    static void LoadBooks(std::vector<std::shared_ptr<Book> > &books, const std::stri

    static void SaveUsers(const std::vector<std::shared_ptr<User> > &users, const std

    static void LoadUsers(std::vector<std::shared_ptr<User> > &users, const std::stri
};

class Administrator : public User {
public:
    Administrator(std::string userName, int password);
};

class library {
    std::shared_ptr<Administrator> administrator;
    std::vector<std::shared_ptr<User> > users;
    std::vector<std::shared_ptr<Book> > books;

public:
    library();

    ~library();

    void RegisterStudent(int userID, std::string userName, int password);

    void LoginUser(bool isAdmin);

    void issueBook(std::shared_ptr<Student> student);

    void addBook();

    void displayBooks();

    void searchBook();

    void displayIssuedBooksForStudent();

    void displaystudents();

    void returnBook(std::shared_ptr<Student> student);

    void AddDummyStudentsToBinaryFile();

    void AddDummyBooksToBinaryFile();
};

class Menu {
public:
    static void Registration(library &lib);

```

```cpp
    static void Login(library &lib);

    static void StudentDashboard(library &lib, std::shared_ptr<Student> activeStudent

    static void AdministratorDashboard(library &lib);
};

#endif // LIBRARY_H

```

Chapter 3

library.cpp

```

#include "library.h"
#include "string"
#include <iostream>

void library::AddDummyBooksToBinaryFile()
{
    books.push_back(std::make_shared<Book>(101, "C++ Programming", "Bjarne Stroustrup
    books.push_back(std::make_shared<Book>(102, "Introduction to Algorithms", "Thomas
    books.push_back(std::make_shared<Book>(103, "Clean Code", "Robert C. Martin"));
    books.push_back(std::make_shared<Book>(104, "The Pragmatic Programmer", "Andrew H
    books.push_back(std::make_shared<Book>(105, "Design Patterns", "Erich Gamma"));
}

// TODO Add input validation for username (Capitalization ignore).
void library::AddDummyStudentsToBinaryFile()
{
    // Add some dummy students
    users.push_back(std::make_shared<Student>(101, "Alice", 1234));
    users.push_back(std::make_shared<Student>(102, "Bob", 5678));
    users.push_back(std::make_shared<Student>(103, "Charlie", 91011));
}

library::library()
{
    administrator = std::make_shared<Administrator>("Diljot", 1234);
    FileManager::LoadBooks(books, "./books.dat");
    FileManager::LoadUsers(users, "./students.dat");
}

library::~library()
{
    FileManager::SaveBooks(books, "./books.dat");
    FileManager::SaveUsers(users, "./students.dat");
}
```

```

Book::Book(int bookID, std::string bookName, std::string author)
{
    this->bookID = bookID;
    this->bookName = bookName;
    this->author = author;
}

void Book::BookInformation()
{
    std::cout << "Book ID: " << bookID
              << "\nTitle: " << bookName
              << "\nAuthor: " << author
              << "\nIssued: " << (isIssued ? "Yes" : "No") << '\n';
}

User::User(std::string userName, int password)
{
    this->userName = userName;
    this->password = password;
}

Student::Student(int userID, std::string userName, int password) : User(userName, pas
{
    this->userID = userID;
}

Administrator::Administrator(std::string userName, int password) : User(userName, pas
{
}

void FileManager::SaveBooks(const std::vector<std::shared_ptr<Book>> &books, const st
{
    std::ofstream outFile(filename, std::ios::binary);
    if (!outFile)
    {
        std::cerr << "Error opening file for saving.\n";
        return;
    }

    size_t count = books.size();
    outFile.write(reinterpret_cast<const char *>(&count), sizeof(count));

    for (const auto &book : books)
    {
        book->Save(outFile);
    }

```

```cpp
    if (!outFile)
    {
        std::cerr << "Error writing to file!\n";
    }
}

void FileManager::LoadBooks(std::vector<std::shared_ptr<Book>> &books, const std::str
{
    std::ifstream inFile(filename, std::ios::binary);
    if (!inFile)
    {
        std::cerr << "Error opening file for loading book data.\n";
        return;
    }
    size_t count;
    inFile.read(reinterpret_cast<char *>(&count), sizeof(count));
    books.clear();
    for (size_t i = 0; i < count; ++i)
    {
        auto book = std::make_shared<Book>(inFile);
        books.push_back(book);
    }
    if (!inFile)
    {
        std::cerr << "Error reading from book data file.!\n";
    }
}

void FileManager::SaveUsers(const std::vector<std::shared_ptr<User>> &users, const st
{
    std::ofstream outFile(filename, std::ios::binary);
    if (!outFile)
    {
        std::cerr << "Error opening file for saving users.\n";
        return;
    }

    size_t count = users.size();
    outFile.write(reinterpret_cast<const char *>(&count), sizeof(count));

    for (const auto &user : users)
    {
        if (auto student = std::dynamic_pointer_cast<Student>(user))
        {
            student->SaveStudent(outFile);
        }
    }

    if (!outFile)
    {
        std::cerr << "Error writing to user data file!\n";
    }
}

void FileManager::LoadUsers(std::vector<std::shared_ptr<User>> &users, const std::str
{
    std::ifstream inFile(filename, std::ios::binary);
    if (!inFile)
    {
        std::cerr << "Error opening file for loading users.\n";
        return;
    }

    size_t count;
    inFile.read(reinterpret_cast<char *>(&count), sizeof(count));

    users.clear();
    for (size_t i = 0; i < count; ++i)
    {
        auto student = std::make_shared<Student>(inFile);
        users.push_back(student);
    }

    if (!inFile)
    {
        std::cerr << "Error reading from user data file!\n";
    }
}

Student::Student(std::ifstream &inFile) : User("", 0)
{
    size_t nameSize;
    inFile.read(reinterpret_cast<char *>(&nameSize), sizeof(nameSize));
    userName.resize(nameSize);
    inFile.read(&userName[0], nameSize);

    inFile.read(reinterpret_cast<char *>(&password), sizeof(password));

    inFile.read(reinterpret_cast<char *>(&userID), sizeof(userID));
    //
}

Book::Book(std::ifstream &inFile)
{
    inFile.read(reinterpret_cast<char *>(&bookID), sizeof(bookID));

    size_t nameSize;
    inFile.read(reinterpret_cast<char *>(&nameSize), sizeof(nameSize));
    bookName.resize(nameSize);
    inFile.read(&bookName[0], nameSize);

    size_t authorSize;
    inFile.read(reinterpret_cast<char *>(&authorSize), sizeof(authorSize));
    author.resize(authorSize);
    inFile.read(&author[0], authorSize);

    inFile.read(reinterpret_cast<char *>(&isIssued), sizeof(isIssued));
}

void Book::Save(std::ofstream &outFile)
{

    outFile.write(reinterpret_cast<const char *>(&bookID), sizeof(bookID));

    size_t nameSize = bookName.size();
    outFile.write(reinterpret_cast<const char *>(&nameSize), sizeof(nameSize));
    outFile.write(bookName.c_str(), nameSize);

    size_t authorSize = author.size();
    outFile.write(reinterpret_cast<const char *>(&authorSize), sizeof(authorSize));
    outFile.write(author.c_str(), authorSize);

    outFile.write(reinterpret_cast<const char *>(&isIssued), sizeof(isIssued));
}

void Student::DisplayIssuedBook()
{
    if (borrowedBook)
    {
        std::cout << "Issued Book Details:\n";
        borrowedBook->BookInformation();
    }
    else
    {
        std::cout << "No book has been issued to you.\n";
    }
}

void library::RegisterStudent(int userID, const std::string userName, int password)
{
    auto student = std::make_shared<Student>(userID, userName, password);
    users.push_back(student);
    std::cout << "Student registered successfully.\n";
}
```

```cpp

void library::LoginUser(bool isAdmin)
{
    std::string userName;
    int password;
    std::cout << "Enter Username: ";
    std::cin.ignore();
    getline(std::cin, userName);
    std::cout << "Enter Password: ";
    std::cin >> password;

    if (!isAdmin)
    {
        for (const auto &user : users)
        {
            if (dynamic_cast<Student *>(user.get()))
            {
                if (user->getUserName() == userName && user->getPassword() == passwor
                {
                    std::cout << "Student login successful!\n";
                    Menu::StudentDashboard(*this, std::dynamic_pointer_cast<Student>(
                    return;
                }
            }
        }
    }
    else
    {
        if (administrator->getUserName() == userName && administrator->getPassword()
        {
            std::cout << "Admin login successful!\n";
            Menu::AdministratorDashboard(*this);
            return;
        }
    }
    std::cout << "Invalid credentials!\n";
}

void library::issueBook(std::shared_ptr<Student> student)
{
    int bookID;
    std::cout << "Enter Book ID to issue: ";
    std::cin >> bookID;
    if (!student->hasIssuedBook())
    {
        for (auto &book : books)
        {
            if (book->getBookID() == bookID && !(book->isBookIssued()))
```

```cpp
                {
                    book->setIssued(true);
                    student->setBorrowedBook(book);
                    std::cout << "Book issued successfully.\n";
                    return;
                }
            }
        }

        std::cout << "Book not available.\n";
}

void library::returnBook(std::shared_ptr<Student> student)
{
    if (student->hasIssuedBook())
    {
        student->getBorrowedBook()->setIssued(true);
        student->setBorrowedBook(nullptr);
        std::cout << "Book returned successfully.\n";
    }
    else
    {
        std::cout << "You don't have any issued books..\n";
    }
}

void library::addBook()
{
    int bookID;
    std::string bookName, author;

    std::cout << "Enter Book ID: ";
    std::cin >> bookID;
    std::cin.ignore();
    std::cout << "Enter Book Name: ";
    getline(std::cin, bookName);
    std::cout << "Enter Author Name: ";
    getline(std::cin, author);

    auto newBook = std::make_shared<Book>(bookID, bookName, author);
    books.push_back(newBook);

    std::cout << "Book added successfully.\n";
}

void library::displayBooks()
{
    std::cout << "Books in the library:\n";
    for (const auto &book : books)
    {
        book->BookInformation();
    }
}

void library::displaystudents()
{
    for (const auto &user : users)
    {
        std::dynamic_pointer_cast<Student>(user)->studentInformation();
    }
}

void Student::studentInformation()
{
    std::cout << "Student ID: " << userID
              << "\nName: " << userName
              << "\nPassword: " << password << std::endl;
}

void Student::SaveStudent(std::ofstream &outFile)
{

    size_t nameSize = userName.size();
    outFile.write(reinterpret_cast<const char *>(&nameSize), sizeof(nameSize));
    outFile.write(userName.c_str(), nameSize);
    outFile.write(reinterpret_cast<const char *>(&password), sizeof(password));

    outFile.write(reinterpret_cast<const char *>(&userID), sizeof(userID));
}
// unused
void library::searchBook()
{
    int bookID;
    std::cout << "Enter Book ID to search: ";
    std::cin >> bookID;

    for (const auto &book : books)
    {
        if (book->isBookIssued())
        {
            book->BookInformation();
            return;
        }
    }

    std::cout << "Book not found.\n";
```

```cpp
}

void library::displayIssuedBooksForStudent()
{
    std::string studentName;
    std::cout << "Enter your username: ";
    getline(std::cin, studentName);

    for (const auto &user : users)
    {
        if (auto student = std::dynamic_pointer_cast<Student>(user))
        {
            if (student->getUserName() == studentName)
            {
                student->DisplayIssuedBook();
                return;
            }
        }
    }
    std::cout << "Student not found or no books issued.\n";
}

// Menu Class Definitions
void Menu::Registration(library &lib)
{
    std::cout << "Registration Menu\n1. Register as Student\n2. Exit\n";
    int choice;
    std::cin >> choice;

    if (choice == 1)
    {
        int userID, password;
        std::string userName;
        std::cout << "Enter User ID: ";
        std::cin >> userID;
        std::cin.ignore();
        std::cout << "Enter Username: ";

        getline(std::cin, userName);
        std::cout << "Enter Password: ";
        std::cin >> password;
        std::cin.ignore();
        lib.RegisterStudent(userID, userName, password);
    }
}

void Menu::Login(library &lib)
{
    std::cout << "Login Menu\n1. Student Login\n2. Admin Login\n3. Exit\n";
    int choice;
    std::cin >> choice;
    if (choice == 1)
    {
        lib.LoginUser(false);
    }
    else if (choice == 2)
    {
        lib.LoginUser(true);
    }
}

void Menu::StudentDashboard(library &lib, std::shared_ptr<Student> activeStudent)
{
    bool running = true;
    while (running)
    {
        std::cout << "Student Dashboard\n1.Display Books\n2.Issue Book\n3.Return Book
        int choice;
        std::cin >> choice;
        switch (choice)
        {
        case 1:
            lib.displayBooks();
            break;
        case 2:
            lib.issueBook(activeStudent);
            break;
        case 3:
            lib.returnBook(activeStudent);
            break;
        case 4:
            activeStudent->DisplayIssuedBook();
            break;

        default:
            running = false;
            break;
        }
    }
}

void Menu::AdministratorDashboard(library &lib)
{
    bool running = true;
    while (running)
    {
        std::cout << "Administrator Dashboard\n1. Add Book\n2. Display Users\n3. Disp
        int choice;
        std::cin >> choice;
        switch (choice)
        {
        case 1:
            lib.addBook();
            break;
        case 2:
            lib.displaystudents();
            break;
        case 3:
            lib.displayBooks();
            break;
        default:
            running = false;
        }
    }
}
```

Chapter 4

Outputs