KIET Group of Institutions, Ghaziabad

Report

on

Library Management System

January

(2024)

Submitted By: Piyush Raj

Course/Branch/Sem/Sec:

B-Tech (CSE) IIIrd B

Roll No: 2200290100114

Index of Report

Abstract

This project presents a Library Management System (LMS) developed in the C programming language, leveraging linked lists as a fundamental data structure. The system focuses on efficient user and book management through the implementation of key functionalities, including user login, user addition, book addition, and book issuance.

Utilizing linked lists, the LMS ensures dynamic and efficient storage of user and book information. The login functionality allows secure access for librarians, while the user addition feature enables easy registration and management of library patrons. The system also facilitates the addition of new books to the library inventory and the seamless issuance of books to users.

The project emphasizes a user-friendly interface, making the LMS accessible and intuitive for both librarians and patrons. The linked list implementation optimizes memory usage and enhances the speed of data retrieval, contributing to the overall efficiency of the system.

Through the integration of C language and linked lists, this Library Management System offers a scalable, organized, and robust solution for effective library administration and user services. The project showcases the prowess of C programming and linked list structures in developing a streamlined and functional software system for library operations.

OVERVIEW OF PROJECT

The concept of storing or recording the details of books embedded within the user's system is known as Library Management System. It details the type of books, the list of books, etc. Only a person with the login credentials can access the Library Management System. That person can perform many operations like adding the book details, removing the book details, displaying the book details, modifying the book details, etc.

We must use the Library Management System in order to have secured storage of book details contained within the Library (probably). This feature is generally enabled in order to protect the data, which is highly confidential. This is one of the simplest Management systems built within the system using the C programming language.

The Library Management System overviews the concept of storing and generating all the data or records of the book contained within the library. This can be known as a general database which stores the data of the book details. It helps in searching the details by reducing time consumption. Not only protects the details of the books in the library but also saves all the data up to date without missing any. This is the major benefit of the Library Management System.

The Library Management System, developed in C, safeguards and organizes book details within a secure database, accessible only with valid login credentials. This efficient system allows users to seamlessly perform operations such as adding, removing, displaying, and modifying book details. Its fundamental purpose lies in ensuring the confidentiality and accuracy of stored information, contributing to effective library management and timely data retrieval.

Key Features of Project

The Library Management System project boasts several key features designed to streamline and enhance library operations.

Some features are:-
1.Login Functionality: Secure access to the system is granted through user authentication.

2. User Management:

Add a User : Enables the addition of new users to the system.

Delete a User : Allows administrators to remove user profiles efficiently.
Book Management:

3. Add a Book : Facilitates the inclusion of new books in the library catalog.

4.Delete a Book : Permits the removal of outdated or irrelevant book entries.

5.View Books : Displays a comprehensive list of the library's collection, aiding in efficient navigation and search.

6.Book Issuance : Allows librarians to issue books, keeping track of borrowing transactions.

7.View Issued Books : Provides a detailed overview of borrowed items, enhancing accountability and monitoring.

8.Exit Option : Ensures a secure and seamless logout process for users.

These key features collectively contribute to an organized, user-friendly Library Management System, offering secure access, efficient user and book management, and comprehensive tracking of library transactions.

1. **Login Feature:-**

Securing system access, the login functionality requires user authentication, ensuring only authorized individuals can interact with the Library Management System.

Code:-

```
  switch (choice) {
            case '1':
                printf("Enter your username: ");
                scanf("%s", username);
                printf("Enter your password: ");
                scanf("%s", password);

                // if the user exists
                struct User* currentUser = userList;
                int isUserFound = 0;
                while (currentUser != NULL) {
                    if (strcmp(currentUser->username, username) == 0 &&
strcmp(currentUser->password, password) == 0) {
                        isUserFound = 1;
                        break;
                    }
                    currentUser = currentUser->next;
                }

                if (isUserFound) {
                    printf("Login successful. Welcome, %s!\n", username);
                } else {
                    printf("Login failed. Incorrect username or password.\n");
                }
                break;
```

2. User Management:

Add a User : Enables the addition of new users to the system.

Delete a User : Allows administrators to remove user profiles efficiently.
Book Management:

```
struct User* createUserNode(char* username, char* password) {
    struct User* newUser = (struct User*)malloc(sizeof(struct User));
    if (newUser == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    strcpy(newUser->username, username);
    strcpy(newUser->password, password);
    newUser->next = NULL;
    return newUser;
}
```

```c
void insertUser(struct User** head, struct User* newUser) {
    if (*head == NULL) {
        *head = newUser;
    } else {
        struct User* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newUser;
    }
}

//  delete a user from the linked list
void deleteUser(struct User** head, char* username) {
    struct User* current = *head;
    struct User* prev = NULL;

    while (current != NULL) {
        if (strcmp(current->username, username) == 0) {
            if (prev == NULL) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }

    printf("User '%s' not found in the system.\n", username);
}
```

3. Add a Book : Facilitates the inclusion of new books in the library catalog.

```c
// create a new book node
struct Book* createBookNode(int bookId, char* title, char* author) {
    struct Book* newBook = (struct Book*)malloc(sizeof(struct Book));
    if (newBook == NULL) {
        printf("Memory allocation failed.\n");
        exit(1);
    }
    newBook->bookId = bookId;
    strcpy(newBook->title, title);
    strcpy(newBook->author, author);
    newBook->isIssued = 0; // Initially, the book is not issued
    newBook->issueDate = 0; // Initialize issue date to 0
    newBook->next = NULL;
    return newBook;
```

```
}

// insert a new book into the linked list
void insertBook(struct Book** head, struct Book* newBook) {
    if (*head == NULL) {
        *head = newBook;
    } else {
        struct Book* current = *head;
        while (current->next != NULL) {
            current = current->next;
        }
        current->next = newBook;
    }
}
```

4.Delete a Book : Permits the removal of outdated or irrelevant book entries.

```
// delete a book from the linked list
void deleteBook(struct Book** head, int bookId) {
    struct Book* current = *head;
    struct Book* prev = NULL;

    while (current != NULL) {
        if (current->bookId == bookId) {
            if (prev == NULL) {
                *head = current->next;
            } else {
                prev->next = current->next;
            }
            free(current);
            return;
        }
        prev = current;
        current = current->next;
    }

    printf("Book with ID %d not found in the library.\n", bookId);
}
```

5.View Books : Displays a comprehensive list of the library's collection, aiding in efficient navigation and search.

```c
//  view the list of books
void viewBooks(struct Book* head) {
    struct Book* current = head;
    if (current == NULL) {
        printf("The library is empty.\n");
        return;
    }
    printf("List of books:\n");
    while (current != NULL) {
        printf("Book ID: %d\n", current->bookId);
        printf("Title: %s\n", current->title);
        printf("Author: %s\n", current->author);
        if (current->isIssued) {
            printf("Status: Issued\n");
            printf("Issue Date: %s", ctime(&current->issueDate));
        } else {
            printf("Status: Available\n");
        }
        printf("\n");
        current = current->next;
    }
}
```

6.Book Issuance : Allows librarians to issue books, keeping track of borrowing transactions.

```c
//  issue a book
void issueBook(struct Book* head, int bookId) {
    struct Book* current = head;

    while (current != NULL) {
        if (current->bookId == bookId) {
            if (current->isIssued) {
                printf("Book with ID %d is already issued.\n", bookId);
            } else {
                current->isIssued = 1;
                current->issueDate = time(NULL);

                printf("Book with ID %d has been issued.\n", bookId);
            }
            return;
        }
        current = current->next;
    }

    printf("Book with ID %d not found in the library.\n", bookId);
}
```

7.View Issued Books : Provides a detailed overview of borrowed items, enhancing accountability and monitoring.

Files Handling:-

```
// Function to save user data to file
void saveUsersToFile(struct User* head, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("\t\t\t\tError opening file for writing.\n");
        exit(1);
    }

    struct User* current = head;
    while (current != NULL) {
        fprintf(file, "%s %s\n", current->username, current->password);
        current = current->next;
    }

    fclose(file);
}

// Function to load user data from file
void loadUsersFromFile(struct User** head, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        return; // File doesn't
        return; // File doesn't exist or error opening file; assume no users
initially
    }

    char username[50];
    char password[50];

    while (fscanf(file, "%s %s", username, password) == 2) {
        struct User* newUser = createUserNode(username, password);
        insertUser(head, newUser);
    }

    fclose(file);
}

void saveBooksToFile(struct Book* head, const char* filename) {
    FILE* file = fopen(filename, "w");
    if (file == NULL) {
        printf("\t\t\t\tError opening file for writing.\n");
        exit(1);
    }

    struct Book* current = head;
```

```
    while (current != NULL) {
        fprintf(file, "%d,%s,%s,%d,%ld\n", current->bookId, current->title,
current->author, current->isIssued, current->issueDate);
        current = current->next;
    }

    fclose(file);
}


// Function to load book data from file
void loadBooksFromFile(struct Book** head, const char* filename) {
    FILE* file = fopen(filename, "r");
    if (file == NULL) {
        printf("\t\t\t\tError opening file for reading or file not found:
%s\n", filename);
        return;
    }

    int bookId;
    char title[100];
    char author[100];
    int isIssued;
    time_t issueDate;

    while (fscanf(file, "%d,%99[^,],%99[^,],%d,%ld", &bookId, title, author,
&isIssued, &issueDate) == 5) {
        struct Book* newBook = createBookNode(bookId, title, author);
        newBook->isIssued = isIssued;
        newBook->issueDate = issueDate;
        insertBook(head, newBook);
    }

    fclose(file);
}
```

Main function:-

```
int main() {
    struct User* userList = NULL;
    struct Book* library = NULL;

    char username[50];
    char password[50];
    int bookId;
    char title[100];
    char author[100];
    char choice;

    loadUsersFromFile(&userList, "users.txt");
    loadBooksFromFile(&library, "books.txt");
```

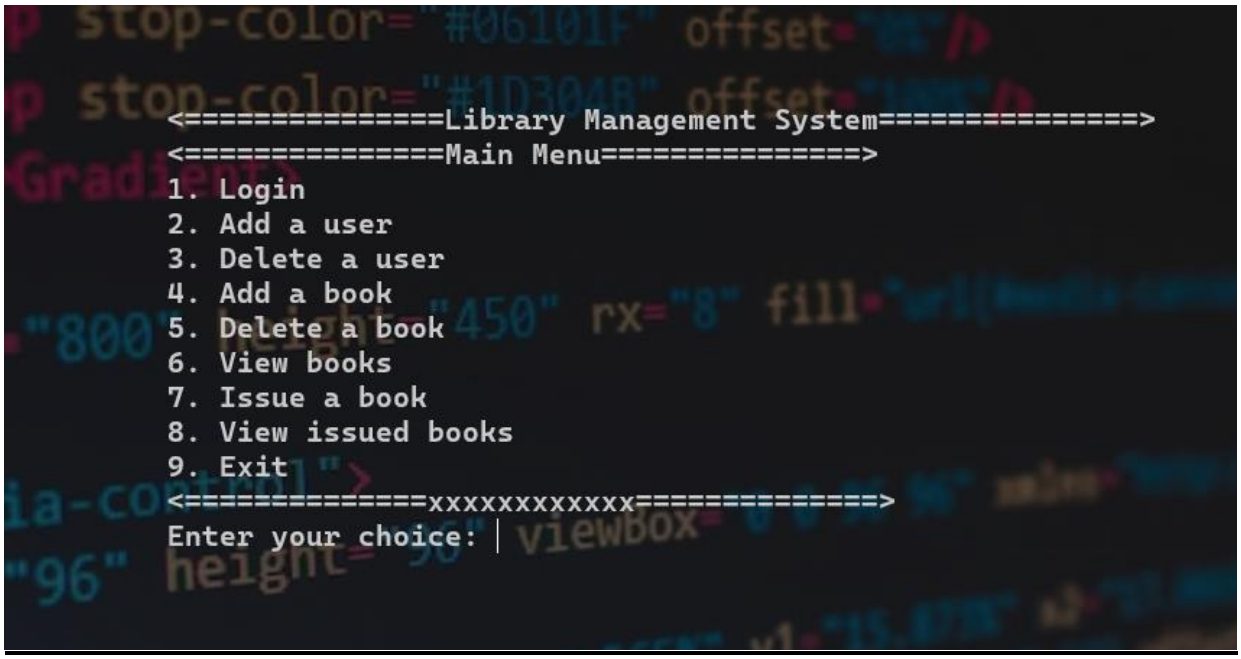
```c
    do {
        printf("\n\n\n\t\t\t<==============Library Management
System==============>\n");
        printf("\t\t\t\t<==============Main Menu==============>\n");
        printf("\t\t\t\t1. Login\n");
        printf("\t\t\t\t2. Add a user\n");
        printf("\t\t\t\t3. Delete a user\n");
        printf("\t\t\t\t4. Add a book\n");
        printf("\t\t\t\t5. Delete a book\n");
        printf("\t\t\t\t6. View books\n");
        printf("\t\t\t\t7. Issue a book\n");
        printf("\t\t\t\t8. View issued books\n");
        printf("\t\t\t\t9. Exit\n");
        printf("\t\t\t\t<=============xxxxxxxxxxxx=============>\n");
        printf("\t\t\t\tEnter your choice: ");
        scanf(" %c", &choice);

        switch (choice) {
            case '1':
                printf("\t\t\t\tEnter your username: ");
                scanf("%s", username);
                printf("\t\t\t\tEnter your password: ");
                scanf("%s", password);

                // Check if the user exists
                struct User* currentUser = userList;
                int isUserFound = 0;
                while (currentUser != NULL) {
                    if (strcmp(currentUser->username, username) == 0 &&
strcmp(currentUser->password, password) == 0) {
                        isUserFound = 1;
                        break;
                    }
                    currentUser = currentUser->next;
                }

                if (isUserFound) {
                    printf("\t\t\t\tLogin successful. Welcome, %s!\n",
username);
                } else {
                    printf("\t\t\t\tLogin failed. Incorrect username or
password.\n");
                }
                break;
            case '2':
                printf("\t\t\t\tEnter the username for the new user: ");
                scanf("%s", username);

                // Check if the username already exists
                struct User* existingUser = userList;
                int isUsernameTaken = 0;
                while (existingUser != NULL) {
                    if (strcmp(existingUser->username, username) == 0) {
```

```
                    isUsernameTaken = 1;
                    break;
                }
                existingUser = existingUser->next;
            }

            if (isUsernameTaken) {
                printf("\t\t\t\tUsername already exists. Please choose a
different one.\n");
            } else {
                printf("\t\t\t\tEnter the password for the new user: ");
                scanf("%s", password);
                struct User* newUser = createUserNode(username, password);
                insertUser(&userList, newUser);
                printf("\t\t\t\tUser '%s' has been added to the system.\n",
username);
            }
            break;
        case '3':
            printf("\t\t\t\tEnter the username to delete: ");
            scanf("%s", username);
            deleteUser(&userList, username);
            break;
        case '4':
            printf("\t\t\t\tEnter the book ID: ");
            scanf("%d", &bookId);
            printf("\t\t\t\tEnter the book title: ");
            scanf("\t\t\t\t%[^\n]", title);
            printf("\t\t\t\tEnter the author: ");
            scanf("\t\t\t\t%[^\n]", author);
            struct Book* newBook = createBookNode(bookId, title, author);
            insertBook(&library, newBook);
            printf("\t\t\t\tBook with ID %d has been added to the
library.\n", bookId);
            break;
        case '5':
            printf("\t\t\t\tEnter the book ID to delete: ");
            scanf("%d", &bookId);
            deleteBook(&library, bookId);
            break;
        case '6':
            viewBooks(library);
            break;
        case '7':
            printf("\t\t\t\tEnter the book ID to issue: ");
            scanf("%d", &bookId);
            issueBook(library, bookId);
            break;
        case '8':
            viewIssuedBooks(library);
            break;
        case '9':
            printf("\t\t\t\tExiting the program.\n");
```

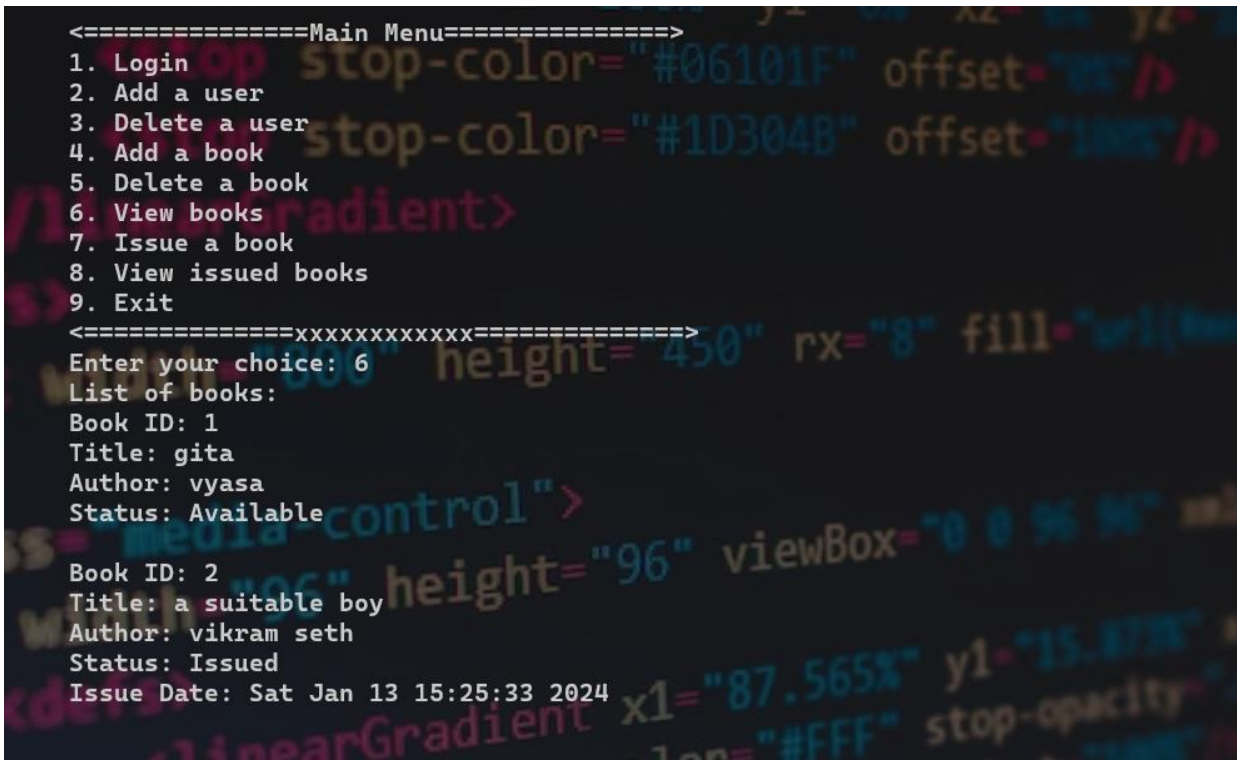
```
                break;
            default:
                printf("\t\t\t\tInvalid choice. Please try again.\n");
        }
    } while (choice != '9');

    saveUsersToFile(userList, "users.txt");
    saveBooksToFile(library, "books.txt");

    // Free allocated memory
    while (userList != NULL) {
        struct User* temp = userList;
        userList = userList->next;
        free(temp);
    }

    while (library != NULL) {
        struct Book* temp = library;
        library = library->next;
        free(temp);
    }

    return 0;
}
```

Output:-

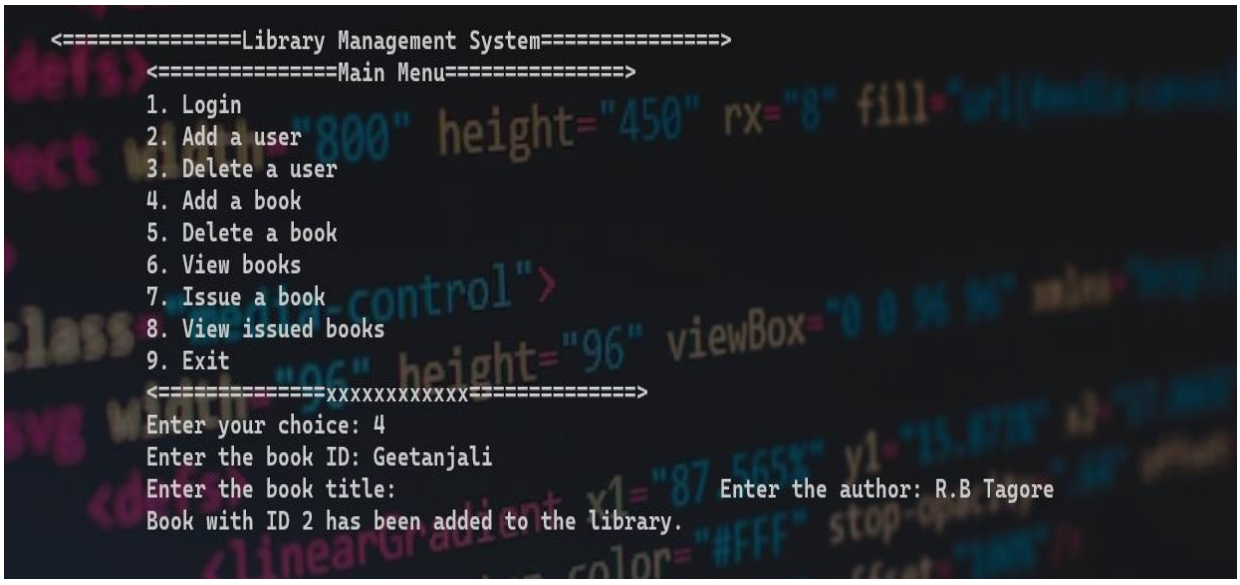
Main Menu



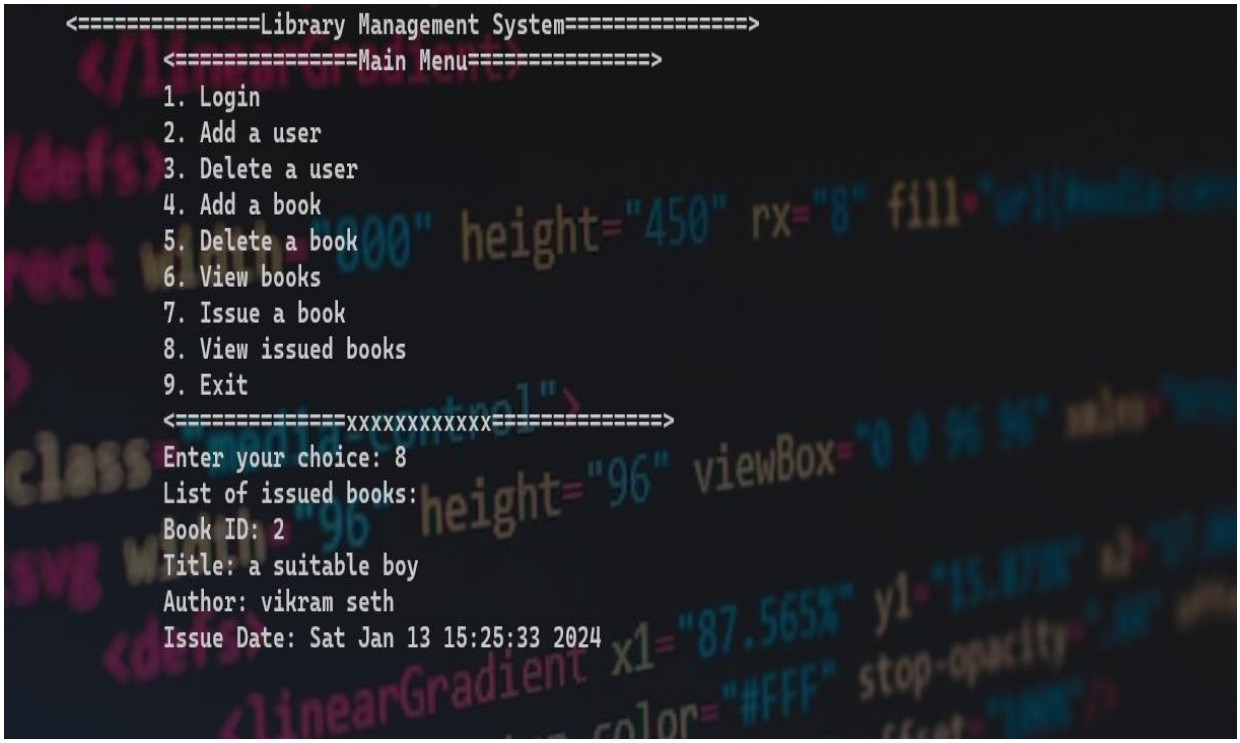
View Books



Add Books



Issued Books



Conclusion

The Library Management System project in C has successfully addressed the critical challenges associated with manual library operations, offering a streamlined and efficient solution. By incorporating features such as secure login authentication, user and book management functionalities, and comprehensive transaction tracking, the system significantly improves the overall management of library resources.

The implementation of a user-friendly interface empowers librarians to perform essential tasks seamlessly, from adding and removing books to efficiently handling user profiles. The project's utilization of the C programming language showcases its capability to create robust and effective software systems.

In essence, the Library Management System project exemplifies the power of technology in optimizing traditional library processes. It not only ensures the security and accuracy of stored information but also offers a user-centric approach to managing library resources. As technology continues to advance, the project serves as a testament to the adaptability and efficiency that well-designed software systems, rooted in C programming, can bring to real-world applications like library management.

References

- https://www.w3schools.com/
- https://alistapart.com/
- https://www.tutorialspoint.com/css/index.htm
- https://www.javatpoint.com/css-tutorial
- https://openai.com/blog/chatgpt
- https://www.google.com/
- https://stackoverflow.com/
- https://www.codewithc.com/mini-project-in-c-library-management-system/