Library Management System

Project Overview

The **Library Management System** is a simple yet effective C++ application designed to handle the operations of a library. The system helps manage library books, user registration and login, borrowing, returning books, and fine calculation for overdue books.

The key features of the system include:

- **User Authentication**: Registration and login system for both regular users and an admin.
- **Book Management**: Admins can add books, while users can search, borrow, and return books.
- **Fine Calculation**: Calculates overdue fines based on a 14-day borrowing period.
- **Persistent Storage**: Books and user data are stored in text files.

System Requirements

Hardware Requirements

- Processor: Minimum 1 GHz or higher
- RAM: 512 MB or higher
- Storage: 50 MB of available disk space

Software Requirements

- C++ Compiler (e.g., GCC, MSVC)
- Operating System: Windows/Linux/macOS
- Text Editor or IDE (e.g., Visual Studio, Code::Blocks, Sublime Text)

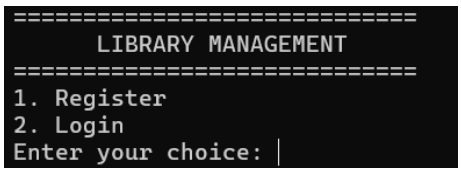
Installation Instructions

1. **Download the Project Files**: Download or clone the repository containing the project source files.
2. **Set Up the Development Environment**: Ensure you have a C++ compiler installed (GCC or MSVC). Install an IDE or text editor like **Code::Blocks**, **Visual Studio**, or **Sublime Text** for easy editing.
3. **Compile the Code**:
    - Open the project folder and Compile the library_management_system.cpp using your C++ compiler.
    - For example, using GCC:

      ```
      g++ library_management_system.cpp -o library_management_system
      ```
4. **Run the Executable**: After compilation, execute the program by running the output file.

   In terminal: `./library_management_system`

   The system will prompt you to register or log in to start using the system.



System Architecture

The **Library Management System** follows a modular architecture where different components handle specific responsibilities. The architecture ensures maintainability, scalability, and simplicity in extending the system.

Key Components

1. **User Login**:
   - Users are required to enter their credentials (username and password).
   - The system checks the provided credentials against the stored user data in `users.txt`.
2. **Validation**:
   - If the credentials match an existing record in `users.txt`, access is granted.
   - If not, an error message is displayed, and access is denied.
3. **Role-Based Access**:
   - Admins have additional privileges such as adding books.
   - Regular users can borrow or return books but cannot modify the library's inventory or access administrative features.
4. **Book Management Module**:
   - Manages book-related operations, such as adding, searching, borrowing, and returning books.
   - Updates book availability and stores borrow/return details.
5. **Data Storage Module**:
   - Uses file-based storage (`books.txt` and `users.txt`) for data persistence.
   - Handles reading and writing data to ensure updates are saved across sessions.
6. **Fine Calculation Module**:
   - Calculates overdue fines based on return dates.
   - Provides a detailed breakdown of the fines for each transaction.

System Design

Data Storage Design

The **Library Management System** utilizes a **file-based database** for storing books and user data. The files are:

- **books.txt**: Stores the book details.
- **users.txt**: Stores user credentials.

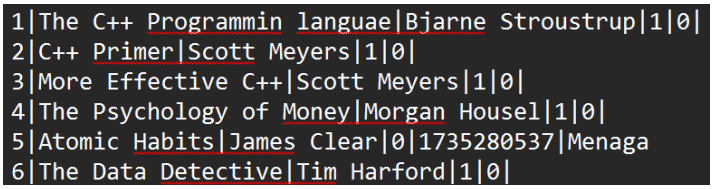
Each book is represented by the following structure:

```
struct Book {
    int id;
    string title;
    string author;
    bool isAvailable;
    time_t borrowDate; // Used for overdue calculation
    string borrowedBy; // To track the borrower
};
```

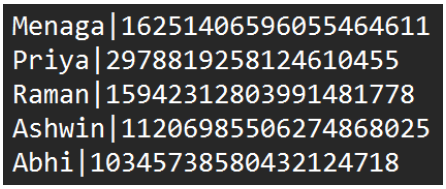
File Structure

The data is stored in plain text files:

- **books.txt**: Each line contains a book’s details separated by the delimiter |.
- **users.txt**: Each line contains a user’s username and hashed password separated by |.



books.txt



users.txt

Main Operations

1. **User Registration**: Allows new users to create an account by providing a username and password.
2. **User Login**: Users can log in using their credentials, while the admin logs in using a hardcoded "admin" username and password.
3. **Add Book**: The admin can add new books to the system.
4. **Search Books**: Users can search for books by title or author.
5. **Borrow Book**: Users can borrow available books. The book's status is updated to unavailable when borrowed.
6. **Return Book**: Users can return books they have borrowed.
7. **Calculate Fine**: If a book is returned after the due date (14 days), the system calculates a fine of ₹10 per day.

Code Explanation

User Authentication

- **registerUser()**: Registers a new user by saving their username and hashed password to users.txt.
- **loginUser()**: Allows users to log in with their credentials. The system compares the entered password (hashed) with stored passwords.

Book Management

- **addBook()**: Allows the admin to add books to the library by entering the book title and author.
- **searchBooks()**: Users can search for books based on the title or author.

Borrow and Return Books

- **borrowBook()**: Allows users to borrow books. When a book is borrowed, its availability status is updated to false, and the borrow date is recorded.
- **returnBook()**: Allows users to return borrowed books. The system ensures that only the person who borrowed the book can return it.

Fine Calculation

- **calculateFine()**: If a borrowed book is returned after the 14-day period, the system calculates the fine at ₹10 per day of delay.

Saving and Loading Data

- **saveBooks()**: Saves the updated book data to the books.txt file.
- **loadBooks()**: Loads book data from the books.txt file when the program starts.

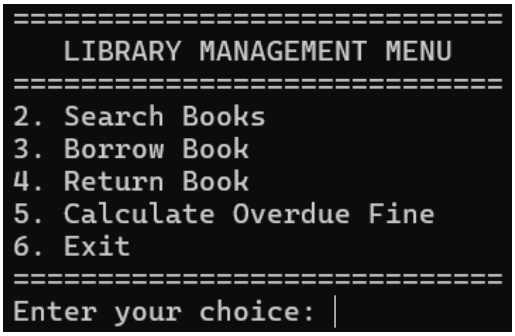
Security Features

- **Password Hashing**: Passwords are not stored in plaintext. Instead, they are hashed using a basic hash function to ensure user security.
- **Data Persistence**: All data (books, user accounts) is saved to text files, ensuring that no data is lost when the application closes.

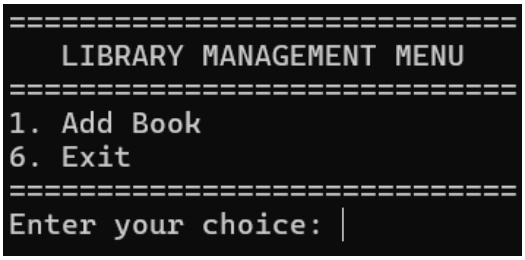
User Interface (UI)

The system uses a **text-based interface**, and users interact with the system through the console. After logging in, users will be presented with a menu that differs based on their role (admin or regular user). The interface is simple, displaying options and requesting inputs accordingly.

Example:



User Menu



Admin Menu

The **Library Management System** is a fully functional C++ application that manages library operations such as book addition, searching, borrowing, and returning, along with overdue fine calculations. This project demonstrates proficiency in C++, file handling, and system design, providing both a practical and user-friendly solution to managing library operations, along with a secure and extendable structure.

Future Enhancements

While the current system is functional, there are several ways to improve and scale it in the future:

- **Database Integration**: Move from file-based storage to a relational database like MySQL or SQLite for better scalability.
- **Graphical User Interface (GUI)**: Develop a GUI version using frameworks like Qt or GTK for a more user-friendly interface.
- **User Notifications**: Implement email or SMS notifications for overdue books and upcoming due dates.
- **Search Optimization**: Implement advanced search functionality with filters (e.g., by genre, publication year).