Library Management System Documentation

This document provides an overview of the Library Management System project, including its features, team roles, and implementation details across the various F# source files you provided.

1. Project Overview

The Library Management System is a desktop application built using F# and Windows Forms, relying on SQL Server for data persistence. It facilitates core library operations such as adding, searching, borrowing, and returning books, along with user management (login/register) for both Admin and standard User roles.

2. Features

The system supports the following key functionalities:

Add, Search, Borrow, Return Books: Core operations for managing the collection.

Track Availability: Books are tracked with a `Status` (Available or Borrowed).

Save/Load Library Data: All data (Users, Books, BorrowedBooks) is persisted in a SQL Server database.

UI for Browsing Books: Users and Admins can view books through a dedicated graphical interface.

User Roles: Separate UIs and permissions for Admin (full CRUD operations on books) and User (borrow/return books, maximum 3 borrowed items).

3. Team Roles and Responsibilities

The following roles were identified for the development of this project:

Role	Primary Responsibilities
Book Model Developer	Defines core data types (`User`, `Book`, `Role`, `BookStatus`) in `Models.fs`.
CRUD Developer	Implements database functions for creating, reading, updating, and deleting book/user data (`Database.fs`).
Search Developer	Implements search logic for books in `AdminForm.fs` and `UserForm.fs`.
Borrow/Return Logic Developer	Implements borrowing rules (max 3 books) and updates status/database records for borrow/return operations (`Database.fs`, `UserForm.fs`).
Storage Developer	Manages database connections and SQL query implementation (`Database.fs`).
UI Developer	Creates the graphical user interface components using Windows Forms (`AdminForm.fs`, `UserForm.fs`, `LoginForm.fs`, `RegisterForm.fs`, `UIHelpers.fs`).
Tester	Ensures all features (especially login, CRUD, borrow limits) work as expected.
Documentation Lead	Maintains project documentation.

4. Code Structure (F# Files)

The project is organized into several F# source files (`.fs`), each handling a specific domain:

File Name	Purpose	Key Functions/Types
`Models.fs`	Defines all core data types used throughout the application.	`Role`, `User`, `BookStatus`, `Book`
`Database.fs`	Handles all database interactions using `System.Data.SqlClient`.	`addUser`, `verifyLoginEmail`, `loadBooksFromDb`, `borrowBook`, `returnBook`, `hashPassword`
`UIHelpers.fs`	Contains reusable functions for creating UI components, specifically book cards.	`createCardWithSelection`, `reloadBooks`
`LoginForm.fs`	Provides the entry point for user login and delegates to the appropriate form	

(Admin/User). | `LoginForm`, `LoginSuccessful` event |

| `RegisterForm.fs` | Allows new users to register an account with the default `User` role. | `RegisterForm`, calls `Database.addUser` |

| `AdminForm.fs`| The main UI for administrators, allowing full CRUD operations on books. | `AdminForm`, book card display with Update/Delete buttons. |

| `UserForm.fs` | The main UI for standard users, allowing searching, borrowing, and returning of books. | `UserForm`, logic to enforce max 3 borrowed books. |

| `Program.fs`| The application's main entry point, initializes and runs the `LoginForm`. | `[<STAThread>]`, `Application.Run(loginForm)` |

5. Key Implementation Details

5.1. Database Connection & Security

Connection: The database connection string is defined in `Database.fs`.

    ```fsharp

    let connectionString =
"Server=localhost\MSSQLSERVER01;Database=LibraryDB;Trusted_Connection=True;"

    ```

Password Hashing: User passwords are secured using **SHA256** hashing before being stored:

    ```fsharp

    let hashPassword (password:string) = use sha =
System.Security.Cryptography.SHA256.Create() // ... hashing logic

    ```

5.2. User Login & Roles

* The `LoginForm` uses `Database.verifyLoginEmail` to validate credentials against the stored hash.

* Upon successful login, the `Program.fs` logic redirects the user to either the `AdminForm` or `UserForm` based on the user's `Role`.

5.3. Book Management (Admin)

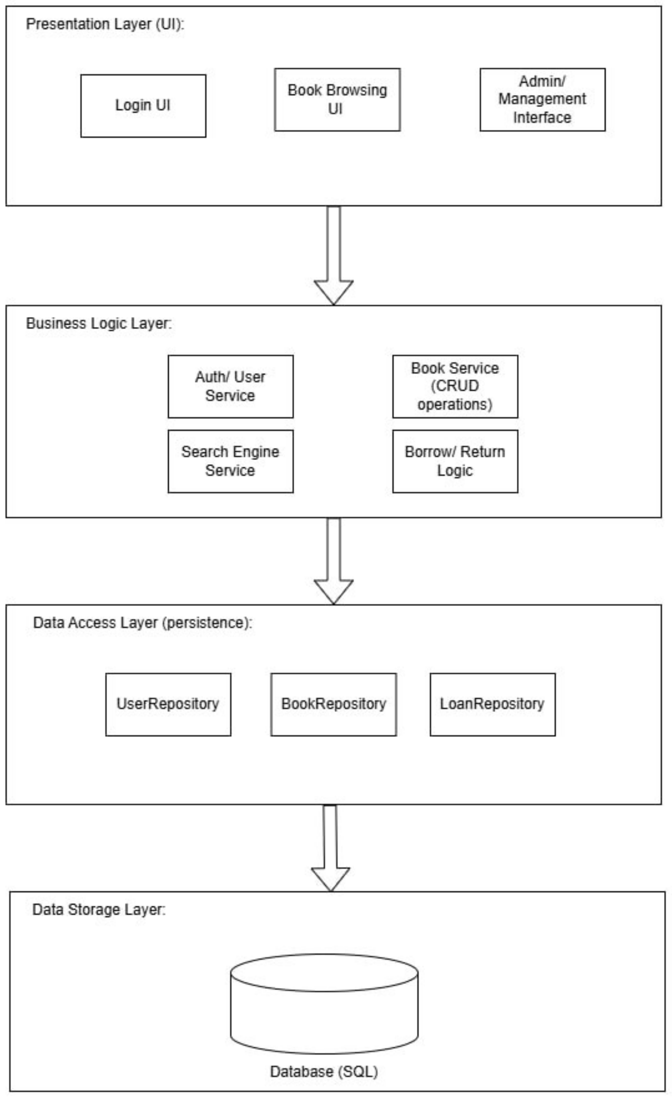
* The `AdminForm` displays book information in custom `Panel` cards within a `FlowLayoutPanel`.

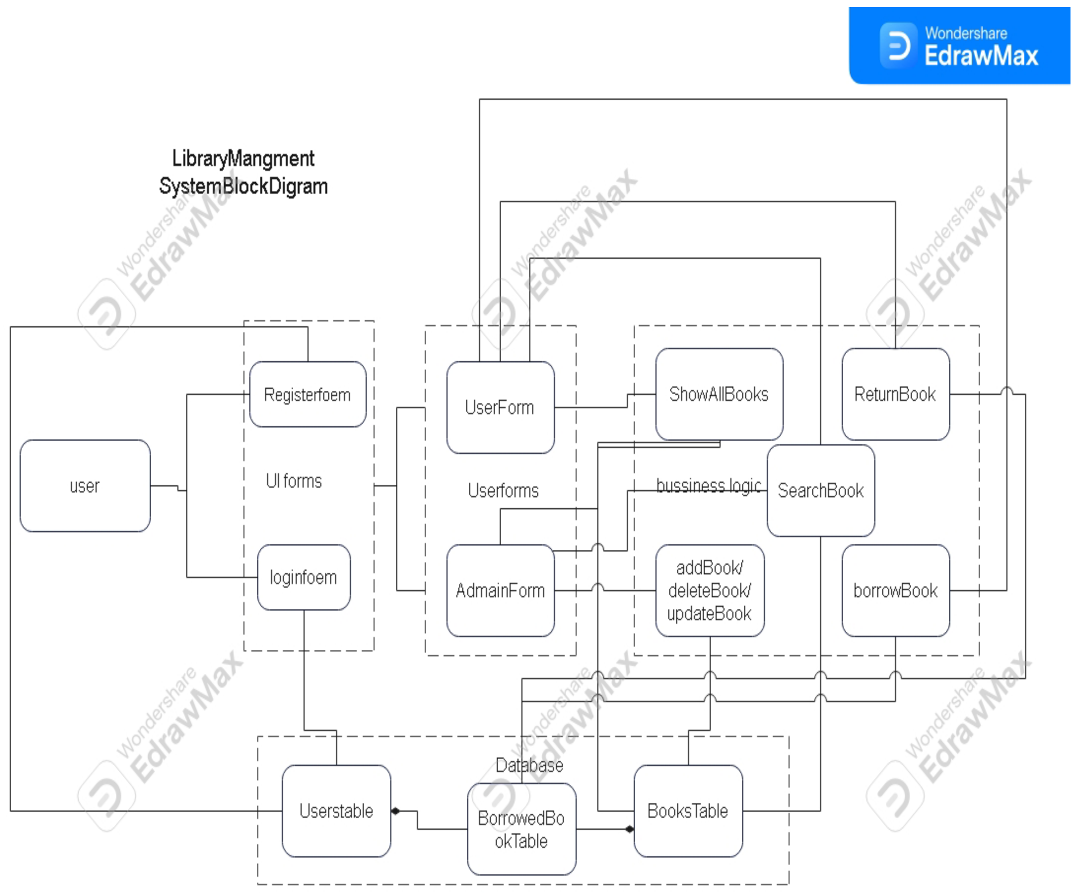
Update/Delete: Admin book cards include dedicated buttons that call `Database.updateBookInDb` or `Database.deleteBookFromDb`.

5.4. Book Operations (User)

* The `UserForm` implements logic to check and enforce the maximum borrowing limit of 3 books using `Database.getBorrowedCountByUser`.

* The Borrow and Return buttons call `Database.borrowBook` and `Database.returnBook`, respectively, to update the book's status and the `BorrowedBooks` tracking table.


Presentation Layer (UI):
Login UI
Book Browsing UI
Admin/ Management Interface
Business Logic Layer:
Auth/ User Service
Book Service (CRUD operations)
Search Engine Service
Borrow/ Return Logic
Data Access Layer (persistence):
UserRepository
BookRepository
LoanRepository
Data Storage Layer:
Database (SQL)


Wondershare EdrawMax
LibraryMangment SystemBlockDigram
user
Registerfoem
UI forms
loginfoem
UserForm
Userforms
AdmainForm
ShowAllBooks
ReturnBook
bussiness logic
SearchBook
addBook/ deleteBook/ updateBook
borrowBook
Database
Userstable
BorrowedBookTable
BooksTable