**Project Proposal: Library Management System for Limkokwing University**

**1. Project Title**

**Library Management System for Limkokwing University**

**2. Objectives**

The primary objective of this project is to design and implement a user-friendly and efficient Library Management System tailored for Limkokwing University. The specific objectives include:

* Developing a robust system with full CRUD (Create, Read, Update, Delete) functionality for managing book records.
* Designing an intuitive graphical user interface using **PyQt**.
* Utilizing **SQLite** to ensure efficient and secure data storage and retrieval.
* Promoting team collaboration by adopting a structured software development methodology.
* Producing high-quality documentation from both technical and user perspectives to facilitate ease of use and maintenance.

**3. Scope**

The Library Management System will deliver the following core functionalities:

* **Create:** Allow authorized users to add new books with details such as title, author, ISBN, genre, and publication year.
* **Read:** Enable viewing and searching of the library’s collection, with advanced filtering by attributes like title, author, ISBN, or genre.
* **Update:** Provide functionality to edit and update existing book records.
* **Delete:** Facilitate the removal of book records from the system when necessary.
* **Search:** Include a robust search feature to quickly locate specific books.
* **Clear Form:** Allow users to reset input fields for streamlined data entry.

**Technology Stack:**

* **Frontend:** PyQt for the development of a responsive graphical user interface.
* **Backend:** SQLite for reliable database management and data integrity.
* **Version Control:** Git/GitHub to ensure efficient collaboration and version management.

**4. Methodology**

The project will be developed using the **Agile Software Development Methodology**, ensuring iterative progress, adaptability, and continuous feedback. The methodology includes:

* **Sprint Planning:** Dividing the project into manageable sprints (e.g., UI design, database integration, testing).
* **Daily Stand-ups:** Regular short meetings to evaluate progress, address challenges, and plan next steps.
* **Iteration Reviews:** Demonstrating deliverables at the end of each sprint to assess alignment with objectives.

This approach will enable flexibility, improve team communication, and ensure timely delivery of high-quality outputs.

**5. Timeline**

|  |  |  |
| --- | --- | --- |
| Phase | Timeline | Key Deliverables |
| Proposal & Requirements Analysis | Day 1–5 | Finalized project proposal and requirement list. |
| System Design | Day 6–9 | UML diagrams and database schema. |
| Frontend Development | Day 10–14 | Interactive and user-friendly GUI using PyQt. |
| Backend Development | Day 15–19 | Database integration and CRUD functionality. |
| Testing & Debugging | Day 20–24 | Comprehensive testing and debugging reports. |
| Documentation | Day 25–29 | Technical and user documentation. |
| Final Presentation | Day 30 | Completed project submission and presentation. |

**6. Deliverables**

1. **Project Proposal:** A detailed document outlining objectives, scope, methodology, and timeline.
2. **System Design:** UML diagrams (e.g., Class, Use Case) and database schema.
3. **Implementation:** Fully functional Library Management System with all required features.
4. **Testing Reports:** Documented test cases, results, and resolutions.
5. **Documentation:**
   * **Technical Documentation:** Comprehensive system architecture and code structure details.
   * **User Manual:** Step-by-step guide for operating the system.

**7. Resources**

**Software Tools:**

* **PyQt:** GUI development.
* **SQLite:** Database management.
* **Git/GitHub:** Version control and team collaboration.
* **PyCharm (or equivalent IDE):** Coding and development.

**Hardware:**

* Personal computers or laptops for development, testing, and deployment.

**8. Team Members and Roles**

|  |  |
| --- | --- |
| **Role** | **Responsibilities** |
| **Frontend Developer** | Design and implement the user interface with PyQt. |
| **Backend Developer** | Develop the database and implement CRUD functionality. |
| **Tester/Debugger** | Conduct rigorous testing and debugging to ensure system reliability. |
| **Documentation Lead** | Produce comprehensive technical and user documentation. |

**9. Evaluation Criteria**

The success of the project will be assessed using the following criteria:

* **Functional Effectiveness:**
  + The application’s ability to meet the defined requirements, including full implementation of CRUD functionality, an intuitive user interface, and seamless database management.
* **Code Quality:**
  + Adherence to object-oriented programming principles, proper coding conventions, and efficient use of resources.
* **Design and Usability:**
  + The user interface should be visually appealing, easy to navigate, and accessible for all users.
  + Responsiveness and efficiency in performing operations like adding, searching, updating, and deleting book records.
* **Documentation Quality:**
  + Completeness and clarity of both technical documentation (e.g., system architecture, database schema, codebase structure) and the user manual.
  + The documentation should support future maintenance and ease of use for non-technical users.
* **Team Collaboration and Contributions:**
  + Evidence of effective teamwork and collaboration throughout the development process.
  + Balanced and meaningful contributions from all team members, as reflected in task completion and participation in meetings.

**System Design: Use Case Diagram for Library Management System**

The **Use Case Diagram** visually represents how users interact with the Library Management System. It highlights the roles (actors) and the actions (use cases) they can perform, emphasizing the functionality of the system.

**Actors:**

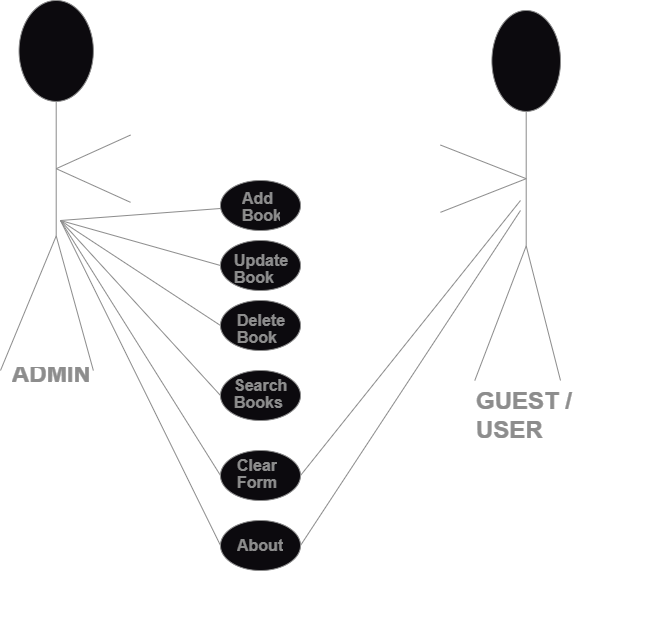
1. **Admin (Primary User):**
   * Responsible for managing the library’s collection, including CRUD operations and search functionality.
2. **Guest/User (Optional):**
   * Has limited access to the system, primarily for viewing and searching books.

**Use Cases:**

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Description** | **Actors** |
| **Add Book** | Admin can add new books to the system by providing required details like title and author. | Admin |
| **View Books** | Admin and Guest can view the list of all books in the library collection. | Admin, Guest/User |
| **Update Book** | Admin can update book details such as title, author, and genre. | Admin |
| **Delete Book** | Admin can delete books from the library collection. | Admin |
| **Search Books** | Admin and Guest can search for specific books using filters like title, author, and genre. | Admin, Guest/User |
| **Clear Form** | Admin can reset form fields during data entry or updates. | Admin |

**Diagram Description:**

* The **Admin** actor has complete access to the system, including CRUD operations and clearing forms.
* The **Guest/User** actor, if included, has restricted access and can only view and search books.

**Use Case Diagram**

**System Design: Class Diagram for Library Management System**

**Classes and Responsibilities**

**1. Book Class**

* **Attributes**:
  + title: String: Represents the title of the book.
  + author: String: Denotes the author of the book.
  + isbn: String: The International Standard Book Number of the book.
  + genre: String: Categorizes the book by genre.
  + publication\_year: Integer: Indicates the year the book was published.
* **Methods**:
  + Constructor: Initializes the attributes with their respective values.

**2. LoginDialog Class**

* **Attributes**:
  + username: String: A hardcoded username used for authentication.
  + password: String: A hardcoded password used for authentication.
* **Methods**:
  + \_\_init\_\_(): Configures the login dialog and initializes the user interface components.
  + authenticate(): Verifies the provided username and password to grant or deny access.

**3. LibraryApp Class**

* **Attributes**:
  + **Core Components**:
    - main\_widget: Represents the main widget of the application.
    - tabs: Manages tabbed sections like "Manage Books" and "About".
  + **Manage Books Tab**:
    - title\_input, author\_input, isbn\_input, genre\_input, year\_input: Input fields for book details.
    - add\_button, update\_button, delete\_button, search\_button, clear\_button: Buttons for CRUD operations.
    - table: Displays the book collection in a tabular format.
  + **Theme Settings**:
    - light\_mode\_style, dark\_mode\_style: Defines CSS for light and dark mode themes.
* **Methods**:
  + **CRUD Operations**:
    - add\_book(): Adds a new book to the collection.
    - update\_book(): Modifies details of the selected book.
    - delete\_book(): Removes a book from the collection.
    - search\_books(): Performs search queries on the book data.
    - clear\_search(): Clears search filters and resets the book list.
  + **Utility Methods**:
    - populate\_table(books: List): Displays books in the tabular interface.
    - toggle\_theme(): Switches between light and dark themes.
    - export\_to\_csv(): Exports book data to a CSV file.
    - show\_message(title: str, message: str): Displays informational or error messages.
    - clear\_inputs(): Clears input fields after operations.

**4. DatabaseManager Class**

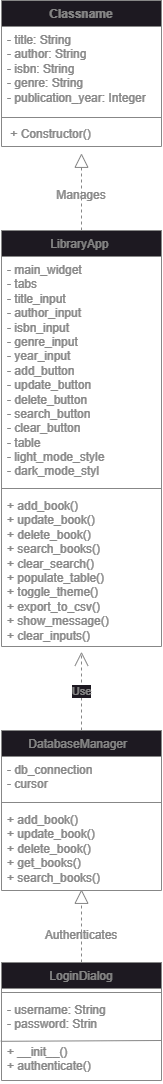
* **Attributes**:
  + db\_connection: Manages the connection to the database.
  + cursor: Executes SQL queries on the connected database.
* **Methods**:
  + **Data Manipulation**:
    - add\_book(title: str, author: str, isbn: str, genre: str, year: int): Adds a book record to the database.
    - update\_book(book\_id: int, title: str, author: str, isbn: str, genre: str, year: int): Updates an existing book record.
    - delete\_book(book\_id: int): Deletes a book record.
    - get\_books(): Retrieves all books stored in the database.
    - search\_books(query: str): Searches for books that match specific criteria.

**Relationships**

* **LibraryApp → DatabaseManager**:
  + Represents a dependency where LibraryApp interacts with DatabaseManager to handle persistent storage and retrieval of book data.
* **LibraryApp → Book**:
  + Indicates that LibraryApp manages Book objects to execute CRUD operations.
* **LoginDialog → LibraryApp**:
  + Ensures only authenticated users can access the main application interface.

**Attributes and Methods Summary**

* **Attributes**:
  + Define the state and structure of the system, such as book details, user credentials, input fields, and database connections.
* **Methods**:
  + Specify system behavior, including user interface updates, data operations, and interactions between components.

**Class Diagram**

### Explanation of Connections

1. **LibraryApp → Book**:
   * Relationship: Manages
   * LibraryApp uses Book instances to perform operations such as adding, updating, and searching for books.
2. **LibraryApp → DatabaseManager**:
   * Relationship: Uses
   * LibraryApp relies on DatabaseManager to interact with the database for storing and retrieving book data.
3. **LoginDialog → LibraryApp**:
   * Relationship: Authenticates
   * LoginDialog ensures only authenticated users can access LibraryApp. This establishes a security layer.

### Implementation of the Library Management System

### ****1. Implementation Overview****

The **Library Management System** is a desktop application developed using **PyQt5** for the user interface and **SQLite** as the database backend. It provides functionalities to manage books, including adding, updating, searching, deleting, and exporting book data to a CSV file. The application features a modern, user-friendly UI with options for light and dark themes.

### ****2. Setup Development Environment****

#### ****Prerequisites****

To develop and run the Library Management System, ensure the following tools and libraries are installed:

1. **Python** (Version 3.8 or later)  
   Download and install Python from the [official Python website](https://www.python.org/).
2. **pip** (Python Package Installer)  
   Pip is typically bundled with Python. Confirm installation by running:

bash

Copy code

pip --version

1. **Virtual Environment**  
   Set up a virtual environment to manage dependencies:

bash

Copy code

python -m venv library\_env

1. **PyQt5**  
   Install PyQt5 and its tools:

bash

Copy code

pip install pyqt5 pyqt5-tools

1. **SQLite**  
   SQLite comes bundled with Python. Verify by running:

bash

Copy code

python -c "import sqlite3; print(sqlite3.sqlite\_version)"

1. **Additional Libraries**  
   Install other necessary libraries:

bash

Copy code

pip install pandas

#### ****Development Tools****

* **Code Editor:** [Visual Studio Code (VS Code)](https://code.visualstudio.com/) or [PyCharm](https://www.jetbrains.com/pycharm/).
* **SQLite Database Browser:** Use a tool like [DB Browser for SQLite](https://sqlitebrowser.org/) for manual database management during development.

#### ****Cloning the Repository****

If you're collaborating or using version control, clone the project repository:

bash

Copy code

git clone <repository-url>

cd library\_management\_system

#### ****Running the Application****

Activate the virtual environment and run the application:

bash

Copy code

# Activate virtual environment (Windows)

library\_env\Scripts\activate

# Activate virtual environment (Mac/Linux)

source library\_env/bin/activate

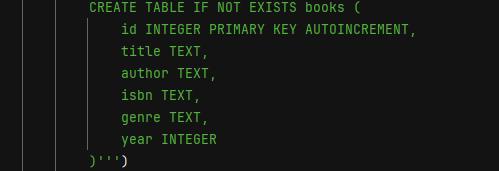
# Run the application

python main.py

### ****3. Database Setup (SQLite)****

#### ****Database Structure****

SQLite is used for the database backend, with a single table named books.

Schema:

#### ****DatabaseManager Class****

The DatabaseManager class encapsulates all database operations, providing methods for CRUD functionality.

**Initialization**:

import sqlite3  
  
  
class DatabaseManager:  
 def \_\_init\_\_(self):  
 self.connection = sqlite3.connect('library.db')  
 self.cursor = self.connection.cursor()  
 self.create\_books\_table()  
  
 def create\_books\_table(self):  
 self.cursor.execute('''CREATE TABLE IF NOT EXISTS books (  
 id INTEGER PRIMARY KEY AUTOINCREMENT,  
 title TEXT,  
 author TEXT,  
 isbn TEXT,  
 genre TEXT,  
 year INTEGER  
 )''')  
 self.connection.commit()  
  
 def add\_book(self, title, author, isbn, genre, year):  
 self.cursor.execute('''INSERT INTO books (title, author, isbn, genre, year)  
 VALUES (?, ?, ?, ?, ?)''', (title, author, isbn, genre, year))  
 self.connection.commit()  
  
 def update\_book(self, book\_id, title, author, isbn, genre, year):  
 self.cursor.execute('''UPDATE books SET title=?, author=?, isbn=?, genre=?, year=?   
 WHERE id=?''', (title, author, isbn, genre, year, book\_id))  
 self.connection.commit()  
  
 def delete\_book(self, book\_id):  
 self.cursor.execute('DELETE FROM books WHERE id=?', (book\_id,))  
 self.connection.commit()  
  
 def search\_books(self, term):  
 *"""  
 Searches for books based on a search term.  
 Supports partial matches across multiple fields.  
 """* try:  
 query = """  
 SELECT \* FROM books  
 WHERE LOWER(title) LIKE ?  
 OR LOWER(author) LIKE ?  
 OR isbn LIKE ?  
 OR LOWER(genre) LIKE ?  
 OR year LIKE ?  
 """  
 # Add wildcard for partial matching  
 params = tuple(f"%{term.lower()}%" for \_ in range(5))  
 self.cursor.execute(query, params)  
 return self.cursor.fetchall()  
 except Exception as e:  
 logging.error(f"Error in search\_books query: {e}")  
 raise  
  
 def get\_books(self):  
 self.cursor.execute('SELECT \* FROM books')  
 return self.cursor.fetchall()  
  
 def close\_connection(self):  
 self.connection.close()

### ****Database Setup (SQLite)****

#### ****Database Structure****

The database is implemented using SQLite, providing a lightweight and easy-to-integrate solution for managing data. The structure includes a single table, books, with the following schema:

|  |  |  |
| --- | --- | --- |
| **Column** | **Type** | **Description** |
| id | INTEGER | Primary key, auto-incremented |
| title | TEXT | Title of the book |
| author | TEXT | Author of the book |
| isbn | TEXT | ISBN number (unique) |
| genre | TEXT | Genre of the book |
| year | INTEGER | Year of publication |

#### ****DatabaseManager Class****

The DatabaseManager class handles all interactions with the SQLite database, including CRUD operations.

**Key Methods:**

* **add\_book(title, author, isbn, genre, year)**  
  Adds a new book to the database.
* **update\_book(book\_id, title, author, isbn, genre, year)**  
  Updates an existing book's details using its unique book\_id.
* **delete\_book(book\_id)**  
  Deletes a book from the database.
* **search\_books(search\_term)**  
  Searches for books by title, author, or ISBN and returns matching records.
* **get\_books()**  
  Retrieves all books from the database.

### ****3. MAIN & LIBRARY UI Implementation (PyQt5)****

#### ****Login Dialog****

The LoginDialog is a simple dialog for authenticating users. It prompts for a username and password and allows access only when correct credentials are provided.

* Default username: Joshua
* Default password: 16474

#### ****Main Window****

The LibraryApp class defines the primary user interface, built using a QVBoxLayout to organize widgets. It includes:

* A **date and time display** updated dynamically.
* A **QTabWidget** with two tabs:
  + **Manage Books Tab**: For CRUD operations.
  + **About Tab**: Displays application details.
* A **theme toggle button** for switching between light and dark modes.

### ****4. CRUD Operations in the GUI****

#### ****Create (Add Book)****

* Users can enter book details in the provided form fields.
* Clicking the **Add Book** button validates inputs and calls add\_book to store the new record in the database.

#### ****Read (View Books)****

* All books are displayed in a QTableWidget, fetched via get\_books().
* Users can also search books using the **Search** functionality, which filters and displays matching results.

#### ****Update****

* Users select a book in the table, populate the form with its data, make changes, and click **Update Book**.
* The update\_book method applies changes to the database.

#### ****Delete****

* Users select a book in the table and click **Delete Book** to remove it.
* Confirmation dialogs ensure accidental deletions are avoided.

#### ****Export to CSV****

* Clicking **Export to CSV** saves all book records to a .csv file using the Python csv module.

### ****5. User Interface Design for Library Management System****

#### ****Light Theme****

* White background with contrasting dark text.
* Buttons styled with a light blue background.
* Form fields have rounded borders for a modern aesthetic.

#### ****Dark Theme****

* Dark gray background with light text for reduced eye strain.
* Table headers styled with darker tones for better visibility.

#### ****Additional UI Features****

* Search bar for real-time filtering of book records.
* Dynamic resizing of the table columns for improved readability.
* Form validation to ensure inputs meet requirements (e.g., ISBN must be numeric, year must be 4 digits).

### ****6. Testing****

#### ****Test Scenarios****

* **Authentication**: Test login with correct and incorrect credentials.
* **CRUD Operations**: Verify the correctness of adding, updating, deleting, and searching books.
* **UI Responsiveness**: Ensure smooth switching between light and dark themes.
* **CSV Export**: Validate the content and format of exported files.

#### ****Error Handling****

* Invalid inputs are captured with detailed error messages using QMessageBox.
* Database errors are logged in a app.log file for debugging.

### ****7. Conclusion****

This Library Management System offers a robust and intuitive platform for managing book inventories. With its SQLite backend, PyQt5 GUI, and well-designed functionalities, it meets the essential needs of users while providing room for scalability.

**USER MANUAL FOR THE LIBRARY MANAGEMENT SYSTEM**

1. **Login Screen**

**Purpose:** Restrict unauthorized access.

**Steps:**

1. Open the application.

2. Enter the Username (Joshua) and Password (16474).

3. Click the Login button.

4. If credentials are valid, the main application will load. If invalid, a warning message will appear.

**2. Main Dashboard**

**Tabs:**

**1. Manage Books:** Handle all book-related CRUD operations.

**2. About:** Displays application version and developer details.

**Additional Features:**

**Date & Time:** Displays current date and time at the top right.

**Theme Toggle:** Switch between Light Mode and Dark Mode using the button at the bottom.

**3. Manage Books Tab**

**Adding a Book:**

1. Fill in the form fields: Title, Author, ISBN, Genre, and Year.

2. Click the Add Book button.

3. A success message confirms the book has been added to the database.

**Updating a Book:**

1. Select a row from the table.

2. The form fields auto-populate with the selected book's details.

3. Modify the fields as required and click Update Book.

**Deleting a Book:**

1. Select a row from the table.

2. Click the Delete Book button.

3. The selected book is removed, and the table updates automatically.

**Searching for Books:**

1. Enter a keyword (e.g., title, author, or genre) in the search bar.

2. Click the Search button to filter books.

3. To reset the table, click Clear.

**Exporting to CSV:**

1. Click Export to CSV.

2. Choose a save location and filename in the dialog box.

3. The current book list is saved as a .csv file.

**4. About Tab**

View basic details about the application, including version and the developer's name.

**5. Theme Toggle**

Toggle between Light Mode and Dark Mode:

Light Mode: Default theme with a white background.

Dark Mode: Modern, dark-colored UI.

**References**

1. PyQt5 Documentation:

PyQt5 Official Guide

2. Python Official Documentation:

Python Documentation

3. CSV Library:

CSV Module - Python Documentation

4. Logging in Python:

Logging Module - Python Documentation

5. SQLite Integration:

Resources or personal notes for setting up a DatabaseManager.

**WIREFRAME MOCKUP DESIGNED IN FIGMA**