**ABSTRACT**

The Library Management System developed using Python's Tkinter library is a simple graphical user interface (GUI) application designed to manage basic library functions efficiently. The system begins with a login and registration interface that allows librarians to create new accounts or log in using their credentials. Once authenticated, the user is granted access to the main management screen.

From this interface, the librarian can perform essential tasks such as adding new books to the library's collection, removing books that are no longer available or needed, and issuing books to users. When a book is issued, it is moved from the available books list to the issued (lend) list. Additionally, there is a feature to view all currently available books in the library, which are displayed in a message box.

All data, including user credentials and book records, are stored in memory using Python lists, making it a lightweight solution for small-scale library environments. The application uses message boxes to provide immediate feedback to the user, ensuring a smooth and interactive experience. Overall, this project demonstrates how Tkinter can be effectively used to create functional and user-friendly desktop applications for managing everyday tasks.

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| **S.NO** | **TOPICS** | **PAGE.NO** |
|  | ABSTRACT |  |
| 1. | INTRODUCTION | 1 |
| 2. | PROJECT DESCRIPTION | 2 |
| 3. | VARIABLE NAMES | 3 |
| 4. | FUNCTIONS / METHODS | 4 |
| 5. | USAGE OF TKINTER | 5 |
| 6. | DATA FLOW DIAGRAM | 7 |
| 7. | CODING | 8 |
| 8. | SCREEN SHOTS | 12 |
| 9. | REFERENCES | 14 |
| 10. | CONCLUSION | 15 |

**1.INTRODUCTION**

Library management plays a crucial role in organizing, maintaining, and tracking books and other resources in educational institutions and public libraries. Traditionally, this process involves manual record-keeping, which can be time-consuming and prone to errors. With the advancement of technology, software-based solutions have become an efficient way to streamline library operations.This project introduces a **Library Management System** built using **Python's Tkinter library**, which provides a graphical user interface (GUI) for interacting with the system. The application is designed to help librarians perform core functions such as registering users, logging in, adding or removing books, issuing books, and viewing the current list of available books. By providing a simple and intuitive interface, this system minimizes the need for manual record-keeping and enhances operational efficiency, making it ideal for small libraries or educational environments seeking a lightweight management solution.

**2.PROJECT DESCRIPTION**

The **Library Management System** is a desktop application developed using **Python** and its built-in **Tkinter** module for creating a graphical user interface (GUI). The main objective of this project is to simplify and automate basic library operations such as managing books and librarian access.

The system starts with a **login and registration interface**. Librarians can register by creating a username and password, which are stored in memory using a list. Upon successful login, the user is directed to the main management interface.

The core features of the application include:

1. **Add Book**: Allows the librarian to enter the name of a book and add it to the library’s collection.
2. **Remove Book**: Enables removal of a specific book from the library if it exists in the collection.
3. **Issue Book**: Issues a book to a user by moving it from the available books list to a separate issued list (lend list).
4. **View Books**: Displays all currently available books in the library that have not been issued.

The system uses simple list structures (books, lend\_list, librarians) to manage the book inventory and librarian credentials. All interactions are performed through user-friendly widgets like labels, entry fields, and buttons. Feedback for each operation (success or error) is provided through message boxes.

This project is best suited for small-scale libraries or as a learning tool for understanding GUI development, event handling, and basic data management in Python. It can be further enhanced by integrating file handling or databases for permanent data storage and retrieval.

**3.VARIABLE NAMES**

### **Books**

* BookId: Unique identifier for each book in the system.
* Title*:* The name/title of the book.
* Author*:* The author(s) of the book.
* Genre*:* The category or type of the book (e.g., fiction, science).
* Isbn*:* International Standard Book Number, unique for each edition.
* CopiesAvailable*:* Number of copies currently available for borrowing.

### **Users**

* userId:Unique identifier for each registered user/member.
* username:Name of the user.
* Email:User’s email address for communication.
* phoneNumber:User’s contact number.
* membershipType:Type of membership (e.g., student, faculty, guest).

### **Borrowing**

* borrowId:Unique identifier for each borrow transaction.
* borrowDate*:*Date when the book was borrowed.
* dueDate:Date by which the book should be returned.
* returnDate:Actual date the book was returned.
* isReturned*:*Boolean flag indicating if the book has been returned.
* fineAmount:Fine charged for late returns, if any.

### **Authentication**

* username:Login name used by the user.
* Password:User’s password (stored securely, usually hashed).
* authToken:Token used to authenticate user sessions.

### **Admin**

* adminId:Unique identifier for the administrator.
* Settings:System or library configuration settings.

**4.FUNCTIONS / METHODS**

### **Book Management**

* addBook(book):Adds a new book to the system.
* updateBook(bookId, updatedData):Updates book details like title, author, or stock.
* deleteBook(bookId):Removes a book form the database.
* searchBooks(query):Finds books by title, author, genre, or ISBN.
* getBookDetails(bookId):Returns full details of a specific book.

### **User Management**

* registerUser(userData):Registers a new member in the system.
* updateUser(userId, updatedData):Modifies user information.
* deleteUser(userId):Deletes a user account.
* getUser(userId):Fetches details of a specific user.

### **Borrowing & Returning**

* borrowBook(userId, bookId):Issues a book to a user.
* returnBook(userId, bookId):Processes the return of a borrowed book.
* calculateFine(borrowId):Calculates late return fines, if applicable.
* getBorrowHistory(userId):Shows all books borrowed by a user.

### **Authentication**

* login(username, password):Verifies credentials and logs the user in.
* logout(userId):Ends the user's session.
* resetPassword(userId):Allows the user to change or recover their password.

### **Admin/System**

* getSystemLogs():Fetches activity logs for monitoring.
* updateSettings(settings):Applies changes to system configuration.

**5.USAGE OF TKINTER**

**1. Main Application Window**

* Tk() root window is created and configured with a title, size, and background color. This acts as the base container for all UI components.

**2. Labels**

* tk.Label is used to display static text such as:
* The title ("Library Management System")
* Field labels ("Username", "Password", "Add Book", etc.)
* These guide the user in interacting with the system.

**3. Entry Fields**

* tk.Entry allows users to input text. It is used for:
* Entering usernames and passwords
* Typing book names for add, remove, or issue actions
* The password field uses show="\*" to mask input.

**4. Buttons**

* tk.Button triggers specific actions when clicked:
* Login/Register buttons process authentication
* Add/Remove/Issue/View Book buttons perform corresponding library functions
* Each button is linked to a method via the command parameter.

**5. Message Boxes**

* messagebox.showinfo() and messagebox.showerror() display pop-up messages:
* Show confirmation when actions succeed (e.g., book added)
* Display error messages when operations fail (e.g., invalid login)

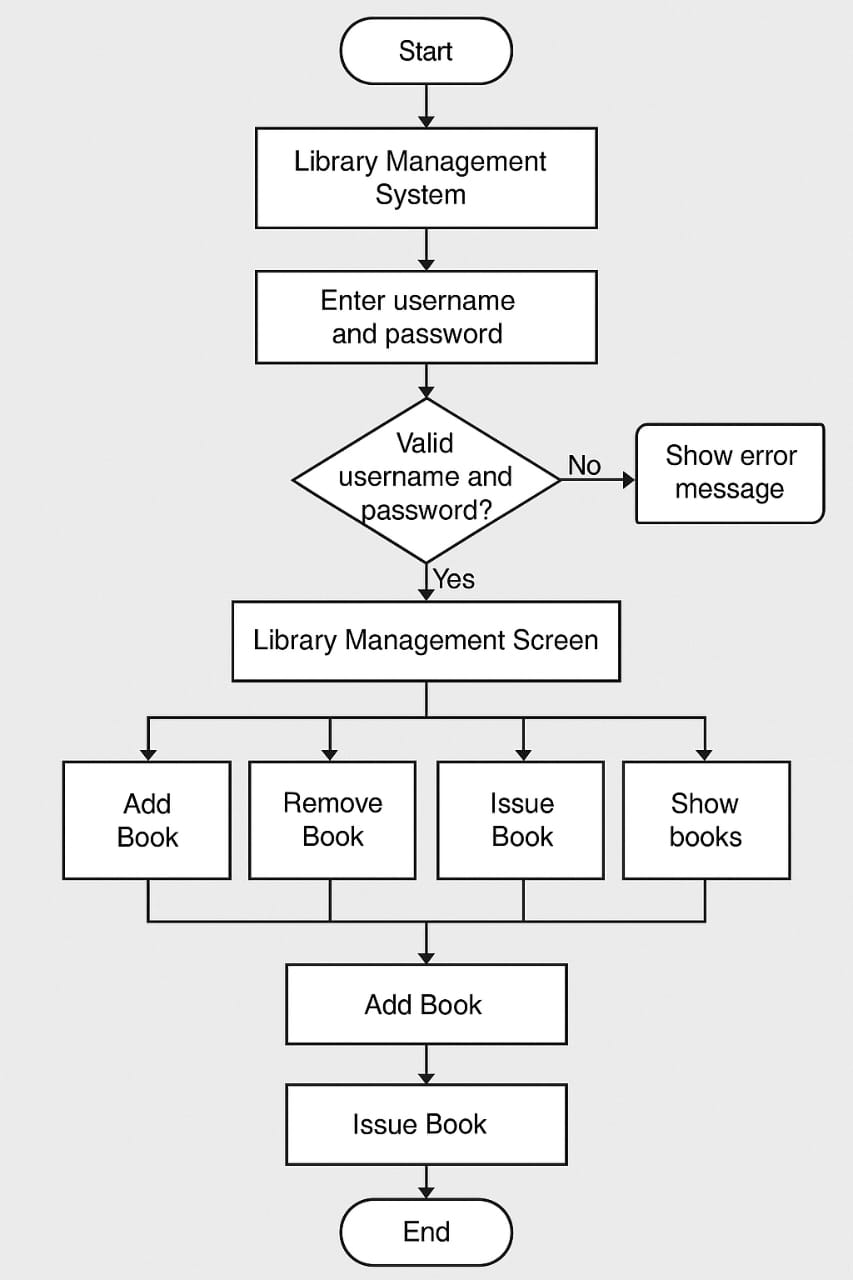
**6. Dynamic UI Updates**

* After a successful login:
* The login interface elements are removed using .destroy()
* The book management interface is created dynamically
* This simulates navigating to a new screen without opening a new window**.**

**7. Book Management Interface**

* Tkinter is used again to create a new layout for managing books, including:
* Labels and entry fields for book names
* Buttons for each operation (add, remove, issue, view)

**6.DATA FLOW DIAGRAM**



**7.CODING**

import tkinter as tk from tkinter import messagebox class LibraryManagement: def \_\_init\_\_(self, master):

self.master = master self.master.title("Library Management System") self.master.geometry("600x600") self.master.config(bg='#708090') self.books = [] self.lend\_list = [] self.login\_label = tk.Label(self.master, text="Library Management System", font=("Helvetica",

16), bg='#708090', fg='white') self.login\_label.pack() self.username\_label = tk.Label(self.master, text="Username", font=("Helvetica", 12), bg='#708090', fg='white') self.username\_label.pack() self.username\_entry = tk.Entry(self.master, font=("Helvetica", 12)) self.username\_entry.pack() self.password\_label = tk.Label(self.master, text="Password", font=("Helvetica", 12), bg='#708090', fg='white') self.password\_label.pack()

self.password\_entry = tk.Entry(self.master, font=("Helvetica", 12), show="\*") self.password\_entry.pack() self.login\_button = tk.Button(self.master, text="Login", command=self.login, font=("Helvetica",

12)) self.login\_button.pack() self.register\_button = tk.Button(self.master, text="Register", command=self.register, font=("Helvetica", 12)) self.register\_button.pack()

self.username = "" self.password = "" self.librarians = [] def login(self):

self.username = self.username\_entry.get() self.password = self.password\_entry.get() for librarian in self.librarians: if self.username == librarian[0] and self.password == librarian[1]:

self.username\_entry.delete(0, tk.END) self.password\_entry.delete(0, tk.END) self.login\_label.destroy() self.username\_label.destroy() self.username\_entry.destroy() self.password\_label.destroy() self.password\_entry.destroy() self.login\_button.destroy() self.register\_button.destroy() self.library\_management\_screen()

return

messagebox.showerror("Error", "Invalid username or password") def register(self):

self.username = self.username\_entry.get() self.password = self.password\_entry.get() self.librarians.append([self.username, self.password]) self.username\_entry.delete(0, tk.END) self.password\_entry.delete(0, tk.END) def library\_management\_screen(self):

self.add\_book\_label = tk.Label(self.master, text="Add Book", font=("Helvetica", 16), bg='#708090', fg='white') self.add\_book\_label.pack() self.add\_book\_entry = tk.Entry(self.master, font=("Helvetica", 12)) self.add\_book\_entry.pack() self.add\_book\_button = tk.Button(self.master, text="Add Book", command=self.add\_book, font=("Helvetica", 12)) self.add\_book\_button.pack() self.remove\_book\_label = tk.Label(self.master, text="Remove Book", font=("Helvetica", 16), bg='#708090', fg='white') self.remove\_book\_label.pack() self.remove\_book\_entry = tk.Entry(self.master, font=("Helvetica", 12)) self.remove\_book\_entry.pack() self.remove\_book\_button = tk.Button(self.master, text="Remove Book", command=self.remove\_book, font=("Helvetica", 12)) self.remove\_book\_button.pack() self.issue\_book\_label = tk.Label(self.master, text="Issue Book", font=("Helvetica", 16), bg='#708090', fg='white') self.issue\_book\_label.pack() self.issue\_book\_entry = tk.Entry(self.master, font=("Helvetica", 12)) self.issue\_book\_entry.pack() self.issue\_book\_button = tk.Button(self.master, text="Issue Book", command=self.issue\_book, font=("Helvetica", 12)) self.issue\_book\_button.pack() self.view\_books\_button = tk.Button(self.master, text="View Books", command=self.view\_books, font=("Helvetica", 12)) self.view\_books\_button.pack()

def add\_book(self): book = self.add\_book\_entry.get() self.books.append(book) messagebox.showinfo("Success", "Book added successfully") self.add\_book\_entry.delete(0, tk.END)

def remove\_book(self):

book = self.remove\_book\_entry.get()

if book in self.books: self.books.remove(book) messagebox.showinfo("Success", "Book removed successfully") else:

messagebox.showerror("Error", "Book not found") self.remove\_book\_entry.delete(0, tk.END)

def issue\_book(self):

book = self.issue\_book\_entry.get() if book in self.books: self.lend\_list.append(book) self.books.remove(book) messagebox.showinfo("Success", "Book issued successfully") else:

messagebox.showerror("Error", "Book not found") self.issue\_book\_entry.delete(0, tk.END)

def view\_books(self):

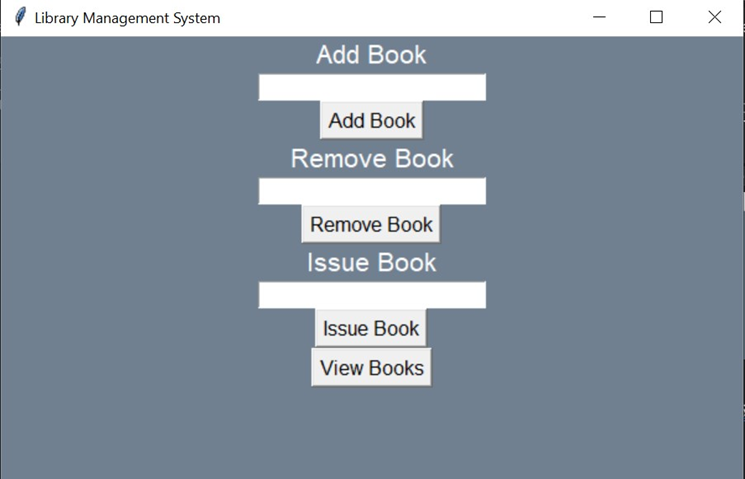
message = "\n".join(self.books) messagebox.showinfo("Books", message)

if \_\_name\_\_ == "\_\_main\_\_": root = tk.Tk() app = LibraryManagement(root) root.mainloop()

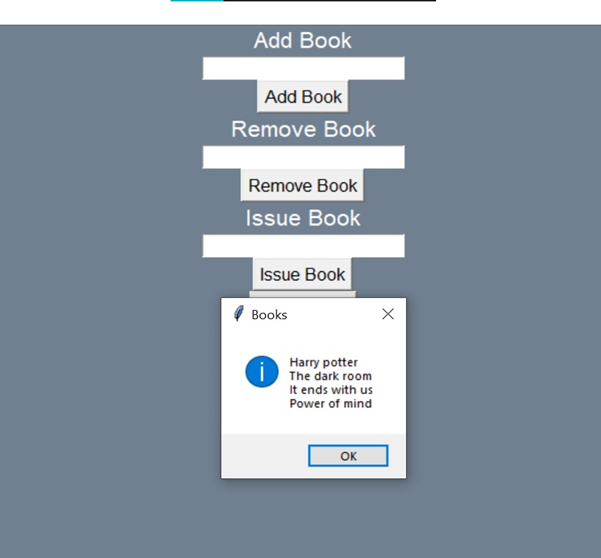
**8.SCREENSHOTS**



### **FIG 1. LOGIN PAGE**



### **FIG 2. LIBRARY MANAGEMENT PAGE**



**FIG 3. AVAILABLE BOOKPAGE**

### **9. CONCLUSION**

In conclusion, the Simple Online Library Management System project written in Python offers a straightforward and user-friendly GUI-based system for managing Library-related tasks. With a focus on simplicity and ease of use, the system provides essential functions such as entering user details like username and password. Notably, the absence of a login system allows for seamless access to all available features without constraints, making it a convenient choice for users. Designed with beginners in mind, the project serves as an introductory console-based application, ensuring accessibility for those new to programming. The availability of the project's source code for free download further encourages learning and customization within the realm of Python programming

# **10.** **REFERENCES**

### **1**. **GeeksforGeeks – DFD for Library Management System**

This article offers a comprehensive explanation of DFDs for a Library Management System, including Level 0, Level 1, and Level 2 diagrams. It details the flow of information and transformations applied when data moves in and out of the system. ([geeksforgeeks.org](https://www.geeksforgeeks.org/dfd-for-library-management-system/?utm_source=chatgpt.com" \o "DFD for Library Management System - GeeksforGeeks))

**2**. **Visual Paradigm – DFD Example: Library Management System**

Visual Paradigm provides an editable DFD template for a Library Management System. You can customize this diagram to suit your specific requirements and gain insights into the system's data flow. ([online.visual-paradigm.com](https://online.visual-paradigm.com/diagrams/templates/data-flow-diagram/dfd-example-library-management-system/?utm_source=chatgpt.com" \o "DFD Example: Library Management System | Data Flow Diagram Template))

### **3**. **EdrawMax – Library Management System Data Flow Diagram Template**

EdrawMax offers a comprehensive DFD template for a Library Management System, illustrating the interaction between library staff, students, and book data. This template is useful for understanding the workflow of a library's operations, including student registration, book information management, and staff roles. ([edrawmax.com](https://www.edrawmax.com/templates/1012733/?utm_source=chatgpt.com" \o "Library Management System Data Flow Diagram Template | EdrawMax Templates))

### **4**. **123ProjectLab – DFD for Library Management System**

This resource provides detailed Level 2 DFDs for a Library Management System, breaking down processes such as book management, author and publisher information, and membership reports. It's particularly helpful for understanding the system's internal processes and data flow. ([123projectlab.com](https://123projectlab.com/dfd-for-library-management-system/?utm_source=chatgpt.com" \o "DFD for Library Management System - 123projectlab.com))