LIBRARY MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

RAJAKUMARAN BHAVANISHRAJ 220701215

RAMYA P 220701217

In partial fulfilment for the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE (AUTONOMOUS)

THANDALAM

CHENNAI-602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this project report "**LIBRARY MANAGEMENT SYSTEM**"

is the bonafide work of

" RAJAKUMARAN BHAVANISHRAJ (220701215) &

RAMYA P (220701217)"

who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

I would like to extend my sincere gratitude to everyone who has contributed to the successful completion of this mini project.

First and foremost, I am deeply thankful to my Professor **Mrs. K. Maheshmeena ,** my project advisor, for their invaluable guidance, insightful feedback, and continuous support throughout the duration of this mini project. Their expertise and encouragement have been instrumental in shaping my research and bringing this mini project to fruition.

I would also like to express my appreciation to the faculty and staff of the **Computer Science and Engineering** Department at **Rajalakshmi Engineering College** for providing the necessary resources and a conducive learning environment. We express our sincere thanks to **Dr. P. Kumar, M.E., Ph.D.,** Professor and Head of the Department Computer Science and Engineering for his guidance and encouragement throughout the project work.

My heartfelt thanks go to my peers and friends for their collaboration, constructive criticism, and moral support.

Thank you all for your contributions, both direct and indirect, to the success of this project.

ABSTRACT

In the digital era, managing library resources and services efficiently is paramount. This project involves developing a Library Management System (LMS) to automate and optimize library operations. The LMS addresses key challenges such as resource management, user administration, inventory control, and circulation.

Key features of the LMS include a user-friendly interface, advanced search capabilities patrons. The system also includes automated notifications and robust data security measures. By automating routine tasks, the LMS enhances operational efficiency, allowing librarians to focus on strategic activities.

The implementation of the LMS offers significant benefits, including improved accuracy in material tracking, enhanced user satisfaction through easy access to resources, and data-driven decision-making supported by detailed reporting and analytics. This project highlights the potential of technology to modernize library operations and meet the evolving needs of patrons in an information-driven world.

TABLE OF CONTENTS

CHAPTER – 1

1.1 INTRODUCTION:

Libraries have long been repositories of knowledge, serving as essential resources for students, researchers, and the public. However, as the volume of materials and the demand for services increase, the need for efficient and effective management of library operations becomes critical. This is where a Library Management System comes into play.

A Library Management System is a software application designed to automate and manage various library functions, including cataloguing, circulation, inventory, and patron management. The primary objective of an Library Management System is to streamline the operations of a library, making it easier for librarians to manage resources and for patrons to access the information they need.

1.2 OBJECTIVES:

The creation of a Database Management System (DBMS) project for a library system aims to achieve several key objectives:

1. Streamlined Resource Management:

Advanced Search Functionality: Enable users to quickly locate materials using various search criteria.

2. Enhanced User Experience:

Patron Account Management: Maintain detailed records of patrons, including current check-ins and checkouts.

3. Improved Inventory Control:

Accurate Tracking: Keep precise records of all library materials, including acquisitions and removals.

4. Robust Security Measures:

Access Control: Ensure that only authorized users can access specific system features and data.

5. User-Friendly Interface:

Intuitive Design: Create an easy-to-use interface for librarians

6. Cost Efficiency:

Resource Optimization: Reduce costs associated with manual processes and physical paperwork.

Time Savings: Minimize the time required for routine tasks, freeing up staff for more strategic activities.

By achieving these objectives, the DBMS project aims to improve the overall efficiency, accuracy, and quality of library management, ultimately leading to better patient outcomes and streamlined administrative processes.

1.3 MODULES

1. Admin Management Module:

Admin Authentication: Manages admin login and access control, allows the admin to add data and manage both the Student and Book Data.

2. Catalogue Management Module:

Cataloguing: Facilitates the entry and classification of new materials, including books, journals, and digital resources.

Advanced Search: Provides powerful search tools for users to find materials by various criteria (title, author, subject, keywords).

3. Check-In/Check-Out: Manages the borrowing and returning of library materials.

4. Inventory Tracking: Keeps a detailed record of all items in the library's collection.

6. User Activity Reports: Tracks user activity and engagement with the library system.

7. Security and Access Control Module:

User Roles: Defines and manages different admin roles and their access permissions.

Data Security: Implements measures to protect sensitive data by providing admin passwords.

CHAPTER 2

SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION:

Operating System	Windows 11 pro
Python Version	3.10.2
Database	SQLite
GUI	TKinter
Mysql -connector-python	For database connectivity with the program

This Library Management System (LMS) is a software solution developed using Python programming language with SQLite as the database management system and Tkinter for the graphical user interface (GUI). Together, these technologies create a robust and user-friendly page for Library Management System

2.2 Languages

2.2.1 Python

Python powers the entire Library Management System (LMS), serving as the core programming language for its development.

1. **Backend Logic**: Python handles the backend logic of the LMS, including functionalities like book management, member management, borrowing and returning processes, and generating reports. Python's versatility allows for efficient implementation of complex business rules and algorithms required for these tasks.

2. **Database Connectivity**: Python interacts with the SQLite database, enabling seamless storage and retrieval of library data. Through Python's built-in SQLite module, the LMS can execute SQL queries to manage book records, member information, and transaction history efficiently.

3 .**Tkinter GUI Development**: Python's Tkinter library is utilized to create the graphical user interface (GUI) for the LMS. Tkinter provides a simple and intuitive way to design interactive interfaces, allowing librarians and members to interact with the system effortlessly. Python's integration with Tkinter facilitates the creation of windows, buttons, text boxes, and other GUI elements required for user interaction.

4. **Error Handling and Exception Management**: Python's robust error handling capabilities ensure smooth operation of the LMS by gracefully handling exceptions and errors that may occur during runtime. Proper error handling enhances the reliability and stability of the system, providing a seamless user experience.

5. **Third-party Libraries**: Python's extensive ecosystem of third-party libraries enriches the functionality of the LMS. Additional libraries such as Pandas can be leveraged for advanced data analysis and reporting, further enhancing the system's capabilities.

2.2.2 SQLite

SQLite plays a crucial role in the Library Management System (LMS) as the database management system.

1. **Data Storage**: SQLite serves as the primary data storage mechanism for the LMS, housing information such as book details, member records, transaction history, and other relevant data. Its lightweight nature makes it ideal for embedded applications like the LMS, ensuring efficient data storage and retrieval without the need for a separate database server.

2. **Data Manipulation**: SQLite enables the LMS to perform various data manipulation operations, including inserting new records, updating existing data, deleting obsolete entries, and querying information based on specific criteria. Through SQL (Structured Query

Language), the LMS interacts with SQLite to manage the library's database efficiently.

3. **Data Integrity**: SQLite ensures data integrity within the LMS by supporting features such as transactions, constraints, and foreign key relationships. These mechanisms help maintain the consistency and reliability of the database, preventing data corruption and ensuring accurate information management.

4. **Portability**: SQLite's portability allows the LMS to be deployed across different platforms without compatibility issues. Since SQLite databases are stored in a single file, the LMS can easily be transferred between systems, making it convenient for deployment and distribution.

CHAPTER 3
REQUIREMENT ANALYSIS

3.1 REQUIREMENT SPECIFICATION:

Functional Requirements:

1. Book Management:

- Add new books to the system with details such as title, author, ISBN, genre, and availability status.
- Update existing book records.
- Delete books that are no longer in circulation.
- Automatically update availability status when books are borrowed or returned.

2. Member Management:

- Add new members to the library system.

3. Borrowing and Returning Books:

- Allow members to borrow books by providing their membership ID and selecting desired books.
- Record borrowing transactions and update book availability accordingly.
- Provide functionality for members to return borrowed books.

4. Search and Filter Functionality:

- Allow users to search for books by title, author, genre, or ISBN.
- Provide advanced search options and filters for precise book retrieval.
- Display search results in a user-friendly manner.

5. Reporting and Analytics:

- Generate reports such as overdue books, most borrowed books, and membership statistics.
- Provide insights into library usage patterns.
- Present reports in a clear and understandable format.

Non-Functional Requirements

1. Usability:

- The system should have an intuitive and user-friendly interface.
- Navigation should be easy and straightforward for both librarians and library members.

4.2 Performance:

- The system should be responsive and performant, even with a large volume of data.
- Queries and transactions should execute quickly to minimize waiting time for users.

4.3 Security:

- User authentication should be implemented to ensure secure access to the system.
- Member and librarian data should be stored securely to protect privacy.

4.4 Reliability:

- The system should be robust and reliable, with minimal downtime.
- Error handling should be implemented to gracefully manage exceptions and errors.

5. Constraints:

- The system will be developed using Python programming language.
- SQLite will be used as the database management system.
- The graphical user interface will be built using Tkinter library.

6. Assumptions:

- The system will be used in a single-library environment.
- The system will be deployed on desktop computers running Windows, Linux, or macOS.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

HARDWARE SPECIFICATION

PROCESSOR : INTEL i3

MEMORY SIZE : 4GB

HDD : 256GB

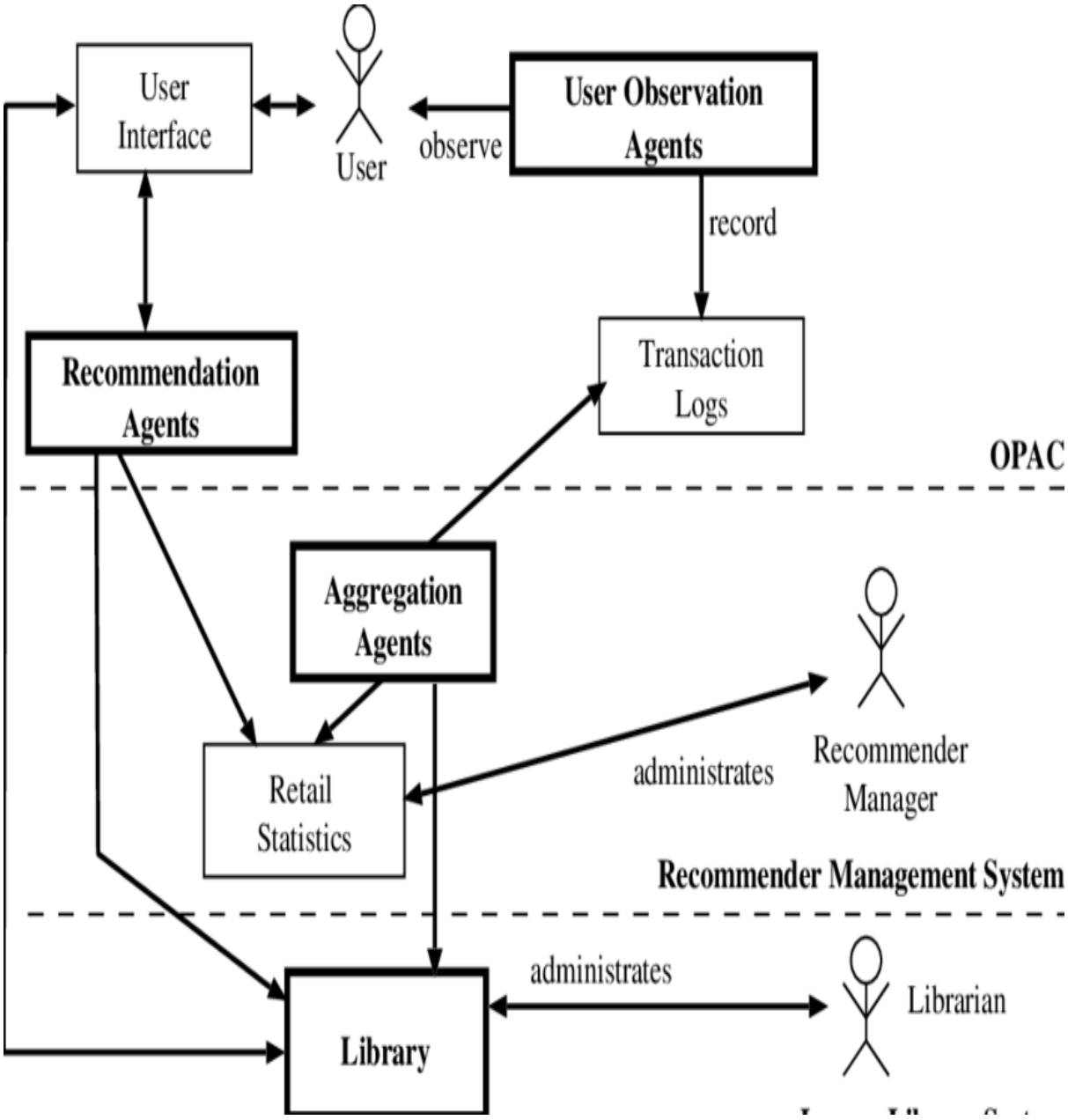
SOFTWARE SPECTFICATION

OPERATING SYSTEM : WINDOWS 11

GUI INTERFACE : PYTHON

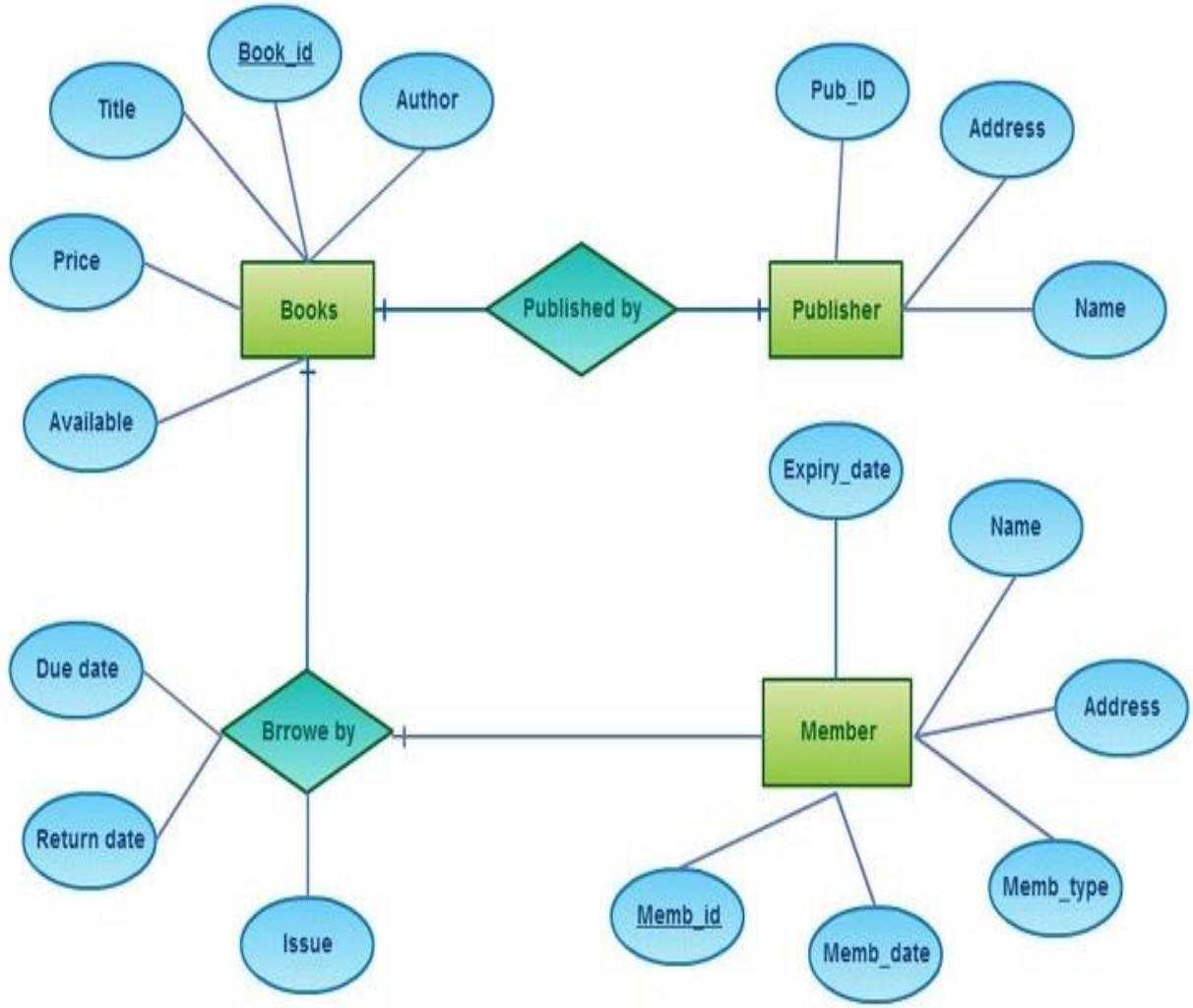
BACKEND : SQLite

3.3 ARCHITECTURE DIAGRAM:



3.4 ER DIAGRAM:

An Entity-Relationship (ER) diagram is a visual representation of the entities in a database and the relationships between them. For a Library Management System, the ER diagram helps to model the various components and their interactions within the system.



3.5 NORMALIZATION:

Normalization is the process of organizing the attributes and tables of a relational database to minimize redundancy and dependency. In the context of a Library Management System (LMS), normalization ensures efficient data storage and retrieval while reducing the risk of data anomalies. Here's how normalization can be applied to an LMS:

1. First Normal Form (1NF):

- Each table should have a primary key to uniquely identify each record.
- Ensure atomicity by avoiding repeating groups within tables.
- For example, the Books table should have a primary key (e.g., Book_ID) and should not contain repeating groups like multiple authors in a single field.

2. Second Normal Form (2NF):

- Ensure that each non-key attribute is fully functionally dependent on the primary key.
- Break down tables to remove partial dependencies.
- For example, if the Books table has attributes like Book_ID, Title, Author, and Author_Address, where Author_Address depends only on Author, create a separate Authors table with Author_ID as the primary key and move Author and Author_Address to it.

3. Third Normal Form (3NF):

- Eliminate transitive dependencies by ensuring that non-key attributes depend only on the primary key.
- Break down tables further to remove dependencies on non-key attributes.
- For example, if the Books table has attributes like Book_ID, Genre, and Genre_Description, where Genre_Description depends only on Genre, create a separate Genres table with Genre_ID as the primary key and move Genre and Genre_Description to it.

Additional Considerations:

- Create separate tables for entities with many-to-many relationships, such as Books and Members, using junction tables.
- Use foreign keys to establish relationships between tables and enforce referential integrity.

- Normalize tables to an appropriate level based on the specific requirements and complexities of the LMS.

Normalized Tables Example:

Books Table:

Book ID	Title	Author	GENRE	COPIES	LOCATION
1	Harry Potter	JK Rowling	Fiction	3	Chennai
2	Wings Of Fire	Abdul Kalam	Motivation	5	Chennai

Authors Table:

Author ID	Author
1	Author1
2	Author2

Genres Table:

Genre ID	Genre	Description
101	Fiction	...
102	Non-Fiction	...

Junction Table (Books Authors):

Book ID	Author ID
1	1
1	2
2	2

CHAPTER - 4

PYTHON CODE:

```
from tkinter import*
from tkinter import ttk
from tkinter import messagebox
from PIL import Image,ImageTk
import random
import sqlite3
image1='booknew.png'
image2='mdl.png'
image3='mainnew.png'
#pip install Pillow
class menu:

    def __init__(self):
        self.root=Tk()
        self.root.title('Menu')
        self.root.state('zoomed')
        conn=sqlite3.connect('test.db')
        conn.execute('''create table if not exists book_info
        (ID VARCHAR PRIMARY KEY NOT NULL,
        TITLE VARTEXT NOT NULL,
        AUTHOR VARTEXT NOT NULL,
        GENRE VARTEXT NOT NULL,
        COPIES VARINT NOT NULL,
        LOCATION VARCHAR NOT NULL);''')
        conn.commit()
        conn.execute('''create table if not exists book_issued
        (BOOK_ID VARCHAR NOT NULL,
        STUDENT_ID VARCHAR NOT NULL,
        ISSUE_DATE DATE NOT NULL,
        RETURN_DATE DATE NOT NULL,
```

```
        PRIMARY KEY (BOOK_ID,STUDENT_ID));'")
        conn.commit()
        conn.close()
        self.a=self.canvases(image1)
        l1=Button(self.a,text='BOOK DATA',font='Papyrus 22
bold',fg='Yellow',bg='Black',width=19,padx=10,borderwidth=0,comm
and=self.book).place(x=100,y=500)
        l2=Button(self.a,text='STUDENT DATA',font='Papyrus 22
bold',fg='Yellow',bg='Black',width=19,padx=10,borderwidth=0,comm
and=self.student).place(x=800,y=500)
        self.root.mainloop()
    def canvases(self,images):
        w = self.root.winfo_screenwidth()
        h = self.root.winfo_screenheight()
        #photo=PhotoImage(file=images)
        photo=Image.open(images)
        photo1=photo.resize((w,h),Image.LANCZOS)
        photo2=ImageTk.PhotoImage(photo1)

        #photo2 = ImageTk.PhotoImage(Image.open(images).resize((w,
h)),Image.LANCZOS)
        self.canvas = Canvas(self.root, width='%d'%w, height='%d'%h)
        self.canvas.grid(row = 0, column = 0)
        self.canvas.grid_propagate(0)
        self.canvas.create_image(0, 0, anchor = NW, image=photo2)
        self.canvas.image=photo2
        return self.canvas
    def book(self):
        self.a.destroy()
        self.a=self.canvases(image2)
        l1=Button(self.a,text='Add Books',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.addbo
ok).place(x=12,y=100)
```

```
        l2=Button(self.a,text='Search Books',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.searc
h).place(x=12,y=200)

        l4=Button(self.a,text='All Books',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.all).pl
ace(x=12,y=300)
        l4=Button(self.a,text='<< Main Menu',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.main
menu).place(x=12,y=500)




    def addbook(self):
        self.aid=StringVar()
        self.aauthor=StringVar()
        self.aname=StringVar()
        self.acopies=IntVar()
        self.agenre=StringVar()
        self.aloc=StringVar()
        self.f1=Frame(self.a,height=500,width=650,bg='black')
        self.f1.place(x=500,y=100)
        l1=Label(self.f1,text='Book ID : ',font='Papyrus 12
bold',fg='Orange',bg='Black',pady=1).place(x=50,y=50)

e1=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.aid
).place(x=150,y=50)
        l2=Label(self.f1,text='Title : ',font='Papyrus 12
bold',fg='Orange',bg='Black',pady=1).place(x=50,y=100)

e2=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.an
ame).place(x=150,y=100)
```

```
        l3=Label(self.f1,text='Author : ',font='Papyrus 12
bold',fg='orange',bg='Black',pady=1).place(x=50,y=150)

e3=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.aa
uthor).place(x=150,y=150)
        l4=Label(self.f1,text='Genre : ',font='Papyrus 12
bold',fg='orange',bg='Black',pady=1).place(x=50,y=200)

e2=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.ag
enre).place(x=150,y=200)
        l4=Label(self.f1,text='Copies : ',font='Papyrus 12
bold',fg='orange',bg='Black',pady=1).place(x=50,y=250)

e2=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.ac
opies).place(x=150,y=250)
        l5=Label(self.f1,text='Location : ',font='Papyrus 12
bold',fg='orange',bg='Black',pady=1).place(x=50,y=300)

e3=Entry(self.f1,width=45,bg='orange',fg='black',textvariable=self.alo
c).place(x=150,y=300)
        self.f1.grid_propagate(0)
        b1=Button(self.f1,text='Add',font='Papyrus 10
bold',fg='black',bg='orange',width=15,bd=3,command=self.adddata).p
lace(x=150,y=400)
        b2=Button(self.f1,text='Back',font='Papyrus 10
bold',fg='black',bg='orange',width=15,bd=3,command=self.rm).place(
x=350,y=400)

    def rm(self):
        self.f1.destroy()
    def mainmenu(self):
        self.root.destroy()
        a=menu()

    def adddata(self):
```

```
        a=self.aid.get()
        b=self.aname.get()
        c=self.aauthor.get()
        d=self.agenre.get()
        e=self.acopies.get()
        f=self.aloc.get()
        conn=sqlite3.connect('test.db')
        try:
            if (a and b and c and d  and f)=="":
                messagebox.showinfo("Error","Fields cannot be empty.")
            else:
                conn.execute("insert into book_info \
                values
(?,?,?,?,?,?)",(a.capitalize(),b.capitalize(),c.capitalize(),d.capitalize(),e,
f.capitalize(),));
                conn.commit()
                messagebox.showinfo("Success","Book added
successfully")
        except sqlite3.IntegrityError:
            messagebox.showinfo("Error","Book is already present.")



        conn.close()

    def search(self):
        #self.search.state('zoomed')
        self.sid=StringVar()
        self.f1=Frame(self.a,height=500,width=650,bg='black')
        self.f1.place(x=500,y=100)
        l1=Label(self.f1,text='Book ID/Title/Author/Genre:
',font=('Papyrus 10 bold'),bd=2,
fg='orange',bg='black').place(x=20,y=40)

e1=Entry(self.f1,width=25,bd=5,bg='orange',fg='black',textvariable=s
elf.sid).place(x=260,y=40)
```

```
        b1=Button(self.f1,text='Search',bg='orange',font='Papyrus 10
bold',width=9,bd=2,command=self.search1).place(x=500,y=37)
        b1=Button(self.f1,text='Back',bg='orange',font='Papyrus 10
bold',width=10,bd=2,command=self.rm).place(x=250,y=450)

    def create_tree(self,plc,lists):

self.tree=ttk.Treeview(plc,height=13,column=(lists),show='headings')
        n=0
        while n is not len(lists):
            self.tree.heading("#"+str(n+1),text=lists[n])
            self.tree.column(""+lists[n],width=100)
            n=n+1
        return self.tree


    def search1(self):
        k=self.sid.get()
        if k!="":
            self.list4=("BOOK
ID","TITLE","AUTHOR","GENRE","COPIES","LOCATION")
            self.trees=self.create_tree(self.f1,self.list4)
            self.trees.place(x=25,y=150)
            conn=sqlite3.connect('test.db')

            c=conn.execute("select * from book_info where ID=? OR
TITLE=? OR AUTHOR=? OR
GENRE=?",(k.capitalize(),k.capitalize(),k.capitalize(),k.capitalize(),))
            a=c.fetchall()
            if len(a)!=0:
                for row in a:

                    self.trees.insert("",END,values=row)
                conn.commit()
                conn.close()
```

```
            self.trees.bind('<<TreeviewSelect>>')
            self.variable = StringVar(self.f1)
            self.variable.set("Select Action:")


            self.cm =ttk.Combobox(self.f1,textvariable=self.variable
,state='readonly',font='Papyrus 15 bold',height=50,width=15,)
            self.cm.config(values =('Add Copies', 'Delete Copies',
'Delete Book'))

            self.cm.place(x=50,y=100)
            self.cm.pack_propagate(0)


            self.cm.bind("<<ComboboxSelected>>",self.combo)
            self.cm.selection_clear()
        else:
            messagebox.showinfo("Error","Data not found")



    else:
        messagebox.showinfo("Error","Search field cannot be
empty.")


  def combo(self,event):
    self.var_Selected = self.cm.current()
    #l7=Label(self.f1,text='copies to update: ',font='Papyrus 10
bold',bd=1).place(x=250,y=700)
    if self.var_Selected==0:
        self.copies(self.var_Selected)
    elif self.var_Selected==1:
        self.copies(self.var_Selected)
    elif self.var_Selected==2:
```

```
            self.deleteitem()
    def deleteitem(self):
        try:
            self.curItem = self.trees.focus()

            self.c1=self.trees.item(self.curItem,"values")[0]
            b1=Button(self.f1,text='Update',font='Papyrus 10
bold',width=9,bd=3,command=self.delete2).place(x=500,y=97)

        except:
            messagebox.showinfo("Empty","Please select something.")
    def delete2(self):
        conn=sqlite3.connect('test.db')
        cd=conn.execute("select * from book_issued where
BOOK_ID=?",(self.c1,))
        ab=cd.fetchall()
        if ab!=0:
            conn.execute("DELETE FROM book_info where
ID=?",(self.c1,));
            conn.commit()
            messagebox.showinfo("Successful","Book Deleted
sucessfully.")
            self.trees.delete(self.curItem)
        else:
            messagebox.showinfo("Error","Book is Issued.\nBook cannot
be deleted.")
        conn.commit()
        conn.close()


    def copies(self,varr):
        try:
            curItem = self.trees.focus()
            self.c1=self.trees.item(curItem,"values")[0]
            self.c2=self.trees.item(curItem,"values")[4]
```

```
            self.scop=IntVar()
            self.e5=Entry(self.f1,width=20,textvariable=self.scop)
            self.e5.place(x=310,y=100)
            if varr==0:
                b5=Button(self.f1,text='Update',font='Papyrus 10
bold',bg='orange',fg='black',width=9,bd=3,command=self.copiesadd).
place(x=500,y=97)
            if varr==1:
                b6=Button(self.f1,text='Update',font='Papyrus 10
bold',bg='orange',fg='black',width=9,bd=3,command=self.copiesdelet
e).place(x=500,y=97)
        except:
            messagebox.showinfo("Empty","Please select something.")

    def copiesadd(self):
        no=self.e5.get()
        if int(no)>=0:

            conn=sqlite3.connect('test.db')

            conn.execute("update book_info set COPIES=COPIES+?
where ID=?",(no,self.c1,))
            conn.commit()

            messagebox.showinfo("Updated","Copies added sucessfully.")
            self.search1()
            conn.close()

        else:
            messagebox.showinfo("Error","No. of copies cannot be
negative.")

    def copiesdelete(self):
        no1=self.e5.get()
        if int(no1)>=0:
```

```
            if int(no1)<=int(self.c2):
                conn=sqlite3.connect('test.db')

                conn.execute("update book_info set COPIES=COPIES-?
where ID=?",(no1,self.c1,))
                conn.commit()
                conn.close()

                messagebox.showinfo("Updated","Deleted sucessfully")
                self.search1()

            else:
                messagebox.showinfo("Maximum","No. of copies to delete
exceed available copies.")
        else:
            messagebox.showinfo("Error","No. of copies cannot be
negative.")

    def all(self):
        self.f1=Frame(self.a,height=500,width=650,bg='black')
        self.f1.place(x=500,y=100)
        b1=Button(self.f1,text='Back',bg='orange'
,fg='black',width=10,bd=3,command=self.rm).place(x=250,y=400)
        conn=sqlite3.connect('test.db')
        self.list3=("BOOK
ID","TITLE","AUTHOR","GENRE","COPIES","LOCATION")
        self.treess=self.create_tree(self.f1,self.list3)
        self.treess.place(x=25,y=50)
        c=conn.execute("select * from book_info")
        g=c.fetchall()
        if len(g)!=0:
            for row in g:
                self.treess.insert('',END,values=row)
        conn.commit()
        conn.close()
```

```
    def student(self):
        self.a.destroy()
        self.a=self.canvases(image2)
        l1=Button(self.a,text='Issue book',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.issue)
.place(x=12,y=100)
        l2=Button(self.a,text='Return Book',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.return
n).place(x=12,y=200)
        l3=Button(self.a,text='Student Activity',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.activit
y).place(x=12,y=300)
        l4=Button(self.a,text='<< Main Menu',font='Papyrus 22
bold',fg='Orange',bg='Black',width=15,padx=10,command=self.main
menu).place(x=12,y=600)




    def issue(self):
        self.aidd=StringVar()
        self.astudentt=StringVar()
        self.f1=Frame(self.a,height=550,width=500,bg='black')
        self.f1.place(x=500,y=100)
        l1=Label(self.f1,text='Book ID : ',font='papyrus 15
bold',bg='black',fg='orange').place(x=50,y=100)

e1=Entry(self.f1,width=25,bd=4,bg='orange',textvariable=self.aidd).pl
ace(x=180,y=100)
        l2=Label(self.f1,text='Student Id : ',font='papyrus 15
bold',bg='black',fg='orange').place(x=50,y=150)

e2=Entry(self.f1,width=25,bd=4,bg='orange',textvariable=self.astuden
tt).place(x=180,y=150)
```

```
        b1=Button(self.f1,text='Back',font='Papyrus 10
bold',fg='black',bg='orange',width=10,bd=3,command=self.rm).place(
x=50,y=250)
        b1=Button(self.f1,text='Issue',font='Papyrus 10
bold',fg='black',bg='orange',width=10,bd=3,command=self.issuedboo
k).place(x=200,y=250)

    def issuedbook(self):
        bookid=self.aidd.get()
        studentid=self.astudentt.get()
        conn=sqlite3.connect('test.db')
        cursor=conn.cursor()
        cursor.execute("select ID,COPIES from book_info where
ID=?",(bookid.capitalize(),))
        an=cursor.fetchall()
        if (bookid and studentid!=""):
            if an!=[]:
                for i in an:
                    if i[1]>0:
                        try:
                            conn.execute("insert into book_issued \
                            values (?,?,date('now'),date('now','+7
day'))",(bookid.capitalize(),studentid.capitalize(),));
                            conn.commit()
                            conn.execute("update book_info set
COPIES=COPIES-1 where ID=?",(bookid.capitalize(),))
                            conn.commit()
                            conn.close()
                            messagebox.showinfo("Updated","Book Issued
sucessfully.")
                        except:
                            messagebox.showinfo("Error","Book is already
issued by student.")

                    else:
```

```
                    messagebox.showinfo("Unavailable","Book
unavailable.\nThere are 0 copies of the book.")
            else:
                messagebox.showinfo("Error","No such Book in
Database.")
        else:
            messagebox.showinfo("Error","Fields cannot be blank.")

    def returnn(self):
        self.aidd=StringVar()
        self.astudentt=StringVar()

        self.f1=Frame(self.a,height=550,width=500,bg='black')
        self.f1.place(x=500,y=100)
        l1=Label(self.f1,text='Book ID : ',font='papyrus 15
bold',fg='orange', bg='black').place(x=50,y=100)

e1=Entry(self.f1,width=25,bd=4,bg='orange',textvariable=self.aidd).pl
ace(x=180,y=100)
        l2=Label(self.f1,text='Student Id : ',font='papyrus 15
bold',fg='orange', bg='black').place(x=50,y=150)

e2=Entry(self.f1,width=25,bd=4,bg='orange',textvariable=self.astuden
tt).place(x=180,y=150)
        b1=Button(self.f1,text='Back',font='Papyrus 10
bold',bg='orange',fg='black',width=10,bd=3,command=self.rm).place(
x=50,y=250)
        b1=Button(self.f1,text='Return',font='Papyrus 10
bold',bg='orange',fg='black',width=10,bd=3,command=self.returnboo
k).place(x=200,y=250)
        self.f1.grid_propagate(0)

    def returnbook(self):
        a=self.aidd.get()
        b=self.astudentt.get()
```

```
        conn=sqlite3.connect('test.db')

        fg=conn.execute("select ID from book_info where
ID=?",(a.capitalize(),))
        fh=fg.fetchall()
        conn.commit()
        if fh!=None:
            c=conn.execute("select * from book_issued where
BOOK_ID=? and STUDENT_ID=?",(a.capitalize(),b.capitalize(),))
            d=c.fetchall()
            conn.commit()
            if len(d)!=0:
                c.execute("DELETE FROM book_issued where
BOOK_ID=? and STUDENT_ID=?",(a.capitalize(),b.capitalize(),));
                conn.commit()
                conn.execute("update book_info set COPIES=COPIES+1
where ID=?",(a.capitalize(),))
                conn.commit()

                messagebox.showinfo("Success","Book Returned
sucessfully.")
            else:
                messagebox.showinfo("Error","Data not found.")
        else:
            messagebox.showinfo("Error","No such book.\nPlease add the
book in database.")
        conn.commit()
        conn.close()

    def activity(self):
        self.aidd=StringVar()
        self.astudentt=StringVar()
        self.f1=Frame(self.a,height=550,width=500,bg='black')
        self.f1.place(x=500,y=80)
```

```
        self.list2=("BOOK ID","STUDENT ID","ISSUE
DATE","RETURN DATE")
        self.trees=self.create_tree(self.f1,self.list2)
        self.trees.place(x=50,y=150)


        l1=Label(self.f1,text='Book/Student ID : ',font='Papyrus 15
bold',fg='Orange',bg='black').place(x=50,y=30)

e1=Entry(self.f1,width=20,bd=4,bg='orange',textvariable=self.aidd).pl
ace(x=280,y=35)
        #l2=Label(self.f1,text='Student Id : ',font='papyrus 15
bold',fg='orange',bg='black').place(x=50,y=80)

#e2=Entry(self.f1,width=20,bd=4,bg='orange',textvariable=self.astude
ntt).place(x=180,y=80)
        b1=Button(self.f1,text='Back',bg='orange',font='Papyrus 10
bold',width=10,bd=3,command=self.rm).place(x=340,y=450)
        b1=Button(self.f1,text='Search',bg='orange',font='Papyrus 10
bold',width=10,bd=3,command=self.searchact).place(x=40,y=450)
        b1=Button(self.f1,text='All',bg='orange',font='Papyrus 10
bold',width=10,bd=3,command=self.searchall).place(x=190,y=450)
        self.f1.grid_propagate(0)

    def searchact(self):
        self.list2=("BOOK ID","STUDENT ID","ISSUE
DATE","RETURN DATE")
        self.trees=self.create_tree(self.f1,self.list2)
        self.trees.place(x=50,y=150)
        conn=sqlite3.connect('test.db')
        bid=self.aidd.get()
        #sid=self.astudentt.get()
        try:
            c=conn.execute("select * from book_issued where
BOOK_ID=? or STUDENT_ID=?",(bid.capitalize(),bid.capitalize(),))
```

```
            d=c.fetchall()
            if len(d)!=0:
                for row in d:
                    self.trees.insert("",END,values=row)
            else:
                messagebox.showinfo("Error","Data not found.")
            conn.commit()

        except Exception as e:
            messagebox.showinfo(e)
        conn.close()

    def searchall(self):
        self.list2=("BOOK ID","STUDENT ID","ISSUE
DATE","RETURN DATE")
        self.trees=self.create_tree(self.f1,self.list2)
        self.trees.place(x=50,y=150)
        conn=sqlite3.connect('test.db')
        try:
            c=conn.execute("select * from book_issued")
            d=c.fetchall()
            for row in d:
                self.trees.insert("",END,values=row)

            conn.commit()

        except Exception as e:
            messagebox.showinfo(e)
        conn.close()

#====================START=======================
def canvases(images,w,h):
    photo=Image.open(images)
    photo1=photo.resize((w,h),Image.LANCZOS)
```

```
    photo2=ImageTk.PhotoImage(photo1)

#photo2 = ImageTk.PhotoImage(Image.open(images).resize((w,
h)),Image.LANCZOS)
    canvas = Canvas(root, width='%d'%w, height='%d'%h)
    canvas.grid(row = 0, column = 0)
    canvas.grid_propagate(0)
    canvas.create_image(0, 0, anchor = NW, image=photo2)
    canvas.image=photo2
    return canvas
root = Tk()
root.title("LOGIN")
"""width = 400
height = 280
screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
x = (screen_width/2) - (width/2)
y = (screen_height/2) - (height/2)"""

#root.state('zoomed')
#root.resizable(0, 0)
w = root.winfo_screenwidth()
h = root.winfo_screenheight()
canvas=canvases(image3,w,h)
#photo=PhotoImage(file=images)


#================================METHODS==========
================================
def Database():
    global conn, cursor
    conn = sqlite3.connect("python1.db")
    cursor = conn.cursor()
```

```
    cursor.execute("CREATE TABLE IF NOT EXISTS `login`
(mem_id INTEGER NOT NULL PRIMARY KEY
AUTOINCREMENT, username TEXT, password TEXT)")
    cursor.execute("SELECT * FROM `login` WHERE `username` =
'admin' AND `password` = 'admin'")
    if cursor.fetchone() is None:
        cursor.execute("INSERT INTO `login` (username, password)
VALUES('ramya', 'root')")
        conn.commit()

def Login(event=None):
    Database()


    if USERNAME.get() == "" or PASSWORD.get() == "":
        messagebox.showinfo("Error","Please complete the required
field!")
        lbl_text.config(text="Please complete the required field!",
fg="red")
    else:
        cursor.execute("SELECT * FROM `login` WHERE `username`
= ? AND `password` = ?", (USERNAME.get(), PASSWORD.get()))
        if cursor.fetchone() is not None:
            #HomeWindow()
            #Top.destroy()
            root.destroy()

            #print("hello logged in ")
            a=menu()
            #USERNAME.set("")
            #PASSWORD.set("")
            #lbl_text.config(text="")
        else:
            messagebox.showinfo("Error","Invalid username or
password.")
```

```
        #lbl_text.config(text="Invalid username or password",
fg="red")
        USERNAME.set("")
        PASSWORD.set("")
    cursor.close()
    conn.close()


#================================VARIABLES=========
================================
USERNAME = StringVar()
PASSWORD = StringVar()

#================================FRAMES============
================================
'''Top = Frame(root, bd=2,  relief=RIDGE)
Top.pack(side=TOP, fill=X)
Form = Frame(root, height=200)
Form.pack(side=BOTTOM, pady=20)'''
#================================LABELS============
================================
lbl_title = Label(canvas, text = "ADMIN   LOGIN", font=('Papyrus',
30,'bold', ),bg='black', fg='orange')
lbl_title.place(x=500,y=100)
lbl_username = Label(canvas, text = "Username:", font=('Papyrus',
15,'bold'),bd=4,bg='black', fg='orange')
lbl_username.place(x=500,y=230)
lbl_password = Label(canvas, text = "Password :", font=('Papyrus',
15,'bold'),bd=3, bg='black', fg='orange')
lbl_password.place(x=500, y=330)
lbl_text = Label(canvas)
lbl_text.place(x=450,y=500)
lbl_text.grid_propagate(0)
```

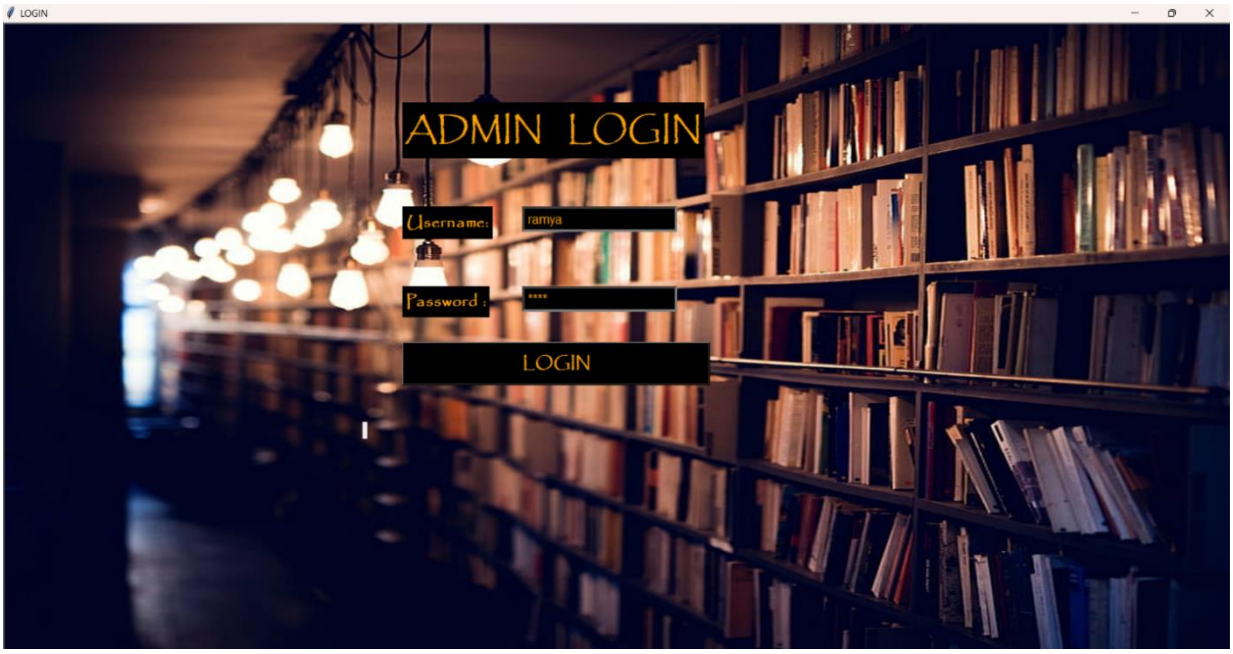
```
#=====================ENTRY
WIDGETS=====================================
username = Entry(canvas, textvariable=USERNAME, font=(14),
bg='black', fg='orange',bd=6)
username.place(x=650, y=230,)
password = Entry(canvas, textvariable=PASSWORD, show="*",
font=(14),bg='black', fg='orange',bd=6)
password.place(x=650, y=330)

#==============================BUTTON
WIDGETS==================================
btn_login = Button(canvas, text="LOGIN", font=('Papyrus 15
bold'),width=25,command=Login, bg='black', fg='orange')
btn_login.place(x=500,y=400)
btn_login.bind('<Return>', Login)
root.mainloop()
```

CHAPTER - 5

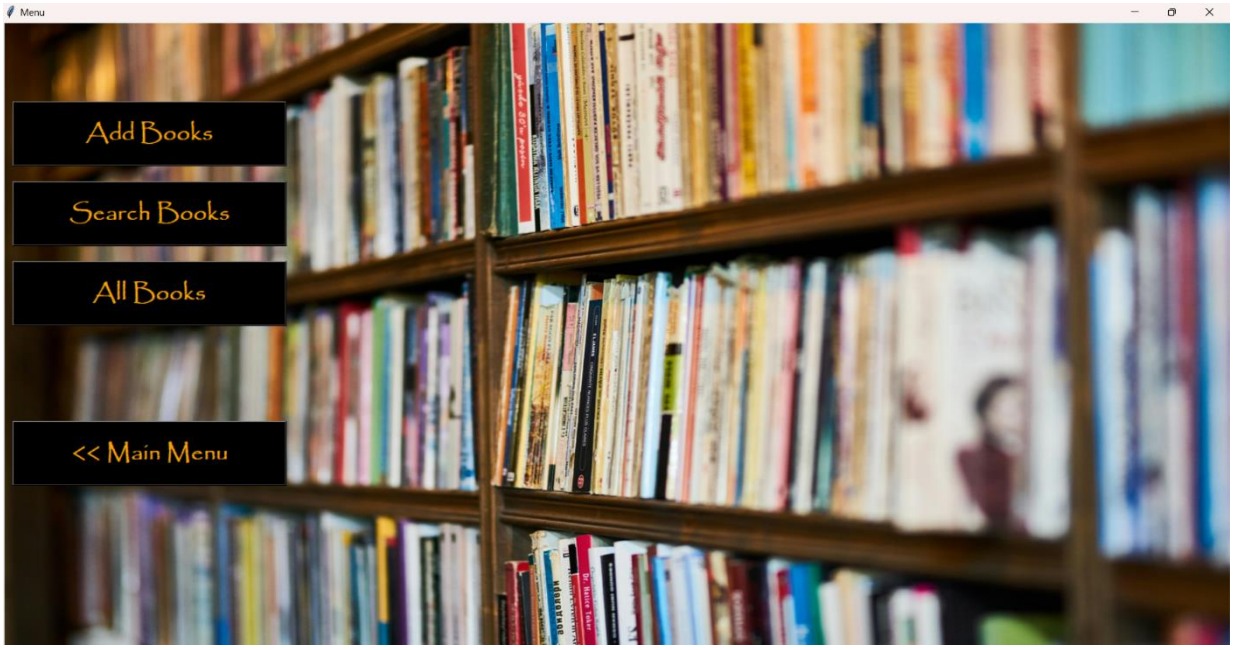
RESULTS AND DISCUSSION

ADMIN LOGIN

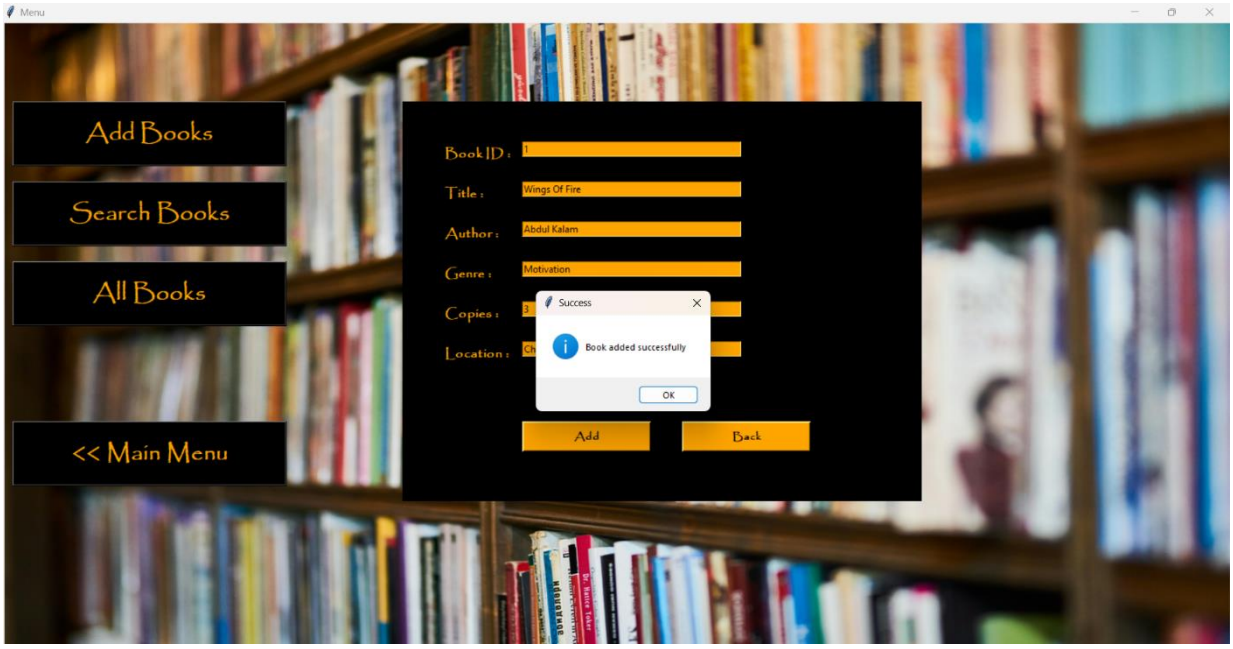


MENU PAGE

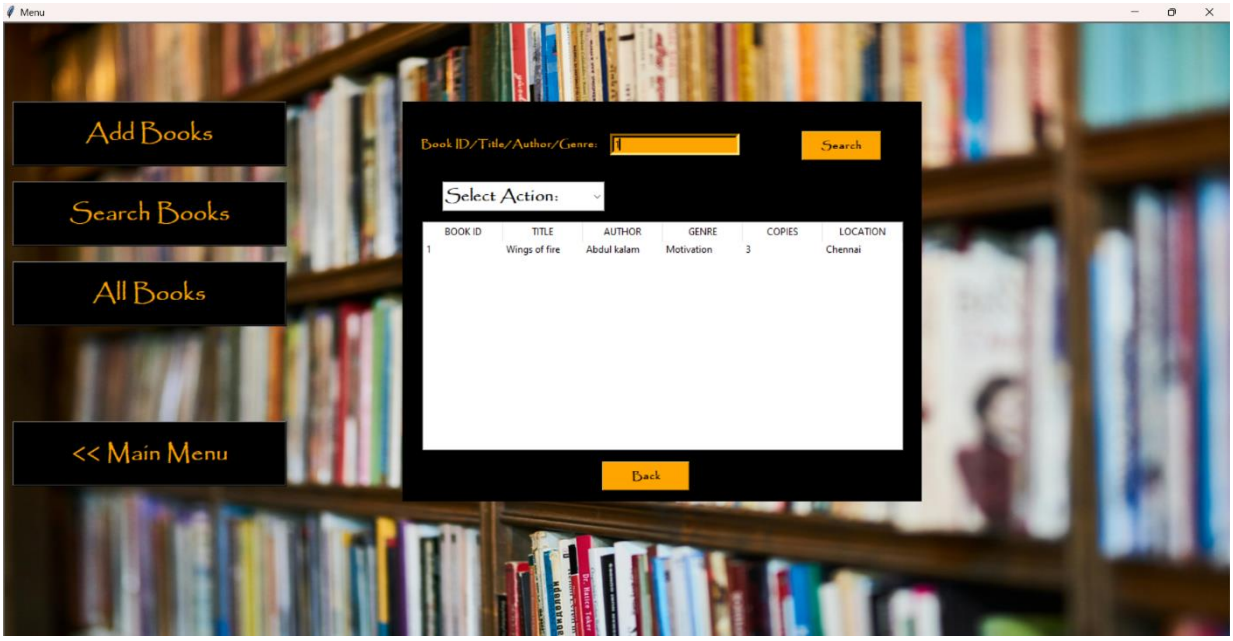
BOOK MENU



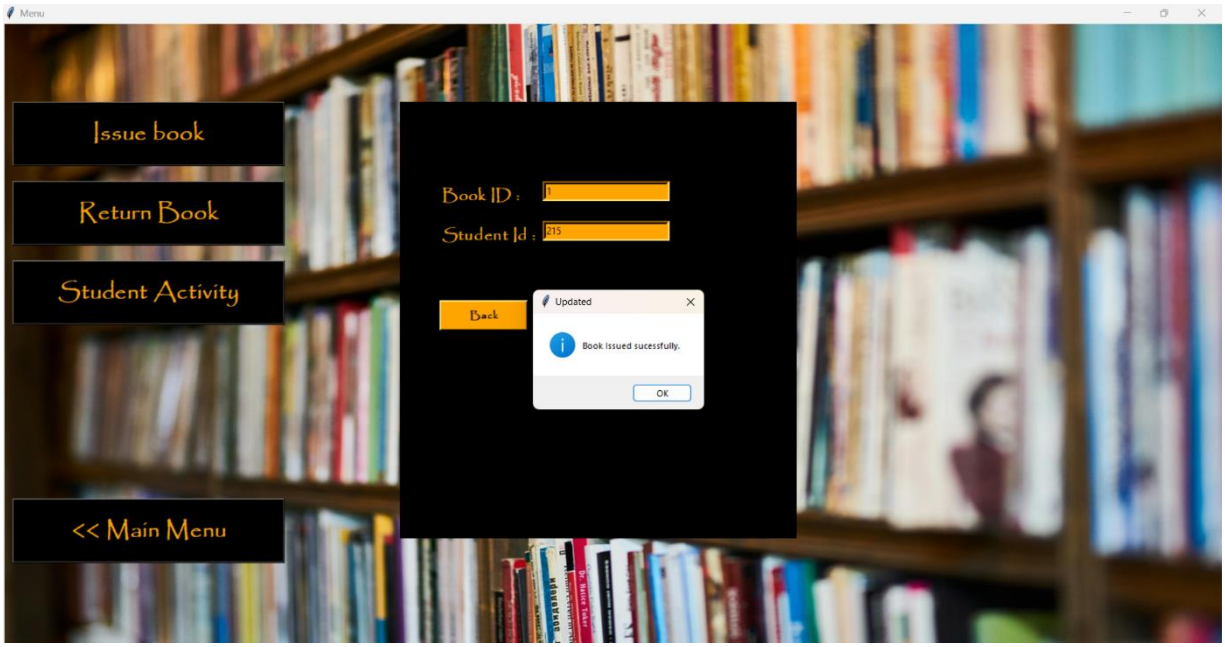
BOOK DETAILS



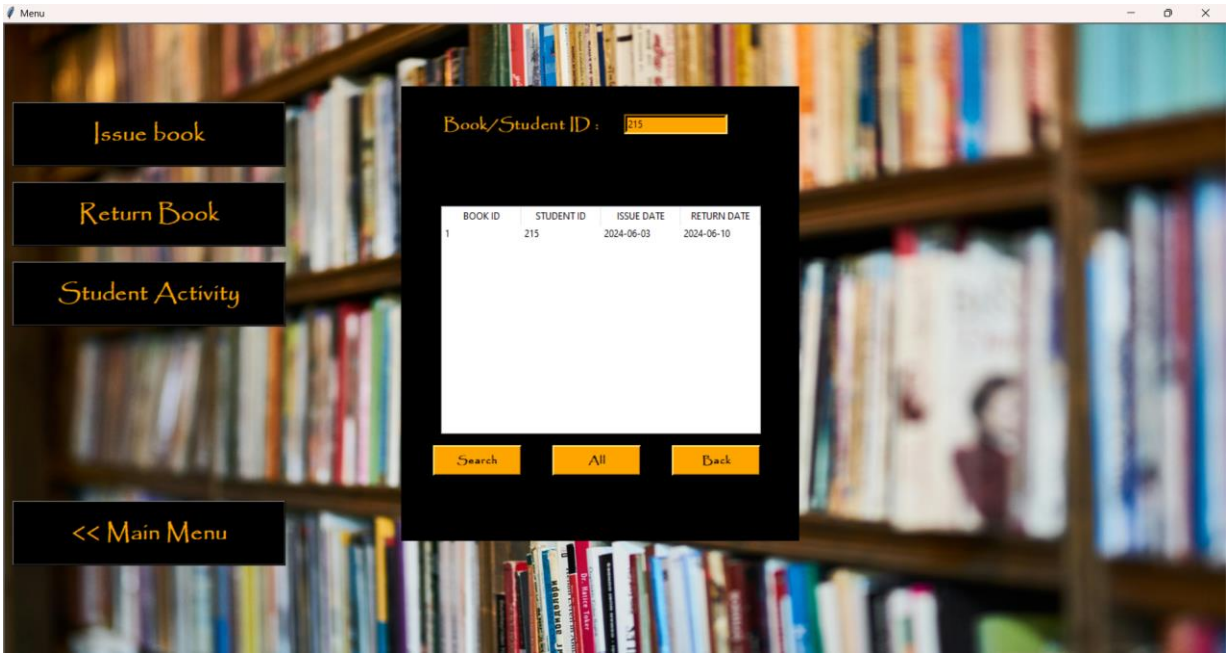
ADD BOOKS



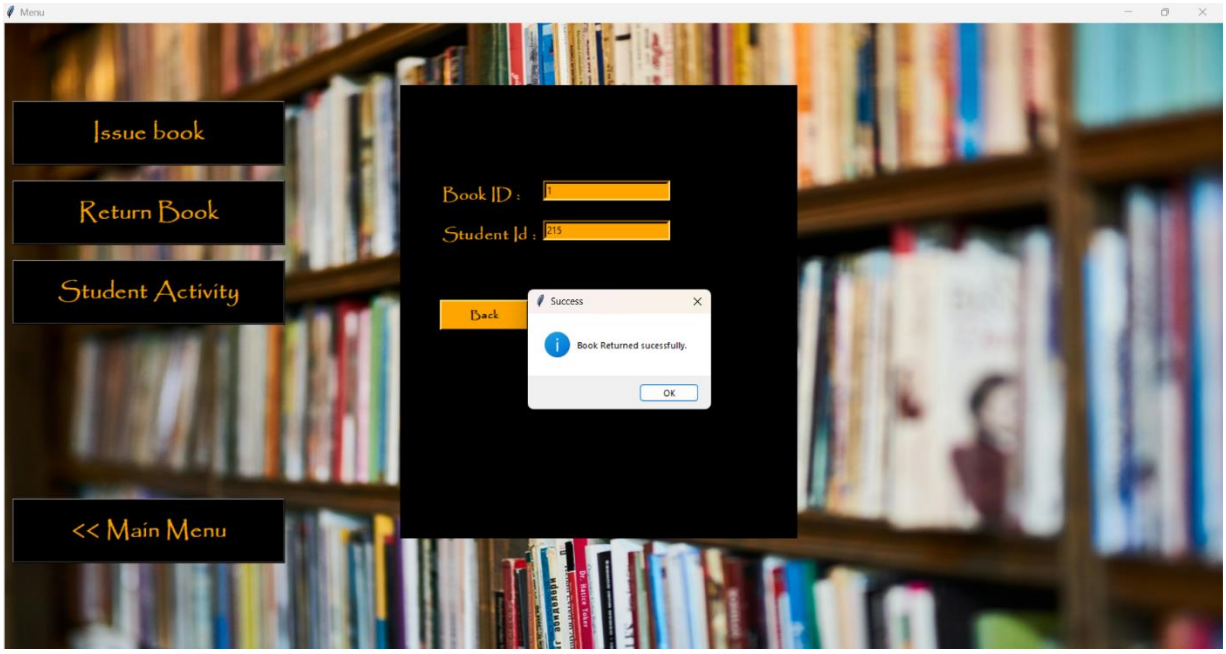
BOOK ISSUED SUCCESSFULLY



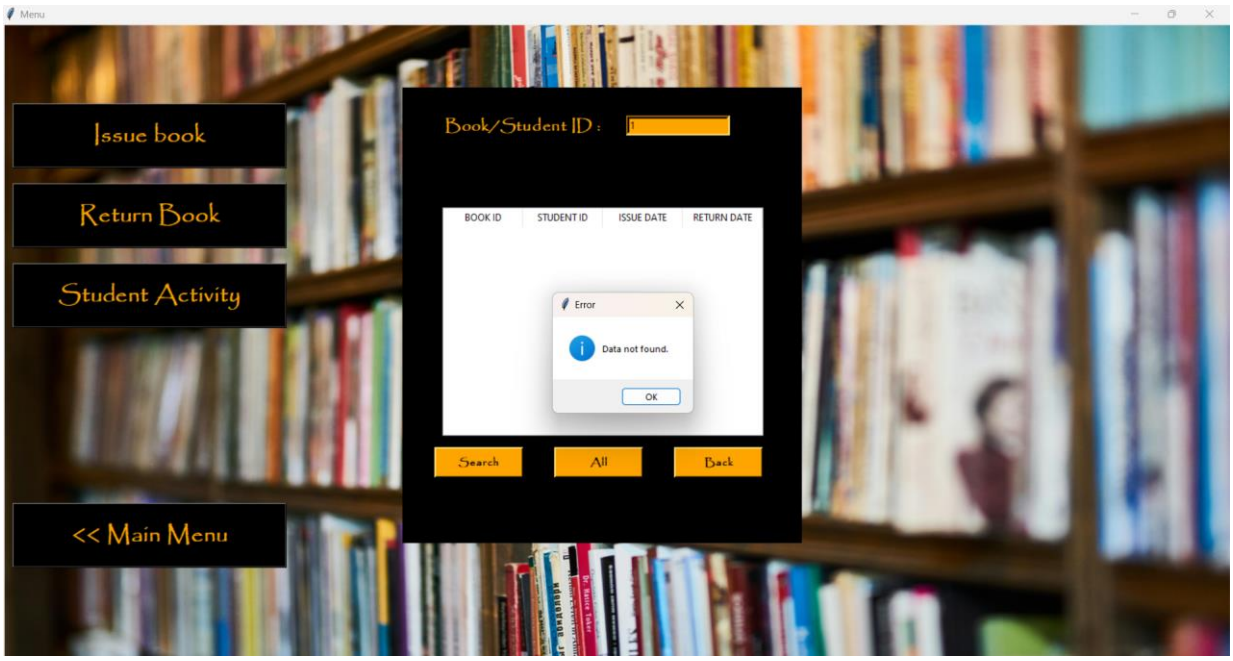
STUDENT ACTIVITY



RETURN BOOK



BOOK RETURNED SUCCESSFULLY



CHAPTER 6

CONCLUSION

In conclusion, the Python Library Management System marks a significant milestone in revolutionizing traditional library management practices. Through the fusion of Python's flexibility, SQLite's efficiency, and Tkinter's intuitive interface, this system offers a comprehensive solution tailored to meet the evolving needs of modern libraries.

By addressing key pain points such as manual data handling and cumbersome administrative tasks, this system empowers librarians to streamline their operations and focus on delivering enhanced services to patrons. From seamless book and member management to efficient borrowing processes and insightful reporting, every aspect of the system is designed to optimize efficiency and user experience.

Moreover, the emphasis on usability, performance, security, and reliability underscores our commitment to delivering a robust and user-friendly solution. By prioritizing user feedback and continuous improvement, we ensure that the Python Library Management System remains relevant and effective in an ever-changing technological landscape.

Looking ahead, the potential impact of this system extends beyond individual libraries to shape the broader landscape of knowledge dissemination and access. As libraries embrace digital transformation, this system serves as a beacon of innovation, empowering institutions to adapt and thrive in the digital age.

In essence, the Python Library Management System represents a paradigm shift in how libraries harness technology to serve their communities better. With its blend of cutting-edge technology and user-centric design, this system paves the way for a new era of library management, where efficiency, accessibility, and innovation converge to enrich the lives of patrons worldwide.

CHAPTER - 7

REFERENCES

1. **"Python Programming: A Modern Approach" by R. Nageswara Rao**: This book covers Python programming concepts in a comprehensive manner, making it suitable for beginners as well as intermediate learners. It provides a solid foundation in Python programming, which is essential for developing a Library Management System.

2. **"Mastering Python Design Patterns" by Sakis Kasampalis**: Design patterns play a crucial role in developing scalable and maintainable software systems. This book explores various design patterns in Python and how they can be applied to real-world projects like a Library Management System.

3. **"Tkinter GUI Application Development Blueprints" by Bhaskar Chaudhary**: Since Tkinter is used for developing the graphical user interface (GUI) of the Library Management System, this book provides valuable insights into building interactive and user-friendly GUI applications using Tkinter. It covers various GUI development techniques and best practices, making it useful for implement