RAJALAKSHMI
ENGINEERING COLLEGE
An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

LIBRARY MANAGEMENT SYSTEM

MINI PROJECT REPORT

Submitted by

SABHARISHRAJA B	**231801143**
SARATH KUMAR P	**231801156**
SABARISH P	**231801142**

CS23332 DATABASE MANAGEMENT SYSTEM

Department of Artificial Intelligence and Data Science

Rajalakshmi Engineering College Thandalam

RAJALAKSHMI ENGINEERING COLLEGE

An AUTONOMOUS Institution
Affiliated to ANNA UNIVERSITY, Chennai

BONAFIDE CERTIFICATE

Certified that this project report "**LIBRARY MANAGEMENT SYSTEM**" is bonafide work of "**SABHARISHRAJA B**(231801143), **SARATH KUMAR P** (231801156), **SABARISH P** (231801142) "who carried out the project work under my supervision.

Submitted for the Practical Examination held on _____

SIGNATURE

Dr.MANORANJINI J
Professor,Artificial Intelligence and Data Science,RajalakshmiEngineering College (Autonomous), Thandalam, Chennai-602105

SIGNATURE

Dr.GNANASEKAR J M
Head of the Department, Artificial intelligence and data Science,Rajalakshmi Engineering College (Autonomous),Chennai-602105

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The Library Management System is a robust solution for managing library operations, built using Python's Tkinter for its user-friendly interface and MongoDB for scalable, efficient database management. It supports key functionalities such as adding, updating, issuing, and returning books, with each record containing details like title, author, unique ID, availability status, and issuer card ID. MongoDB's NoSQL structure ensures fast and reliable handling of large datasets, enabling efficient insertion, updating, and deletion of records. Features like a powerful search function and real-time updates ensure data accuracy and accessibility, streamlining library workflows. Designed for scalability and performance, this system demonstrates the potential of NoSQL databases in handling structured and semi-structured data, making it a future-proof solution for modern library management.

TABLE OF CONTENTS

INTRODUCTION:

This project is a **Library Management System** designed to streamline the management of books and records within a library. Developed using Python's Tkinter for a simple and intuitive user interface and MongoDB as the backend database, this system offers core functionalities such as adding, updating, issuing, and returning books. By leveraging MongoDB's flexible document-based structure, the system can handle large collections of data efficiently and scale as the library grows. This system provides library administrators with a reliable tool to manage book availability, track issuers, and maintain an organized record system, enhancing both library operations and user satisfaction.

Objectives

1. **Efficient Book Tracking:** Streamline the management of book information, including titles, authors, and availability, using MongoDB to enhance data retrieval and storage.

2. **User-Friendly Interface:** Provide a Tkinter interface for seamless interaction, allowing users to easily add, update, delete, and view book records.

3. **Improved Borrowing System:** Implement clear tracking of book issuance and returns with real-time updates to ensure accurate record-keeping.

4. **Data Consistency:** Use MongoDB's robust database structure to maintain data integrity, ensuring accurate and consistent information for all library records.

5. **Scalability and Flexibility:** Design a scalable solution that can handle increased data and functionality as the library collection grows.

MODULES:

For a Library Management System (LMS), essential modules should be structured to support efficient handling of books, users, borrowing activities, and reporting needs. Here's a suggested list of core and additional modules, along with important database considerations:

Core Modules:

- Book Management
- Member Management
- Borrowing and Return Management
- Catalog Search and Browsing

Additional Modules:

- Inventory Management
- Reporting and Analytics
- Supplier and Acquisition Management
- Fines and Payments

Database Considerations:

- DBMS Selection
- Data Normalization
- Security Protocols
- Scalability Design

SURVEY OF TECHNOLOGIES:

Software Description :

This project utilizes a combination of software tools to create a comprehensive and efficient supermarket management system:

- **Database Management System (DBMS):** MongoDB is chosen as the DBMS for the library management system due to its flexibility with unstructured data, ease of scalability, and ability to handle a diverse range of data types efficiently.
- **Integrated Development Environment (IDE):** PyCharm is selected as the IDE for its Python-specific features, code completion, debugging tools, and seamless integration with MongoDB.

Languages :

- **MongoDB**: MongoDB Query Language (MQL) is used to interact with the MongoDB database, enabling data retrieval, document manipulation, and management of collections, making it ideal for handling complex, document-oriented data structures in the library management system.
- **Python**: A versatile programming language is used to develop the backend logic, implement business rules, and interact with the Mongodb database through the pymongo.

2.2.1.MONGODB:

MongoDB plays a crucial role in the library management system by:

- **Storing Document-based Records:** Using collections to store books, members, borrowing history, and other library data in a flexible schema.
- **Efficient Data Retrieval:** Performing targeted queries to find specific books, authors, or borrowing statuses within the database.
- **Managing Data Relationships:** Using document embedding or referencing to handle relationships between books, authors, and borrowers.
- **Generating Insights:** Creating reports through MongoDB aggregation to analyze book popularity, member borrowing patterns, and other key metrics.

2.2.2 PYTHON:

• **Develop backend logic:** Implement core functionalities like book issue/return, fine calculations, and user account management.

• **Interact with the MongoDB database:** Use PyMongo to connect and manage MongoDB collections, enabling flexible data handling and retrieval.

• **Create the user interface:** Design a user-friendly interface with Tkinter, allowing users to perform library tasks and view records.

• **Integrate with external systems:** Connect the library system to other educational or resource management tools for seamless data sharing.

• **Implement data validation:** Ensure accurate data entry, enforce library rules, and maintain system integrity.

REQUIREMENTS AND ANALYSIS :

Requirement Specification

Functional Requirements :

Book Management:

- Add, edit, and delete book records
- Track book availability and current status (issued/available)
- Manage categories and authors

Member Management:

- Create, update, and view member profiles
- Track borrowing history and overdue fines
- Manage contact information and membership status

Transaction Management:

- Issue and return books to members
- Update book availability and member records
- Handle overdue fines and renewals

Reporting and Analytics:

- Generate reports on issued/available books, overdue items, and fines
- Analyze borrowing trends, popular books, and member activity

Non-Functional Requirements

Performance: Efficiently handle numerous member transactions and large book collections.

Scalability: Accommodate a growing number of books, members, and transactions.

Security: Protect sensitive member data and secure access to library records.

Usability: Provide a user-friendly interface for both staff and members.

Reliability: Ensure system stability with minimal downtime and accurate data handling.

Hardware and Software Requirements :

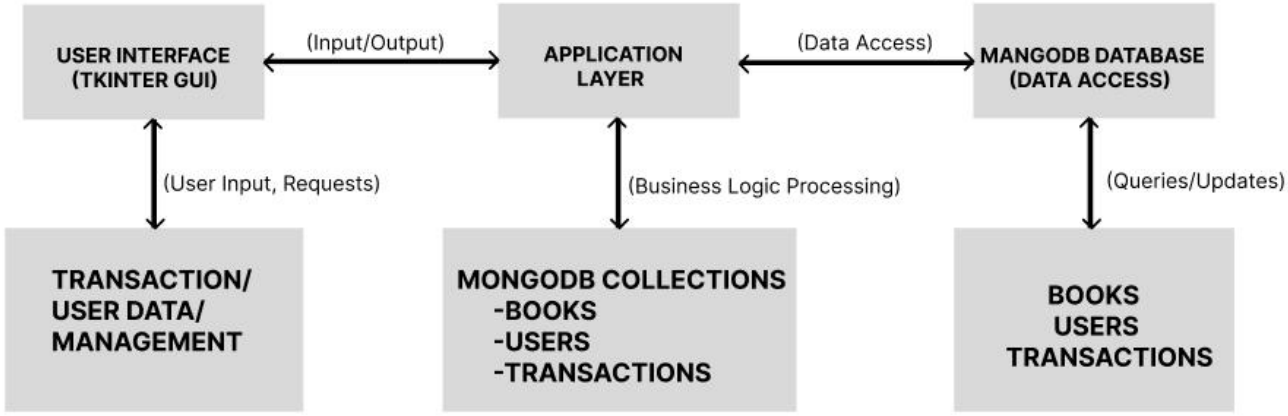
❖Hardware :

- **Server:** A powerful server with sufficient CPU, RAM, and storage to handle the database and application workload.
- **Network:** A reliable network connection to allow access to the system from different locations.
- **POS Terminals:** Point-of-sale terminals for processing sales transactions.

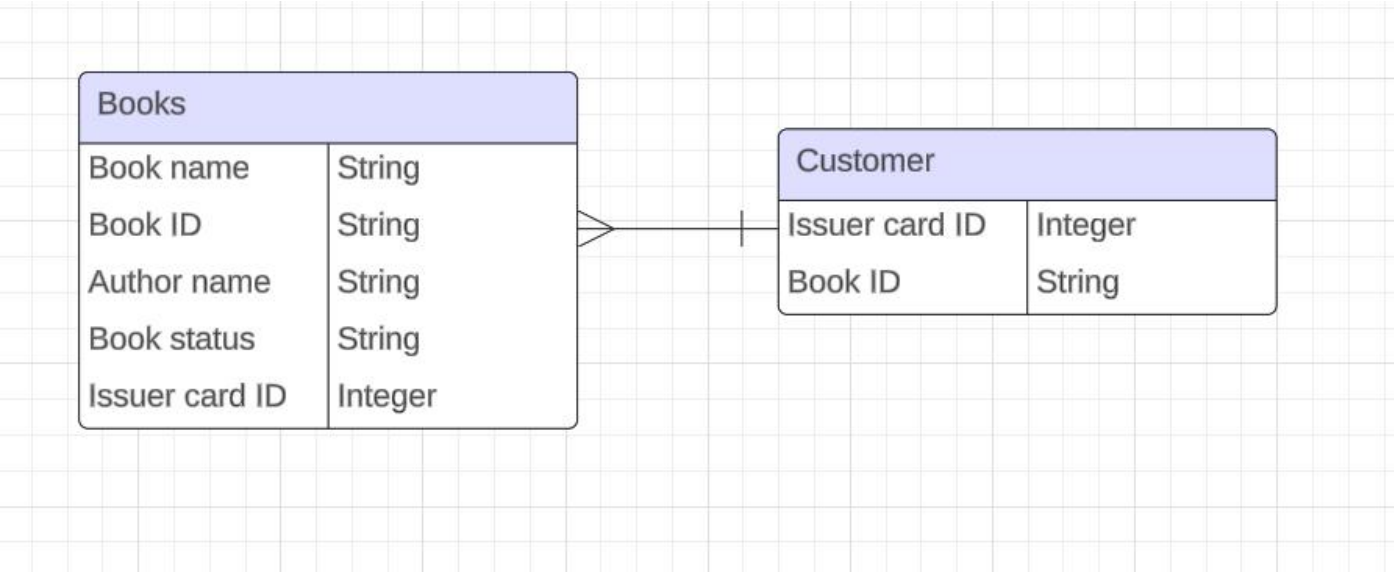
❖Software :

- **Database Management System (DBMS):** MySQL, PostgreSQL, or SQL Server or Mongodb.
- **Integrated Development Environment (IDE):** PyCharm or Visual Studio Code.
- **Operating System:** Linux or Windows.
- **Web Server:** Apache or Nginx.

3.3 ARCHITECTURE DIAGRAM :



3.4 ER DIAGRAM :



PROGRAM CODE:

Login page:

```
import tkinter as tk
from tkinter import messagebox
import random

def generate_captcha():
    captcha_value = ''.join([str(random.randint(0, 9)) for _ in range(4)])
    captcha_var.set(captcha_value)

def login():
    username = entry_username.get()
    password = entry_password.get()
    entered_captcha = entry_captcha.get()

    if username == "admin" and password == "password" and entered_captcha ==
captcha_var.get():
        root.destroy()
        import code2
    elif entered_captcha != captcha_var.get():
        messagebox.showerror("Login Failed", "Incorrect CAPTCHA. Please try again.")
        generate_captcha()
    else:
        messagebox.showerror("Login Failed", "Invalid username or password")

root = tk.Tk()
root.title("Library Management System - Login")
root.geometry("1100x600")

# Background color
root.configure(bg="#6891FF")

# Title Label
label_title = tk.Label(root, text="Library Management System", font=("Arial", 24, "bold"),
bg="#5E4FFF", fg="white")
label_title.pack(pady=40)

# Frame for the login form
frame = tk.Frame(root, bg="white", padx=40, pady=30)
frame.place(relx=0.5, rely=0.5, anchor="center")
```

```
# Username Label and Entry
label_username = tk.Label(frame, text="USERNAME:", font=("Arial", 14), bg="white")
label_username.grid(row=0, column=0, sticky="w", pady=10)
entry_username = tk.Entry(frame, font=("Arial", 14), width=25)
entry_username.grid(row=0, column=1, pady=10)

# Password Label and Entry
label_password = tk.Label(frame, text="PASSWORD:", font=("Arial", 14), bg="white")
label_password.grid(row=1, column=0, sticky="w", pady=10)
entry_password = tk.Entry(frame, font=("Arial", 14), show="*", width=25)
entry_password.grid(row=1, column=1, pady=10)

# CAPTCHA generation
captcha_var = tk.StringVar()  # StringVar to hold CAPTCHA text
generate_captcha()  # Generate initial CAPTCHA

# CAPTCHA Label and Entry
label_captcha = tk.Label(frame, text="CAPTCHA:", font=("Arial", 14), bg="white")
label_captcha.grid(row=2, column=0, sticky="w", pady=10)
label_generated_captcha = tk.Label(frame, textvariable=captcha_var, font=("Arial", 14),
bg="white", fg="black")
label_generated_captcha.grid(row=2, column=1, sticky="w", pady=10)

entry_captcha = tk.Entry(frame, font=("Arial", 14), width=25)
entry_captcha.grid(row=3, column=1, pady=10)

# Login Button
btn_login = tk.Button(frame, text="Login", font=("Arial", 14), width=15, bg="#4CAF50",
fg="white", command=login)
btn_login.grid(row=4, columnspan=2, pady=20)
root.mainloop()
```

Main program:

```
# Importing all necessary modules
import pymongo
from tkinter import *
import tkinter.ttk as ttk
import tkinter.messagebox as mb
import tkinter.simpledialog as sd

# Connecting to MongoDB
client = pymongo.MongoClient('mongodb://localhost:27017/')  # Change if MongoDB is
hosted elsewhere
db = client['libraryDB']
```

```
collection = db['Library']

# Functions
def issuer_card():
    Cid = sd.askstring('Issuer Card ID', 'What is the Issuer\'s Card ID?\t\t\t')

    if not Cid:
        mb.showerror('Error', 'Issuer ID cannot be empty, it must have a value')
    else:
        return Cid

def display_records():
    global collection, tree
    tree.delete(*tree.get_children())
    data = collection.find({})
    for record in data:
        tree.insert('', END, values=(record['BK_NAME'], record['BK_ID'],
record['AUTHOR_NAME'],
                                     record['BK_STATUS'], record['CARD_ID']))

def clear_fields():
    global bk_status, bk_id, bk_name, author_name, card_id

    bk_status.set('Available')
    for i in ['bk_id', 'bk_name', 'author_name', 'card_id']:
        exec(f"{i}.set('')")
        bk_id_entry.config(state='normal')
    try:
        tree.selection_remove(tree.selection()[0])
    except:
        pass

def clear_and_display():
    clear_fields()
    display_records()

def add_record():
    global collection
    global bk_name, bk_id, author_name, bk_status
    if bk_status.get() == 'Issued':
        card_id.set(issuer_card())
    else:
        card_id.set('N/A')
    surety = mb.askyesno('Confirmation required', 'Are you sure?\nNote:Book ID cannot be
changed later.')
    if surety:
        new_record = {
```

```
            'BK_NAME': bk_name.get(),
            'BK_ID': bk_id.get(),
            'AUTHOR_NAME': author_name.get(),
            'BK_STATUS': bk_status.get(),
            'CARD_ID': card_id.get()
        }
        try:
            collection.insert_one(new_record)
            clear_and_display()
        except pymongo.errors.DuplicateKeyError:
            mb.showerror('Book already in use!',
                         'The entered Book ID exists in the database, please alter the book ID')

def view_record():
    global bk_name, bk_id, bk_status, author_name, card_id
    global tree

    if not tree.focus():
        mb.showerror('Select a row!',
                     'To view a record, you must select it in the table. Please do so before
continuing.')
        return
    current_item_selected = tree.focus()
    values_in_selected_item = tree.item(current_item_selected)
    selection = values_in_selected_item['values']
    bk_name.set(selection[0]);
    bk_id.set(selection[1]);
    bk_status.set(selection[3])
    author_name.set(selection[2])
    try:
        card_id.set(selection[4])
    except:
        card_id.set('')

def update_record():
    def update():
        global bk_status, bk_name, bk_id, author_name, card_id
        global collection, tree
        if bk_status.get() == 'Issued':
            card_id.set(issuer_card())
        else:
            card_id.set('N/A')
        collection.update_one({'BK_ID': bk_id.get()}, {'$set': {
            'BK_NAME': bk_name.get(),
            'BK_STATUS': bk_status.get(),
            'AUTHOR_NAME': author_name.get(),
            'CARD_ID': card_id.get()
```

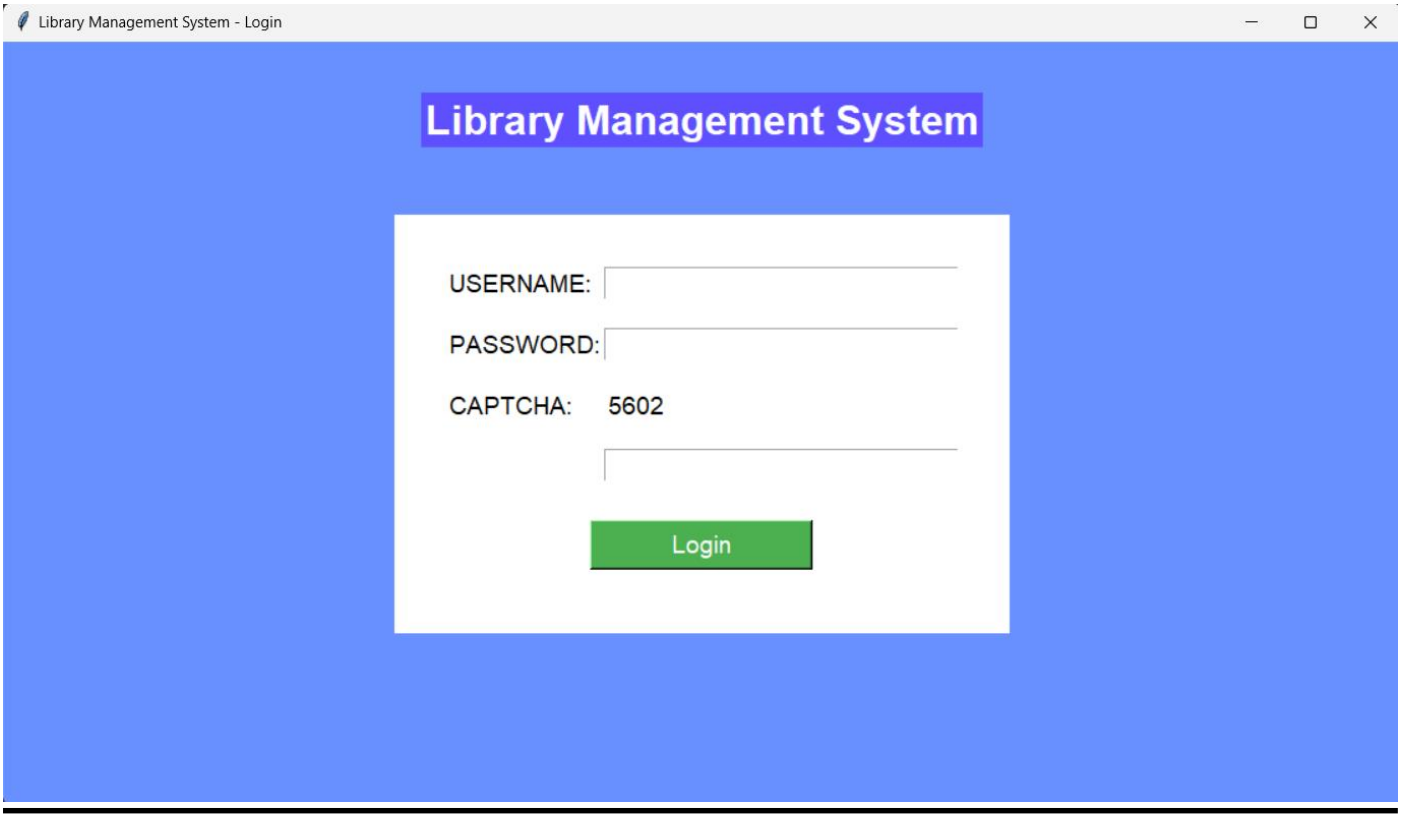
```
        }})
        clear_and_display()
        edit.destroy()
        bk_id_entry.config(state='normal')
        clear.config(state='normal')
    view_record()
    bk_id_entry.config(state='disable')
    clear.config(state='disable')
    edit = Button(left_frame, text='Update Record', font=btn_font, bg=btn_hlb_bg, width=20,
command=update)
    edit.place(x=50, y=375)

def remove_record():
    if not tree.selection():
        mb.showerror('Error!', 'Please select an item from the database')
        return
    current_item = tree.focus()
    values = tree.item(current_item)
    selection = values["values"]
    collection.delete_one({'BK_ID': selection[1]})
    tree.delete(current_item)
    mb.showinfo('Done', 'The selected record deleted successfully')
    clear_and_display()

def delete_inventory():
    if mb.askyesno('Confirmation required', 'Are you sure to delete all records?'):
        tree.delete(*tree.get_children())
        collection.delete_many({})
    else:
        return

def change_availability():
    global card_id, tree, collection

    if not tree.selection():
        mb.showerror('Error!', 'Please select a book from the database')
        return
    current_item = tree.focus()
    values = tree.item(current_item)
    BK_id = values['values'][1]
    BK_status = values["values"][3]
    if BK_status == 'Issued':
        surety = mb.askyesno('Confirmation', 'Book returned?')
        if surety:
            collection.update_one({'BK_ID': BK_id}, {'$set': {'BK_STATUS': 'Available',
'CARD_ID': 'N/A'}})
        else:
```

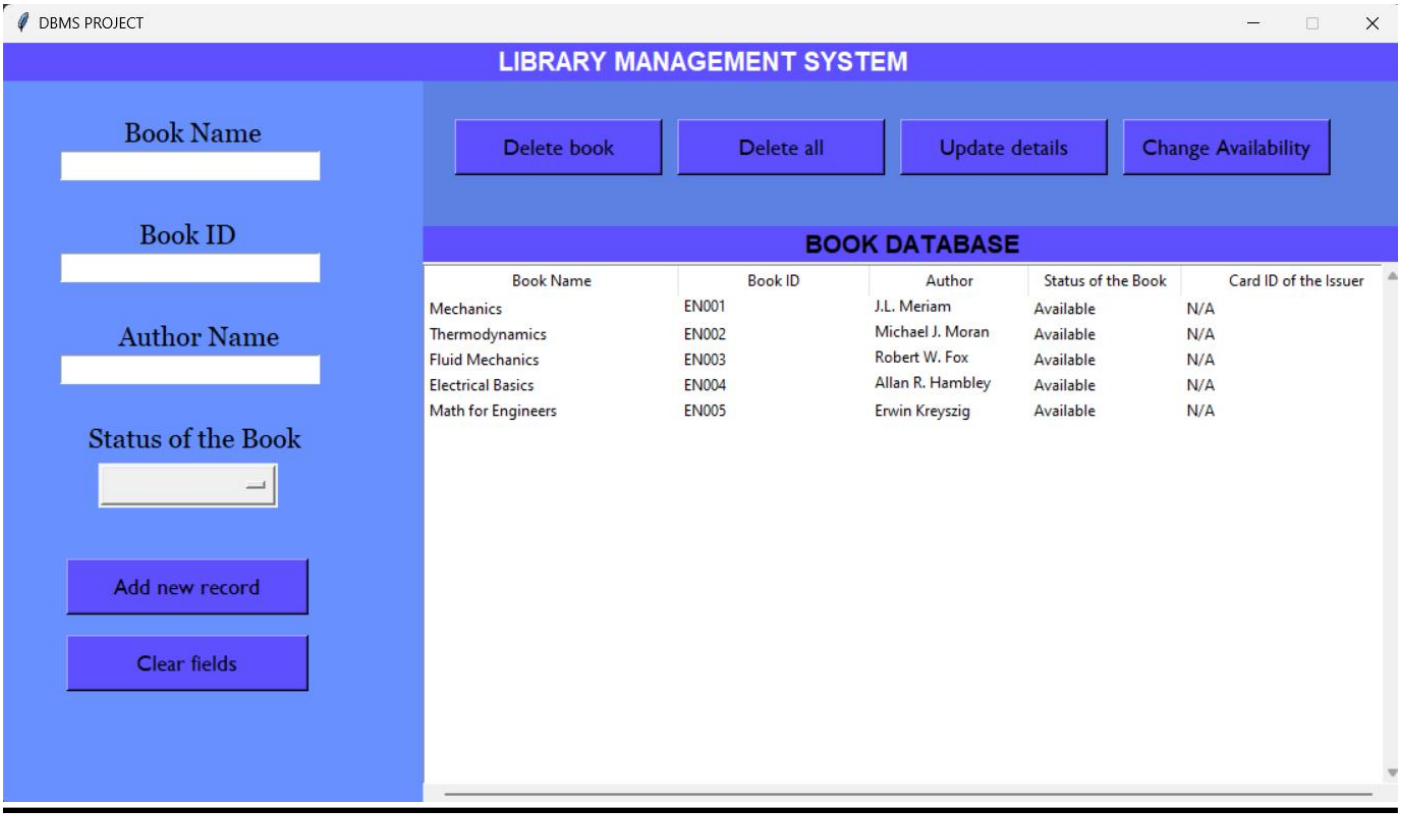
```
            mb.showinfo('Alert!!', 'The book needs to be returned to change status')
    else:
        collection.update_one({'BK_ID': BK_id}, {'$set': {'BK_STATUS': 'Issued', 'CARD_ID':
issuer_card()}})

    clear_and_display()

# Variables for the GUI
lf_bg = '#6891FF'  # Left Frame Background Color
rtf_bg = '#5D81E3'  # Right Top Frame Background Color
rbf_bg = '#5E4FFF'  # Right Bottom Frame Background Color
btn_hlb_bg = '#5E4FFF'  # Background color for Head Labels and Buttons

lbl_font = ('Georgia', 16)  # Font for all labels
entry_font = ('Times New Roman', 12)  # Font for all Entry widgets
btn_font = ('Gill Sans MT', 13)

# Initializing the main GUI window
root = Tk()
root.title('DBMS PROJECT')
root.geometry('1100x600')
root.resizable(0, 0)
Label(root, text='LIBRARY MANAGEMENT SYSTEM', font=("Noto Sans CJK TC", 15,
'bold'), bg=btn_hlb_bg, fg='White').pack(
    side=TOP, fill=X)

# StringVars
bk_status = StringVar()
bk_name = StringVar()
bk_id = StringVar()
author_name = StringVar()
card_id = StringVar()

# Frames
left_frame = Frame(root, bg=lf_bg)
left_frame.place(x=0, y=30, relwidth=0.3, relheight=0.96)

RT_frame = Frame(root, bg=rtf_bg)
RT_frame.place(relx=0.3, y=30, relheight=0.2, relwidth=0.7)

RB_frame = Frame(root)
RB_frame.place(relx=0.3, rely=0.24, relheight=0.785, relwidth=0.7)

# Left Frame
Label(left_frame, text='Book Name', bg=lf_bg, font=lbl_font).place(x=92, y=25)
Entry(left_frame, width=25, font=entry_font, text=bk_name).place(x=45, y=55)
```

```
Label(left_frame, text='Book ID', bg=lf_bg, font=lbl_font).place(x=104, y=105)
bk_id_entry = Entry(left_frame, width=25, font=entry_font, text=bk_id)
bk_id_entry.place(x=45, y=135)

Label(left_frame, text='Author Name', bg=lf_bg, font=lbl_font).place(x=88, y=185)
Entry(left_frame, width=25, font=entry_font, text=author_name).place(x=45, y=215)

Label(left_frame, text='Status of the Book', bg=lf_bg, font=lbl_font).place(x=64, y=265)
dd = OptionMenu(left_frame, bk_status, *['Available', 'Issued'])
dd.configure(font=entry_font, width=12)
dd.place(x=75, y=300)

submit = Button(left_frame, text='Add new record', font=btn_font, bg=btn_hlb_bg,
width=20, command=add_record)
submit.place(x=50, y=375)

clear = Button(left_frame, text='Clear fields', font=btn_font, bg=btn_hlb_bg, width=20,
command=clear_fields)
clear.place(x=50, y=435)

# Right Top Frame
Button(RT_frame, text='Delete book', font=btn_font, bg=btn_hlb_bg, width=17,
command=remove_record).place(x=25, y=30)
Button(RT_frame, text='Delete all', font=btn_font, bg=btn_hlb_bg, width=17,
command=delete_inventory).place(x=200, y=30)
Button(RT_frame, text='Update details', font=btn_font, bg=btn_hlb_bg, width=17,
command=update_record).place(x=375,y=30)
Button(RT_frame, text="Change Availability", font=btn_font, bg=btn_hlb_bg, width=17,
command=change_availability).place(x=550,y=30)
# Right Bottom Frame (Treeview)
Label(RB_frame, text='BOOK DATABASE', font=("Noto Sans CJK TC", 14, 'bold'),
bg=rbf_bg, fg='Black').pack(side=TOP,fill=X)
tree = ttk.Treeview(RB_frame, height=100, selectmode=BROWSE,columns=('Book Name',
"Book ID", "Author", "Status", "Issuer's Card ID"))

X_scroller = Scrollbar(tree, orient=HORIZONTAL, command=tree.xview)
Y_scroller = Scrollbar(tree, orient=VERTICAL, command=tree.yview)
X_scroller.pack(side=BOTTOM, fill=X)
Y_scroller.pack(side=RIGHT, fill=Y)
tree.config(yscrollcommand=Y_scroller.set, xscrollcommand=X_scroller.set)

tree.heading('Book Name', text='Book Name', anchor=CENTER)
tree.heading('Book ID', text='Book ID', anchor=CENTER)
tree.heading('Author', text='Author', anchor=CENTER)
tree.heading('Status', text='Status of the Book', anchor=CENTER)
tree.heading("Issuer's Card ID", text="Card ID of the Issuer", anchor=CENTER)
```

```
tree.column('#0', width=0, stretch=NO)
tree.column('#1', width=200, stretch=NO)
tree.column('#2', width=150, stretch=NO)
tree.column('#3', width=125, stretch=NO)
tree.column('#4', width=120, stretch=NO)
tree.column('#5', width=180, stretch=NO)

tree.place(y=30, relwidth=1, relheight=0.9, relx=0)
display_records()

root.update()
root.mainloop()
```

RESULTS SCREENSHOT:





CONCLUSION:

• In conclusion, the Library Management System (LMS) utilizing MongoDB as its database solution offers an efficient and flexible approach to managing library operations, such as book tracking, user management, and issue/return processes.

• The use of MongoDB enables scalability, allowing the system to handle large volumes of data while maintaining fast query performance, which is essential for managing vast collections of books and user records.

• The system's real-time data access empowers administrators to manage books, users, and transactions seamlessly, while the integration with MongoDB ensures data integrity and easy updates.

• With robust security features, the system ensures the safety and privacy of user information, and its adaptability supports future growth in terms of data volume and functionality.

• Overall, the Library Management System with MongoDB represents a modern, scalable, and efficient solution that optimizes library operations, enhances user experience, and simplifies management tasks.

REFERENCES :

- **MongoDB Documentation:** https://docs.mongodb.com/
- **Python MongoDB Driver:** https://pymongo.readthedocs.io/
- **"Learning MongoDB"** by Jason C. Brown
- **Tkinter Documentation:**

  https://docs.python.org/3/library/tkinter.html
- **Python Tkinter Tutorial (GeeksforGeeks):**

  https://www.geeksforgeeks.org/python-tkinter-tutorial/