

# LIBRARY MANAGEMENT SYSTEM

---

using Python and SQLite Database

### Team Members Details

SL. No	Name	College ID	University Roll
1.	Subrata Mukherjee	CSE/19-20/046	16500119013
2.	Sagnik Das	CSE/19-20/036	16500119020
3.	Yuvraj Basu	CSE/19-20/058	16500119051
4.	Kushagradhy Saha	CSE/19-20/022	16500119030
5.	Raunak Das	Lateral	16500120058
6.	Sagar Maiti	CSE/19-20/035	16500119016
7.	Chitram Dasgupta	CSE/19-20/014	16500119015



## **INTRODUCTION**

A library management system, also known as an automated library system is software that has been developed to handle basic housekeeping functions of a library.

→ It's a well organized software solution for a library.

→ It help to provide information on any book present in library to the user as well as staff member.

→ It keeps a track of book issued, returned and added to library

### **AIM**

The project's main aim is to build a management system more like a website which

- Can create a computerized management system for a library
- Has the capability to issue books
- Has the capability to return books
- Has an administrator account
- Can manage users through administrator account
- Has customized profile with photo of the user
- Tracks the books that users have issued
- Keeps the databases correct and up-to-date
- Can store all the book and user data in a proper manner

With the blooming of technology I thought it would be of a good sense to automate some of our oldest systems. So I asked myself which system is

- Not up-to-date
- Requires management
- Has its components value very high in the education department

That's when I came up with a plan to create a website 'College Library Management'

- Advanced Python
- Using PyCharm Ide
- Using SQL server management

## **USER CHARACTERISTICS:**

The users of the system are members and the administrator who maintains the system. The member is assumed to have basic knowledge of the computers and Internet browsing. The administrator of the system has more knowledge of the internals of the system and is able to rectify the small problems that may arise due to disk crashes, power failures and other catastrophes to maintain the system. The proper user interface, user's manual, College help and the guide to install and maintain the system must be sufficient to educate the users on how to use the system without any problems.



## Constraints

- The information of all the users must be stored in a database that is accessible by the College Library System. .
- The users must have their correct usernames and passwords to enter into the College Library System.

## Assumptions and dependencies

- The users have sufficient knowledge of computers.
- The users know the English language, as the user interface will be provided in English
- The product can access the student database

## Document Conventions

The following are the list of conventions and acronyms used in this document and the project as well:

**Admin:** A login id representing a user with user administration privileges to the software

**User:** A general login id assigned to most users

**Client:** Intended users for the software

**SQL:** Structured Query Language; used to retrieve information from a database

**SQL Server:** A server used to store data in an organized format

**ASP:** Active Server Pages: A Web Page formatted on the server and delivered to the browser.

**Layer:** Represents a section of the project

**User Interface Layer:** The section of the assignment referring to what the user interacts with directly.

**Application Logic Layer:** The section of the assignment referring to the Web Server. This is where all computations are completed.

**Data Storage Layer:** The section of the assignment referring to where all data is recorded.

**Data flow diagram:** It shows the dataflow between the entities.

**Use Case:** A broad level diagram of the project showing a basic overview



**Boolean:** A true/false notation

**Interface:** Something used to communicate across different mediums

**Unique Key:** Used to differentiate entries in a database

In feasibility study phase various steps have been taken:

1. Identify information at different levels.
2. Identify the expectation of user from an automated system.
3. Analyze the importance of automated system

**Feasibility study:**

In order to make sure that the project is feasible, following feasibility studies have been conducted: -

**Technical Feasibility study:**

It is possible to develop the system using simple platform. All the functions of a project for communication can be implemented in the new system. Hence the system is technically feasible.

### **Economic Feasibility study:**

College library management system is a worth making project. This project is economically feasible in every sense that it takes less effort, less time, and nominal cost of purchasing the tools for developing the software.

### **Legal feasibility study:**

Since the project needs no copyright, patenting, and doesn't intent to have any relation with anybody else's intellectual property rights, it can be considered as a legally feasible project.

### **Time feasibility study:**

As it has been more probable (as per the requirements, functions, and performance specifications of the system) that the project can be completed within the given time frame, it is considered that the undertaking this project is feasible in the context of time

### **Operational feasibility study:**

This system is completely operational and can be successfully implemented. College library management system is easy to understand not only for any sophisticated users but for the naïve users as well .It provides simple ambience in which user can feel free to work faster, easier, and more accurately. Therefore, it can be socially and behaviorally accepted and is feasible too.



## **Project planning:**

During Planning all the activities that are to be performed to create the system are planned. Following are some of the issues that are well devised so that proper monitoring and controlling of the project could be easily done: - **Project development:**

To avoid being stuck in dilemma during the development of project, one need to be sure that the process model he's using is right for the project. Since all the requirements about the problem can't be easily understood and may not be stable during the development of the system.

## **Quality Planning:**

To ensure that the final software for-"College library management system" is of high quality, some quality control activities should be decided /planned in advance to perform them throughout the development of the software.

Following is a list of quality control activities that are used to identify and remove defects from the software, hence making it a high quality controlled system: -

Requirements Review

Design Reviews

Code Reviews

Testing

### **Risk control planning:**

A risk is a probabilistic event – it may or may not occur. The aim of risk control planning is to minimize the impact of risks (if they occur) in the project.

Following are some known risks that might occur and their mitigation plan: -

Unclear project requirements ~>Keep in touch with the faculty in charge.

Data loss ~>Use CD's and/or pen drives to have some extra backups of the data.

Project delays ~>Use proper scheduling of the project as soon as possible so that the project could be completed.

### **Software size planning:**

It has been taken into account that there are some functions in project that are indispensable .And these should not be excluded from project .Such functions /modules are like login, Sign up, admin add, admin remove, add user, remove user. So there are at least a minimum number of modules that have to be there in project.

### **Effort / cost Estimation:**

#### **Project scheduling:**

During early stages of project planning, a kind of macroscopic schedule was already planned which gave a rough idea about activities that should be carried out for developing the project .In project scheduling , those sets of activities are refined into a detailed schedule.



## **OVERALL DESCRIPTION**

### **Product Perspective:**

College library management system is a product which does not intent to have any relation with any other product. It is a complete system in itself. It is an exclusive product which is to be concerned with the optimization.

**Product Features:** The project mainly use the concepts of .Net, simple tools of programming and SQL server for database storage

### **User Classes and Characteristics**

There are number of functions that the system/product is supposed to perform which is as follows: -

- Issue books
- Remove books
- Admin login
- User login
- Add books
- Welcome page
- New user

The user of this product need not be computer expert. Even a naïve user can also operate the system. The user interfaces are to be made so simple that anybody can be comfortable in working with the system in just a few minutes. The basic things which are required in a user are:

- User should know what the computer is.
- User should understand English.
- User must know which key (button or keyboard) does what.
- User must have an experience of library management.

### **Operating Environment:**

The product will be operating in windows environment. Also it will be compatible with the higher version of this explorer.

### **Design and Implementation Constraints:**

The general constraints which are to be introduced in project are as follows: -

- ❖ Only that person can operate the system's who knows the ID and Password of the valid user of the system.
- ❖ In case admin does not remembers his/her ID/Password then system won't consider him/her as a valid user for College library management system.



## **Functional Requirement:**

**1. Accession number, roll number and teacher identification must all be unique as they form the primary keys of the respective tables. → 2. All new books must be entered in the accession table first, to avoid problems later. → 3. A book must not be deleted from student profile unless and until she pays the appropriate fine or the same book. → 4. While inserting values in the database, only valid values must be entered**

## **Data Requirement:**

**1. The Library Management System shall be required to maintain information about its users and books. → 2. It shall store databases for students, teachers and books. → 3. The student database stores information about a student's roll no, name, address, course and year. → 4. The book database stores information about a book title, author, publisher, cost, bill number, year of publishing and pages. → 5. The teacher database stores information about a teacher's id, name, department, and designation, address and telephone number. Methodology of the Stu**

## **PERFORMANCE REQUIREMENT**

### **Static Requirements:**

Project College library management system is to support many users at a time. All necessary operations shall have been carried out with the help of many clients and a server.

### **Dynamic Requirements:**

Project College management system has to avoid degradation of its performance by processing one or at most two forms at a time. If the user finishes his work related to one form and opens the other, then the previous form will be unloaded in order to save some memory space.

### **Logical Database Requirements:**

All the information should be stored in separate databases. These databases should be categorized and maintained in a logical manner. For example:-

- User information in User database.
- Administrator information in Admin database.

### **Design Constraints**

SQLite and PyCharm will have to be operated on Windows XP, VISTA, or 7. It will require at least 512MB of RAM, and few Megabytes of Hard Disk. As the security feature included in the system, no one except the administrator and member can access the system.

### **SOFTWARE SYSTEM ATTRIBUTES:**

Following are some of the basic attributes that the project should have:-

**Reliability:** - The system should be reliable in the sense that there should be no room for mistakes. Every activity/ function of project should be indefectible.



**Availability:** - The system is to be available as and when needed.


**Security:** - The system is to be secure in the sense that nobody except the authenticated users can login and use the system.

**Portability:** - The project should be made such that there are as less operating system dependencies as possible, or the software should be portable with only few modifications.

**Maintainability:** - The system should be maintainable in the sense that if any error occurs, it should be easily rectified and the cost incurred in maintenance should be as low as possible.

### **Organizing Specific Requirements**

There are some requirements of the system that are indispensable while the others are lesser. Following is the organization of specific requirements in descending order of their importance: -

- 
- i. Functional requirements.
  - ii. External Interface requirements.
  - iii. Performance requirements.
  - iv. Design Constraints.
  - v. Logical Database requirements.
  - vi. Software system attributes.

## **Description**

The user interface must be customizable by the administrator

## **Criticality**

This issue is essential to the overall system. All the modules provided with the software must fit into this graphical user interface and accomplish to the standard defined.

## **Technical issues**

In order to satisfy this requirement the design should be simple and all the different interfaces should follow a standard template. There will be the possibility of changing colors and images, plus switching between interfaces with the minimum impact for the users.

## **Risks**

To reduce the circumstances under which this requirement might not be satisfied, all the designers must have been developed web sites previously and they must be aware of html restriction and cross browsers implementations before starting the designing. In order to reduce the probability of this occurrence the entire design team will be trained in basic html development and macromedia fireworks, this tool will be used instead of Photoshop.

## **Dependencies with other requirements**

All user interfaces should be able to interact with the user management module and a part of the interface must be dedicated to the login/logout module.



## METHODOLOGY OF STUDY DATA SOURCE:

1. Employees of the SMU Library : Since the librarian officers are the main beneficiaries of this Project they are one of our data source. Internet: → 2. Students : this are another beneficiaries of the project because they are main users of the system and they know all needed requirements and they well know the problems behind this. → 3. We used internet for getting tools we will be using to develop this system

## SERVICES PROVIDED TO USER:

The Library Management System automates the basic library functions to aid in the day-to-day operations of a library. It supports functions such as issue, return, the very basic functions of searching for a particular book, etc. It also maintains data about books about books, teachers, students records that are required during various library operations. The software aims to make the system user friendly and efficient.

## TASK OF LIBRARIAN:

→Add New Book →Add New Member →Issue Book →Return Book →List of Books →List of Members →Update/Delete Member →Update/Delete Book →Search For a Book →Check In Book →Check Out Book

## **COST & BENEFITS ANALYSIS**

### **COST :**

Various costs that are incurred in making the system are:-

#### **1. Hardware Cost**

- a) Computer purchase
- b) SQL server software purchase & installation.
- c) .NET software purchase & installation.

#### **2. Facility Cost**

- a) Proper lightning.
- b) Air Conditioning

#### **3. Operating Cost**

- a) Consumption of electricity

#### **4. Supply Cost**

- a) Compact Discs/Pen drives.



## **BENEFITS:**

Various benefits that are to be obtained with project are: -

### **1. Improved performance**

- a) Accuracy improvement.
- b) Time Saving.

### **2. Decreased Supply cost**

- a) No use of registers.

### **3. Decreased personnel cost**

- a) Fewer staff.
- b) Fewer payments.

## **CONCLUSION OF THE COST AND BENEFIT ANALYSIS:**

Most of the costs that have been incurred in developing the system are onetime costs. While all the benefits that we would get are not for one time only, they shall be obtained on a regular basis. In this way, benefits will exceed costs by a substantial margin, hence this project can be considered as cost effective.

## **PROCESS INVOLVED**

The various processes involved in making the software successfully running and documenting the project are as follows: -

### **❑ Preliminary Investigation**

- **Making enquiries about the current system's working.**
- **Conducting feasibility studies for the project to be undertaken.**
- **Deciding whether to undertake the project or not.**

### **❑ System Requirement**

**Gathering the information about the need to build the project**

### **❑ System Design**

**Designing Data Flow Diagrams (DFD's), data dictionary, databases schemas, user interfaces.**

### **❑ System Development**

**Developing Databases, Creating Forms, Coding of software, and integrating the components.**



## ☐ System Testing

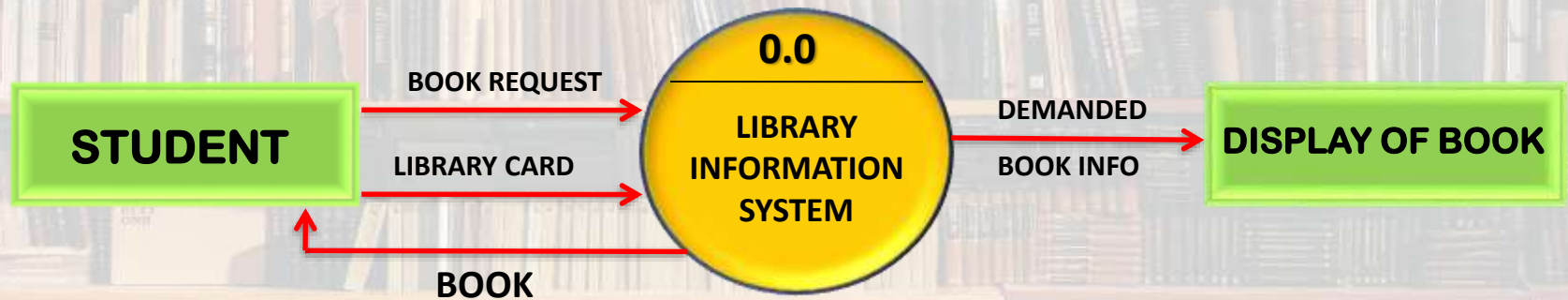
- ✓ Testing Individual Units of the software as and when they are created.
- ✓ Testing the integration of units of the software.
- ✓ Testing the whole functioning of the software.
- ✓ Testing the software at the developer's end.
- ✓ Testing informally the software with general people.
- ✓ Testing the software at the clients end

## ☐ Implementation

- ✓ Installing the software at the user's side. Direct conversion approach has been taken.
- ✓ Getting feedback from the user.

# DATA FLOW DIAGRAM

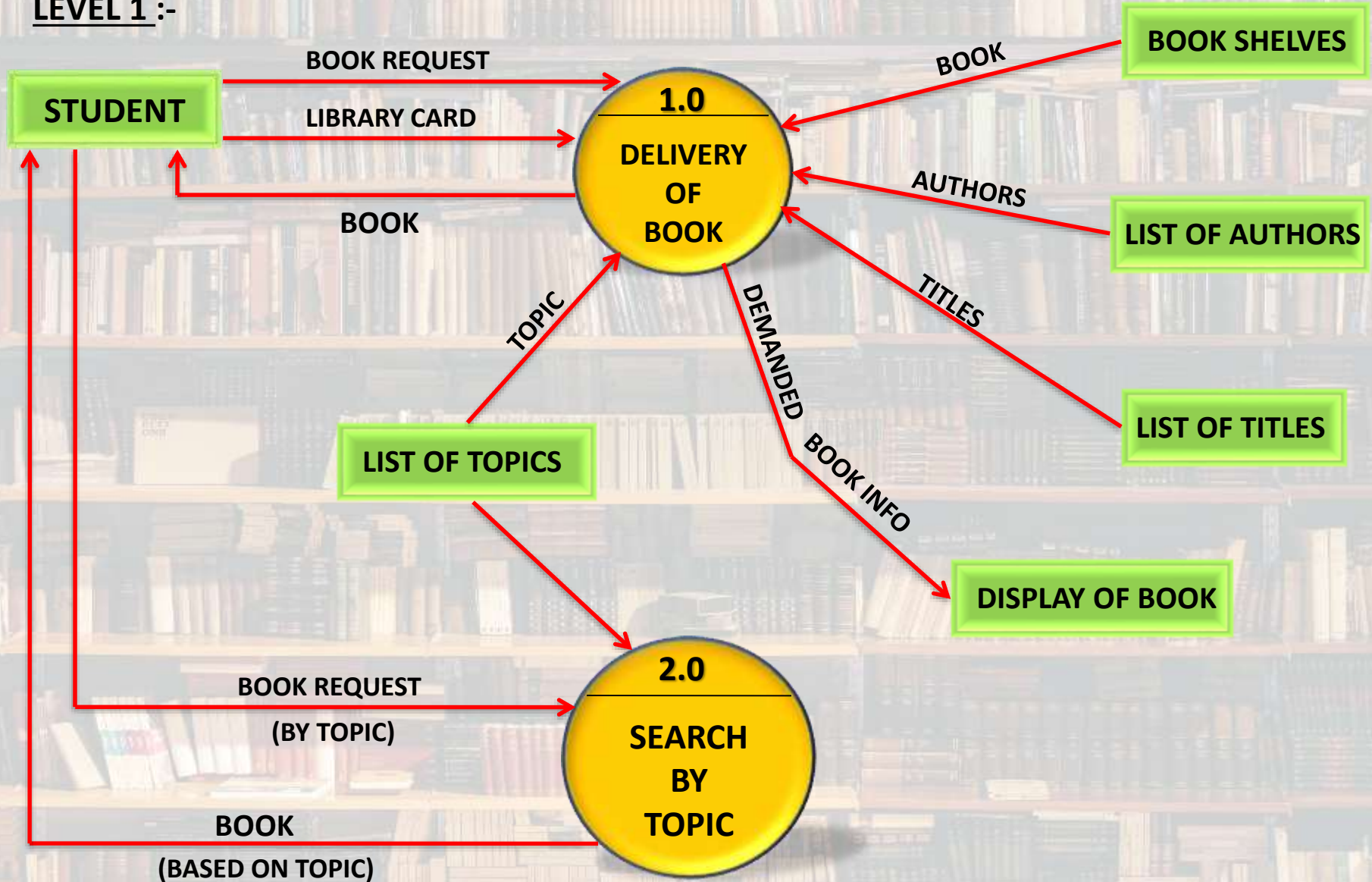
Level 0 :-





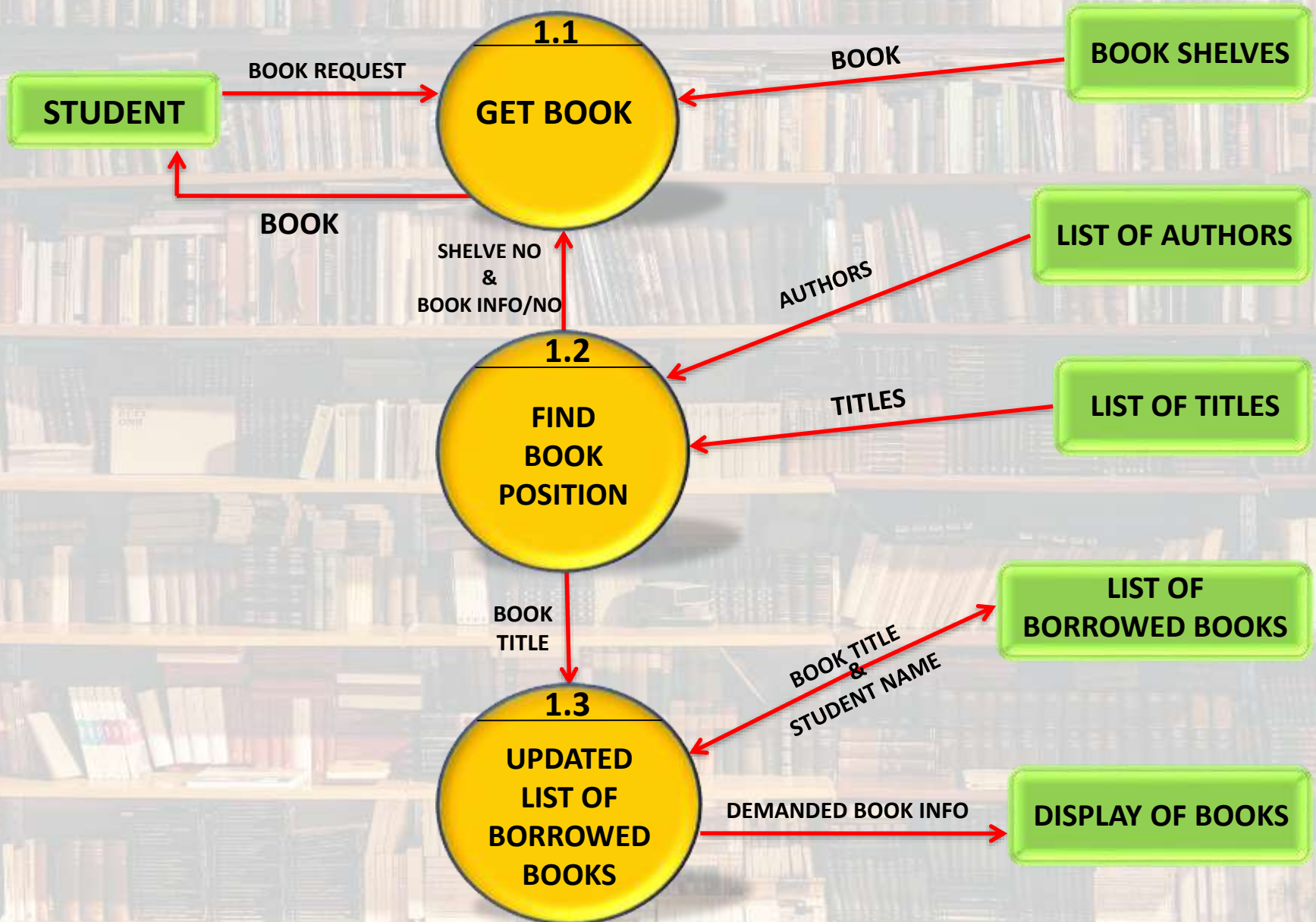
# DATA FLOW DIAGRAM

LEVEL 1 :-



# DATA FLOW DIAGRAM

LEVEL 2 :-





# CODING

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox
from PIL import Image, ImageTk
import random
import sqlite3

image1 = 'bb.png'
image2 = 'images17.png'
image3 = 'LMS1.png'
image4 = 'books.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='black', bg='yellow',  
width=19, padx=10, borderwidth=0, command=self.book).place(x=80, y=500)  
  
l2 = Button(self.a, text='STUDENT DATA', font='Papyrus 22 bold', fg='black', bg='yellow',  
width=18, padx=10, borderwidth=0, command=self.student).place(x=600, 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.ANTIALIAS)
photo2 = ImageTk.PhotoImage(photo1)

#photo2 = ImageTk.PhotoImage(Image.open(images).resize((w,
h)),Image.ANTIALIAS)
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(image4)

l1 = Button(self.a, text='Add Books', font='Papyrus 22 bold', fg='black', bg='yellow',
width=15, padx=10, command=self.addbook).place(x=12, y=100)
l2 = Button(self.a, text='Search Books', font='Papyrus 22 bold', fg='black', bg='yellow',
width=15, padx=10, command=self.search).place(x=12, y=200)
```

```
l3 = Button(self.a, text='All Books', font='Papyrus 22 bold', fg='black', bg='yellow',  
width=15, padx=10, command=self.all).place(x=12, y=300)
```

```
l4 = Button(self.a, text='<- Main Menu', font='Papyrus 22 bold', fg='black', bg='yellow',  
width=15, padx=10, command=self.mainmenu).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.aname).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.aauthor).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.agenre).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.acopies).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.aloc).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).place(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=self.sid).place(x=260, y=40)
    b1 = Button(self.f1, text='Search', bg='orange', font='Papyrus 10 bold', width=9, bd=2,
command=self.serch1).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 serch1(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('<<ReviewSelect>>')
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 successfully.")
    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.copiesdelete).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 successfully.")
```

```
self.serch1()
```

```
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 successfully")
            self.serch1()

        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='black', bg='yellow',
width=15, padx=10, command=self.issue).place(x=12, y=100)
l2 = Button(self.a, text='Return Book', font='Papyrus 22 bold', fg='black', bg='yellow',
width=15, padx=10, command=self.returnn).place(x=12, y=200)
```



```
l3 = Button(self.a, text='Student Activity', font='Papyrus 22 bold', fg='black', bg='yellow',  
width=15, padx=10, command=self.activity).place(x=12, y=300)
```

```
l4 = Button(self.a, text='<< Main Menu', font='Papyrus 22 bold', fg='black', bg='yellow',  
width=15, padx=10, command=self.mainmenu).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).place(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.astudentt).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.issuedbook).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 successfully.")
                    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).place(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.astudentt).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.returnbook).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 successfully.")
```

```
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).place(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.astudentt).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.ANTIALIAS)  
    photo2 = ImageTk.PhotoImage(photo1)
```

```
#photo2 = ImageTk.PhotoImage(Image.open(images).resize((w, h)),Image.ANTIALIAS)  
    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("python2.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('shubham', '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='white',
fg='black')
lbl_title.place(x=500, y=80)
lbl_username = Label(canvas, text="Username:", font=('Papyrus', 15, 'bold'), bd=4,
bg='orange', fg='black')
lbl_username.place(x=500, y=230)
lbl_password = Label(canvas, text="Password :", font=('Papyrus', 15, 'bold'), bd=3,
bg='orange', fg='black')
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=10,  
command=Login, bg='#abc123', fg='black')
```

```
btn_login.place(x=580, y=420)
```

```
btn_login.bind('<Return>', Login)
```

```
root.mainloop()
```

## WORKING AND IMPLEMENTATION

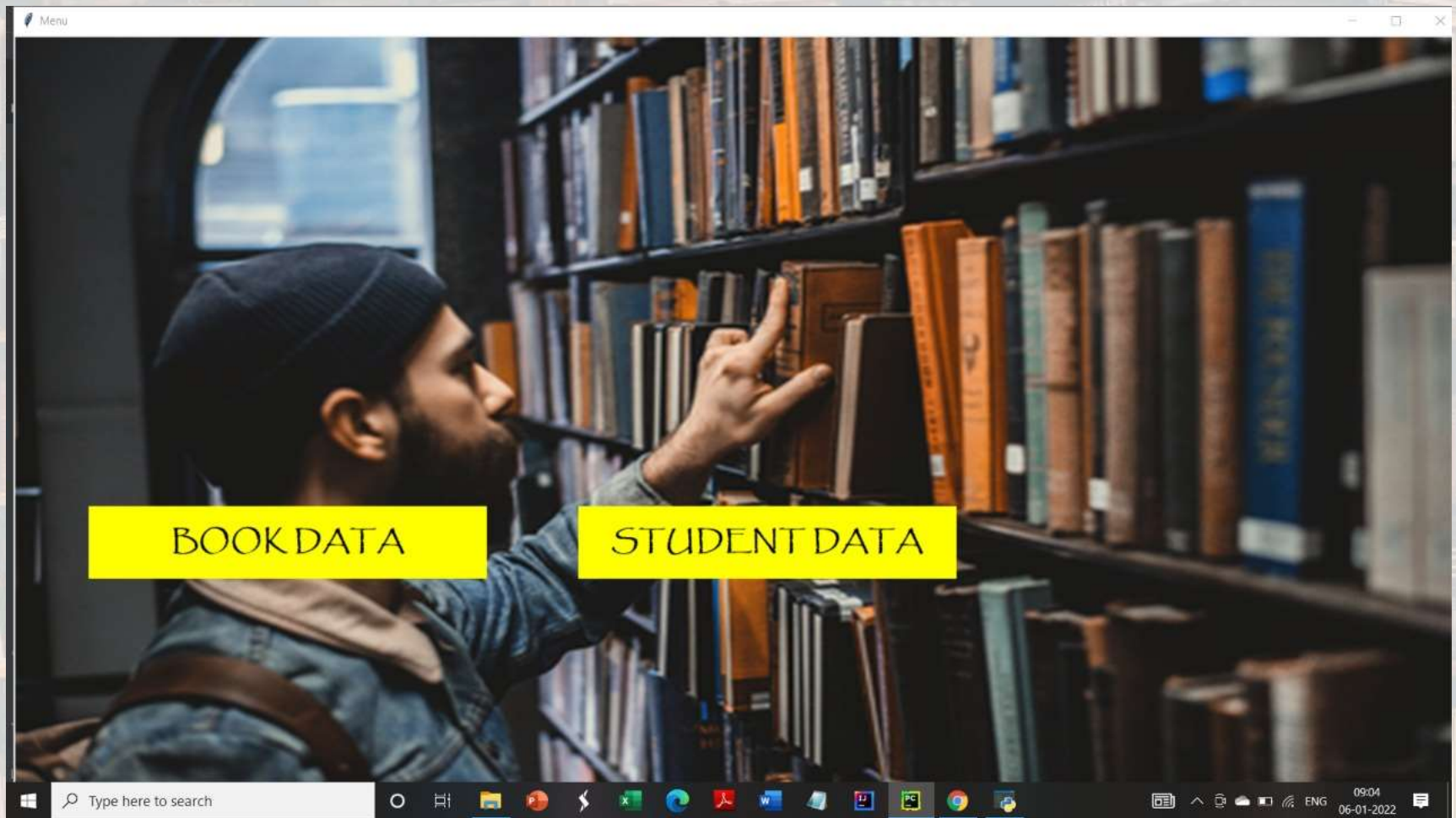
### Members login window:-



mem_id	username	password
1	Subrata	27082000
2	Sagnik	02102001
3	Yuvraj	27022001
4	Chitram	28062001
5	Raunak	29121999
6	Kush	29122000
7	Sagar	26092000

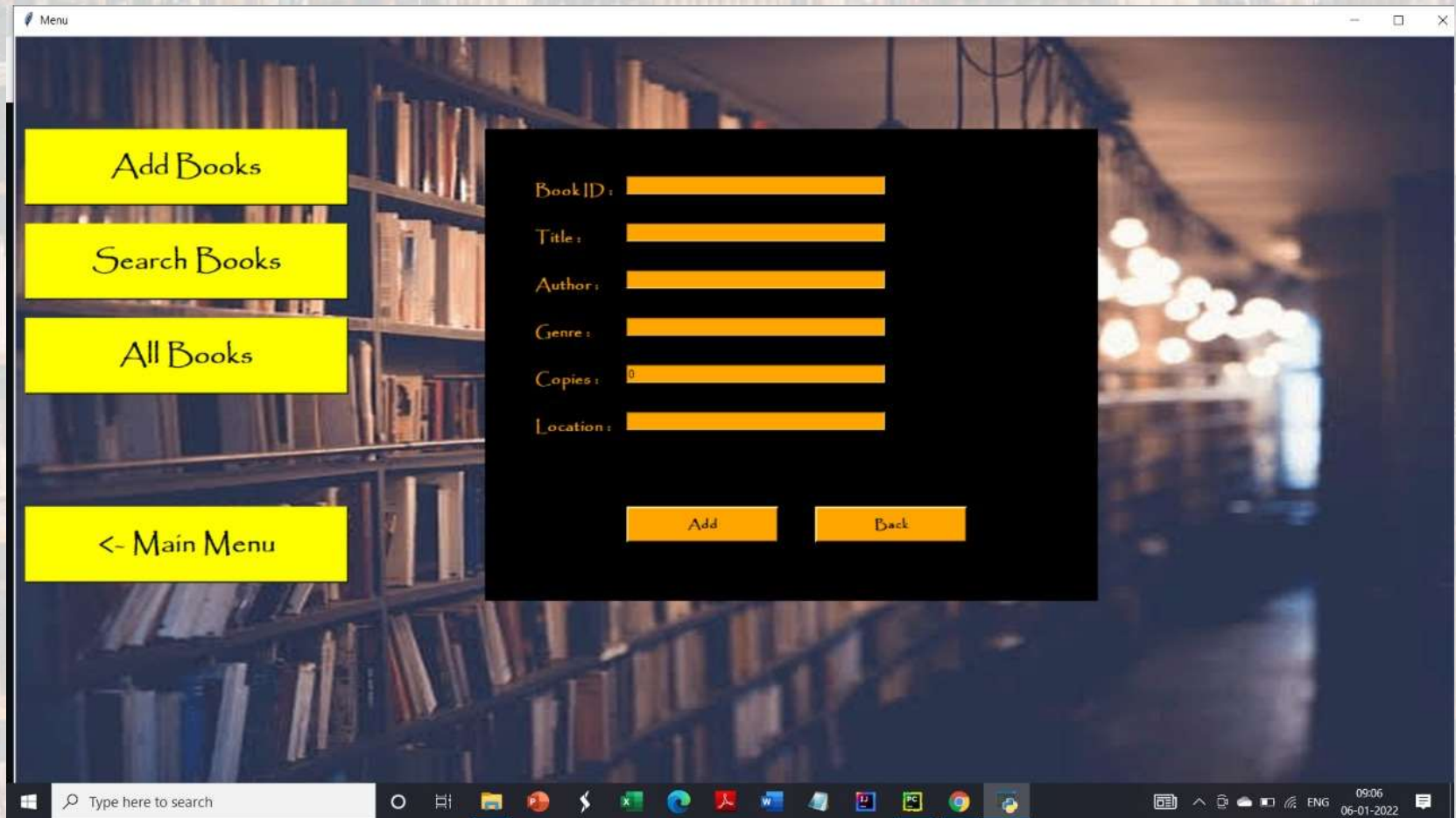
only these members with their respective username and password can enter otherwise invalid no entry/no login





**After successful login this page will show.**

## Book Data:-



Menu

Add Books

Search Books

All Books

<- Main Menu

BookID :

Title :

Author :

Genre :

Copies : 0

Location :

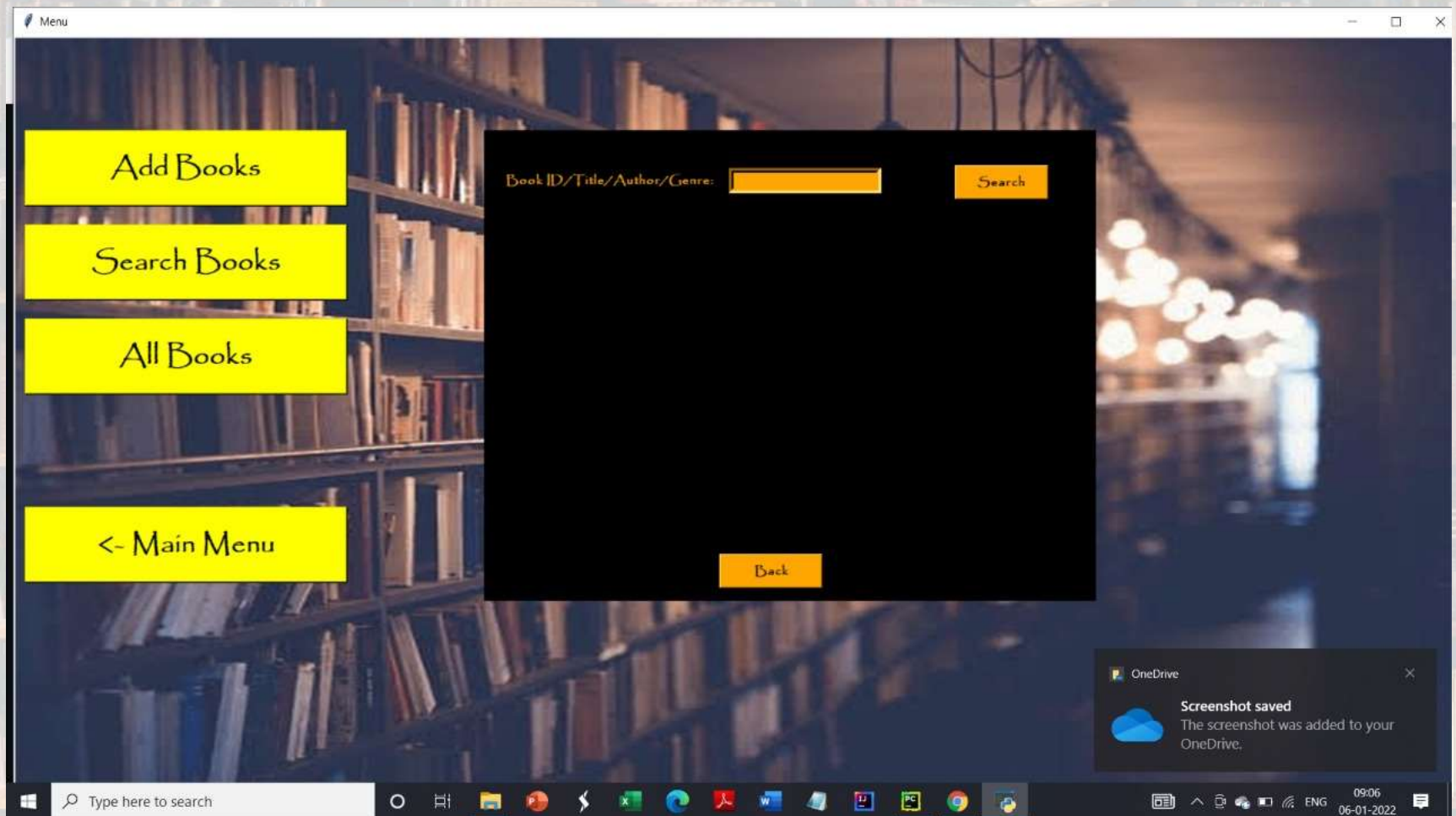
Add Back

Type here to search

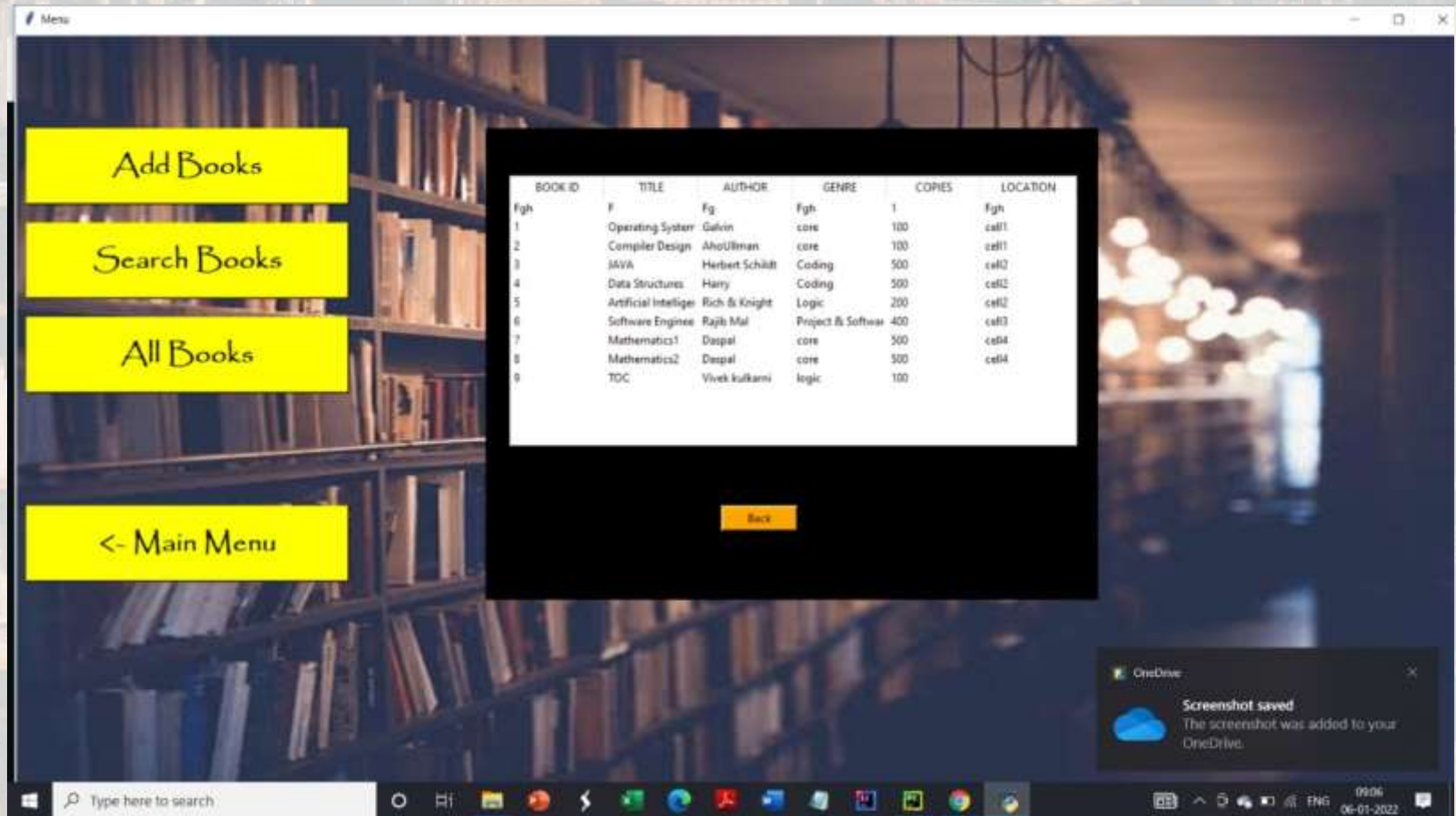
09:06 06-01-2022

## Add Books





Search Books



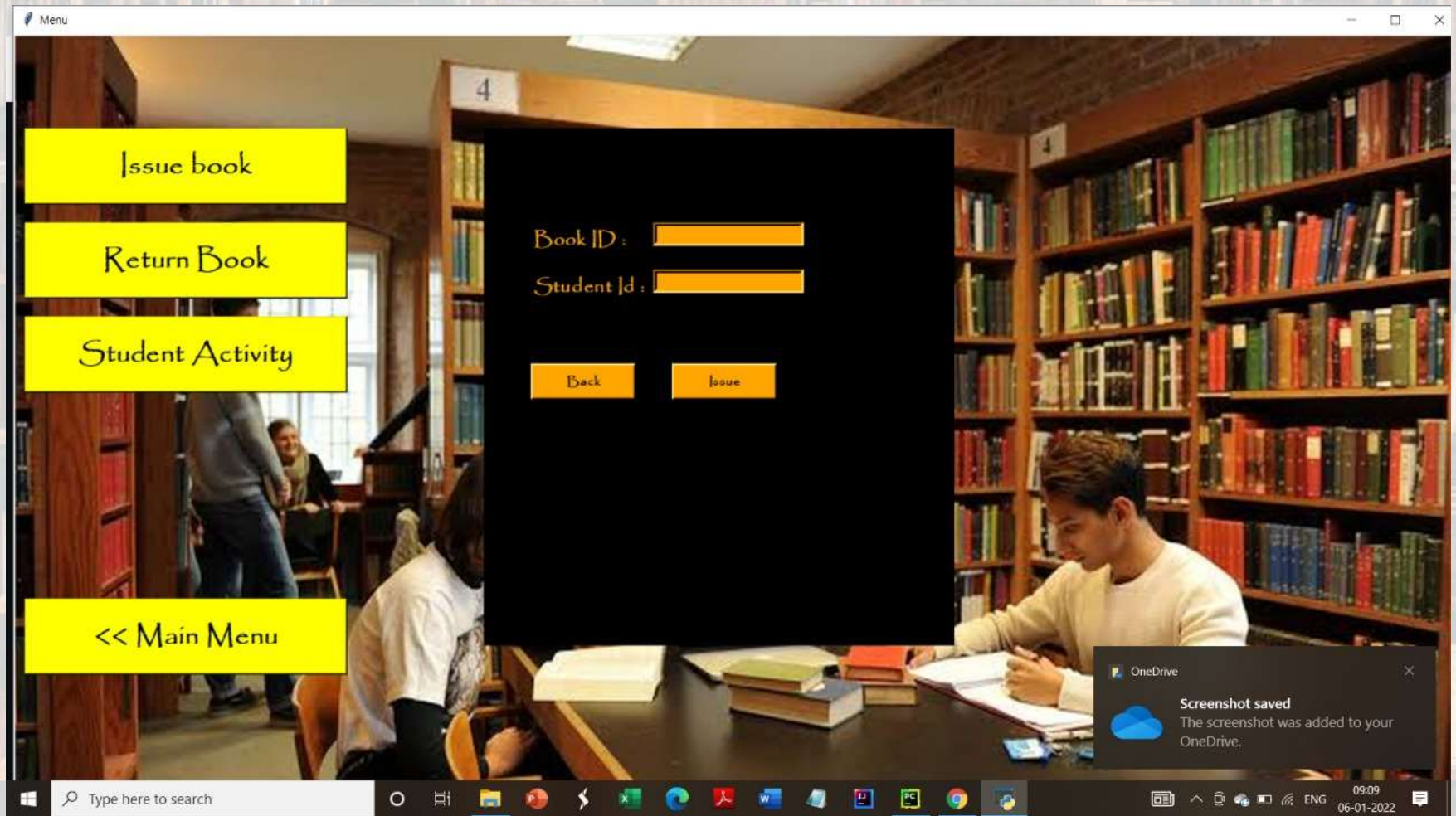
**All Books which members can access of limited number of copies.**



## STUDENT ACTIVITY:-

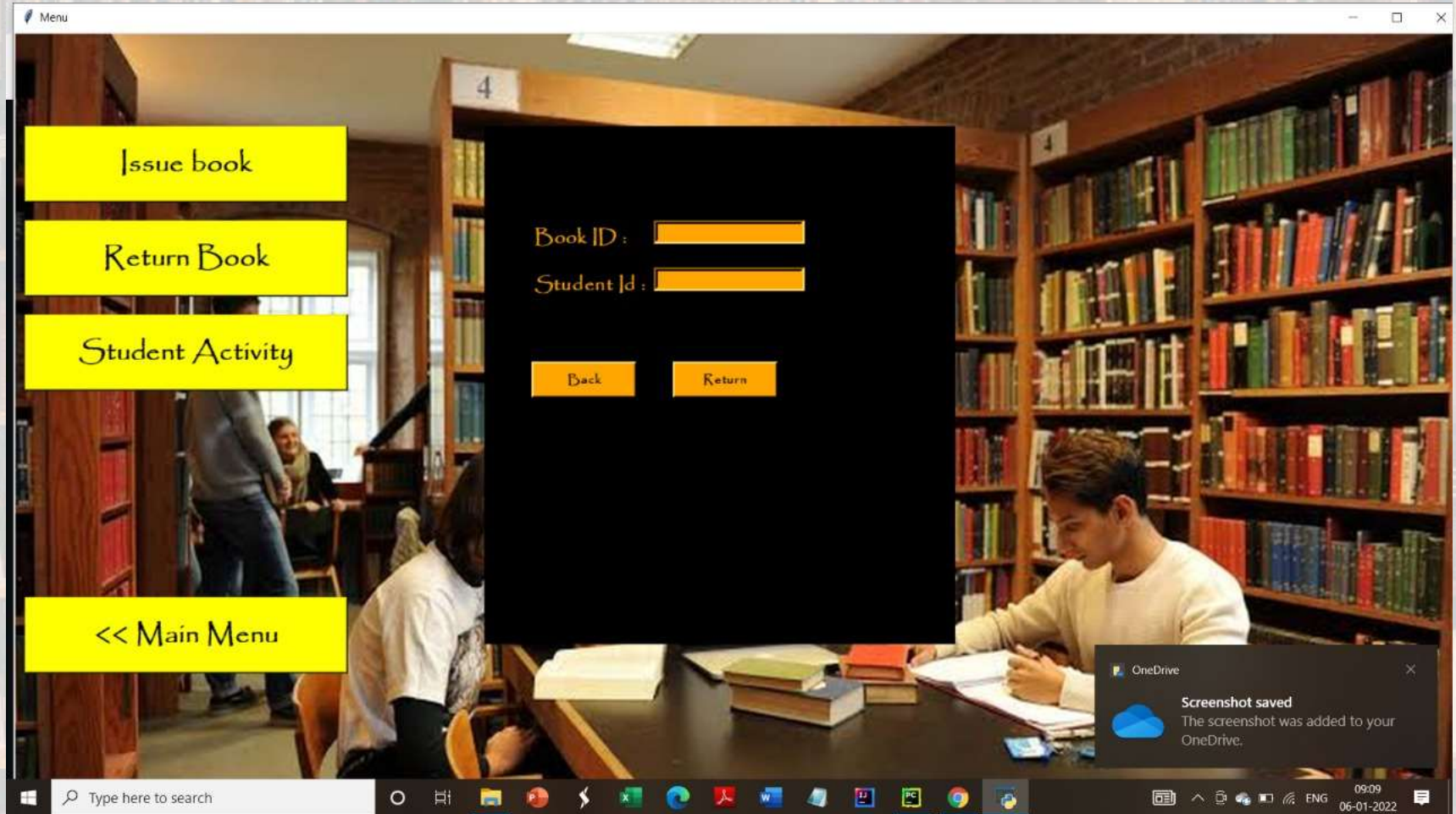


**Activity Page**

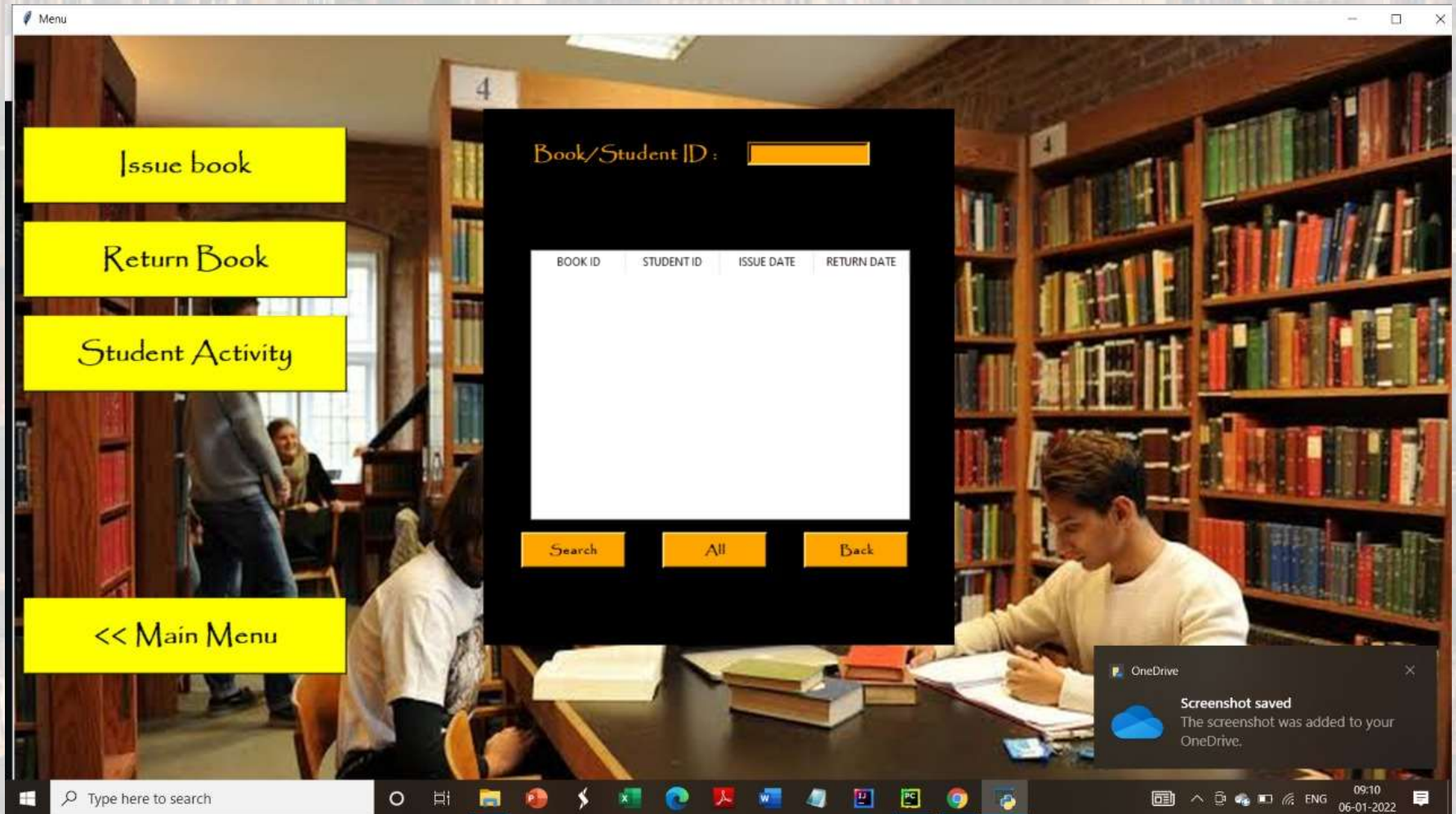


Issue





Return



Activity of any valid student is shown here.



# **PROJECT OUTCOME**

To make the existing system more efficient.

- To provide a user friendly environment where user can be serviced better.
- Make functioning of library faster.
- Provide a system where the library staff can catch defaulters and not let them escape.
- To minimize the loss done to books

# **CONCLUSION**

After we completed the project we were sure the problems in the existing system are overcome. The “LIBRARY MANAGEMENT SYSTEM” process made computerized to reduce human errors and to increase the efficiency. The main focus of this project is to lessen human efforts. The maintenance of the records is made efficient, as all the records are stored in the SQLite database, through which data can be retrieved easily. The editing is also made simpler. The user has to just

type in the required field and update the desired field. Our main aim of the project is to get the correct information about a particular student and books available in the library. The problems, which existed, have been removed to a large extent. And it is expected that this project will go a long way in satisfying user's requirements. The computerization of the Library Management will not only improve the efficiency but will also reduce human stress thereby indirectly improving human recourses.

## **REFERENCES**

- 1) For querying a doubt: [www.google.com](http://www.google.com)
- 2) For knowledge on Database and Advanced Python we took help from Youtube tutorials.





**Thank You...**