**Hackathon - Jackfruit Level Problem solving**

Date: 02/01/2025

**Team Details**:

Team 5

Srivani : PES1UG24CS471

Spoorthi P : PES1UG24AM279

Suraj HP : PES1UG24EC221

Srijan Akshit : PES1UG24AM284

Srijan S K: PES1UG24EC214

Class: P7

**Code:**

import csv

from datetime import datetime, timedelta

import tkinter as tk

from tkinter import ttk, messagebox

class Book:

def \_\_init\_\_(self, book\_id, title, author, available\_copies):

self.book\_id = book\_id

self.title = title

self.author = author

self.available\_copies = int(available\_copies)

def to\_csv(self):

return f"{self.book\_id},{self.title},{self.author},{self.available\_copies}"

class Student:

def \_\_init\_\_(self, student\_id, name):

self.student\_id = student\_id

self.name = name

self.borrowed\_books = []

def add\_borrowed\_book(self, book\_id):

self.borrowed\_books.append(book\_id)

def remove\_borrowed\_book(self, book\_id):

if book\_id in self.borrowed\_books:

self.borrowed\_books.remove(book\_id)

def to\_csv(self):

return f"{self.student\_id},{self.name},{','.join(self.borrowed\_books)}"

def is\_eligible(self):

return len(self.borrowed\_books) < 3 # Assuming a student can borrow up to 3 books

class Librarian:

def \_\_init\_\_(self):

self.books = []

self.students = []

self.logs = []

self.load\_data()

def load\_data(self):

with open('books.csv', 'r') as file:

reader = csv.reader(file)

next(reader) # Skip header

self.books = [Book(row[0], row[1], row[2], row[3]) for row in reader]

with open('students.csv', 'r') as file:

reader = csv.reader(file)

next(reader) # Skip header

self.students = []

for row in reader:

student\_id, name, \*borrowed\_books = row

student = Student(student\_id, name)

student.borrowed\_books = borrowed\_books # Load borrowed books correctly

self.students.append(student)

with open('logs.csv', 'r') as file:

reader = csv.reader(file)

next(reader) # Skip header

self.logs = [row for row in reader]

def save\_data(self):

with open('books.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['book\_id', 'title', 'author', 'available\_copies'])

writer.writerows([book.to\_csv().split(',') for book in self.books])

with open('students.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['student\_id', 'name', 'borrowed\_books'])

writer.writerows([student.to\_csv().split(',') for student in self.students])

with open('logs.csv', 'w', newline='') as file:

writer = csv.writer(file)

writer.writerow(['transaction\_id', 'transaction\_type', 'book\_id', 'student\_id', 'issue\_date', 'return\_date', 'penalty'])

writer.writerows(self.logs)

def check\_stock(self):

return [book for book in self.books if book.available\_copies > 0]

def search\_book(self, query):

return [book for book in self.books if query in (book.title, book.author, str(book.book\_id))]

def search\_student(self, query):

return [student for student in self.students if query in (student.student\_id, student.name)]

def issue\_book(self, book\_id, student\_id):

book = next((b for b in self.books if b.book\_id == book\_id), None)

student = next((s for s in self.students if s.student\_id == student\_id), None)

if book and student and book.available\_copies > 0 and student.is\_eligible():

book.available\_copies -= 1

student.add\_borrowed\_book(book\_id)

issue\_date = datetime.now().strftime('%m/%d/%Y')

self.update\_logs('issue', book\_id, student\_id, issue\_date)

self.save\_data()

return True

return False

def return\_book(self, book\_id, student\_id):

book = next((b for b in self.books if b.book\_id == book\_id), None)

student = next((s for s in self.students if s.student\_id == student\_id), None)

if book and student and book\_id in student.borrowed\_books:

book.available\_copies += 1

student.remove\_borrowed\_book(book\_id)

return\_date = datetime.now().strftime('%m/%d/%Y')

issue\_date = next((log[4] for log in self.logs if log[2] == book\_id and log[3] == student\_id and log[1] == 'issue'), None)

penalty = self.calculate\_penalty(issue\_date)

self.update\_logs('return', book\_id, student\_id, issue\_date, return\_date, penalty)

self.save\_data()

return penalty

return None

def calculate\_penalty(self, issue\_date):

issue\_date = datetime.strptime(issue\_date, '%m/%d/%Y')

print(issue\_date)

return\_date = datetime.now()

days\_late = (return\_date - issue\_date).days - 2

if days\_late > 0:

return min(days\_late \* 1, 50)

return 0

def update\_logs(self, transaction\_type, book\_id, student\_id, issue\_date, return\_date=None, penalty=0):

transaction\_id = len(self.logs) + 1

self.logs.append([transaction\_id, transaction\_type, book\_id, student\_id, issue\_date, return\_date, penalty])

class LibraryApp:

def \_\_init\_\_(self, root):

self.root = root

self.root.title("Library Management System")

self.librarian = Librarian()

self.create\_widgets()

def create\_widgets(self):

# Book Management Section

tk.Label(self.root, text="Search Book:").pack()

self.book\_search\_entry = tk.Entry(self.root)

self.book\_search\_entry.pack()

tk.Button(self.root, text="Search", command=self.search\_book).pack()

tk.Button(self.root, text="Check Stock", command=self.check\_stock).pack()

self.book\_results = tk.Text(self.root, height=10, width=50)

self.book\_results.pack()

# Student Management Section

tk.Label(self.root, text="Search Student:").pack()

self.student\_search\_entry = tk.Entry(self.root)

self.student\_search\_entry.pack()

tk.Button(self.root, text="Search", command=self.search\_student).pack()

self.student\_results = tk.Text(self.root, height=10, width=50)

self.student\_results.pack()

# Transaction Section

tk.Label(self.root, text="Issue Book:").pack()

self.issue\_book\_id\_entry = tk.Entry(self.root)

self.issue\_book\_id\_entry.pack()

self.issue\_student\_id\_entry = tk.Entry(self.root)

self.issue\_student\_id\_entry.pack()

tk.Button(self.root, text="Issue", command=self.issue\_book).pack()

tk.Label(self.root, text="Return Book:").pack()

self.return\_book\_id\_entry = tk.Entry(self.root)

self.return\_book\_id\_entry.pack()

self.return\_student\_id\_entry = tk.Entry(self.root)

self.return\_student\_id\_entry.pack()

tk.Button(self.root, text="Return", command=self.return\_book).pack()

self.penalty\_label = tk.Label(self.root, text="")

self.penalty\_label.pack()

def search\_book(self):

query = self.book\_search\_entry.get()

results = self.librarian.search\_book(query)

self.book\_results.delete(1.0, tk.END)

if results:

for book in results:

self.book\_results.insert(tk.END, f"{book.book\_id}, {book.title}, {book.author}, {book.available\_copies}\n")

else:

self.book\_results.insert(tk.END, "No results found.")

def check\_stock(self):

results = self.librarian.check\_stock()

self.book\_results.delete(1.0, tk.END)

if results:

for book in results:

self.book\_results.insert(tk.END, f"{book.book\_id}, {book.title}, {book.author}, {book.available\_copies}\n")

else:

self.book\_results.insert(tk.END, "No books available.")

def search\_student(self):

query = self.student\_search\_entry.get()

results = self.librarian.search\_student(query)

self.student\_results.delete(1.0, tk.END)

if results:

for student in results:

self.student\_results.insert(tk.END, f"{student.student\_id}, {student.name}, {', '.join(student.borrowed\_books)}\n")

else:

self.student\_results.insert(tk.END, "No results found.")

def issue\_book(self):

book\_id = self.issue\_book\_id\_entry.get().strip()

student\_id = self.issue\_student\_id\_entry.get().strip()

if not book\_id or not student\_id:

messagebox.showerror("Input Error", "Please enter both Book ID and Student ID.")

return

try:

success = self.librarian.issue\_book(book\_id, student\_id)

if success:

messagebox.showinfo("Success", "Book issued successfully.")

self.issue\_book\_id\_entry.delete(0, tk.END)

self.issue\_student\_id\_entry.delete(0, tk.END)

else:

messagebox.showerror("Error", "Failed to issue book. Please check the details.")

except PermissionError as e:

messagebox.showerror("Permission Error", f"Permission error: {e}")

except Exception as e:

messagebox.showerror("Error", f"An unexpected error occurred: {e}")

def return\_book(self):

book\_id = self.return\_book\_id\_entry.get()

student\_id = self.return\_student\_id\_entry.get()

penalty = self.librarian.return\_book(book\_id, student\_id)

if penalty is not None:

messagebox.showinfo("Success", f"Book returned successfully. Penalty: ${penalty}")

self.penalty\_label.config(text=f"Penalty: ${penalty}")

else:

messagebox.showerror("Error", "Failed to return book.")

if \_\_name\_\_ == "\_\_main\_\_":

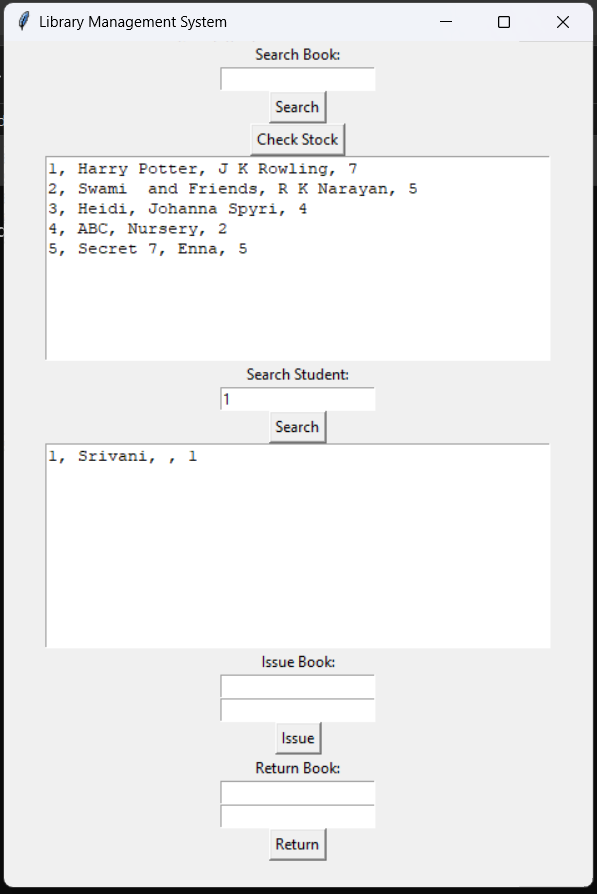
root = tk.Tk()

app = LibraryApp(root)

root.mainloop()

**Output (GUI) Screenshots:**

A screenshot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generatedA screenshot of a computer

Description automatically generated

**Handling Cases like :**

A screenshot of a computer

Description automatically generated1 Books out of Stock -

2. Returning a book not issued

A screenshot of a computer

Description automatically generated