Laxmi Charitable Trust's

Sheth L.U.J College of Arts & Sir M.V. College of Science and Commerce

Department of Information Technology (Bsc.IT Semester IV)

LIBRARY ENTRY MANAGEMENT SYSTEM

Group Members

Name Of Members	Roll No:-
1.Dakshata Naresh Kamble	S024
2.Janhavi Manoj Kandu	S025
3.Sandhya Vijay Tiwari	S047

Library Entry Management System GUI

Description:-

This application is a desktop system built with Python's Tkinter and SQLite, offering a user-friendly interface to manage book entries in a library. It supports basic CRUD operations—adding, viewing, modifying, and deleting book records.

Key Features:

1. **Main GUI Window:** The interface includes a dynamic, resizable background image and input fields for book details such as name, title, author, and publication date.

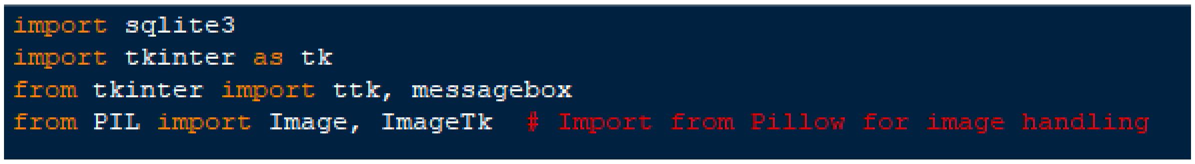
2.Treeview Display: Book records are displayed in a table format, with sorting based on ID, Name, Title, Author, and Publication Date.

3.Database Management: The system connects to a SQLite database, creating and managing a books table for storing records.

4.CRUD Operations:

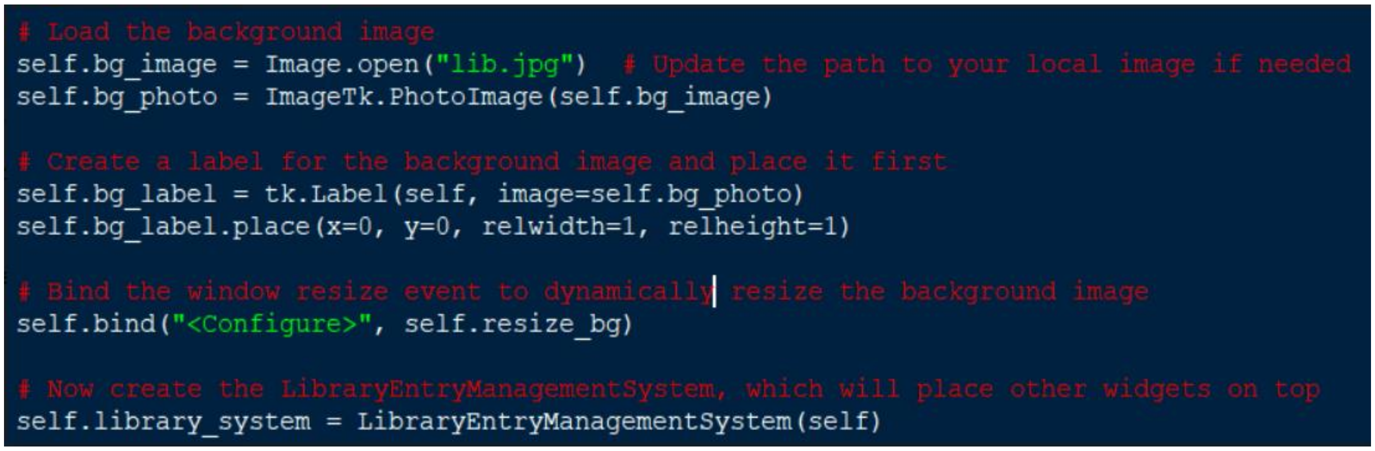
- Add Book: Inserts a new book record.
- Modify Record: Updates an existing record.
- Delete Record: Removes a record by ID.
- Auto-Refresh: The treeview updates automatically after each operation.

1: Importing Libraries and Initial Setup



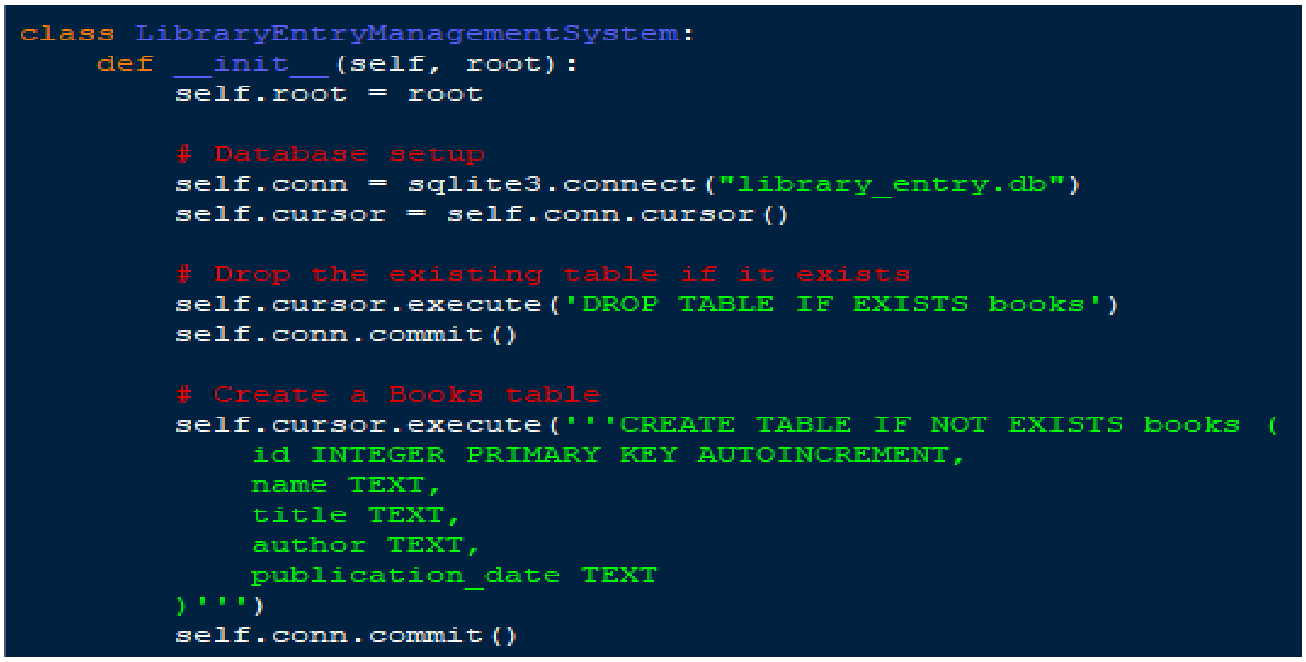
```python
import sqlite3
import tkinter as tk
from tkinter import ttk, messagebox
from PIL import Image, ImageTk  # Import from Pillow for image handling
```

2: LibraryEntryManagementGUI Class with Background Image Setup



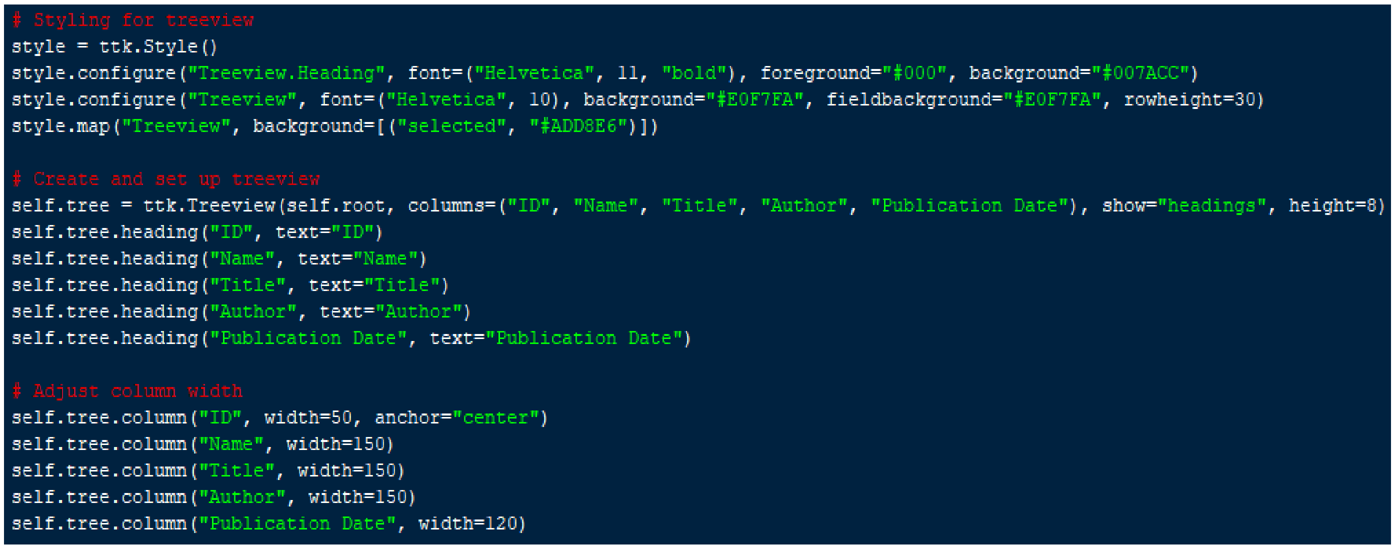
```python
# Load the background image
self.bg_image = Image.open("lib.jpg")  # Update the path to your local image if needed
self.bg_photo = ImageTk.PhotoImage(self.bg_image)

# Create a label for the background image and place it first
self.bg_label = tk.Label(self, image=self.bg_photo)
self.bg_label.place(x=0, y=0, relwidth=1, relheight=1)

# Bind the window resize event to dynamically resize the background image
self.bind("<Configure>", self.resize_bg)

# Now create the LibraryEntryManagementSystem, which will place other widgets on top
self.library_system = LibraryEntryManagementSystem(self)
```

3: LibraryEntryManagementSystem Class Initialization and Database Setup



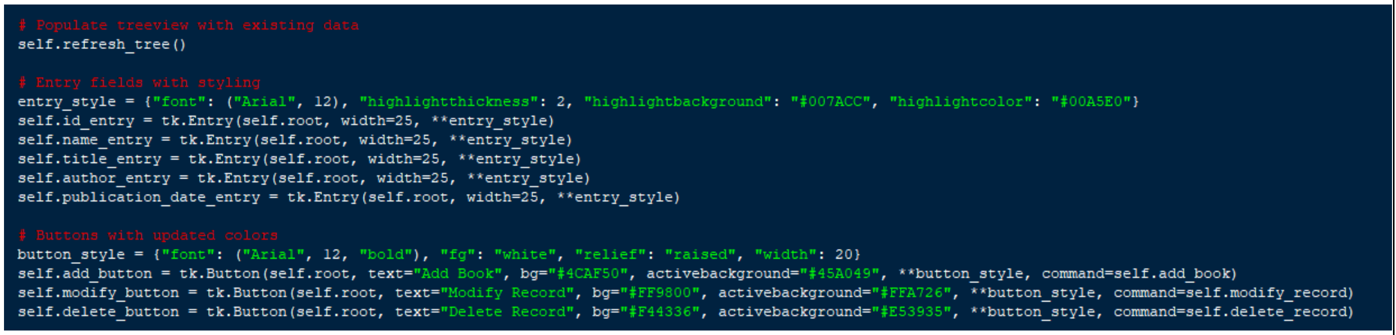
```python
class LibraryEntryManagementSystem:
    def __init__(self, root):
        self.root = root

        # Database setup
        self.conn = sqlite3.connect("library_entry.db")
        self.cursor = self.conn.cursor()

        # Drop the existing table if it exists
        self.cursor.execute('DROP TABLE IF EXISTS books')
        self.conn.commit()

        # Create a Books table
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS books (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            title TEXT,
            author TEXT,
            publication_date TEXT
        )''')
        self.conn.commit()
```

4: Treeview Setup

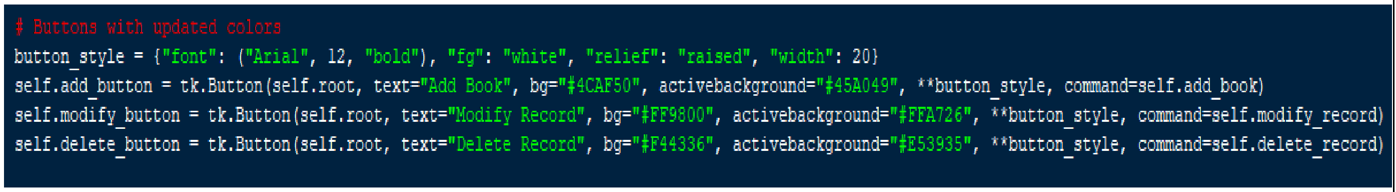


```
# Styling for treeview
style = ttk.Style()
style.configure("Treeview.Heading", font=("Helvetica", 11, "bold"), foreground="#000", background="#007ACC")
style.configure("Treeview", font=("Helvetica", 10), background="#E0F7FA", fieldbackground="#E0F7FA", rowheight=30)
style.map("Treeview", background=[("selected", "#ADD8E6")])

# Create and set up treeview
self.tree = ttk.Treeview(self.root, columns=("ID", "Name", "Title", "Author", "Publication Date"), show="headings", height=8)
self.tree.heading("ID", text="ID")
self.tree.heading("Name", text="Name")
self.tree.heading("Title", text="Title")
self.tree.heading("Author", text="Author")
self.tree.heading("Publication Date", text="Publication Date")

# Adjust column width
self.tree.column("ID", width=50, anchor="center")
self.tree.column("Name", width=150)
self.tree.column("Title", width=150)
self.tree.column("Author", width=150)
self.tree.column("Publication Date", width=120)
```

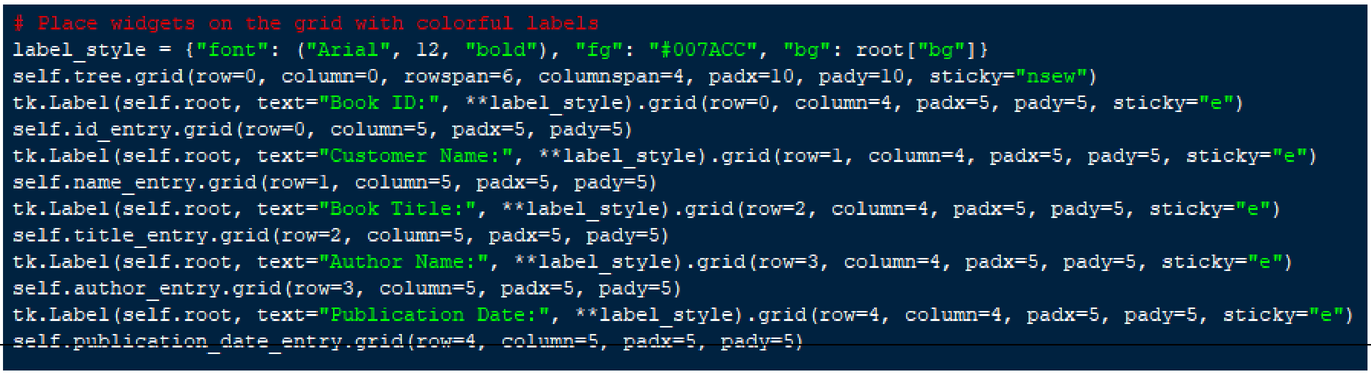
5: Entry Fields and Buttons Setup



```
# Populate treeview with existing data
self.refresh_tree()

# Entry fields with styling
entry_style = {"font": ("Arial", 12), "highlightthickness": 2, "highlightbackground": "#007ACC", "highlightcolor": "#00A5E0"}
self.id_entry = tk.Entry(self.root, width=25, **entry_style)
self.name_entry = tk.Entry(self.root, width=25, **entry_style)
self.title_entry = tk.Entry(self.root, width=25, **entry_style)
self.author_entry = tk.Entry(self.root, width=25, **entry_style)
self.publication_date_entry = tk.Entry(self.root, width=25, **entry_style)

# Buttons with updated colors
button_style = {"font": ("Arial", 12, "bold"), "fg": "white", "relief": "raised", "width": 20}
self.add_button = tk.Button(self.root, text="Add Book", bg="#4CAF50", activebackground="#45A049", **button_style, command=self.add_book)
self.modify_button = tk.Button(self.root, text="Modify Record", bg="#FF9800", activebackground="#FFA726", **button_style, command=self.modify_record)
self.delete_button = tk.Button(self.root, text="Delete Record", bg="#F44336", activebackground="#E53935", **button_style, command=self.delete_record)
```



```
# Buttons with updated colors
button_style = {"font": ("Arial", 12, "bold"), "fg": "white", "relief": "raised", "width": 20}
self.add_button = tk.Button(self.root, text="Add Book", bg="#4CAF50", activebackground="#45A049", **button_style, command=self.add_book)
self.modify_button = tk.Button(self.root, text="Modify Record", bg="#FF9800", activebackground="#FFA726", **button_style, command=self.modify_record)
self.delete_button = tk.Button(self.root, text="Delete Record", bg="#F44336", activebackground="#E53935", **button_style, command=self.delete_record)
```



```
# Place widgets on the grid with colorful labels
label_style = {"font": ("Arial", 12, "bold"), "fg": "#007ACC", "bg": root["bg"]}
self.tree.grid(row=0, column=0, rowspan=6, columnspan=4, padx=10, pady=10, sticky="nsew")
tk.Label(self.root, text="Book ID:", **label_style).grid(row=0, column=4, padx=5, pady=5, sticky="e")
self.id_entry.grid(row=0, column=5, padx=5, pady=5)
tk.Label(self.root, text="Customer Name:", **label_style).grid(row=1, column=4, padx=5, pady=5, sticky="e")
self.name_entry.grid(row=1, column=5, padx=5, pady=5)
tk.Label(self.root, text="Book Title:", **label_style).grid(row=2, column=4, padx=5, pady=5, sticky="e")
self.title_entry.grid(row=2, column=5, padx=5, pady=5)
tk.Label(self.root, text="Author Name:", **label_style).grid(row=3, column=4, padx=5, pady=5, sticky="e")
self.author_entry.grid(row=3, column=5, padx=5, pady=5)
tk.Label(self.root, text="Publication Date:", **label_style).grid(row=4, column=4, padx=5, pady=5, sticky="e")
self.publication_date_entry.grid(row=4, column=5, padx=5, pady=5)
```

6: Add Book Function

```python
def add_book(self):
    # Get values from entry fields
    name = self.name_entry.get()
    title = self.title_entry.get()
    author = self.author_entry.get()
    publication_date = self.publication_date_entry.get()

    if not name or not title or not author or not publication_date:
        messagebox.showwarning("Input Error", "Please fill all fields.")
        return

    # Insert the new book into the database
    try:
        self.cursor.execute('''INSERT INTO books (name, title, author, publication_date) VALUES (?, ?, ?, ?)''',
                            (name, title, author, publication_date))
        self.conn.commit()
        messagebox.showinfo("Success", "Book added successfully.")
        self.refresh_tree()
    except Exception as e:
        messagebox.showerror("Database Error", f"Failed to add book: {e}")

    # Clear entry fields after adding a book
    self.name_entry.delete(0, tk.END)
    self.title_entry.delete(0, tk.END)
    self.author_entry.delete(0, tk.END)
    self.publication_date_entry.delete(0, tk.END)
```

7: Modify and Delete Functions

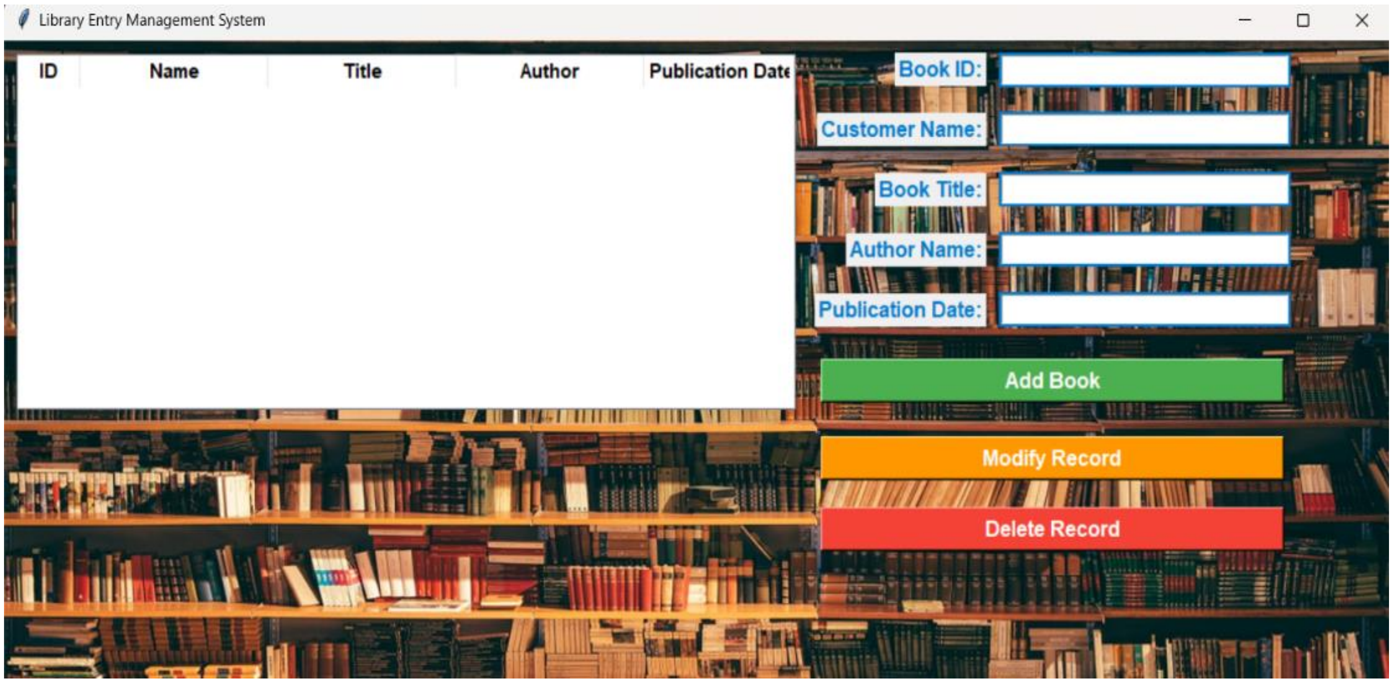
```python
def modify_record(self):
    # Get values from entry fields
    book_id = self.id_entry.get()
    name = self.name_entry.get()
    title = self.title_entry.get()
    author = self.author_entry.get()
    publication_date = self.publication_date_entry.get()

    if not book_id or not name or not title or not author or not publication_date:
        messagebox.showwarning("Input Error", "Please fill all fields.")
        return

    # Update the record in the database
    try:
        self.cursor.execute('''UPDATE books SET name=?, title=?, author=?, publication_date=? WHERE id=?''',
                            (name, title, author, publication_date, book_id))
        self.conn.commit()

        # Check if any row was updated
        if self.cursor.rowcount == 0:
            messagebox.showinfo("No Record Found", "No record found with that ID.")
        else:
            messagebox.showinfo("Success", "Record updated successfully.")
            self.refresh_tree()

    except Exception as e:
        messagebox.showerror("Database Error", f"Failed to modify record: {e}")

    # Clear entry fields after modifying a record
    self.id_entry.delete(0, tk.END)
    self.name_entry.delete(0, tk.END)
    self.title_entry.delete(0, tk.END)
    self.author_entry.delete(0, tk.END)
    self.publication_date_entry.delete(0, tk.END)
```

8: Closing the Application and Database Connection

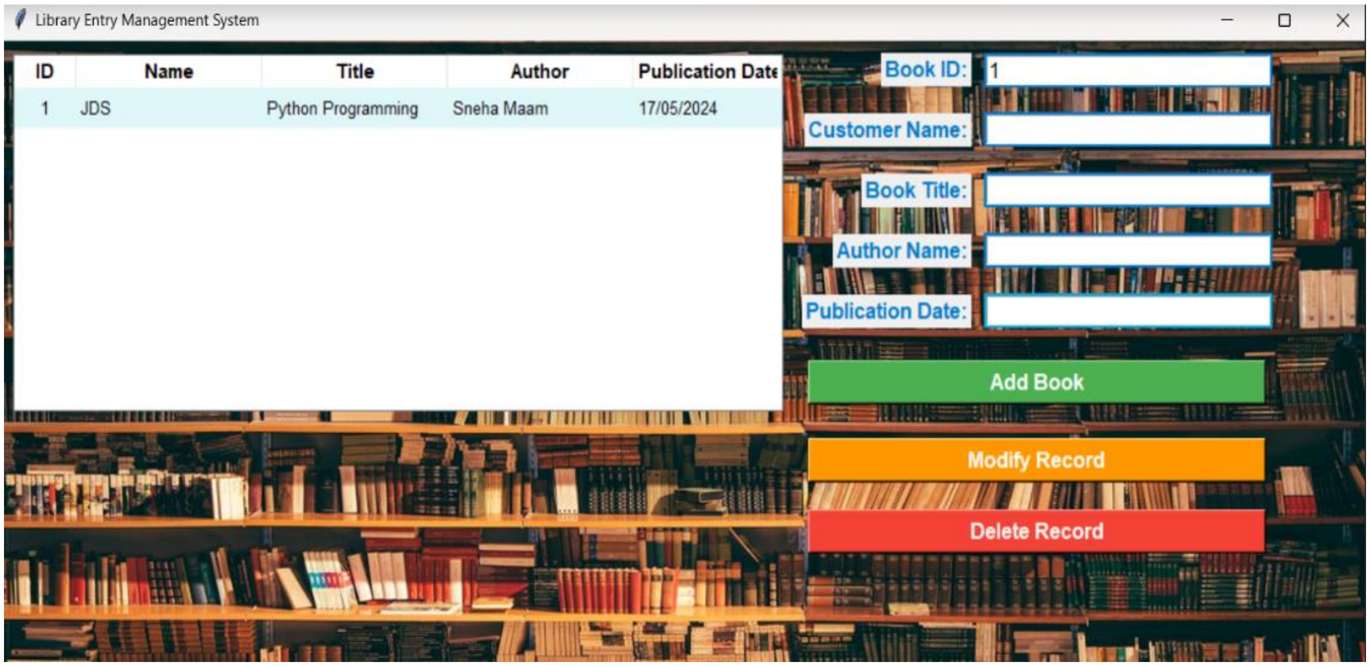
```
    def on_close(self):
        # Close the database connection before closing the application
        self.conn.close()
        self.root.destroy()

if __name__ == "__main__":
    app = LibraryEntryManagementGUI()
    app.mainloop()
```

Output:-

1.Graphic User Interferace



2.Details entered



Library Entry Management System

ID	Name	Title	Author	Publication Date
1	JDS	Python Programming	Sneha Maam	17/05/2024

Book ID:	1
Customer Name:	
Book Title:	
Author Name:	
Publication Date:	

Conclusion:

1. User-Friendly Interface: The Library Entry Management System offers a simple and intuitive GUI for managing book collections.

2. Essential Functionality: Users can easily add, view, modify, and delete book records from a SQLite database, making it ideal for small libraries or personal collections.

3. Minimal Setup: Designed for convenience, this tool provides core library management features without requiring complex installation.

4. Enhanced User Experience: The application includes dynamic features like resizable background images and stylish treeviews for a visually appealing and responsive design.

5. Practical and Efficient: It combines aesthetics and practicality, offering an organized and efficient way to manage a library's collection.