Library management system documentation

Architecture

Library management system is software that is designed by a client server architecture. It provides the ability to find books, manage books, track borrowed books and manage fines in one place. It helps the librarian manage the books and books borrowed by members and automate most of the library activities. It increases efficiency and reduces cost needed for maintaining a library and saves time and effort for both the user and the librarian.

Functionality

1. User Registration and Authentication: (10mks)

• Users should be able to register and create their accounts.

• Implement an authentication system to ensure secure access.

1. Book Management: (15mks)

• Allow librarians to add, edit, and delete book records.

• Include attributes such as book title, author, ISBN, publication date, etc.

• Implement a search functionality to find books based on various criteria.

1. Patron Management: (10mks)

• Enable librarians to add, edit, and delete patron records.

• Include attributes such as name, contact information, membership status, etc.

1. Borrowing System: (15mks)

• Design a system to handle book check-in and check-out processes.

• Track borrowing history, due dates, and fine calculations.

• Notify patrons about due dates and overdue books.

1. Reporting: (15mks)

• Generate reports such as the list of available books, borrowed books, overdue books, etc.

• Provide options to filter and sort reports based on specific criteria.

6. User Interface: (10mks)

• Create an intuitive and user-friendly interface using either Tkinter or Django's template system.

• Use appropriate widgets, forms, and layouts to enhance the user experience.

7. Data Persistence: (15mks)

• Implement a database system (e.g., SQLite, MySQL) to store and retrieve library data.

• Ensure proper data validation and integrity.

8. Documentation: (10mks)

• Provide detailed documentation explaining the system's architecture, functionality, and usage instructions.

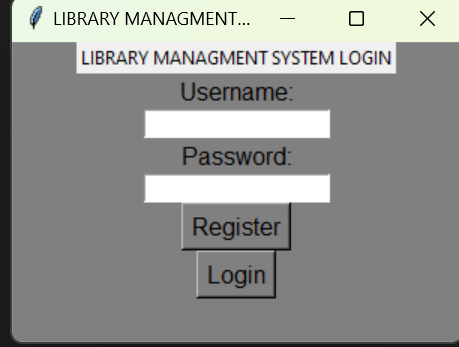
• Include diagrams, screenshots, and code snippets where necessary.

Usage Instructions

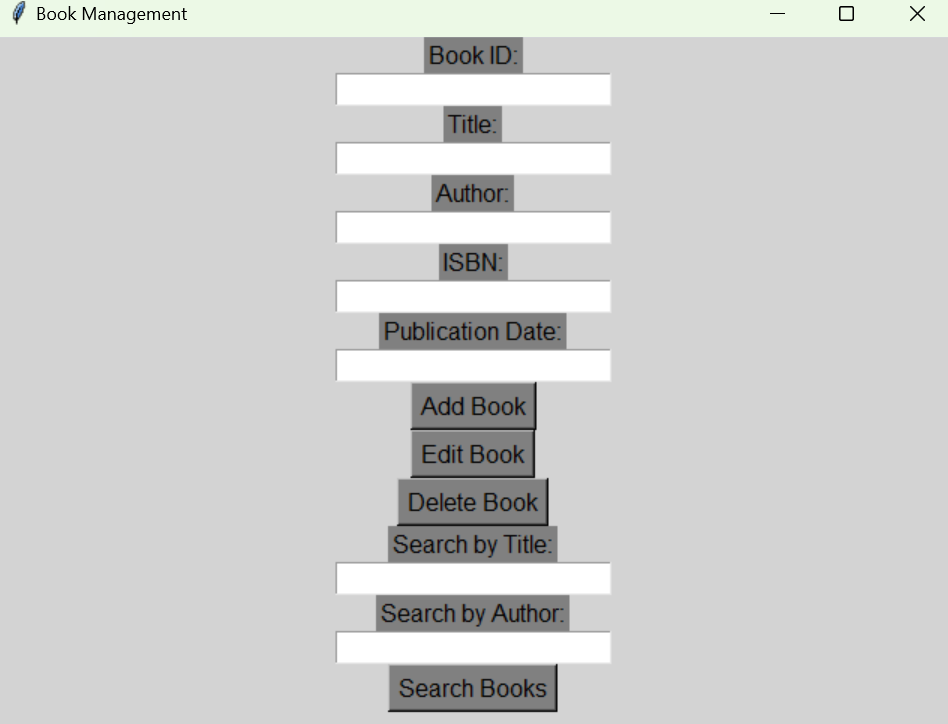
1. Install the necessary dependencies and libraries required to run the system. These may include a database system (e.g., SQLite, MySQL) and any specific Python libraries used by the system.
2. Set up the database by executing the provided database schema script or running the database migration commands, if applicable.
3. Start the server application by running the corresponding server script or command.
4. Launch the client application, which opens the user interface. You should see a login or registration screen.
5. If you are a new user, click on the "Register" button and fill in the required details to create an account. If you are an existing user, enter your credentials and click on the "Login" button.
6. Once logged in, you will be presented with the main menu or dashboard of the library management system. From here, you can navigate to different modules such as book management, patron management, borrowing system, or reporting.
7. Use the provided forms, buttons, and menus to perform various operations. For example, to add a new book, click on the "Add Book" button and fill in the necessary details in the book form. Click on "Save" to add the book to the system.
8. Explore the different features of the library management system, such as searching for books, editing patron records, checking in/out books, generating reports, and managing fines.
9. Ensure that you regularly backup the database to prevent data loss and maintain data integrity.
10. When you are done using the system, properly log out to ensure the security of your account and the system.

Screenshots

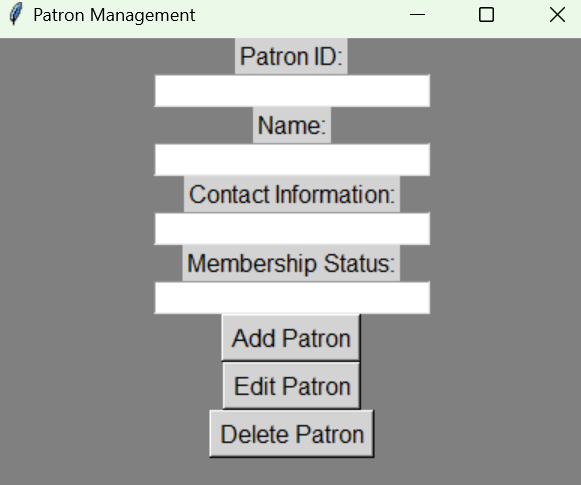
User Interface – User registration and authentication



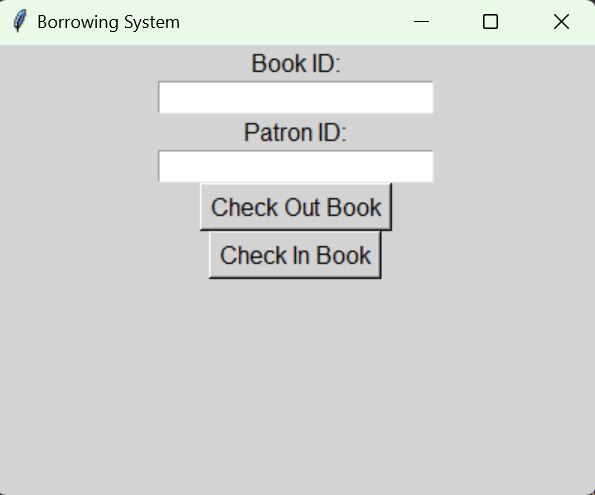
User Interface – Book management



User interface- Patron management



User Interface - Borrowing System



User Interface - Reporting



Code snippets

User registration and authentication

import tkinter as tk

from tkinter import Message, messagebox

import sqlite3

# Database initialization

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

c = conn.cursor()

# Create table if it doesn't exist

c.execute('''CREATE TABLE IF NOT EXISTS users

             (username TEXT NOT NULL, password TEXT NOT NULL)''')

conn.commit()

def register():

    username = username\_entry.get()

    password = password\_entry.get()

    # Check if the username already exists

    c.execute("SELECT \* FROM users WHERE username=?", (username,))

    existing\_user = c.fetchone()

    if existing\_user:

        messagebox.showerror("Error", "Username already exists. Please choose a different username.")

    else:

        # Insert the new user into the database

        c.execute("INSERT INTO users VALUES (?, ?)", (username, password))

        conn.commit()

        messagebox.showinfo("Success", "Registration successful!")

def login():

    username = username\_entry.get()

    password = password\_entry.get()

    # Check if the username and password match

    c.execute("SELECT \* FROM users WHERE username=? AND password=?", (username, password))

    existing\_user = c.fetchone()

    if existing\_user:

        messagebox.showinfo("Success", "Login successful!")

    else:

        messagebox.showerror("Error", "Invalid username or password.")

# Create the Tkinter window

root = tk.Tk()

root.title("LIBRARY MANAGMENT SYSTEM")

message=tk.Label(root,text="LIBRARY MANAGMENT SYSTEM LOGIN")

message.pack()

root.geometry("300x200")

root.config(bg="gray")

# Create username label and entry

username\_label = tk.Label(root, text="Username:",bg="gray",fg="black",font=("Arial", 12))

username\_label.pack()

username\_entry = tk.Entry(root)

username\_entry.pack()

# Create password label and entry

password\_label = tk.Label(root, text="Password:",bg="gray", fg="black", font=("Arial", 12))

password\_label.pack()

password\_entry = tk.Entry(root, show="\*")

password\_entry.pack()

# Create register and login buttons

register\_button = tk.Button(root, text="Register", command=register,bg="gray", fg="black", font=("Arial", 12))

register\_button.pack()

login\_button = tk.Button(root, text="Login", command=login,bg="gray", fg="black", font=("Arial", 12))

login\_button.pack()

# Start the Tkinter event loop

root.mainloop()

# Close the database connection

conn.close()

Book management

import tkinter as tk

from tkinter import messagebox

import sqlite3

# Create a database connection

conn = sqlite3.connect("books.db")

c = conn.cursor()

# Create a table to store book records

c.execute("""CREATE TABLE IF NOT EXISTS books (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                title TEXT,

                author TEXT,

                isbn TEXT,

                publication\_date TEXT

            )""")

conn.commit()

# Create the main application window

root = tk.Tk()

root.title("Book Management")

root.geometry("400x300")

root.config(bg="lightgray")

# Function to add a new book record

def add\_book():

    title = entry\_title.get()

    author = entry\_author.get()

    isbn = entry\_isbn.get()

    publication\_date = entry\_publication\_date.get()

    # Check if any field is empty

    if not title or not author or not isbn or not publication\_date:

        messagebox.showerror("Error", "Please enter all fields.")

        return

    # Insert the new book record into the database

    c.execute("INSERT INTO books (title, author, isbn, publication\_date) VALUES (?, ?, ?, ?)",

              (title, author, isbn, publication\_date))

    conn.commit()

    messagebox.showinfo("Success", "Book record added successfully.")

    # Clear the entry fields

    entry\_title.delete(0, tk.END)

    entry\_author.delete(0, tk.END)

    entry\_isbn.delete(0, tk.END)

    entry\_publication\_date.delete(0, tk.END)

# Function to edit a book record

def edit\_book():

    book\_id = entry\_book\_id.get()

    title = entry\_title.get()

    author = entry\_author.get()

    isbn = entry\_isbn.get()

    publication\_date = entry\_publication\_date.get()

    # Check if any field is empty

    if not book\_id or not title or not author or not isbn or not publication\_date:

        messagebox.showerror("Error", "Please enter all fields.")

        return

    # Update the book record in the database

    c.execute("UPDATE books SET title=?, author=?, isbn=?, publication\_date=? WHERE id=?",

              (title, author, isbn, publication\_date, book\_id))

    conn.commit()

    if c.rowcount > 0:

        messagebox.showinfo("Success", "Book record updated successfully.")

    else:

        messagebox.showerror("Error", "Invalid book ID.")

    # Clear the entry fields

    entry\_book\_id.delete(0, tk.END)

    entry\_title.delete(0, tk.END)

    entry\_author.delete(0, tk.END)

    entry\_isbn.delete(0, tk.END)

    entry\_publication\_date.delete(0, tk.END)

# Function to delete a book record

def delete\_book():

    book\_id = entry\_book\_id.get()

    # Check if the book ID is empty

    if not book\_id:

        messagebox.showerror("Error", "Please enter a book ID.")

        return

    # Delete the book record from the database

    c.execute("DELETE FROM books WHERE id=?", (book\_id,))

    conn.commit()

    if c.rowcount > 0:

        messagebox.showinfo("Success", "Book record deleted successfully.")

    else:

        messagebox.showerror("Error", "Invalid book ID.")

    # Clear the entry fields

    entry\_book\_id.delete(0, tk.END)

# Function to search for books

def search\_books():

    search\_title = entry\_search\_title.get()

    search\_author = entry\_search\_author.get()

    # Construct the query based on the search criteria

    query = "SELECT \* FROM books"

    if search\_title:

        query += f" WHERE title LIKE '%{search\_title}%'"

        if search\_author:

            query += f" AND author LIKE '%{search\_author}%'"

    elif search\_author:

        query += f" WHERE author LIKE '%{search\_author}%'"

    # Execute the query and fetch the results

    c.execute(query)

    books = c.fetchall()

    # Display the search results in a message box

    if books:

        result = ""

        for book in books:

            result += f"ID: {book[0]}\nTitle: {book[1]}\nAuthor: {book[2]}\nISBN: {book[3]}\nPublication Date: {book[4]}\n\n"

        messagebox.showinfo("Search Results", result)

    else:

        messagebox.showinfo("Search Results", "No books found.")

    # Clear the entry fields

    entry\_search\_title.delete(0, tk.END)

    entry\_search\_author.delete(0, tk.END)

# Book ID label and entry field

label\_book\_id = tk.Label(root, text="Book ID:", bg="gray", fg="black", font=("Arial", 12))

label\_book\_id.pack()

entry\_book\_id = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_book\_id.pack()

# Title label and entry field

label\_title = tk.Label(root, text="Title:", bg="gray", fg="black", font=("Arial", 12))

label\_title.pack()

entry\_title = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_title.pack()

# Author label and entry field

label\_author = tk.Label(root, text="Author:", bg="gray", fg="black", font=("Arial", 12))

label\_author.pack()

entry\_author = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_author.pack()

# ISBN label and entry field

label\_isbn = tk.Label(root, text="ISBN:", bg="gray", fg="black", font=("Arial", 12))

label\_isbn.pack()

entry\_isbn = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_isbn.pack()

# Publication Date label and entry field

label\_publication\_date = tk.Label(root, text="Publication Date:", bg="gray", fg="black", font=("Arial", 12))

label\_publication\_date.pack()

entry\_publication\_date = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_publication\_date.pack()

# Add Book button

btn\_add\_book = tk.Button(root, text="Add Book", command=add\_book, bg="gray", fg="black", font=("Arial", 12))

btn\_add\_book.pack()

# Edit Book button

btn\_edit\_book = tk.Button(root, text="Edit Book", command=edit\_book, bg="gray", fg="black", font=("Arial", 12))

btn\_edit\_book.pack()

# Delete Book button

btn\_delete\_book = tk.Button(root, text="Delete Book", command=delete\_book, bg="gray", fg="black", font=("Arial", 12))

btn\_delete\_book.pack()

# Search Title label and entry field

label\_search\_title = tk.Label(root, text="Search by Title:", bg="gray", fg="black", font=("Arial", 12))

label\_search\_title.pack()

entry\_search\_title = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_search\_title.pack()

# Search Author label and entry field

label\_search\_author = tk.Label(root, text="Search by Author:", bg="gray", fg="black", font=("Arial", 12))

label\_search\_author.pack()

entry\_search\_author = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_search\_author.pack()

# Search Books button

btn\_search\_books = tk.Button(root, text="Search Books", command=search\_books, bg="gray", fg="black", font=("Arial", 12))

btn\_search\_books.pack()

# Run the Tkinter event loop

root.mainloop()

# Close the database connection when the application is closed

conn.close()

Patron management

import tkinter as tk

from tkinter import messagebox

import sqlite3

# Create a database connection

conn = sqlite3.connect("patrons.db")

c = conn.cursor()

# Create a table to store patron records

c.execute("""CREATE TABLE IF NOT EXISTS patrons (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                name TEXT,

                contact\_info TEXT,

                membership\_status TEXT

            )""")

conn.commit()

# Create the main application window

root = tk.Tk()

root.title("Patron Management")

root.geometry("400x300")

root.config(bg="gray")

# Function to add a new patron record

def add\_patron():

    name = entry\_name.get()

    contact\_info = entry\_contact\_info.get()

    membership\_status = entry\_membership\_status.get()

    # Check if any field is empty

    if not name or not contact\_info or not membership\_status:

        messagebox.showerror("Error", "Please enter all fields.")

        return

    # Insert the new patron record into the database

    c.execute("INSERT INTO patrons (name, contact\_info, membership\_status) VALUES (?, ?, ?)",

              (name, contact\_info, membership\_status))

    conn.commit()

    messagebox.showinfo("Success", "Patron record added successfully.")

    # Clear the entry fields

    entry\_name.delete(0, tk.END)

    entry\_contact\_info.delete(0, tk.END)

    entry\_membership\_status.delete(0, tk.END)

# Function to edit a patron record

def edit\_patron():

    patron\_id = entry\_patron\_id.get()

    name = entry\_name.get()

    contact\_info = entry\_contact\_info.get()

    membership\_status = entry\_membership\_status.get()

    # Check if any field is empty

    if not patron\_id or not name or not contact\_info or not membership\_status:

        messagebox.showerror("Error", "Please enter all fields.")

        return

    # Update the patron record in the database

    c.execute("UPDATE patrons SET name=?, contact\_info=?, membership\_status=? WHERE id=?",

              (name, contact\_info, membership\_status, patron\_id))

    conn.commit()

    if c.rowcount > 0:

        messagebox.showinfo("Success", "Patron record updated successfully.")

    else:

        messagebox.showerror("Error", "Invalid patron ID.")

    # Clear the entry fields

    entry\_patron\_id.delete(0, tk.END)

    entry\_name.delete(0, tk.END)

    entry\_contact\_info.delete(0, tk.END)

    entry\_membership\_status.delete(0, tk.END)

# Function to delete a patron record

def delete\_patron():

    patron\_id = entry\_patron\_id.get()

    # Check if the patron ID is empty

    if not patron\_id:

        messagebox.showerror("Error", "Please enter a patron ID.")

        return

    # Delete the patron record from the database

    c.execute("DELETE FROM patrons WHERE id=?", (patron\_id,))

    conn.commit()

    if c.rowcount > 0:

        messagebox.showinfo("Success", "Patron record deleted successfully.")

    else:

        messagebox.showerror("Error", "Invalid patron ID.")

    # Clear the entry fields

    entry\_patron\_id.delete(0, tk.END)

# Patron ID label and entry field

label\_patron\_id = tk.Label(root, text="Patron ID:", bg="lightgray", fg="black", font=("Arial", 12))

label\_patron\_id.pack()

entry\_patron\_id = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_patron\_id.pack()

# Name label and entry field

label\_name = tk.Label(root, text="Name:", bg="lightgray", fg="black", font=("Arial", 12))

label\_name.pack()

entry\_name = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_name.pack()

# Contact Information label and entry field

label\_contact\_info = tk.Label(root, text="Contact Information:", bg="lightgray", fg="black", font=("Arial", 12))

label\_contact\_info.pack()

entry\_contact\_info = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_contact\_info.pack()

# Membership Status label and entry field

label\_membership\_status = tk.Label(root, text="Membership Status:", bg="lightgray", fg="black", font=("Arial", 12))

label\_membership\_status.pack()

entry\_membership\_status = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_membership\_status.pack()

# Add Patron button

btn\_add\_patron = tk.Button(root, text="Add Patron", command=add\_patron, bg="lightgray", fg="black", font=("Arial", 12))

btn\_add\_patron.pack()

# Edit Patron button

btn\_edit\_patron = tk.Button(root, text="Edit Patron", command=edit\_patron, bg="lightgray", fg="black", font=("Arial", 12))

btn\_edit\_patron.pack()

# Delete Patron button

btn\_delete\_patron = tk.Button(root, text="Delete Patron", command=delete\_patron, bg="lightgray", fg="black", font=("Arial", 12))

btn\_delete\_patron.pack()

# Run the Tkinter event loop

root.mainloop()

# Close the database connection when the application is closed

conn.close()

Borrowing system

import tkinter as tk

from tkinter import messagebox

import sqlite3

from datetime import date, timedelta

# Create a database connection

conn = sqlite3.connect("library.db")

c = conn.cursor()

# Create a table to store book records

c.execute("""CREATE TABLE IF NOT EXISTS books (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                title TEXT,

                author TEXT,

                isbn TEXT,

                publication\_date TEXT

            )""")

conn.commit()

# Create a table to store patron records

c.execute("""CREATE TABLE IF NOT EXISTS patrons (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                name TEXT,

                contact\_info TEXT,

                membership\_status TEXT

            )""")

conn.commit()

# Create a table to store borrowing records

c.execute("""CREATE TABLE IF NOT EXISTS borrowings (

                id INTEGER PRIMARY KEY AUTOINCREMENT,

                book\_id INTEGER,

                patron\_id INTEGER,

                borrow\_date TEXT,

                due\_date TEXT,

                return\_date TEXT

            )""")

conn.commit()

# Create the main application window

root = tk.Tk()

root.title("Borrowing System")

root.geometry("400x300")

root.config(bg="lightgray")

# Function to handle book check-out

def check\_out\_book():

    book\_id = entry\_book\_id.get()

    patron\_id = entry\_patron\_id.get()

    borrow\_date = date.today().strftime("%Y-%m-%d")

    due\_date = (date.today() + timedelta(days=14)).strftime("%Y-%m-%d")  # Due date is set to 14 days from the borrow date

    # Check if any field is empty

    if not book\_id or not patron\_id:

        messagebox.showerror("Error", "Please enter all fields.")

        return

    # Check if the book and patron exist in the database

    c.execute("SELECT \* FROM books WHERE id=?", (book\_id,))

    book = c.fetchone()

    c.execute("SELECT \* FROM patrons WHERE id=?", (patron\_id,))

    patron = c.fetchone()

    if not book or not patron:

        messagebox.showerror("Error", "Invalid book ID or patron ID.")

        return

    # Check if the book is already checked out

    c.execute("SELECT \* FROM borrowings WHERE book\_id=? AND return\_date IS NULL", (book\_id,))

    existing\_borrowing = c.fetchone()

    if existing\_borrowing:

        messagebox.showerror("Error", "The book is already checked out.")

        return

    # Insert the borrowing record into the database

    c.execute("INSERT INTO borrowings (book\_id, patron\_id, borrow\_date, due\_date) VALUES (?, ?, ?, ?)",

              (book\_id, patron\_id, borrow\_date, due\_date))

    conn.commit()

    messagebox.showinfo("Success", "Book checked out successfully.")

    # Clear the entry fields

    entry\_book\_id.delete(0, tk.END)

    entry\_patron\_id.delete(0, tk.END)

# Function to handle book check-in

def check\_in\_book():

    book\_id = entry\_book\_id.get()

    # Check if the book ID is empty

    if not book\_id:

        messagebox.showerror("Error", "Please enter a book ID.")

        return

    # Check if the book exists in the database

    c.execute("SELECT \* FROM books WHERE id=?", (book\_id,))

    book = c.fetchone()

    if not book:

        messagebox.showerror("Error", "Invalid book ID.")

        return

    # Check if the book is currently checked out

    c.execute("SELECT \* FROM borrowings WHERE book\_id=? AND return\_date IS NULL", (book\_id,))

    borrowing = c.fetchone()

    if not borrowing:

        messagebox.showerror("Error", "The book is not checked out.")

        return

    # Update the borrowing record in the database

    return\_date = date.today().strftime("%Y-%m-%d")

    c.execute("UPDATE borrowings SET return\_date=? WHERE id=?", (return\_date, borrowing[0]))

    conn.commit()

    messagebox.showinfo("Success", "Book checked in successfully.")

    # Clear the entry fields

    entry\_book\_id.delete(0, tk.END)

# Book ID label and entry field

label\_book\_id = tk.Label(root, text="Book ID:", bg="lightgray", fg="black", font=("Arial", 12))

label\_book\_id.pack()

entry\_book\_id = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_book\_id.pack()

# Patron ID label and entry field

label\_patron\_id = tk.Label(root, text="Patron ID:", bg="lightgray", fg="black", font=("Arial", 12))

label\_patron\_id.pack()

entry\_patron\_id = tk.Entry(root, bg="white", fg="black", font=("Arial", 12))

entry\_patron\_id.pack()

# Check Out Book button

btn\_check\_out\_book = tk.Button(root, text="Check Out Book", command=check\_out\_book, bg="lightgray", fg="black", font=("Arial", 12))

btn\_check\_out\_book.pack()

# Check In Book button

btn\_check\_in\_book = tk.Button(root, text="Check In Book", command=check\_in\_book, bg="lightgray", fg="black", font=("Arial", 12))

btn\_check\_in\_book.pack()

# Run the Tkinter event loop

root.mainloop()

# Close the database connection when the application is closed

conn.close()

Reporting

import tkinter as tk

from tkinter import messagebox

import sqlite3

# Create a database connection

conn = sqlite3.connect("library.db")

c = conn.cursor()

# Create the main application window

root = tk.Tk()

root.title("Reporting")

root.geometry("400x300")

root.config(bg="lightgray")

# Function to generate the list of available books report

def generate\_available\_books\_report():

    c.execute("SELECT \* FROM books WHERE id NOT IN (SELECT book\_id FROM borrowings WHERE return\_date IS NULL)")

    books = c.fetchall()

    report\_text = ""

    if books:

        report\_text = "Available Books:\n"

        for book in books:

            report\_text += f"Title: {book[1]}\nAuthor: {book[2]}\nISBN: {book[3]}\nPublication Date: {book[4]}\n\n"

    else:

        report\_text = "No available books."

    # Display the report in a message box

    messagebox.showinfo("Available Books Report", report\_text)

# Function to generate the list of borrowed books report

def generate\_borrowed\_books\_report():

    c.execute("SELECT \* FROM books WHERE id IN (SELECT book\_id FROM borrowings WHERE return\_date IS NULL)")

    books = c.fetchall()

    report\_text = ""

    if books:

        report\_text = "Borrowed Books:\n"

        for book in books:

            report\_text += f"Title: {book[1]}\nAuthor: {book[2]}\nISBN: {book[3]}\nPublication Date: {book[4]}\n\n"

    else:

        report\_text = "No borrowed books."

    # Display the report in a message box

    messagebox.showinfo("Borrowed Books Report", report\_text)

# Function to generate the list of overdue books report

def generate\_overdue\_books\_report():

    c.execute("SELECT \* FROM books WHERE id IN (SELECT book\_id FROM borrowings WHERE return\_date IS NULL AND due\_date < DATE('now'))")

    books = c.fetchall()

    report\_text = ""

    if books:

        report\_text = "Overdue Books:\n"

        for book in books:

            report\_text += f"Title: {book[1]}\nAuthor: {book[2]}\nISBN: {book[3]}\nPublication Date: {book[4]}\n\n"

    else:

        report\_text = "No overdue books."

    # Display the report in a message box

    messagebox.showinfo("Overdue Books Report", report\_text)

# Generate Available Books Report button

btn\_available\_books\_report = tk.Button(root, text="Generate Available Books Report", command=generate\_available\_books\_report, bg="lightgray", fg="black", font=("Arial", 12))

btn\_available\_books\_report.pack()

# Generate Borrowed Books Report button

btn\_borrowed\_books\_report = tk.Button(root, text="Generate Borrowed Books Report", command=generate\_borrowed\_books\_report, bg="lightgray", fg="black", font=("Arial", 12))

btn\_borrowed\_books\_report.pack()

# Generate Overdue Books Report button

btn\_overdue\_books\_report = tk.Button(root, text="Generate Overdue Books Report", command=generate\_overdue\_books\_report, bg="lightgray", fg="black", font=("Arial", 12))

btn\_overdue\_books\_report.pack()

# Run the Tkinter event loop

root.mainloop()

# Close the database connection when the application is closed

conn.close()