Source Code

Fontend

Admin UI

admin.py

```python
from tkinter import messagebox
import ttkbootstrap as ttk
from admin.login_screen import login_screen

def on_close():
    if messagebox.askyesno("Exit", "Are you sure you want to exit?"):
        app.destroy()

app = ttk.Window(themename="darkly")
app.geometry("1200x800")
app.state('zoomed')
app.title("LibPro | Library Management App")

app.protocol("WM_DELETE_WINDOW", on_close)

style = ttk.Style()
style.configure("crimson.TButton",
                background="#dc143c",
                foreground="white",
                borderwidth=0,
                focusthickness=0)

login_screen(app)
app.mainloop()
```

admin/login_screen.py

```python
import ttkbootstrap as ttk
from tkinter import messagebox
import tkinter as tk

from admin.dashboard import welcome_screen

ADMIN_CREDENTIALS = {
    "Pratham": "123",
    "Thejas": "456",
    "Thoshit": "789",
}

def login_screen(app):
    global login_frame, username_var, password_var

    login_frame = ttk.Frame(app, padding=30)
    login_frame.pack(expand=True, fill="both")

    branding_frame = ttk.Frame(login_frame, padding=20)
    branding_frame.pack(side="left", fill="both", expand=True, padx=20, pady=20)

    ttk.Label(branding_frame, anchor="center").pack(pady=70)

    ttk.Label(branding_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)

    ttk.Label(branding_frame, text="Library Management App", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)

    ttk.Label(branding_frame, text="By Pratham, Thejas and Thoshit.", font=("Arial", 12), anchor="center").pack(pady=50)

    login_section = ttk.Frame(login_frame, padding=20)
    login_section.pack(side="right", fill="both", expand=True, padx=20, pady=20)

    canvas = tk.Canvas(login_section, highlightthickness=0)
    canvas.pack(fill="both", expand=True)

    width = 500
    height = 500
    radius = 20
    canvas.create_oval(0, 0, radius * 2, radius * 2, outline="#ffffff")
    canvas.create_rectangle(radius, 0, width - radius, height, outline="#ffffff")
    canvas.create_oval(width - radius * 2, 0, width, radius * 2, outline="#ffffff")
    canvas.create_rectangle(0, radius, width, height - radius, outline="#ffffff")
    canvas.create_oval(0, height - radius * 2, radius * 2, height, outline="#ffffff")
    canvas.create_oval(width - radius * 2, height - radius * 2, width, height, outline="#ffffff")

    form_frame = ttk.Frame(canvas)
    canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")

    ttk.Label(form_frame, text="Admin Login", font=("Cambria", 30, "bold")).pack(pady=20)

    ttk.Label(form_frame, text="Username:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
    username_var = ttk.StringVar()
    ttk.Entry(form_frame, textvariable=username_var, font=("Century Gothic", 12)).pack(fill="x", pady=5)

    ttk.Label(form_frame, text="Password:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
    password_var = ttk.StringVar()
    ttk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 12), show="*").pack(fill="x", pady=5)

    ttk.Button(form_frame, text="Login", command=lambda: validate_login(app), style="crimson.TButton").pack(pady=20, fill="x")

    ttk.Label(form_frame, text="Welcome to the modern library experience.", font=("Calibri", 10, "italic")).pack(side="bottom", pady=10)


def validate_login(app):
    username = username_var.get().strip()
    password = password_var.get().strip()

    if not username or not password:
        messagebox.showerror("Error", "Username and password cannot be empty.")
        return

    if username in ADMIN_CREDENTIALS and ADMIN_CREDENTIALS[username] == password:
        messagebox.showinfo("Success", "Login successful!")
        login_frame.pack_forget()
        welcome_screen(app)
    else:
        messagebox.showerror("Error", "Invalid username or password.")

```

admin/dashboard.py

```python
import time
from tkinter import messagebox
import ttkbootstrap as ttk

from admin.shelf_manage import shelf_manage
from admin.membership_manage import membership_manage
from admin.books_manage import books_manage

def update_time(label):
    current_time = time.strftime("%H:%M:%S")
    current_date = time.strftime("%A, %B %d, %Y")
    label.config(text=f"Current Time: {current_time}\n{current_date}")
    label.after(1000, update_time, label)

def open_book_management(app):
    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return
    welcome_frame.pack_forget()
    books_manage(app)

def open_rack_management(app):
    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return
    welcome_frame.pack_forget()
    shelf_manage(app)

def logout(app):
    res = messagebox.askyesno("Confirm Logout", "Are you sure you want to logout?")
    if res:
        from ui.login_screen import login_screen
        welcome_frame.pack_forget()
        login_screen(app)

def open_membership_management(app):
    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return
    welcome_frame.pack_forget()
    membership_manage(app)

def welcome_screen(app):
    global welcome_frame
    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return
    welcome_frame = ttk.Frame(app, padding=30)
    welcome_frame.pack(expand=True, fill="both", pady=100)

    form_frame = ttk.Frame(welcome_frame)
    form_frame.pack(fill="both", expand=True)

    left_frame = ttk.Frame(form_frame, padding=20)
    left_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    ttk.Label(left_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
    ttk.Label(left_frame, text="Library Management App", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)

    time_label = ttk.Label(left_frame, font=("Arial", 12))
    time_label.pack(pady=10)
    update_time(time_label)

    greeting_label = ttk.Label(left_frame, text="Welcome Admin!", font=("Arial", 10, "bold"))
    greeting_label.pack(pady=10)

    right_frame = ttk.Frame(form_frame, padding=20)
    right_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

    book_management_button = ttk.Button(right_frame, text="Books Management", command=lambda: open_book_management(app), style="crimson.TButton")
    book_management_button.pack(pady=10, fill="x")

    book_rack_management_button = ttk.Button(right_frame, text="Book Rack Management", command=lambda: open_rack_management(app), style="crimson.TButton")
    book_rack_management_button.pack(pady=10, fill="x")

    membership_management_button = ttk.Button(right_frame, text="Membership Management", command=lambda: open_membership_management(app), style="crimson.TButton")
    membership_management_button.pack(pady=10, fill="x")

    logout_button = ttk.Button(right_frame, text="Logout", command=lambda: logout(app), style="crimson.TButton")
    logout_button.pack(pady=50, fill="both")

    form_frame.grid_columnconfigure(0, weight=1)
    form_frame.grid_columnconfigure(1, weight=1)
    form_frame.grid_rowconfigure(0, weight=1)

```

admin/books_manage.py

```python
from tkinter import ttk, messagebox
from admin.book_popups import open_add_book_popup, open_download_barcodes_popup, update_book_popup, open_delete_book_popup
from backend.books import Books, read_book

def back_to_dashboard(app):
    from admin.dashboard import welcome_screen

    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return

    for widget in app.winfo_children():
        widget.grid_forget()

    welcome_screen(app)

def books_manage(app):
    global table, table_frame

    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return

    app.grid_columnconfigure(0, weight=1)
    app.grid_rowconfigure(0, weight=1)

    table_frame = ttk.Frame(app)
    table_frame.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
    table_frame.grid_columnconfigure(0, weight=1)

    columns = ("ID", "ISBN", "Title", "Description", "Category", "Quantity", "SKU", "Author", "Language", "Publisher")
    table = ttk.Treeview(table_frame, columns=columns, show="headings", height=40)

    for col in columns:
        table.heading(col, text=col)
        table.column(col, width=100, anchor="center", stretch=True)

    scrollbar = ttk.Scrollbar(table_frame, orient="horizontal", command=table.xview)
    table.configure(xscrollcommand=scrollbar.set)
    scrollbar.grid(row=1, column=0, sticky="ew")

    table.grid(row=0, column=0, padx=20, pady=10, sticky="nsew")

    button_frame = ttk.Frame(app)
    button_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
    button_frame.grid_columnconfigure(0, weight=1)

    ttk.Button(button_frame, text="Add Book", command=lambda: open_add_book_popup(app, display_books), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Update Book", command=lambda: open_update_book_popup(app), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Delete Book", command=lambda: open_delete_book_popup(app, table, display_books), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Download Barcode", command=lambda: open_download_barcodes_popup(app, table), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").pack(fill="x", pady=50)

    app.grid_columnconfigure(0, weight=3)
    app.grid_columnconfigure(1, weight=1)

    display_books()

def open_update_book_popup(app):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No book selected!")
        return

    ISBN = str(table.item(selected_item)["values"][1])
    book = read_book(ISBN)

    if isinstance(book, str):
        messagebox.showerror("Error", "Book not found!")
        return

    update_book_popup(app, book, display_books)

def display_books():
    for row in table.get_children():
        table.delete(row)

    for index, book in enumerate(Books):
        table.insert("", "end", values=(
            index + 1,
            book["ISBN"],
            book["Title"],
            book["Description"],
            book["Category"],
            book["Quantity"],
            ", ".join(book["SKU"].keys()),
            book["Author"],
            book["Language"],
            book["Publisher"],
        ))

def update_table(table):
    for row in table.get_children():
        table.delete(row)

    for book in Books:
        table.insert("", "end", values=(book["ISBN"], book["Title"], book["Author"], book["Quantity"]))

```

admin/shelf_manage.py

```python
import tkinter as tk
from tkinter import StringVar, ttk, messagebox
from backend.shelfing import deshelve, search, shelf, categorise, BookShelf, DeshelvedBooks
from backend.utils import open_barcode_scanner

def back_to_dashboard(app):
    from admin.dashboard import welcome_screen

    for widget in app.winfo_children():
        widget.destroy()

    welcome_screen(app)

def show_deshelved_books():
    if not DeshelvedBooks:
        messagebox.showinfo("Deshelved Books", "No books have been deshelved.")
    else:
        books_list = "\n".join([f"SKU: {sku}, {details}" for sku, details in DeshelvedBooks.items()])
        messagebox.showinfo("Deshelved Books", f"Deshelved Books:\n\n{books_list}")


def open_shelve_popup(app, update_shelf_view):
    def shelve_book():
        try:
            rack = int(rack_entry.get())
            shelf_num = int(shelf_entry.get())
            sku = sku_entry.get()

            if not sku:
                raise ValueError("SKU cannot be empty.")

            result = shelf(rack, shelf_num, sku)
            messagebox.showinfo("Success", result)

            update_shelf_view()
            popup.destroy()

        except ValueError as e:
            messagebox.showerror("Error", str(e))

    popup = tk.Toplevel(app)
    popup.title("Shelve Book")

    tk.Label(popup, text="Rack Number").grid(row=0, column=0, padx=10, pady=5)
    rack_entry = tk.Entry(popup)
    rack_entry.grid(row=0, column=1, padx=10, pady=5)

    tk.Label(popup, text="Shelf Number").grid(row=1, column=0, padx=10, pady=5)
    shelf_entry = tk.Entry(popup)
    shelf_entry.grid(row=1, column=1, padx=10, pady=5)

    sku_var = StringVar()
    tk.Label(popup, text="SKU").grid(row=2, column=0, padx=10, pady=5)
    sku_entry = tk.Entry(popup, textvariable=sku_var)
    sku_entry.grid(row=2, column=1, padx=10, pady=5)

    def start_scanning():
        open_barcode_scanner(sku_var)
    ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=3, column=0, columnspan=2, pady=20)

    ttk.Button(popup, text="Submit", command=shelve_book, style="crimson.TButton").grid(row=4, column=0, columnspan=2, pady=20)


def open_search_popup(app):
    def search_book():
        sku = sku_entry.get()

        if not sku:
            messagebox.showerror("Error", "SKU cannot be empty.")
            return

        result = search(sku)
        if "Error" in result:
            messagebox.showerror("Error", result)
        else:
            messagebox.showinfo("Success", result)
            popup.destroy()

    popup = tk.Toplevel(app)
    popup.title("Search Book")
    sku_var = StringVar()

    tk.Label(popup, text="Enter SKU").grid(row=0, column=0, padx=10, pady=5)
    sku_entry = tk.Entry(popup, textvariable=sku_var)
    sku_entry.grid(row=0, column=1, padx=10, pady=5)
```

```python
    def start_scanning():
        open_barcode_scanner(sku_var)
    ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=1, column=0, columnspan=2, pady=

    ttk.Button(popup, text="Search", command=search_book, style="crimson.TButton").grid(row=2, column=0, columnspan=2, pady=20)


def open_deshelve_popup(app, update_shelf_view):
    sku_var = StringVar()
    def deshelve_book():
        sku = sku_entry.get()

        if not sku:
            messagebox.showerror("Error", "SKU cannot be empty.")
            return

        result = deshelve(sku)
        if "Error" in result:
            messagebox.showerror("Error", result)
        else:
            messagebox.showinfo("Success", result)

        update_shelf_view()
        popup.destroy()

    popup = tk.Toplevel(app)
    popup.title("Deshelve Book")
    def start_scanning():
        open_barcode_scanner(sku_var)

    tk.Label(popup, text="Enter SKU").grid(row=0, column=0, padx=10, pady=5)
    sku_entry = tk.Entry(popup, textvariable=sku_var)
    sku_entry.grid(row=0, column=1, padx=10, pady=5)

    ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=1, column=0, columnspan=2, pady=
    ttk.Button(popup, text="Deshelve", command=deshelve_book, style="crimson.TButton").grid(row=2, column=0, columnspan=2, pady=10)


def open_categorise_popup(app, update_shelf_view):
    def categorise_book():
        try:
            rack = int(rack_entry.get())
            shelf_num = int(shelf_entry.get())
            category = category_entry.get()

            if not category:
                raise ValueError("Category cannot be empty.")

            result = categorise(rack, shelf_num, category)
            messagebox.showinfo("Success", result)

            update_shelf_view()
            popup.destroy()

        except ValueError as e:
            messagebox.showerror("Error", str(e))

    popup = tk.Toplevel(app)
    popup.title("Categorize Book")

    tk.Label(popup, text="Rack Number").grid(row=0, column=0, padx=10, pady=5)
    rack_entry = tk.Entry(popup)
    rack_entry.grid(row=0, column=1, padx=10, pady=5)

    tk.Label(popup, text="Shelf Number").grid(row=1, column=0, padx=10, pady=5)
    shelf_entry = tk.Entry(popup)
    shelf_entry.grid(row=1, column=1, padx=10, pady=5)

    tk.Label(popup, text="Category").grid(row=2, column=0, padx=10, pady=5)
    category_entry = tk.Entry(popup)
    category_entry.grid(row=2, column=1, padx=10, pady=5)

    ttk.Button(popup, text="Submit", command=categorise_book, style="crimson.TButton").grid(row=3, column=0, columnspan=2, pady=10)
```

```python
def shelf_manage(app):
    def update_shelf_view():
        for widget in scrollable_frame.winfo_children():
            widget.destroy()

        if not BookShelf:
            no_rack_label = ttk.Label(scrollable_frame, text="No rack found!", font=("Arial", 16, "bold"))
            no_rack_label.grid(row=0, column=0, padx=10, pady=10)
            return

        row = 0
        col = 0
        max_columns = 3

        for rack_number, categories in enumerate(BookShelf, 1):
            rack_frame = ttk.Frame(scrollable_frame, padding=10, relief="solid", width=200, height=300)
            rack_frame.grid(row=row, column=col, padx=20, pady=20, sticky="nsew")

            rack_label = ttk.Label(rack_frame, text=f"Rack {rack_number}", font=("Arial", 14, "bold"))
            rack_label.grid(row=0, column=0, padx=10, pady=5)

            shelf_row = 1
            for category_dict in categories:
                if category_dict:
                    for category_name, skus in category_dict.items():
                        shelf_frame = ttk.Frame(rack_frame, padding=10, relief="groove")
                        shelf_frame.grid(row=shelf_row, column=0, padx=5, pady=5, sticky="nsew")

                        shelf_label = ttk.Label(shelf_frame, text=category_name, font=("Arial", 12))
                        shelf_label.grid(row=0, column=0, padx=5, pady=5)

                        table_frame = ttk.Frame(shelf_frame)
                        table_frame.grid(row=1, column=0, padx=5, pady=5)

                        columns = ("SKU",)
                        table = ttk.Treeview(table_frame, columns=columns, show="headings", height=4)
                        table.grid(row=0, column=0, sticky="nsew")

                        table.heading("SKU", text="SKU")

                        for sku in skus:
                            table.insert("", "end", values=(sku,))

                        shelf_row += 1

            col += 1
            if col >= max_columns:
                col = 0
                row += 1

        scrollable_frame.update_idletasks()
        canvas.config(scrollregion=canvas.bbox("all"))

    paned_window = ttk.PanedWindow(app, orient="horizontal")
    paned_window.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)

    left_frame = ttk.Frame(paned_window)
    paned_window.add(left_frame, weight=4)

    right_frame = ttk.Frame(paned_window, width=200)
    paned_window.add(right_frame, weight=1)

    canvas = tk.Canvas(left_frame)
    canvas.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)

    scrollable_frame = ttk.Frame(canvas)
    canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")

    scrollbar = ttk.Scrollbar(left_frame, orient="vertical", command=canvas.yview)
    scrollbar.grid(row=0, column=1, sticky="ns")
    canvas.configure(yscrollcommand=scrollbar.set)

    left_frame.grid_rowconfigure(0, weight=1)
    left_frame.grid_columnconfigure(0, weight=1)

    ttk.Button(right_frame, text="Categorize Book", command=lambda: open_categorise_popup(app, update_shelf_view), style="crimson.TButton").grid(row=0, column=0, pady=10)
    ttk.Button(right_frame, text="Shelve Book", command=lambda: open_shelve_popup(app,update_shelf_view), style="crimson.TButton").grid(row=1, column=0, pady=10)
    ttk.Button(right_frame, text="Search Book", command=lambda: open_search_popup(app), style="crimson.TButton").grid(row=2, column=0, pady=10)
    ttk.Button(right_frame, text="Deshelve Book", command=lambda: open_deshelve_popup(app,update_shelf_view), style="crimson.TButton").grid(row=3, column=0, pady=10)
    ttk.Button(right_frame, text="View Deshelved Books", command=lambda: show_deshelved_books(), style="crimson.TButton").grid(row=4, column=0, pady=10)
    ttk.Button(right_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").grid(row=5, column=0, pady=50)

    app.grid_rowconfigure(0, weight=1)
    app.grid_columnconfigure(0, weight=1)

    update_shelf_view()
```

admin/membership_manage.py

```python
from tkinter import ttk, messagebox
from admin.member_popups import open_add_member_popup, open_update_member_book_popup, update_member_popup, open_delete_member_popup
from backend.members import Members, read_member

def back_to_dashboard(app):
    from admin.dashboard import welcome_screen

    for widget in app.winfo_children():
        widget.grid_forget()

    welcome_screen(app)

def membership_manage(app):
    global table, table_frame

    app.grid_columnconfigure(0, weight=1)
    app.grid_rowconfigure(0, weight=1)

    table_frame = ttk.Frame(app)
    table_frame.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
    table_frame.grid_columnconfigure(0, weight=1)

    columns = ("UID", "Name", "Email", "Borrowed Books", "Join Date")
    table = ttk.Treeview(table_frame, columns=columns, show="headings", height=40)

    for col in columns:
        table.heading(col, text=col)
        table.column(col, width=100, anchor="center", stretch=True)

    scrollbar = ttk.Scrollbar(table_frame, orient="horizontal", command=table.xview)
    table.configure(xscrollcommand=scrollbar.set)
    scrollbar.grid(row=1, column=0, sticky="ew")

    table.grid(row=0, column=0, padx=20, pady=10, sticky="nsew")

    button_frame = ttk.Frame(app)
    button_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
    button_frame.grid_columnconfigure(0, weight=1)

    ttk.Button(button_frame, text="Add New Member", command=lambda: open_add_member_popup(app, display_members), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Update Member Details", command=lambda: open_update_member_popup(app), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Delete Member", command=lambda: open_delete_member_popup(app, table, display_members), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Borrow Book", command=lambda: open_update_member_book_popup(app, table, True, display_members), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Return Book", command=lambda: open_update_member_book_popup(app, table, False, display_members), style="crimson.TButton").pack(fill="x", pady=5)
    ttk.Button(button_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").pack(fill="x", pady=50)

    app.grid_columnconfigure(0, weight=3)
    app.grid_columnconfigure(1, weight=1)

    display_members()

def open_update_member_popup(app):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No member selected!")
        return

    UID = table.item(selected_item)["values"][0]
    if not UID:
        messagebox.showerror("Error", "Invalid UID!")
        return

    member = read_member(UID)

    if isinstance(member, str):
        messagebox.showerror("Error", "Member not found!")
        return

    update_member_popup(app, member, display_members)

def display_members():
    for row in table.get_children():
        table.delete(row)

    for index, member in enumerate(Members):
        table.insert("", "end", values=(
            member["UID"],
            member["Name"],
            member["Email"],
            member["SKU"],
            member["JoinedOn"],
        ))

def update_table(table):
    for row in table.get_children():
        table.delete(row)

    for member in Members:
        table.insert("", "end", values=(member["UID"], member["Name"], member["Email"], member["SKU"], member["JoinedOn"]))
```

admin/book_popup.py

```python
from tkinter import filedialog
import ttkbootstrap as ttk
import tkinter as tk
from tkinter import StringVar, messagebox
from backend.books import add_book, download_barcodes, update_books, remove_books

def get_selected_book(table):
    selected = table.focus()
    if selected:
        return table.item(selected, "values")
    return None

def open_add_book_popup(app, refresh_table_callback):
    popup = ttk.Toplevel(app)
    popup.geometry("600x400")
    popup.title("Add Book")
    popup.configure(bg="white")

    canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
    canvas.pack(fill="both", expand=True)

    width = 600
    height = 400

    form_frame = ttk.Frame(canvas)
    canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
    form_frame.grid_rowconfigure(0, weight=1)

    ttk.Label(form_frame, text="Add Book", font=("Cambria", 18, "bold")).grid(row=0, columnspan=2, pady=20)

    isbn_var = StringVar()
    title_var = StringVar()
    description_var = StringVar()
    category_var = StringVar()
    quantity_var = StringVar()
    author_var = StringVar()
    publisher_var = StringVar()
    language_var = StringVar()

    fields = [
        ("ISBN", isbn_var),
        ("Title", title_var),
        ("Description", description_var),
        ("Category", category_var),
        ("Quantity", quantity_var),
        ("Author", author_var),
        ("Publisher", publisher_var),
        ("Language", language_var),
    ]

    entry_vars = {}

    for i, (label, var) in enumerate(fields):
        row = i // 2
        col = i % 2

        entry_vars[label] = var

        ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).grid(row=row + 1, column=col * 2, padx=5, pady=5, sticky="w")
        ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).grid(row=row + 1, column=col * 2 + 1, padx=5, pady=5, sticky="ew")

    def handle_add_book():
        try:
            isbn = isbn_var.get().strip()
            title = title_var.get().strip()
            description = description_var.get().strip()
            category = category_var.get().strip()
            quantity = int(quantity_var.get())
            author = author_var.get().strip()
            publisher = publisher_var.get().strip()
            language = language_var.get().strip()

            if not isbn or not title or not description or not category or not author or not publisher or not language:
                messagebox.showerror("Error", "All fields must be filled in.")
                return

            result = add_book(
                ISBN=isbn,
                Title=title,
                Description=description,
                Category=category,
                Quantity=quantity,
                Author=author,
                Publisher=publisher,
                Language=language,
            )

```

admin/book_popup.py

```python

            if result.startswith("Invalid"):
                messagebox.showerror("Error", result)
            elif result.startswith("Book quantity updated") or result.startswith("Book added"):
                messagebox.showinfo("Success", result)
                refresh_table_callback()
                popup.destroy()

        except ValueError:
            messagebox.showerror("Error", "Please enter valid data for Quantity.")

    ttk.Button(form_frame, text="Add Book", command=handle_add_book, style="crimson.TButton").grid(row=len(fields) // 2 + 2, columnspan=2, pady=10)

    ttk.Label(form_frame, text="Fill the details and click 'Add Book' to add the book.", font=("Calibri", 10, "italic")).grid(row=len(fields) // 2 + 3, columnspan=2, pady=10)

def update_book_popup(app, book_data, refresh_table_callback):
    popup = ttk.Toplevel(app)
    popup.geometry("600x400")
    popup.title("Update Book")
    popup.configure(bg="white")

    canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
    canvas.pack(fill="both", expand=True)

    width = 600
    height = 400

    form_frame = ttk.Frame(canvas)
    canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
    form_frame.grid_rowconfigure(0, weight=1)

    ttk.Label(form_frame, text="Update Book", font=("Cambria", 18, "bold")).grid(row=0, columnspan=2, pady=20)

    title_var = StringVar(value=book_data.get("Title", ""))
    description_var = StringVar(value=book_data.get("Description", ""))
    category_var = StringVar(value=book_data.get("Category", ""))
    author_var = StringVar(value=book_data.get("Author", ""))
    publisher_var = StringVar(value=book_data.get("Publisher", ""))
    language_var = StringVar(value=book_data.get("Language", ""))

    fields = [
        ("Title", title_var),
        ("Description", description_var),
        ("Category", category_var),
        ("Author", author_var),
        ("Publisher", publisher_var),
        ("Language", language_var),
    ]

    entry_vars = {}

    for i, (label, var) in enumerate(fields):
        row = i // 2
        col = i % 2

        entry_vars[label] = var

        ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).grid(row=row + 1, column=col * 2, padx=5, pady=5, sticky="w")
        ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).grid(row=row + 1, column=col * 2 + 1, padx=5, pady=5, sticky="ew")

    def save_changes():
        updated_data = {label: var.get().strip() for label, var in entry_vars.items()}
        updated_data["ISBN"] = book_data.get("ISBN", "")

        if not updated_data["ISBN"]:
            messagebox.showerror("Error", "ISBN is required.")
            return

        if any(not value for value in updated_data.values()):
            messagebox.showerror("Error", "All fields must be filled in.")
            return

        result = update_books(
            updated_data["ISBN"],
            updated_data["Title"],
            updated_data["Description"],
            updated_data["Category"],
            updated_data["Author"],
            updated_data["Publisher"],
            updated_data["Language"]
        )

        if result.startswith("Invalid"):
            messagebox.showerror("Error", result)
        elif result.startswith("Book details updated"):
            messagebox.showinfo("Success", result)
            refresh_table_callback()
            popup.destroy()

    ttk.Button(form_frame, text="Save Changes", command=save_changes, style="crimson.TButton").grid(row=len(fields) // 2 + 2, columnspan=2, pady=10)

    ttk.Label(form_frame, text="Modify the details and click 'Save Changes' to update the book.", font=("Calibri", 10, "italic")).grid(row=len(fields) // 2 + 3, columnspan=2, pady=10)
```

admin/book_popup.py

```python
def open_delete_book_popup(app, table, refresh_table_callback):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No book selected!")
        return

    popup = ttk.Toplevel(app)
    popup.geometry("300x300")
    popup.title("Delete Book")
    popup.configure(bg="white")

    canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
    canvas.pack(fill="both", expand=True)

    width, height = 300, 300

    form_frame = ttk.Frame(canvas)
    canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
    form_frame.grid_rowconfigure(0, weight=1)

    ttk.Label(form_frame, text="Enter SKU no. to delete.", font=("Cambria", 8, "bold")).grid(row=0, columnspan=2, pady=20)
    sku_var = ttk.StringVar()

    ttk.Label(form_frame, text="SKU:", font=("Century Gothic", 8)).grid(row=1, column=0, padx=5, pady=5, sticky="e")
    sku_entry = ttk.Entry(form_frame, textvariable=sku_var, font=("Century Gothic", 10))
    sku_entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")

    def delete_book_by_sku():
        sku_number = sku_var.get().strip()

        if not sku_number:
            messagebox.showerror("Error", "Please enter a valid SKU number.")
            return

        try:
            sku_list = table.item(selected_item)["values"][6].split(",")
            sku = sku_list[int(sku_number) - 1]

            result = remove_books(sku)

            if "Error:" in result:
                messagebox.showerror("Error", result)
            else:
                messagebox.showinfo("Success", result)
                refresh_table_callback()
            popup.destroy()

        except IndexError:
            messagebox.showerror("Error", "Invalid SKU number. Please enter a valid SKU index.")
        except ValueError:
            messagebox.showerror("Error", "Invalid SKU number. Please enter a valid number.")

    def delete_all_books():
        res = messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete all books?")

        if res:
            isbn = table.item(selected_item)["values"][1]
            deletion_result = remove_books(isbn, delete_all=True)

            if deletion_result == "All books with ISBN removed successfully":
                messagebox.showinfo("Success", "All books deleted successfully!")
                popup.destroy()
                refresh_table_callback()
            else:
                messagebox.showerror("Error", deletion_result)

    ttk.Button(form_frame, text="Delete SKU", command=delete_book_by_sku, style="crimson.TButton").grid(row=2, columnspan=2, pady=10)
    ttk.Button(form_frame, text="Delete All", command=delete_all_books, style="crimson.TButton").grid(row=3, columnspan=2, pady=10)

    ttk.Label(form_frame, text="This action cannot be undone.", font=("Calibri", 10, "italic")).grid(row=4, columnspan=2, pady=10)

def open_download_barcodes_popup(app, table):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No book selected!")
        return

    selected_book = table.item(selected_item)["values"]
    if not selected_book:
        messagebox.showerror("Error", "Unable to retrieve selected book data!")
        return

    book = {
        "ISBN": selected_book[0],
        "Title": str(selected_book[1]),
        "SKU": {sku: "date" for sku in selected_book[6].split(",")}
    }

    confirm = messagebox.askyesno("Confirmation", f"Do you want to download barcodes for '{book['Title']}'?")
    if not confirm:
        return

    save_file = filedialog.asksaveasfilename(
        title="Save Barcode PDF",
        defaultextension=".pdf",
        filetypes=[("PDF Files", "*.pdf")],
        initialfile=f"{book['Title'].replace(' ', '_')}_barcodes.pdf"
    )

    if save_file:
        result = download_barcodes(book, save_path=save_file)
    else:
        result = download_barcodes(book)
    if result.startswith("Success"):
        messagebox.showinfo("Success", result)
    else:
        messagebox.showerror("Error", result)
```

admin/member_popup.py

```python
import ttkbootstrap as ttk
from tkinter import StringVar, messagebox
import re
from backend.members import add_member, remove_member, update_member, update_member_details

from backend.utils import open_barcode_scanner

def is_valid_email(email):
    email_regex = r'^[a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
    return re.match(email_regex, email) is not None

def open_add_member_popup(app, refresh_table_callback):
    popup = ttk.Toplevel(app)
    popup.geometry("600x400")
    popup.title("Add Member")
    popup.configure(bg="white")

    form_frame = ttk.Frame(popup)
    form_frame.pack(fill="both", expand=True)

    ttk.Label(form_frame, text="Add New Member", font=("Cambria", 18, "bold")).pack(pady=20)

    name_var = StringVar()
    email_var = StringVar()
    password_var = StringVar()

    fields = [
        ("Full Name", name_var),
        ("Email Address", email_var),
    ]

    for label, var in fields:
        ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).pack(pady=5)
        ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).pack(pady=5)

    ttk.Label(form_frame, text="Password", font=("Century Gothic", 8)).pack(pady=5)
    ttk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 10), show="*").pack(pady=5)

    def handle_add_member():
        name = name_var.get()
        email = email_var.get()
        password = password_var.get()

        if not is_valid_email(email):
            messagebox.showerror("Error", "Please enter a valid email address.")
            return

        try:
            res = add_member(
                Name=name,
                Email=email,
                Password=password,
            )
            if "Error:" in res:
                messagebox.showerror("Error", res)
            else:
                messagebox.showinfo("Success", "Member added successfully!")
            refresh_table_callback()
            popup.destroy()
        except ValueError:
            messagebox.showerror("Error", "Please enter valid data for Quantity.")

    ttk.Button(form_frame, text="Add Member", command=handle_add_member, style="crimson.TButton").pack(pady=10)

    ttk.Label(form_frame, text="Fill the details and click 'Add Member' to create new membership.", font=("Calibri", 10, "italic")).pack(pady=10)

def update_member_popup(app, member_data, refresh_table_callback):
    popup = ttk.Toplevel(app)
    popup.geometry("600x400")
    popup.title("Update Member Details")
    popup.configure(bg="white")

    form_frame = ttk.Frame(popup)
    form_frame.pack(fill="both", expand=True)

    ttk.Label(form_frame, text="Update Member Details", font=("Cambria", 18, "bold")).pack(pady=20)

    name_var = StringVar(value=member_data.get("Name", ""))
    old_password_var = StringVar()
    email_var = StringVar(value=member_data.get("Email", ""))
    new_password_var = StringVar()

    fields = [
        ("Full Name", name_var),
        ("Email Address", email_var),
    ]

    for label, var in fields:
        ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).pack(pady=5)
        ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).pack(pady=5)

    ttk.Label(form_frame, text="Old Password", font=("Century Gothic", 8)).pack(pady=5)
    ttk.Entry(form_frame, textvariable=old_password_var, font=("Century Gothic", 10), show="*").pack(pady=5)
    ttk.Label(form_frame, text="New Password", font=("Century Gothic", 8)).pack(pady=5)
    ttk.Entry(form_frame, textvariable=new_password_var, font=("Century Gothic", 10), show="*").pack(pady=5)

    def save_changes():
        updated_data = {label: var.get() for label, var in fields}
        updated_data["UID"] = member_data.get("UID", "")
        updated_data["New Password"] = new_password_var.get().strip()
        updated_data["Old Password"] = old_password_var.get().strip()

        if 'UID' not in updated_data or not updated_data["UID"]:
            messagebox.showerror("Error", "UID is required.")
            return

        if not is_valid_email(updated_data["Email Address"]):
            messagebox.showerror("Error", "Please enter a valid email address.")
            return
```

admin/member_popup.py

```
        result = update_member_details(
            updated_data["UID"],
            updated_data["Full Name"],
            updated_data["Email Address"],
            updated_data["New Password"],
            updated_data["Old Password"],
        )
        if result == "Member details updated successfully":
            messagebox.showinfo("Success", result)
            refresh_table_callback()
            popup.destroy()
        elif result == "Old password is incorrect":
            messagebox.showerror("Error", "The old password you entered is incorrect. Please try again.")
        elif result == "No member found":
            messagebox.showerror("Error", "No member found with the given UID.")
        else:
            messagebox.showerror("Error", result+".")
        refresh_table_callback()
        popup.destroy()

    ttk.Button(form_frame, text="Save Changes", command=save_changes, style="crimson.TButton").pack(pady=10)

    ttk.Label(form_frame, text="Note: If you do not want to change the password, enter old password as new password.", font=("Calibri", 10, "italic")).pack(pady=10)
    ttk.Label(form_frame, text="Modify the details and click 'Save Changes' to update the member's details.", font=("Calibri", 10, "italic")).pack(pady=10)

def open_delete_member_popup(app, table, refresh_table_callback):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No member selected!")
        return
    res = messagebox.askyesno("Confirm Deletion", "Are you sure you want to cancel membership?")
    if res:
        uid = table.item(selected_item)["values"][0]
        deletion_result = remove_member(uid)

        if deletion_result == "Member removed successfully":
            messagebox.showinfo("Success", "Membership cancelled successfully!")
            refresh_table_callback()
        else:
            messagebox.showerror("Error", deletion_result)


def open_update_member_book_popup(app, table, addbook, refresh_table_callback):
    selected_item = table.selection()
    if not selected_item:
        messagebox.showerror("Error", "No member selected!")
        return

    popup = ttk.Toplevel(app)
    popup.geometry("600x400")
    popup.title("Borrow or Return Book")
    popup.configure(bg="white")

    form_frame = ttk.Frame(popup)
    form_frame.pack(fill="both", expand=True)

    ttk.Label(form_frame, text="Borrow or Return Book", font=("Cambria", 18, "bold")).pack(pady=20)

    uid = table.item(selected_item)["values"][0]

    sku_var = StringVar()
    ttk.Label(form_frame, text="Enter SKU of Book", font=("Century Gothic", 10)).pack(pady=10)
    sku_entry = ttk.Entry(form_frame, textvariable=sku_var, font=("Century Gothic", 10))
    sku_entry.pack(pady=10)

    def handle_action():
        sku = sku_var.get()
        if not sku:
            messagebox.showerror("Error", "Please enter the SKU of the book!")
            return

        result = update_member(uid, sku, addbook)

        if "Fine incurred" in result:
            fine_confirmation = messagebox.askyesno(
                "Fine Confirmation",
                f"{result}\n\nHas the fine been paid?"
            )
            if fine_confirmation:
                result = update_member(uid, sku, addbook, fine_paid=True)
                messagebox.showinfo("Success", result)
            else:
                messagebox.showinfo("Info", "The book will remain with member until the fine is paid.")

        elif "successfully" in result:
            messagebox.showinfo("Success", result)

        else:
            messagebox.showerror("Error", result)

        refresh_table_callback()
        popup.destroy()

    def start_scanning():
        open_barcode_scanner(sku_var)

    ttk.Button(form_frame, text="Scan Barcode", command=start_scanning, style="crimson.TButton").pack(pady=5)
    ttk.Button(form_frame, text="Submit", command=handle_action, style="crimson.TButton").pack(pady=20)

    ttk.Label(form_frame, text="Fill in the SKU and click Submit to either borrow or return the book.", font=("Calibri", 10, "italic")).pack(pady=10)

    def on_close():
        popup.destroy()

    popup.protocol("WM_DELETE_WINDOW", on_close)
```

Member UI

client.py

```python
from tkinter import messagebox
import ttkbootstrap as ttk
from backend.utils import BOOKS_FILE, MEMBERS_FILE, save_data
from client.login_screen import login_screen
from constants import APP_NAME
from backend.books import Books
from backend.members import Members

def on_close():
    if messagebox.askyesno("Exit", "Are you sure you want to exit?"):
        app.destroy()

app = ttk.Window(themename="darkly")
app.geometry("1200x800")
app.state('zoomed')
app.title("LibPro | A Library App")

app.protocol("WM_DELETE_WINDOW", on_close)

style = ttk.Style()
style.configure("crimson.TButton",
                background="#dc143c",
                foreground="white",
                borderwidth=0,
                focusthickness=0)

login_screen(app)
app.mainloop()
```

client/login_screen.py

```python
import ttkbootstrap as ttk
from tkinter import messagebox
import tkinter as tk

from backend.members import sign_in
from client.dashboard import welcome_screen

def login_screen(app):
    global login_frame, email_var, password_var

    login_frame = ttk.Frame(app, padding=30)
    login_frame.pack(expand=True, fill="both")

    branding_frame = ttk.Frame(login_frame, padding=20)
    branding_frame.pack(side="left", fill="both", expand=True, padx=20, pady=20)

    ttk.Label(branding_frame, anchor="center").pack(pady=70)

    ttk.Label(branding_frame, text="LibAmma", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)

    ttk.Label(branding_frame, text="A Library App.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)

    ttk.Label(branding_frame, text="By Pratham, Thejas and Thoshit.", font=("Arial", 12), anchor="center").pack(pady=50)

    login_section = ttk.Frame(login_frame, padding=20)
    login_section.pack(side="right", fill="both", expand=True, padx=20, pady=20)

    canvas = tk.Canvas(login_section, highlightthickness=0)
    canvas.pack(fill="both", expand=True)

    width = 500
    height = 500
    radius = 20
    canvas.create_oval(0, 0, radius * 2, radius * 2, outline="#ffffff")
    canvas.create_rectangle(radius, 0, width - radius, height, outline="#ffffff")
    canvas.create_oval(width - radius * 2, 0, width, radius * 2, outline="#ffffff")
    canvas.create_rectangle(0, radius, width, height - radius, outline="#ffffff")
    canvas.create_oval(0, height - radius * 2, radius * 2, height, outline="#ffffff")
    canvas.create_oval(width - radius * 2, height - radius * 2, width, height, outline="#ffffff")

    form_frame = ttk.Frame(canvas)
    canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")

    ttk.Label(form_frame, text="Member Login", font=("Cambria", 30, "bold")).pack(pady=20)

    ttk.Label(form_frame, text="Email Address:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
    email_var = ttk.StringVar()
    ttk.Entry(form_frame, textvariable=email_var, font=("Century Gothic", 12)).pack(fill="x", pady=5)

    ttk.Label(form_frame, text="Password:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
    password_var = ttk.StringVar()
    ttk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 12), show="*").pack(fill="x", pady=5)

    ttk.Button(form_frame, text="Login", command=lambda: validate_login(app), style="crimson.TButton").pack(pady=20, fill="x")

    ttk.Label(form_frame, text="Welcome to the modern library experience.", font=("Calibri", 10, "italic")).pack(side="bottom", pady=10)


def validate_login(app):
    email = email_var.get().strip()
    password = password_var.get().strip()

    if not email or not password:
        messagebox.showerror("Error", "Email and password cannot be empty.")
        return
    res = sign_in(email, password)
    if res == "Invalid credentials" or res == "Valid email is required" or res == "Password is required":
        messagebox.showerror("Error", res)
    else:
        messagebox.showinfo("Success", "Login successful!")
        login_frame.pack_forget()
        welcome_screen(app, res)
```

client/dashboard.py

```python
import time
from tkinter import messagebox
import ttkbootstrap as ttk

from client.view_books import view_books
from client.view_borrowed_books import view_borrowed_books
from client.wishlist import wishlist

def update_time(label):
    current_time = time.strftime("%H:%M:%S")
    current_date = time.strftime("%A, %B %d, %Y")
    label.config(text=f"Current Time: {current_time}\n{current_date}")
    label.after(1000, update_time, label)

def logout(app):
    res = messagebox.askyesno("Confirm Logout", "Are you sure you want to logout?")
    if res:
        from client.login_screen import login_screen
        welcome_frame.pack_forget()
        login_screen(app)


def welcome_screen(app, member):
    def view_profile():
        messagebox.showinfo("Your Profile", f"""Name: {member["Name"]}\n
        Email ID: {member["Email"]}\nBooks Taken(SKUs): {member["SKU"]}\n
        Wishlist(ISBNs): {member["Wishlist"]}\n\nUnique ID: {member["UID"]}\n
        Membership Since: {member["JoinedOn"]}""")
    global welcome_frame
    if not app:
        messagebox.showerror("Error", "Application instance not found.")
        return
    welcome_frame = ttk.Frame(app, padding=30)
    welcome_frame.pack(expand=True, fill="both", pady=100)

    form_frame = ttk.Frame(welcome_frame)
    form_frame.pack(fill="both", expand=True)

    left_frame = ttk.Frame(form_frame, padding=20)
    left_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")

    ttk.Label(left_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
    ttk.Label(left_frame, text="A Library App.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)

    time_label = ttk.Label(left_frame, font=("Arial", 12))
    time_label.pack(pady=10)
    update_time(time_label)

    greeting_label = ttk.Label(left_frame, text=f"Welcome {member['Name']}!", font=("Arial", 10, "bold"))
    greeting_label.pack(pady=10)

    right_frame = ttk.Frame(form_frame, padding=20)
    right_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")

    borrowed_books_button = ttk.Button(
        right_frame,
        text="View Borrowed Books",
        command=lambda: view_borrowed_books(app, member),
        style="crimson.TButton"
    )
    borrowed_books_button.pack(pady=10, fill="x")

    view_books_button = ttk.Button(
        right_frame,
        text="View Books",
        command=lambda: view_books(app, member),
        style="crimson.TButton"
    )
    view_books_button.pack(pady=10, fill="x")

    membership_management_button = ttk.Button(
        right_frame,
        text="View Wishlist",
        command=lambda: wishlist(app, member),
        style="crimson.TButton"
    )
    membership_management_button.pack(pady=10, fill="x")

    profile_management_button = ttk.Button(
        right_frame,
        text="View Your Profile",
        command=lambda: view_profile(),
        style="crimson.TButton"
    )
    profile_management_button.pack(pady=10, fill="x")

    logout_button = ttk.Button(
        right_frame,
        text="Logout",
        command=lambda: logout(app),
        style="crimson.TButton"
    )
    logout_button.pack(pady=50, fill="both")

    form_frame.grid_columnconfigure(0, weight=1)
    form_frame.grid_columnconfigure(1, weight=1)
    form_frame.grid_rowconfigure(0, weight=1)
```

client/view_books.py

```python
from tkinter import messagebox
import ttkbootstrap as ttk
from backend.books import Books
from backend.members import manage_wishlist
from client import dashboard

ACCENT_COLOR = "#dc143c"


def add_to_wishlist(uid, isbn):
    res = manage_wishlist(True,uid,isbn)
    if "Error:" in res:
        messagebox.showerror("Error", res)
    else:
        messagebox.showinfo("Wishlist", f"{isbn} added to wishlist successfully!")

def view_books(app, member):
    def show_main_page():
        def go_to_dashboard():
            main_frame.pack_forget()
            dashboard.welcome_screen(app, member)

        for widget in app.winfo_children():
            widget.destroy()

        main_frame = ttk.Frame(app, padding=30)
        main_frame.pack(fill="both", expand=True)

        left_panel = ttk.Frame(main_frame, padding=20)
        left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)

        ttk.Label(left_panel, text="Available Books", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
        ttk.Label(left_panel, text="List of Books available in this library.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
        back_button = ttk.Button(
            left_panel,
            text="Back to Dashboard",
            command=go_to_dashboard,
            style="crimson.TButton",
        )
        back_button.pack(pady=60)

        right_panel = ttk.Frame(main_frame, padding=20)
        right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)

        canvas = ttk.Canvas(right_panel, highlightthickness=0)
        scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
        )
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        style = ttk.Style()
        style.configure('hover.TFrame', background='#f0f0f0', borderwidth=0)

        if len(Books)>0:
            for book in Books:
                book_frame = ttk.Frame(scrollable_frame, padding=10, style="default.TFrame")
                book_frame.pack(fill="x", pady=(0, 1))

                def on_hover(e, frame=book_frame):
                    frame.configure(style="hover.TFrame")

                def on_leave(e, frame=book_frame):
                    frame.configure(style="default.TFrame")

                book_frame.bind("<Enter>", on_hover)
                book_frame.bind("<Leave>", on_leave)
                title_label = ttk.Label(
                    book_frame,
                    text=book["Title"],
                    font=("Helvetica", 14, "bold"),
                    foreground=ACCENT_COLOR,
                    anchor="w"
                )
                title_label.pack(anchor="w", padx=5)
                title_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))

                isbn_label = ttk.Label(
                    book_frame,
                    text=f"ISBN: {book['ISBN']}",
                    font=("Helvetica", 10),
                    anchor="w"
                )
                isbn_label.pack(anchor="w", padx=5)
                isbn_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))

                book_frame.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
```

client/view_books.py

```
        else:
            ttk.Label(
                scrollable_frame,
                text="No books found.",
                font=("Helvetica", 14, "bold"),
            ).pack(anchor="w", padx=5)



    def show_details_page(isbn):
        for widget in app.winfo_children():
            widget.destroy()

        details_frame = ttk.Frame(app, padding=30)
        details_frame.pack(fill="both", expand=True)

        left_panel = ttk.Frame(details_frame, padding=20)
        left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)

        ttk.Label(
            left_panel,
            text="Book Details",
            font=("Century Gothic", 40, "bold"),
            anchor="center",
        ).pack(pady=10)

        ttk.Label(
            left_panel,
            text="Detailed Information about the selected book.",
            font=("Arial", 18, "italic"),
            anchor="center",
        ).pack(pady=5)

        right_panel = ttk.Frame(details_frame, padding=20)
        right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)

        canvas = ttk.Canvas(right_panel, highlightthickness=0)
        scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
        )
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        book = next((b for b in Books if b["ISBN"] == isbn), None)
        if not book:
            messagebox.showerror("Error", "Book not found.")
            show_main_page()
            return

        book_details = [
            ("Title", book["Title"]),
            ("ISBN", book["ISBN"]),
            ("Description", book["Description"]),
            ("Category", book["Category"]),
            ("Quantity", book["Quantity"]),
            ("Author", book["Author"]),
            ("Publisher", book["Publisher"]),
            ("Language", book["Language"]),
        ]

        for label, adetail in book_details:
            detail_frame = ttk.Frame(scrollable_frame, padding=10)
            detail_frame.pack(fill="x", pady=(0, 1))

            detail = adetail

            if label == "Quantity":
                if int(detail) == 0:
                    detail = "All copies have been borrowed."
                else:
                    detail = f"{detail} copies in library."

            ttk.Label(
                detail_frame,
                text=f"{label}: {detail}",
                font=("Helvetica", 12),
                anchor="w",
                wraplength=350
            ).pack(anchor="w", padx=5)

        wishlist_button = ttk.Button(
            scrollable_frame,
            text="Add to Wishlist",
            command=lambda:add_to_wishlist(member["UID"], isbn),
            style="crimson.TButton",
        )
        wishlist_button.pack(pady=20)

        back_button = ttk.Button(
            scrollable_frame,
            text="Back",
            command=show_main_page,
            style="crimson.TButton",
        )
        back_button.pack(pady=30)

    show_main_page()
```

client/view_borrowed_books.py

```python
from datetime import datetime
import ttkbootstrap as ttk
from backend.books import read_book
from client import dashboard

ACCENT_COLOR = "#dc143c"

def view_borrowed_books(app, member):
    def go_to_dashboard():
        main_frame.pack_forget()
        dashboard.welcome_screen(app, member)

    for widget in app.winfo_children():
        widget.destroy()

    main_frame = ttk.Frame(app, padding=30)
    main_frame.pack(fill="both", expand=True)

    left_panel = ttk.Frame(main_frame, padding=20)
    left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)

    ttk.Label(left_panel, text="Borrowed Books", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
    ttk.Label(left_panel, text="List of Books You have borrowed from the library.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
    back_button = ttk.Button(
        left_panel,
        text="Back to Dashboard",
        command=go_to_dashboard,
        style="crimson.TButton",
    )
    back_button.pack(pady=60)

    right_panel = ttk.Frame(main_frame, padding=20)
    right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)

    canvas = ttk.Canvas(right_panel, highlightthickness=0)
    scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
    scrollable_frame = ttk.Frame(canvas)

    scrollable_frame.bind(
        "<Configure>",
        lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
    )
    canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
    canvas.configure(yscrollcommand=scrollbar.set)

    canvas.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    style = ttk.Style()
    style.configure('hover.TFrame', background='#f0f0f0', borderwidth=0)

    if len(member["SKU"]) > 0:
        for sku, duedate in member["SKU"].items():
            days_late = (datetime.now() - datetime.strptime(duedate, "%d/%m/%Y %H:%M:%S")).days
            day_text = "days late" if days_late > 0 else "days left"

            isbn = str(sku).split("-")[0].strip()
            book = read_book(isbn)

            book_frame = ttk.Frame(scrollable_frame, padding=10)
            book_frame.pack(fill="x", pady=(0, 1))

            def on_hover(e, frame=book_frame):
                frame.configure(style="hover.TFrame")

            def on_leave(e, frame=book_frame):
                frame.configure(style="default.TFrame")

            book_frame.bind("<Enter>", on_hover)
            book_frame.bind("<Leave>", on_leave)

            ttk.Label(
                book_frame,
                text=book["Title"],
                font=("Helvetica", 14, "bold"),
                foreground=ACCENT_COLOR,
            ).pack(anchor="w", padx=5)

            ttk.Label(
                book_frame,
                text=f"Status: {abs(days_late)} {day_text}",
                font=("Helvetica", 10),
            ).pack(anchor="w", padx=5)
    else:
        ttk.Label(
            scrollable_frame,
            text="No book borrowed.",
            font=("Helvetica", 14, "bold"),
        ).pack(anchor="w", padx=5)
```

client/wishlist.py

```python
from tkinter import messagebox
import ttkbootstrap as ttk
from backend.books import read_book
from backend.members import manage_wishlist
from client import dashboard

ACCENT_COLOR = "#dc143c"


def remove_from_wishlist(uid, isbn):
    res = manage_wishlist(False,uid,isbn)
    if "Error:" in res:
        messagebox.showerror("Error", res)
    else:
        messagebox.showinfo("Wishlist", f"{isbn} removed from wishlist successfully!")
        show_main_page()

def wishlist(app, member):
    global show_main_page
    def show_main_page():
        def go_to_dashboard():
            main_frame.pack_forget()
            dashboard.welcome_screen(app, member)

        for widget in app.winfo_children():
            widget.destroy()

        main_frame = ttk.Frame(app, padding=30)
        main_frame.pack(fill="both", expand=True)

        left_panel = ttk.Frame(main_frame, padding=20)
        left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)

        ttk.Label(left_panel, text="Your Wishlist", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
        ttk.Label(left_panel, text="Your favourite/saved books!", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
        back_button = ttk.Button(
            left_panel,
            text="Back to Dashboard",
            command=go_to_dashboard,
            style="crimson.TButton",
        )
        back_button.pack(pady=60)

        right_panel = ttk.Frame(main_frame, padding=20)
        right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)

        canvas = ttk.Canvas(right_panel, highlightthickness=0)
        scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
        )
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        style = ttk.Style()
        style.configure('hover.TFrame', background='#f0f0f0', borderwidth=0)


        if len(member["Wishlist"])>0:
            for isbn in member["Wishlist"]:
                book = read_book(isbn)
                book_frame = ttk.Frame(scrollable_frame, padding=10, style="default.TFrame")
                book_frame.pack(fill="x", pady=(0, 1))

                def on_hover(e, frame=book_frame):
                    frame.configure(style="hover.TFrame")

                def on_leave(e, frame=book_frame):
                    frame.configure(style="default.TFrame")

                book_frame.bind("<Enter>", on_hover)
                book_frame.bind("<Leave>", on_leave)


                wishlist_label = ttk.Label(
                    book_frame,
                    text=book["Title"],
                    font=("Helvetica", 14, "bold"),
                    foreground=ACCENT_COLOR,
                    anchor="w"
                )
                wishlist_label.pack(anchor="w", padx=5)
                wishlist_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))

                isbn_label = ttk.Label(
                    book_frame,
                    text=f"ISBN: {book['ISBN']}",
                    font=("Helvetica", 10),
                    anchor="w"
                )
                isbn_label.pack(anchor="w", padx=5)
                isbn_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))

                book_frame.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
```

client/wishlist.py

```python
else:
            ttk.Label(
                scrollable_frame,
                text="No book added to wishlist.",
                font=("Helvetica", 14, "bold"),
            ).pack(anchor="w", padx=5)


    def show_details_page(isbn):
        for widget in app.winfo_children():
            widget.destroy()

        details_frame = ttk.Frame(app, padding=30)
        details_frame.pack(fill="both", expand=True)

        left_panel = ttk.Frame(details_frame, padding=20)
        left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)

        ttk.Label(
            left_panel,
            text="Book Details",
            font=("Century Gothic", 40, "bold"),
            anchor="center",
        ).pack(pady=10)

        ttk.Label(
            left_panel,
            text="Detailed Information about this wishlisted book.",
            font=("Arial", 18, "italic"),
            anchor="center",
        ).pack(pady=5)

        right_panel = ttk.Frame(details_frame, padding=20)
        right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)

        canvas = ttk.Canvas(right_panel, highlightthickness=0)
        scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
        scrollable_frame = ttk.Frame(canvas)

        scrollable_frame.bind(
            "<Configure>",
            lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
        )
        canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
        canvas.configure(yscrollcommand=scrollbar.set)

        canvas.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")

        isbn = next((b for b in member["Wishlist"] if b == isbn), None)
        if not isbn:
            messagebox.showerror("Error", "Book not found.")
            show_main_page()
            return

        book = read_book(isbn)
        book_details = [
            ("Title", book["Title"]),
            ("ISBN", book["ISBN"]),
            ("Description", book["Description"]),
            ("Category", book["Category"]),
            ("Quantity", book["Quantity"]),
            ("Author", book["Author"]),
            ("Publisher", book["Publisher"]),
            ("Language", book["Language"]),
        ]

        for label, adetail in book_details:
            detail_frame = ttk.Frame(scrollable_frame, padding=10)
            detail_frame.pack(fill="x", pady=(0, 1))

            detail = adetail

            if label == "Quantity":
                if int(detail) == 0:
                    detail = "All copies have been borrowed."
                else:
                    detail = f"{detail} copies in library."

            ttk.Label(
                detail_frame,
                text=f"{label}: {detail}",
                font=("Helvetica", 12),
                anchor="w",
                wraplength=350
            ).pack(anchor="w", padx=5)

        wishlist_button = ttk.Button(
            scrollable_frame,
            text="Remove from Wishlist",
            command=lambda:remove_from_wishlist(member["UID"], isbn),
            style="crimson.TButton",
        )
        wishlist_button.pack(pady=20)

        back_button = ttk.Button(
            scrollable_frame,
            text="Back",
            command=show_main_page,
            style="crimson.TButton",
        )
        back_button.pack(pady=30)

    show_main_page()
```

Backend

backend/books.py

```python
from datetime import datetime
import os
import shutil
from barcode import Code128
from barcode.writer import ImageWriter
from reportlab.lib.pagesizes import letter
from reportlab.pdfgen import canvas


Books = []

def is_valid_isbn(isbn):
    return len(isbn) == 10 or len(isbn) == 13 and isbn.isdigit()

def add_book(ISBN, Title, Description, Category, Quantity, Author, Publisher, Language, READD=False, SKU=None):
    if not ISBN or int(Quantity) <= 0 or not is_valid_isbn(ISBN):
        return "Invalid input for ISBN (must be 10 or 13 digits) or Quantity."

    if READD:
        for book in Books:
            if book["ISBN"] == str(SKU).split("-")[0]:
                book["Quantity"] += 1
                date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")

                if not isinstance(book.get("SKU"), dict):
                    book["SKU"] = {}

                book["SKU"].setdefault(SKU, date)
                return "Book quantity updated successfully"

        sku_dict = {SKU: datetime.now().strftime("%d/%m/%Y %H:%M:%S")}
        Books.append({
            "ISBN": str(SKU).split("-")[0],
            "Title": Title,
            "Description": Description,
            "Category": Category,
            "Quantity": 1,
            "SKU": sku_dict,
            "Author": Author,
            "Publisher": Publisher,
            "Language": Language,
        })
        return "Book re-added as a new entry"

    for book in Books:
        if book["ISBN"] == ISBN:
            for i in range(int(Quantity)):
                sku = f"{ISBN}-{len(book['SKU']) + i + 1}"
                date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
                book["SKU"].setdefault(sku, date)
            book["Quantity"] += int(Quantity)
            return "Book quantity updated successfully"

    sku_dict = {}
    for i in range(int(Quantity)):
        sku = f"{ISBN}-{i + 1}"
        date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
        sku_dict[sku] = date

    Books.append({
        "ISBN": ISBN,
        "Title": Title,
        "Description": Description,
        "Category": Category,
        "Quantity": int(Quantity),
        "SKU": sku_dict,
        "Author": Author,
        "Publisher": Publisher,
        "Language": Language,
    })
    return "Book added successfully"

def generate_barcodes_and_pdf(barcodes):
    output_dir = "barcodes"
    pdf_filename = "barcodes.pdf"

    if not os.path.exists(output_dir):
        os.makedirs(output_dir)

    barcode_images = []
    for barcode_data in barcodes:
        barcode = Code128(barcode_data, writer=ImageWriter())
        image_path = os.path.abspath(os.path.join(output_dir, f"{barcode_data}.png"))
        try:
            barcode.save(image_path[:-4])
            barcode_images.append(image_path)
        except Exception as e:
            return f"Error saving barcode {barcode_data}: {e}"

    if not barcode_images:
        return "No barcodes generated."
```

backend/books.py

```python
    try:
        c = canvas.Canvas(pdf_filename, pagesize=letter)
        x, y = 50, 750

        for image in barcode_images:
            if not os.path.exists(image):
                continue
            try:
                c.drawImage(image, x, y, width=200, height=50)
                y -= 100
                if y < 100:
                    c.showPage()
                    y = 750
            except Exception as e:
                return f"Error adding image {image} to PDF: {e}"

        c.save()
    except Exception as e:
        return f"Error creating PDF: {e}"

    try:
        shutil.rmtree(output_dir)
    except Exception as e:
        return "PDF created but error deleting temporary folder."

    return f"Barcodes PDF saved as {os.path.abspath(pdf_filename)}"

def download_barcodes(book, save_path=None):
    if not book or "SKU" not in book:
        return "Error: Invalid book selected or no SKUs found."

    barcodes = list(book["SKU"].keys())
    if not barcodes:
        return "Error: No barcodes available for the selected book."

    pdf_result = generate_barcodes_and_pdf(barcodes)
    if pdf_result.startswith("Error"):
        return pdf_result

    if save_path:
        try:
            shutil.move("barcodes.pdf", save_path)
            return f"Success: Barcodes saved at {save_path}"
        except Exception as e:
            return f"Error saving file: {e}"

    return "Success: Barcodes PDF generated but not saved."

def update_books(ISBN, Title, Description, Category, Author, Publisher, Language):
    if not ISBN or not Title or not is_valid_isbn(ISBN):
        return "Invalid input for ISBN (must be 10 or 13 digits) or Title."

    for book in Books:
        if book["ISBN"] == ISBN:
            book["Title"] = Title
            book["Description"] = Description
            book["Category"] = Category
            book["Author"] = Author
            book["Publisher"] = Publisher
            book["Language"] = Language
            return "Book details updated successfully"
    return "No book found"

def remove_books(SKU, delete_all=False):
    if not SKU:
        return "Invalid SKU input."

    ISBN = (str(SKU).split("-")[0]).strip()
    sku = str(SKU).strip()

    for book in Books:
        if book["ISBN"] == ISBN:
            if delete_all:
                book["SKU"].clear()
                book["Quantity"] = 0
                return "All books with ISBN removed successfully"

            if sku in book["SKU"]:
                book["SKU"].pop(sku)
                if int(book["Quantity"]) > 0:
                    book["Quantity"] = int(book["Quantity"]) - 1
                    return "Book removed successfully"
                else:
                    return "Error: Book quantity is already 0, cannot remove further"
            else:
                return f"Error: SKU {sku} not found in the database"

    return f"Error Book with ISBN {ISBN} not found"

def read_book(ISBN):
    if not ISBN or not is_valid_isbn(ISBN):
        return "Invalid ISBN input (must be 10 or 13 digits)."

    for book in Books:
        if book["ISBN"] == ISBN:
            return book
    return "No book found"
```

backend/members.py

```python
import math
import re
from backend.books import add_book, Books
from datetime import datetime, timedelta

Members = []

def is_valid_email(email):
    return re.match(r"[^@]+@[^@]+\.[^@]+", email) is not None

def add_member(Name, Email, Password):
    if not Name.strip():
        return "Error: Name is required"
    if not Email.strip() or not is_valid_email(Email):
        return "Error: Valid email is required"
    if not Password.strip() or len(Password) < 6:
        return "Error: Password must be at least 6 characters"
    Members.append({
        "UID": len(Members) + 1,
        "Name": Name.strip(),
        "Email": Email.strip(),
        "Password": Password.strip(),
        "SKU": {},
        "Wishlist": [],
        "JoinedOn": datetime.now().strftime("%d/%m/%Y %H:%M:%S")
    })
    return "Member added successfully"

def update_member_details(UID, NAME, EMAIL, PASSWORD, OLD_PASSWORD):
    if not UID or not str(UID).isdigit():
        return "Valid UID is required"
    if not NAME.strip():
        return "Name is required"
    if not EMAIL.strip() or not is_valid_email(EMAIL):
        return "Valid email is required"
    if not PASSWORD.strip() or len(PASSWORD) < 6:
        return "Password must be at least 6 characters"
    if not OLD_PASSWORD.strip():
        return "Old password is required"

    for member in Members:
        if str(member["UID"]) == str(UID):
            if str(member["Password"]) == str(OLD_PASSWORD):
                member["Name"] = NAME.strip()
                member["Email"] = EMAIL.strip()
                member["Password"] = PASSWORD.strip()
                return "Member details updated successfully"
            return "Old password is incorrect"
    return "No member found"

def update_member(UID, SKU, ADD_BOOK, fine_paid=False):
    if not UID or not str(UID).isdigit():
        return "Valid UID is required"
    if not SKU.strip():
        return "SKU is required"
    if ADD_BOOK not in [True, False]:
        return "ADD_BOOK must be either True or False"

    future_date = (datetime.now() + timedelta(days=15)).strftime("%d/%m/%Y %H:%M:%S")

    if ADD_BOOK:
        for book in Books:
            if str(book["ISBN"]) == str(SKU).split("-")[0]:
                if SKU in book["SKU"]:
                    for member in Members:
                        if member["UID"] == int(UID):
                            member["SKU"][SKU] = future_date
                            return "Book borrowed successfully"
        return "The book you are searching for is not available"
    else:
        for member in Members:
            if member["UID"] == int(UID):
                if SKU in member["SKU"]:
                    borrow_date = datetime.strptime(member["SKU"][SKU], "%d/%m/%Y %H:%M:%S")
                    days_late = (datetime.now() - borrow_date).days

                    total_fine = 0
```

backend/members.py

```python
                    if days_late > 0:
                        fine = 5
                        for day in range(1, days_late + 1):
                            total_fine += math.ceil(fine)
                            fine += fine * 0.02

                        if not fine_paid:
                            return f"Days Late: {days_late}\nFine incurred: \u20B9{total_fine}/-."

                    member["SKU"].pop(SKU)

                    res = add_book(
                        ISBN=str(SKU).split("-")[0],
                        Title="",
                        Description="",
                        Category="",
                        Quantity=1,
                        Author="",
                        Publisher="",
                        Language="",
                        READD=True,
                        SKU=SKU,
                    )
                    if res == "Invalid input for ISBN, Title, or Quantity.":
                        return "Error: Unable to add book to database."

                    if total_fine > 0 and fine_paid:
                        return f'Book returned successfully. Fine of \u20B9{total_fine}/- was paid.'
                    return "Book returned successfully. No fine incurred."

                return "Book not borrowed by this member"
        return "Member not found"

def remove_member(UID):
    if not UID or not str(UID).isdigit():
        return "Valid UID is required"
    for member in Members:
        if member["UID"] == int(UID):
            Members.remove(member)
            return "Member removed successfully"
    return "Member not found"

def sign_in(Email, Password):
    if not Email.strip() or not is_valid_email(Email):
        return "Valid email is required"
    if not Password.strip():
        return "Password is required"
    for member in Members:
        if member["Email"] == Email.strip() and member["Password"] == Password.strip():
            return member
    return "Invalid credentials"

def read_member(UID):
    if not UID or not str(UID).isdigit():
        return "Valid UID is required"
    for member in Members:
        if member["UID"] == int(UID):
            return member
    return "No member found"

def manage_wishlist(ADD, UID, ISBN):
    if not UID or not str(UID).isdigit() or not ISBN or ADD not in [True, False]:
        return "Valid parameters required"

    for mem in Members:
        if str(mem["UID"]) == str(UID):
            if ADD:
                if ISBN not in mem["Wishlist"]:
                    mem["Wishlist"].append(ISBN)
                else:
                    return "Error: Book already in wishlist"
            else:
                if ISBN in mem["Wishlist"]:
                    mem["Wishlist"].remove(ISBN)
                else:
                    return "Book not present in wishlist"
            return "Success"

    return "Error: UID is Invalid"
```

backend/shelfing.py

```python
from datetime import datetime

BookShelf = []
DeshelvedBooks = {}

def categorise(RACK, SHELF, CATEGORY):
    if not RACK or not SHELF or not CATEGORY:
        return "Error: Invalid input. RACK, SHELF, and CATEGORY are required."

    while len(BookShelf) < RACK:
        BookShelf.append([])

    while len(BookShelf[RACK - 1]) < SHELF:
        BookShelf[RACK - 1].append({})

    category_dict = BookShelf[RACK - 1][SHELF - 1]

    if CATEGORY not in category_dict:
        category_dict[CATEGORY] = []

    return f"Category '{CATEGORY}' added to rack {RACK}, shelf {SHELF}."

def shelf(RACK, SHELF, SKU):
    if not RACK or not SHELF or not SKU:
        return "Error: Invalid input. RACK, SHELF, and SKU are required."

    if RACK > len(BookShelf) or SHELF > len(BookShelf[RACK - 1]):
        return "Error: Rack or shelf does not exist."

    category_dict = BookShelf[RACK - 1][SHELF - 1]

    if not category_dict:
        return "Error: Shelf is uncategorized."

    for category in category_dict:
        if SKU not in category_dict[category]:
            category_dict[category].append(SKU)
            if SKU in DeshelvedBooks:
                DeshelvedBooks.pop(SKU)
            return f"Book with SKU '{SKU}' shelved successfully on rack {RACK}, shelf {SHELF} under category '{category}'."
    return "Error: SKU already exists in this category."

def search(SKU):
    if not SKU:
        return "Error: SKU is required for searching."

    for r, rack in enumerate(BookShelf, start=1):
        for s, shelf in enumerate(rack, start=1):
            if not shelf:
                continue

            for category, skus in shelf.items():
                if SKU in skus:
                    return f"Book with SKU '{SKU}' is located on rack {r}, shelf {s} under category '{category}'."
    return f"Error: Book with SKU '{SKU}' not found."


def deshelve(SKU):
    if not SKU:
        return "Error: SKU is required for deshelving."

    for r, rack in enumerate(BookShelf, start=1):
        for s, shelf in enumerate(rack, start=1):
            if not shelf:
                continue

            for category, skus in shelf.items():
                if SKU in skus:
                    skus.remove(SKU)
                    timestamp = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
                    DeshelvedBooks[SKU] = f"Rack: {r}, Shelf: {s}, Timestamp: {timestamp}"
                    return f"Book with SKU '{SKU}' deshelved from rack {r}, shelf {s}."
    return f"Error: Book with SKU '{SKU}' not found."
```

backend/utils.py

```python
import threading
import numpy
from tkinter import messagebox
import cv2
from pyzbar.pyzbar import decode

def clear_fields(*vars):
    for var in vars:
        var.set("")

def open_barcode_scanner(sku_var):
    sku_var.set(None)
    def get_available_camera():
        for camera_id in range(10):
            cap = cv2.VideoCapture(camera_id)
            if cap.isOpened():
                return cap
            cap.release()
        return None

    cap = get_available_camera()

    if not cap:
        messagebox.showerror("Error", "No available camera found.")
        return

    line_position = 0
    line_direction = 1

    def stop_scanning():
        cap.release()
        cv2.destroyAllWindows()

    def scan_thread():
        nonlocal line_position, line_direction

        while True:
            ret, frame = cap.read()
            if not ret:
                break

            barcodes = decode(frame)

            for barcode in barcodes:
                barcode_data = barcode.data.decode('utf-8').strip()
                sku_var.set(barcode_data)

                rect_points = barcode.polygon
                if len(rect_points) == 4:
                    pts = numpy.array(rect_points, dtype=numpy.int32)
                    cv2.polylines(frame, [pts], True, (0, 0, 255), 2)

                x, y, w, h = barcode.rect
                cv2.putText(frame, barcode_data, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)

                stop_scanning()
                return

            cv2.line(frame, (0, line_position), (frame.shape[1], line_position), (0, 255, 0), 2)
            line_position += line_direction
            if line_position >= frame.shape[0] or line_position <= 0:
                line_direction *= -1

            cv2.imshow("Barcode Scanner", frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        stop_scanning()

    threading.Thread(target=scan_thread, daemon=True).start()

```