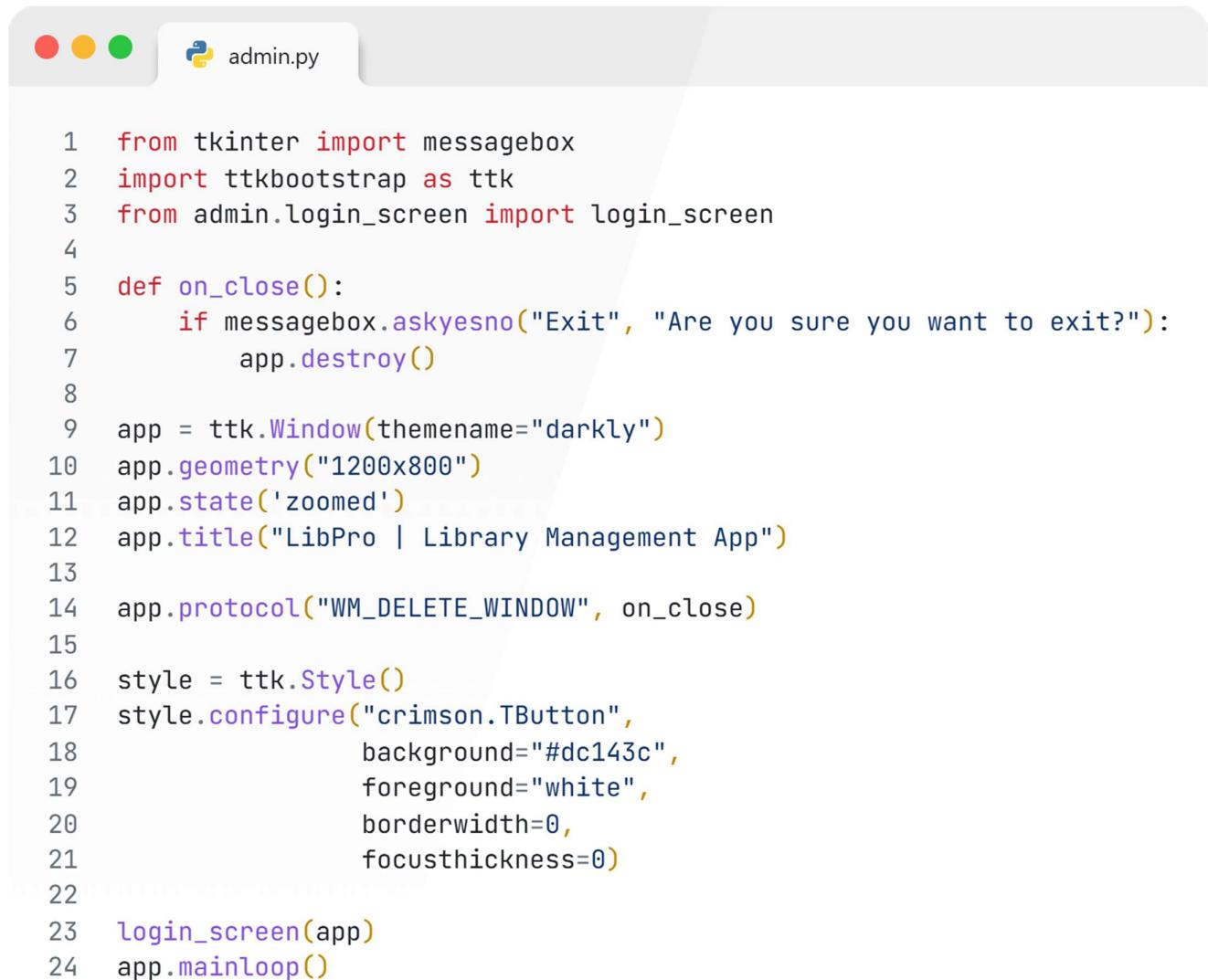


Source Code

Fontend

Admin UI



```
 1  from tkinter import messagebox
 2  import ttkbootstrap as ttk
 3  from admin.login_screen import login_screen
 4
 5  def on_close():
 6      if messagebox.askyesno("Exit", "Are you sure you want to exit?"):
 7          app.destroy()
 8
 9  app = ttk.Window(themename="darkly")
10 app.geometry("1200x800")
11 app.state('zoomed')
12 app.title("LibPro | Library Management App")
13
14 app.protocol("WM_DELETE_WINDOW", on_close)
15
16 style = ttk.Style()
17 style.configure("crimson.TButton",
18                 background="#dc143c",
19                 foreground="white",
20                 borderwidth=0,
21                 focusthickness=0)
22
23 login_screen(app)
24 app.mainloop()
```

```
1 import ttkbootstrap as ttk
2 from tkinter import messagebox
3 import tkinter as tk
4
5 from admin.dashboard import welcome_screen
6
7 ADMIN_CREDENTIALS = {
8     "Pratham": "123",
9     "Thejas": "456",
10    "Thoshit": "789",
11 }
12
13 def login_screen(app):
14     global login_frame, username_var, password_var
15
16     login_frame = ttk.Frame(app, padding=30)
17     login_frame.pack(expand=True, fill="both")
18
19     branding_frame = ttk.Frame(login_frame, padding=20)
20     branding_frame.pack(side="left", fill="both", expand=True, padx=20, pady=20)
21
22     ttk.Label(branding_frame, anchor="center").pack(pady=70)
23
24     ttk.Label(branding_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
25
26     ttk.Label(branding_frame, text="Library Management App", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
27
28     ttk.Label(branding_frame, text="By Pratham, Thejas and Thoshit.", font=("Arial", 12), anchor="center").pack(pady=50)
29
30     login_section = ttk.Frame(login_frame, padding=20)
31     login_section.pack(side="right", fill="both", expand=True, padx=20, pady=20)
32
33     canvas = tk.Canvas(login_section, highlightthickness=0)
34     canvas.pack(fill="both", expand=True)
35
36     width = 500
37     height = 500
38     radius = 20
39     canvas.create_oval(0, 0, radius * 2, radius * 2, outline="#ffffff")
40     canvas.create_rectangle(radius, 0, width - radius, height, outline="#ffffff")
41     canvas.create_oval(width - radius * 2, 0, width, radius * 2, outline="#ffffff")
42     canvas.create_rectangle(0, radius, width, height - radius, outline="#ffffff")
43     canvas.create_oval(0, height - radius * 2, radius * 2, height, outline="#ffffff")
44     canvas.create_oval(width - radius * 2, height - radius * 2, width, height, outline="#ffffff")
45
46     form_frame = ttk.Frame(canvas)
47     canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
48
49     ttk.Label(form_frame, text="Admin Login", font=("Cambria", 30, "bold")).pack(pady=20)
50
51     ttk.Label(form_frame, text="Username:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
52     username_var = tk.StringVar()
53     tk.Entry(form_frame, textvariable=username_var, font=("Century Gothic", 12)).pack(fill="x", pady=5)
54
55     ttk.Label(form_frame, text="Password:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
56     password_var = tk.StringVar()
57     tk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 12), show="*").pack(fill="x", pady=5)
58
59     ttk.Button(form_frame, text="Login", command=lambda: validate_login(app), style="crimson.TButton").pack(pady=20, fill="x")
60
61     ttk.Label(form_frame, text="Welcome to the modern library experience.", font=("Calibri", 10, "italic")).pack(side="bottom", pady=10)
62
63
64 def validate_login(app):
65     username = username_var.get().strip()
66     password = password_var.get().strip()
67
68     if not username or not password:
69         messagebox.showerror("Error", "Username and password cannot be empty.")
70         return
71
72     if username in ADMIN_CREDENTIALS and ADMIN_CREDENTIALS[username] == password:
73         messagebox.showinfo("Success", "Login successful!")
74         login_frame.pack_forget()
75         welcome_screen(app)
76     else:
77         messagebox.showerror("Error", "Invalid username or password.")
78
```

```
1 import time
2 from tkinter import messagebox
3 import ttkbootstrap as ttk
4
5 from admin.shelf_manage import shelf_manage
6 from admin.membership_manage import membership_manage
7 from admin.books_manage import books_manage
8
9 def update_time(label):
10    current_time = time.strftime("%H:%M:%S")
11    current_date = time.strftime("%A, %B %d, %Y")
12    label.config(text=f"Current Time: {current_time}\n{current_date}")
13    label.after(1000, update_time, label)
14
15 def open_book_management(app):
16    if not app:
17        messagebox.showerror("Error", "Application instance not found.")
18        return
19    welcome_frame.pack_forget()
20    books_manage(app)
21
22 def open_rack_management(app):
23    if not app:
24        messagebox.showerror("Error", "Application instance not found.")
25        return
26    welcome_frame.pack_forget()
27    shelf_manage(app)
28
29 def logout(app):
30    res = messagebox.askyesno("Confirm Logout", "Are you sure you want to logout?")
31    if res:
32        from ui.login_screen import login_screen
33        welcome_frame.pack_forget()
34        login_screen(app)
35
36 def open_membership_management(app):
37    if not app:
38        messagebox.showerror("Error", "Application instance not found.")
39        return
40    welcome_frame.pack_forget()
41    membership_manage(app)
42
43 def welcome_screen(app):
44    global welcome_frame
45    if not app:
46        messagebox.showerror("Error", "Application instance not found.")
47        return
48    welcome_frame = ttk.Frame(app, padding=30)
49    welcome_frame.pack(expand=True, fill="both", pady=100)
50
51    form_frame = ttk.Frame(welcome_frame)
52    form_frame.pack(fill="both", expand=True)
53
54    left_frame = ttk.Frame(form_frame, padding=20)
55    left_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
56
57    ttk.Label(left_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
58    ttk.Label(left_frame, text="Library Management App", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
59
60    time_label = ttk.Label(left_frame, font=("Arial", 12))
61    time_label.pack(pady=10)
62    update_time(time_label)
63
64    greeting_label = ttk.Label(left_frame, text="Welcome Admin!", font=("Arial", 10, "bold"))
65    greeting_label.pack(pady=10)
66
67    right_frame = ttk.Frame(form_frame, padding=20)
68    right_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")
69
70    book_management_button = ttk.Button(right_frame, text="Books Management", command=lambda: open_book_management(app), style="crimson.TButton")
71    book_management_button.pack(pady=10, fill="x")
72
73    book_rack_management_button = ttk.Button(right_frame, text="Book Rack Management", command=lambda: open_rack_management(app), style="crimson.TButton")
74    book_rack_management_button.pack(pady=10, fill="x")
75
76    membership_management_button = ttk.Button(right_frame, text="Membership Management", command=lambda: open_membership_management(app), style="crimson.TButton")
77    membership_management_button.pack(pady=10, fill="x")
78
79    logout_button = ttk.Button(right_frame, text="Logout", command=lambda: logout(app), style="crimson.TButton")
80    logout_button.pack(pady=50, fill="both")
81
82    form_frame.grid_columnconfigure(0, weight=1)
83    form_frame.grid_columnconfigure(1, weight=1)
84    form_frame.grid_rowconfigure(0, weight=1)
85
```



The screenshot shows a Python code editor with a file named `admin/books_manage.py`. The code is a Tkinter application for managing books. It includes functions for displaying books, opening update book popups, and updating the main table. The code uses global variables for the table and its frame, and it handles various book details like ISBN, Title, Description, Category, Quantity, SKU, Author, Language, and Publisher.

```
1 from tkinter import ttk, messagebox
2 from admin.book_popups import open_add_book_popup, open_download_barcodes_popup, update_book_popup, open_delete_book_popup
3 from backend.books import Books, read_book
4
5 def back_to_dashboard(app):
6     from admin.dashboard import welcome_screen
7
8     if not app:
9         messagebox.showerror("Error", "Application instance not found.")
10    return
11
12    for widget in app.winfo_children():
13        widget.grid_forget()
14
15    welcome_screen(app)
16
17 def books_manage(app):
18     global table, table_frame
19
20     if not app:
21         messagebox.showerror("Error", "Application instance not found.")
22         return
23
24     app.grid_columnconfigure(0, weight=1)
25     app.grid_rowconfigure(0, weight=1)
26
27     table_frame = ttk.Frame(app)
28     table_frame.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
29     table_frame.grid_columnconfigure(0, weight=1)
30
31     columns = ("ID", "ISBN", "Title", "Description", "Category", "Quantity", "SKU", "Author", "Language", "Publisher")
32     table = ttk.Treeview(table_frame, columns=columns, show="headings", height=40)
33
34     for col in columns:
35         table.heading(col, text=col)
36         table.column(col, width=100, anchor="center", stretch=True)
37
38     scrollbar = ttk.Scrollbar(table_frame, orient="horizontal", command=table.xview)
39     table.configure(xscrollcommand=scrollbar.set)
40     scrollbar.grid(row=1, column=0, sticky="ew")
41
42     table.grid(row=0, column=0, padx=20, pady=10, sticky="nsew")
43
44     button_frame = ttk.Frame(app)
45     button_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
46     button_frame.grid_columnconfigure(0, weight=1)
47
48     ttk.Button(button_frame, text="Add Book", command=lambda: open_add_book_popup(app, display_books), style="crimson.TButton").pack(fill="x", pady=5)
49     ttk.Button(button_frame, text="Update Book", command=lambda: open_update_book_popup(app), style="crimson.TButton").pack(fill="x", pady=5)
50     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)
51     ttk.Button(button_frame, text="Download Barcode", command=lambda: open_download_barcodes_popup(app, table), style="crimson.TButton").pack(fill="x", pady=5)
52     ttk.Button(button_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").pack(fill="x", pady=50)
53
54     app.grid_columnconfigure(0, weight=3)
55     app.grid_columnconfigure(1, weight=1)
56
57     display_books()
58
59 def open_update_book_popup(app):
60     selected_item = table.selection()
61     if not selected_item:
62         messagebox.showerror("Error", "No book selected!")
63         return
64
65     ISBN = str(table.item(selected_item)["values"][1])
66     book = read_book(ISBN)
67
68     if isinstance(book, str):
69         messagebox.showerror("Error", "Book not found!")
70         return
71
72     update_book_popup(app, book, display_books)
73
74 def display_books():
75     for row in table.get_children():
76         table.delete(row)
77
78     for index, book in enumerate(Books):
79         table.insert("", "end", values=(
80             index + 1,
81             book["ISBN"],
82             book["Title"],
83             book["Description"],
84             book["Category"],
85             book["Quantity"],
86             ", ".join(book["SKU"].keys()),
87             book["Author"],
88             book["Language"],
89             book["Publisher"],
90         ))
91
92     def update_table(table):
93         for row in table.get_children():
94             table.delete(row)
95
96         for book in Books:
97             table.insert("", "end", values=(book["ISBN"], book["Title"], book["Author"], book["Quantity"]))
98
```

```
● ● ● admin/shelf_manage.py

 1 import tkinter as tk
 2 from tkinter import StringVar, ttk, messagebox
 3 from backend.shelfing import deshelve, search, shelf, categorise, BookShelf, DeshelvedBooks
 4 from backend.utils import open_barcode_scanner
 5
 6 def back_to_dashboard(app):
 7     from admin.dashboard import welcome_screen
 8
 9     for widget in app.winfo_children():
10         widget.destroy()
11
12     welcome_screen(app)
13
14 def show_deshelved_books():
15     if not DeshelvedBooks:
16         messagebox.showinfo("Deshelved Books", "No books have been deshelfed.")
17     else:
18         books_list = "\n".join([f"SKU: {sku}, {details}" for sku, details in DeshelvedBooks.items()])
19         messagebox.showinfo("Deshelved Books", f"Deshelved Books:\n\n{books_list}")
20
21
22 def open_shelfe_popup(app, update_shelf_view):
23     def shelve_book():
24         try:
25             rack = int(rack_entry.get())
26             shelf_num = int(shelf_entry.get())
27             sku = sku_entry.get()
28
29             if not sku:
30                 raise ValueError("SKU cannot be empty.")
31
32             result = shelf(rack, shelf_num, sku)
33             messagebox.showinfo("Success", result)
34
35             update_shelf_view()
36             popup.destroy()
37
38         except ValueError as e:
39             messagebox.showerror("Error", str(e))
40
41     popup = tk.Toplevel(app)
42     popup.title("Shelve Book")
43
44     tk.Label(popup, text="Rack Number").grid(row=0, column=0, padx=10, pady=5)
45     rack_entry = tk.Entry(popup)
46     rack_entry.grid(row=0, column=1, padx=10, pady=5)
47
48     tk.Label(popup, text="Shelf Number").grid(row=1, column=0, padx=10, pady=5)
49     shelf_entry = tk.Entry(popup)
50     shelf_entry.grid(row=1, column=1, padx=10, pady=5)
51
52     sku_var = StringVar()
53     tk.Label(popup, text="SKU").grid(row=2, column=0, padx=10, pady=5)
54     sku_entry = tk.Entry(popup, textvariable=sku_var)
55     sku_entry.grid(row=2, column=1, padx=10, pady=5)
56
57     def start_scanning():
58         open_barcode_scanner(sku_var)
59     ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=3, column=0, columnspan=2, pady=20)
60
61     ttk.Button(popup, text="Submit", command=shelve_book, style="crimson.TButton").grid(row=4, column=0, columnspan=2, pady=20)
62
63
64 def open_search_popup(app):
65     def search_book():
66         sku = sku_entry.get()
67
68         if not sku:
69             messagebox.showerror("Error", "SKU cannot be empty.")
70             return
71
72         result = search(sku)
73         if "Error" in result:
74             messagebox.showerror("Error", result)
75         else:
76             messagebox.showinfo("Success", result)
77             popup.destroy()
78
79     popup = tk.Toplevel(app)
80     popup.title("Search Book")
81     sku_var = StringVar()
82
83     tk.Label(popup, text="Enter SKU").grid(row=0, column=0, padx=10, pady=5)
84     sku_entry = tk.Entry(popup, textvariable=sku_var)
85     sku_entry.grid(row=0, column=1, padx=10, pady=5)
```

```

86     def start_scanning():
87         open_barcode_scanner(sku_var)
88         ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=1, column=0, columnspan=2, pady=10)
89
90     ttk.Button(popup, text="Search", command=search_book, style="crimson.TButton").grid(row=2, column=0, columnspan=2, pady=20)
91
92
93 def open_deshelve_popup(app, update_shelf_view):
94     sku_var = StringVar()
95     def deshelve_book():
96         sku = sku_entry.get()
97
98         if not sku:
99             messagebox.showerror("Error", "SKU cannot be empty.")
100            return
101
102         result = deshelve(sku)
103         if "Error" in result:
104             messagebox.showerror("Error", result)
105         else:
106             messagebox.showinfo("Success", result)
107
108         update_shelf_view()
109         popup.destroy()
110
111     popup = tk.Toplevel(app)
112     popup.title("Deshelve Book")
113     def start_scanning():
114         open_barcode_scanner(sku_var)
115
116     tk.Label(popup, text="Enter SKU").grid(row=0, column=0, padx=10, pady=5)
117     sku_entry = tk.Entry(popup, textvariable=sku_var)
118     sku_entry.grid(row=0, column=1, padx=10, pady=5)
119
120     ttk.Button(popup, text="Scan Barcode", command=start_scanning, style="crimson.TButton").grid(row=1, column=0, columnspan=2, pady=10)
121     ttk.Button(popup, text="Deshelve", command=deshelve_book, style="crimson.TButton").grid(row=2, column=0, columnspan=2, pady=10)
122
123
124 def open_categorise_popup(app, update_shelf_view):
125     def categorise_book():
126         try:
127             rack = int(rack_entry.get())
128             shelf_num = int(shelf_entry.get())
129             category = category_entry.get()
130
131             if not category:
132                 raise ValueError("Category cannot be empty.")
133
134             result = categorise(rack, shelf_num, category)
135             messagebox.showinfo("Success", result)
136
137             update_shelf_view()
138             popup.destroy()
139
140         except ValueError as e:
141             messagebox.showerror("Error", str(e))
142
143     popup = tk.Toplevel(app)
144     popup.title("Categorize Book")
145
146     tk.Label(popup, text="Rack Number").grid(row=0, column=0, padx=10, pady=5)
147     rack_entry = tk.Entry(popup)
148     rack_entry.grid(row=0, column=1, padx=10, pady=5)
149
150     tk.Label(popup, text="Shelf Number").grid(row=1, column=0, padx=10, pady=5)
151     shelf_entry = tk.Entry(popup)
152     shelf_entry.grid(row=1, column=1, padx=10, pady=5)
153
154     tk.Label(popup, text="Category").grid(row=2, column=0, padx=10, pady=5)
155     category_entry = tk.Entry(popup)
156     category_entry.grid(row=2, column=1, padx=10, pady=5)
157
158     ttk.Button(popup, text="Submit", command=categorise_book, style="crimson.TButton").grid(row=3, column=0, columnspan=2, pady=10)
159
160

```

```

161 def shelf_manage(app):
162     def update_shelf_view():
163         for widget in scrollable_frame.winfo_children():
164             widget.destroy()
165
166     if not BookShelf:
167         no_rack_label = ttk.Label(scrollable_frame, text="No rack found!", font=("Arial", 16, "bold"))
168         no_rack_label.grid(row=0, column=0, padx=10, pady=10)
169     return
170
171     row = 0
172     col = 0
173     max_columns = 3
174
175     for rack_number, categories in enumerate(BookShelf, 1):
176         rack_frame = ttk.Frame(scrollable_frame, padding=10, relief="solid", width=200, height=300)
177         rack_frame.grid(row=row, column=col, padx=20, pady=20, sticky="nsew")
178
179         rack_label = ttk.Label(rack_frame, text=f"Rack {rack_number}", font=("Arial", 14, "bold"))
180         rack_label.grid(row=0, column=0, padx=10, pady=5)
181
182         shelf_row = 1
183         for category_dict in categories:
184             if category_dict:
185                 for category_name, skus in category_dict.items():
186                     shelf_frame = ttk.Frame(rack_frame, padding=10, relief="groove")
187                     shelf_frame.grid(row=shelf_row, column=0, padx=5, pady=5, sticky="nsew")
188
189                     shelf_label = ttk.Label(shelf_frame, text=category_name, font=(("Arial", 12)))
190                     shelf_label.grid(row=0, column=0, padx=5, pady=5)
191
192                     table_frame = ttk.Frame(shelf_frame)
193                     table_frame.grid(row=1, column=0, padx=5, pady=5)
194
195                     columns = ("SKU",)
196                     table = ttk.Treeview(table_frame, columns=columns, show="headings", height=4)
197                     table.grid(row=0, column=0, sticky="nsew")
198
199                     table.heading("SKU", text="SKU")
200
201                     for sku in skus:
202                         table.insert("", "end", values=(sku,))
203
204                     shelf_row += 1
205
206                     col += 1
207                     if col > max_columns:
208                         col = 0
209                         row += 1
210
211             scrollable_frame.update_idletasks()
212             canvas.config(scrollregion=canvas.bbox("all"))
213
214     paned_window = ttk.PanedWindow(app, orient="horizontal")
215     paned_window.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)
216
217     left_frame = ttk.Frame(paned_window)
218     paned_window.add(left_frame, weight=4)
219
220     right_frame = ttk.Frame(paned_window, width=200)
221     paned_window.add(right_frame, weight=1)
222
223     canvas = tk.Canvas(left_frame)
224     canvas.grid(row=0, column=0, sticky="nsew", padx=10, pady=10)
225
226     scrollable_frame = ttk.Frame(canvas)
227     canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
228
229     scrollbar = ttk.Scrollbar(left_frame, orient="vertical", command=canvas.yview)
230     scrollbar.grid(row=0, column=1, sticky="ns")
231     canvas.configure(yscrollcommand=scrollbar.set)
232
233     left_frame.grid_rowconfigure(0, weight=1)
234     left_frame.grid_columnconfigure(0, weight=1)
235
236     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)
237     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)
238     ttk.Button(right_frame, text="Search Book", command=lambda: open_search_popup(app), style="crimson.TButton").grid(row=2, column=0, pady=10)
239     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)
240     ttk.Button(right_frame, text="View Deshelfed Books", command=lambda: show_deshelfed_books(), style="crimson.TButton").grid(row=4, column=0, pady=10)
241     ttk.Button(right_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").grid(row=5, column=0, pady=50)
242
243     app.grid_rowconfigure(0, weight=1)
244     app.grid_columnconfigure(0, weight=1)
245
246     update_shelf_view()

```

```
 1  from tkinter import ttk, messagebox
 2  from admin.member_popups import open_add_member_popup, open_update_member_book_popup, update_member_popup, open_delete_member_popup
 3  from backend.members import Members, read_member
 4
 5  def back_to_dashboard(app):
 6      from admin.dashboard import welcome_screen
 7
 8      for widget in app.winfo_children():
 9          widget.grid_forget()
10
11  welcome_screen(app)
12
13  def membership_manage(app):
14      global table, table_frame
15
16      app.grid_columnconfigure(0, weight=1)
17      app.grid_rowconfigure(0, weight=1)
18
19      table_frame = ttk.Frame(app)
20      table_frame.grid(row=0, column=0, padx=20, pady=20, sticky="nsew")
21      table_frame.grid_columnconfigure(0, weight=1)
22
23      columns = ("UID", "Name", "Email", "Borrowed Books", "Join Date")
24      table = ttk.Treeview(table_frame, columns=columns, show="headings", height=40)
25
26      for col in columns:
27          table.heading(col, text=col)
28          table.column(col, width=100, anchor="center", stretch=True)
29
30      scrollbar = ttk.Scrollbar(table_frame, orient="horizontal", command=table.xview)
31      table.configure(xscrollcommand=scrollbar.set)
32      scrollbar.grid(row=1, column=0, sticky="ew")
33
34      table.grid(row=0, column=0, padx=20, pady=10, sticky="nsew")
35
36      button_frame = ttk.Frame(app)
37      button_frame.grid(row=0, column=1, padx=20, pady=20, sticky="nsew")
38      button_frame.grid_columnconfigure(0, weight=1)
39
40      tk.Button(button_frame, text="Add New Member", command=lambda: open_add_member_popup(app, display_members), style="crimson.TButton").pack(fill="x", pady=5)
41      tk.Button(button_frame, text="Update Member Details", command=lambda: open_update_member_popup(app), style="crimson.TButton").pack(fill="x", pady=5)
42      tk.Button(button_frame, text="Delete Member", command=lambda: open_delete_member_popup(app, table, display_members), style="crimson.TButton").pack(fill="x", pady=5)
43      tk.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)
44      tk.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)
45      tk.Button(button_frame, text="Dashboard", command=lambda: back_to_dashboard(app), style="crimson.TButton").pack(fill="x", pady=50)
46
47      app.grid_columnconfigure(0, weight=3)
48      app.grid_columnconfigure(1, weight=1)
49
50  display_members()
51
52  def open_update_member_popup(app):
53      selected_item = table.selection()
54      if not selected_item:
55          messagebox.showerror("Error", "No member selected!")
56          return
57
58      UID = table.item(selected_item)["values"][0]
59      if not UID:
60          messagebox.showerror("Error", "Invalid UID!")
61          return
62
63      member = read_member(UID)
64
65      if isinstance(member, str):
66          messagebox.showerror("Error", "Member not found!")
67          return
68
69      update_member_popup(app, member, display_members)
70
71  def display_members():
72      for row in table.get_children():
73          table.delete(row)
74
75      for index, member in enumerate(Members):
76          table.insert("", "end", values=(
77              member["UID"],
78              member["Name"],
79              member["Email"],
80              member["SKU"],
81              member["JoinedOn"],
82          ))
83
84  def update_table(table):
85      for row in table.get_children():
86          table.delete(row)
87
88      for member in Members:
89          table.insert("", "end", values=(member["UID"], member["Name"], member["Email"], member["SKU"], member["JoinedOn"]))
```

```
 1  from tkinter import filedialog
 2  import ttkbootstrap as ttk
 3  import tkinter as tk
 4  from tkinter import StringVar, messagebox
 5  from backend.books import add_book, download_barcodes, update_books, remove_books
 6
 7  def get_selected_book(table):
 8      selected = table.focus()
 9      if selected:
10          return table.item(selected, "values")
11      return None
12
13 def open_add_book_popup(app, refresh_table_callback):
14     popup = ttk.Toplevel(app)
15     popup.geometry("600x400")
16     popup.title("Add Book")
17     popup.configure(bg="white")
18
19     canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
20     canvas.pack(fill="both", expand=True)
21
22     width = 600
23     height = 400
24
25     form_frame = ttk.Frame(canvas)
26     canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
27     form_frame.grid_rowconfigure(0, weight=1)
28
29     ttk.Label(form_frame, text="Add Book", font=("Cambria", 18, "bold")).grid(row=0, columnspan=2, pady=20)
30
31     isbn_var = StringVar()
32     title_var = StringVar()
33     description_var = StringVar()
34     category_var = StringVar()
35     quantity_var = StringVar()
36     author_var = StringVar()
37     publisher_var = StringVar()
38     language_var = StringVar()
39
40     fields = [
41         ("ISBN", isbn_var),
42         ("Title", title_var),
43         ("Description", description_var),
44         ("Category", category_var),
45         ("Quantity", quantity_var),
46         ("Author", author_var),
47         ("Publisher", publisher_var),
48         ("Language", language_var),
49     ]
50
51     entry_vars = {}
52
53     for i, (label, var) in enumerate(fields):
54         row = i // 2
55         col = i % 2
56
57         entry_vars[label] = var
58
59         ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).grid(row=row + 1, column=col * 2, padx=5, pady=5, sticky="w")
60         ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).grid(row=row + 1, column=col * 2 + 1, padx=5, pady=5, sticky="ew")
61
62     def handle_add_book():
63         try:
64             isbn = isbn_var.get().strip()
65             title = title_var.get().strip()
66             description = description_var.get().strip()
67             category = category_var.get().strip()
68             quantity = int(quantity_var.get())
69             author = author_var.get().strip()
70             publisher = publisher_var.get().strip()
71             language = language_var.get().strip()
72
73             if not isbn or not title or not description or not category or not author or not publisher or not language:
74                 messagebox.showerror("Error", "All fields must be filled in.")
75                 return
76
77             result = add_book(
78                 ISBN=isbn,
79                 Title=title,
80                 Description=description,
81                 Category=category,
82                 Quantity=quantity,
83                 Author=author,
84                 Publisher=publisher,
85                 Language=language,
86             )
87

```

```
88
89     if result.startswith("Invalid"):
90         messagebox.showerror("Error", result)
91     elif result.startswith("Book quantity updated") or result.startswith("Book added"):
92         messagebox.showinfo("Success", result)
93         refresh_table_callback()
94         popup.destroy()
95
96     except ValueError:
97         messagebox.showerror("Error", "Please enter valid data for Quantity.")
98
99     ttk.Button(form_frame, text="Add Book", command=handle_add_book, style="crimson.TButton").grid(row=len(fields) // 2 + 2, columnspan=2, pady=10)
100
101    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)
102
103 def update_book_popup(app, book_data, refresh_table_callback):
104     popup = tk.Toplevel(app)
105     popup.geometry("600x400")
106     popup.title("Update Book")
107     popup.configure(bg="white")
108
109     canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
110     canvas.pack(fill="both", expand=True)
111
112     width = 600
113     height = 400
114
115     form_frame = ttk.Frame(canvas)
116     canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
117     form_frame.grid_rowconfigure(0, weight=1)
118
119     ttk.Label(form_frame, text="Update Book", font=("Cambria", 18, "bold")).grid(row=0, columnspan=2, pady=20)
120
121     title_var = StringVar(value=book_data.get("Title", ""))
122     description_var = StringVar(value=book_data.get("Description", ""))
123     category_var = StringVar(value=book_data.get("Category", ""))
124     author_var = StringVar(value=book_data.get("Author", ""))
125     publisher_var = StringVar(value=book_data.get("Publisher", ""))
126     language_var = StringVar(value=book_data.get("Language", ""))
127
128     fields = [
129         ("Title", title_var),
130         ("Description", description_var),
131         ("Category", category_var),
132         ("Author", author_var),
133         ("Publisher", publisher_var),
134         ("Language", language_var),
135     ]
136
137     entry_vars = {}
138
139     for i, (label, var) in enumerate(fields):
140         row = i // 2
141         col = i % 2
142
143         entry_vars[label] = var
144
145         ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).grid(row=row + 1, column=col * 2, padx=5, pady=5, sticky="w")
146         ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).grid(row=row + 1, column=col * 2 + 1, padx=5, pady=5, sticky="ew")
147
148     def save_changes():
149         updated_data = {label: var.get().strip() for label, var in entry_vars.items()}
150         updated_data["ISBN"] = book_data.get("ISBN", "")
151
152         if not updated_data["ISBN"]:
153             messagebox.showerror("Error", "ISBN is required.")
154             return
155
156         if any(not value for value in updated_data.values()):
157             messagebox.showerror("Error", "All fields must be filled in.")
158             return
159
160         result = update_books(
161             updated_data["ISBN"],
162             updated_data["Title"],
163             updated_data["Description"],
164             updated_data["Category"],
165             updated_data["Author"],
166             updated_data["Publisher"],
167             updated_data["Language"]
168         )
169
170         if result.startswith("Invalid"):
171             messagebox.showerror("Error", result)
172         elif result.startswith("Book details updated"):
173             messagebox.showinfo("Success", result)
174             refresh_table_callback()
175             popup.destroy()
176
177     ttk.Button(form_frame, text="Save Changes", command=save_changes, style="crimson.TButton").grid(row=len(fields) // 2 + 2, columnspan=2, pady=10)
178
179     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)
```

```
 180 def open_delete_book_popup(app, table, refresh_table_callback):
 181     selected_item = table.selection()
 182     if not selected_item:
 183         messagebox.showerror("Error", "No book selected!")
 184         return
 185
 186     popup = tk.Toplevel(app)
 187     popup.geometry("300x300")
 188     popup.title("Delete Book")
 189     popup.configure(bg="white")
 190
 191     canvas = tk.Canvas(popup, highlightthickness=0, bg="white")
 192     canvas.pack(fill="both", expand=True)
 193
 194     width, height = 300, 300
 195
 196     form_frame = ttk.Frame(canvas)
 197     canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
 198     form_frame.grid_rowconfigure(0, weight=1)
 199
 200     ttk.Label(form_frame, text="Enter SKU no. to delete.", font=("Cambria", 8, "bold")).grid(row=0, columnspan=2, pady=20)
 201     sku_var = ttk.StringVar()
 202
 203     ttk.Label(form_frame, text="SKU:", font=("Century Gothic", 8)).grid(row=1, column=0, padx=5, pady=5, sticky="e")
 204     sku_entry = ttk.Entry(form_frame, textvariable=sku_var, font=("Century Gothic", 10))
 205     sku_entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")
 206
 207     def delete_book_by_sku():
 208         sku_number = sku_var.get().strip()
 209
 210         if not sku_number:
 211             messagebox.showerror("Error", "Please enter a valid SKU number.")
 212             return
 213
 214         try:
 215             sku_list = table.item(selected_item)["values"][6].split(",")
 216             sku = sku_list[int(sku_number)] - 1
 217
 218             result = remove_books(sku)
 219
 220             if "Error:" in result:
 221                 messagebox.showerror("Error", result)
 222             else:
 223                 messagebox.showinfo("Success", result)
 224                 refresh_table_callback()
 225             popup.destroy()
 226
 227         except IndexError:
 228             messagebox.showerror("Error", "Invalid SKU number. Please enter a valid SKU index.")
 229         except ValueError:
 230             messagebox.showerror("Error", "Invalid SKU number. Please enter a valid number.")
 231
 232     def delete_all_books():
 233         res = messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete all books?")
 234
 235         if res:
 236             isbn = table.item(selected_item)["values"][1]
 237             deletion_result = remove_books(isbn, delete_all=True)
 238
 239             if deletion_result == "All books with ISBN removed successfully":
 240                 messagebox.showinfo("Success", "All books deleted successfully!")
 241                 popup.destroy()
 242                 refresh_table_callback()
 243             else:
 244                 messagebox.showerror("Error", deletion_result)
 245
 246     ttk.Button(form_frame, text="Delete SKU", command=delete_book_by_sku, style="crimson.TButton").grid(row=2, columnspan=2, pady=10)
 247     ttk.Button(form_frame, text="Delete All", command=delete_all_books, style="crimson.TButton").grid(row=3, columnspan=2, pady=10)
 248
 249     ttk.Label(form_frame, text="This action cannot be undone.", font=("Calibri", 10, "italic")).grid(row=4, columnspan=2, pady=10)
 250
 251     def open_download_barcodes_popup(app, table):
 252         selected_item = table.selection()
 253         if not selected_item:
 254             messagebox.showerror("Error", "No book selected!")
 255             return
 256
 257         selected_book = table.item(selected_item)["values"]
 258         if not selected_book:
 259             messagebox.showerror("Error", "Unable to retrieve selected book data!")
 260             return
 261
 262         book = {
 263             "ISBN": selected_book[0],
 264             "Title": str(selected_book[1]),
 265             "SKU": {sku: "date" for sku in selected_book[6].split(",")}}
 266     }
 267
 268     confirm = messagebox.askyesno("Confirmation", f"Do you want to download barcodes for '{book['Title']}'?")
 269     if not confirm:
 270         return
 271
 272     save_file = filedialog.asksaveasfilename(
 273         title="Save Barcode PDF",
 274         defaultextension=".pdf",
 275         filetypes=[("PDF Files", "*.*")],
 276         initialfile=f'{book["Title"].replace(' ', '_)}_barcodes.pdf'
 277     )
 278
 279     if save_file:
 280         result = download_barcodes(book, save_path=save_file)
 281     else:
 282         result = download_barcodes(book)
 283     if result.startswith("Success"):
 284         messagebox.showinfo("Success", result)
 285     else:
 286         messagebox.showerror("Error", result)
```

```
 1  import ttkbootstrap as ttk
 2  from tkinter import StringVar, messagebox
 3  import re
 4  from backend.members import add_member, remove_member, update_member, update_member_details
 5
 6  from backend.utils import open_barcode_scanner
 7
 8  def is_valid_email(email):
 9      email_regex = r'^[a-zA-Z0-9_.%+-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,}$'
10      return re.match(email_regex, email) is not None
11
12 def open_add_member_popup(app, refresh_table_callback):
13     popup = tk.Toplevel(app)
14     popup.geometry("600x400")
15     popup.title("Add Member")
16     popup.configure(bg="white")
17
18     form_frame = ttk.Frame(popup)
19     form_frame.pack(fill="both", expand=True)
20
21     ttk.Label(form_frame, text="Add New Member", font=("Cambria", 18, "bold")).pack(pady=20)
22
23     name_var = StringVar()
24     email_var = StringVar()
25     password_var = StringVar()
26
27     fields = [
28         ("Full Name", name_var),
29         ("Email Address", email_var),
30     ]
31
32     for label, var in fields:
33         ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).pack(pady=5)
34         ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).pack(pady=5)
35
36     ttk.Label(form_frame, text="Password", font=("Century Gothic", 8)).pack(pady=5)
37     ttk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 10), show="*").pack(pady=5)
38
39     def handle_add_member():
40         name = name_var.get()
41         email = email_var.get()
42         password = password_var.get()
43
44         if not is_valid_email(email):
45             messagebox.showerror("Error", "Please enter a valid email address.")
46             return
47
48         try:
49             res = add_member(
50                 Name=name,
51                 Email=email,
52                 Password=password,
53             )
54             if "Error:" in res:
55                 messagebox.showerror("Error", res)
56             else:
57                 messagebox.showinfo("Success", "Member added successfully!")
58                 refresh_table_callback()
59                 popup.destroy()
60         except ValueError:
61             messagebox.showerror("Error", "Please enter valid data for Quantity.")
62
63     ttk.Button(form_frame, text="Add Member", command=handle_add_member, style="crimson.TButton").pack(pady=10)
64
65     ttk.Label(form_frame, text="Fill the details and click 'Add Member' to create new membership.", font=("Calibri", 10, "italic")).pack(pady=10)
66
67 def update_member_popup(app, member_data, refresh_table_callback):
68     popup = tk.Toplevel(app)
69     popup.geometry("600x400")
70     popup.title("Update Member Details")
71     popup.configure(bg="white")
72
73     form_frame = ttk.Frame(popup)
74     form_frame.pack(fill="both", expand=True)
75
76     ttk.Label(form_frame, text="Update Member Details", font=("Cambria", 18, "bold")).pack(pady=20)
77
78     name_var = StringVar(value=member_data.get("Name", ""))
79     old_password_var = StringVar()
80     email_var = StringVar(value=member_data.get("Email", ""))
81     new_password_var = StringVar()
82
83     fields = [
84         ("Full Name", name_var),
85         ("Email Address", email_var),
86     ]
87
88     for label, var in fields:
89         ttk.Label(form_frame, text=label, font=("Century Gothic", 8)).pack(pady=5)
90         ttk.Entry(form_frame, textvariable=var, font=("Century Gothic", 10)).pack(pady=5)
91
92     ttk.Label(form_frame, text="Old Password", font=("Century Gothic", 8)).pack(pady=5)
93     ttk.Entry(form_frame, textvariable=old_password_var, font=("Century Gothic", 10), show="*").pack(pady=5)
94     ttk.Label(form_frame, text="New Password", font=("Century Gothic", 8)).pack(pady=5)
95     ttk.Entry(form_frame, textvariable=new_password_var, font=("Century Gothic", 10), show="*").pack(pady=5)
96
97     def save_changes():
98         updated_data = {label: var.get() for label, var in fields}
99         updated_data["UID"] = member_data.get("UID", "")
100        updated_data["New Password"] = new_password_var.get().strip()
101        updated_data["Old Password"] = old_password_var.get().strip()
102
103        if 'UID' not in updated_data or not updated_data["UID"]:
104            messagebox.showerror("Error", "UID is required.")
105            return
106
107        if not is_valid_email(updated_data["Email Address"]):
108            messagebox.showerror("Error", "Please enter a valid email address.")
109            return
```

```
admin/member_popup.py

110     result = update_member_details(
111         updated_data["UID"],
112         updated_data["Full Name"],
113         updated_data["Email Address"],
114         updated_data["New Password"],
115         updated_data["Old Password"],
116     )
117     if result == "Member details updated successfully":
118         messagebox.showinfo("Success", result)
119         refresh_table_callback()
120         popup.destroy()
121     elif result == "Old password is incorrect":
122         messagebox.showerror("Error", "The old password you entered is incorrect. Please try again.")
123     elif result == "No member found":
124         messagebox.showerror("Error", "No member found with the given UID.")
125     else:
126         messagebox.showerror("Error", result+".")
127     refresh_table_callback()
128     popup.destroy()
129
130     ttk.Button(form_frame, text="Save Changes", command=save_changes, style="crimson.TButton").pack(pady=10)
131
132     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)
133     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)
134
135 def open_delete_member_popup(app, table, refresh_table_callback):
136     selected_item = table.selection()
137     if not selected_item:
138         messagebox.showerror("Error", "No member selected!")
139         return
140     res = messagebox.askyesno("Confirm Deletion", "Are you sure you want to cancel membership?")
141     if res:
142         uid = table.item(selected_item)["values"][0]
143         deletion_result = remove_member(uid)
144
145         if deletion_result == "Member removed successfully":
146             messagebox.showinfo("Success", "Membership cancelled successfully!")
147             refresh_table_callback()
148         else:
149             messagebox.showerror("Error", deletion_result)
150
151
152 def open_update_member_book_popup(app, table, addbook, refresh_table_callback):
153     selected_item = table.selection()
154     if not selected_item:
155         messagebox.showerror("Error", "No member selected!")
156         return
157
158     popup = tk.Toplevel(app)
159     popup.geometry("600x400")
160     popup.title("Borrow or Return Book")
161     popup.configure(bg="white")
162
163     form_frame = ttk.Frame(popup)
164     form_frame.pack(fill="both", expand=True)
165
166     ttk.Label(form_frame, text="Borrow or Return Book", font=("Cambria", 18, "bold")).pack(pady=20)
167
168     uid = table.item(selected_item)["values"][0]
169
170     sku_var = StringVar()
171     ttk.Label(form_frame, text="Enter SKU of Book", font=("Century Gothic", 10)).pack(pady=10)
172     sku_entry = ttk.Entry(form_frame, textvariable=sku_var, font=("Century Gothic", 10))
173     sku_entry.pack(pady=10)
174
175     def handle_action():
176         sku = sku_var.get()
177         if not sku:
178             messagebox.showerror("Error", "Please enter the SKU of the book!")
179             return
180
181         result = update_member(uid, sku, addbook)
182
183         if "Fine incurred" in result:
184             fine_confirmation = messagebox.askyesno(
185                 "Fine Confirmation",
186                 f"{result}\n\nHas the fine been paid?"
187             )
188             if fine_confirmation:
189                 result = update_member(uid, sku, addbook, fine_paid=True)
190                 messagebox.showinfo("Success", result)
191             else:
192                 messagebox.showinfo("Info", "The book will remain with member until the fine is paid.")
193
194         elif "successfully" in result:
195             messagebox.showinfo("Success", result)
196
197         else:
198             messagebox.showerror("Error", result)
199
200         refresh_table_callback()
201         popup.destroy()
202
203     def start_scanning():
204         open_barcode_scanner(sku_var)
205
206     ttk.Button(form_frame, text="Scan Barcode", command=start_scanning, style="crimson.TButton").pack(pady=5)
207     ttk.Button(form_frame, text="Submit", command=handle_action, style="crimson.TButton").pack(pady=20)
208
209     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)
210
211     def on_close():
212         popup.destroy()
213
214     popup.protocol("WM_DELETE_WINDOW", on_close)
```

Member UI



A screenshot of a code editor window titled "client.py". The window has a light gray header bar with three colored dots (red, yellow, green) on the left and a file icon with "client.py" on the right. The main area contains Python code for a Tkinter application.

```
1  from tkinter import messagebox
2  import ttkbootstrap as ttk
3  from backend.utils import BOOKS_FILE, MEMBERS_FILE, save_data
4  from client.login_screen import login_screen
5  from constants import APP_NAME
6  from backend.books import Books
7  from backend.members import Members
8
9  def on_close():
10     if messagebox.askyesno("Exit", "Are you sure you want to exit?"):
11         app.destroy()
12
13 app = ttk.Window(themename="darkly")
14 app.geometry("1200x800")
15 app.state('zoomed')
16 app.title("LibPro | A Library App")
17
18 app.protocol("WM_DELETE_WINDOW", on_close)
19
20 style = ttk.Style()
21 style.configure("crimson.TButton",
22                 background="#dc143c",
23                 foreground="white",
24                 borderwidth=0,
25                 focusthickness=0)
26
27 login_screen(app)
28 app.mainloop()
```

```
 1 import ttkbootstrap as ttk
 2 from tkinter import messagebox
 3 import tkinter as tk
 4
 5 from backend.members import sign_in
 6 from client.dashboard import welcome_screen
 7
 8 def login_screen(app):
 9     global login_frame, email_var, password_var
10
11     login_frame = ttk.Frame(app, padding=30)
12     login_frame.pack(expand=True, fill="both")
13
14     branding_frame = ttk.Frame(login_frame, padding=20)
15     branding_frame.pack(side="left", fill="both", expand=True, padx=20, pady=20)
16
17     ttk.Label(branding_frame, anchor="center").pack(pady=70)
18
19     ttk.Label(branding_frame, text="LibAmma", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
20
21     ttk.Label(branding_frame, text="A Library App.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
22
23     ttk.Label(branding_frame, text="By Pratham, Thejas and Thoshit.", font=("Arial", 12), anchor="center").pack(pady=50)
24
25     login_section = ttk.Frame(login_frame, padding=20)
26     login_section.pack(side="right", fill="both", expand=True, padx=20, pady=20)
27
28     canvas = tk.Canvas(login_section, highlightthickness=0)
29     canvas.pack(fill="both", expand=True)
30
31     width = 500
32     height = 500
33     radius = 20
34     canvas.create_oval(0, 0, radius * 2, radius * 2, outline="#ffffff")
35     canvas.create_rectangle(radius, 0, width - radius, height, outline="#ffffff")
36     canvas.create_oval(width - radius * 2, 0, width, radius * 2, outline="#ffffff")
37     canvas.create_rectangle(0, radius, width, height - radius, outline="#ffffff")
38     canvas.create_oval(0, height - radius * 2, radius * 2, height, outline="#ffffff")
39     canvas.create_oval(width - radius * 2, height - radius * 2, width, height, outline="#ffffff")
40
41     form_frame = ttk.Frame(canvas)
42     canvas.create_window(width // 2, height // 2, window=form_frame, anchor="center")
43
44     ttk.Label(form_frame, text="Member Login", font=("Cambria", 30, "bold")).pack(pady=20)
45
46     ttk.Label(form_frame, text="Email Address:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
47     email_var = ttk.StringVar()
48     ttk.Entry(form_frame, textvariable=email_var, font=("Century Gothic", 12)).pack(fill="x", pady=5)
49
50     ttk.Label(form_frame, text="Password:", font=("Century Gothic", 14)).pack(anchor="w", pady=5)
51     password_var = ttk.StringVar()
52     ttk.Entry(form_frame, textvariable=password_var, font=("Century Gothic", 12), show="*").pack(fill="x", pady=5)
53
54     ttk.Button(form_frame, text="Login", command=lambda: validate_login(app), style="crimson.TButton").pack(pady=20, fill="x")
55
56     ttk.Label(form_frame, text="Welcome to the modern library experience.", font=("Calibri", 10, "italic")).pack(side="bottom", pady=10)
57
58
59 def validate_login(app):
60     email = email_var.get().strip()
61     password = password_var.get().strip()
62
63     if not email or not password:
64         messagebox.showerror("Error", "Email and password cannot be empty.")
65         return
66     res = sign_in(email, password)
67     if res == "Invalid credentials" or res == "Valid email is required" or res == "Password is required":
68         messagebox.showerror("Error", res)
69     else:
70         messagebox.showinfo("Success", "Login successful!")
71         login_frame.pack_forget()
72         welcome_screen(app, res)
```

```
client/dashboard.py
```

```
1 import time
2 from tkinter import messagebox
3 import ttkbootstrap as ttk
4
5 from client.view_books import view_books
6 from client.view_borrowed_books import view_borrowed_books
7 from client.wishlist import wishlist
8
9 def update_time(label):
10     current_time = time.strftime("%H:%M:%S")
11     current_date = time.strftime("%A %d, %Y")
12     label.config(text=f"Current Time: {current_time}\n{current_date}")
13     label.after(1000, update_time, label)
14
15 def logout(app):
16     res = messagebox.askyesno("Confirm Logout", "Are you sure you want to logout?")
17     if res:
18         from client.login_screen import login_screen
19         welcome_frame.pack_forget()
20         login_screen(app)
21
22
23 def welcome_screen(app, member):
24     def view_profile():
25         messagebox.showinfo("Your Profile", f"""Name: {member["Name"]}\n
26 Email ID: {member["Email"]}\nBooks Taken(SKUs): {member["SKU"]}\n
27 Wishlist(ISBNs): {member["Wishlist"]}\n\nUnique ID: {member["UID"]}\n
28 Membership Since: {member["JoinedOn"]}""")
29     global welcome_frame
30     if not app:
31         messagebox.showerror("Error", "Application instance not found.")
32     return
33     welcome_frame = ttk.Frame(app, padding=30)
34     welcome_frame.pack(expand=True, fill="both", pady=100)
35
36     form_frame = ttk.Frame(welcome_frame)
37     form_frame.pack(fill="both", expand=True)
38
39     left_frame = ttk.Frame(form_frame, padding=20)
40     left_frame.grid(row=0, column=0, padx=10, pady=10, sticky="nsew")
41
42     ttk.Label(left_frame, text="LibPro", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
43     ttk.Label(left_frame, text="A Library App.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
44
45     time_label = ttk.Label(left_frame, font=("Arial", 12))
46     time_label.pack(pady=10)
47     update_time(time_label)
48
49     greeting_label = ttk.Label(left_frame, text=f"Welcome {member['Name']}!", font=("Arial", 10, "bold"))
50     greeting_label.pack(pady=10)
51
52     right_frame = ttk.Frame(form_frame, padding=20)
53     right_frame.grid(row=0, column=1, padx=10, pady=10, sticky="nsew")
54
55     borrowed_books_button = ttk.Button(
56         right_frame,
57         text="View Borrowed Books",
58         command=lambda: view_borrowed_books(app, member),
59         style="crimson.TButton"
60     )
61     borrowed_books_button.pack(pady=10, fill="x")
62
63     view_books_button = ttk.Button(
64         right_frame,
65         text="View Books",
66         command=lambda: view_books(app, member),
67         style="crimson.TButton"
68     )
69     view_books_button.pack(pady=10, fill="x")
70
71     membership_management_button = ttk.Button(
72         right_frame,
73         text="View Wishlist",
74         command=lambda: wishlist(app, member),
75         style="crimson.TButton"
76     )
77     membership_management_button.pack(pady=10, fill="x")
78
79     profile_management_button = ttk.Button(
80         right_frame,
81         text="View Your Profile",
82         command=lambda: view_profile(),
83         style="crimson.TButton"
84     )
85     profile_management_button.pack(pady=10, fill="x")
86
87     logout_button = ttk.Button(
88         right_frame,
89         text="Logout",
90         command=lambda: logout(app),
91         style="crimson.TButton"
92     )
93     logout_button.pack(pady=50, fill="both")
94
95     form_frame.grid_columnconfigure(0, weight=1)
96     form_frame.grid_columnconfigure(1, weight=1)
97     form_frame.grid_rowconfigure(0, weight=1)
```

```
client/view_books.py

 1  from tkinter import messagebox
 2  import ttkbootstrap as ttk
 3  from backend.books import Books
 4  from backend.members import manage_wishlist
 5  from client import dashboard
 6
 7  ACCENT_COLOR = "#dc143c"
 8
 9
10 def add_to_wishlist(uid, isbn):
11     res = manage_wishlist(True, uid, isbn)
12     if "Error:" in res:
13         messagebox.showerror("Error", res)
14     else:
15         messagebox.showinfo("Wishlist", f"{isbn} added to wishlist successfully!")
16
17 def view_books(app, member):
18     def show_main_page():
19         def go_to_dashboard():
20             main_frame.pack_forget()
21             dashboard.welcome_screen(app, member)
22
23             for widget in app.winfo_children():
24                 widget.destroy()
25
26             main_frame = ttk.Frame(app, padding=30)
27             main_frame.pack(fill="both", expand=True)
28
29             left_panel = ttk.Frame(main_frame, padding=20)
30             left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)
31
32             ttk.Label(left_panel, text="Available Books", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
33             ttk.Label(left_panel, text="List of Books available in this library.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
34             back_button = ttk.Button(
35                 left_panel,
36                 text="Back to Dashboard",
37                 command=go_to_dashboard,
38                 style="crimson.TButton",
39             )
40             back_button.pack(pady=60)
41
42             right_panel = ttk.Frame(main_frame, padding=20)
43             right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)
44
45             canvas = ttk.Canvas(right_panel, highlightthickness=0)
46             scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
47             scrollable_frame = ttk.Frame(canvas)
48
49             scrollable_frame.bind(
50                 "<Configure>",
51                 lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
52             )
53             canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
54             canvas.configure(yscrollcommand=scrollbar.set)
55
56             canvas.pack(side="left", fill="both", expand=True)
57             scrollbar.pack(side="right", fill="y")
58
59             style = ttk.Style()
60             style.configure('hover.TFrame', background="#f0f0f0", borderwidth=0)
61
62             if len(Books)>0:
63                 for book in Books:
64                     book_frame = ttk.Frame(scrollable_frame, padding=10, style="default.TFrame")
65                     book_frame.pack(fill="x", pady=(0, 1))
66
67                     def on_hover(e, frame=book_frame):
68                         frame.configure(style="hover.TFrame")
69
70                     def on_leave(e, frame=book_frame):
71                         frame.configure(style="default.TFrame")
72
73                     book_frame.bind("<Enter>", on_hover)
74                     book_frame.bind("<Leave>", on_leave)
75                     title_label = ttk.Label(
76                         book_frame,
77                         text=book["Title"],
78                         font=("Helvetica", 14, "bold"),
79                         foreground=ACCENT_COLOR,
80                         anchor="w"
81                     )
82                     title_label.pack(anchor="w", padx=5)
83                     title_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
84
85                     isbn_label = ttk.Label(
86                         book_frame,
87                         text=f"ISBN: {book['ISBN']}",
88                         font=("Helvetica", 10),
89                         anchor="w"
90                     )
91                     isbn_label.pack(anchor="w", padx=5)
92                     isbn_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
93
94                     book_frame.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
```

```
client/view_books.py

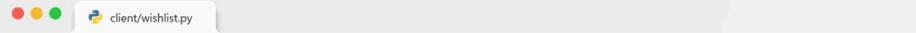
95     else:
96         ttk.Label(
97             scrollable_frame,
98             text="No books found.",
99             font=("Helvetica", 14, "bold"),
100            ).pack(anchor="w", padx=5)
101
102
103
104 def show_details_page(isbn):
105     for widget in app.winfo_children():
106         widget.destroy()
107
108     details_frame = ttk.Frame(app, padding=30)
109     details_frame.pack(fill="both", expand=True)
110
111     left_panel = ttk.Frame(details_frame, padding=20)
112     left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)
113
114     ttk.Label(
115         left_panel,
116         text="Book Details",
117         font=("Century Gothic", 40, "bold"),
118         anchor="center",
119        ).pack(pady=10)
120
121     ttk.Label(
122         left_panel,
123         text="Detailed Information about the selected book.",
124         font=("Arial", 18, "italic"),
125         anchor="center",
126        ).pack(pady=5)
127
128     right_panel = ttk.Frame(details_frame, padding=20)
129     right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)
130
131     canvas = ttk.Canvas(right_panel, highlightthickness=0)
132     scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
133     scrollable_frame = ttk.Frame(canvas)
134
135     scrollable_frame.bind(
136         "<Configure>",
137         lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
138     )
139     canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
140     canvas.configure(yscrollcommand=scrollbar.set)
141
142     canvas.pack(side="left", fill="both", expand=True)
143     scrollbar.pack(side="right", fill="y")
144
145     book = next((b for b in Books if b["ISBN"] == isbn), None)
146     if not book:
147         messagebox.showerror("Error", "Book not found.")
148         show_main_page()
149         return
150
151     book_details = [
152         ("Title", book["Title"]),
153         ("ISBN", book["ISBN"]),
154         ("Description", book["Description"]),
155         ("Category", book["Category"]),
156         ("Quantity", book["Quantity"]),
157         ("Author", book["Author"]),
158         ("Publisher", book["Publisher"]),
159         ("Language", book["Language"]),
160     ]
161
162     for label, adetail in book_details:
163         detail_fframe = ttk.Frame(scrollable_frame, padding=10)
164         detail_fframe.pack(fill="x", pady=(0, 1))
165
166         detail = adetail
167
168         if label == "Quantity":
169             if int(detail) == 0:
170                 detail = "All copies have been borrowed."
171             else:
172                 detail = f"{detail} copies in library."
173
174         ttk.Label(
175             detail_fframe,
176             text=f"{label}: {detail}",
177             font=("Helvetica", 12),
178             anchor="w",
179             wraplength=350
180            ).pack(anchor="w", padx=5)
181
182     wishlist_button = ttk.Button(
183         scrollable_frame,
184         text="Add to Wishlist",
185         command=lambda:add_to_wishlist(member["UID"], isbn),
186         style="crimson.TButton",
187     )
188     wishlist_button.pack(pady=20)
189
190     back_button = ttk.Button(
191         scrollable_frame,
192         text="Back",
193         command=show_main_page,
194         style="crimson.TButton",
195     )
196     back_button.pack(pady=30)
197
198     show_main_page()
```

```
client/view_borrowed_books.py
```

```
1  from datetime import datetime
2  import ttkbootstrap as ttk
3  from backend.books import read_book
4  from client import dashboard
5
6  ACCENT_COLOR = "#dc143c"
7
8  def view_borrowed_books(app, member):
9      def go_to_dashboard():
10          main_frame.pack_forget()
11          dashboard.welcome_screen(app, member)
12
13      for widget in app.winfo_children():
14          widget.destroy()
15
16      main_frame = ttk.Frame(app, padding=30)
17      main_frame.pack(fill="both", expand=True)
18
19      left_panel = ttk.Frame(main_frame, padding=20)
20      left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)
21
22      ttk.Label(left_panel, text="Borrowed Books", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
23      ttk.Label(left_panel, text="List of Books You have borrowed from the library.", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
24      back_button = ttk.Button(
25          left_panel,
26          text="Back to Dashboard",
27          command=go_to_dashboard,
28          style="crimson.TButton",
29      )
30      back_button.pack(pady=60)
31
32      right_panel = ttk.Frame(main_frame, padding=20)
33      right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)
34
35      canvas = ttk.Canvas(right_panel, highlightthickness=0)
36      scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
37      scrollable_frame = ttk.Frame(canvas)
38
39      scrollable_frame.bind(
40          "<Configure>",
41          lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
42      )
43      canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
44      canvas.configure(yscrollcommand=scrollbar.set)
45
46      canvas.pack(side="left", fill="both", expand=True)
47      scrollbar.pack(side="right", fill="y")
48
49      style = ttk.Style()
50      style.configure('hover.TFrame', background='#f0f0f0', borderwidth=0)
51
52      if len(member["SKU"]) > 0:
53          for sku, due date in member["SKU"].items():
54              days_late = (datetime.now() - datetime.strptime(due date, "%d/%m/%Y %H:%M:%S")).days
55              day_text = "days late" if days_late > 0 else "days left"
56
57              isbn = str(sku).split("-")[0].strip()
58              book = read_book(isbn)
59
60              book_frame = ttk.Frame(scrollable_frame, padding=10)
61              book_frame.pack(fill="x", pady=(0, 1))
62
63              def on_hover(e, frame=book_frame):
64                  frame.configure(style="hover.TFrame")
65
66              def on_leave(e, frame=book_frame):
67                  frame.configure(style="default.TFrame")
68
69              book_frame.bind("<Enter>", on_hover)
70              book_frame.bind("<Leave>", on_leave)
71
72              ttk.Label(
73                  book_frame,
74                  text=book["Title"],
75                  font=("Helvetica", 14, "bold"),
76                  foreground=ACCENT_COLOR,
77              ).pack(anchor="w", padx=5)
78
79              ttk.Label(
80                  book_frame,
81                  text=f"Status: {abs(days_late)} {day_text}",
82                  font=("Helvetica", 10),
83              ).pack(anchor="w", padx=5)
84      else:
85          ttk.Label(
86              scrollable_frame,
87              text="No book borrowed.",
88              font=("Helvetica", 14, "bold"),
89          ).pack(anchor="w", padx=5)
```

```
client/wishlist.py

 1  from tkinter import messagebox
 2  import ttkbootstrap as ttk
 3  from backend.books import read_book
 4  from backend.members import manage_wishlist
 5  from client import dashboard
 6
 7  ACCENT_COLOR = "#dc143c"
 8
 9
10 def remove_from_wishlist(uid, isbn):
11     res = manage_wishlist(False, uid, isbn)
12     if "Error:" in res:
13         messagebox.showerror("Error", res)
14     else:
15         messagebox.showinfo("Wishlist", f"{isbn} removed from wishlist successfully!")
16         show_main_page()
17
18 def wishlist(app, member):
19     global show_main_page
20     def show_main_page():
21         def go_to_dashboard():
22             main_frame.pack_forget()
23             dashboard.welcome_screen(app, member)
24
25         for widget in app.winfo_children():
26             widget.destroy()
27
28         main_frame = ttk.Frame(app, padding=30)
29         main_frame.pack(fill="both", expand=True)
30
31         left_panel = ttk.Frame(main_frame, padding=20)
32         left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)
33
34         ttk.Label(left_panel, text="Your Wishlist", font=("Century Gothic", 40, "bold"), anchor="center").pack(pady=10)
35         ttk.Label(left_panel, text="Your favourite/saved books!", font=("Arial", 18, "italic"), anchor="center").pack(pady=0)
36         back_button = ttk.Button(
37             left_panel,
38             text="Back to Dashboard",
39             command=go_to_dashboard,
40             style="crimson.TButton",
41         )
42         back_button.pack(pady=60)
43
44         right_panel = ttk.Frame(main_frame, padding=20)
45         right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)
46
47         canvas = ttk.Canvas(right_panel, highlightthickness=0)
48         scrollbar = ttk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
49         scrollable_frame = ttk.Frame(canvas)
50
51         scrollable_frame.bind(
52             "<Configure>",
53             lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
54         )
55         canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
56         canvas.configure(yscrollcommand=scrollbar.set)
57
58         canvas.pack(side="left", fill="both", expand=True)
59         scrollbar.pack(side="right", fill="y")
60
61         style = ttk.Style()
62         style.configure('hover.TFrame', background="#f0f0f0", borderwidth=0)
63
64
65     if len(member["Wishlist"])>0:
66         for isbn in member["Wishlist"]:
67             book = read_book(isbn)
68             book_frame = ttk.Frame(scrollable_frame, padding=10, style="default.TFrame")
69             book_frame.pack(fill="x", pady=(0, 1))
70
71             def on_hover(e, frame=book_frame):
72                 frame.configure(style="hover.TFrame")
73
74             def on_leave(e, frame=book_frame):
75                 frame.configure(style="default.TFrame")
76
77             book_frame.bind("<Enter>", on_hover)
78             book_frame.bind("<Leave>", on_leave)
79
80
81             wishlist_label = ttk.Label(
82                 book_frame,
83                 text=book["Title"],
84                 font=("Helvetica", 14, "bold"),
85                 foreground=ACCENT_COLOR,
86                 anchor="w"
87             )
88             wishlist_label.pack(anchor="w", padx=5)
89             wishlist_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
90
91             isbn_label = ttk.Label(
92                 book_frame,
93                 text=f"ISBN: {book['ISBN']}",
94                 font=("Helvetica", 10),
95                 anchor="w"
96             )
97             isbn_label.pack(anchor="w", padx=5)
98             isbn_label.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
99
100            book_frame.bind("<Button-1>", lambda e, isbn=book["ISBN"]: show_details_page(isbn))
```



```
101     else:
102         ttk.Label(
103             scrollable_frame,
104             text="No book added to wishlist.",
105             font=("Helvetica", 14, "bold"),
106         ).pack(anchor="w", padx=5)
107
108
109     def show_details_page(isbn):
110         for widget in app.winfo_children():
111             widget.destroy()
112
113         details_frame = ttk.Frame(app, padding=30)
114         details_frame.pack(fill="both", expand=True)
115
116         left_panel = ttk.Frame(details_frame, padding=20)
117         left_panel.pack(side="left", fill="both", expand=True, padx=20, pady=20)
118
119         ttk.Label(
120             left_panel,
121             text="Book Details",
122             font="Century Gothic", 40, "bold"),
123             anchor="center",
124         ).pack(pady=10)
125
126         ttk.Label(
127             left_panel,
128             text="Detailed Information about this wishlisted book.",
129             font="Arial", 18, "italic"),
130             anchor="center",
131         ).pack(pady=5)
132
133         right_panel = ttk.Frame(details_frame, padding=20)
134         right_panel.pack(side="right", fill="both", expand=True, padx=20, pady=20)
135
136         canvas = tk.Canvas(right_panel, highlightthickness=0)
137         scrollbar = tk.Scrollbar(right_panel, orient="vertical", command=canvas.yview)
138         scrollable_frame = ttk.Frame(canvas)
139
140         scrollable_frame.bind(
141             "<Configure>",
142             lambda e: canvas.configure(scrollregion=canvas.bbox("all")),
143         )
144         canvas.create_window((0, 0), window=scrollable_frame, anchor="nw")
145         canvas.configure(yscrollcommand=scrollbar.set)
146
147         canvas.pack(side="left", fill="both", expand=True)
148         scrollbar.pack(side="right", fill="y")
149
150         isbn = next((b for b in member["Wishlist"] if b == isbn), None)
151
152         if not isbn:
153             messagebox.showerror("Error", "Book not found.")
154             show_main_page()
155
156         book = read_book(isbn)
157         book_details = [
158             ("Title", book["Title"]),
159             ("ISBN", book["ISBN"]),
160             ("Description", book["Description"]),
161             ("Category", book["Category"]),
162             ("Quantity", book["Quantity"]),
163             ("Author", book["Author"]),
164             ("Publisher", book["Publisher"]),
165             ("Language", book["Language"]),
166         ]
167
168         for label, adetail in book_details:
169             detail_frame = ttk.Frame(scrollable_frame, padding=10)
170             detail_frame.pack(fill="x", pady=(0, 1))
171
172             detail = adetail
173
174             if label == "Quantity":
175                 if int(detail) == 0:
176                     detail = "All copies have been borrowed."
177                 else:
178                     detail = f"{detail} copies in library."
179
180             ttk.Label(
181                 detail_frame,
182                 text=f"{label}: {detail}",
183                 font="Helvetica", 12,
184                 anchor="w",
185                 wraplength=350
186             ).pack(anchor="w", padx=5)
187
188         wishlist_button = ttk.Button(
189             scrollable_frame,
190             text="Remove from Wishlist",
191             command=lambda:remove_from_wishlist(member["UID"], isbn),
192             style="crimson.TButton",
193         )
194         wishlist_button.pack(pady=20)
195
196         back_button = ttk.Button(
197             scrollable_frame,
198             text="Back",
199             command=show_main_page,
200             style="crimson.TButton",
201         )
202         back_button.pack(pady=30)
203
204         show_main_page()
```

Backend



```
backend/books.py

1  from datetime import datetime
2  import os
3  import shutil
4  from barcode import Code128
5  from barcode.writer import ImageWriter
6  from reportlab.lib.pagesizes import letter
7  from reportlab.pdfgen import canvas
8
9
10 Books = []
11
12 def is_valid_isbn(isbn):
13     return len(isbn) == 10 or len(isbn) == 13 and isbn.isdigit()
14
15 def add_book(ISBN, Title, Description, Category, Quantity, Author, Publisher, Language, READD=False, SKU=None):
16     if not ISBN or int(Quantity) < 0 or not is_valid_isbn(ISBN):
17         return "Invalid input for ISBN (must be 10 or 13 digits) or Quantity."
18
19     if READD:
20         for book in Books:
21             if book["ISBN"] == str(SKU).split("-")[0]:
22                 book["Quantity"] += 1
23                 date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
24
25                 if not isinstance(book.get("SKU"), dict):
26                     book["SKU"] = {}
27
28                 book["SKU"].setdefault(SKU, date)
29         return "Book quantity updated successfully"
30
31     sku_dict = {SKU: datetime.now().strftime("%d/%m/%Y %H:%M:%S")}
32     Books.append({
33         "ISBN": str(SKU).split("-")[0],
34         "Title": Title,
35         "Description": Description,
36         "Category": Category,
37         "Quantity": 1,
38         "SKU": sku_dict,
39         "Author": Author,
40         "Publisher": Publisher,
41         "Language": Language,
42     })
43     return "Book re-added as a new entry"
44
45 for book in Books:
46     if book["ISBN"] == ISBN:
47         for i in range(int(Quantity)):
48             sku = f"{ISBN}-{len(book['SKU']) + i + 1}"
49             date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
50             book["SKU"].setdefault(sku, date)
51             book["Quantity"] += int(Quantity)
52         return "Book quantity updated successfully"
53
54     sku_dict = {}
55     for i in range(int(Quantity)):
56         sku = f"{ISBN}-{i + 1}"
57         date = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
58         sku_dict[sku] = date
59
60     Books.append({
61         "ISBN": ISBN,
62         "Title": Title,
63         "Description": Description,
64         "Category": Category,
65         "Quantity": int(Quantity),
66         "SKU": sku_dict,
67         "Author": Author,
68         "Publisher": Publisher,
69         "Language": Language,
70     })
71     return "Book added successfully"
72
73 def generate_barcodes_and_pdf(barcodes):
74     output_dir = "barcodes"
75     pdf_filename = "barcodes.pdf"
76
77     if not os.path.exists(output_dir):
78         os.makedirs(output_dir)
79
80     barcode_images = []
81     for barcode_data in barcodes:
82         barcode = Code128(barcode_data, writer=ImageWriter())
83         image_path = os.path.abspath(os.path.join(output_dir, f'{barcode_data}.png'))
84
85         try:
86             barcode.save(image_path[:-4])
87             barcode_images.append(image_path)
88         except Exception as e:
89             return f"Error saving barcode {barcode_data}: {e}"
90
91     if not barcode_images:
92         return "No barcodes generated."
```



backend/books.py

```
93     try:
94         c = canvas.Canvas(pdf_filename, pagesize=letter)
95         x, y = 50, 750
96
97         for image in barcode_images:
98             if not os.path.exists(image):
99                 continue
100            try:
101                c.drawImage(image, x, y, width=200, height=50)
102                y -= 100
103                if y < 100:
104                    c.showPage()
105                    y = 750
106            except Exception as e:
107                return f"Error adding image {image} to PDF: {e}"
108
109            c.save()
110        except Exception as e:
111            return f"Error creating PDF: {e}"
112
113    try:
114        shutil.rmtree(output_dir)
115    except Exception as e:
116        return "PDF created but error deleting temporary folder."
117
118    return f"Barcodes PDF saved as {os.path.abspath(pdf_filename)}"
119
120 def download_barcodes(book, save_path=None):
121     if not book or "SKU" not in book:
122         return "Error: Invalid book selected or no SKUs found."
123
124     barcodes = list(book["SKU"].keys())
125     if not barcodes:
126         return "Error: No barcodes available for the selected book."
127
128     pdf_result = generate_barcodes_and_pdf(barcodes)
129     if pdf_result.startswith("Error"):
130         return pdf_result
131
132     if save_path:
133         try:
134             shutil.move("barcodes.pdf", save_path)
135             return f"Success: Barcodes saved at {save_path}"
136         except Exception as e:
137             return f"Error saving file: {e}"
138
139     return "Success: Barcodes PDF generated but not saved."
140
141 def update_books(ISBN, Title, Description, Category, Author, Publisher, Language):
142     if not ISBN or not Title or not is_valid_isbn(ISBN):
143         return "Invalid input for ISBN (must be 10 or 13 digits) or Title."
144
145     for book in Books:
146         if book["ISBN"] == ISBN:
147             book["Title"] = Title
148             book["Description"] = Description
149             book["Category"] = Category
150             book["Author"] = Author
151             book["Publisher"] = Publisher
152             book["Language"] = Language
153             return "Book details updated successfully"
154     return "No book found"
155
156 def remove_books(SKU, delete_all=False):
157     if not SKU:
158         return "Invalid SKU input."
159
160     ISBN = (str(SKU).split("-")[0]).strip()
161     sku = str(SKU).strip()
162
163     for book in Books:
164         if book["ISBN"] == ISBN:
165             if delete_all:
166                 book["SKU"].clear()
167                 book["Quantity"] = 0
168             return "All books with ISBN removed successfully"
169
170             if sku in book["SKU"]:
171                 book["SKU"].pop(sku)
172                 if int(book["Quantity"]) > 0:
173                     book["Quantity"] = int(book["Quantity"]) - 1
174                     return "Book removed successfully"
175                 else:
176                     return "Error: Book quantity is already 0, cannot remove further"
177             else:
178                 return f"Error: SKU {sku} not found in the database"
179
180     return f"Error Book with ISBN {ISBN} not found"
181
182 def read_book(ISBN):
183     if not ISBN or not is_valid_isbn(ISBN):
184         return "Invalid ISBN input (must be 10 or 13 digits)."
185
186     for book in Books:
187         if book["ISBN"] == ISBN:
188             return book
189     return "No book found"
```

```
backend/members.py

1 import math
2 import re
3 from backend.books import add_book, Books
4 from datetime import datetime, timedelta
5
6 Members = []
7
8 def is_valid_email(email):
9     return re.match(r"^[^@]+@[^@]+\.[^@]+", email) is not None
10
11 def add_member(Name, Email, Password):
12     if not Name.strip():
13         return "Error: Name is required"
14     if not Email.strip() or not is_valid_email(Email):
15         return "Error: Valid email is required"
16     if not Password.strip() or len(Password) < 6:
17         return "Error: Password must be at least 6 characters"
18     Members.append({
19         "UID": len(Members) + 1,
20         "Name": Name.strip(),
21         "Email": Email.strip(),
22         "Password": Password.strip(),
23         "SKU": {},
24         "Wishlist": [],
25         "JoinedOn": datetime.now().strftime("%d/%m/%Y %H:%M:%S")
26     })
27     return "Member added successfully"
28
29 def update_member_details(UID, NAME, EMAIL, PASSWORD, OLD_PASSWORD):
30     if not UID or not str(UID).isdigit():
31         return "Valid UID is required"
32     if not NAME.strip():
33         return "Name is required"
34     if not EMAIL.strip() or not is_valid_email(EMAIL):
35         return "Valid email is required"
36     if not PASSWORD.strip() or len(PASSWORD) < 6:
37         return "Password must be at least 6 characters"
38     if not OLD_PASSWORD.strip():
39         return "Old password is required"
40
41     for member in Members:
42         if str(member["UID"]) == str(UID):
43             if str(member["Password"]) == str(OLD_PASSWORD):
44                 member["Name"] = NAME.strip()
45                 member["Email"] = EMAIL.strip()
46                 member["Password"] = PASSWORD.strip()
47                 return "Member details updated successfully"
48             return "Old password is incorrect"
49     return "No member found"
50
51 def update_member(UID, SKU, ADD_BOOK, fine_paid=False):
52     if not UID or not str(UID).isdigit():
53         return "Valid UID is required"
54     if not SKU.strip():
55         return "SKU is required"
56     if ADD_BOOK not in [True, False]:
57         return "ADD_BOOK must be either True or False"
58
59     future_date = (datetime.now() + timedelta(days=15)).strftime("%d/%m/%Y %H:%M:%S")
60
61     if ADD_BOOK:
62         for book in Books:
63             if str(book["ISBN"]) == str(SKU).split("-")[0]:
64                 if SKU in book["SKU"]:
65                     for member in Members:
66                         if member["UID"] == int(UID):
67                             member["SKU"][SKU] = future_date
68                             return "Book borrowed successfully"
69             return "The book you are searching for is not available"
70     else:
71         for member in Members:
72             if member["UID"] == int(UID):
73                 if SKU in member["SKU"]:
74                     borrow_date = datetime.strptime(member["SKU"][SKU], "%d/%m/%Y %H:%M:%S")
75                     days_late = (datetime.now() - borrow_date).days
76
77                     total_fine = 0
```

```
78             if days_late > 0:
79                 fine = 5
80                 for day in range(1, days_late + 1):
81                     total_fine += math.ceil(fine)
82                     fine += fine * 0.02
83
84             if not fine_paid:
85                 return f"Days Late: {days_late}\nFine incurred: \u20B9{total_fine}/-."
86
87         member["SKU"].pop(SKU)
88
89         res = add_book(
90             ISBN=str(SKU).split("-")[0],
91             Title="",
92             Description="",
93             Category="",
94             Quantity=1,
95             Author="",
96             Publisher="",
97             Language="",
98             READD=True,
99             SKU=SKU,
100        )
101        if res == "Invalid input for ISBN, Title, or Quantity.":
102            return "Error: Unable to add book to database."
103
104        if total_fine > 0 and fine_paid:
105            return f'Book returned successfully. Fine of \u20B9{total_fine}/- was paid.'
106        return "Book returned successfully. No fine incurred."
107
108    return "Book not borrowed by this member"
109    return "Member not found"
110
111 def remove_member(UUID):
112     if not UUID or not str(UUID).isdigit():
113         return "Valid UID is required"
114     for member in Members:
115         if member["UID"] == int(UUID):
116             Members.remove(member)
117             return "Member removed successfully"
118     return "Member not found"
119
120 def sign_in(Email, Password):
121     if not Email.strip() or not is_valid_email(Email):
122         return "Valid email is required"
123     if not Password.strip():
124         return "Password is required"
125     for member in Members:
126         if member["Email"] == Email.strip() and member["Password"] == Password.strip():
127             return member
128     return "Invalid credentials"
129
130 def read_member(UUID):
131     if not UUID or not str(UUID).isdigit():
132         return "Valid UID is required"
133     for member in Members:
134         if member["UID"] == int(UUID):
135             return member
136     return "No member found"
137
138 def manage_wishlist(ADD, UID, ISBN):
139     if not UID or not str(UID).isdigit() or not ISBN or ADD not in [True, False]:
140         return "Valid parameters required"
141
142     for mem in Members:
143         if str(mem["UID"]) == str(UID):
144             if ADD:
145                 if ISBN not in mem["Wishlist"]:
146                     mem["Wishlist"].append(ISBN)
147                 else:
148                     return "Error: Book already in wishlist"
149             else:
150                 if ISBN in mem["Wishlist"]:
151                     mem["Wishlist"].remove(ISBN)
152                 else:
153                     return "Book not present in wishlist"
154             return "Success"
155
156     return "Error: UID is Invalid"
```

```
backend/shelfing.py
```

```
1 from datetime import datetime
2
3 BookShelf = []
4 DeshelfedBooks = {}
5
6 def categorise(RACK, SHELF, CATEGORY):
7     if not RACK or not SHELF or not CATEGORY:
8         return "Error: Invalid input. RACK, SHELF, and CATEGORY are required."
9
10    while len(BookShelf) < RACK:
11        BookShelf.append([])
12
13    while len(BookShelf[RACK - 1]) < SHELF:
14        BookShelf[RACK - 1].append({})
15
16    category_dict = BookShelf[RACK - 1][SHELF - 1]
17
18    if CATEGORY not in category_dict:
19        category_dict[CATEGORY] = []
20
21    return f"Category '{CATEGORY}' added to rack {RACK}, shelf {SHELF}."
22
23 def shelf(RACK, SHELF, SKU):
24     if not RACK or not SHELF or not SKU:
25         return "Error: Invalid input. RACK, SHELF, and SKU are required."
26
27     if RACK > len(BookShelf) or SHELF > len(BookShelf[RACK - 1]):
28         return "Error: Rack or shelf does not exist."
29
30     category_dict = BookShelf[RACK - 1][SHELF - 1]
31
32     if not category_dict:
33         return "Error: Shelf is uncategorized."
34
35     for category in category_dict:
36         if SKU not in category_dict[category]:
37             category_dict[category].append(SKU)
38         if SKU in DeshelfedBooks:
39             DeshelfedBooks.pop(SKU)
40     return f"Book with SKU '{SKU}' shelved successfully on rack {RACK}, shelf {SHELF} under category '{category}'."
41 return "Error: SKU already exists in this category."
42
43 def search(SKU):
44     if not SKU:
45         return "Error: SKU is required for searching."
46
47     for r, rack in enumerate(BookShelf, start=1):
48         for s, shelf in enumerate(rack, start=1):
49             if not shelf:
50                 continue
51
52             for category, skus in shelf.items():
53                 if SKU in skus:
54                     return f"Book with SKU '{SKU}' is located on rack {r}, shelf {s} under category '{category}'."
55     return f"Error: Book with SKU '{SKU}' not found."
56
57
58 def deshelve(SKU):
59     if not SKU:
60         return "Error: SKU is required for deshelving."
61
62     for r, rack in enumerate(BookShelf, start=1):
63         for s, shelf in enumerate(rack, start=1):
64             if not shelf:
65                 continue
66
67             for category, skus in shelf.items():
68                 if SKU in skus:
69                     skus.remove(SKU)
70                     timestamp = datetime.now().strftime("%d/%m/%Y %H:%M:%S")
71                     DeshelfedBooks[SKU] = f"Rack: {r}, Shelf: {s}, Timestamp: {timestamp}"
72                     return f"Book with SKU '{SKU}' deshveled from rack {r}, shelf {s}."
73     return f"Error: Book with SKU '{SKU}' not found."
```

backend/utils.py

```
1 import threading
2 import numpy
3 from tkinter import messagebox
4 import cv2
5 from pyzbar.pyzbar import decode
6
7 def clear_fields(*vars):
8     for var in vars:
9         var.set("")
10
11 def open_barcode_scanner(sku_var):
12     sku_var.set(None)
13     def get_available_camera():
14         for camera_id in range(10):
15             cap = cv2.VideoCapture(camera_id)
16             if cap.isOpened():
17                 return cap
18             cap.release()
19     return None
20
21 cap = get_available_camera()
22
23 if not cap:
24     messagebox.showerror("Error", "No available camera found.")
25     return
26
27 line_position = 0
28 line_direction = 1
29
30 def stop_scanning():
31     cap.release()
32     cv2.destroyAllWindows()
33
34 def scan_thread():
35     nonlocal line_position, line_direction
36
37     while True:
38         ret, frame = cap.read()
39         if not ret:
40             break
41
42         barcodes = decode(frame)
43
44         for barcode in barcodes:
45             barcode_data = barcode.data.decode('utf-8').strip()
46             sku_var.set(barcode_data)
47
48             rect_points = barcode.polygon
49             if len(rect_points) == 4:
50                 pts = numpy.array(rect_points, dtype=numpy.int32)
51                 cv2.polylines(frame, [pts], True, (0, 0, 255), 2)
52
53                 x, y, w, h = barcode.rect
54                 cv2.putText(frame, barcode_data, (x, y - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2)
55
56             stop_scanning()
57             return
58
59             cv2.line(frame, (0, line_position), (frame.shape[1], line_position), (0, 255, 0), 2)
60             line_position += line_direction
61             if line_position >= frame.shape[0] or line_position <= 0:
62                 line_direction *= -1
63
64             cv2.imshow("Barcode Scanner", frame)
65
66             if cv2.waitKey(1) & 0xFF == ord('q'):
67                 break
68
69             stop_scanning()
70
71     threading.Thread(target=scan_thread, daemon=True).start()
```