Project Code:

```
import tkinter as tk
from tkinter import ttk, messagebox, filedialog
import sqlite3
import os
from ttkbootstrap import Style
from datetime import datetime
from PIL import Image, ImageDraw, ImageFont, ImageTk
import io
import hashlib  # For password hashing

# --- Global Constants and Paths ---
DATABASE_NAME = 'student_records.db'
LOGO_PATH = 'logo.png'
COLLEGE_INFO_PATH = 'college info.png'
COLLEGE_VIEW_PATH = 'collegeview.jpeg'
IDENTITY_CARD_BACKGROUND_PATH = 'identitycard.jpg'

# --- Database Setup and Utilities ---
def hash_password(password):
    """Hashes a password using SHA-256."""
    return hashlib.sha256(password.encode('utf-8')).hexdigest()

def init_db():
    conn = sqlite3.connect(DATABASE_NAME)
    cursor = conn.cursor()
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS users (
            id INTEGER PRIMARY KEY AUTOINCREMENT,
            username TEXT NOT NULL UNIQUE,
            password TEXT NOT NULL
        )
    ''')
    # Insert a default admin user if not exists (for testing)
    cursor.execute("INSERT OR IGNORE INTO users (username, password) VALUES
(?, ?)", ('admin', hash_password('admin')))

    # Create faculties table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS faculties (
            faculty_id INTEGER PRIMARY KEY AUTOINCREMENT,
            faculty_name TEXT NOT NULL UNIQUE
        )
    ''')
```

```
    # Insert sample faculties if not exists
    faculties = [('BCA',), ('BBA',), ('MCA',), ('IBCA',), ('IMCA',)]
    for faculty in faculties:
        cursor.execute("INSERT OR IGNORE INTO faculties (faculty_name) VALUES
(?)", faculty)

    # Create academic_years table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS academic_years (
            year_id INTEGER PRIMARY KEY AUTOINCREMENT,
            year_name TEXT NOT NULL UNIQUE
        )
    ''')
    # Insert sample academic years if not exists
    academic_years = [('First Year',), ('Second Year',), ('Third Year',), ('Fourth Year',),
('Fifth Year',)]
    for year in academic_years:
        cursor.execute("INSERT OR IGNORE INTO academic_years (year_name)
VALUES (?)", year)

    # Create students table with expanded fields
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS students (
            student_id INTEGER PRIMARY KEY AUTOINCREMENT,
            roll_number TEXT UNIQUE NOT NULL,
            name TEXT NOT NULL,
            contact_number TEXT,
            email TEXT,
            address TEXT,
            aadhaar_no TEXT UNIQUE,
            date_of_birth TEXT,
            gender TEXT,
            tenth_percent REAL,
            twelfth_percent REAL,
            blood_group TEXT,
            mother_name TEXT,
            enrollment_status INTEGER DEFAULT 1, -- 1 for Yes, 0 for No
            enrollment_date TEXT NOT NULL,
            course_id INTEGER,
            academic_year_id INTEGER,
            faculty_id INTEGER,
            profile_picture_path TEXT,
            FOREIGN KEY (course_id) REFERENCES courses(course_id),
            FOREIGN KEY (academic_year_id) REFERENCES
academic_years(year_id),
            FOREIGN KEY (faculty_id) REFERENCES faculties(faculty_id)
```

```
    )
''')

# Create courses table (existing, ensure it's compatible)
cursor.execute('''
    CREATE TABLE IF NOT EXISTS courses (
        course_id INTEGER PRIMARY KEY AUTOINCREMENT,
        course_name TEXT NOT NULL UNIQUE,
        course_code TEXT UNIQUE,
        duration TEXT,
        department TEXT
    )
''')
# Insert sample courses if not exists (ensure these match faculties)
courses = [
    ('Computer Applications', 'MCA', '2 Years', 'Computer Science'),
    ('Business Administration', 'MBA', '2 Years', 'Management'),
    ('Science', 'B.Sc', '3 Years', 'Science'),
    ('Computer Applications', 'BCA', '3 Years', 'Computer Science'),
    ('Computer Applications', 'IBCA', '5 Years', 'Computer Science'), # Integrated BCA
    ('Computer Applications', 'IMCA', '5 Years', 'Computer Science')  # Integrated MCA
]
for course_name, course_code, duration, department in courses:
    cursor.execute("INSERT OR IGNORE INTO courses (course_name, course_code, duration, department) VALUES (?, ?, ?, ?)",
                   (course_name, course_code, duration, department))

# Create marks table (existing)
cursor.execute('''
    CREATE TABLE IF NOT EXISTS marks (
        mark_id INTEGER PRIMARY KEY AUTOINCREMENT,
        student_id INTEGER,
        course_id INTEGER,
        subject_name TEXT,
        semester INTEGER,
        marks_obtained REAL,
        max_marks REAL,
        grade TEXT,
        FOREIGN KEY (student_id) REFERENCES students(student_id),
        FOREIGN KEY (course_id) REFERENCES courses(course_id)
    )
''')
```

```
    # Create payments table (existing)
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS payments (
            payment_id INTEGER PRIMARY KEY AUTOINCREMENT,
            student_id INTEGER,
            amount_paid REAL NOT NULL,
            payment_date TEXT NOT NULL,
            payment_type TEXT,
            receipt_number TEXT UNIQUE,
            description TEXT,
            FOREIGN KEY (student_id) REFERENCES students(student_id)
        )
    ''')

    # Create feedback table
    cursor.execute('''
        CREATE TABLE IF NOT EXISTS feedback (
            feedback_id INTEGER PRIMARY KEY AUTOINCREMENT,
            name TEXT,
            email TEXT,
            feedback_text TEXT NOT NULL,
            timestamp TEXT NOT NULL
        )
    ''')

    conn.commit()
    conn.close()

def get_db_connection():
    return sqlite3.connect(DATABASE_NAME)

def load_image(path, size=None):
    try:
        img = Image.open(path)
        if size:
            img = img.resize(size, Image.LANCZOS)
        return ImageTk.PhotoImage(img)
    except FileNotFoundError:
        messagebox.showerror("Image Error", f"Image file not found: {path}")
        return None
    except Exception as e:
        messagebox.showerror("Image Error", f"Error loading image {path}: {e}")
        return None
```

```
# --- Custom Title Bar Class ---
class CustomTitleBar(tk.Frame):
    def __init__(self, parent, title_text, style_obj):
        super().__init__(parent, bg=style_obj.colors.primary) # Use primary color for title bar
        self.parent = parent
        self.style_obj = style_obj # Keep style_obj for colors if needed
        self.pack(side="top", fill="x")

        self.title_label = ttk.Label(self, text=title_text, bootstyle="inverse-primary", font=("Helvetica", 10, "bold"))
        self.title_label.pack(side="left", padx=10, pady=5)

        # Buttons on the right
        self.close_button = ttk.Button(self, text="✕", command=self.parent.destroy, bootstyle="danger", width=3)
        self.close_button.pack(side="right", padx=2, pady=2)

        self.maximize_button = ttk.Button(self, text="□", command=self.toggle_maximize, bootstyle="info", width=3)
        self.maximize_button.pack(side="right", padx=2, pady=2)

        self.minimize_button = ttk.Button(self, text="—", command=self.parent.iconify, bootstyle="info", width=3)
        self.minimize_button.pack(side="right", padx=2, pady=2)

        # Make title bar draggable
        self.bind("<ButtonPress-1>", self.start_move)
        self.bind("<ButtonRelease-1>", self.stop_move)
        self.bind("<B1-Motion>", self.do_move)

        self.x = 0
        self.y = 0
        self.is_maximized = False

    def start_move(self, event):
        self.x = event.x
        self.y = event.y

    def stop_move(self, event):
        self.x = None
        self.y = None
```

```
    def do_move(self, event):
        if self.is_maximized:
            return # Prevent dragging when maximized

        deltax = event.x - self.x
        deltay = event.y - self.y
        x = self.parent.winfo_x() + deltax
        y = self.parent.winfo_y() + deltay
        self.parent.geometry(f"+{x}+{y}")

    def toggle_maximize(self):
        if self.is_maximized:
            self.parent.wm_state('normal')
            self.is_maximized = False
            self.maximize_button.config(text="🗖")
        else:
            self.parent.wm_state('zoomed') # 'zoomed' for full screen without losing taskbar
            self.is_maximized = True
            self.maximize_button.config(text="🗗") # Restore button icon

# --- Registration Window Class ---
class RegistrationWindow:
    def __init__(self, master, login_window_instance):
        self.master = master
        self.login_window_instance = login_window_instance
        self.master.withdraw()

        self.reg_root = tk.Toplevel(master)
        self.reg_root.title("Register New User")
        self.reg_root.geometry("450x500")
        self.reg_root.resizable(False, False)
        self.reg_root.overrideredirect(True)

        self.style = Style(theme="superhero")
        self.title_bar = CustomTitleBar(self.reg_root, "Register New User", self.style)

        self.reg_root.update_idletasks()
        x = self.reg_root.winfo_screenwidth() // 2 - self.reg_root.winfo_width() // 2
        y = self.reg_root.winfo_screenheight() // 2 - self.reg_root.winfo_height() // 2
        self.reg_root.geometry(f"+{x}+{y}")

        self.create_widgets()
        self.reg_root.protocol("WM_DELETE_WINDOW", self.on_reg_window_close)
```

```
    def create_widgets(self):
        main_frame = ttk.Frame(self.reg_root, padding=20)
        main_frame.pack(expand=True, fill="both")

        ttk.Label(main_frame, text="New User Registration", font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=20)

        ttk.Label(main_frame, text="Full Name:", font=("Helvetica", 12)).pack(pady=(10, 5))
        self.fullname_entry = ttk.Entry(main_frame, width=30, font=("Helvetica", 12))
        self.fullname_entry.pack(pady=5)
        self.fullname_entry.focus_set()

        ttk.Label(main_frame, text="User ID:", font=("Helvetica", 12)).pack(pady=(10, 5))
        self.userid_entry = ttk.Entry(main_frame, width=30, font=("Helvetica", 12))
        self.userid_entry.pack(pady=5)

        ttk.Label(main_frame, text="Password:", font=("Helvetica", 12)).pack(pady=5)
        self.password_entry = ttk.Entry(main_frame, width=30, show="*", font=("Helvetica", 12))
        self.password_entry.pack(pady=5)

        ttk.Label(main_frame, text="Confirm Password:", font=("Helvetica", 12)).pack(pady=5)
        self.confirm_password_entry = ttk.Entry(main_frame, width=30, show="*", font=("Helvetica", 12))
        self.confirm_password_entry.pack(pady=5)

        register_button = ttk.Button(main_frame, text="Register", command=self.register_user, bootstyle="success")
        register_button.pack(pady=20)

        back_button = ttk.Button(main_frame, text="Back to Login", command=self.on_reg_window_close, bootstyle="secondary")
        back_button.pack(pady=5)

        self.reg_root.bind('<Return>', lambda event=None: self.register_user())

    def register_user(self):
        fullname = self.fullname_entry.get().strip()
        username = self.userid_entry.get().strip()
        password = self.password_entry.get().strip()
        confirm_password = self.confirm_password_entry.get().strip()
```

```
        if not fullname or not username or not password or not confirm_password:
            messagebox.showwarning("Input Error", "All fields are required.", parent=self.reg_root)
            return

        if password != confirm_password:
            messagebox.showerror("Password Mismatch", "Passwords do not match.", parent=self.reg_root)
            return

        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT * FROM users WHERE username=?", (username,))
            if cursor.fetchone():
                messagebox.showerror("Registration Failed", "User ID already exists. Please choose a different one.", parent=self.reg_root)
                conn.close()
                return
            hashed_pw = hash_password(password)
            cursor.execute("INSERT INTO users (username, password) VALUES (?, ?)", (username, hashed_pw))
            conn.commit()
            messagebox.showinfo("Registration Successful", f"User '{fullname}' registered successfully! You can now log in.", parent=self.reg_root)
            self.on_reg_window_close()
        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"An error occurred during registration: {e}", parent=self.reg_root)
        finally:
            if conn:
                conn.close()

    def on_reg_window_close(self):
        self.reg_root.destroy()
        self.login_window_instance.login_root.deiconify()

# --- Password Update Window ---
class UpdatePasswordWindow:
    def __init__(self, master, login_window_instance):
        self.master = master
        self.login_window_instance = login_window_instance
        self.master.withdraw()

        self.update_root = tk.Toplevel(master)
        self.update_root.title("Update Password")
```

```
        self.update_root.geometry("400x350")
        self.update_root.resizable(False, False)
        self.update_root.overrideredirect(True)

        self.style = Style(theme="superhero")
        self.title_bar = CustomTitleBar(self.update_root, "Update Password", self.style)

        self.update_root.update_idletasks()
        x = self.update_root.winfo_screenwidth() // 2 - self.update_root.winfo_width() //
2
        y = self.update_root.winfo_screenheight() // 2 - self.update_root.winfo_height() //
2
        self.update_root.geometry(f"+{x}+{y}")

        self.create_widgets()
        self.update_root.protocol("WM_DELETE_WINDOW",
self.on_update_window_close)

    def create_widgets(self):
        main_frame = ttk.Frame(self.update_root, padding=20)
        main_frame.pack(expand=True, fill="both")

        ttk.Label(main_frame, text="Update Password", font=("Helvetica", 16, "bold"),
bootstyle="primary").pack(pady=20)

        ttk.Label(main_frame, text="User ID:", font=("Helvetica", 12)).pack(pady=(10,
5))
        self.userid_entry = ttk.Entry(main_frame, width=30, font=("Helvetica", 12))
        self.userid_entry.pack(pady=5)
        self.userid_entry.focus_set()

        ttk.Label(main_frame, text="Old Password:", font=("Helvetica",
12)).pack(pady=5)
        self.old_password_entry = ttk.Entry(main_frame, width=30, show="*",
font=("Helvetica", 12))
        self.old_password_entry.pack(pady=5)

        ttk.Label(main_frame, text="New Password:", font=("Helvetica",
12)).pack(pady=5)
        self.new_password_entry = ttk.Entry(main_frame, width=30, show="*",
font=("Helvetica", 12))
        self.new_password_entry.pack(pady=5)

        ttk.Label(main_frame, text="Confirm New Password:", font=("Helvetica",
12)).pack(pady=5)
```

```
        self.confirm_new_password_entry = ttk.Entry(main_frame, width=30, show="*", font=("Helvetica", 12))
        self.confirm_new_password_entry.pack(pady=5)

        update_button = ttk.Button(main_frame, text="Update Password", command=self.update_password, bootstyle="success")
        update_button.pack(pady=20)

        back_button = ttk.Button(main_frame, text="Back to Login", command=self.on_update_window_close, bootstyle="secondary")
        back_button.pack(pady=5)

        self.update_root.bind('<Return>', lambda event=None: self.update_password())

    def update_password(self):
        username = self.userid_entry.get().strip()
        old_password = self.old_password_entry.get().strip()
        new_password = self.new_password_entry.get().strip()
        confirm_new_password = self.confirm_new_password_entry.get().strip()

        if not username or not old_password or not new_password or not confirm_new_password:
            messagebox.showwarning("Input Error", "All fields are required.", parent=self.update_root)
            return

        if new_password != confirm_new_password:
            messagebox.showerror("Password Mismatch", "New passwords do not match.", parent=self.update_root)
            return

        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            cursor.execute("SELECT password FROM users WHERE username=?", (username,))
            row = cursor.fetchone()
            if not row or hash_password(old_password) != row[0]:
                messagebox.showerror("Authentication Failed", "User ID or old password is incorrect.", parent=self.update_root)
                return
            cursor.execute("UPDATE users SET password=? WHERE username=?", (hash_password(new_password), username))
            conn.commit()
            messagebox.showinfo("Password Updated", "Password updated successfully! Please login with your new password.", parent=self.update_root)
```

```
            self.on_update_window_close()
        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}",
parent=self.update_root)
        finally:
            if conn:
                conn.close()

    def on_update_window_close(self):
        self.update_root.destroy()
        self.login_window_instance.login_root.deiconify()

# --- Login Window Class ---
class LoginWindow:
    def __init__(self, master):
        self.master = master
        self.master.withdraw()

        self.login_root = tk.Toplevel(master)
        self.login_root.title("Login - Student Database Management System")
        self.login_root.geometry("450x450")
        self.login_root.resizable(False, False)
        self.login_root.overrideredirect(True)

        self.style = Style(theme="superhero")
        self.title_bar = CustomTitleBar(self.login_root, "Student Database Management
System", self.style)

        self.login_root.update_idletasks()
        x = self.login_root.winfo_screenwidth() // 2 - self.login_root.winfo_width() // 2
        y = self.login_root.winfo_screenheight() // 2 - self.login_root.winfo_height() // 2
        self.login_root.geometry(f"+{x}+{y}")

        self.create_widgets()
        self.login_root.protocol("WM_DELETE_WINDOW",
self.on_login_window_close)

    def create_widgets(self):
        main_frame = ttk.Frame(self.login_root, padding=20)
        main_frame.pack(expand=True, fill="both")

        ttk.Label(main_frame, text="System Login", font=("Helvetica", 18, "bold"),
bootstyle="primary").pack(pady=20)

        ttk.Label(main_frame, text="User ID:", font=("Helvetica", 12)).pack(pady=(10,
5))
```

```
        self.username_entry = ttk.Entry(main_frame, width=35, font=("Helvetica", 12))
        self.username_entry.pack(pady=5)
        self.username_entry.focus_set()

        ttk.Label(main_frame, text="Password:", font=("Helvetica", 12)).pack(pady=5)
        self.password_entry = ttk.Entry(main_frame, width=35, show="*",
font=("Helvetica", 12))
        self.password_entry.pack(pady=5)

        login_button = ttk.Button(main_frame, text="Login",
command=self.authenticate_user, bootstyle="success", width=20)
        login_button.pack(pady=20)

        register_button = ttk.Button(main_frame, text="Register New User",
command=self.open_registration_window, bootstyle="info", width=20)
        register_button.pack(pady=5)

        update_pw_button = ttk.Button(main_frame, text="Update Password",
command=self.open_update_password_window, bootstyle="warning", width=20)
        update_pw_button.pack(pady=5)

        self.login_root.bind('<Return>', lambda event=None: self.authenticate_user())

    def authenticate_user(self):
        username = self.username_entry.get()
        password = self.password_entry.get()

        if not username or not password:
            messagebox.showwarning("Login Error", "Please enter both User ID and
password.", parent=self.login_root)
            return

        try:
            conn = get_db_connection()
            cursor = conn.cursor()
            hashed_pw = hash_password(password)
            cursor.execute("SELECT * FROM users WHERE username=? AND
password=?", (username, hashed_pw))
            user = cursor.fetchone()
            if user:
                messagebox.showinfo("Login Successful", "Welcome to the Student
Database Management System!", parent=self.login_root)
                self.login_root.destroy()
                self.master.deiconify()
                MainApplication(self.master)
```

```
 else:
            messagebox.showerror("Login Failed", "Invalid User ID or password.", parent=self.login_root)
        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"An error occurred: {e}", parent=self.login_root)
        finally:
            if conn:
                conn.close()

    def open_registration_window(self):
        self.login_root.withdraw()
        RegistrationWindow(self.master, self)

    def open_update_password_window(self):
        self.login_root.withdraw()
        UpdatePasswordWindow(self.master, self)

    def on_login_window_close(self):
        if messagebox.askokcancel("Exit", "Do you want to exit the application?", parent=self.login_root):
            self.master.destroy()

# --- Main Application Class ---
class MainApplication:
    def __init__(self, master):
        self.master = master
        self.master.title("Student Database Management System")
        self.master.geometry("1200x800") # Adjust size as needed
        self.master.overrideredirect(True) # Remove default title bar for main window
        self.master.resizable(True, True) # Allow resizing by mouse arrow

        self.style = Style(theme="superhero") # Ensure consistent theme

        # Custom Title Bar for Main Application Window
        self.title_bar = CustomTitleBar(self.master, "Student Database Management Application", self.style)

        # Center the main window (after login)
        self.master.update_idletasks()
        x = self.master.winfo_screenwidth() // 2 - self.master.winfo_width() // 2
        y = self.master.winfo_screenheight() // 2 - self.master.winfo_height() // 2
        self.master.geometry(f"+{x}+{y}")
```

```
# Load images for the application
        self.app_logo = load_image(LOGO_PATH, size=(50, 50))
        self.college_info_img = load_image(COLLEGE_INFO_PATH, size=(600, 150))
# Adjusted size for home page
        self.college_view_bg = None # Will be loaded dynamically and resized
        self.load_college_view_background() # Load it initially


        self.create_main_widgets()

    def load_college_view_background(self):
        """Loads and sets the college view background image."""
        if os.path.exists(COLLEGE_VIEW_PATH):
            original_image = Image.open(COLLEGE_VIEW_PATH)
            # Resize it to fit the current window size
            win_width = self.master.winfo_width()
            win_height = self.master.winfo_height() - self.title_bar.winfo_height()
            if win_width > 0 and win_height > 0:
                resized_image = original_image.resize((win_width, win_height),
Image.LANCZOS)
                self.college_view_bg = ImageTk.PhotoImage(resized_image)
            else:
                # Default size if window size is not yet determined
                self.college_view_bg = ImageTk.PhotoImage(original_image.resize((1200,
800 - self.title_bar.winfo_height()), Image.LANCZOS))
        else:
            messagebox.showerror("Image Error", f"College View Image file not found:
{COLLEGE_VIEW_PATH}")


    def create_main_widgets(self):
        # Create a main frame to hold the notebook (tabs)
        # We need a canvas to hold the background image and other widgets transparently
        self.canvas = tk.Canvas(self.master, bd=0, highlightthickness=0)
        self.canvas.place(x=0, y=self.title_bar.winfo_height(), relwidth=1, relheight=1)
        self.canvas.bind('<Configure>', self._on_canvas_resize) # Bind resize event to
canvas

        if self.college_view_bg:
            self.bg_image_id = self.canvas.create_image(0, 0,
image=self.college_view_bg, anchor="nw")
```

```
        # Create a frame inside the canvas to hold the notebook
        self.content_frame = ttk.Frame(self.canvas, padding=10, style="Transparent.TFrame") # Transparent frame
        self.content_window_id = self.canvas.create_window((0, 0), window=self.content_frame, anchor="nw", width=self.master.winfo_width(), height=self.master.winfo_height() - self.title_bar.winfo_height())


        # Apply a transparent style to the frame if background image is used
        # We need to configure styles for all relevant widgets to make them somewhat transparent
        # or have a contrasting background. For ttkbootstrap, setting background is tricky for full transparency.
        # We can set the background to match the theme's default background, which might be slightly transparent
        # or at least not opaque white.
        self.style.configure("Transparent.TFrame", background=self.style.colors.bg)
        self.style.map("Transparent.TFrame", background=[("active", self.style.colors.bg)])
        self.style.configure("Transparent.TLabel", background=self.style.colors.bg)
        self.style.configure("Transparent.TLabelframe", background=self.style.colors.bg)
        self.style.configure("Transparent.TLabelframe.Label", background=self.style.colors.bg)

        # Add app logo to the main application window (e.g., top left of content area)
        if self.app_logo:
            logo_label = ttk.Label(self.content_frame, image=self.app_logo, bootstyle="inverse-primary")
            logo_label.pack(side="top", anchor="nw", padx=10, pady=5)

        self.notebook = ttk.Notebook(self.content_frame)
        self.notebook.pack(expand=True, fill="both", padx=10, pady=10)

        # Tab 0: Home Page
        home_frame = ttk.Frame(self.notebook, style="Transparent.TFrame") # Apply transparent style
        self.notebook.add(home_frame, text="Home")
        self.setup_home_tab(home_frame)

        # Tab 1: Student Management
        student_frame = ttk.Frame(self.notebook)
        self.notebook.add(student_frame, text="Student Management")
        self.setup_student_management_tab(student_frame)
```

```
# Tab 2: Reports
        reports_frame = ttk.Frame(self.notebook)
        self.notebook.add(reports_frame, text="Reports")
        self.setup_reports_tab(reports_frame)

        # Tab 3: ID Card Generation
        id_card_frame = ttk.Frame(self.notebook)
        self.notebook.add(id_card_frame, text="ID Card Generation")
        self.setup_id_card_tab(id_card_frame)

        # Tab 4: Receipt Generation
        receipt_frame = ttk.Frame(self.notebook)
        self.notebook.add(receipt_frame, text="Receipt Generation")
        self.setup_receipt_tab(receipt_frame)

        # Tab 5: Analytics & Insights
        analytics_frame = ttk.Frame(self.notebook)
        self.notebook.add(analytics_frame, text="Analytics & Insights")
        self.setup_analytics_tab(analytics_frame)

        # Tab 6: Feedback
        feedback_frame = ttk.Frame(self.notebook)
        self.notebook.add(feedback_frame, text="Feedback")
        self.setup_feedback_tab(feedback_frame)

    def _on_canvas_resize(self, event):
        """Resizes the content frame to fit the new canvas size (no background image)."""
        new_width = event.width
        new_height = event.height
        self.canvas.coords(self.content_window_id, 0, 0)
        self.canvas.itemconfigure(self.content_window_id, width=new_width,
height=new_height)
        if self.college_view_bg:
            self.canvas.tag_lower(self.bg_image_id)

    # --- Home Tab ---
    def setup_home_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Welcome to Student Database Management
System", font=("Helvetica", 20, "bold"), bootstyle="primary",
style="Transparent.TLabel").pack(pady=20)
```

```
        if self.college_info_img:
            ttk.Label(parent_frame, image=self.college_info_img, style="Transparent.TLabel").pack(pady=10)
        else:
            ttk.Label(parent_frame, text="College Info Image Not Loaded", font=("Helvetica", 14), bootstyle="danger", style="Transparent.TLabel").pack(pady=10)

        # College Introduction Details
        college_details_frame = ttk.LabelFrame(parent_frame, text="About Saraswati College, Shegaon", padding=15, bootstyle="info", style="Transparent.TLabelframe")
        college_details_frame.pack(pady=20, padx=50, fill="x", expand=False)

        college_info = """
Saraswati College, Shegaon is affiliated with Sant Gadge Baba Amaravati University, Amaravati.
Established in 2009, as a premier Techno-Management institute in the vicinity-Vidarbha, Maharashtra.
Gaulkhed Road, Shegaon Dist:- Buldhana, State:-Maharashtra (INDIA) Pin: 444 203.

Contact Information:
MCA: +919356970144
BCA: +917666612738
BBA: +919322120165
Email: enquiry@saraswaticollege.edu.in , principal@saraswaticollege.edu.in
        """
        ttk.Label(college_details_frame, text=college_info, font=("Helvetica", 11), justify="left", style="Transparent.TLabel").pack(pady=10, padx=10)

        ttk.Label(parent_frame, text="Your comprehensive solution for managing student records efficiently.", font=("Helvetica", 14), bootstyle="info", style="Transparent.TLabel").pack(pady=10)
        ttk.Label(parent_frame, text="Navigate through the tabs above to access different functionalities.", font=("Helvetica", 12), style="Transparent.TLabel").pack(pady=5)

        # Add developer name
        ttk.Label(parent_frame, text="@developed by Rushikesh Atole and Team", font=("Helvetica", 10, "italic"), bootstyle="secondary", style="Transparent.TLabel").pack(side="bottom", pady=10)
```

```
    # --- Student Management Tab ---
    def setup_student_management_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Student Record Management (CRUD Operations)", font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=10)

        # Input fields for student details
        input_frame = ttk.LabelFrame(parent_frame, text="Student Details", padding=10, bootstyle="info")
        input_frame.pack(pady=10, padx=10, fill="x", expand=False)

        # Grid layout for input fields
        input_frame.columnconfigure(1, weight=1) # Make entry columns expandable
        input_frame.columnconfigure(3, weight=1)

        row = 0
        ttk.Label(input_frame, text="Roll No:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_roll_entry = ttk.Entry(input_frame, width=30)
        self.student_roll_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="Name:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_name_entry = ttk.Entry(input_frame, width=30)
        self.student_name_entry.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
        ttk.Label(input_frame, text="Contact No:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_contact_entry = ttk.Entry(input_frame, width=30)
        self.student_contact_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="Email:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_email_entry = ttk.Entry(input_frame, width=30)
        self.student_email_entry.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
        ttk.Label(input_frame, text="Address:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_address_entry = ttk.Entry(input_frame, width=30)
        self.student_address_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")
```

```
        ttk.Label(input_frame, text="Aadhaar No:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_aadhaar_entry = ttk.Entry(input_frame, width=30)
        self.student_aadhaar_entry.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
        ttk.Label(input_frame, text="Date of Birth (YYYY-MM-DD):").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_dob_entry = ttk.Entry(input_frame, width=30)
        self.student_dob_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="Gender:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_gender_combobox = ttk.Combobox(input_frame, values=["Male", "Female", "Other"])
        self.student_gender_combobox.grid(row=row, column=3, padx=5, pady=2, sticky="ew")
        self.student_gender_combobox.set("Male") # Default value

        row += 1
        ttk.Label(input_frame, text="10th %:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_tenth_entry = ttk.Entry(input_frame, width=30)
        self.student_tenth_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="12th %:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_twelfth_entry = ttk.Entry(input_frame, width=30)
        self.student_twelfth_entry.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
        ttk.Label(input_frame, text="Blood Group:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_blood_group_entry = ttk.Entry(input_frame, width=30)
        self.student_blood_group_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="Mother's Name:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_mother_name_entry = ttk.Entry(input_frame, width=30)
        self.student_mother_name_entry.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
```

```
        ttk.Label(input_frame, text="Enrollment Date (YYYY-MM-DD):").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_enrollment_date_entry = ttk.Entry(input_frame, width=30)
        self.student_enrollment_date_entry.grid(row=row, column=1, padx=5, pady=2, sticky="ew")
        self.student_enrollment_date_entry.insert(0, datetime.now().strftime("%Y-%m-%d"))

        ttk.Label(input_frame, text="Enrollment Status:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_enrollment_status_combobox = ttk.Combobox(input_frame, values=["Yes", "No"])
        self.student_enrollment_status_combobox.grid(row=row, column=3, padx=5, pady=2, sticky="ew")
        self.student_enrollment_status_combobox.set("Yes") # Default value

        row += 1
        ttk.Label(input_frame, text="Course:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_course_combobox = ttk.Combobox(input_frame, values=self._get_course_names())
        self.student_course_combobox.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(input_frame, text="Academic Year:").grid(row=row, column=2, padx=5, pady=2, sticky="w")
        self.student_academic_year_combobox = ttk.Combobox(input_frame, values=self._get_academic_year_names())
        self.student_academic_year_combobox.grid(row=row, column=3, padx=5, pady=2, sticky="ew")

        row += 1
        ttk.Label(input_frame, text="Faculty:").grid(row=row, column=0, padx=5, pady=2, sticky="w")
        self.student_faculty_combobox = ttk.Combobox(input_frame, values=self._get_faculty_names())
        self.student_faculty_combobox.grid(row=row, column=1, padx=5, pady=2, sticky="ew")

        # Profile Picture Upload
        profile_pic_frame = ttk.LabelFrame(input_frame, text="Profile Picture", padding=5)
        profile_pic_frame.grid(row=0, column=4, rowspan=8, padx=10, pady=5, sticky="nsew") # Adjusted rowspan
        self.profile_pic_label = ttk.Label(profile_pic_frame, text="No Image", anchor="center")
```

```
        self.profile_pic_label.pack(fill="both", expand=True)
        upload_button = ttk.Button(profile_pic_frame, text="Upload Image", command=self.upload_profile_picture)
        upload_button.pack(pady=5)
        self.profile_picture_path = "" # Store the path to the uploaded image

        # CRUD Buttons
        button_frame = ttk.Frame(parent_frame, padding=10)
        button_frame.pack(pady=10, padx=10, fill="x", expand=False)
        ttk.Button(button_frame, text="Add Student", command=self.add_student, bootstyle="success").pack(side="left", padx=5)
        ttk.Button(button_frame, text="Update Student", command=self.update_student, bootstyle="info").pack(side="left", padx=5)
        ttk.Button(button_frame, text="Delete Student", command=self.delete_student, bootstyle="danger").pack(side="left", padx=5)
        ttk.Button(button_frame, text="Clear Fields", command=self.clear_student_fields, bootstyle="secondary").pack(side="left", padx=5)

        # Search and Display
        search_frame = ttk.LabelFrame(parent_frame, text="Search & View Students", padding=10, bootstyle="primary")
        search_frame.pack(pady=10, padx=10, fill="both", expand=True)
        ttk.Label(search_frame, text="Search by Roll No/Name:").pack(side="left", padx=5)
        self.search_entry = ttk.Entry(search_frame, width=40)
        self.search_entry.pack(side="left", padx=5, fill="x", expand=True)
        ttk.Button(search_frame, text="Search", command=self.search_students, bootstyle="primary").pack(side="left", padx=5)
        ttk.Button(search_frame, text="Refresh", command=lambda: self.display_students(), bootstyle="primary").pack(side="left", padx=5)

        # Student List Treeview
        self.student_tree = ttk.Treeview(search_frame, columns=(
            "ID", "Roll No", "Name", "Contact", "Email", "Address", "Aadhaar",
            "DOB", "Gender", "10th%", "12th%", "Blood Group", "Mother",
            "Enroll Status", "Enroll Date", "Course", "Acad Year", "Faculty"
        ), show="headings", bootstyle="primary")

        # Define column headings
        for col in self.student_tree["columns"]:
            self.student_tree.heading(col, text=col)
            self.student_tree.column(col, width=100, anchor="center")
        self.student_tree.column("ID", width=40)
        self.student_tree.column("Roll No", width=80)
        self.student_tree.column("Name", width=120)
```

```
        self.student_tree.column("Contact", width=100)
        self.student_tree.column("Email", width=150)
        self.student_tree.column("Address", width=150)
        self.student_tree.column("Aadhaar", width=100)
        self.student_tree.column("DOB", width=90)
        self.student_tree.column("Gender", width=70)
        self.student_tree.column("10th%", width=60)
        self.student_tree.column("12th%", width=60)
        self.student_tree.column("Blood Group", width=80)
        self.student_tree.column("Mother", width=100)
        self.student_tree.column("Enroll Status", width=80)
        self.student_tree.column("Enroll Date", width=90)
        self.student_tree.column("Course", width=120)
        self.student_tree.column("Acad Year", width=90)
        self.student_tree.column("Faculty", width=100)
        self.student_tree.pack(pady=10, fill="both", expand=True)

        # Scrollbar for treeview
        scrollbar = ttk.Scrollbar(search_frame, orient="vertical",
command=self.student_tree.yview)
        scrollbar.pack(side="right", fill="y")
        self.student_tree.configure(yscrollcommand=scrollbar.set)

        hscrollbar = ttk.Scrollbar(search_frame, orient="horizontal",
command=self.student_tree.xview)
        hscrollbar.pack(side="bottom", fill="x")
        self.student_tree.configure(xscrollcommand=hscrollbar.set)

        self.student_tree.bind("<<TreeviewSelect>>", self.load_selected_student)
        self.display_students() # Initial display

    def _get_course_names(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT course_name FROM courses")
        courses = [row[0] for row in cursor.fetchall()]
        conn.close()
        return courses

    def _get_academic_year_names(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT year_name FROM academic_years")
        years = [row[0] for row in cursor.fetchall()]
        conn.close()
        return years
```

```
    def _get_faculty_names(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT faculty_name FROM faculties")
        faculties = [row[0] for row in cursor.fetchall()]
        conn.close()
        return faculties

    def upload_profile_picture(self):
        file_path = filedialog.askopenfilename(
            title="Select Profile Picture",
            filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif")]
        )
        if file_path:
            try:
                self.profile_picture_path = file_path
                img = Image.open(file_path)
                img = img.resize((100, 100), Image.LANCZOS)
                self.profile_pic_display = ImageTk.PhotoImage(img)
                self.profile_pic_label.config(image=self.profile_pic_display, text="")
                self.profile_pic_label.image = self.profile_pic_display
            except Exception as e:
                messagebox.showerror("Image Error", f"Failed to load image: {e}")
                self.profile_picture_path = ""
                self.profile_pic_label.config(image="", text="No Image")
        else:
            self.profile_picture_path = ""
            self.profile_pic_label.config(image="", text="No Image")

    def add_student(self):
        roll_number = self.student_roll_entry.get().strip()
        name = self.student_name_entry.get().strip()
        contact_number = self.student_contact_entry.get().strip()
        email = self.student_email_entry.get().strip()
        address = self.student_address_entry.get().strip()
        aadhaar_no = self.student_aadhaar_entry.get().strip()
        date_of_birth = self.student_dob_entry.get().strip()
        gender = self.student_gender_combobox.get().strip()
        tenth_percent = self.student_tenth_entry.get().strip()
        twelfth_percent = self.student_twelfth_entry.get().strip()
        blood_group = self.student_blood_group_entry.get().strip()
        mother_name = self.student_mother_name_entry.get().strip()
        enrollment_status = 1 if self.student_enrollment_status_combobox.get() == "Yes"
else 0
        enrollment_date = self.student_enrollment_date_entry.get().strip()
```

```
        course_name = self.student_course_combobox.get().strip()
        academic_year_name = self.student_academic_year_combobox.get().strip()
        faculty_name = self.student_faculty_combobox.get().strip()
        profile_picture_path = self.profile_picture_path

        if not all([roll_number, name, enrollment_date, course_name, academic_year_name, faculty_name]):
            messagebox.showwarning("Input Error", "Roll Number, Name, Enrollment Date, Course, Academic Year, and Faculty are required fields.")
            return

        # Validate numeric fields
        try:
            tenth_percent = float(self.student_tenth_entry.get().strip()) if self.student_tenth_entry.get().strip() else None
            twelfth_percent = float(self.student_twelfth_entry.get().strip()) if self.student_twelfth_entry.get().strip() else None
        except ValueError:
            messagebox.showerror("Input Error", "10th % and 12th % must be numbers.")
            return

        # Validate date fields
        try:
            if self.student_dob_entry.get().strip():
                datetime.strptime(self.student_dob_entry.get().strip(), "%Y-%m-%d")
            datetime.strptime(self.student_enrollment_date_entry.get().strip(), "%Y-%m-%d")
        except ValueError:
            messagebox.showerror("Input Error", "Date fields must be in YYYY-MM-DD format.")
            return

        conn = get_db_connection()
        cursor = conn.cursor()
        try:
            # Get course_id
            cursor.execute("SELECT course_id FROM courses WHERE course_name=?", (course_name,))
            course_id = cursor.fetchone()
            if not course_id:
                messagebox.showerror("Error", f"Course '{course_name}' not found.")
                return
            course_id = course_id[0]

            # Get academic_year_id
```

```
            cursor.execute("SELECT year_id FROM academic_years WHERE
year_name=?", (academic_year_name,))
            academic_year_id = cursor.fetchone()
            if not academic_year_id:
                messagebox.showerror("Error", f"Academic Year '{academic_year_name}'
not found.")
                return
            academic_year_id = academic_year_id[0]

            # Get faculty_id
            cursor.execute("SELECT faculty_id FROM faculties WHERE
faculty_name=?", (faculty_name,))
            faculty_id = cursor.fetchone()
            if not faculty_id:
                messagebox.showerror("Error", f"Faculty '{faculty_name}' not found.")
                return
            faculty_id = faculty_id[0]

            cursor.execute("""
                INSERT INTO students (
                    roll_number, name, contact_number, email, address, aadhaar_no,
                    date_of_birth, gender, tenth_percent, twelfth_percent, blood_group,
                    mother_name, enrollment_status, enrollment_date, course_id,
                    academic_year_id, faculty_id, profile_picture_path
                ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
            """, (
                roll_number, name, contact_number, email, address, aadhaar_no,
                date_of_birth, gender, tenth_percent, twelfth_percent, blood_group,
                mother_name, enrollment_status, enrollment_date, course_id,
                academic_year_id, faculty_id, profile_picture_path
            ))
            conn.commit()
            messagebox.showinfo("Success", "Student added successfully!")
            self.clear_student_fields()
            self.display_students()
        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Roll Number or Aadhaar Number already
exists.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {e}")
        finally:
            if conn:
                conn.close()

    def update_student(self):
        selected_item = self.student_tree.focus()
```

```
        if not selected_item:
            messagebox.showwarning("No Selection", "Please select a student to update.")
            return

        student_id = self.student_tree.item(selected_item, "values")[0]

        roll_number = self.student_roll_entry.get().strip()
        name = self.student_name_entry.get().strip()
        contact_number = self.student_contact_entry.get().strip()
        email = self.student_email_entry.get().strip()
        address = self.student_address_entry.get().strip()
        aadhaar_no = self.student_aadhaar_entry.get().strip()
        date_of_birth = self.student_dob_entry.get().strip()
        gender = self.student_gender_combobox.get().strip()
        tenth_percent = self.student_tenth_entry.get().strip()
        twelfth_percent = self.student_twelfth_entry.get().strip()
        blood_group = self.student_blood_group_entry.get().strip()
        mother_name = self.student_mother_name_entry.get().strip()
        enrollment_status = 1 if self.student_enrollment_status_combobox.get() == "Yes"
else 0
        enrollment_date = self.student_enrollment_date_entry.get().strip()
        course_name = self.student_course_combobox.get().strip()
        academic_year_name = self.student_academic_year_combobox.get().strip()
        faculty_name = self.student_faculty_combobox.get().strip()
        profile_picture_path = self.profile_picture_path

        if not all([roll_number, name, enrollment_date, course_name,
academic_year_name, faculty_name]):
            messagebox.showwarning("Input Error", "Roll Number, Name, Enrollment
Date, Course, Academic Year, and Faculty are required fields.")
            return

        try:
            tenth_percent = float(self.student_tenth_entry.get().strip()) if
self.student_tenth_entry.get().strip() else None
            twelfth_percent = float(self.student_twelfth_entry.get().strip()) if
self.student_twelfth_entry.get().strip() else None
        except ValueError:
            messagebox.showerror("Input Error", "10th % and 12th % must be numbers.")
            return
        try:
            if self.student_dob_entry.get().strip():
                datetime.strptime(self.student_dob_entry.get().strip(), "%Y-%m-%d")
            datetime.strptime(self.student_enrollment_date_entry.get().strip(),    "%Y-%m-
%d")
        except ValueError:
```

```
            messagebox.showerror("Input Error", "Date fields must be in YYYY-MM-DD format.")
            return
        conn = get_db_connection()
        cursor = conn.cursor()
        try:
            # Get course_id
            cursor.execute("SELECT course_id FROM courses WHERE course_name=?", (course_name,))
            course_id = cursor.fetchone()
            if not course_id:
                messagebox.showerror("Error", f"Course '{course_name}' not found.")
                return
            course_id = course_id[0]

            # Get academic_year_id
            cursor.execute("SELECT year_id FROM academic_years WHERE year_name=?", (academic_year_name,))
            academic_year_id = cursor.fetchone()
            if not academic_year_id:
                messagebox.showerror("Error", f"Academic Year '{academic_year_name}' not found.")
                return
            academic_year_id = academic_year_id[0]

            # Get faculty_id
            cursor.execute("SELECT faculty_id FROM faculties WHERE faculty_name=?", (faculty_name,))
            faculty_id = cursor.fetchone()
            if not faculty_id:
                messagebox.showerror("Error", f"Faculty '{faculty_name}' not found.")
                return
            faculty_id = faculty_id[0]

            cursor.execute("""
                UPDATE students SET
                    roll_number=?, name=?, contact_number=?, email=?, address=?, aadhaar_no=?,
                    date_of_birth=?, gender=?, tenth_percent=?, twelfth_percent=?, blood_group=?,
                    mother_name=?, enrollment_status=?, enrollment_date=?, course_id=?,
                    academic_year_id=?, faculty_id=?, profile_picture_path=?
                WHERE student_id=?
            """, (
                roll_number, name, contact_number, email, address, aadhaar_no,
                date_of_birth, gender, tenth_percent, twelfth_percent, blood_group,
```

```
                mother_name, enrollment_status, enrollment_date, course_id,
                academic_year_id, faculty_id, profile_picture_path, student_id
            ))
            conn.commit()
            messagebox.showinfo("Success", "Student updated successfully!")
            self.clear_student_fields()
            self.display_students()
        except sqlite3.IntegrityError:
            messagebox.showerror("Error", "Roll Number or Aadhaar Number already exists for another student.")
        except Exception as e:
            messagebox.showerror("Error", f"An error occurred: {e}")
        finally:
            if conn:
                conn.close()

    def delete_student(self):
        selected_item = self.student_tree.focus()
        if not selected_item:
            messagebox.showwarning("No Selection", "Please select a student to delete.")
            return

        student_id = self.student_tree.item(selected_item, "values")[0]
        name = self.student_tree.item(selected_item, "values")[2] # Get name for confirmation

        if messagebox.askyesno("Confirm Delete", f"Are you sure you want to delete student: {name} (ID: {student_id})?"):
            conn = get_db_connection()
            cursor = conn.cursor()
            try:
                cursor.execute("DELETE FROM students WHERE student_id=?", (student_id,))
                conn.commit()
                messagebox.showinfo("Success", "Student deleted successfully!")
                self.clear_student_fields()
                self.display_students()
            except Exception as e:
                messagebox.showerror("Error", f"An error occurred: {e}")
            finally:
                if conn:
                    conn.close()

    def clear_student_fields(self):
        self.student_roll_entry.delete(0, tk.END)
        self.student_name_entry.delete(0, tk.END)
```

```
        self.student_contact_entry.delete(0, tk.END)
        self.student_email_entry.delete(0, tk.END)
        self.student_address_entry.delete(0, tk.END)
        self.student_aadhaar_entry.delete(0, tk.END)
        self.student_dob_entry.delete(0, tk.END)
        self.student_gender_combobox.set("Male")
        self.student_tenth_entry.delete(0, tk.END)
        self.student_twelfth_entry.delete(0, tk.END)
        self.student_blood_group_entry.delete(0, tk.END)
        self.student_mother_name_entry.delete(0, tk.END)
        self.student_enrollment_status_combobox.set("Yes")
        self.student_enrollment_date_entry.delete(0, tk.END)
        self.student_enrollment_date_entry.insert(0,    datetime.now().strftime("%Y-%m-
%d"))
        self.student_course_combobox.set("")
        self.student_academic_year_combobox.set("")
        self.student_faculty_combobox.set("")
        self.profile_picture_path = ""
        self.profile_pic_label.config(image="", text="No Image")


    def display_students(self):
        for item in self.student_tree.get_children():
            self.student_tree.delete(item)

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT s.*, c.course_name, a.year_name, f.faculty_name
            FROM students s
            LEFT JOIN courses c ON s.course_id = c.course_id
            LEFT JOIN academic_years a ON s.academic_year_id = a.year_id
            LEFT JOIN faculties f ON s.faculty_id = f.faculty_id
            ORDER BY s.student_id DESC
        """)
        students = cursor.fetchall()
        conn.close()

        for student in students:
            # Ensure all values are strings for insertion into Treeview
            student_data = [str(x) if x is not None else "N/A" for x in student]
            # Replace enrollment status (1=Yes, 0=No)
            enroll_status_text = "Yes" if student[13] == 1 else "No"
            student_data[13] = enroll_status_text # Update the status in the list

            self.student_tree.insert("", "end", values=student_data)
```

```
def search_students(self):
    search_term = self.search_entry.get().strip()
    for item in self.student_tree.get_children():
        self.student_tree.delete(item)

    conn = get_db_connection()
    cursor = conn.cursor()
    query = """
        SELECT s.*, c.course_name, a.year_name, f.faculty_name
        FROM students s
        LEFT JOIN courses c ON s.course_id = c.course_id
        LEFT JOIN academic_years a ON s.academic_year_id = a.year_id
        LEFT JOIN faculties f ON s.faculty_id = f.faculty_id
        WHERE s.roll_number LIKE ? OR s.name LIKE ?
        ORDER BY s.student_id DESC
    """
    cursor.execute(query, (f"%{search_term}%", f"%{search_term}%"))
    students = cursor.fetchall()
    conn.close()

    for student in students:
        student_data = [str(x) if x is not None else "N/A" for x in student]
        enroll_status_text = "Yes" if student[13] == 1 else "No"
        student_data[13] = enroll_status_text
        self.student_tree.insert("", "end", values=student_data)

def load_selected_student(self, event):
    selected_item = self.student_tree.focus()
    if not selected_item:
        return

    values = self.student_tree.item(selected_item, "values")
    # values[0] is student_id, values[1] is roll_number, etc.
    # Ensure there are enough values before trying to access them
    if len(values) < 18: # Check against the number of columns in the treeview
        messagebox.showwarning("Data Error", "Incomplete student data selected.")
        self.clear_student_fields()
        return

    self.clear_student_fields() # Clear previous data

    self.student_roll_entry.insert(0, values[1])
    self.student_name_entry.insert(0, values[2])
    self.student_contact_entry.insert(0, values[3])
    self.student_email_entry.insert(0, values[4])
```

```
        self.student_address_entry.insert(0, values[5])
        self.student_aadhaar_entry.insert(0, values[6])
        self.student_dob_entry.insert(0, values[7])
        self.student_gender_combobox.set(values[8])
        self.student_tenth_entry.insert(0, values[9])
        self.student_twelfth_entry.insert(0, values[10])
        self.student_blood_group_entry.insert(0, values[11])
        self.student_mother_name_entry.insert(0, values[12])
        self.student_enrollment_status_combobox.set(values[13])
        self.student_enrollment_date_entry.insert(0, values[14])
        self.student_course_combobox.set(values[15])
        self.student_academic_year_combobox.set(values[16])
        self.student_faculty_combobox.set(values[17])

        # Load profile picture if path exists
        student_id = values[0]
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("SELECT profile_picture_path FROM students WHERE
student_id=?", (student_id,))
        profile_path = cursor.fetchone()
        conn.close()

        if profile_path and profile_path[0] and os.path.exists(profile_path[0]):
            try:
                self.profile_picture_path = profile_path[0]
                img = Image.open(self.profile_picture_path)
                img = img.resize((100, 100), Image.LANCZOS)
                self.profile_pic_display = ImageTk.PhotoImage(img)
                self.profile_pic_label.config(image=self.profile_pic_display, text="")
                self.profile_pic_label.image = self.profile_pic_display
            except Exception:
                self.profile_picture_path = ""
                self.profile_pic_label.config(image="", text="No Image")
        else:
            self.profile_picture_path = ""
            self.profile_pic_label.config(image="", text="No Image")


    # --- Reports Tab ---
    def setup_reports_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Reports and Data Export", font=("Helvetica", 16,
"bold"), bootstyle="primary").pack(pady=10)

        reports_frame = ttk.LabelFrame(parent_frame, text="Generate Reports",
padding=15, bootstyle="info")
```

```
        reports_frame.pack(pady=20, padx=20, fill="x")

        # Report 1: Student Enrollment Report
        ttk.Label(reports_frame, text="Student Enrollment Report:", font=("Helvetica", 12)).grid(row=0, column=0, padx=5, pady=5, sticky="w")
        ttk.Button(reports_frame, text="Generate Enrollment Report", command=self.generate_enrollment_report, bootstyle="primary").grid(row=0, column=1, padx=5, pady=5, sticky="e")

        # Report 2: Marks Report by Course/Semester
        ttk.Label(reports_frame, text="Marks Report (by Course & Semester):", font=("Helvetica", 12)).grid(row=1, column=0, padx=5, pady=5, sticky="w")

        ttk.Label(reports_frame, text="Course:").grid(row=2, column=0, padx=5, pady=2, sticky="w")
        self.report_marks_course_combobox = ttk.Combobox(reports_frame, values=self._get_course_names())
        self.report_marks_course_combobox.grid(row=2, column=1, padx=5, pady=2, sticky="ew")

        ttk.Label(reports_frame, text="Semester:").grid(row=3, column=0, padx=5, pady=2, sticky="w")
        self.report_marks_semester_entry = ttk.Entry(reports_frame)
        self.report_marks_semester_entry.grid(row=3, column=1, padx=5, pady=2, sticky="ew")

        ttk.Button(reports_frame, text="Generate Marks Report", command=self.generate_marks_report, bootstyle="primary").grid(row=4, column=1, padx=5, pady=5, sticky="e")

        # Report 3: Payment History Report
        ttk.Label(reports_frame, text="Payment History Report:", font=("Helvetica", 12)).grid(row=5, column=0, padx=5, pady=5, sticky="w")
        ttk.Button(reports_frame, text="Generate Payment Report", command=self.generate_payment_report, bootstyle="primary").grid(row=5, column=1, padx=5, pady=5, sticky="e")

        # Report Output Area
        ttk.Label(parent_frame, text="Report Output:", font=("Helvetica", 12, "bold")).pack(pady=(10, 5))
        self.report_output_text = tk.Text(parent_frame, wrap="word", height=10, font=("Consolas", 10))
        self.report_output_text.pack(pady=10, padx=20, fill="both", expand=True)
        self.report_output_text.config(state=tk.DISABLED) # Make it read-only

    def generate_enrollment_report(self):
```

```
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT s.roll_number, s.name, s.enrollment_date, c.course_name,
a.year_name, f.faculty_name,
                CASE WHEN s.enrollment_status = 1 THEN 'Active' ELSE 'Inactive' END
AS status
            FROM students s
            LEFT JOIN courses c ON s.course_id = c.course_id
            LEFT JOIN academic_years a ON s.academic_year_id = a.year_id
            LEFT JOIN faculties f ON s.faculty_id = f.faculty_id
            ORDER BY s.enrollment_date DESC
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Student Enrollment Report\n"
        output_content += "----------------------------------------------------------------------------
-------------------------\n"
        output_content += f"{'Roll No':<10}{'Name':<25}{'Enroll
Date':<15}{'Course':<20}{'Acad Year':<15}{'Faculty':<15}{'Status':<10}\n"
        output_content += "----------------------------------------------------------------------------
-------------------------\n"
        for row in data:
            output_content += 
f"{row[0]:<10}{row[1]:<25}{row[2]:<15}{row[3]:<20}{row[4]:<15}{row[5]:<15}{
row[6]:<10}\n"

        self.report_output_text.config(state=tk.NORMAL)
        self.report_output_text.delete(1.0, tk.END)
        self.report_output_text.insert(tk.END, output_content)
        self.report_output_text.config(state=tk.DISABLED)

    def generate_marks_report(self):
        course_name = self.report_marks_course_combobox.get().strip()
        semester_str = self.report_marks_semester_entry.get().strip()

        if not course_name or not semester_str:
            messagebox.showwarning("Input Error", "Please select a Course and enter a
Semester for the Marks Report.")
            return

        try:
            semester = int(semester_str)
        except ValueError:
            messagebox.showerror("Input Error", "Semester must be a number.")
```

```
            return

        conn = get_db_connection()
        cursor = conn.cursor()

        cursor.execute("SELECT course_id FROM courses WHERE course_name=?",
(course_name,))
        course_id_data = cursor.fetchone()
        if not course_id_data:
            messagebox.showerror("Error", f"Course '{course_name}' not found.")
            conn.close()
            return
        course_id = course_id_data[0]

        cursor.execute("""
            SELECT s.roll_number, s.name, m.subject_name, m.marks_obtained,
m.max_marks, m.grade
            FROM marks m
            JOIN students s ON m.student_id = s.student_id
            WHERE m.course_id = ? AND m.semester = ?
            ORDER BY s.name, m.subject_name
        """, (course_id, semester))
        data = cursor.fetchall()
        conn.close()

        output_content = f"Marks Report for {course_name}, Semester {semester}\n"
        output_content += "-----------------------------------------------------------------------\n"
        output_content                                    +=                                    f"{'Roll
No':<10}{'Name':<20}{'Subject':<25}{'Marks':<8}{'Max':<8}{'Grade':<8}\n"
        output_content += "-----------------------------------------------------------------------\n"
        if not data:
            output_content += "No marks found for the selected criteria.\n"
        else:
            for row in data:
                output_content                                                              +=
f"{row[0]:<10}{row[1]:<20}{row[2]:<25}{row[3]:<8.2f}{row[4]:<8.2f}{row[5]:<8}
\n"

        self.report_output_text.config(state=tk.NORMAL)
        self.report_output_text.delete(1.0, tk.END)
        self.report_output_text.insert(tk.END, output_content)
        self.report_output_text.config(state=tk.DISABLED)

        # Reset the form fields after report generation
        self.report_marks_course_combobox.set("")
        self.report_marks_semester_entry.delete(0, tk.END)
```

```
    def generate_payment_report(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT s.roll_number, s.name, p.amount_paid, p.payment_date,
p.payment_type, p.receipt_number, p.description
            FROM payments p
            JOIN students s ON p.student_id = s.student_id
            ORDER BY p.payment_date DESC
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Payment History Report\n"
        output_content += "--------------------------------------------------------------------------------
-------------------------\n"
        output_content += f"{'Roll No':<10}{'Student
Name':<25}{'Amount':<10}{'Date':<15}{'Type':<15}{'Receipt
No':<15}{'Description':<25}\n"
        output_content += "--------------------------------------------------------------------------------
-------------------------\n"
        if not data:
            output_content += "No payment records found.\n"
        else:
            for row in data:
                output_content += f"{row[0]:<10}{row[1]:<25}{row[2]:<10.2f}{row[3]:<15}{row[4]:<15}{row[5] if
row[5] else 'N/A':<15}{row[6] if row[6] else 'N/A':<25}\n"

        self.report_output_text.config(state=tk.NORMAL)
        self.report_output_text.delete(1.0, tk.END)
        self.report_output_text.insert(tk.END, output_content)
        self.report_output_text.config(state=tk.DISABLED)


    # --- ID Card Generation Tab ---
    def setup_id_card_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Generate Student ID Cards", font=("Helvetica", 16,
"bold"), bootstyle="primary").pack(pady=10)

        input_frame = ttk.LabelFrame(parent_frame, text="Student Selection",
padding=10, bootstyle="info")
        input_frame.pack(pady=10, padx=10, fill="x", expand=False)
```

```
        ttk.Label(input_frame, text="Enter Student Roll Number:").grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.id_card_roll_entry = ttk.Entry(input_frame, width=30)
        self.id_card_roll_entry.grid(row=0, column=1, padx=5, pady=5, sticky="ew")

        ttk.Button(input_frame, text="Generate ID Card", command=self.generate_id_card, bootstyle="success").grid(row=0, column=2, padx=10, pady=5)

        # ID Card Display Area
        ttk.Label(parent_frame, text="Generated ID Card Preview:", font=("Helvetica", 12, "bold")).pack(pady=(10, 5))
        self.id_card_canvas = tk.Canvas(parent_frame, width=400, height=250, bg="white", relief="solid", bd=1)
        self.id_card_canvas.pack(pady=10, padx=10)
        self.id_card_photo = None # To hold the PhotoImage

    def generate_id_card(self):
        roll_number = self.id_card_roll_entry.get().strip()
        if not roll_number:
            messagebox.showwarning("Input Error", "Please enter a student roll number.")
            return

        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT s.name, s.roll_number, c.course_name, a.year_name, s.date_of_birth, s.blood_group,
                s.contact_number, s.profile_picture_path, s.enrollment_date
            FROM students s
            LEFT JOIN courses c ON s.course_id = c.course_id
            LEFT JOIN academic_years a ON s.academic_year_id = a.year_id
            WHERE s.roll_number = ?
        """, (roll_number,))
        student_data = cursor.fetchone()
        conn.close()

        if not student_data:
            messagebox.showerror("Not Found", f"No student found with Roll Number: {roll_number}")
            return

        name, roll_number, course_name, academic_year, dob, blood_group, contact_number, profile_pic_path, enrollment_date = student_data

        # ID Card Dimensions
```

```
        card_width = 400
        card_height = 250

        try:
            # Use the provided background image
            if os.path.exists(IDENTITY_CARD_BACKGROUND_PATH):
                id_card_image = Image.open(IDENTITY_CARD_BACKGROUND_PATH).resize((card_width, card_height), Image.LANCZOS)
            else:
                id_card_image = Image.new('RGB', (card_width, card_height), color = (255, 255, 255)) # White background if not found
                messagebox.showwarning("Image Warning", f"Identity card background image not found: {IDENTITY_CARD_BACKGROUND_PATH}. Using plain white background.")


            draw = ImageDraw.Draw(id_card_image)

            # Define fonts (adjust paths if fonts are not system-wide)
            try:
                font_title = ImageFont.truetype("arialbd.ttf", 20)
                font_header = ImageFont.truetype("arialbd.ttf", 14)
                font_normal = ImageFont.truetype("arial.ttf", 12)
            except IOError:
                font_title = ImageFont.load_default()
                font_header = ImageFont.load_default()
                font_normal = ImageFont.load_default()

            # College Name and Address
            college_name = "Saraswati College, Shegaon"
            college_address = "Gaulkhed Road, Shegaon Dist:- Buldhana, State:- Maharashtra (INDIA) Pin: 444 203"

            draw.text((card_width / 2, 20), college_name, fill=(0, 0, 0), font=font_title, anchor="mm")
            draw.text((card_width / 2, 45), "STUDENT IDENTITY CARD", fill=(0, 0, 0), font=font_header, anchor="mm")
            draw.text((card_width / 2, 65), college_address, fill=(0, 0, 0), font=font_normal, anchor="mm")


            # Student details
            y_offset = 90
            text_color = (0, 0, 0) # Black color for text
```

```
        # Profile Picture
        if profile_pic_path and os.path.exists(profile_pic_path):
            profile_img = Image.open(profile_pic_path)
            profile_img = profile_img.resize((80, 80), Image.LANCZOS)
            # Paste the profile picture onto the card
            id_card_image.paste(profile_img, (20, y_offset), profile_img if profile_img.mode == 'RGBA' else None) # Use mask for transparency
        else:
            draw.text((20, y_offset + 30), "No Photo", fill=text_color, font=font_normal)

        x_start_details = 120
        draw.text((x_start_details, y_offset), f"Name: {name}", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 20), f"Roll No: {roll_number}", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 40), f"Course: {course_name} ({academic_year})", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 60), f"DOB: {dob}", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 80), f"Blood Group: {blood_group}", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 100), f"Contact: {contact_number}", fill=text_color, font=font_normal)
        draw.text((x_start_details, y_offset + 120), f"Enrollment Date: {enrollment_date}", fill=text_color, font=font_normal)

        # Display the generated ID card
        self.id_card_photo = ImageTk.PhotoImage(id_card_image)
        self.id_card_canvas.delete("all")
        self.id_card_canvas.create_image(0, 0, anchor="nw", image=self.id_card_photo)

        # Save option
        if messagebox.askyesno("ID Card Generated", "ID Card generated successfully! Do you want to save it as an image?"):
            file_path = filedialog.asksaveasfilename(
                defaultextension=".png",
                filetypes=[("PNG files", "*.png"), ("JPEG files", "*.jpg"), ("All files", "*.*")],
                initialfile=f"ID_Card_{roll_number}.png"
            )
            if file_path:
                id_card_image.save(file_path)
                messagebox.showinfo("Save Success", f"ID Card saved to {file_path}")

    except Exception as e:
```

```
            messagebox.showerror("ID Card Error", f"Failed to generate ID Card: {e}")

    # --- Receipt Generation Tab ---
    def setup_receipt_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Generate Payment Receipts", font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=10)

        input_frame = ttk.LabelFrame(parent_frame, text="Payment Details", padding=10, bootstyle="info")
        input_frame.pack(pady=10, padx=10, fill="x", expand=False)

        # Roll Number and Amount
        ttk.Label(input_frame, text="Student Roll Number:").grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.receipt_roll_entry = ttk.Entry(input_frame, width=30)
        self.receipt_roll_entry.grid(row=0, column=1, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Amount Paid (INR):").grid(row=1, column=0, padx=5, pady=5, sticky="w")
        self.receipt_amount_entry = ttk.Entry(input_frame, width=30)
        self.receipt_amount_entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")

        # Payment Type
        ttk.Label(input_frame, text="Payment Type:").grid(row=2, column=0, padx=5, pady=5, sticky="w")
        self.receipt_type_combobox = ttk.Combobox(input_frame, values=["Tuition Fee", "Exam Fee", "Library Fine", "Other"])
        self.receipt_type_combobox.grid(row=2, column=1, padx=5, pady=5, sticky="ew")
        self.receipt_type_combobox.set("Tuition Fee") # Default

        # Description
        ttk.Label(input_frame, text="Description (Optional):").grid(row=3, column=0, padx=5, pady=5, sticky="w")
        self.receipt_description_entry = ttk.Entry(input_frame, width=30)
        self.receipt_description_entry.grid(row=3, column=1, padx=5, pady=5, sticky="ew")

        ttk.Button(input_frame, text="Generate Receipt", command=self.generate_receipt, bootstyle="success").grid(row=4, column=0, columnspan=2, pady=15)

        # Receipt Output Area
        ttk.Label(parent_frame, text="Generated Receipt Preview:", font=("Helvetica", 12, "bold")).pack(pady=(10, 5))
```

```
        self.receipt_output_text = tk.Text(parent_frame, wrap="word", height=10, font=("Consolas", 10))
        self.receipt_output_text.pack(pady=10, padx=10, fill="both", expand=True)
        self.receipt_output_text.config(state=tk.DISABLED) # Make it read-only

    def generate_receipt(self):
        roll_number = self.receipt_roll_entry.get().strip()
        amount_paid_str = self.receipt_amount_entry.get().strip()
        payment_type = self.receipt_type_combobox.get().strip()
        description = self.receipt_description_entry.get().strip()
        payment_date = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        if not roll_number or not amount_paid_str:
            messagebox.showwarning("Input Error", "Please enter student Roll Number and Amount Paid.")
            return

        try:
            amount_paid = float(amount_paid_str)
            if amount_paid <= 0:
                raise ValueError("Amount must be positive.")
        except ValueError:
            messagebox.showerror("Input Error", "Amount Paid must be a valid positive number.")
            return

        conn = get_db_connection()
        cursor = conn.cursor()

        try:
            cursor.execute("SELECT student_id, name, course_id FROM students WHERE roll_number=?", (roll_number,))
            student_data = cursor.fetchone()

            if not student_data:
                messagebox.showerror("Error", f"No student found with Roll Number: {roll_number}")
                return

            student_id, student_name, course_id = student_data

            # Get course name from course_id
            course_name = "N/A"
            if course_id:
                cursor.execute("SELECT course_name FROM courses WHERE course_id=?", (course_id,))
```

```
            course_result = cursor.fetchone()
            if course_result:
                course_name = course_name[0]

        # Generate a simple receipt number (e.g., timestamp + roll_number)
        receipt_number = f"REC-{datetime.now().strftime('%Y%m%d%H%M%S')}-{roll_number}"

        cursor.execute("""
            INSERT INTO payments (student_id, amount_paid, payment_date, payment_type, receipt_number, description)
            VALUES (?, ?, ?, ?, ?, ?)
        """, (student_id, amount_paid, payment_date, payment_type, receipt_number, description))
        conn.commit()

        receipt_content = f"""
--------------------------------------------------
      Saraswati College,Shegaon
      PAYMENT RECEIPT
--------------------------------------------------

Receipt No:   {receipt_number}
Date:         {payment_date}

Student Name: {student_name}
Roll Number:  {roll_number}
Course:       {course_name}

Amount Paid:  INR {amount_paid:.2f}
Payment Type: {payment_type}
Description:  {description if description else 'N/A'}


                        signature
--------------------------------------------------
Thank you for your payment!
--------------------------------------------------
        """
        self.receipt_output_text.config(state=tk.NORMAL)
        self.receipt_output_text.delete(1.0, tk.END)
        self.receipt_output_text.insert(tk.END, receipt_content)
        self.receipt_output_text.config(state=tk.DISABLED)

        messagebox.showinfo("Receipt Generated", "Receipt generated and recorded successfully!", parent=self.master)
```

```
            self.clear_receipt_fields()

        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"Failed to generate receipt: {e}", parent=self.master)
        finally:
            conn.close()

    def clear_receipt_fields(self):
        self.receipt_roll_entry.delete(0, tk.END)
        self.receipt_amount_entry.delete(0, tk.END)

        self.receipt_type_combobox.set("Tuition Fee")
        self.receipt_description_entry.delete(0, tk.END)

    # --- Analytics & Insights Tab ---
    def setup_analytics_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Analytics and Performance Insights", font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=10)

        analytics_frame = ttk.LabelFrame(parent_frame, text="Generate Analytics", padding=15, bootstyle="info")
        analytics_frame.pack(pady=20, padx=20, fill="x")

        ttk.Label(analytics_frame, text="Choose Insight:", font=("Helvetica", 12)).grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.analytics_combobox = ttk.Combobox(analytics_frame, values=[
            "Students per Course",
            "Average Marks per Course",
            "Enrollment Status Breakdown",
            "Faculty Academic Performance"
        ])
        self.analytics_combobox.grid(row=0, column=1, padx=5, pady=5, sticky="ew")
        self.analytics_combobox.set("Students per Course") # Default

        ttk.Button(analytics_frame, text="Generate Insight", command=self.generate_analytics, bootstyle="primary").grid(row=0, column=2, padx=10, pady=5)

        # Analytics Output Area
        ttk.Label(parent_frame, text="Analytics Output:", font=("Helvetica", 12, "bold")).pack(pady=(10, 5))
        self.performance_output_text = tk.Text(parent_frame, wrap="word", height=10, font=("Consolas", 10))
        self.performance_output_text.pack(pady=10, padx=20, fill="both", expand=True)
        self.performance_output_text.config(state=tk.DISABLED)
```

```
    def generate_analytics(self):
        selected_insight = self.analytics_combobox.get()
        self.performance_output_text.config(state=tk.NORMAL)
        self.performance_output_text.delete(1.0, tk.END)

        if selected_insight == "Students per Course":
            self._students_per_course_report()
        elif selected_insight == "Average Marks per Course":
            self._average_marks_per_course_report()
        elif selected_insight == "Enrollment Status Breakdown":
            self._enrollment_status_breakdown()
        elif selected_insight == "Faculty Academic Performance":
            self._faculty_academic_performance()
        else:
            self.performance_output_text.insert(tk.END, "Please select a valid insight to generate.")

        self.performance_output_text.config(state=tk.DISABLED)

    def _students_per_course_report(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT c.course_name, COUNT(s.student_id) AS total_students
            FROM courses c
            LEFT JOIN students s ON c.course_id = s.course_id
            GROUP BY c.course_name
            ORDER BY total_students DESC
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Students Enrolled Per Course\n"
        output_content += "----------------------------------------\n"
        output_content += f"{'Course':<25}{'Total Students':<15}\n"
        output_content += "----------------------------------------\n"
        for course, count in data:
            output_content += f"{course:<25}{count:<15}\n"
        self.performance_output_text.insert(tk.END, output_content)

    def _average_marks_per_course_report(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
```

```
            SELECT c.course_name, AVG(m.marks_obtained * 1.0 / m.max_marks) * 100
AS average_percentage
            FROM marks m
            JOIN courses c ON m.course_id = c.course_id
            GROUP BY c.course_name
            ORDER BY average_percentage DESC
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Average Marks Percentage Per Course\n"
        output_content += "-----------------------------------------------\n"
        output_content += f"{'Course':<25}{'Average Percentage':<20}\n"
        output_content += "-----------------------------------------------\n"
        if not data:
            output_content += "No marks data available for courses.\n"
        else:
            for course, avg_percent in data:
                avg_percent_str = f"{avg_percent:.2f}%" if avg_percent is not None else
"N/A"
                output_content += f"{course:<25}{avg_percent_str:<20}\n"

        self.performance_output_text.config(state=tk.NORMAL)
        self.performance_output_text.delete(1.0, tk.END)
        self.performance_output_text.insert(tk.END, output_content)
        self.performance_output_text.config(state=tk.DISABLED)

    def _enrollment_status_breakdown(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT
                CASE WHEN enrollment_status = 1 THEN 'Active' ELSE 'Inactive' END
AS status,
                COUNT(student_id) AS total_students
            FROM students
            GROUP BY status
            ORDER BY status DESC
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Student Enrollment Status Breakdown\n"
        output_content += "----------------------------------\n"
        output_content += f"{'Status':<15}{'Total Students':<15}\n"
        output_content += "----------------------------------\n"
```

```
        for status, count in data:
            output_content += f"{status:<15}{count:<15}\n"
        self.performance_output_text.insert(tk.END, output_content)

    def _faculty_academic_performance(self):
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
            SELECT f.faculty_name,
                AVG(s.tenth_percent) AS avg_10th_percent,
                AVG(s.twelfth_percent) AS avg_12th_percent,
                COUNT(s.student_id) AS total_students
            FROM faculties f
            LEFT JOIN students s ON f.faculty_id = s.faculty_id
            GROUP BY f.faculty_name
            ORDER BY avg_10th_percent DESC, avg_12th_percent DESC,
COUNT(s.student_id) AS total_students
        """)
        data = cursor.fetchall()
        conn.close()

        output_content = "Faculty Academic Performance (Avg 10th/12th %)\n"
        output_content += "-----------------------------------------------------------------\n"
        output_content += f"{'Faculty':<15}{'Avg 10th %':<15}{'Avg 12th
%':<15}{'Total Students':<15}\n"
        output_content += "-----------------------------------------------------------------\n"
        for faculty, avg_10th, avg_12th, total_students in data:
            avg_10th_str = f"{avg_10th:.2f}" if avg_10th is not None else "N/A"
            avg_12th_str = f"{avg_12th:.2f}" if avg_12th is not None else "N/A"
            output_content                                                    +=
f"{faculty:<15}{avg_10th_str:<15}{avg_12th_str:<15}{total_students:<15}\n"

        self.performance_output_text.config(state=tk.NORMAL)
        self.performance_output_text.delete(1.0, tk.END)
        self.performance_output_text.insert(tk.END, output_content)
        self.performance_output_text.config(state=tk.DISABLED)

    # --- Feedback Tab ---
    def setup_feedback_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Provide Feedback or Suggestions",
font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=10)

        feedback_frame = ttk.LabelFrame(parent_frame, text="Your Feedback",
padding=15, bootstyle="info")
        feedback_frame.pack(pady=20, padx=20, fill="both", expand=True)
```

```
        ttk.Label(feedback_frame, text="Your Name (Optional):").pack(pady=(10, 5), anchor="w")
        self.feedback_name_entry = ttk.Entry(feedback_frame, width=50)
        self.feedback_name_entry.pack(pady=5, fill="x")

        ttk.Label(feedback_frame, text="Your Email (Optional):").pack(pady=(10, 5), anchor="w")
        self.feedback_email_entry = ttk.Entry(feedback_frame, width=50)
        self.feedback_email_entry.pack(pady=5, fill="x")

        ttk.Label(feedback_frame, text="Feedback/Suggestions:").pack(pady=(10, 5), anchor="w")
        self.feedback_text_area = tk.Text(feedback_frame, wrap="word", height=10, font=("Helvetica", 10))
        self.feedback_text_area.pack(pady=5, fill="both", expand=True)

        ttk.Button(feedback_frame, text="Submit Feedback", command=self.submit_feedback, bootstyle="success").pack(pady=15)

    def submit_feedback(self):
        feedback = self.feedback_text_area.get("1.0", tk.END).strip()
        name = self.feedback_name_entry.get().strip()
        email = self.feedback_email_entry.get().strip()

        if not feedback:
            messagebox.showwarning("Input Error", "Please enter your feedback before submitting.")
            return

        timestamp = datetime.now().strftime("%Y-%m-%d %H:%M:%S")

        conn = get_db_connection()
        cursor = conn.cursor()
        try:
            cursor.execute("INSERT INTO feedback (name, email, feedback_text, timestamp) VALUES (?, ?, ?, ?)",
                           (name, email, feedback, timestamp))
            conn.commit()
            messagebox.showinfo("Feedback Submitted", "Thank you for your feedback! It has been recorded.", parent=self.master)
            self.feedback_name_entry.delete(0, tk.END)
            self.feedback_email_entry.delete(0, tk.END)
            self.feedback_text_area.delete("1.0", tk.END)
        except sqlite3.Error as e:
            messagebox.showerror("Database Error", f"Failed to submit feedback: {e}", parent=self.master)
```

```
        finally:
            conn.close()

# --- Main Execution ---
if __name__ == "__main__":
    init_db()
    root = tk.Tk()
    login_app = LoginWindow(root)
    root.mainloop()    # ...existing code...

    def setup_marks_entry_tab(self, parent_frame):
        ttk.Label(parent_frame, text="Marks Entry", font=("Helvetica", 16, "bold"), bootstyle="primary").pack(pady=10)

        input_frame = ttk.LabelFrame(parent_frame, text="Enter Marks", padding=10, bootstyle="info")
        input_frame.pack(pady=10, padx=10, fill="x", expand=False)

        ttk.Label(input_frame, text="Student Roll Number:").grid(row=0, column=0, padx=5, pady=5, sticky="w")
        self.marks_roll_entry = ttk.Entry(input_frame, width=25)
        self.marks_roll_entry.grid(row=0, column=1, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Course:").grid(row=0, column=2, padx=5, pady=5, sticky="w")
        self.marks_course_combobox = ttk.Combobox(input_frame, values=self._get_course_names(), width=20)
        self.marks_course_combobox.grid(row=0, column=3, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Semester:").grid(row=1, column=0, padx=5, pady=5, sticky="w")
        self.marks_semester_entry = ttk.Entry(input_frame, width=25)
        self.marks_semester_entry.grid(row=1, column=1, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Subject Name:").grid(row=1, column=2, padx=5, pady=5, sticky="w")
        self.marks_subject_entry = ttk.Entry(input_frame, width=20)
        self.marks_subject_entry.grid(row=1, column=3, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Marks Obtained:").grid(row=2, column=0, padx=5, pady=5, sticky="w")
        self.marks_obtained_entry = ttk.Entry(input_frame, width=25)
        self.marks_obtained_entry.grid(row=2, column=1, padx=5, pady=5, sticky="ew")
```

```
        ttk.Label(input_frame, text="Max Marks:").grid(row=2, column=2, padx=5, pady=5, sticky="w")
        self.marks_max_entry = ttk.Entry(input_frame, width=20)
        self.marks_max_entry.grid(row=2, column=3, padx=5, pady=5, sticky="ew")

        ttk.Label(input_frame, text="Grade:").grid(row=3, column=0, padx=5, pady=5, sticky="w")
        self.marks_grade_entry = ttk.Entry(input_frame, width=25)
        self.marks_grade_entry.grid(row=3, column=1, padx=5, pady=5, sticky="ew")

        ttk.Button(input_frame, text="Add Marks", command=self.add_marks, bootstyle="success").grid(row=4, column=0, columnspan=4, pady=10)

        # Marks Display
        display_frame = ttk.LabelFrame(parent_frame, text="Student Marks", padding=10, bootstyle="primary")
        display_frame.pack(pady=10, padx=10, fill="both", expand=True)

        self.marks_tree = ttk.Treeview(display_frame, columns=("Subject", "Semester", "Marks", "Max", "Grade"), show="headings")
        for col in self.marks_tree["columns"]:
            self.marks_tree.heading(col, text=col)
            self.marks_tree.column(col, width=100, anchor="center")
        self.marks_tree.pack(fill="both", expand=True)

        ttk.Button(display_frame, text="Show Marks", command=self.display_student_marks, bootstyle="info").pack(pady=5)

    def add_marks(self):
        roll = self.marks_roll_entry.get().strip()
        course = self.marks_course_combobox.get().strip()
        semester = self.marks_semester_entry.get().strip()
        subject = self.marks_subject_entry.get().strip()
        marks = self.marks_obtained_entry.get().strip()
        max_marks = self.marks_max_entry.get().strip()
        grade = self.marks_grade_entry.get().strip()

        if not all([roll, course, semester, subject, marks, max_marks, grade]):
            messagebox.showwarning("Input Error", "All fields are required.")
            return

        try:
            semester = int(semester)
            marks = float(marks)
            max_marks = float(max_marks)
        except ValueError:
```

```
            messagebox.showerror("Input Error", "Semester, Marks, and Max Marks must
be numbers.")
            return

        conn = get_db_connection()
        cursor = conn.cursor()
        try:
            cursor.execute("SELECT student_id FROM students WHERE roll_number=?",
(roll,))
            student = cursor.fetchone()
            if not student:
                messagebox.showerror("Error", "Student not found.")
                return
            student_id = student[0]

            cursor.execute("SELECT course_id FROM courses WHERE course_name=?",
(course,))
            course_row = cursor.fetchone()
            if not course_row:
                messagebox.showerror("Error", "Course not found.")
                return
            course_id = course_row[0]

            cursor.execute(
                "INSERT INTO marks (student_id, course_id, subject_name, semester,
marks_obtained, max_marks, grade) VALUES (?, ?, ?, ?, ?, ?, ?)",
                (student_id, course_id, subject, semester, marks, max_marks, grade)
            )
            conn.commit()
            messagebox.showinfo("Success", "Marks added successfully!")
            self.display_student_marks()
        except Exception as e:
            messagebox.showerror("Error", f"Failed to add marks: {e}")
        finally:
            conn.close()

    def display_student_marks(self):
        roll = self.marks_roll_entry.get().strip()
        for item in self.marks_tree.get_children():
            self.marks_tree.delete(item)
        if not roll:
            return
        conn = get_db_connection()
        cursor = conn.cursor()
        cursor.execute("""
```

```
            SELECT m.subject_name, m.semester, m.marks_obtained, m.max_marks,
m.grade
            FROM marks m
            JOIN students s ON m.student_id = s.student_id
            WHERE s.roll_number = ?
            ORDER BY m.semester, m.subject_name
        """, (roll,))
        for row in cursor.fetchall():
            self.marks_tree.insert("", "end", values=row)
        conn.close()
    # ...existing code...    # ...existing code in create_main_widgets()...
        # Tab 7: Marks Entry
        marks_entry_frame = ttk.Frame(self.notebook)
        self.notebook.add(marks_entry_frame, text="Marks Entry")
        self.setup_marks_entry_tab(marks_entry_frame)
    # ...existing code...
```