**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...