LITTLE ROCK INDIAN SCHOOL
BRAHMAVAR - 576213

REPORT OF COMPUTER SCIENCE PROJECT
ACADEMIC YEAR
2024 - 2026
XI & XII

SCHOOL MANAGEMENT SYSTEM

NAME: Vishnu

REGISTRATION NUMBER: _____

DATE OF SUBMISSION: _____

LITTLE ROCK INDIAN SCHOOL

BRAHMAVAR - 576213

CERTIFICATE

CLASS XI AND XII
COMPUTER SCIENCE
2024 - 2026

*This is to certify that Mr **Vishnu** has satisfactorily completed a Minor project "**School Management System**" in Computer Science under my supervision at Little Rock Indian School, Brahmavar during the academic years 2024 - 2026*

PROJECT IN-CHARGE:- Mrs. Sona Maria Fernandes

Submitted to the AISSCE Practical Examination held on _____
at Little Rock Indian School, Brahmavar.

REG. NO. :- _____

EXAMINERS: 1) _____
2) _____

NAME:- Vishnu

ACKNOWLEDGEMENT

I take this opportunity to express my deepest sense of gratitude to all those who have guided, supported, and inspired me throughout the course of this Computer Science project. The successful completion of this work would not have been possible without the encouragement and assistance of many individuals.

I am sincerely grateful to our respected Director, Prof. Mathew C. Ninan, and our Principal, Dr. John Thomas, for providing an excellent academic environment and constant encouragement that motivated me to pursue this project with dedication. Their vision for academic excellence and emphasis on practical learning have been a true source of inspiration.

My heartfelt appreciation goes to our Head Teacher, Mr. Harikrishna D, whose guidance and valuable insights have always encouraged us to strive for quality in our work. I am especially thankful to our Computer Science teacher, Mrs. Sona Maria Fernandes, for her patient guidance, technical expertise, and continuous motivation throughout the course of this project. Her constructive feedback and unwavering support helped refine my ideas and bring this work to completion.

I also wish to express my gratitude to Harsha Sir for his kind assistance and cooperation in the computer laboratory, which played a vital role in facilitating the practical aspects of this project.

A special word of appreciation is due to my project partner, Vittal, for his constant collaboration, commitment, and teamwork, which made this project both enjoyable and productive.

Finally, I extend my sincere thanks to all my teachers, friends, and well-wishers who, directly or indirectly, contributed their time, effort, and encouragement towards the successful completion of this project.

-Vishnu

INDEX

AIM

The aim of this project is to develop a School Management System, named EDUMATRIX, using Python and SQLite3 to automate the day-to-day administrative tasks of a school.

The system provides a simple and secure interface for managing student and staff records, generating report cards, and sending them via email.

It aims to reduce manual work, minimize errors, and improve efficiency by integrating modern features such as AI-based performance insights, PDF report generation, and secure admin login with access code verification.
Through this project, the objective is to demonstrate how technology can streamline educational administration and create a smart, paperless, and user-friendly school management environment.

INTRODUCTION

In the modern era of technology, almost every sector has moved towards computerization to improve efficiency, accuracy, and speed of operations. Educational institutions are no exception. Managing a school involves handling large amounts of data related to students, staff, academics, and administration. Performing these tasks manually not only consumes time but also increases the chances of human error. Hence, there is a growing need for a system that can automate school operations and make the process more reliable and organized.

To meet this requirement, the project “EDUMATRIX – School Management System” has been developed using the Python programming language with SQLite3 as its backend database. This system provides a centralized digital platform that automates major school management tasks such as maintaining student and staff records, generating report cards, and communicating academic results through emails. It is designed to be simple, secure, and user-friendly so that even non-technical users can operate it effectively.

The graphical user interface (GUI) of EDUMATRIX is built using Tkinter, which allows users to interact with the system through easy-to-use windows, buttons, and input fields. All data is stored securely in the SQLite database, ensuring quick retrieval and efficient storage. The system also incorporates bcrypt encryption for password security, ensuring that administrator credentials remain protected. To prevent unauthorized users from creating admin accounts, an access-code verification system has been implemented, making the software more secure and controlled.

One of the key highlights of this project is its Report Card Generation Module, which allows the administrator to automatically create professional-looking report cards in PDF format using the ReportLab library. The system can also email these report cards directly to students or parents, eliminating the need for physical copies and making the process faster and eco-friendly. Additionally, it uses AI-based insights to generate remarks about student performance, providing

meaningful feedback and analysis.

By combining automation, data management, and artificial intelligence, EDUMATRIX provides an efficient solution for educational administration. It reduces manual effort, ensures accuracy, and improves communication within the school environment. Moreover, it reflects the importance of programming and database management in solving real-world problems.

In conclusion, EDUMATRIX demonstrates how technology can transform traditional administrative systems into smart, paperless, and highly efficient solutions. The project not only enhances productivity but also encourages sustainable practices in education by reducing paperwork and promoting digital record-keeping. Through this application, school management becomes more organized, transparent, and future-ready.

BASIC PRINCIPLES AND WORKING

The School Management System – EDUMATRIX is based on the principle of computerized data management and process automation.
It uses a structured approach to handle information systematically through the combination of a Graphical User Interface (GUI), Database Connectivity, and Secure Authentication Mechanisms.
The system follows a modular architecture, where each module performs a specific task such as user authentication, record management, or report generation.
All modules interact with a centralized database, ensuring that data remains consistent, up-to-date, and easily accessible.

➤BASIC PRINCIPLES

1. Automation
   The core idea behind EDUMATRIX is automation — replacing repetitive manual work like writing marks, creating report cards, or managing student and staff registers with digital operations. Automation helps save time, increases accuracy, and ensures uniformity across all records.

2. Data Centralization
   All information — whether it belongs to students, staff, or admin users — is stored in one centralized SQLite database. This allows for easy retrieval, modification, and backup without dealing with multiple files or scattered data sources.

3. User Accessibility and Security
   The system ensures that only authorized users can access and modify the data. Each admin must log in using a verified username and password, which are encrypted using bcrypt before storage. Additionally, during signup, an Access Code Verification step is introduced to prevent unauthorized creation of admin accounts.

4. Data Integrity and Reliability
   The system uses database constraints and structured data types to maintain data accuracy. Every action such as adding or editing a record is reflected in real time, ensuring that the database remains reliable.

5. Scalability and Maintainability
   The modular design allows new features to be added easily — for instance, attendance tracking or fee management — without disturbing the existing code structure.

➤WORKING OF THE SYSTEM

The working of EDUMATRIX can be divided into several stages or modules that interact seamlessly to form a complete system.

1. Admin Login and Authentication

When the application starts, the user is directed to a login screen created using Tkinter.

- If the admin already has an account, they can enter their username and password to log in.
- During the login process, the entered password is compared to the bcrypt-hashed version stored in the database to verify authenticity.
- For new users, a signup window is available, but it requires entering a unique admin access code to ensure that only authorized personnel can register.
  This double-layer security approach makes the system safe from unauthorized access.

2. Dashboard and Navigation

After successful login, the user is taken to the main dashboard, which acts as the control center of the application.

It displays multiple buttons and options that lead to various features such as:

- Student Management
- Staff Management
- Report Card Generation
- Email Communication
- Login History

The dashboard is designed using Tkinter frames and buttons, making it visually simple and easy to use.

3. Student and Staff Management Modules

These two modules are the backbone of the system.

Each allows the admin to perform CRUD operations — Create, Read, Update, and Delete — for both student and staff records.

The admin can add new students or staff members by entering their details into a form. Data such as names, contact information, date of birth, address, and photo are stored in the SQLite3 database.

For convenience, a postal code auto-fill feature helps automatically retrieve location details.

Admins can view the entire list of records, edit specific fields, or delete entries if required.

This module ensures proper digital record-keeping for the institution.

4. Report Card Generation

This module automates the process of creating report cards.

The admin can input student marks, and the system calculates grades, percentages, and overall performance.

Using the ReportLab library, the system generates a well-formatted PDF report card containing all details such as student information, subject-wise marks, and remarks.

The highlight feature here is the integration of AI-generated insights, where the system analyzes performance data to produce short comments like “Excellent

progress in Mathematics" or "Needs improvement in Science."
These intelligent remarks make the report more meaningful and personalized.

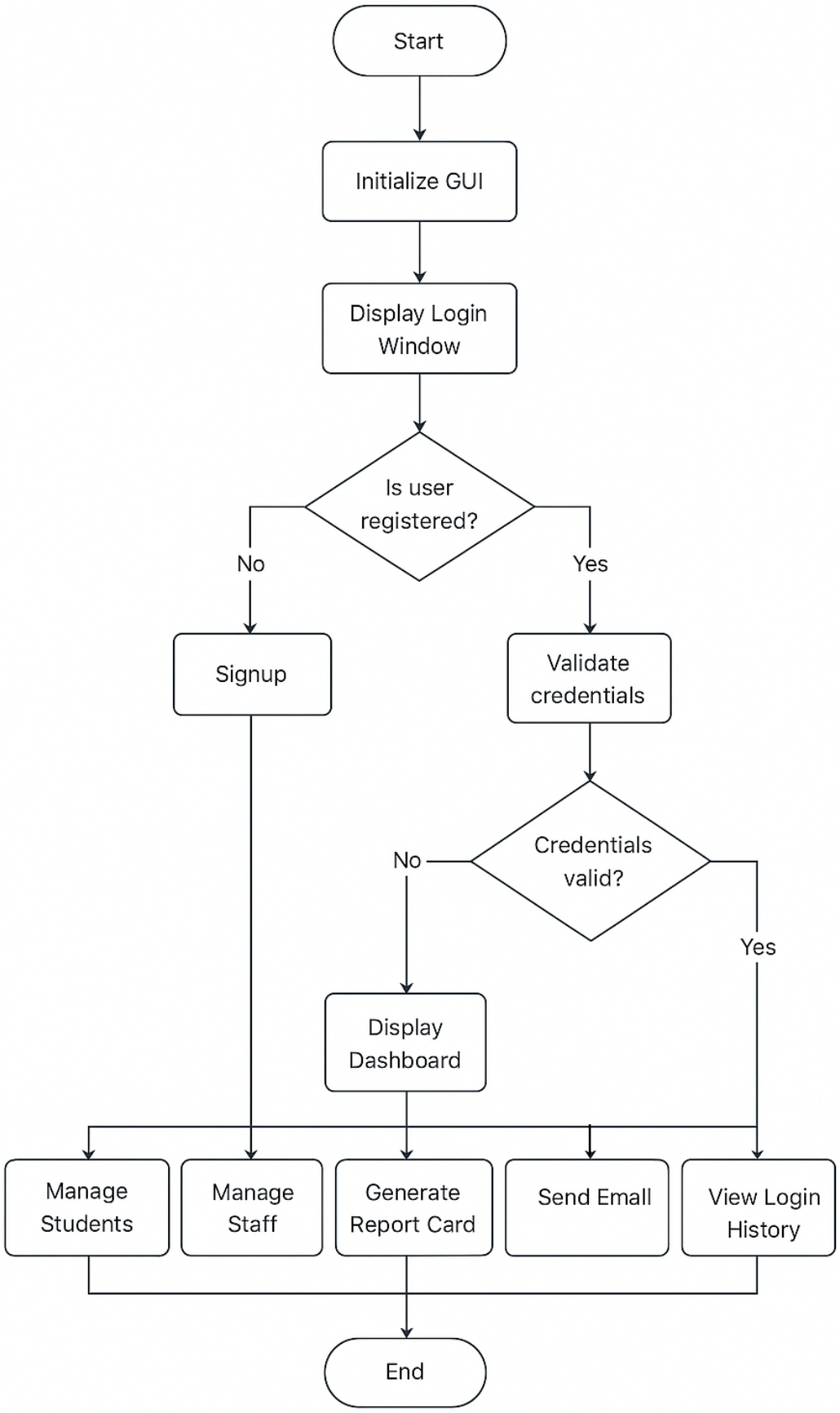
5. Email Integration
To make the system fully automated, EDUMATRIX includes an email-sending module.
Using the smtplib library, the system can automatically attach the generated report card PDF and send it to the student's registered email address.
This promotes eco-friendly, paperless communication between the school and parents.

6. Login History Tracking
Every login and logout event of an admin is recorded in a separate table called login_history.
This helps monitor system usage, track activity, and enhance accountability within the administrative team.

➤CONCLUSION

The entire system works on the principle of digital automation integrated with secure database management.
By combining Python's simplicity, Tkinter's GUI capability, SQLite's stability, and libraries like bcrypt and ReportLab, EDUMATRIX delivers a complete, modern solution to school management.
It is reliable, expandable, and demonstrates how programming can be used to transform traditional administrative systems into efficient, intelligent, and paperless environments.

ALGORITHM FLOWCHART



SOURCE CODE

PROLOGUE:

➤**Developers:** Vishnu
Vittal

➤**Start Date:** 11/09/2024

➤**End Date:** 17/11/2025

➤**Total time spent:** 744 hours = 31 days

➤**Storage required:** 1,54,000 bytes (154 KB on disk)

➤**Total number of lines:** 2677

➤**Project Guide:** Mrs. Sona Maria Fernandes.

➤**Place Developed:** APEX Computer Lab,
Little Rock Indian School,
Brahmavar.

```
import tkinter as tk
from tkinter import ttk, messagebox, simpledialog, filedialog
import sqlite3
import hashlib
import bcrypt
import csv
import os
import smtplib
import random
import re
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from tkinter import ttk
from datetime import datetime, date
from tkinter import ttk, messagebox
```

```
from tkinter import simpledialog
from PIL import Image, ImageTk




# Hash password for new accounts
def hash_password(password: str) -> bytes:
    return bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())

# Dual-mode password verification
def verify_password(password: str, stored_hash) -> bool:
    """
    Verify password against stored hash.
    Supports:
    - bcrypt (bytes) for new accounts
    - SHA256 (hex string) for old accounts
    Automatically upgrades SHA256 accounts to bcrypt on successful login.
    """
    import hashlib, sqlite3

    # Bcrypt hash
    if isinstance(stored_hash, bytes):
        return bcrypt.checkpw(password.encode('utf-8'), stored_hash)

    # SHA256 legacy hash
    elif isinstance(stored_hash, str) and len(stored_hash) == 64 and all(c in "0123456789abcdef" for
c in stored_hash.lower()):
        if hashlib.sha256(password.encode('utf-8')).hexdigest() == stored_hash:
            # Upgrade to bcrypt
            new_hash = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt())
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("UPDATE admins SET password=? WHERE password=?", (new_hash,
stored_hash))
                conn.commit()
            return True
        else:
            return False
    else:
        return False


postal_data = {}

def load_indian_postal_data(csv_file):
    global postal_data
    with open(csv_file, newline='', encoding='utf-8') as csvfile:
        reader = csv.DictReader(csvfile)
        for row in reader:
            postal_data[row['District'].strip().lower()] = (
                row['Pincode'].strip(),
                row['StateName'].strip()
            )

# Load directly from CSV
load_indian_postal_data("/Users/vittalvishnu/Documents/indian_pincodes.csv")

current_user = None
```

```
current_session_id = None
verification_codes = {}

DB_FOLDER = "/Users/vittalvishnu/Documents/aisms"
DB_PATH = os.path.join(DB_FOLDER, "admin.db")

os.makedirs(DB_FOLDER, exist_ok=True)

# Initialize database admin.db
def initialize_database():
    """
    Initialize admin.db and create all required tables with a consistent schema.
    Performs a safe migration when an existing students table has an unexpected schema
    (for example an extra `role_no` column). This ensures the rest of the app can
    rely on a stable column ordering and names.
    """
    try:
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()

            # Admins Table
            c.execute("""
                CREATE TABLE IF NOT EXISTS admins (
                    username TEXT PRIMARY KEY,
                    password TEXT NOT NULL,
                    email TEXT NOT NULL
                )
            """)

            # Desired students schema
            desired_students_cols = [
                "student_id", "name", "father_name", "mother_name",
                "dob", "age", "mobile", "email", "gender",
                "class_name", "address", "photo"
            ]

            # If students table exists, check its columns and migrate if needed
            c.execute("SELECT name FROM sqlite_master WHERE type='table' AND
name='students'")
            if c.fetchone():
                c.execute("PRAGMA table_info(students)")
                existing_cols = [row[1] for row in c.fetchall()]

                # If schema doesn't match desired columns exactly, perform safe migration
                if existing_cols != desired_students_cols:
                    # Create a temporary table with the desired schema
                    c.execute("""
                        CREATE TABLE IF NOT EXISTS students_new (
                            student_id TEXT PRIMARY KEY,
                            name TEXT NOT NULL,
                            father_name TEXT,
                            mother_name TEXT,
                            dob TEXT,
                            age INTEGER,
                            mobile TEXT,
                            email TEXT,
                            gender TEXT,
                            class_name TEXT,
                            address TEXT,
                            photo TEXT
```

```
                )
            """)

            # Copy any matching columns from the old table into the new table.
            cols_to_copy = [col for col in desired_students_cols if col in existing_cols]
            if cols_to_copy:
                cols_list = ",".join(cols_to_copy)
                c.execute(f"INSERT OR IGNORE INTO students_new ({cols_list}) SELECT
{cols_list} FROM students")

            # Replace the old students table with the new one
            c.execute("DROP TABLE students")
            c.execute("ALTER TABLE students_new RENAME TO students")
        else:
            # Create fresh students table with the expected schema
            c.execute("""
                CREATE TABLE IF NOT EXISTS students (
                    student_id TEXT PRIMARY KEY,
                    name TEXT NOT NULL,
                    father_name TEXT,
                    mother_name TEXT,
                    dob TEXT,
                    age INTEGER,
                    mobile TEXT,
                    email TEXT,
                    gender TEXT,
                    class_name TEXT,
                    address TEXT,
                    photo TEXT
                )
            """)

        # Staff Table
        c.execute("""
            CREATE TABLE IF NOT EXISTS staff (
                staff_id TEXT PRIMARY KEY,
                name TEXT NOT NULL,
                dob TEXT,
                age INTEGER,
                mobile TEXT,
                email TEXT,
                gender TEXT,
                designation TEXT,
                subject TEXT,
                salary REAL,
                date_of_joining TEXT,
                address TEXT,
                photo TEXT
            )
        """)

        # Login History Table
        c.execute("""
            CREATE TABLE IF NOT EXISTS login_history (
                id INTEGER PRIMARY KEY AUTOINCREMENT,
                username TEXT,
                login_time TEXT,
                logout_time TEXT
            )
        """)
```

```
            # Postal Codes Table
            c.execute("""
                CREATE TABLE IF NOT EXISTS postal_codes (
                    city TEXT,
                    pin TEXT,
                    state TEXT
                )
            """)

            conn.commit()
            print("Database initialized successfully.")

    except sqlite3.Error as e:
        print("Database error:", e)

initialize_database()

# Recalculate and update student ages based on DOB
def recalculate_student_ages():
    """Update student ages in DB based on their DOB and current date."""
    from datetime import date, datetime
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        c.execute("SELECT student_id, dob FROM students")
        students = c.fetchall()
        today = date.today()
        for sid, dob in students:
            try:
                dob_date = datetime.strptime(dob, "%Y-%m-%d").date()
                new_age = today.year - dob_date.year - ((today.month, today.day) < (dob_date.month,
dob_date.day))
                c.execute("UPDATE students SET age=? WHERE student_id=?", (new_age, sid))
            except Exception:
                continue
        conn.commit()



#ROLL NO. GENERATOR:>
def generate_unique_roll_no():
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        while True:
            roll_no = str(random.randint(100000, 999999))
            c.execute("SELECT 1 FROM students WHERE student_id = ?", (roll_no,))
            if not c.fetchone():
                return roll_no



# Log login and return session ID
def log_login(username):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    login_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute("INSERT INTO login_history (username, login_time) VALUES (?, ?)", (username,
login_time))
    session_id = cursor.lastrowid
    conn.commit()
```

```
    conn.close()
    return session_id



# Log logout using session ID
def log_logout(session_id):
    conn = sqlite3.connect(DB_PATH)
    cursor = conn.cursor()
    logout_time = datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    cursor.execute("UPDATE login_history SET logout_time = ? WHERE id = ?", (logout_time,
session_id))
    conn.commit()
    conn.close()


admin_credentials = {"username": None, "password": None}



#ADMIN PANEL
def open_admin_panel():

    global current_session_id
    if current_session_id:
        log_logout(current_session_id)
        messagebox.showinfo("Logout", "Logged out successfully.")
        current_session_id = None
    for widget in root.winfo_children():
        widget.destroy()
    tk.Label(root, text="ADMIN LOGIN", font=("Arial", 40,
"bold"),fg="darkgoldenrod").pack(pady=30)
    tk.Button(
        root,
        text="Login",
        font=("Arial", 20),
        width=15,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=show_login_form,
    ).pack(pady=10)
    tk.Button(
        root,
        text="Sign Up",
        font=("Arial", 20),
        width=15,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=create_login_form,
    ).pack(pady=10)
    tk.Button(
```

```
        root,
        text="Back",
        font=("Arial", 20),
        width=15,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=main_interface,
    ).pack(pady=30)


#SIGN UP FORM
def create_login_form():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="CREATE ADMIN ACCOUNT", font=("Arial", 40,
"bold"),fg="darkgoldenrod").place(x=500,y=150)

    tk.Label(root, text="Enter Name:", font=("Arial", 20, "bold")).place(x=680,y=300)
    username_entry = tk.Entry(root,width=30)
    username_entry.place(x=600,y=350)

    tk.Label(root, text="Set Password:", font=("Arial", 20, "bold")).place(x=675,y=400)
    password_entry = tk.Entry(root, width=30, show="*")
    password_entry.place(x=600,y=450)



    #TOGGLE PASSWORD
    def toggle_password():
        if password_entry.cget("show") == "":
            password_entry.config(show="*")
            toggle_btn.config(text="🔒 Show Password")
        else:
            password_entry.config(show="")
            toggle_btn.config(text="🔒 Hide Password")



    toggle_btn = tk.Button(root, text="🔒 Show Password", font=("Arial", 15),
command=toggle_password, bg="white", fg="black", bd=0, relief="flat")
    toggle_btn.place(x=660,y=500)

    tk.Label(root, text="Enter Email:", font=("Arial", 20, "bold")).place(x=680,y=550)
    email_entry = tk.Entry(root,width=30)
    email_entry.place(x=600,y=600)

    message_label = tk.Label(root, text="", fg="red")
    message_label.place(x=650, y=690)



    #SAVE SIGN UP CREDENTIALS
```

```
    def save_credentials():
        username = username_entry.get().strip()
        password = password_entry.get().strip()
        email = email_entry.get().strip()

        # Require admin access code before signup
        ACCESS_CODE = "EDU-MATRIX-2025"                                    # Change this code
as needed
        access_code = simpledialog.askstring("Admin Access", "Enter your Admin Access Code:",
show="*")
        if access_code != ACCESS_CODE:
            message_label.config(text="Invalid Access Code. Contact the System Owner.", fg="red")
            return

        if not re.match(r"[^@]+@[^@]+\.[^@]+", email):
            message_label.config(text="Invalid email format", fg="red")
            return

        if not username or not password:
            message_label.config(text="Username and password cannot be empty", fg="red")
            return

        hashed_password = hashlib.sha256(password.encode()).hexdigest()

        try:
            conn = sqlite3.connect(DB_PATH)
            c = conn.cursor()
            # Check for duplicate username
            c.execute("SELECT * FROM admins WHERE username=?", (username,))
            if c.fetchone():
                message_label.config(text="Username already exists. Choose another.", fg="red")
                return
            # Insert new admin
            c.execute("INSERT INTO admins (username, password, email) VALUES (?, ?, ?)",
(username, hashed_password, email))
            conn.commit()
            message_label.config(text="Admin account created!", fg="green")
            def clear_message_label():
                if message_label.winfo_exists():
                    message_label.config(text="")
            root.after(5000, main_interface)
            root.after(5000, clear_message_label)
        except sqlite3.OperationalError as e:
            message_label.config(text=f"Database error: {e}", fg="red")
        except sqlite3.IntegrityError:
            message_label.config(text="Username already exists (IntegrityError)", fg="red")
        finally:
            if conn:
                conn.close()

    tk.Button(root, text="Sign Up", font=("Arial", 15), command=save_credentials,
bg="darkgoldenrod", fg="black").place(x=700,y=650)
    tk.Button(root, text="Back", font=("Arial", 15), command=open_admin_panel,
bg="darkgoldenrod", fg="black").place(x=710,y=750)



# LOGIN FORM
def show_login_form():
```

```
    for widget in root.winfo_children():
        widget.destroy()
    tk.Label(root, text="LOGIN", font=("Arial", 40, "bold"),fg="darkgoldenrod").place(x=670,y=150)
    tk.Label(root, text="Username:", font=("Arial", 20, "bold")).place(x=680,y=300)
    username_entry = tk.Entry(root,width=30)
    username_entry.place(x=600,y=350)
    tk.Label(root, text="Password:", font=("Arial", 20, "bold")).place(x=680,y=400)
    password_entry = tk.Entry(root,width=30, show="*")
    password_entry.place(x=600,y=450)

    #toggle password
    def toggle_password():
        if password_entry.cget("show") == "":
            password_entry.config(show="*")
            toggle_btn.config(text="🔒 Show Password")
        else:
            password_entry.config(show="")
            toggle_btn.config(text="🔓 Hide Password")
    toggle_btn = tk.Button(root, text="🔒 Show Password", font=("Arial", 15),
command=toggle_password, bg="white", fg="black", bd=0, relief="flat")
    toggle_btn.place(x=660,y=500)


    msg_label = tk.Label(root, text="", fg="red")
    msg_label.place(x=680, y=680)
    def authenticate():
        global current_session_id, current_user
        msg_label.config(text="", fg="red")
        conn = sqlite3.connect(DB_PATH)
        cursor = conn.cursor()
        username = username_entry.get()
        password = password_entry.get()
        # Fetch stored password for this username
        cursor.execute("SELECT password FROM admins WHERE username=?", (username,))
        row = cursor.fetchone()
        def clear_msg_label():
            if msg_label.winfo_exists():
                msg_label.config(text="")
        if row:
            stored_hash = row[0]

            try:

                if isinstance(stored_hash, bytes):
                    pass
                elif isinstance(stored_hash, str):
                    try:

                        import binascii
                        maybe_bytes = binascii.unhexlify(stored_hash)

                        if maybe_bytes.startswith(b"$2b$") or maybe_bytes.startswith(b"$2a$"):
                            stored_hash = maybe_bytes
                    except Exception:
                        pass
            except Exception:
                pass
            if verify_password(password, stored_hash):
                current_user = username
```

```
                current_session_id = log_login(username)  # Log login
                msg_label.config(text="Login Successful!", fg="green")
                root.after(1500, open_dashboard)
                root.after(5000, clear_msg_label)
            else:
                msg_label.config(text="Invalid credentials!", fg="red")
                root.after(5000, clear_msg_label)
        else:
            msg_label.config(text="Invalid credentials!", fg="red")
            root.after(5000, clear_msg_label)
        conn.close()

    tk.Button(root, text="Login", font=("Arial", 15), command=authenticate, bg="darkgoldenrod",
fg="black", bd=0, relief="flat", highlightthickness=0).place(x=709,y=550)
    tk.Button(root, text="Back", font=("Arial", 15), command=open_admin_panel,
bg="darkgoldenrod", fg="black", bd=0, relief="flat", highlightthickness=0).place(x=710,y=640)

    tk.Button(root, text="Reset Password", font=("Arial", 15), command=reset_password_form,
bg="darkgoldenrod", fg="black", bd=0, relief="flat", highlightthickness=0).place(x=675, y=600)



# Dashboard after admin login
def open_dashboard():
    for widget in root.winfo_children():
        widget.destroy()

    frame = tk.Frame(root)
    frame.place(relx=0.5, rely=0.5, anchor="center")
    tk.Label(root, text="ADMIN ACCESS", font=("Arial", 40,
"bold"),fg="darkgoldenrod").pack(pady=30)


    root.geometry("2560x1600")


    tk.Button(
        frame,
        text="GENERATE REPORT CARD",
        font=("Arial", 20),
        width=20,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=open_report_card_module
    ).pack(pady=30)

    tk.Button(
        frame,
        text="STUDENT MANAGEMENT",
        font=("Arial", 20),
        width=18,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
```

```
        highlightthickness=0,
        command=open_student_management,
    ).pack(pady=30)

    tk.Button(
        frame,
        text="STAFF MANAGEMENT",
        font=("Arial", 20),
        width=18,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=open_staff_management,
    ).pack(pady=30)

    tk.Button(
        frame,
        text="LOGIN HISTORY",
        font=("Arial", 20),
        width=18,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=open_login_history,
    ).pack(pady=30)

    tk.Button(
        frame,
        text="LOGOUT",
        font=("Arial", 20),
        width=18,
        height=2,
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        bd=0,
        relief="flat",
        highlightthickness=0,
        command=open_admin_panel,
    ).pack(pady=30)


# Report Card Generator Module
import os
from reportlab.lib.pagesizes import A4
from reportlab.pdfgen import canvas
from reportlab.lib.utils import ImageReader
from email.mime.base import MIMEBase
from email import encoders

# grade calculation
def calculate_grade(mark):
    try:
        m = float(mark)
    except Exception:
```

```
        return "E"
    if m >= 90:
        return "A1"
    if m >= 80:
        return "A2"
    if m >= 70:
        return "B1"
    if m >= 60:
        return "B2"
    if m >= 50:
        return "C1"
    if m >= 40:
        return "C2"
    if m >= 33:
        return "D"
    return "E"


def generate_ai_insight(marks_dict):

    valid_marks = []
    low_subj = None
    for s, v in marks_dict.items():
        try:
            vv = float(v)
            valid_marks.append(vv)
            if low_subj is None or vv < low_subj[1]:
                low_subj = (s, vv)
        except Exception:
            continue
    if not valid_marks:
        return "No marks provided to generate insight."
    avg = sum(valid_marks) / len(valid_marks)
    insight = []
    if avg >= 85:
        insight.append("Outstanding performance overall — keep it up!")
    elif avg >= 70:
        insight.append("Good performance with scope to reach excellence.")
    elif avg >= 50:
        insight.append("Average performance — focused effort can improve results.")
    else:
        insight.append("Needs significant improvement; consider extra practice and guidance.")
    if low_subj and low_subj[1] < 50:
        insight.append(f"Focus on {low_subj[0]} which scored {int(low_subj[1])}.")

    if avg < 70:
        insight.append("Recommend weekly practice tests and teacher feedback.")
    return " ".join(insight)

from reportlab.lib.pagesizes import A4
from reportlab.lib import colors
from reportlab.lib.units import inch
from reportlab.lib.utils import ImageReader
from reportlab.pdfgen import canvas
from reportlab.platypus import Table, TableStyle
from datetime import datetime
import textwrap

# PDF Report Card Generation
def generate_report_pdf(student, class_name, marks_dict, remark):
```

```
    import re

    safe_name = re.sub(r"[^A-Za-z0-9]+", "_", student.get('name', 'student')).strip("_") or "student"
    sid = student.get('student_id', '')
    os.makedirs("ReportCards", exist_ok=True)
    file_path = os.path.join("ReportCards", f"ReportCard_{safe_name}_{sid}.pdf")


    now = datetime.now()
    if now.month < 6:
        year_text = f"ACADEMIC YEAR {now.year-1}-{now.year}"
    else:
        year_text = f"ACADEMIC YEAR {now.year}-{now.year+1}"

    width, height = A4
    c = canvas.Canvas(file_path, pagesize=A4)


    header_height = 72
    c.setFillColor(colors.black)
    c.rect(0, height - header_height, width, header_height, stroke=0, fill=1)


    try:
        logo_path = "/Users/vittalvishnu/Documents/aisms/logo.png"
        if os.path.exists(logo_path):
            logo = ImageReader(logo_path)
            logo_w, logo_h = 55, 55
            c.drawImage(logo, 40, height - header_height + (header_height - logo_h) / 2,
width=logo_w, height=logo_h, mask='auto')
    except Exception as e:
        print("Logo draw error:", e)


    c.setFillColor(colors.white)
    c.setFont("Helvetica-Bold", 14)
    c.drawCentredString(width/2, height - 25, "EDUMATRIX SCHOOL MANAGEMENT SYSTEM")
    c.setFont("Helvetica", 9)
    c.drawCentredString(width/2, height - 40, "AFFILIATED TO CBSE | MANAGED BY AI
AUTOMATION UNIT")


    c.saveState()
    c.translate(width/2, height/2)
    c.rotate(45)
    c.setFont("Helvetica-Bold", 80)
    c.setFillColorRGB(0.85, 0.85, 0.85)
    try:
        c.setFillAlpha(0.2)
    except Exception:
        pass
    c.drawCentredString(0, 0, "EDUMATRIX")
    c.restoreState()


    exam_y = height - header_height - 30
    c.setFillColor(colors.black)
    c.setFont("Helvetica-Bold", 14)
    c.drawCentredString(width / 2, exam_y, f"FINAL EXAMINATION – CLASS
{class_name.upper()}")
```

```
    c.setFont("Helvetica", 12)
    c.drawCentredString(width / 2, exam_y - 20, year_text)


    data = [["SUBJECT", "MARKS", "GRADE"]]
    for sub, mark in marks_dict.items():
        grade = calculate_grade(mark)
        data.append([sub, str(mark), grade])

    table = Table(data, colWidths=[2.8*inch, 1.3*inch, 1.2*inch])
    table.setStyle(TableStyle([
        ('BACKGROUND', (0,0), (-1,0), colors.lightgrey),
        ('TEXTCOLOR', (0,0), (-1,0), colors.black),
        ('ALIGN', (0,0), (-1,-1), 'CENTER'),
        ('FONTNAME', (0,0), (-1,0), 'Helvetica-Bold'),
        ('FONTSIZE', (0,0), (-1,-1), 11),
        ('BOTTOMPADDING', (0,0), (-1,0), 8),
        ('GRID', (0,0), (-1,-1), 0.8, colors.black),
        ('ROWBACKGROUNDS', (0,1), (-1,-1), [colors.whitesmoke, colors.lightgrey])
    ]))


    table.wrapOn(c, width, height)
    table_width, table_height = table.wrap(0, 0)
    table_x = (width - table_width) / 2

    table_y = exam_y - 60 - table_height

    footer_margin = 120
    if table_y < footer_margin:
        c.showPage()
        width, height = A4
        c.setFont("Helvetica-Bold", 12)
        c.drawCentredString(width/2, height - 40, f"CONTINUED - FINAL EXAMINATION – CLASS
{class_name.upper()}")
        table_y = height - 150 - table_height
        table.wrapOn(c, width, height)
        table.drawOn(c, table_x, table_y)
    else:
        table.drawOn(c, table_x, table_y)

    # AI Remark
    remark_start_y = table_y - 30
    c.setFont("Helvetica-Bold", 12)
    c.drawString(60, remark_start_y, "AI Remark:")
    c.setFont("Helvetica", 11)
    text = c.beginText(60, remark_start_y - 16)
    max_width = 90
    for line in textwrap.wrap(remark, width=max_width):
        text.textLine(line)
    c.drawText(text)


    footer_y = 80
    c.setFont("Helvetica", 10)
    c.drawString(60, footer_y, f"Generated on: {now.strftime('%d-%m-%Y')}")


    try:
        signature_path = "/Users/vittalvishnu/Documents/aisms/signature.png"
```

```
        if os.path.exists(signature_path):
            sig = ImageReader(signature_path)
            sig_w, sig_h = 120, 50
            sig_x, sig_y = width - sig_w - 70, footer_y - 10
            c.drawImage(sig, sig_x, sig_y, width=sig_w, height=sig_h, mask='auto')
            c.setFont("Helvetica-Bold", 10)
            c.drawString(sig_x, sig_y - 12, "Vishnu Vittal")
            c.setFont("Helvetica", 9)
            c.drawString(sig_x, sig_y - 24, "Administrator, EDUMATRIX")
        else:
            sig_x = width - 220
            c.setFont("Helvetica-Bold", 10)
            c.drawString(sig_x, footer_y + 2, "Vishnu Vittal")
            c.setFont("Helvetica", 9)
            c.drawString(sig_x, footer_y - 10, "Administrator, EDUMATRIX")
    except Exception as e:
        print("Signature draw error:", e)

    c.showPage()
    c.save()
    return file_path

# send PDF via email
def send_report_via_email(to_email, file_path, student_name, class_name):

    if not to_email or not re.match(r"[^@]+@[^@]+\.[^@]+", to_email):
        print("Invalid recipient email:", to_email)
        return False

    try:
        sender_email = "smspythonproj@gmail.com"

        sender_password = os.environ.get("EDUMATRIX_SMTP_PASS")
        if not sender_password:

            sender_password = "glpg cdym oigb cjxi"
            print("Warning: Using fallback SMTP password. Consider setting
EDUMATRIX_SMTP_PASS env var for security.")

        subject = f"EDUMATRIX Report Card - Class {class_name}"
        body = f"Dear {student_name},\n\nYour report card for Class {class_name} is attached.
\n\nBest regards,\nEDUMATRIX School Management System"

        msg = MIMEMultipart()
        msg['From'] = sender_email
        msg['To'] = to_email
        msg['Subject'] = subject
        msg.attach(MIMEText(body, 'plain'))


        if not os.path.exists(file_path):
            print("Attachment not found:", file_path)
            return False
        with open(file_path, 'rb') as f:
            part = MIMEBase('application', 'octet-stream')
            part.set_payload(f.read())
        encoders.encode_base64(part)
        part.add_header('Content-Disposition', f'attachment;
filename="{os.path.basename(file_path)}"')
        msg.attach(part)
```

```
        server = smtplib.SMTP('smtp.gmail.com', 587, timeout=30)
        server.starttls()
        server.login(sender_email, sender_password)
        server.send_message(msg)
        server.quit()
        return True
    except Exception as e:
        print("Email send failed:", e)
        return False


def open_report_card_module():

    win = tk.Toplevel(root)
    win.title("REPORT CARD GENERATOR")
    win.geometry("2560x1600")

    tk.Label(win, text="REPORT CARD GENERATOR", font=("Arial", 30, "bold"),
fg="darkgoldenrod").pack(pady=20)

    # Fetch students
    students = []
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        c.execute("SELECT student_id, name, class_name, email FROM students ORDER BY name
COLLATE NOCASE ASC")
        rows = c.fetchall()
        for r in rows:
            students.append({'student_id': r[0], 'name': r[1], 'class': r[2], 'email': r[3]})

    student_var = tk.StringVar(value="Select Student")
    student_menu = ttk.Combobox(win, values=[f"{s['name']} ({s['student_id']})" for s in students],
state='readonly', width=60)
    student_menu.pack(pady=10)

    info_frame = tk.Frame(win)
    info_frame.pack(pady=10)

    tk.Label(info_frame, text="Class:" ).grid(row=0, column=0, padx=5, pady=5)
    class_entry = tk.Entry(info_frame)
    class_entry.grid(row=0, column=1, padx=5, pady=5)
    tk.Label(info_frame, text="Roll/ID:" ).grid(row=0, column=2, padx=5, pady=5)
    roll_entry = tk.Entry(info_frame)
    roll_entry.grid(row=0, column=3, padx=5, pady=5)
    tk.Label(info_frame, text="Email:" ).grid(row=0, column=4, padx=5, pady=5)
    email_entry = tk.Entry(info_frame, width=40)
    email_entry.grid(row=0, column=5, padx=5, pady=5)

    subjects_frame = tk.Frame(win)
    subjects_frame.pack(pady=20)

    subject_entries = {}

    def load_student(event=None):
        sel = student_menu.get()
        if not sel:
            return
        sid = sel.split('(')[-1].strip(')')
        for s in students:
```

```
            if s['student_id'] == sid:
                class_entry.delete(0, tk.END)
                class_entry.insert(0, s['class'])
                roll_entry.delete(0, tk.END)
                roll_entry.insert(0, s['student_id'])
                email_entry.delete(0, tk.END)
                email_entry.insert(0, s['email'] if s['email'] else '')
                break
        build_subject_fields()

    def build_subject_fields():
        for w in subjects_frame.winfo_children():
            w.destroy()
        subject_entries.clear()
        cls = class_entry.get().strip()
        try:
            cls_num = int(cls)
        except Exception:
            cls_num = None
        if cls_num and 1 <= cls_num <= 10:
            subs = ["Maths", "English", "Science", "Social Science", "Hindi/Kannada"]
        else:
            # 11 and 12
            subs = ["Physics", "Chemistry", "Maths", "English", "Computer Science/Biology"]
        for i, s in enumerate(subs):
            tk.Label(subjects_frame, text=s+":", font=("Arial", 16)).grid(row=i, column=0, sticky="e",
padx=5, pady=6)
            e = tk.Entry(subjects_frame, width=10)
            e.grid(row=i, column=1, padx=8, pady=6)
            subject_entries[s] = e

    student_menu.bind("<<ComboboxSelected>>", load_student)


    btn_frame = tk.Frame(win)
    btn_frame.pack(pady=20)

    def on_save():
        student_name = ''
        student_id = roll_entry.get().strip()
        class_name = class_entry.get().strip()
        email = email_entry.get().strip()


        if not student_id:
            messagebox.showerror("Missing Student", "Please select a student or enter Roll/ID.")
            return
        for s in students:
            if s['student_id'] == student_id:
                student_name = s['name']
                break
        if not student_name:
            messagebox.showerror("Student not found", "Selected student not found in database.")
            return

        # Validate Marks Input (0 - 100 only)
        marks = {}
        for subject, entry in subject_entries.items():
            val = entry.get().strip()
            if val == "":
```

```
                messagebox.showerror("Missing Marks", f"Please enter marks for {subject}.")
                return
            try:
                mark = float(val)
            except ValueError:
                messagebox.showerror("Invalid Input", f"Please enter a valid number for {subject}.")
                return
            if mark < 0 or mark > 100:
                messagebox.showerror("Invalid Marks", f"Marks for {subject} must be between 0 and
100.")
                return
            marks[subject] = mark

        remark = generate_ai_insight({k: v for k, v in marks.items()})
        pdf_path = generate_report_pdf({'name': student_name, 'student_id': student_id, 'email':
email}, class_name, marks, remark)
        if pdf_path:
            messagebox.showinfo("Saved", f"Report card saved: {pdf_path}")
        else:
            messagebox.showerror("Error", "Failed to generate PDF")

    def on_send():
        student_name = ''
        student_id = roll_entry.get().strip()
        class_name = class_entry.get().strip()
        email = email_entry.get().strip()


        if not student_id:
            messagebox.showerror("Missing Student", "Please select a student or enter Roll/ID.")
            return
        for s in students:
            if s['student_id'] == student_id:
                student_name = s['name']
                break
        if not student_name:
            messagebox.showerror("Student not found", "Selected student not found in database.")
            return


        marks = {}
        for subject, entry in subject_entries.items():
            val = entry.get().strip()
            if val == "":
                messagebox.showerror("Missing Marks", f"Please enter marks for {subject}.")
                return
            try:
                mark = float(val)
            except ValueError:
                messagebox.showerror("Invalid Input", f"Please enter a valid number for {subject}.")
                return
            if mark < 0 or mark > 100:
                messagebox.showerror("Invalid Marks", f"Marks for {subject} must be between 0 and
100.")
                return
            marks[subject] = mark


        if not email or not re.match(r"[^@]+@[^@]+\.[^@]+", email):
```

```
            messagebox.showerror("Invalid Email", "Please enter a valid recipient email before
sending.")
            return

        remark = generate_ai_insight({k: v for k, v in marks.items()})
        pdf_path = generate_report_pdf({'name': student_name, 'student_id': student_id, 'email':
email}, class_name, marks, remark)
        if not pdf_path:
            messagebox.showerror("Error", "Failed to generate PDF")
            return

        sent = send_report_via_email(email, pdf_path, student_name, class_name)
        if sent:
            messagebox.showinfo("Sent", "Report card emailed successfully.")
        else:
            messagebox.showerror("Error", "Failed to send email. Check SMTP config and internet
connection.")

    tk.Button(btn_frame, text="Save Report Card", font=("Arial", 16), bg="darkgoldenrod",
fg="black", command=on_save).pack(side="left", padx=10)
    tk.Button(btn_frame, text="Send a Copy", font=("Arial", 16), bg="darkgoldenrod", fg="black",
command=on_send).pack(side="left", padx=10)
    tk.Button(btn_frame, text="Close", font=("Arial", 16), bg="white", fg="black",
command=win.destroy).pack(side="left", padx=10)

# Login History Page
def open_login_history():
    for widget in root.winfo_children():
        widget.destroy()


    root.geometry("2560x1600")


    tk.Label(root, text="LOGIN HISTORY", font=("Arial", 40, "bold"),
fg="darkgoldenrod").pack(pady=20)

    table_frame = tk.Frame(root)
    table_frame.pack(fill="both", expand=True, padx=20, pady=10)

    columns = ("username", "login_time", "logout_time")
    tree = ttk.Treeview(table_frame, columns=columns, show="headings")

    for col in columns:
        tree.heading(col, text=col.replace("_", " ").title())
        tree.column(col, anchor="center", width=300)

    scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
    tree.configure(yscrollcommand=scrollbar.set)
    tree.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    # Fetch login history
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        c.execute("SELECT username, login_time, logout_time FROM login_history ORDER BY
login_time DESC")
        rows = c.fetchall()
        if not rows:
            tree.insert("", "end", values=("No records found", "", ""))
```

```
        else:
            for r in rows:
                tree.insert("", "end", values=r)

    tk.Button(root, text="Back", font=("Arial", 15), command=open_dashboard,
bg="darkgoldenrod", fg="black").pack(pady=20)



#
def generate_unique_staff_id():
    with sqlite3.connect(DB_PATH) as conn:
        c = conn.cursor()
        while True:
            staff_id = "STF" + str(random.randint(1000, 9999))
            c.execute("SELECT 1 FROM staff WHERE staff_id = ?", (staff_id,))
            if not c.fetchone():
                return staff_id

def open_add_staff_form():
    for widget in root.winfo_children():
        widget.destroy()
    tk.Label(root, text="ADD NEW STAFF", font=("Arial", 30, "bold"),
fg="darkgoldenrod").pack(pady=20)

    staff_id = generate_unique_staff_id()

    staff_id_label = tk.Label(root, text=f"Staff ID: {staff_id}", font=("Arial", 14))
    staff_id_label.place(relx=0.9, rely=0.05, anchor="ne")


    tk.Label(root, text="Name:").pack()

    name_frame = tk.Frame(root)
    name_frame.pack(pady=5)
    first_name_entry = tk.Entry(name_frame, width=20)
    first_name_entry.insert(0, "First Name")
    first_name_entry.pack(side="left", padx=5)
    last_name_entry = tk.Entry(name_frame, width=20)
    last_name_entry.insert(0, "Last Name")
    last_name_entry.pack(side="left", padx=5)

    def clear_name_placeholder(e):
        if e.widget.get() in ["First Name", "Last Name"]:
            e.widget.delete(0, tk.END)

    def restore_name_placeholder(e):
        if e.widget.get().strip() == "":
            if e.widget == first_name_entry:
                e.widget.insert(0, "First Name")
            elif e.widget == last_name_entry:
                e.widget.insert(0, "Last Name")

    first_name_entry.bind("<FocusIn>", clear_name_placeholder)
    last_name_entry.bind("<FocusIn>", clear_name_placeholder)
    first_name_entry.bind("<FocusOut>", restore_name_placeholder)
    last_name_entry.bind("<FocusOut>", restore_name_placeholder)

    # upload photo
    photo_path = tk.StringVar()
```

```
    def upload_staff_photo():
        file_path = filedialog.askopenfilename(
            title="Select Staff Photo",
            filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif")]
        )
        if file_path:
            photo_path.set(file_path)
            selected_label.config(text=f"Selected: {os.path.basename(file_path)}")
    tk.Button(root, text="Upload Photo", command=upload_staff_photo, bg="darkgoldenrod",
fg="black").pack(pady=5)
    selected_label = tk.Label(root, text="", fg="gray")
    selected_label.pack()

    # DOB
    tk.Label(root, text="Date of Birth:").pack()
    dob_frame = tk.Frame(root)
    dob_frame.pack()
    dob_day_var = tk.StringVar(value="Day")
    dob_month_var = tk.StringVar(value="Month")
    dob_year_var = tk.StringVar(value="Year")
    tk.OptionMenu(dob_frame, dob_day_var, *[str(i) for i in range(1, 32)]).pack(side="left", padx=2)
    tk.OptionMenu(dob_frame, dob_month_var, *[str(i) for i in range(1, 13)]).pack(side="left",
padx=2)
    tk.OptionMenu(dob_frame, dob_year_var, *[str(i) for i in range(1950, 2031)]).pack(side="left",
padx=2)

    # mobile
    tk.Label(root, text="Mobile:").pack()
    mobile_entry = tk.Entry(root)
    mobile_entry.pack()
    # email
    tk.Label(root, text="Email:").pack()
    email_entry = tk.Entry(root)
    email_entry.pack()
    # designation
    tk.Label(root, text="Designation:").pack()
    designation_entry = tk.Entry(root)
    designation_entry.pack()
    # subject
    tk.Label(root, text="Subject:").pack()
    subject_entry = tk.Entry(root)
    subject_entry.pack()
    # salary
    tk.Label(root, text="Salary:").pack()
    salary_entry = tk.Entry(root)
    salary_entry.pack()
    # date of joining
    tk.Label(root, text="Date of Joining:").pack()
    doj_frame = tk.Frame(root)
    doj_frame.pack()
    doj_day_var = tk.StringVar(value="Day")
    doj_month_var = tk.StringVar(value="Month")
    doj_year_var = tk.StringVar(value="Year")
    tk.OptionMenu(doj_frame, doj_day_var, *[str(i) for i in range(1, 32)]).pack(side="left", padx=2)
    tk.OptionMenu(doj_frame, doj_month_var, *[str(i) for i in range(1, 13)]).pack(side="left", padx=2)
    tk.OptionMenu(doj_frame, doj_year_var, *[str(i) for i in range(1950, 2031)]).pack(side="left",
padx=2)
    # address
    tk.Label(root, text="Address:").pack()
```

```
    street_entry = tk.Entry(root, width=30)
    street_entry.insert(0, "Street Address")
    street_entry.pack(pady=2)

    city_entry = tk.Entry(root, width=30)
    city_entry.insert(0, "City")
    city_entry.pack(pady=2)

    pin_entry = tk.Entry(root, width=30)
    pin_entry.insert(0, "PIN Code")
    pin_entry.pack(pady=2)

    state_entry = tk.Entry(root, width=30)
    state_entry.insert(0, "State")
    state_entry.pack(pady=2)

    country_var = tk.StringVar(value="India")
    country_label = tk.Label(root, text="Country: India", anchor="w", width=30, fg="grey")
    country_label.pack(pady=2)


    def clear_placeholder(e):
        if e.widget.get() in ["Street Address", "City", "PIN Code", "State"]:
            e.widget.delete(0, tk.END)

    def restore_placeholder(e):
        if e.widget.get().strip() == "":
            if e.widget == street_entry:
                e.widget.insert(0, "Street Address")
            elif e.widget == city_entry:
                e.widget.insert(0, "City")
            elif e.widget == pin_entry:
                e.widget.insert(0, "PIN Code")
            elif e.widget == state_entry:
                e.widget.insert(0, "State")

    for entry in [street_entry, city_entry, pin_entry, state_entry]:
        entry.bind("<FocusIn>", clear_placeholder)
        entry.bind("<FocusOut>", restore_placeholder)
    # Postal autofill for city entry
    def autofill_pin_state(event):
        city = city_entry.get().strip().lower()
        if city in postal_data:
            pin_entry.delete(0, tk.END)
            pin_entry.insert(0, postal_data[city][0])
            state_entry.delete(0, tk.END)
            state_entry.insert(0, postal_data[city][1].title())

    city_entry.bind("<KeyRelease>", autofill_pin_state)
    # gender
    gender_var = tk.StringVar()
    gender_frame = tk.Frame(root)
    gender_frame.pack()
    tk.Label(gender_frame, text="Gender:").pack(side="left")
    tk.Radiobutton(gender_frame, text="Male", variable=gender_var,
value="Male").pack(side="left")
    tk.Radiobutton(gender_frame, text="Female", variable=gender_var,
value="Female").pack(side="left")

    error_label = tk.Label(root, text="", fg="red")
```

```
    error_label.pack()

    def save_staff():
        first_name = first_name_entry.get().strip()
        last_name = last_name_entry.get().strip()

        if not first_name or first_name.lower() == "first name":
            error_label.config(text="First Name is required!")
            return
        if not re.fullmatch(r"[A-Za-z ]+", first_name):
            error_label.config(text="First Name must contain only letters and spaces.")
            return
        if last_name and last_name.lower() != "last name":
            if not re.fullmatch(r"[A-Za-z ]+", last_name):
                error_label.config(text="Last Name must contain only letters and spaces.")
                return

        name = first_name
        if last_name and last_name.lower() != "last name":
            name += " " + last_name
        # dob
        try:
            dob = f"{dob_year_var.get()}-{dob_month_var.get().zfill(2)}-{dob_day_var.get().zfill(2)}"
            dob_date = datetime.strptime(dob, "%Y-%m-%d")
            today = date.today()
            age = today.year - dob_date.year - ((today.month, today.day) < (dob_date.month,
dob_date.day))
        except Exception:
            error_label.config(text="Invalid DOB! Please pick valid day, month, year.")
            return
        # mobile
        mobile = mobile_entry.get().strip()
        if not re.match(r'^[6-9]\d{9}$', mobile):
            error_label.config(text="Invalid mobile number")
            return
        # email
        email = email_entry.get().strip()
        if not re.match(r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$', email):
            error_label.config(text="Invalid email format")
            return
        # designation
        designation = designation_entry.get().strip()
        # subject
        subject = subject_entry.get().strip()
        # salary
        salary = salary_entry.get().strip()
        try:
            salary_val = float(salary)
        except Exception:
            error_label.config(text="Invalid salary")
            return
        # date of joining
        try:
            doj = f"{doj_year_var.get()}-{doj_month_var.get().zfill(2)}-{doj_day_var.get().zfill(2)}"

            _ = datetime.strptime(doj, "%Y-%m-%d")
        except Exception:
            error_label.config(text="Invalid Date of Joining! Please pick valid day, month, year.")
            return
        # address
```

```
        address = f"{street_entry.get().strip()}, {city_entry.get().strip()}, {pin_entry.get().strip()},
{state_entry.get().strip()}, INDIA".upper()
        gender = gender_var.get()
        if not name or not dob or not mobile or not email:
            error_label.config(text="Name, DOB, Mobile, Email are required!")
            return
        try:
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()

                c.execute("PRAGMA table_info(staff)")
                columns = [row[1] for row in c.fetchall()]
                if "photo" not in columns:
                    c.execute("ALTER TABLE staff ADD COLUMN photo TEXT")
                c.execute("""
                    INSERT INTO staff (staff_id, name, dob, age, mobile, email, gender, designation,
subject, salary, date_of_joining, address, photo)
                    VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
                """, (staff_id, name, dob, age, mobile, email, gender, designation, subject, salary_val,
doj, address, photo_path.get()))
                conn.commit()
            error_label.config(text="Staff added successfully!", fg="green")
            root.after(1500, open_staff_management)
        except sqlite3.Error as e:
            error_label.config(text=f"Database error: {e}")

    tk.Button(root, text="Save Staff", command=save_staff, bg="darkgoldenrod",
fg="black").pack(pady=10)
    tk.Button(root, text="Back", command=open_staff_management, bg="darkgoldenrod",
fg="black").pack(pady=10)

def view_staff():
    for widget in root.winfo_children():
        widget.destroy()

    root.geometry("1600x900")
    tk.Label(root, text="STAFF LIST", font=("Arial", 25, "bold"), fg="darkgoldenrod").pack(pady=10)

    search_var = tk.StringVar()
    search_frame = tk.Frame(root)
    search_frame.pack(pady=10)
    tk.Label(search_frame, text="🔎Search (ID or Name):").pack(side="left")
    tk.Entry(search_frame, textvariable=search_var, width=30).pack(side="left", padx=5)

    # sort menu for staff
    sort_by = "name"
    def apply_sort(new_sort_by):
        nonlocal sort_by
        sort_by = new_sort_by
        update_treeview()
    sort_button = tk.Menubutton(root, text="Sort", relief="raised", font=("Arial", 12), bg="white",
fg="black")
    sort_menu = tk.Menu(sort_button, tearoff=0)
    sort_menu.add_command(label="Sort by Name", command=lambda: apply_sort("name"))
    sort_menu.add_command(label="Sort by Staff ID", command=lambda: apply_sort("staff_id"))
    sort_menu.add_command(label="Sort by Designation", command=lambda:
apply_sort("designation"))
    sort_button["menu"] = sort_menu
    sort_button.pack(pady=5)
```

```
    table_frame = tk.Frame(root)
    table_frame.pack(fill="both", expand=True, padx=20, pady=10)

    columns = ("staff_id", "name", "designation", "mobile", "email")
    tree = ttk.Treeview(table_frame, columns=columns, show="headings")

    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, anchor="center", width=200, stretch=True)

    scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
    tree.configure(yscrollcommand=scrollbar.set)
    tree.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")

    def update_treeview(*args):
        for item in tree.get_children():
            tree.delete(item)
        keyword = search_var.get()
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            if keyword:
                c.execute(f"SELECT staff_id, name, designation, mobile, email FROM staff WHERE
name LIKE ? OR staff_id LIKE ? ORDER BY {sort_by} COLLATE NOCASE ASC",
                          (f"%{keyword}%", f"%{keyword}%"))
            else:
                c.execute(f"SELECT staff_id, name, designation, mobile, email FROM staff ORDER BY
{sort_by} COLLATE NOCASE ASC")
            rows = c.fetchall()
        if not rows:
            tree.insert("", "end", values=("No records found", "", "", "", ""))
        else:
            for r in rows:
                tree.insert("", "end", values=r)

    search_var.trace_add("write", update_treeview)
    update_treeview()

    def on_double_click(event):
        selected = tree.selection()
        if not selected:
            return
        staff_id = tree.item(selected[0])["values"][0]
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM staff WHERE staff_id = ?", (staff_id,))
            profile = c.fetchone()
        if profile:
            open_staff_profile(profile, refresh_callback=update_treeview)

    tree.bind("<Double-1>", on_double_click)

    tk.Button(root, text="Back", command=open_staff_management, bg="darkgoldenrod",
fg="black").pack(pady=20)

def open_staff_profile(profile, refresh_callback=None):
    window = tk.Toplevel(root)
    window.title("Staff Profile")
    window.geometry("2560x1600")
```

```
    left_frame = tk.Frame(window)
    left_frame.grid(row=0, column=0, padx=50, pady=20, sticky="n")
    right_frame = tk.Frame(window)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")

    labels_ordered = [
        ("Staff ID", profile[0]),
        ("Name", profile[1]),
        ("Gender", profile[6]),
        ("DOB", profile[2]),
        ("Age", profile[3]),
        ("Mobile", profile[4]),
        ("Email", profile[5]),
        ("Designation", profile[7]),
        ("Subject", profile[8]),
        ("Salary", profile[9]),
        ("Date of Joining", profile[10]),
        ("Address", profile[11])
    ]
    editable_fields = {"Name", "Mobile", "Email", "Designation", "Subject", "Salary", "Address"}
    entry_widgets = {}

    for i, (label, value) in enumerate(labels_ordered):
        tk.Label(left_frame, text=label + ":", font=("Arial", 18, "bold")).grid(row=i, column=0,
sticky="w", pady=8)
        if label in editable_fields:
            e = tk.Entry(left_frame, width=40, font=("Arial", 16))
            e.insert(0, value if value else "")
            e.grid(row=i, column=1, pady=8)
            entry_widgets[label] = e
        else:
            tk.Label(left_frame, text=value if value else "", font=("Arial", 16), fg="gray").grid(row=i,
column=1, sticky="w", pady=8)
            entry_widgets[label] = None


    right_frame = tk.Frame(window, width=400, height=900)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")
    right_frame.grid_propagate(False)  # keep frame size fixed


    photo_label = tk.Label(right_frame, bg="lightgray")
    photo_label.pack(fill="both", expand=True)


    photo_path = profile[12] if len(profile) > 12 else None
    if photo_path:
        try:
            img = Image.open(photo_path)
            frame_width = right_frame.winfo_reqwidth()
            frame_height = right_frame.winfo_reqheight()
            img_ratio = img.width / img.height
            frame_ratio = frame_width / frame_height
            if frame_ratio > img_ratio:

                new_height = frame_height
                new_width = int(img_ratio * new_height)
            else:

                new_width = frame_width
```

```
            new_height = int(new_width / img_ratio)
            img = img.resize((new_width, new_height))
            photo_img = ImageTk.PhotoImage(img)
            photo_label.config(image=photo_img)
            photo_label.photo_img = photo_img                    # keep reference
        except:
            photo_label.config(text="Photo not available")


    button_frame = tk.Frame(window)
    button_frame.grid(row=1, column=0, columnspan=2, pady=40)

    def save_changes():
        try:
            updates = {field: entry_widgets[field].get().strip() for field in editable_fields}
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("""
                    UPDATE staff SET name=?, mobile=?, email=?, designation=?, subject=?, salary=?,
address=? WHERE staff_id=?
                """, (updates["Name"], updates["Mobile"], updates["Email"], updates["Designation"],
                      updates["Subject"], updates["Salary"], updates["Address"], profile[0]))
                conn.commit()
            tk.messagebox.showinfo("Success", "Staff updated successfully.")
            window.destroy()
            if refresh_callback:
                refresh_callback()
        except Exception as e:
            tk.messagebox.showerror("Error", str(e))

    def delete_staff():
        confirm = tk.messagebox.askyesno("Confirm Deletion", "Delete this staff?")
        if confirm:
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("DELETE FROM staff WHERE staff_id=?", (profile[0],))
                conn.commit()
            tk.messagebox.showinfo("Deleted", "Staff deleted successfully.")
            window.destroy()
            if refresh_callback:
                refresh_callback()

    tk.Button(button_frame, text="Save", width=15, command=save_changes).pack(side="left",
padx=20)
    tk.Button(button_frame, text="Delete", width=15, command=delete_staff).pack(side="left",
padx=20)

def open_staff_management():
    for widget in root.winfo_children():
        widget.destroy()
    tk.Label(root, text="STAFF MANAGEMENT", font=("Arial", 60, "bold"),
fg="darkgoldenrod").pack(pady=30)

    tk.Button(root, text="Add New Staff", font=("Arial", 20), width=20, height=2,
              bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
              highlightthickness=0, command=open_add_staff_form).pack(pady=10)
    tk.Button(root, text="View Staff", font=("Arial", 20), width=20, height=2,
              bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
              highlightthickness=0, command=view_staff).pack(pady=10)
    tk.Button(root, text="Back", font=("Arial", 20), width=20, height=2,
```

```
            bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
            highlightthickness=0, command=open_dashboard).pack(pady=30)


# Reset Password Form
def reset_password_form():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="Reset Password", font=("Arial", 20, "bold")).pack(pady=20)

    tk.Label(root, text="Enter your username:").pack()
    username_entry = tk.Entry(root)
    username_entry.pack()

    message_label = tk.Label(root, text="", fg="red")
    message_label.pack()



    # email verification code
    from datetime import datetime
    def send_verification_code():
        username = username_entry.get().strip()
        if not username:
            message_label.config(text="Username cannot be empty", fg="red")
            return

        # Fetch email from DB
        conn = sqlite3.connect(DB_PATH)
        c = conn.cursor()
        c.execute("SELECT email FROM admins WHERE username = ?", (username,))
        result = c.fetchone()
        conn.close()

        if not result:
            message_label.config(text="Username not found", fg="red")
            return

        user_email = result[0]
        code = str(random.randint(100000, 999999))
        # Store code, timestamp, and attempts in memory for security
        verification_codes[username] = {
            "code": code,
            "timestamp": datetime.now(),
            "attempts": 0
        }

        # Send the code via email
        try:
            sender_email = "smspythonproj@gmail.com"
            sender_password = "glpg cdym oigb cjxi"
            subject = "Your Password Reset Verification Code"

            msg = MIMEMultipart()
            msg['From'] = sender_email
            msg['To'] = user_email
            msg['Subject'] = subject
```

```
            body = f"Hello {username},\n\nYour verification code is: {code}\n\nUse this ☝️ to reset
your password.\n\n⚠️  Please do not share this code with anybody for security reasons.
⚠️\n\n🙏  Thank You🙏"
            msg.attach(MIMEText(body, 'plain'))

            server = smtplib.SMTP('smtp.gmail.com', 587)
            server.starttls()
            server.login(sender_email, sender_password)
            server.send_message(msg)
            server.quit()

            message_label.config(text="Verification code sent to your email", fg="green")
            def clear_message_label():
                if message_label.winfo_exists():
                    message_label.config(text="")
            root.after(5000, clear_message_label)
            ask_for_code(username)

        except Exception as e:
            message_label.config(text=f"Failed to send email: {str(e)}", fg="red")
            def clear_message_label():
                if message_label.winfo_exists():
                    message_label.config(text="")
            root.after(5000, clear_message_label)

    tk.Button(root, text="Send Code", command=send_verification_code, bg="white",
fg="black").pack(pady=10)
    tk.Button(root, text="Back", command=show_login_form, bg="darkgoldenrod",
fg="black").pack(pady=10)



# Ask for verification code
def ask_for_code(username):
    from datetime import datetime, timedelta
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="Enter Verification Code", font=("Arial", 20, "bold")).pack(pady=20)

    tk.Label(root, text="Code sent to your email:").pack()
    code_entry = tk.Entry(root)
    code_entry.pack()

    message_label = tk.Label(root, text="", fg="red")
    message_label.pack()

    MAX_ATTEMPTS = 3
    OTP_EXPIRY_MINUTES = 5

    def verify_code():
        entered_code = code_entry.get().strip()
        def clear_message_label():
            if message_label.winfo_exists():
                message_label.config(text="")

        code_info = verification_codes.get(username)
        if not code_info:
```

```
            message_label.config(text="No active verification code. Please request a new code.",
fg="red")
            root.after(5000, clear_message_label)
            root.after(2000, reset_password_form)
            return

        # Check expiry
        now = datetime.now()
        code_time = code_info.get("timestamp")
        if now - code_time > timedelta(minutes=OTP_EXPIRY_MINUTES):
            del verification_codes[username]
            message_label.config(text="Code expired. Please request a new code.", fg="red")
            root.after(5000, clear_message_label)
            root.after(2000, reset_password_form)
            return

        # Check attempts
        if code_info.get("attempts", 0) >= MAX_ATTEMPTS:
            del verification_codes[username]
            message_label.config(text="Maximum attempts exceeded. Please request a new code.",
fg="red")
            root.after(5000, clear_message_label)
            root.after(2000, reset_password_form)
            return


        if entered_code == code_info.get("code"):
            del verification_codes[username]
            message_label.config(text="Code verified!", fg="green")
            root.after(1000, lambda: show_reset_password_form(username))
            root.after(5000, clear_message_label)
        else:
            code_info["attempts"] += 1
            if code_info["attempts"] >= MAX_ATTEMPTS:
                del verification_codes[username]
                message_label.config(text="Incorrect code. Maximum attempts exceeded. Please
request a new code.", fg="red")
                root.after(5000, clear_message_label)
                root.after(2000, reset_password_form)
            else:
                message_label.config(text=f"Incorrect code. Attempts left: {MAX_ATTEMPTS -
code_info['attempts']}", fg="red")
                root.after(5000, clear_message_label)

    tk.Button(root, text="Verify", command=verify_code, bg="white", fg="black").pack(pady=10)
    tk.Button(root, text="Back", command=reset_password_form, bg="darkgoldenrod",
fg="black").pack(pady=10)



# Show reset password form
def show_reset_password_form(username):
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="Reset Your Password", font=("Arial", 20, "bold")).pack(pady=20)

    tk.Label(root, text="Enter new password:").pack()
    new_password_entry = tk.Entry(root, show="*")
    new_password_entry.pack()
```

```
    message_label = tk.Label(root, text="", fg="red")
    message_label.pack()



    # Update password in DB
    def update_password():
        new_password = new_password_entry.get().strip()
        if not new_password:
            message_label.config(text="Password cannot be empty", fg="red")
            return

        import hashlib
        hashed_password = hashlib.sha256(new_password.encode()).hexdigest()

        conn = sqlite3.connect(DB_PATH)
        c = conn.cursor()
        c.execute("UPDATE admins SET password = ? WHERE username = ?", (hashed_password,
username))
        conn.commit()
        conn.close()

        message_label.config(text="Password reset successfully!", fg="green")
        def clear_message_label():
            if message_label.winfo_exists():
                message_label.config(text="")
        root.after(5000, main_interface )  # Back to main menu
        root.after(5000, clear_message_label)

    tk.Button(root, text="Reset Password", command=update_password, bg="white",
fg="black").pack(pady=10)



def open_staff_panel():
    for widget in root.winfo_children():
        widget.destroy()
    root.state('zoomed')
    root.attributes("-fullscreen", True)
    tk.Label(root, text="STAFF ACCESS", font=("Arial", 40, "bold"),
fg="darkgoldenrod").pack(pady=30)

    btn_frame = tk.Frame(root)
    btn_frame.pack(pady=40)

    def open_student_list():
        # Show the student list in readonly mode
        for widget in root.winfo_children():
            widget.destroy()
        root.state('zoomed')
        root.attributes("-fullscreen", True)
        tk.Label(root, text="STUDENT LIST", font=("Arial", 25, "bold"),
fg="darkgoldenrod").pack(pady=10)
        search_var = tk.StringVar()
        search_frame = tk.Frame(root)
        search_frame.pack(pady=10)
        tk.Label(search_frame, text="🔎Search (ID or Name):").pack(side="left", padx=5)
        tk.Entry(search_frame, textvariable=search_var, width=30).pack(side="left", padx=5)
```

```
        sort_by = "name"
        def apply_sort(new_sort_by):
            nonlocal sort_by
            sort_by = new_sort_by
            update_treeview()
        sort_button = tk.Menubutton(root, text="Sort", relief="raised", font=("Arial", 12), bg="white",
fg="black")
        sort_menu = tk.Menu(sort_button, tearoff=0)
        sort_menu.add_command(label="Sort by Class", command=lambda:
apply_sort("class_name"))
        sort_menu.add_command(label="Sort by Name", command=lambda: apply_sort("name"))
        sort_menu.add_command(label="Sort by ID", command=lambda: apply_sort("student_id"))
        sort_button["menu"] = sort_menu
        sort_button.pack(pady=5)
        table_frame = tk.Frame(root)
        table_frame.pack(fill="both", expand=True, padx=20, pady=10)
        columns = ("student_id", "name", "age", "class_name")
        tree = ttk.Treeview(table_frame, columns=columns, show="headings")
        for col in columns:
            tree.heading(col, text=col.replace("_", " ").title())
            tree.column(col, anchor="center", width=200, stretch=True)
        scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
        tree.configure(yscrollcommand=scrollbar.set)
        tree.pack(side="left", fill="both", expand=True)
        scrollbar.pack(side="right", fill="y")
        def update_treeview(*args):
            for item in tree.get_children():
                tree.delete(item)
            keyword = search_var.get()
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                if keyword:
                    c.execute(f"SELECT student_id, name, age, class_name FROM students WHERE
name LIKE ? OR student_id LIKE ? ORDER BY {sort_by} COLLATE NOCASE ASC", (f"%
{keyword}%", f"%{keyword}%"))
                else:
                    c.execute(f"SELECT student_id, name, age, class_name FROM students ORDER BY
{sort_by} COLLATE NOCASE ASC")
                students = c.fetchall()
            if not students:
                tree.insert("", "end", values=("No records found", "", "", ""))
            else:
                for s in students:
                    tree.insert("", "end", values=(s[0], s[1], s[2], s[3]))
        search_var.trace_add("write", update_treeview)
        update_treeview()
        def on_double_click(event):
            selected_item = tree.selection()
            if not selected_item:
                return
            student_id = tree.item(selected_item[0])['values'][0]
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("SELECT student_id, name, father_name, mother_name, dob, age, mobile,
email, gender, class_name, address, photo FROM students WHERE student_id = ?", (student_id,))
                profile = c.fetchone()
            if profile:
                open_student_readonly_profile(profile)
            else:
                tk.messagebox.showerror("Error", "Student profile not found.")
```

```
        tree.bind("<Double-1>", on_double_click)
        tk.Button(root, text="Back", font=("Arial", 12), command=open_staff_panel).pack(pady=20)

    def open_staff_list():
        view_staff_readonly()

    tk.Button(btn_frame, text="Student List", font=("Arial", 20), width=20, height=2,
              bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
              highlightthickness=0, command=open_student_list).pack(pady=10)
    tk.Button(btn_frame, text="Staff List", font=("Arial", 20), width=20, height=2,
              bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
              highlightthickness=0, command=open_staff_list).pack(pady=10)
    tk.Button(root, text="Back", font=("Arial", 20), width=20, height=2,
              bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat",
              highlightthickness=0, command=main_interface).pack(pady=30)


#  Read-only staff list
def view_staff_readonly():
    for widget in root.winfo_children():
        widget.destroy()

    root.state('zoomed')
    root.attributes("-fullscreen", True)
    tk.Label(root, text="STAFF LIST", font=("Arial", 25, "bold"), fg="darkgoldenrod").pack(pady=10)
    search_var = tk.StringVar()
    search_frame = tk.Frame(root)
    search_frame.pack(pady=10)
    tk.Label(search_frame, text="🔎Search (ID or Name):").pack(side="left")
    tk.Entry(search_frame, textvariable=search_var, width=30).pack(side="left", padx=5)
    sort_by = "name"
    def apply_sort(new_sort_by):
        nonlocal sort_by
        sort_by = new_sort_by
        update_treeview()
    sort_button = tk.Menubutton(root, text="Sort", relief="raised", font=("Arial", 12), bg="white",
fg="black")
    sort_menu = tk.Menu(sort_button, tearoff=0)
    sort_menu.add_command(label="Sort by Name", command=lambda: apply_sort("name"))
    sort_menu.add_command(label="Sort by Staff ID", command=lambda: apply_sort("staff_id"))
    sort_menu.add_command(label="Sort by Designation", command=lambda:
apply_sort("designation"))
    sort_button["menu"] = sort_menu
    sort_button.pack(pady=5)
    table_frame = tk.Frame(root)
    table_frame.pack(fill="both", expand=True, padx=20, pady=10)
    columns = ("staff_id", "name", "designation", "mobile", "email")
    tree = ttk.Treeview(table_frame, columns=columns, show="headings")
    for col in columns:
        tree.heading(col, text=col)
        tree.column(col, anchor="center", width=200, stretch=True)
    scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
    tree.configure(yscrollcommand=scrollbar.set)
    tree.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")
    def update_treeview(*args):
        for item in tree.get_children():
            tree.delete(item)
        keyword = search_var.get()
        with sqlite3.connect(DB_PATH) as conn:
```

```
            c = conn.cursor()
            if keyword:
                c.execute(f"SELECT staff_id, name, designation, mobile, email FROM staff WHERE
name LIKE ? OR staff_id LIKE ? ORDER BY {sort_by} COLLATE NOCASE ASC",
                          (f"%{keyword}%", f"%{keyword}%"))
            else:
                c.execute(f"SELECT staff_id, name, designation, mobile, email FROM staff ORDER BY
{sort_by} COLLATE NOCASE ASC")
            rows = c.fetchall()
        if not rows:
            tree.insert("", "end", values=("No records found", "", "", "", ""))
        else:
            for r in rows:
                tree.insert("", "end", values=r)
    search_var.trace_add("write", update_treeview)
    update_treeview()
    def on_double_click(event):
        selected = tree.selection()
        if not selected:
            return
        staff_id = tree.item(selected[0])["values"][0]
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM staff WHERE staff_id = ?", (staff_id,))
            profile = c.fetchone()
        if profile:
            open_staff_readonly_profile(profile)
    tree.bind("<Double-1>", on_double_click)
    tk.Button(root, text="Back", command=open_staff_panel, bg="darkgoldenrod",
fg="black").pack(pady=20)

def open_student_readonly_profile(profile):
    window = tk.Toplevel(root)
    window.title("Student Profile (Read-Only)")
    window.geometry("2560x1600")

    left_frame = tk.Frame(window)
    left_frame.grid(row=0, column=0, padx=50, pady=20, sticky="n")
    right_frame = tk.Frame(window)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")

    labels_ordered = [
        ("Student ID", profile[0]),
        ("Name", profile[1]),
        ("Father's Name", profile[2]),
        ("Mother's Name", profile[3]),
        ("Gender", profile[8]),
        ("DOB", profile[4]),
        ("Age", profile[5]),
        ("Class", profile[9]),
        ("Mobile", profile[6]),
        ("Email", profile[7]),
        ("Address", profile[10])
    ]

    for i, (label, value) in enumerate(labels_ordered):
        tk.Label(left_frame, text=label + ":", font=("Arial", 18, "bold")).grid(row=i, column=0,
sticky="w", pady=8)
        tk.Label(left_frame, text=value if value else "", font=("Arial", 16), fg="gray").grid(row=i,
column=1, sticky="w", pady=8)
```

```
    right_frame = tk.Frame(window, width=400, height=900)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")
    right_frame.grid_propagate(False)  # keep frame size fixed


    photo_label = tk.Label(right_frame, bg="lightgray")
    photo_label.pack(fill="both", expand=True)


    photo_path = profile[11] if len(profile) > 11 else None
    if photo_path:
        try:
            img = Image.open(photo_path)
            frame_width = right_frame.winfo_reqwidth()
            frame_height = right_frame.winfo_reqheight()
            img_ratio = img.width / img.height
            frame_ratio = frame_width / frame_height
            if frame_ratio > img_ratio:

                new_height = frame_height
                new_width = int(img_ratio * new_height)
            else:

                new_width = frame_width
                new_height = int(new_width / img_ratio)
            img = img.resize((new_width, new_height))
            photo_img = ImageTk.PhotoImage(img)
            photo_label.config(image=photo_img)
            photo_label.photo_img = photo_img                                    # keep reference
        except:
            photo_label.config(text="Photo not available")

    tk.Button(window, text="Close", width=20, command=window.destroy).grid(row=1, column=0,
columnspan=2, pady=40)


def open_staff_readonly_profile(profile):
    window = tk.Toplevel(root)
    window.title("Staff Profile (Read-Only)")
    window.geometry("2560x1600")

    left_frame = tk.Frame(window)
    left_frame.grid(row=0, column=0, padx=50, pady=20, sticky="n")
    right_frame = tk.Frame(window)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")

    labels_ordered = [
        ("Staff ID", profile[0]),
        ("Name", profile[1]),
        ("Gender", profile[6]),
        ("DOB", profile[2]),
        ("Age", profile[3]),
        ("Mobile", profile[4]),
        ("Email", profile[5]),
        ("Designation", profile[7]),
        ("Subject", profile[8]),
        ("Salary", profile[9]),
        ("Date of Joining", profile[10]),
```

```
        ("Address", profile[11])
    ]

    for i, (label, value) in enumerate(labels_ordered):
        tk.Label(left_frame, text=label + ":", font=("Arial", 18, "bold")).grid(row=i, column=0,
sticky="w", pady=8)
        tk.Label(left_frame, text=value if value else "", font=("Arial", 16), fg="gray").grid(row=i,
column=1, sticky="w", pady=8)


    right_frame = tk.Frame(window, width=400, height=900)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")
    right_frame.grid_propagate(False)


    photo_label = tk.Label(right_frame, bg="lightgray")
    photo_label.pack(fill="both", expand=True)


    photo_path = profile[12] if len(profile) > 12 else None
    if photo_path:
        try:
            img = Image.open(photo_path)
            frame_width = right_frame.winfo_reqwidth()
            frame_height = right_frame.winfo_reqheight()
            img_ratio = img.width / img.height
            frame_ratio = frame_width / frame_height
            if frame_ratio > img_ratio:

                new_height = frame_height
                new_width = int(img_ratio * new_height)
            else:

                new_width = frame_width
                new_height = int(new_width / img_ratio)
            img = img.resize((new_width, new_height))
            photo_img = ImageTk.PhotoImage(img)
            photo_label.config(image=photo_img)
            photo_label.photo_img = photo_img                                    # keep reference
        except:
            photo_label.config(text="Photo not available")

    tk.Button(window, text="Close", width=20, command=window.destroy).grid(row=1, column=0,
columnspan=2, pady=40)



# Entry Interface
def entry_interface():
    for widget in root.winfo_children():
        widget.destroy()

    title_label = tk.Label(
    root,
    text="EDUMATRIX",
    font=("Times New Roman", 233, "bold"),
    fg="darkgoldenrod"
    )
    title_label.place(relx=0.5, rely=0.4, anchor="center")
```

```
title_label = tk.Label(
root,
text="PLUG IN. LEVEL UP. BREAK FREE.",
font=("Times New Roman", 50, "bold"),
fg="darkgoldenrod"
)
title_label.place(relx=0.5, rely=0.55, anchor="center")

#digital clock
def update_clock():
    now = datetime.now()
    current_time = now.strftime("%A, %d %B %Y | %H:%M:%S")
    clock_label.config(text=current_time)
    clock_label.after(1000, update_clock)

clock_label = tk.Label(
    root,
    font=("Courier", 30, "bold"),
    bg="#323232",
    fg="#B0882F",
    padx=10,
    pady=5
)
clock_label.place(relx=0.5, rely=0.02, anchor="n")
update_clock()

btn = tk.Button(
    root,
    text="ENTER EDUMATRIX",
    font=("Times New Roman", 35, "bold"),
    bg="darkgoldenrod",
    fg="darkgoldenrod",
    command=main_interface
)
btn.pack(pady=20)
btn.place(relx=0.5, rely=0.70, anchor="center")


def show_help():
    messagebox.showinfo("HELP", "Developed by: Vittal and Vishnu\nVersion: v.1")

help_btn = tk.Button(
    root,
    text="HELP",
    font=("Times New Roman", 35, "bold"),
    bg="darkgoldenrod",
    fg="darkgoldenrod",
    command=show_about
)
help_btn.place(relx=0.5, rely=0.77, anchor="center")


about_btn = tk.Button(
    root,
    text="ABOUT",
    font=("Times New Roman", 35, "bold"),
    bg="darkgoldenrod",
    fg="darkgoldenrod",
    command=show_help
```

```
    )
    about_btn.place(relx=0.5, rely=0.84, anchor="center")


    exit_btn = tk.Button(
        root,
        text="EXIT",
        font=("Times New Roman", 35, "bold"),
        bg="darkgoldenrod",
        fg="darkgoldenrod",
        command=root.quit
    )
    exit_btn.place(relx=0.5, rely=0.91, anchor="center")

def show_about():
    help_text = (
        "EDUMATRIX School Management System\n\n"
        "This project allows management of students and staff with features such as:\n"
        "- Adding, viewing, and updating student and staff records\n"
        "- Automatic age calculation from DOB\n"
        "- Login and logout history tracking\n"
        "- Password reset via email verification\n"
        "- Postal code autofill for student and staff addresses\n"
        "- Read-only profiles for staff panel access\n\n"
        "Use the ADMIN or STAFF buttons to access respective panels. "
        "The system is designed to simplify school management while keeping data secure and
organized."
    )
    messagebox.showinfo("About", help_text)

def main_interface():
    for widget in root.winfo_children():
        widget.destroy()

    tk.Label(root, text="WELCOME TO EDUMATRIX", font=("Times New Roman", 100, "bold"),
fg="darkgoldenrod").pack(side="top", pady=20)
    frame = tk.Frame(root)
    frame.place(relx=0.5, rely=0.5, anchor="center")
    tk.Label(frame, text="SELECT USER TYPE", font=("Arial",
40,"bold"),fg="darkgoldenrod").pack(pady=20)
    tk.Button(frame, text="ADMIN", font=("Arial", 20), bg='darkgoldenrod', fg='darkgoldenrod',
width=15, height=2, command=open_admin_panel).pack(pady=15)
    tk.Button(frame, text="STAFF", font=("Arial", 20), bg='darkgoldenrod', fg='darkgoldenrod',
width=15, height=2, command=open_staff_panel).pack(pady=15)
    tk.Button(frame, text="BACK", font=("Arial",20), bg = 'darkgoldenrod',fg ='darkgoldenrod',
width = 15, height=2, command=entry_interface).pack(pady=15)



root = tk.Tk()
root.title("EDUMATRIX School Management System")


root.state('zoomed')
root.attributes("-fullscreen", True)


screen_width = root.winfo_screenwidth()
screen_height = root.winfo_screenheight()
root.geometry(f"{screen_width}x{screen_height}")
```

```
def load_indian_postal_data(csv_file_path):
    cache_file = "postal_path.txt"
    if not os.path.exists(csv_file_path) and os.path.exists(cache_file):
        with open(cache_file, "r") as f:
            cached_path = f.read().strip()
            if os.path.exists(cached_path):
                csv_file_path = cached_path

    if not os.path.exists(csv_file_path):
        print(f"CSV file {csv_file_path} not found. Please select the file.")
        selected_file_path = filedialog.askopenfilename(title="Select Indian PIN Code CSV",
filetypes=[("CSV files", "*.csv")])
        if not selected_file_path:
            print("No file selected. Skipping postal code loading.")
            return
        csv_file_path = selected_file_path
        with open(cache_file, "w") as f:
            f.write(csv_file_path)

    # Open a local DB connection
    with sqlite3.connect(DB_PATH) as conn:
        cursor = conn.cursor()
        with open(csv_file_path, newline='', encoding='utf-8') as csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                cursor.execute(
                    'INSERT OR IGNORE INTO postal_codes (city, pin, state) VALUES (?, ?, ?)',
                    (row['District'].strip(), row['Pincode'].strip(), row['StateName'].strip())
                )
        conn.commit()

# Load Indian postal data once at startup
load_indian_postal_data("/Users/vittalvishnu/Documents/indian_pincodes.csv")

# Student Management Section
def open_student_management():
    recalculate_student_ages()
    for widget in root.winfo_children():
        widget.destroy()
    tk.Label(root, text="STUDENT MANAGEMENT", font=("Arial", 60,
"bold"),fg="darkgoldenrod").pack(pady=30)
    tk.Button(root, text="Add New Student", font=("Arial", 20), width=20, height=2,
bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat", highlightthickness=0,
command=open_add_student_form).pack(pady=10)
    tk.Button(root, text="View Students", font=("Arial", 20), width=20, height=2,
bg="darkgoldenrod", fg="darkgoldenrod", bd=0, relief="flat", highlightthickness=0,
command=view_students).pack(pady=10)

    tk.Button(root, text="Back", font=("Arial", 20), width=20, height=2, bg="darkgoldenrod",
fg="darkgoldenrod", bd=0, relief="flat", highlightthickness=0,
command=open_dashboard).pack(pady=30)



def open_add_student_form():
    for widget in root.winfo_children():
        widget.destroy()
```

```
    tk.Label(root, text="ADD NEW STUDENT", font=("Arial", 30, "bold"),
fg="darkgoldenrod").pack(pady=20)

    # Generate a random 6-digit Student ID
    student_id = generate_unique_roll_no()
    tk.Label(root, text=f"Student ID: {student_id}", font=("Arial", 14), anchor="e").pack(side="top",
padx=20, pady=5, fill="x")

    tk.Label(root, text="Student Name:").pack()
    name_frame = tk.Frame(root)
    name_frame.pack()
    first_name_entry = tk.Entry(name_frame, width=20)
    first_name_entry.insert(0, "First Name")
    first_name_entry.pack(side="left", padx=5)
    last_name_entry = tk.Entry(name_frame, width=20)
    last_name_entry.insert(0, "Last Name")
    last_name_entry.pack(side="left", padx=5)


    photo_path = tk.StringVar()

    def upload_photo():
        file_path = filedialog.askopenfilename(
            title="Select Student Photo",
            filetypes=[("Image files", "*.jpg *.jpeg *.png *.gif")]
        )
        if file_path:
            photo_path.set(file_path)
            tk.Label(root, text=f"Selected: {os.path.basename(file_path)}").pack()

    tk.Button(root, text="Upload Photo", command=upload_photo, bg="darkgoldenrod",
fg="black").pack(pady=5)


    def clear_name_placeholder(e):
        if e.widget.get() in ["First Name", "Last Name"]:
            e.widget.delete(0, tk.END)

    def restore_name_placeholder(e):
        if e.widget.get().strip() == "":
            if e.widget == first_name_entry:
                e.widget.insert(0, "First Name")
            elif e.widget == last_name_entry:
                e.widget.insert(0, "Last Name")

    first_name_entry.bind("<FocusIn>", clear_name_placeholder)
    last_name_entry.bind("<FocusIn>", clear_name_placeholder)
    first_name_entry.bind("<FocusOut>", restore_name_placeholder)
    last_name_entry.bind("<FocusOut>", restore_name_placeholder)

    tk.Label(root, text="Father's Name:").pack()
    father_name_entry = tk.Entry(root)
    father_name_entry.pack()

    tk.Label(root, text="Mother's Name:").pack()
    mother_name_entry = tk.Entry(root)
    mother_name_entry.pack()

    tk.Label(root, text="Date of Birth:").pack()
    dob_frame = tk.Frame(root)
```

```
    dob_frame.pack()
    day_var = tk.StringVar(value="Day")
    tk.OptionMenu(dob_frame, day_var, *[str(i) for i in range(1, 32)]).pack(side="left", padx=2)
    month_var = tk.StringVar(value="Month")
    tk.OptionMenu(dob_frame, month_var, *[str(i) for i in range(1, 13)]).pack(side="left", padx=2)
    year_var = tk.StringVar(value="Year")
    tk.OptionMenu(dob_frame, year_var, *[str(i) for i in range(1980, 2031)]).pack(side="left", padx=2)

    tk.Label(root, text="Mobile No:").pack()
    mobile_entry = tk.Entry(root)
    mobile_entry.pack()

    tk.Label(root, text="Email ID:").pack()
    email_entry = tk.Entry(root)
    email_entry.pack()

    tk.Label(root, text="Gender:").pack()
    gender_var = tk.StringVar()
    gender_frame = tk.Frame(root)
    gender_frame.pack()
    tk.Radiobutton(gender_frame, text="Male", variable=gender_var,
value="Male").pack(side="left")
    tk.Radiobutton(gender_frame, text="Female", variable=gender_var,
value="Female").pack(side="left")

    tk.Label(root, text="Class:").pack()
    class_var = tk.StringVar(value="Select Class")
    tk.OptionMenu(root, class_var, *[str(i) for i in range(1, 13)]).pack(pady=5)

    tk.Label(root, text="Address:").pack()
    street_entry = tk.Entry(root, width=30)
    street_entry.insert(0, "Street Address")
    street_entry.pack(pady=2)
    city_entry = tk.Entry(root, width=30)
    city_entry.insert(0, "City")
    city_entry.pack(pady=2)
    pin_entry = tk.Entry(root, width=30)
    pin_entry.insert(0, "PIN Code")
    pin_entry.pack(pady=2)
    state_entry = tk.Entry(root, width=30)
    state_entry.insert(0, "State")
    state_entry.pack(pady=2)

    country_label = tk.Label(root, text="Country: India", anchor="w", width=30, fg="grey")
    country_label.pack(pady=2)


    def clear_placeholder(e):
        if e.widget.get() in ["Street Address", "City", "PIN Code", "State"]:
            e.widget.delete(0, tk.END)
    def restore_placeholder(e):
        if e.widget.get().strip() == "":
            if e.widget == street_entry:
                e.widget.insert(0, "Street Address")
            elif e.widget == city_entry:
                e.widget.insert(0, "City")
            elif e.widget == pin_entry:
                e.widget.insert(0, "PIN Code")
            elif e.widget == state_entry:
                e.widget.insert(0, "State")
```

```
    for entry in [street_entry, city_entry, pin_entry, state_entry]:
        entry.bind("<FocusIn>", clear_placeholder)
        entry.bind("<FocusOut>", restore_placeholder)

    # Pincode autofill for city entry
    def autofill_pin_state(event):
        city = city_entry.get().strip().lower()
        if city in postal_data:
            pin_entry.delete(0, tk.END)
            pin_entry.insert(0, postal_data[city][0])
            state_entry.delete(0, tk.END)
            state_entry.insert(0, postal_data[city][1].title())

    city_entry.bind("<KeyRelease>", autofill_pin_state)

    error_label = tk.Label(root, text="", fg="red")
    error_label.pack(pady=5)

    def save_student():
        error_label.config(text="", fg="red")
        first_name = first_name_entry.get().strip()
        last_name = last_name_entry.get().strip()

        if not first_name or first_name.lower() == "first name":
            error_label.config(text="First Name is required!", fg="red")
            return
        if not re.fullmatch(r"[A-Za-z ]+", first_name):
            error_label.config(text="First Name must contain only letters and spaces.", fg="red")
            return

        valid_last = last_name and last_name.lower() != "last name"
        if valid_last:
            if not re.fullmatch(r"[A-Za-z ]+", last_name):
                error_label.config(text="Last Name must contain only letters and spaces.", fg="red")
                return

        student_name = first_name
        if valid_last:
            student_name += " " + last_name

        father_name = father_name_entry.get().strip()
        mother_name = mother_name_entry.get().strip()

        try:
            dob_date = datetime(int(year_var.get()), int(month_var.get()), int(day_var.get()))
            dob = dob_date.strftime("%Y-%m-%d")

            age = None
        except ValueError:
            error_label.config(text="Invalid Date of Birth!", fg="red")
            return

        mobile = mobile_entry.get().strip()
        email = email_entry.get().strip()
        gender = gender_var.get()
        class_name = class_var.get()
        pin = pin_entry.get().strip()
        address = f"{street_entry.get().strip().title()}, {city_entry.get().strip().title()}, {pin},
{state_entry.get().strip().title()}, India"
```

```
        email_regex = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
        if not re.match(email_regex, email):
            error_label.config(text="Invalid email format!", fg="red")
            return
        mobile_regex = r'^[6-9]\d{9}$'
        if not re.match(mobile_regex, mobile):
            error_label.config(text="Invalid mobile number! Must be 10 digits and start with 6-9.",
fg="red")
            return
        if not pin.isdigit() or len(pin) != 6:
            error_label.config(text="PIN code must be a 6-digit number.", fg="red")
            return
        if not class_name.isdigit() or not (1 <= int(class_name) <= 12):
            error_label.config(text="Please select a valid class (1–12).", fg="red")
            return


        photo_val = photo_path.get().strip()
        if not photo_val:
            photo_val = None

        try:
            with sqlite3.connect(DB_PATH) as conn:
                cursor = conn.cursor()

                cursor.execute(
                    "INSERT INTO students (student_id, name, father_name, mother_name, dob, age,
mobile, email, gender, class_name, address, photo) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)",
                    (
                        student_id, student_name, father_name, mother_name, dob, age,
                        mobile, email, gender, class_name, address, photo_val
                    )
                )
                conn.commit()
            error_label.config(text="Student added successfully!", fg="green")
            root.after(1500, open_student_management)
        except sqlite3.IntegrityError as e:
            error_label.config(text=f"Student ID already exists: {e}", fg="red")
        except sqlite3.Error as e:
            error_label.config(text=f"Database error: {e}", fg="red")

    tk.Button(root, text="Save Student", command=save_student, bg="darkgoldenrod",
fg="black").pack(pady=10)
    tk.Button(root, text="Back", command=open_student_management, bg="darkgoldenrod",
fg="black").pack(pady=5)


def open_profile_window(profile, refresh_callback=None):
    window = tk.Toplevel(root)
    window.title("Student Profile")
    window.geometry("2560x1600")


    student_id = profile[0]

    if len(profile) < 12:
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            c.execute(
```

```
            "SELECT student_id, name, father_name, mother_name, dob, age, mobile, email,
gender, class_name, address, photo FROM students WHERE student_id=?",
            (student_id,))
        row = c.fetchone()
        if row:
            profile = row
        else:
            tk.messagebox.showerror("Error", "Student profile not found.")
            window.destroy()
            return


    left_frame = tk.Frame(window)
    left_frame.grid(row=0, column=0, padx=50, pady=20, sticky="n")
    right_frame = tk.Frame(window)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")

    labels_ordered = [
        ("Student ID", profile[0]),
        ("Name", profile[1]),
        ("Father's Name", profile[2]),
        ("Mother's Name", profile[3]),
        ("Gender", profile[8]),
        ("DOB", profile[4]),
        ("Age", profile[5]),
        ("Class", profile[9]),
        ("Mobile", profile[6]),
        ("Email", profile[7]),
        ("Address", profile[10])
    ]
    editable_fields = {"Name", "Mobile", "Email", "Address"}
    entry_widgets = {}

    for i, (label, value) in enumerate(labels_ordered):
        tk.Label(left_frame, text=label + ":", font=("Arial", 18, "bold")).grid(row=i, column=0,
sticky="w", pady=8)
        if label in editable_fields:
            e = tk.Entry(left_frame, width=40, font=("Arial", 16))
            e.insert(0, value if value else "")
            e.grid(row=i, column=1, pady=8)
            entry_widgets[label] = e
        else:
            tk.Label(left_frame, text=value if value else "", font=("Arial", 16), fg="gray").grid(row=i,
column=1, sticky="w", pady=8)
            entry_widgets[label] = None


    right_frame = tk.Frame(window, width=400, height=900)
    right_frame.grid(row=0, column=1, padx=50, pady=20, sticky="n")
    right_frame.grid_propagate(False)
    photo_label = tk.Label(right_frame, bg="lightgray")
    photo_label.pack(fill="both", expand=True)

    photo_path = profile[11] if len(profile) > 11 else None
    if photo_path and os.path.exists(photo_path):
        try:
            img = Image.open(photo_path)
            frame_width = right_frame.winfo_reqwidth()
            frame_height = right_frame.winfo_reqheight()
            img_ratio = img.width / img.height
```

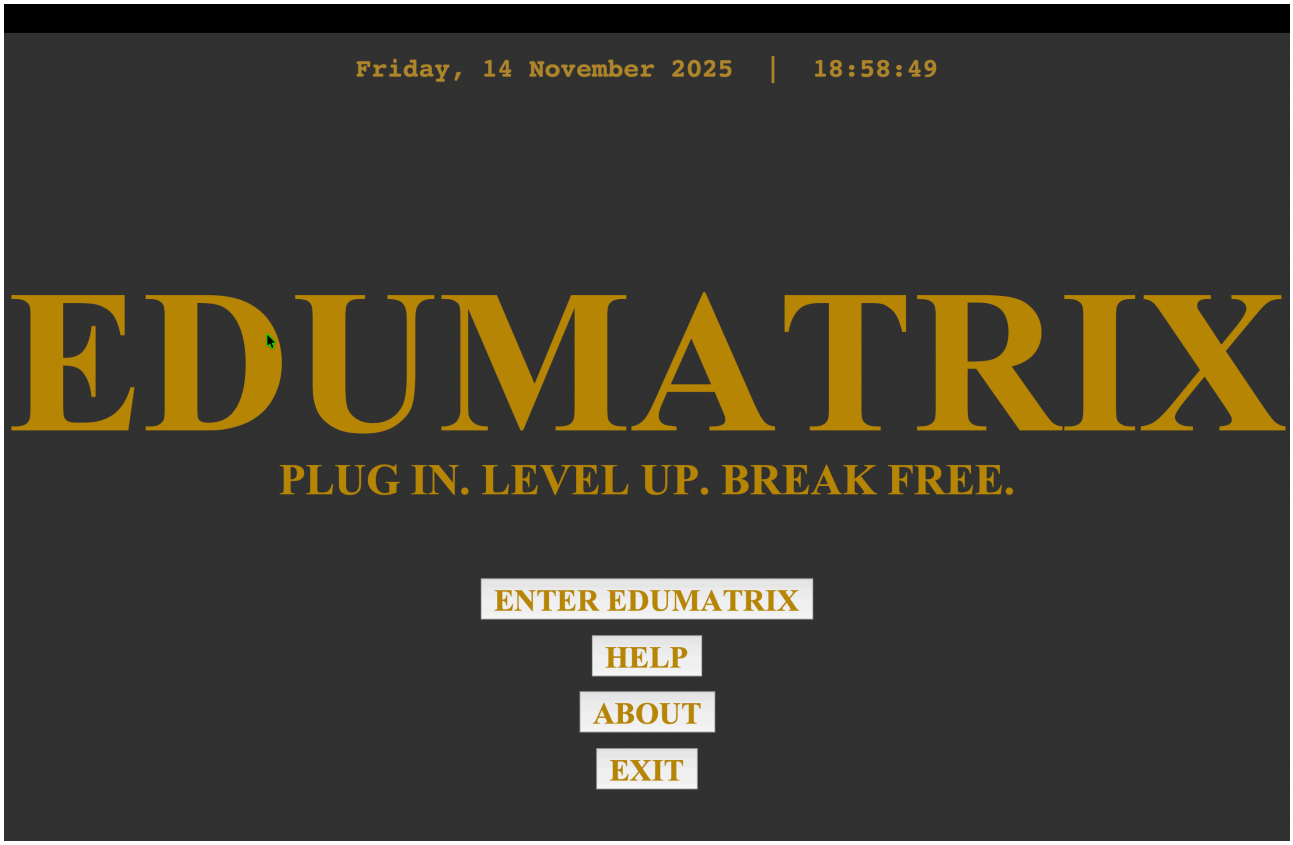
```
            frame_ratio = frame_width / frame_height
            if frame_ratio > img_ratio:
                new_height = frame_height
                new_width = int(img_ratio * new_height)
            else:
                new_width = frame_width
                new_height = int(new_width / img_ratio)
            img = img.resize((new_width, new_height))
            photo_img = ImageTk.PhotoImage(img)
            photo_label.config(image=photo_img)
            photo_label.photo_img = photo_img
        except Exception:
            photo_label.config(text="Photo not available")
    else:
        photo_label.config(text="Photo not available")

    button_frame = tk.Frame(window)
    button_frame.grid(row=1, column=0, columnspan=2, pady=40)

    def save_changes():
        try:
            name = entry_widgets["Name"].get().strip()
            email = entry_widgets["Email"].get().strip()
            mobile = entry_widgets["Mobile"].get().strip()
            address = entry_widgets["Address"].get().strip().title()

            if not name or not re.fullmatch(r"[A-Za-z ]+", name):
                tk.messagebox.showerror("Error", "Name must contain only letters and spaces.")
                return
            email_regex = r'^[a-zA-Z0-9_.+-]+@[a-zA-Z0-9-]+\.[a-zA-Z0-9-.]+$'
            if not re.match(email_regex, email):
                tk.messagebox.showerror("Error", "Invalid email format!")
                return
            mobile_regex = r'^[6-9]\d{9}$'
            if not re.match(mobile_regex, mobile):
                tk.messagebox.showerror("Error", "Invalid mobile number!")
                return
            if not address:
                tk.messagebox.showerror("Error", "Address cannot be empty.")
                return

            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("SELECT age FROM students WHERE student_id=?", (student_id,))
                age = c.fetchone()[0]
                c.execute("""
                    UPDATE students SET name=?, age=?, email=?, mobile=?, address=? WHERE
student_id=?
                """, (name, age, email, mobile, address, student_id))
                conn.commit()
            tk.messagebox.showinfo("Success", "Student profile updated successfully.")
            window.destroy()
            if refresh_callback:
                refresh_callback()
        except Exception as e:
            tk.messagebox.showerror("Error", f"Failed to save changes: {e}")

    def delete_profile():
        confirm = tk.messagebox.askyesno("Confirm Deletion", "Are you sure you want to delete this
student?")
```

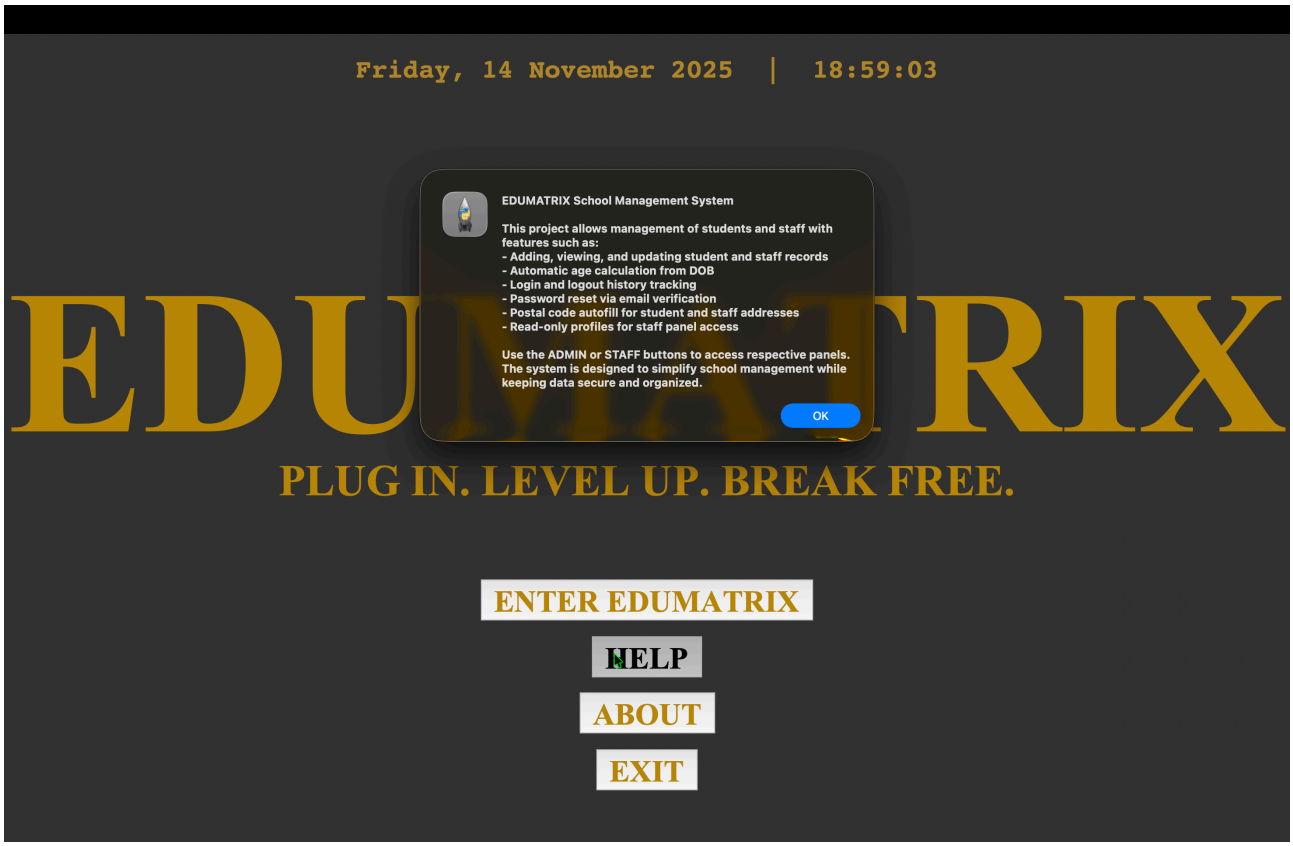
```
        if confirm:
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("DELETE FROM students WHERE student_id=?", (student_id,))
                conn.commit()
            tk.messagebox.showinfo("Deleted", "Student deleted successfully.")
            window.destroy()
            if refresh_callback:
                refresh_callback()

    def promote_student():
        try:
            with sqlite3.connect(DB_PATH) as conn:
                c = conn.cursor()
                c.execute("SELECT class_name FROM students WHERE student_id=?", (student_id,))
                result = c.fetchone()
                if not result:
                    tk.messagebox.showerror("Error", "Student not found.")
                    return
                cls = int(result[0])
                if cls < 12:
                    cls += 1
                    c.execute("UPDATE students SET class_name=? WHERE student_id=?", (str(cls),
student_id))
                    conn.commit()
                    tk.messagebox.showinfo("Promoted", f"{entry_widgets['Name'].get()} promoted to
class {cls}!")
                else:
                    c.execute("DELETE FROM students WHERE student_id=?", (student_id,))
                    conn.commit()
                    tk.messagebox.showinfo("Graduated", f"{entry_widgets['Name'].get()} has
graduated!")
            if refresh_callback:
                refresh_callback()
            window.destroy()
        except Exception as e:
            tk.messagebox.showerror("Error", f"Promotion failed: {e}")

    tk.Button(button_frame, text="Save", width=15, command=save_changes).pack(side="left",
padx=20)
    tk.Button(button_frame, text="Delete Profile", width=15,
command=delete_profile).pack(side="left", padx=20)
    tk.Button(button_frame, text="Promote Student", width=15,
command=promote_student).pack(side="left", padx=20)

def view_students():
    for widget in root.winfo_children():
        widget.destroy()

    root.state('zoomed')
    root.attributes("-fullscreen", True)

    tk.Label(root, text="STUDENT LIST", font=("Arial", 25, "bold"),
fg="darkgoldenrod").pack(pady=10)

    # Search Bar
    search_var = tk.StringVar()
    search_frame = tk.Frame(root)
    search_frame.pack(pady=10)
```

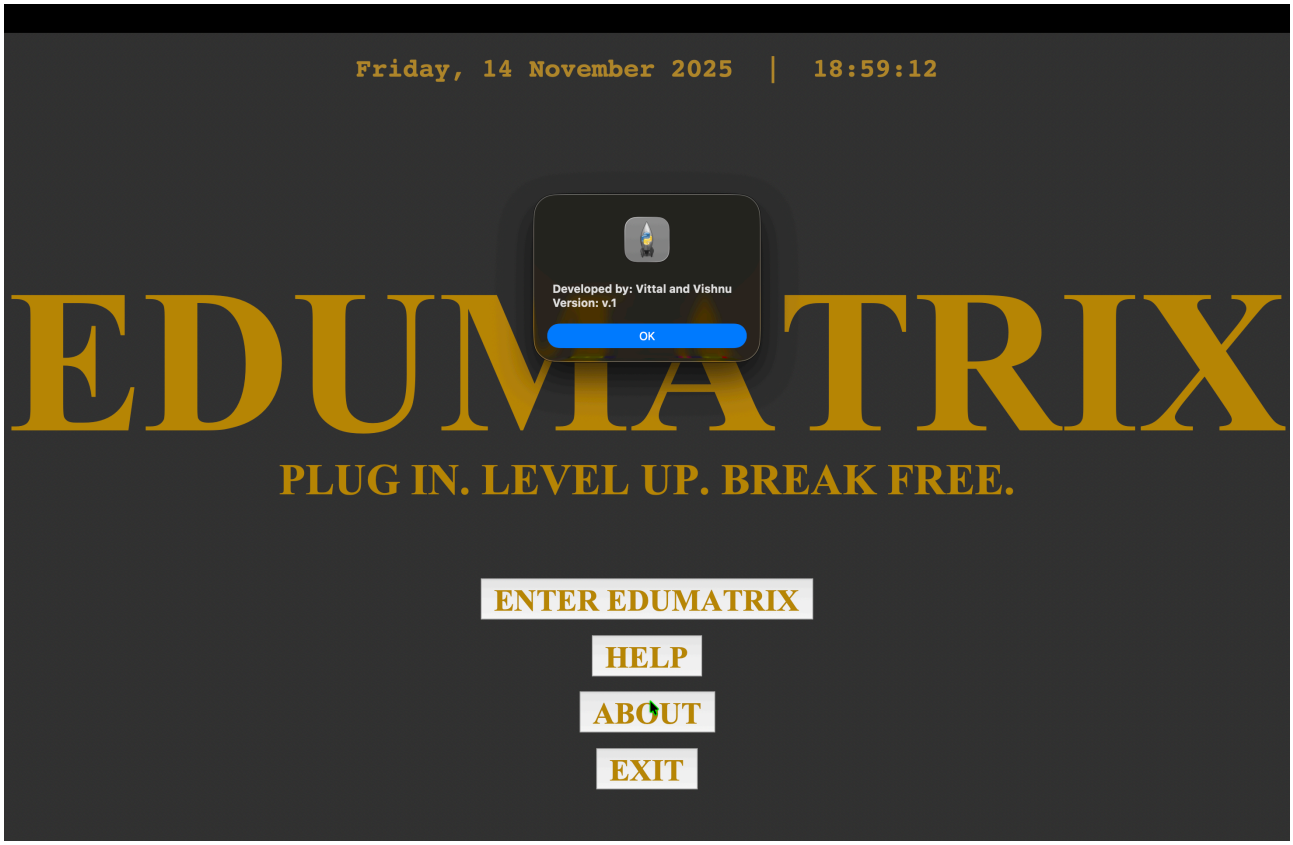
```
    tk.Label(search_frame, text="🔎Search (ID or Name):").pack(side="left", padx=5)
    search_entry = tk.Entry(search_frame, textvariable=search_var, width=30)
    search_entry.pack(side="left", padx=5)

    sort_by = "name"  # default sort column

    def apply_sort(new_sort_by):
        nonlocal sort_by
        sort_by = new_sort_by
        update_treeview()

    # Sorting menu
    sort_button = tk.Menubutton(root, text="Sort", relief="raised", font=("Arial", 12), bg="white",
fg="black")
    sort_menu = tk.Menu(sort_button, tearoff=0)
    sort_menu.add_command(label="Sort by Class", command=lambda: apply_sort("class_name"))
    sort_menu.add_command(label="Sort by Name", command=lambda: apply_sort("name"))
    sort_menu.add_command(label="Sort by ID", command=lambda: apply_sort("student_id"))
    sort_button["menu"] = sort_menu
    sort_button.pack(pady=5)


    table_frame = tk.Frame(root)
    table_frame.pack(fill="both", expand=True, padx=20, pady=10)

    columns = ("student_id", "name", "age", "class_name")
    tree = ttk.Treeview(table_frame, columns=columns, show="headings")

    for col in columns:
        tree.heading(col, text=col.replace("_", " ").title())
        tree.column(col, anchor="center", width=200, stretch=True)

    # Scrollbar
    scrollbar = ttk.Scrollbar(table_frame, orient="vertical", command=tree.yview)
    tree.configure(yscrollcommand=scrollbar.set)
    tree.pack(side="left", fill="both", expand=True)
    scrollbar.pack(side="right", fill="y")


    def update_treeview(*args):
        for item in tree.get_children():
            tree.delete(item)

        keyword = search_var.get()
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            if keyword:
                c.execute(f"""
                    SELECT student_id, name,
                        CAST((julianday('now') - julianday(dob))/365.25 AS INTEGER) AS age,
                        class_name
                    FROM students
                    WHERE name LIKE ? OR CAST(student_id AS TEXT) LIKE ?
                    ORDER BY {sort_by} COLLATE NOCASE ASC
                """, (f'%{keyword}%', f'%{keyword}%'))
            else:
                c.execute(f"SELECT student_id, name, age, class_name FROM students ORDER BY
{sort_by} COLLATE NOCASE ASC")
            students = c.fetchall()
```

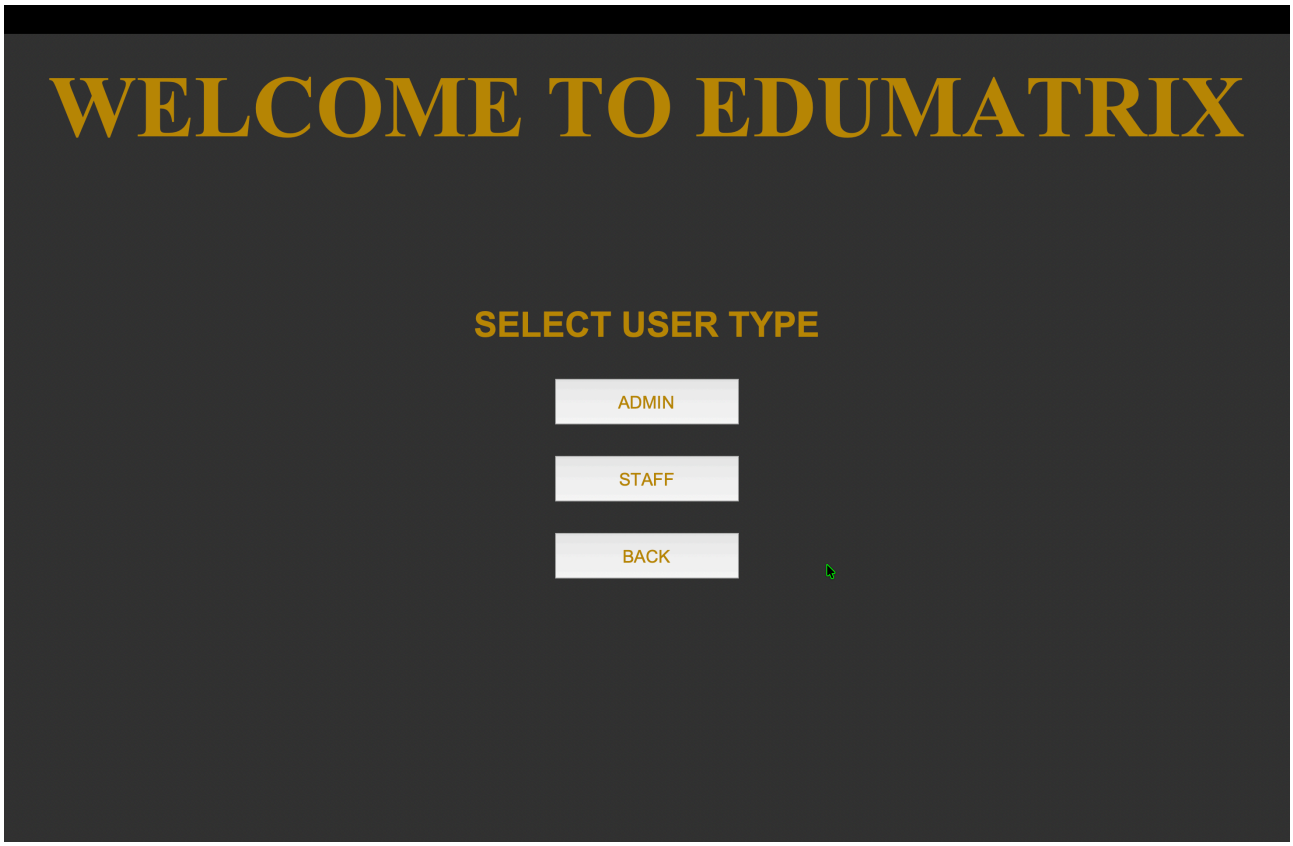
```
        if not students:
            tree.insert("", "end", values=("No records found", "", "", ""))
        else:
            for student in students:
                tree.insert("", "end", values=(
                    student[0],  # student_id
                    student[1],  # name
                    student[2],  # age
                    student[3]   # class_name
                ))

    search_var.trace_add("write", update_treeview)
    update_treeview()


    def on_double_click_tree(event):
        selected_item = tree.focus()
        if not selected_item:
            return
        student_data = tree.item(selected_item, "values")
        if not student_data or student_data[0] == "No records found":
            return
        student_id = student_data[0]

        # Fetch the latest data from DB
        with sqlite3.connect(DB_PATH) as conn:
            c = conn.cursor()
            c.execute("SELECT * FROM students WHERE student_id = ?", (student_id,))
            profile = c.fetchone()

        if profile:
            open_profile_window(profile, refresh_callback=update_treeview)
        else:
            tk.messagebox.showerror("Error", "Student profile not found.")

    tree.bind("<Double-1>", on_double_click_tree)

    # Back button
    tk.Button(root, text="Back", font=("Arial", 12),
              command=open_student_management).pack(pady=20)

    update_treeview()

root.bind('<Escape>', lambda e: root.attributes("-fullscreen", False))
entry_interface()
root.mainloop()
```

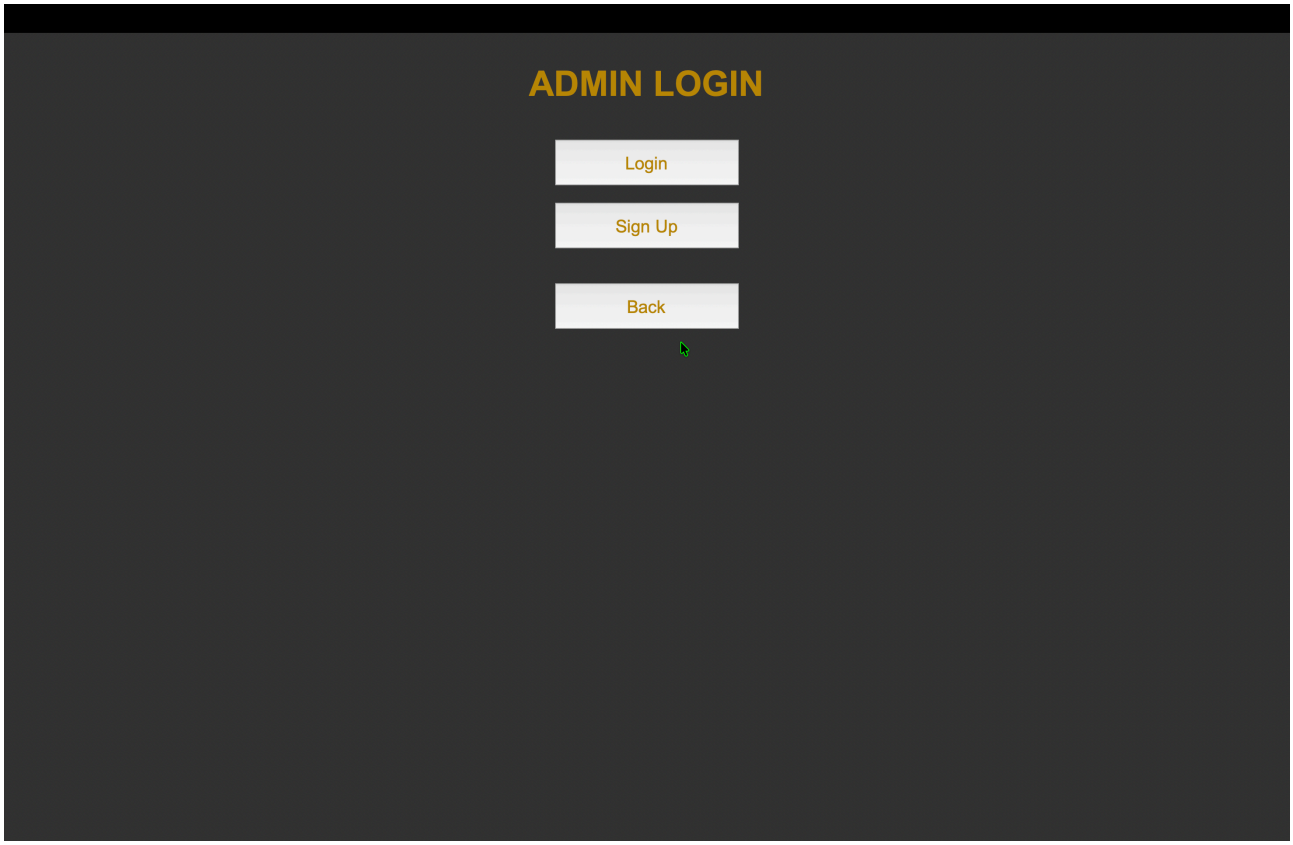
SCREENSHOTS

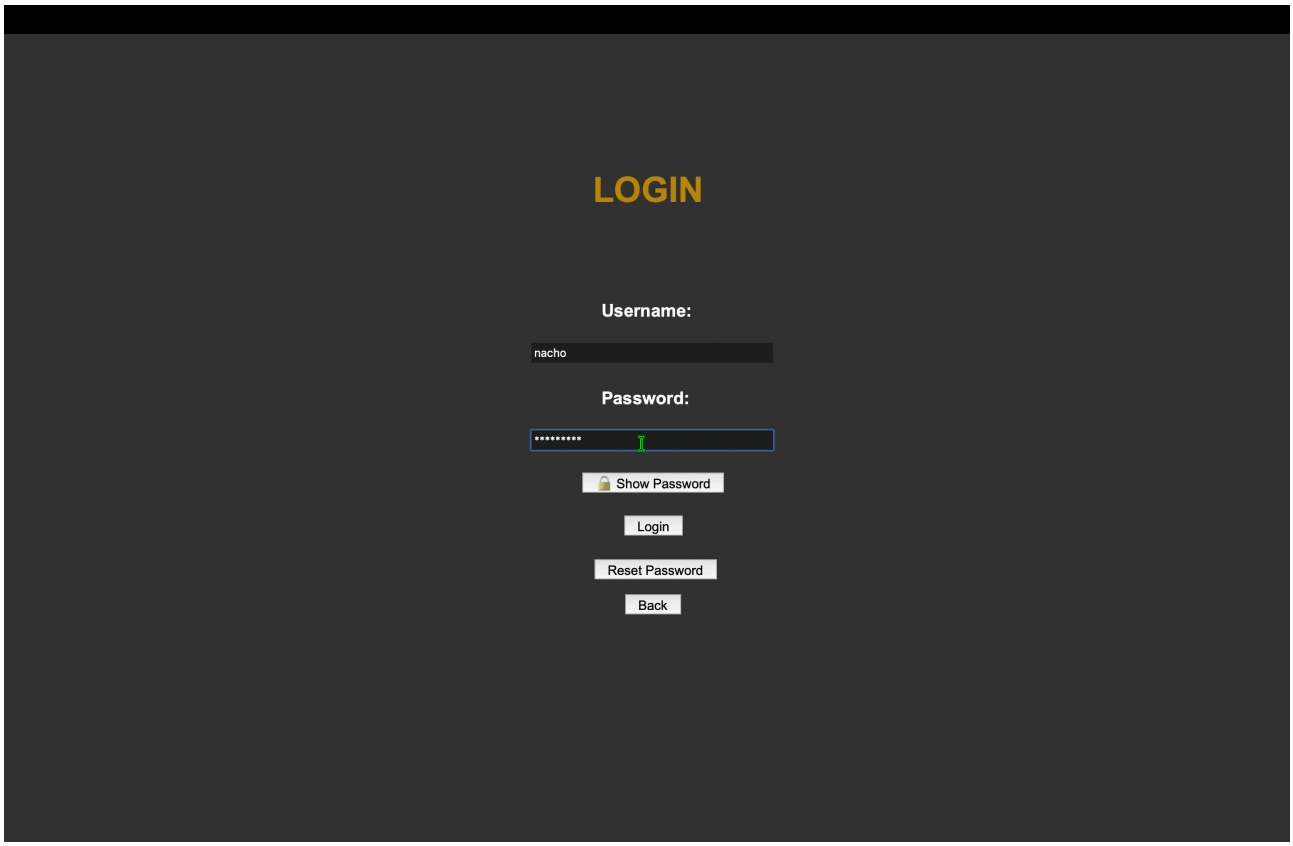


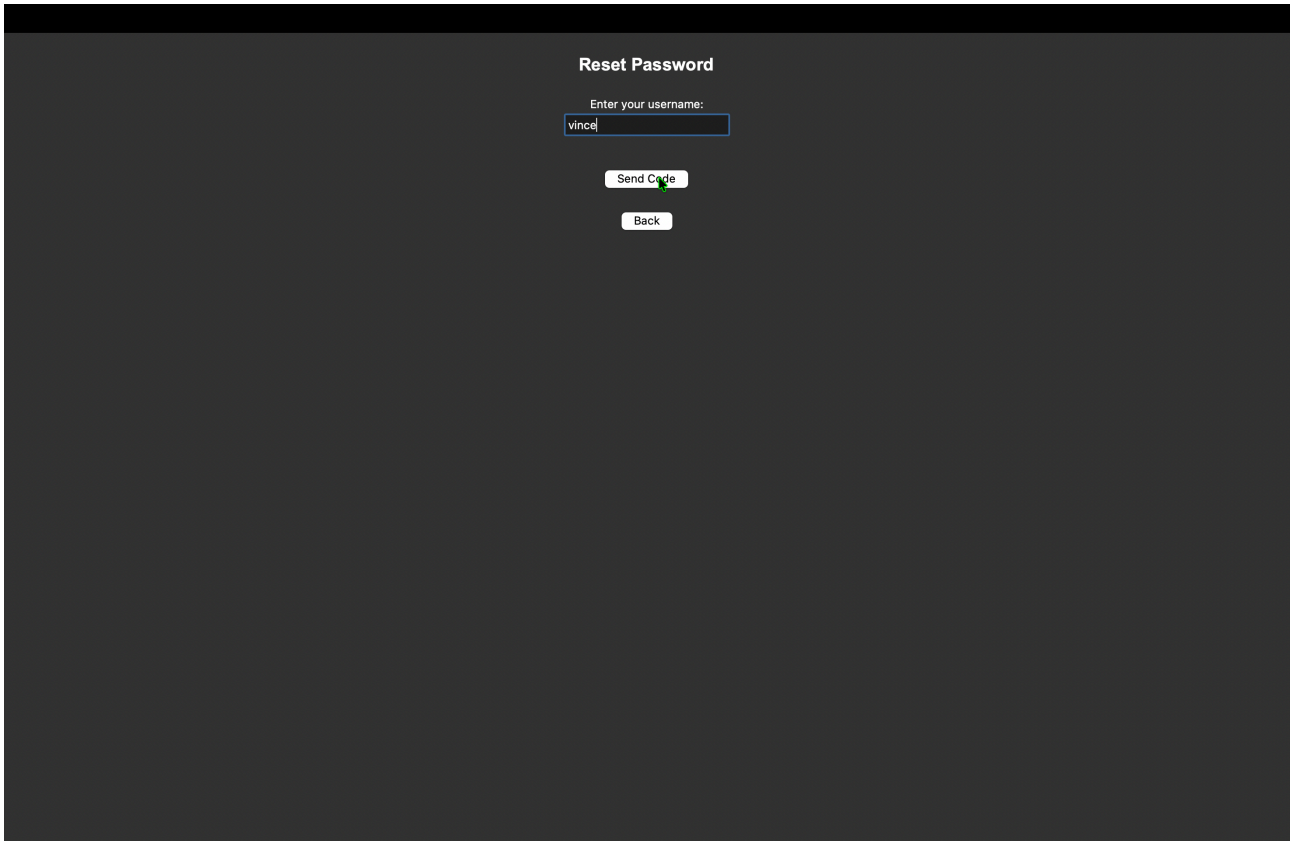


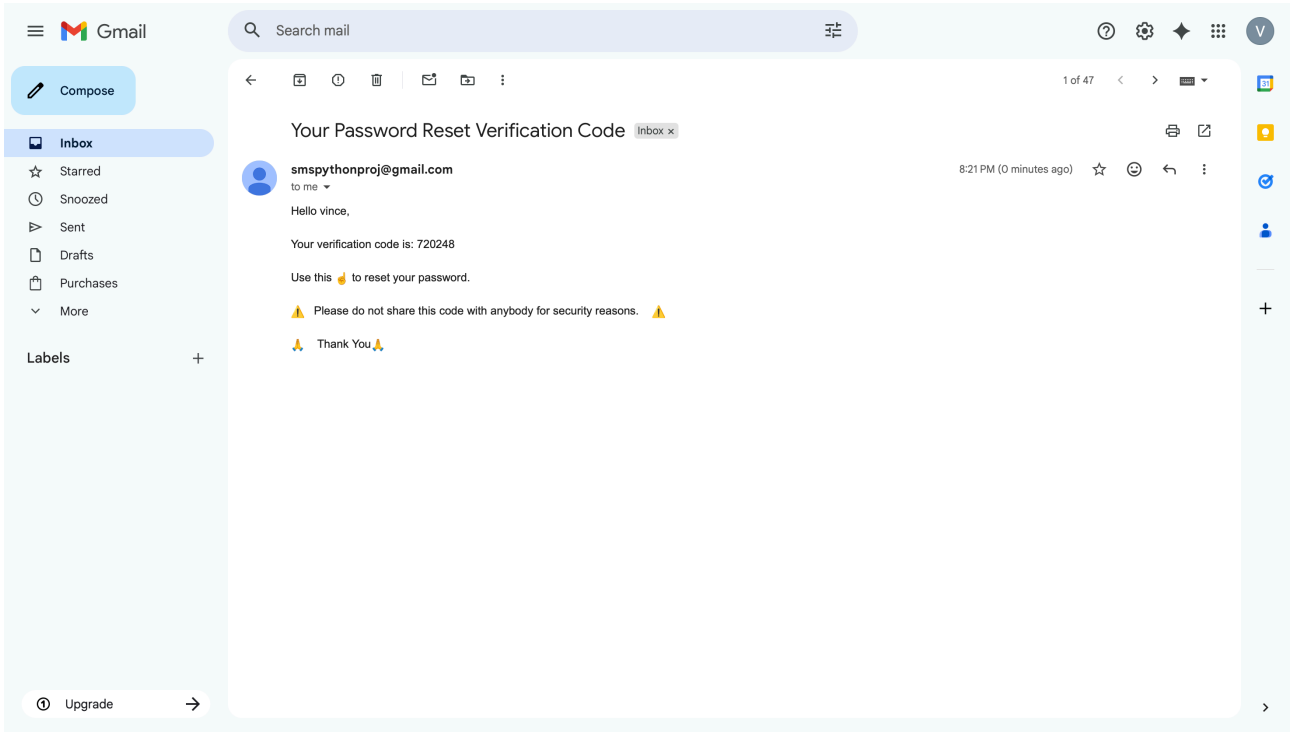

Friday, 14 November 2025 | 18:59:12
Developed by: Vittal and Vishnu
Version: v.1
OK
EDUMATRIX
PLUG IN. LEVEL UP. BREAK FREE.
ENTER EDUMATRIX
HELP
ABOUT
EXIT

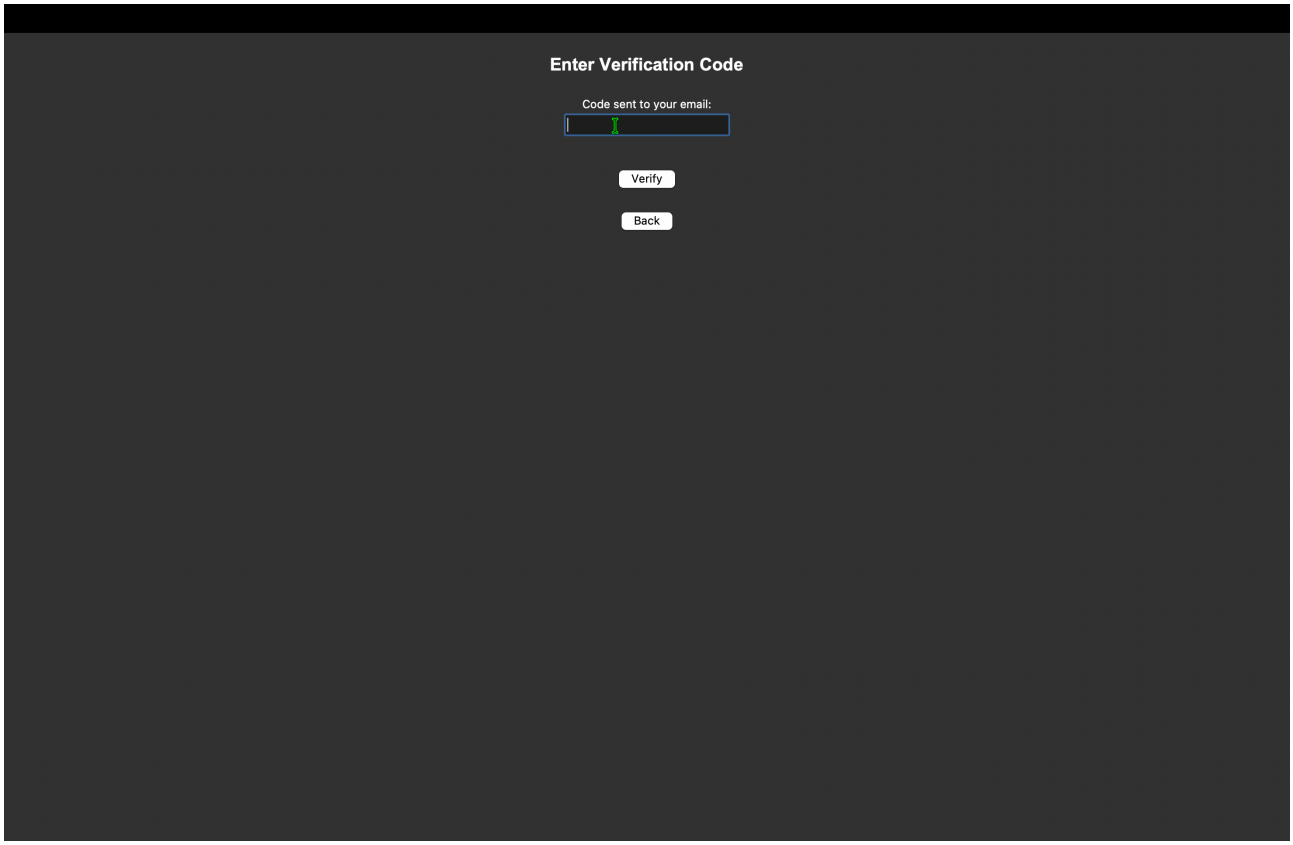

WELCOME TO EDUMATRIX
SELECT USER TYPE
ADMIN
STAFF
BACK

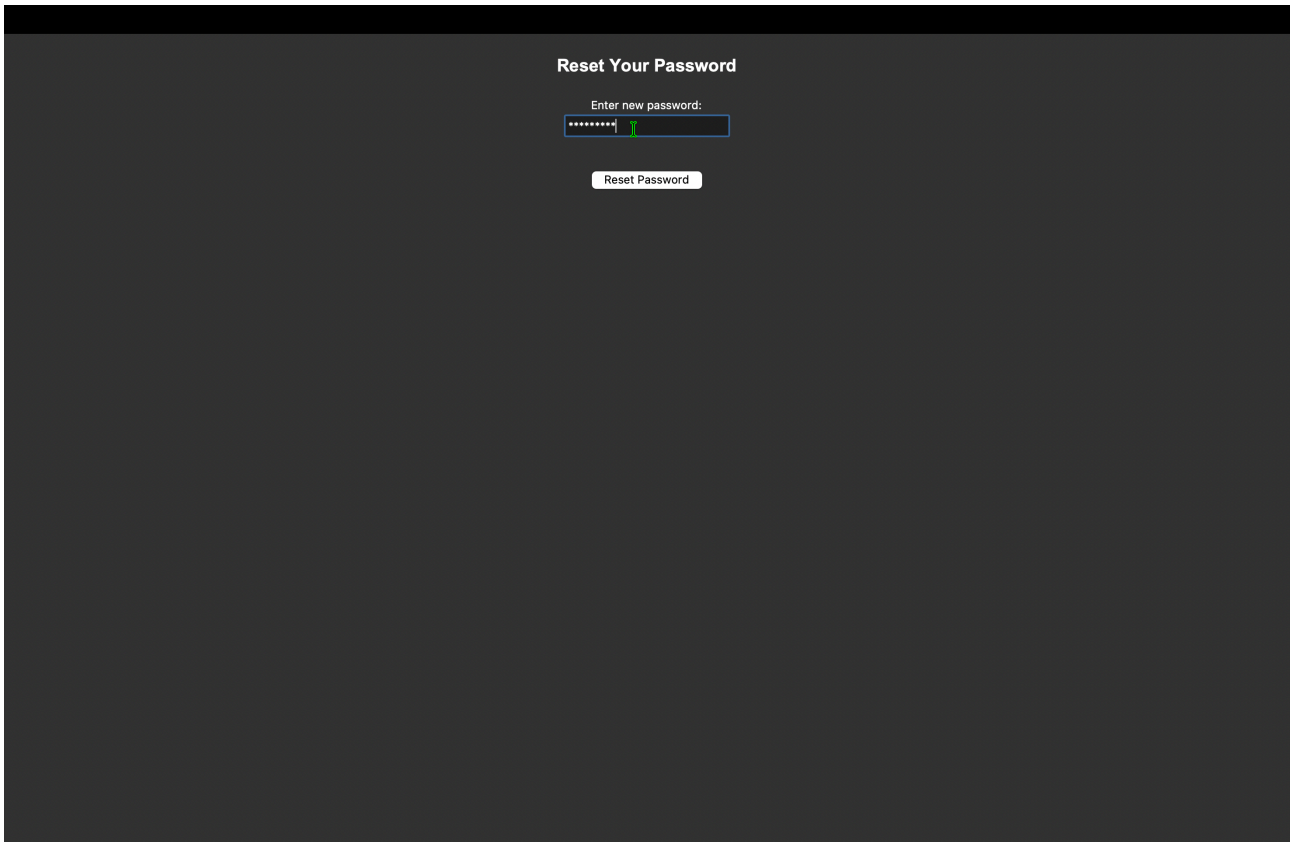

ADMIN LOGIN
Login
Sign Up
Back

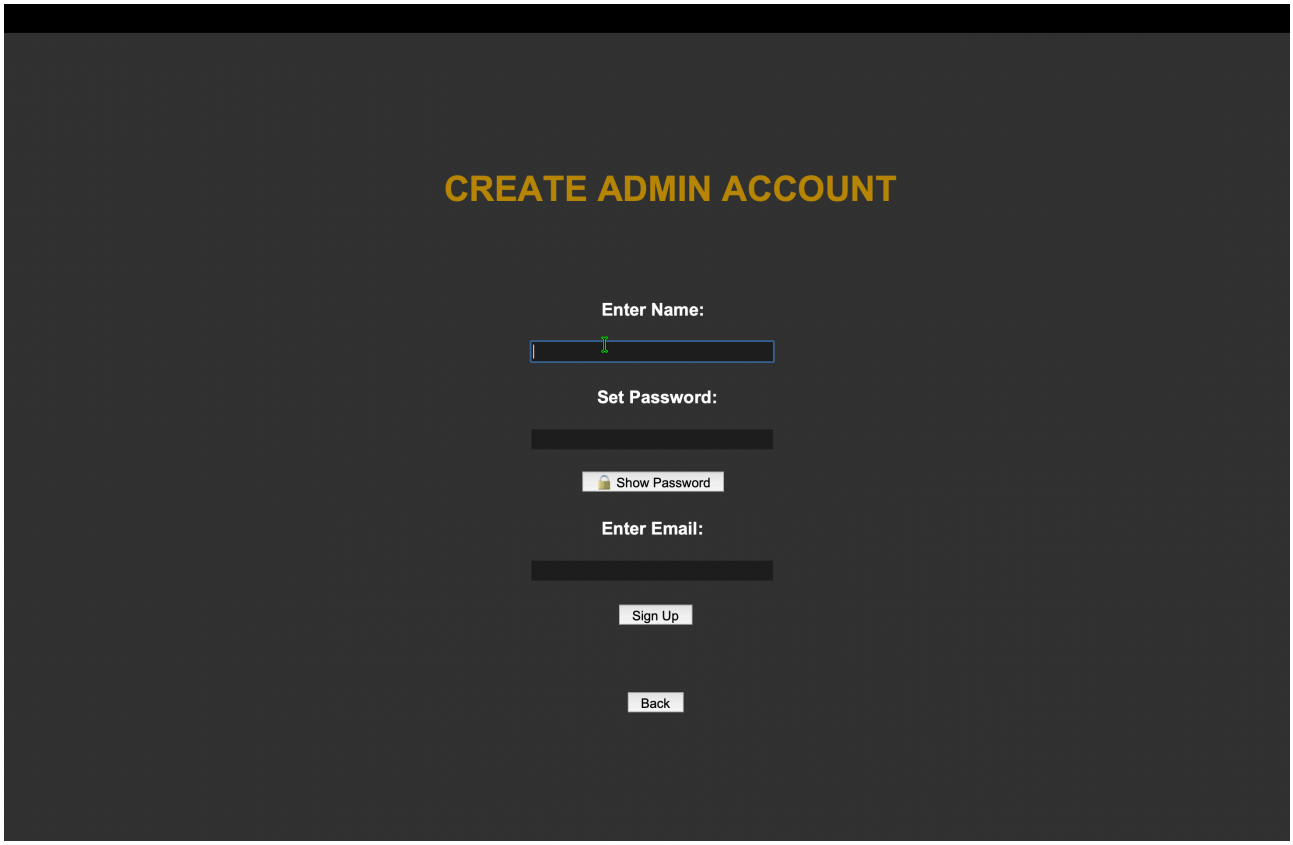

LOGIN
Username:
nacho
Password:
****
Show Password
Login
Reset Password
Back

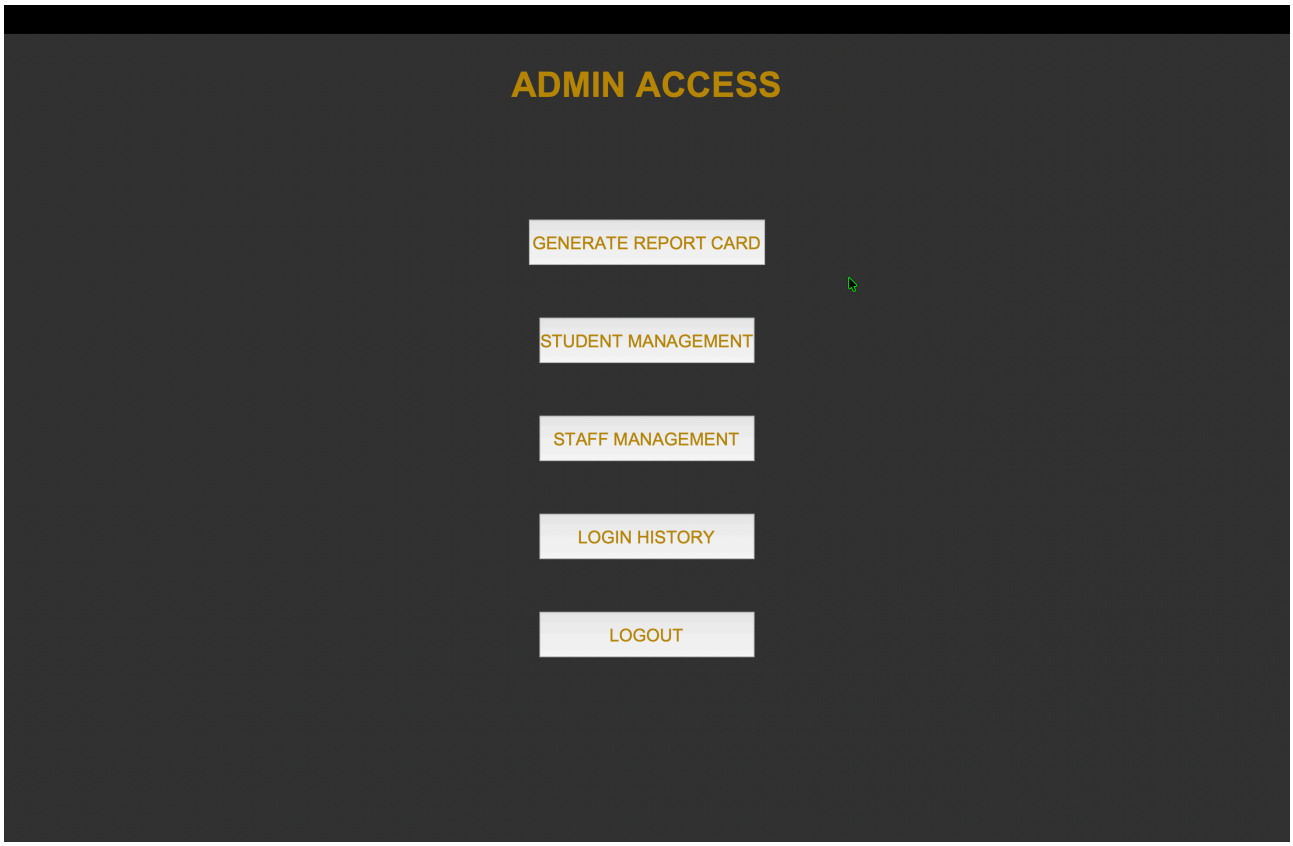

Reset Password
Enter your username:
vince
Send Code
Back

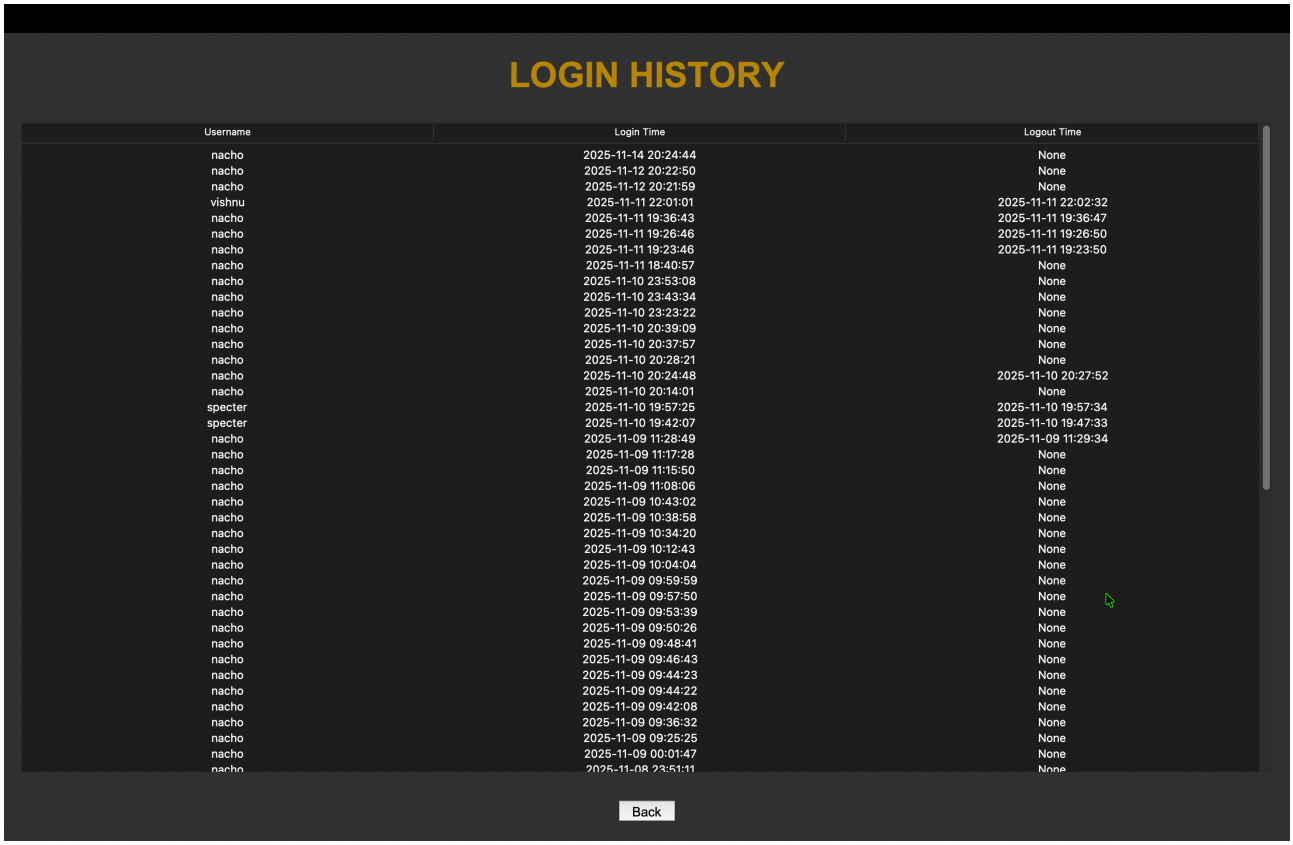

Gmail
Search mail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
Purchases
More
Labels
Upgrade
1 of 47
Your Password Reset Verification Code
Inbox ×
smspythonproj@gmail.com
to me
8:21 PM (0 minutes ago)
Hello vince,
Your verification code is: 720248
Use this ☝ to reset your password.
⚠ Please do not share this code with anybody for security reasons. ⚠
🙏 Thank You 🙏

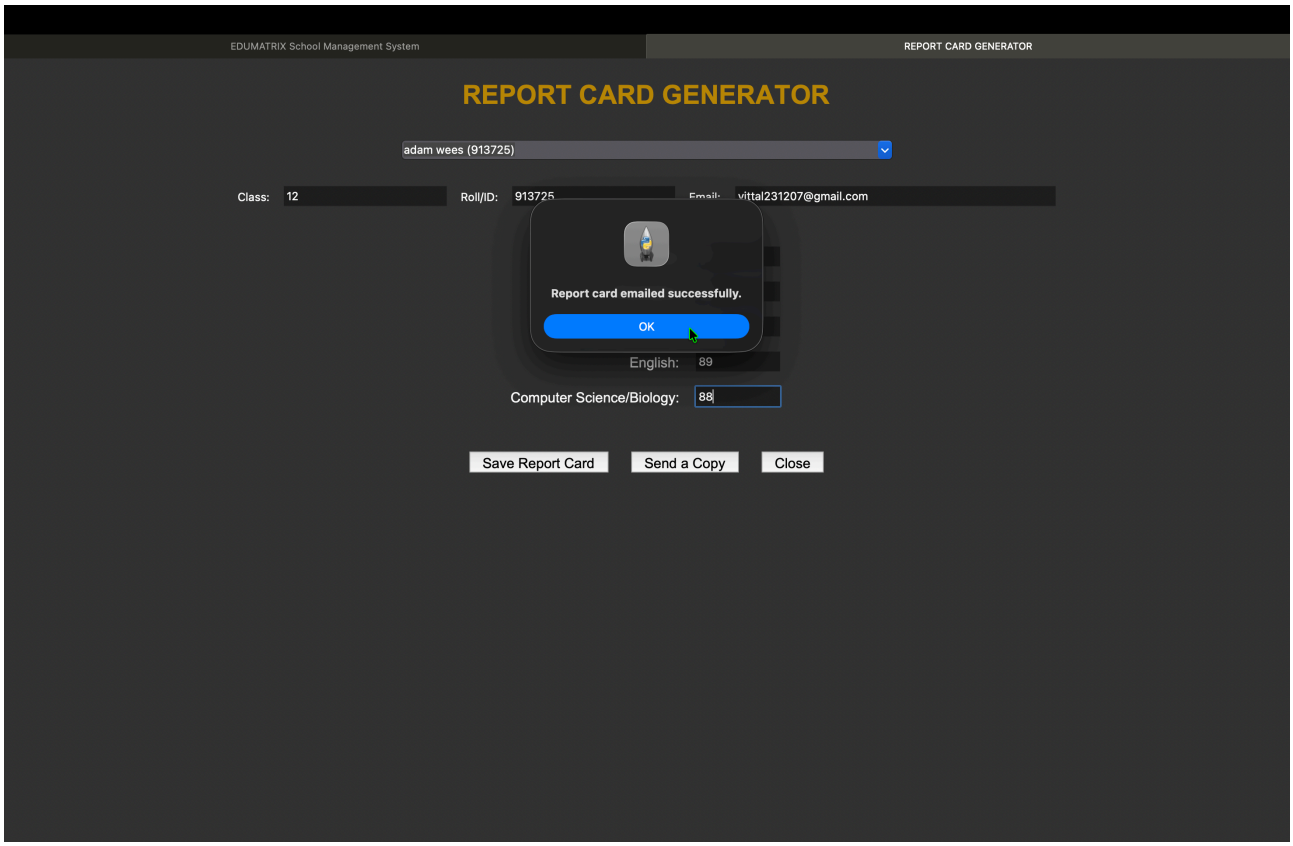

Enter Verification Code
Code sent to your email:
Verify
Back

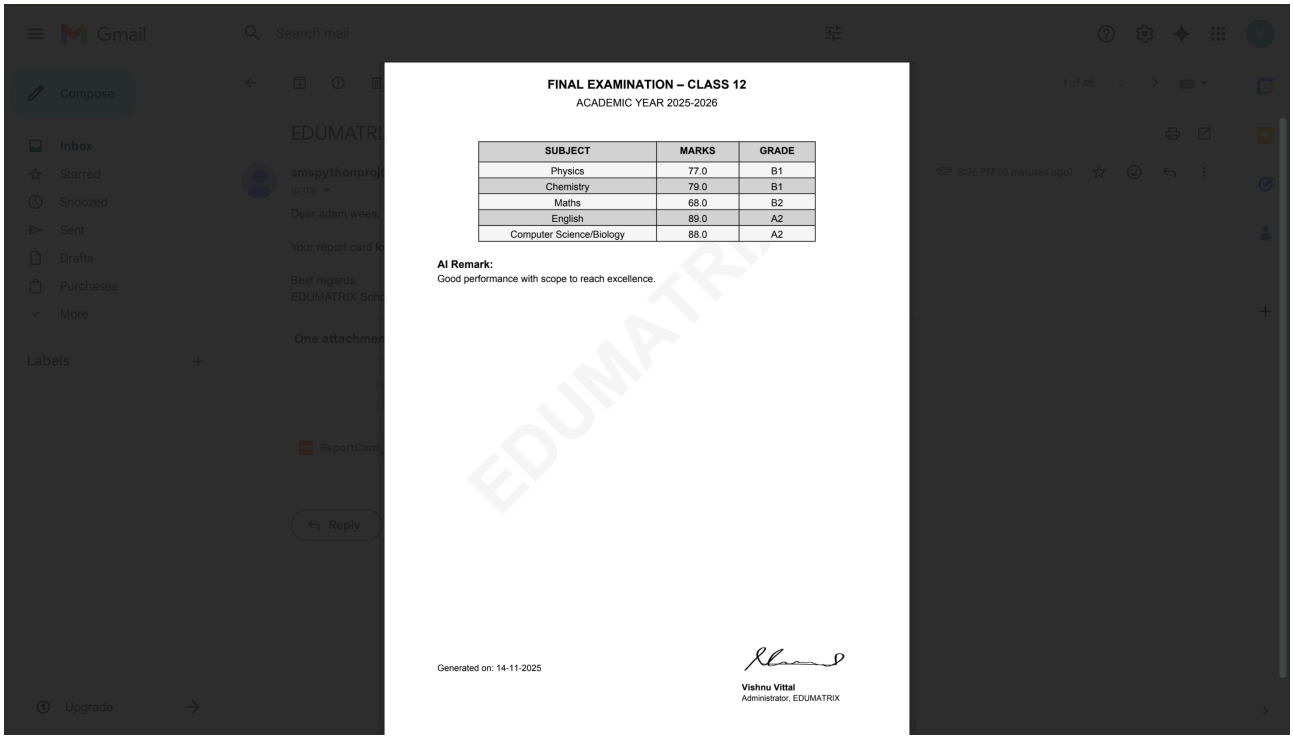

Reset Your Password
Enter new password:
Reset Password

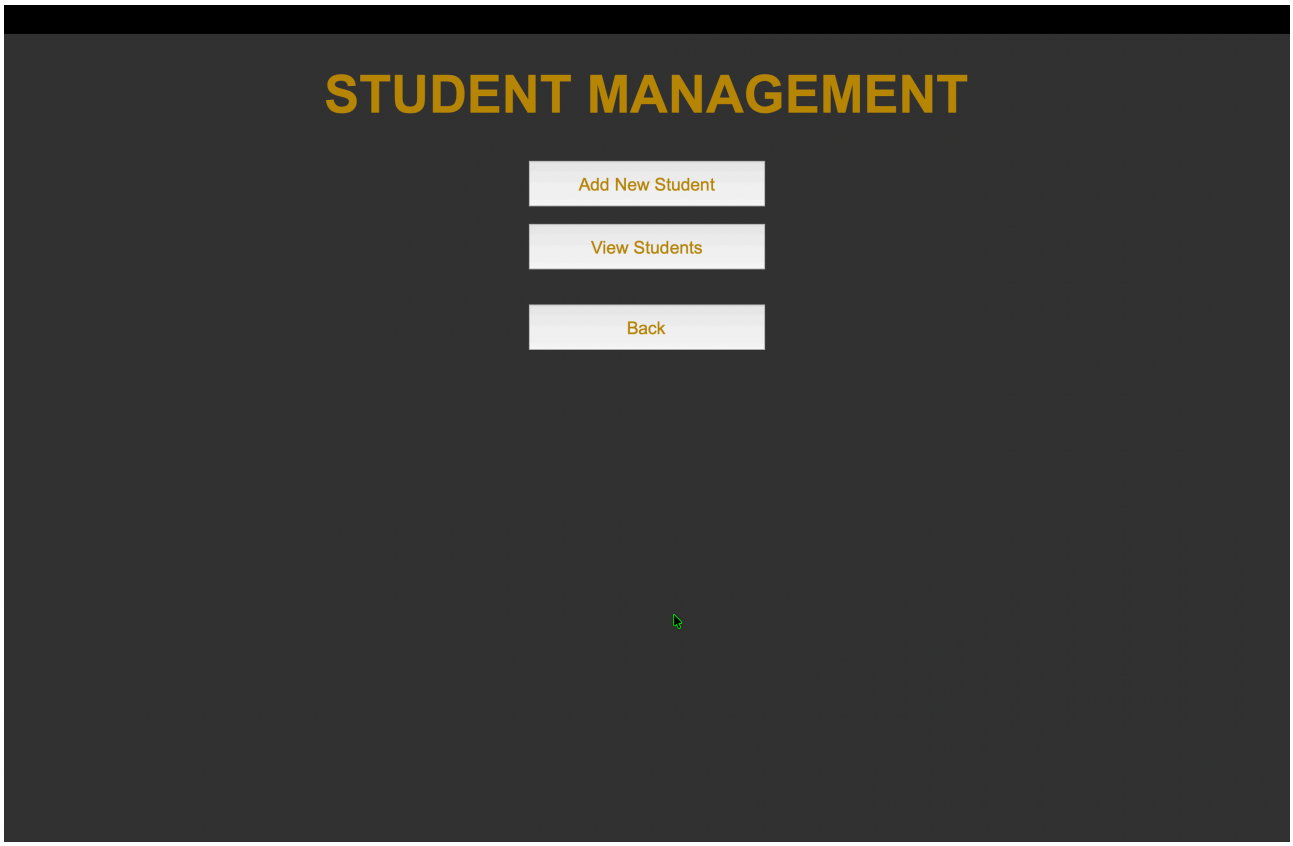

CREATE ADMIN ACCOUNT
Enter Name:
Set Password:
Show Password
Enter Email:
Sign Up
Back

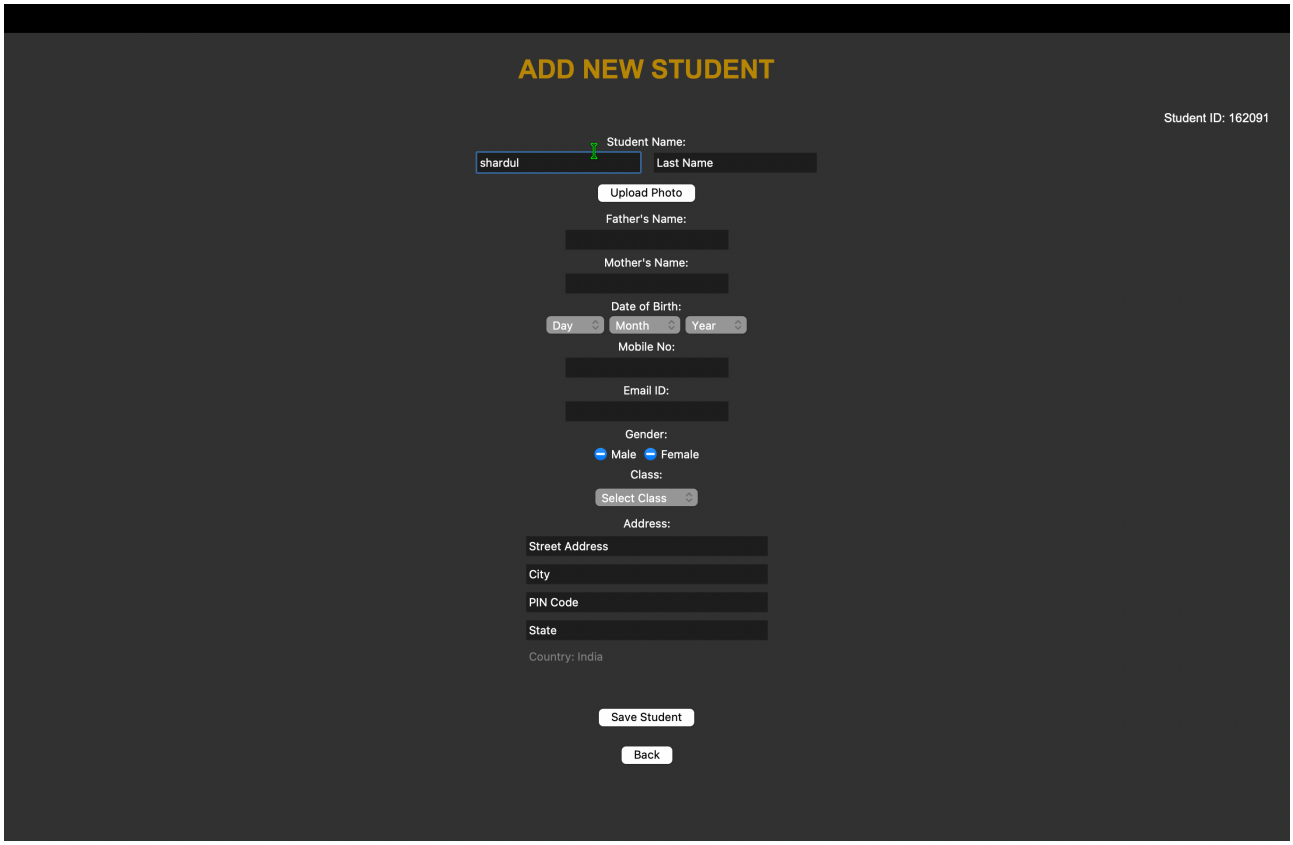

ADMIN ACCESS
GENERATE REPORT CARD
STUDENT MANAGEMENT
STAFF MANAGEMENT
LOGIN HISTORY
LOGOUT

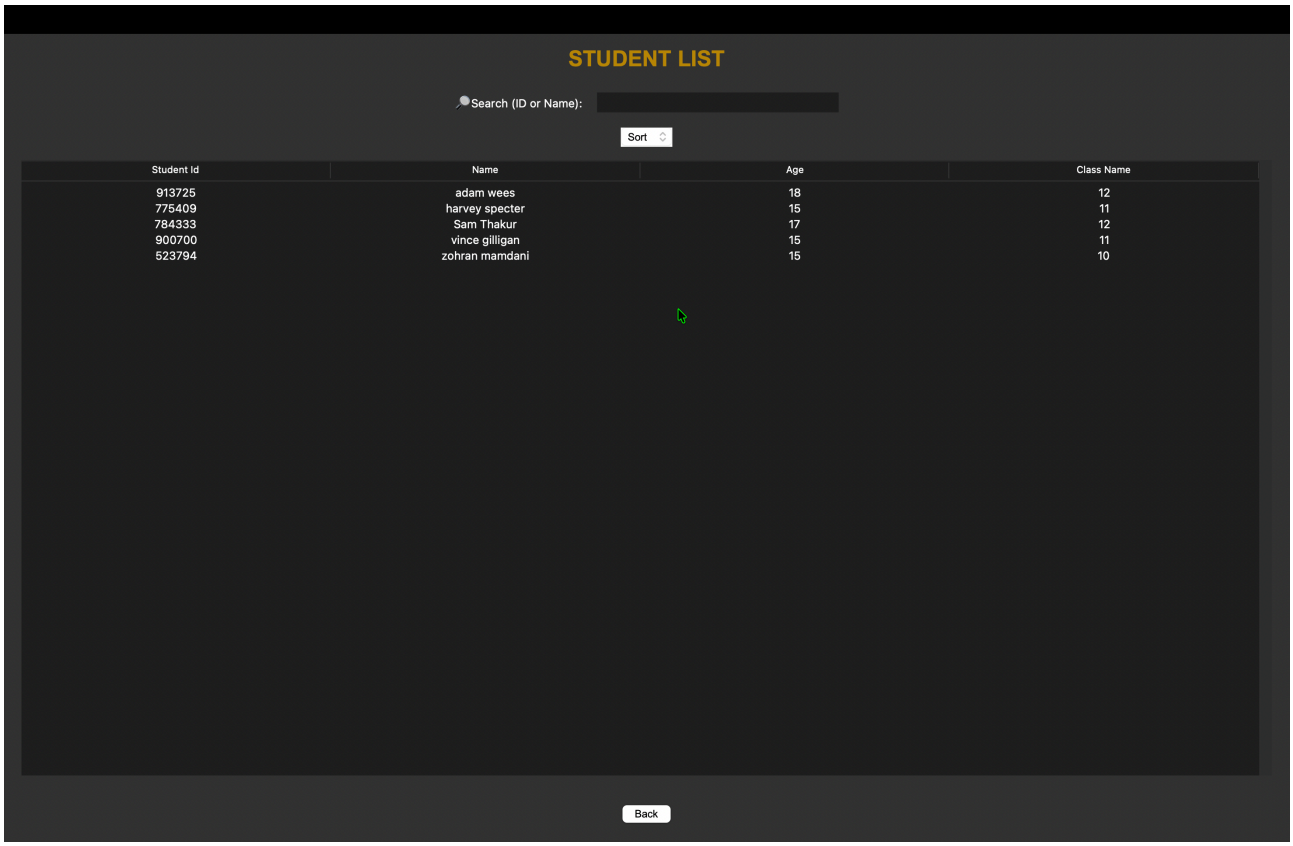

LOGIN HISTORY
Username
Login Time
Logout Time
nacho 2025-11-14 20:24:44 None
nacho 2025-11-12 20:22:50 None
nacho 2025-11-12 20:21:59 None
vishnu 2025-11-11 22:01:01 2025-11-11 22:02:32
nacho 2025-11-11 19:36:43 2025-11-11 19:36:47
nacho 2025-11-11 19:26:46 2025-11-11 19:26:50
nacho 2025-11-11 19:23:46 2025-11-11 19:23:50
nacho 2025-11-11 18:40:57 None
nacho 2025-11-10 23:53:08 None
nacho 2025-11-10 23:43:34 None
nacho 2025-11-10 23:23:22 None
nacho 2025-11-10 20:39:09 None
nacho 2025-11-10 20:37:57 None
nacho 2025-11-10 20:28:21 None
nacho 2025-11-10 20:24:48 2025-11-10 20:27:52
nacho 2025-11-10 20:14:01 None
specter 2025-11-10 19:57:25 2025-11-10 19:57:34
specter 2025-11-10 19:42:07 2025-11-10 19:47:33
nacho 2025-11-09 11:28:49 2025-11-09 11:29:34
nacho 2025-11-09 11:17:28 None
nacho 2025-11-09 11:15:50 None
nacho 2025-11-09 11:08:06 None
nacho 2025-11-09 10:43:02 None
nacho 2025-11-09 10:38:58 None
nacho 2025-11-09 10:34:20 None
nacho 2025-11-09 10:12:43 None
nacho 2025-11-09 10:04:04 None
nacho 2025-11-09 09:59:59 None
nacho 2025-11-09 09:57:50 None
nacho 2025-11-09 09:53:39 None
nacho 2025-11-09 09:50:26 None
nacho 2025-11-09 09:48:41 None
nacho 2025-11-09 09:46:43 None
nacho 2025-11-09 09:44:23 None
nacho 2025-11-09 09:44:22 None
nacho 2025-11-09 09:42:08 None
nacho 2025-11-09 09:36:32 None
nacho 2025-11-09 09:25:25 None
nacho 2025-11-09 00:01:47 None
Back


EDUMATRIX School Management System
REPORT CARD GENERATOR
REPORT CARD GENERATOR
adam wees (913725)
Class: 12
Roll/ID: 913725
vittal231207@gmail.com
Report card emailed successfully.
OK
English: 89
Computer Science/Biology: 88
Save Report Card
Send a Copy
Close


Gmail
Search mail
Compose
Inbox
Starred
Snoozed
Sent
Drafts
Purchases
More
Labels
Upgrade
Reply
FINAL EXAMINATION – CLASS 12
ACADEMIC YEAR 2025-2026
SUBJECT | MARKS | GRADE
Physics | 77.0 | B1
Chemistry | 79.0 | B1
Maths | 68.0 | B2
English | 89.0 | A2
Computer Science/Biology | 88.0 | A2
AI Remark:
Good performance with scope to reach excellence.
Generated on: 14-11-2025
Vishnu Vittal
Administrator, EDUMATRIX


STUDENT MANAGEMENT
Add New Student
View Students
Back


ADD NEW STUDENT
Student ID: 162091
Student Name:
shardul
Last Name
Upload Photo
Father's Name:
Mother's Name:
Date of Birth:
Day
Month
Year
Mobile No:
Email ID:
Gender:
Male
Female
Class:
Select Class
Address:
Street Address
City
PIN Code
State
Country: India
Save Student
Back


STUDENT LIST
Search (ID or Name):
Sort
Student Id
Name
Age
Class Name
913725 adam wees 18 12
775409 harvey specter 15 11
784333 Sam Thakur 17 12
900700 vince gilligan 15 11
523794 zohran mamdani 15 10
Back

EDUMATRIX School Management System

Student Profile

Student ID: 784333

Name: Sam Thakur

Father's Name: Shardul

Mother's Name: sakshi

Gender: Male

DOB: 2007-12-03

Age: 17

Class: 12

Mobile: 7483609183

Email: sam23@gmail.com

Address: 1St Cross, Mumbai, 400012, Maharashtra, India

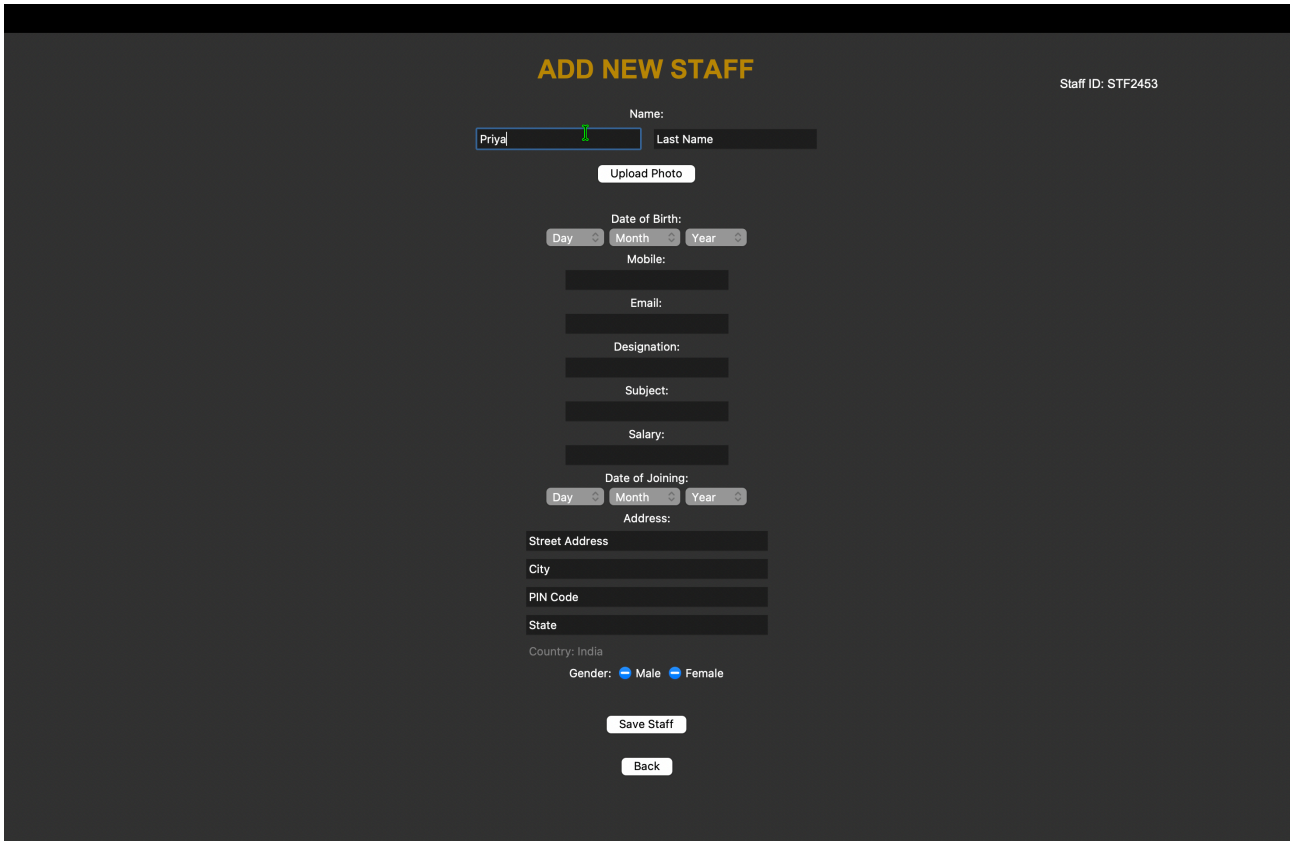
Save | Delete Profile | Promote Student

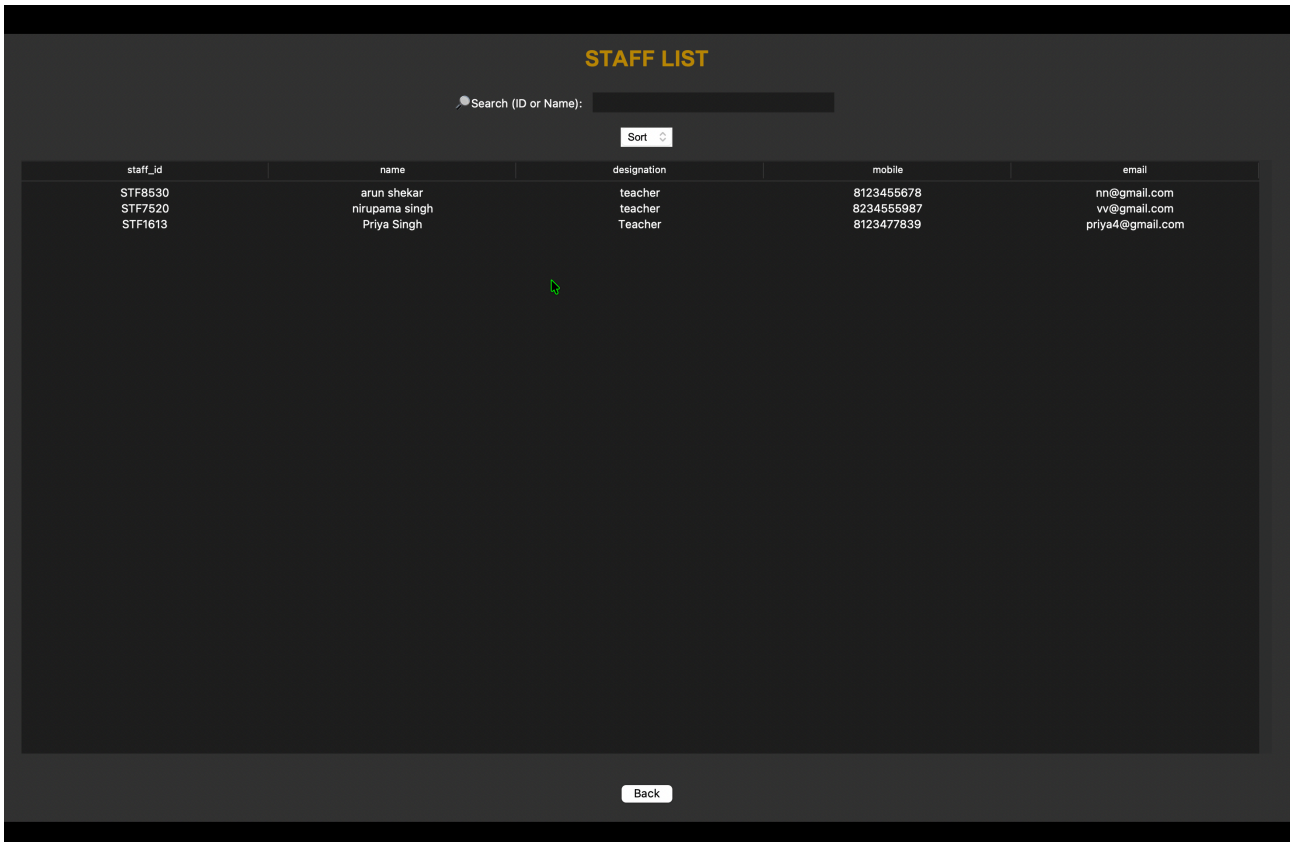
STAFF MANAGEMENT

Add New Staff

View Staff

Back


ADD NEW STAFF
Staff ID: STF2453
Name:
Priya
Last Name
Upload Photo
Date of Birth:
Day
Month
Year
Mobile:
Email:
Designation:
Subject:
Salary:
Date of Joining:
Day
Month
Year
Address:
Street Address
City
PIN Code
State
Country: India
Gender: Male Female
Save Staff
Back


STAFF LIST
Search (ID or Name):
Sort
staff_id
name
designation
mobile
email
STF8530
arun shekar
teacher
8123455678
nn@gmail.com
STF7520
nirupama singh
teacher
8234555987
vv@gmail.com
STF1613
Priya Singh
Teacher
8123477839
priya4@gmail.com
Back

EDUMATRIX School Management System

Staff Profile

Staff ID:	STF1613
Name:	Priya Singh
Gender:	Female
DOB:	2000-06-04
Age:	25
Mobile:	8123477839
Email:	priya4@gmail.com
Designation:	Teacher
Subject:	Science
Salary:	40000.0
Date of Joining:	2024-07-07
Address:	2 CROSS, MUMBAI, 400012, MAHARASHTRA, IN

Save Delete

STAFF ACCESS

Student List

Staff List

Back

STUDENT LIST

Search (ID or Name):

Sort

Student Id	Name	Age	Class Name
913725	adam wees	18	12
775409	harvey specter	15	11
784333	Sam Thakur	17	12
900700	vince gilligan	15	11
523794	zohran mamdani	15	10

Back

EDUMATRIX School Management System

Student Profile (Read-Only)

Student ID: 784333

Name: Sam Thakur

Father's Name: Shardul

Mother's Name: sakshi

Gender: Male

DOB: 2007-12-03

Age: 17

Class: 12

Mobile: 7483609183

Email: sam23@gmail.com

Address: 1St Cross, Mumbai, 400012, Maharashtra, India

Close

STAFF LIST

Search (ID or Name):

Sort

staff_id	name	designation	mobile	email
STF8530	arun shekar	teacher	8123455678	nn@gmail.com
STF7520	nirupama singh	teacher	8234555987	vv@gmail.com
STF1613	Priya Singh	Teacher	8123477839	priya4@gmail.com

Back

EDUMATRIX School Management System

Staff Profile (Read-Only)

Staff ID: STF1613

Name: Priya Singh

Gender: Female

DOB: 2000-06-04

Age: 25

Mobile: 8123477839

Email: priya4@gmail.com

Designation: Teacher

Subject: Science

Salary: 40000.0

Date of Joining: 2024-07-07

Address: 2 CROSS, MUMBAI, 400012, MAHARASHTRA, INDIA

Close

```
sqlite> .tables
admins         login_history  postal_codes   staff          students
sqlite> .schema
CREATE TABLE admins (
                    username TEXT PRIMARY KEY,
                    password TEXT NOT NULL,
                    email TEXT NOT NULL
                );
CREATE TABLE students (
                        student_id TEXT PRIMARY KEY,
                        name TEXT NOT NULL,
                        father_name TEXT,
                        mother_name TEXT,
                        dob TEXT,
                        age INTEGER,
                        mobile TEXT,
                        email TEXT,
                        gender TEXT,
                        class_name TEXT,
                        address TEXT,
                        photo TEXT
                    );
CREATE TABLE staff (
                    staff_id TEXT PRIMARY KEY,
                    name TEXT NOT NULL,
                    dob TEXT,
                    age INTEGER,
                    mobile TEXT,
                    email TEXT,
                    gender TEXT,
                    designation TEXT,
                    subject TEXT,
                    salary REAL,
                    date_of_joining TEXT,
                    address TEXT,
                    photo TEXT
                );
CREATE TABLE login_history (
                    id INTEGER PRIMARY KEY AUTOINCREMENT,
                    username TEXT,
                    login_time TEXT,
                    logout_time TEXT
                );
CREATE TABLE sqlite_sequence(name,seq);
CREATE TABLE postal_codes (
                    city TEXT,
                    pin TEXT,
                    state TEXT
                );
```

```
sqlite> PRAGMA table_info(admins);
0|username|TEXT|0||1
1|password|TEXT|1||0
2|email|TEXT|1||0
```

```
sqlite> PRAGMA table_info(login_history);
0|id|INTEGER|0||1
1|username|TEXT|0||0
2|login_time|TEXT|0||0
3|logout_time|TEXT|0||0
```

```
sqlite> PRAGMA table_info(postal_codes);
0|city|TEXT|0||0
1|pin|TEXT|0||0
2|state|TEXT|0||0
sqlite>
```

```
sqlite> PRAGMA table_info(staff);
0|staff_id|TEXT|0||1
1|name|TEXT|1||0
2|dob|TEXT|0||0
3|age|INTEGER|0||0
4|mobile|TEXT|0||0
5|email|TEXT|0||0
6|gender|TEXT|0||0
7|designation|TEXT|0||0
8|subject|TEXT|0||0
9|salary|REAL|0||0
10|date_of_joining|TEXT|0||0
11|address|TEXT|0||0
12|photo|TEXT|0||0
```

```
sqlite> PRAGMA table_info(students);
0|student_id|TEXT|0||1
1|name|TEXT|1||0
2|father_name|TEXT|0||0
3|mother_name|TEXT|0||0
4|dob|TEXT|0||0
5|age|INTEGER|0||0
6|mobile|TEXT|0||0
7|email|TEXT|0||0
8|gender|TEXT|0||0
9|class_name|TEXT|0||0
10|address|TEXT|0||0
11|photo|TEXT|0||0
```

FUTURE UPDATES

The **EDUMATRIX – School Management System** can be further improved with several new features and enhancements in the future. Some possible updates include:

1. **Complete Attendance Management Integration**
   The system can be upgraded to include automatic attendance tracking for both students and staff. This may include daily attendance entry, monthly reports, and attendance percentage calculations directly linked to each profile.

2. **Advanced Marks & Result Analytics**
   EDUMATRIX can be improved with visual charts and statistics showing student performance trends, subject-wise strengths/weaknesses, and batch comparisons using analytical graphs.

3. **Fee & Payment Tracking Module**
   A dedicated fee management feature can be included for handling fee submission dates, payment status, pending dues, and automated reminders for students and parents.

4. **Parent Portal (Web or App Extension)**
   A simple website or mobile-friendly portal can be added where parents can view report cards, attendance, announcements, and receive notifications in real time.

5. **Cloud Sync & Online Database**
   The system can be upgraded to store all records on a cloud server (like Firebase or MySQL online), allowing remote access, automatic backup, and use on multiple devices.

6. **Auto-Generated Timetable & Scheduling**
   A timetable creation tool can be added to automatically generate class schedules, assign teachers, and avoid subject/teacher clashes using smart algorithms.

LIMITATIONS

Although the **EDUMATRIX – School Management System** provides a reliable and efficient way to manage school data, it has certain limitations that can be improved in future versions:

1. **Single-System Operation:**
   The current version works only on a single computer and does not support multi-user networking or online access.

2. **Local Database Storage:**
   Since data is stored in an SQLite database, it can only be accessed locally. Cloud or remote access is not available.

3. **Limited User Roles:**
   The system currently supports only two roles — Admin and Staff. Parent and Student access modules are not yet included.

4. **No Real-Time Notifications:**
   The system does not send automatic email or SMS alerts for updates, reminders, or announcements.

5. **Manual Data Entry:**
   Most data (like attendance or grades) must be entered manually, which can be time-consuming for large institutions.

6. **Offline System:**
   The system cannot be used over the internet or mobile devices; it must be run locally on a desktop or laptop.

MINIMUM SYSTEM REQUIREMENTS

Hardware Requirements

- **Processor:** Intel Core i3 or higher
- **RAM:** Minimum 4 GB
- **Storage:** At least 500 MB of free disk space
- **Display:** 1366 × 768 resolution or higher
- **Input Devices:** Keyboard and Mouse
- **Optional:** Internet connection (for email verification and future updates)

Software Requirements

- **Operating System:** Windows 10 / 11, macOS, or Linux
- **Programming Language:** Python 3.10 or above
- **Database:** SQLite3 (pre-installed with Python)

CONCLUSION

The **EDUMATRIX – School Management System** successfully demonstrates how computer applications can simplify and enhance the administrative functioning of a school. By integrating Python, SQLite, Tkinter, and various supporting libraries, the project provides a secure and user-friendly platform for managing critical tasks such as storing student and staff information, generating report cards, and emailing results.

The system replaces traditional manual processes with automated digital workflows, reducing errors, saving time, and improving accuracy. Features such as encrypted admin authentication, access-code verification, PDF report generation, and AI-based performance remarks further strengthen the reliability and usefulness of the software.

Although the current version has certain limitations, it establishes a strong foundation for further enhancements like cloud integration, attendance tracking, fee management, and role-based access. Overall, this project highlights the importance of programming and database management in solving real-world problems and demonstrates how technology can be used to create a smarter, more efficient, and paperless school administration system.

REFERENCES

1. COMPUTER SCIENCE WITH PYTHON - Textbook for class XI and XII by Sumita Arora

2. Computer teacher - Mrs. Sona Maria Fernandes

3. Youtube

4. Google