DATABASE MANAGEMENT SYSTEM PROJECT

STUDENT MANAGEMENT SYSTEM

Submitted By:

[redacted]

TABLE OF CONTENTS

INTRODUCTION

The Student Management System is a Python-based graphical user interface (GUI) application designed to manage and organize student records. Developed using the Tkinter library for the frontend and SQLite for backend data storage, this system provides a user-friendly interface for adding, viewing, and deleting student information. The program utilizes features such as themed GUI elements, date entry using the tkcalendar module, and database operations for efficient data management.

The system incorporates a login functionality to ensure secure access, and upon successful authentication, users are granted access to the main application window. The main features of the application include adding new student records, deleting records, viewing details, and resetting form fields. The records are displayed in a treeview widget, allowing for easy navigation and management.

The code emphasizes a clean and organized structure, with well-defined functions for each operation. It integrates error handling to ensure data integrity and provides informative messages to users. Additionally, the application includes a database reset option and a visually appealing theme for enhanced user experience.

This report will further delve into the abstract, functional and non-functional requirements, database structure, source code overview, screenshots, and references, providing a comprehensive understanding of the Student Management System.

ABSTRACT

The Student Management System (SMS) is a Python-based application developed to streamline the management of student records in an educational environment. This system offers a user-friendly graphical interface using the Tkinter library, facilitating easy interaction and efficient handling of student data. The primary purpose of the SMS is to provide administrators with a centralized tool for adding, viewing, and deleting student records, enhancing the overall management process.

Key features of the system include a secure login mechanism to control access, a well-organized user interface, and integration with a SQLite database for data storage. The login functionality ensures that only authorized personnel can access and manipulate the student records, adding a layer of security to the application. Upon successful authentication, users are presented with a main window that hosts various functionalities for managing student information.

The SMS employs a structured approach to data entry, utilizing themed GUI elements for a visually appealing experience. It incorporates a date entry feature using the tkcalendar module, enabling the input of accurate and standardized date information. The system also implements error handling to validate user inputs, ensuring data integrity and preventing potential issues.

Additionally, the SMS provides essential operations such as adding new student records, deleting existing records, viewing detailed information, and resetting form fields. The use of a treeview widget facilitates a clear display of student records, allowing for easy navigation and manipulation of data. The system also includes an option to reset the entire database, providing administrators with a comprehensive tool for database management.

This abstract sets the stage for a detailed exploration of the Student Management System, covering functional and non-functional requirements, database structure, source code, screenshots, and references in subsequent sections.

FUNCTIONAL REQUIREMENTS

1. User Authentication

- The system shall have a secure login mechanism to control access to the application.
- Only authorized users with valid credentials (username and password) are allowed to log in.
- Default credentials for initial access are set to "admin" for both username and password.

2. Main Window

- After successful login, the system shall display a main window providing access to various functionalities.
- The main window shall include options for adding, viewing, and deleting student records.

3. Student Record Management

- Add Record:

- The system shall allow the user to input student details, including name, email, contact number, gender, date of birth, and stream.
- Mandatory fields (name, email, contact, gender, DOB, stream) must be filled to add a new record.
- Contact number validation ensures a 10-digit numeric entry.
- The system shall display informative messages on successful record addition or error in case of invalid inputs.

- View Record:

- The system shall provide an option to view detailed information for a selected student record.
- Clicking on a record in the displayed treeview shall populate the corresponding fields with the record details.

- Delete Record:

- Users can delete a selected record, and the system shall confirm the deletion with a prompt.
- On successful deletion, the system shall update the display to reflect the changes.

- Reset Fields:

  - There shall be an option to reset all input fields to their default or empty state.

  - Resetting fields shall not affect the existing records in the database.

4. Database Operations

- The system shall utilize an SQLite database for storing student records.
- It shall create the necessary table structure if not already present.
- A reset option shall be available to delete the entire database, requiring confirmation to prevent accidental data loss.

5. Data Display

- The system shall use a treeview widget to display student records in a clear and organized manner.
- The displayed records shall include student ID, name, email, contact number, gender, date of birth, and stream.

6. Error Handling

- The system shall incorporate error handling mechanisms to validate user inputs.
- Informative error messages shall be displayed in case of missing or incorrect data.

7. Date Entry

- The system shall employ the tkcalendar module for accurate and standardized date entry.
- The date of birth (DOB) for each student shall be selected using the date entry feature.

8. User Interface (UI)

- ThemedTk from ttkthemes shall be used to enhance the visual appeal of the user interface.
- Fonts and colors shall be chosen to provide a cohesive and pleasant user experience.

These functional requirements define the core features and operations of the Student Management System, ensuring effective student record management and user interaction.

NON-FUNCTIONAL REQUIREMENTS

1. Usability

- User Interface Design:

  - The system shall have an intuitive and visually appealing user interface designed for ease of use.
  - ThemedTk shall be utilized to enhance the aesthetics of the application.

- User Guidance:

  - The system shall provide informative prompts, tooltips, and error messages to guide the user during interactions.

2. Performance

- Response Time:

  - The system shall exhibit low latency in responding to user inputs and actions.
  - Operations such as adding, viewing, and deleting records shall execute promptly.

- Database Performance:

  - Database queries and operations shall be optimized for efficiency to ensure fast data retrieval and manipulation.

3. Security

- Login Security:

  - User authentication shall be secure, and passwords shall be stored using encryption.
  - Default credentials shall be changed upon first login.

- Data Integrity:

  - The system shall validate user inputs to maintain data integrity, preventing the addition of incomplete or invalid records.

4. Reliability

- Error Handling:

  - The system shall gracefully handle errors and exceptions to prevent unexpected crashes.

- User-friendly error messages shall be displayed to assist users in resolving issues.

- Database Reliability:

- The database connection and operations shall be robust, ensuring reliability even in the presence of unexpected events.

5. Scalability

- Data Volume:

- The system shall accommodate a moderate to large volume of student records without significant performance degradation.

6. Portability

- Cross-Platform Compatibility:

- The application shall be designed to run seamlessly across different operating systems, including Windows, Linux, and macOS.

- External Dependency:

- Dependencies such as the tkcalendar module and ttkthemes shall be managed to ensure consistent behavior across platforms.

7. Maintainability

- Code Readability:

- The source code shall follow best practices for readability and maintainability.
- Comments and documentation shall be provided where necessary.

- Database Schema Updates:

- The system shall accommodate future updates to the database schema without causing data loss or disruption.

8. Availability

- System Uptime:

- The system shall have high availability, with minimal downtime for maintenance or updates.

- Data Backup:

  - Regular backups of the database shall be performed to prevent data loss in the event of system failures.

9. Theme Customization

- User Theme Selection:

  - The system shall allow users to choose from different themes for personalization.

  - Theme changes shall not affect the functionality of the application.

These non-functional requirements address aspects related to the usability, performance, security, reliability, scalability, portability, maintainability, availability, and theme customization of the Student Management System, ensuring a robust and user-friendly experience.

DATABASE

The Student Management System utilizes an SQLite database for the storage and retrieval of student records. The database is named "StudentManagement.db" and is created with the following table structure:

```
CREATE TABLE IF NOT EXISTS STUDENT_MANAGEMENT (
    STUDENT_ID INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL,
    NAME TEXT,
    EMAIL TEXT,
    PHONE_NO TEXT,
    GENDER TEXT,
    DOB TEXT,
    STREAM TEXT
);
```

The table `STUDENT_MANAGEMENT` includes the following fields:
1. STUDENT_ID (INTEGER): A unique identifier for each student, automatically incremented for each new record.
2. NAME (TEXT):The name of the student.
3. EMAIL (TEXT):The email address of the student.
4. PHONE_NO (TEXT):The contact number of the student.
5. GENDER (TEXT):The gender of the student.
6. DOB (TEXT):The date of birth of the student, stored as text.
7. STREAM (TEXT):The academic stream or course that the student is enrolled in.

This database schema provides a structured format for storing student information and enables efficient retrieval and manipulation of records. The use of SQLite as the database engine ensures simplicity and portability for small to medium-scale applications like the Student Management System.

SOURCE CODE

```
from tkinter import *
from tkinter import ttk, simpledialog
import tkinter.messagebox as mb
from tkcalendar import DateEntry
import sqlite3
import datetime
from ttkthemes import ThemedTk

def login():
    create_login_window()

# Create a hidden main window
main = ThemedTk(theme="breeze")
main.withdraw()

# Creating the universal font variables
headlabelfont = ("Noto Sans CJK TC", 15, 'bold')
labelfont = ('Palatino', 14)
entryfont = ('Palatino', 12)

# Connecting to the Database where all information will be stored
connector = sqlite3.connect('StudentManagement.db',
detect_types=sqlite3.PARSE_DECLTYPES | sqlite3.PARSE_COLNAMES)
cursor = connector.cursor()
connector.execute(
    "CREATE TABLE IF NOT EXISTS STUDENT_MANAGEMENT (STUDENT_ID
INTEGER PRIMARY KEY AUTOINCREMENT NOT NULL, NAME TEXT, EMAIL TEXT,
PHONE_NO TEXT, GENDER TEXT, DOB TEXT, STREAM TEXT)"
)

def create_login_window():
    login_window = Toplevel(main)
    login_window.title("Login")
    login_window.geometry("400x200")
    login_window.resizable(0, 0)
    login_window.configure(bg='#3498DB')

    Label(login_window, text="Username:", font=("Helvetica", 14),
bg='#3498DB', fg='white').grid(row=0, column=0, pady=10, padx=10)
    Label(login_window, text="Password:", font=("Helvetica", 14),
bg='#3498DB', fg='white').grid(row=1, column=0, pady=10, padx=10)

    username_entry = Entry(login_window, font=("Helvetica", 14))
    password_entry = Entry(login_window, show="*",
font=("Helvetica", 14))
```

```
    username_entry.grid(row=0, column=1, pady=10, padx=10)
    password_entry.grid(row=1, column=1, pady=10, padx=10)

    def validate_login():
        username = username_entry.get()
        password = password_entry.get()

        #authentication logic goes here
        if username == "admin" and password == "admin123":
            mb.showinfo("Login Successful", "Welcome, Admin!")
            main.deiconify()  # Show the main window after
successful login
            login_window.destroy()  # Close the login window
        else:
            mb.showerror("Login Failed", "Invalid username or
password")

    def on_login_window_close():
        main.destroy()

    login_window.protocol("WM_DELETE_WINDOW", on_login_window_close)

    login_button = Button(login_window, text="Login",
command=validate_login, font=("Helvetica", 14))
    login_button.grid(row=2, column=1, pady=20, padx=10)

# Create a login window
login()
# Creating the functions
def reset_fields():
    global name_strvar, email_strvar, contact_strvar, gender_strvar,
dob, stream_strvar
    for i in ['name_strvar', 'email_strvar', 'contact_strvar',
'gender_strvar', 'stream_strvar']:
        exec(f"{i}.set('')")
    dob.set_date(datetime.datetime.now().date())

def reset_form():
    global tree
    tree.delete(*tree.get_children())
    reset_fields()

def display_records():
    tree.delete(*tree.get_children())
    curr = connector.execute('SELECT * FROM STUDENT_MANAGEMENT')
    data = curr.fetchall()
    for records in data:
```

```
        tree.insert('', END, values=records)

def add_record():
    global name_strvar, email_strvar, contact_strvar, gender_strvar,
dob, stream_strvar
    name = name_strvar.get()
    email = email_strvar.get()
    contact = contact_strvar.get()
    gender = gender_strvar.get()
    DOB = dob.get_date()
    stream = stream_strvar.get()

    if not name or not email or not contact or not gender or not DOB
or not stream:
        mb.showerror('Error!', "Please fill all the missing
fields!!")
    elif not contact.isdigit() or len(contact) != 10:
        mb.showerror('Error!', 'Please enter a valid 10-digit phone
number.')
    else:
        try:
            connector.execute(
                'INSERT INTO STUDENT_MANAGEMENT (NAME, EMAIL,
PHONE_NO, GENDER, DOB, STREAM) VALUES (?,?,?,?,?,?)',
                (name, email, contact, gender, DOB, stream)
            )
            connector.commit()
            mb.showinfo('Record added', f"Record of {name} was
successfully added")
            reset_fields()
            display_records()
        except:
            mb.showerror('Wrong type',
                         'The type of the values entered is not
accurate. Pls note that the contact field can only contain numbers')
def remove_record():
    if not tree.selection():
        mb.showerror('Error!', 'Please select an item from the
database')
    else:
        current_item = tree.focus()
        values = tree.item(current_item)
        selection = values["values"]
        tree.delete(current_item)
        connector.execute('DELETE FROM STUDENT_MANAGEMENT WHERE
STUDENT_ID=%d' % selection[0])
        connector.commit()
```

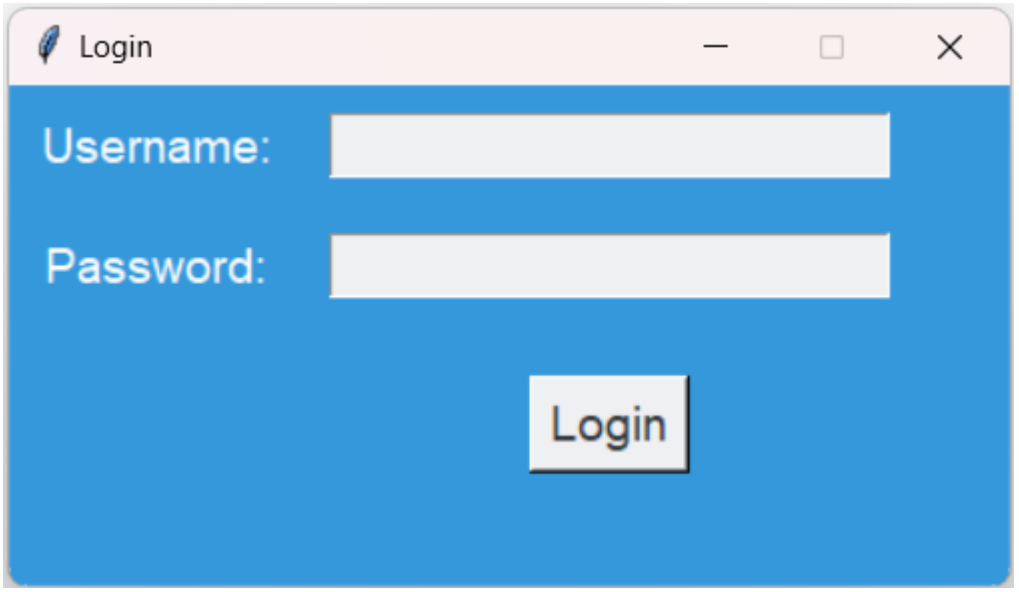
```
        mb.showinfo('Done', 'The record you wanted deleted was
successfully deleted.')
        display_records()

def view_record():
    global name_strvar, email_strvar, contact_strvar, gender_strvar,
dob, stream_strvar
    if not tree.selection():
        mb.showerror('Error!', 'Please select a record to view')
    else:
        current_item = tree.focus()
        values = tree.item(current_item)
        selection = values["values"]

        name_strvar.set(selection[1]);
        email_strvar.set(selection[2])
        contact_strvar.set(selection[3]);
        gender_strvar.set(selection[4])
        date = datetime.date(int(selection[5][:4]),
int(selection[5][5:7]), int(selection[5][8:]))
        dob.set_date(date);
        stream_strvar.set(selection[6])
def reset_form():
    global tree, connector
    if mb.askyesno('Delete Database', 'Are you sure you want to
delete the entire database?'):
        connector.execute('DROP TABLE IF EXISTS STUDENT_MANAGEMENT')
        connector.commit()
        mb.showinfo('Database Deleted', 'The entire database has
been deleted successfully.')
        tree.delete(*tree.get_children())
        reset_fields()

# Initializing the GUI window
main.title('DBMS Student Management System')
main.geometry('1000x600')
main.resizable(0, 0)

# Creating the background and foreground color variables
lf_bg = '#2E3B4E'  # Dark Blue
cf_bg = '#34495E'  # Steel Blue
btn_bg = '#3498DB'  # Dodger Blue
btn_fg = 'white'

# Creating the StringVar or IntVar variables
name_strvar = StringVar()
email_strvar = StringVar()
contact_strvar = StringVar()
```

```
gender_strvar = StringVar()
stream_strvar = StringVar()

# Placing the components in the main window
Label(main, text="STUDENT MANAGEMENT SYSTEM", font=headlabelfont,
bg='#3498DB', fg='white').pack(side=TOP, fill=X)
left_frame = Frame(main, bg=lf_bg)
left_frame.place(x=0, y=30, relheight=1, relwidth=0.2)
center_frame = Frame(main, bg=cf_bg)
center_frame.place(relx=0.2, y=30, relheight=1, relwidth=0.2)
right_frame = Frame(main, bg="Gray35")
right_frame.place(relx=0.4, y=30, relheight=1, relwidth=0.6)

# Placing components in the left frame
Label(left_frame, text="Name", font=labelfont, bg=lf_bg,
fg='white').place(relx=0.375, rely=0.05)
Label(left_frame, text="Contact Number", font=labelfont, bg=lf_bg,
fg='white').place(relx=0.175, rely=0.18)
Label(left_frame, text="Email Address", font=labelfont, bg=lf_bg,
fg='white').place(relx=0.2, rely=0.31)
Label(left_frame, text="Gender", font=labelfont, bg=lf_bg,
fg='white').place(relx=0.3, rely=0.44)
Label(left_frame, text="Date of Birth (DOB)", font=labelfont,
bg=lf_bg, fg='white').place(relx=0.1, rely=0.57)
Label(left_frame, text="Stream", font=labelfont, bg=lf_bg,
fg='white').place(relx=0.3, rely=0.7)

Entry(left_frame, width=19, textvariable=name_strvar,
font=entryfont).place(x=20, rely=0.1)
Entry(left_frame, width=19, textvariable=contact_strvar,
font=entryfont).place(x=20, rely=0.23)
Entry(left_frame, width=19, textvariable=email_strvar,
font=entryfont).place(x=20, rely=0.36)
Entry(left_frame, width=19, textvariable=stream_strvar,
font=entryfont).place(x=20, rely=0.75)

OptionMenu(left_frame, gender_strvar, 'Male', "Female").place(x=45,
rely=0.49, relwidth=0.5)
dob = DateEntry(left_frame, font=("Open Sans", 12), width=15)
dob.place(x=20, rely=0.62)

Button(left_frame, text='Submit and Add Record', font=labelfont,
command=add_record, width=18,
       bg=btn_bg, fg=btn_fg).place(relx=0.025, rely=0.85)

# Placing components in the center frame
Button(center_frame, text='Delete Record', font=labelfont,
command=remove_record, width=15,
```

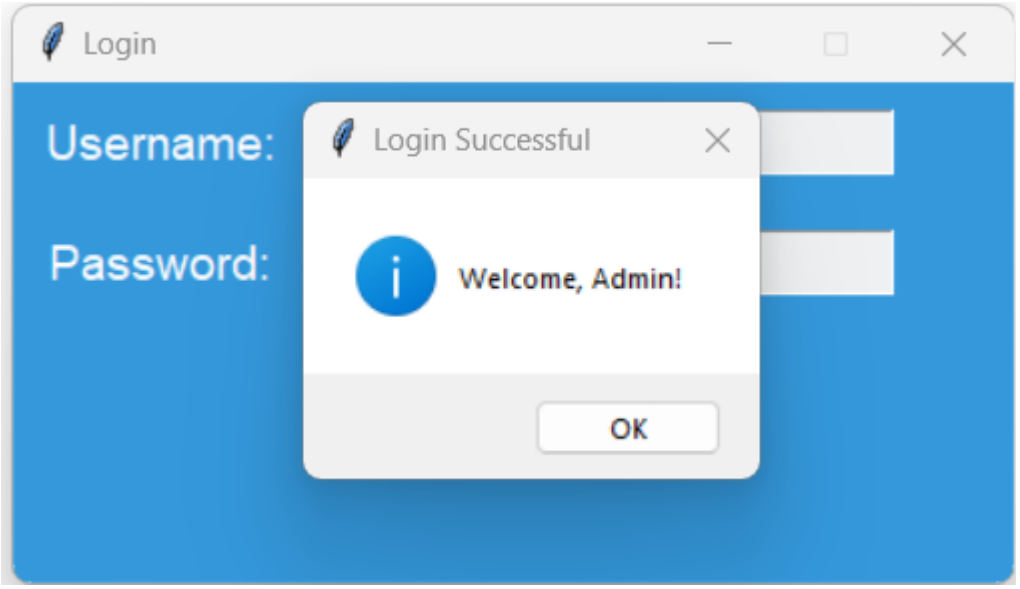
```
       bg=btn_bg, fg=btn_fg).place(relx=0.1, rely=0.25)

Button(center_frame, text='View Record', font=labelfont,
command=view_record, width=15,
       bg=btn_bg, fg=btn_fg).place(relx=0.1, rely=0.35)

Button(center_frame, text='Reset Fields', font=labelfont,
command=reset_fields, width=15,
       bg=btn_bg, fg=btn_fg).place(relx=0.1, rely=0.45)

Button(center_frame, text='Delete database', font=labelfont,
command=reset_form, width=15,
       bg=btn_bg, fg=btn_fg).place(relx=0.1, rely=0.55)

# Placing components in the right frame
Label(right_frame, text='Students Records', font=headlabelfont,
bg='#3498DB', fg='white').pack(side=TOP, fill=X)
tree = ttk.Treeview(right_frame, height=100, selectmode=BROWSE,
                    columns=('Student ID', "Name", "Email Address",
"Contact Number", "Gender", "Date of Birth", "Stream"))
X_scroller = Scrollbar(tree, orient=HORIZONTAL, command=tree.xview)
Y_scroller = Scrollbar(tree, orient=VERTICAL, command=tree.yview)
X_scroller.pack(side=BOTTOM, fill=X)
Y_scroller.pack(side=RIGHT, fill=Y)
tree.config(yscrollcommand=Y_scroller.set,
xscrollcommand=X_scroller.set)
tree.heading('Student ID', text='ID', anchor=CENTER)
tree.heading('Name', text='Name', anchor=CENTER)
tree.heading('Email Address', text='Email ID', anchor=CENTER)
tree.heading('Contact Number', text='Phone No', anchor=CENTER)
tree.heading('Gender', text='Gender', anchor=CENTER)
tree.heading('Date of Birth', text='DOB', anchor=CENTER)
tree.heading('Stream', text='Stream', anchor=CENTER)
tree.column('#0', width=0, stretch=NO)
tree.column('#1', width=40, stretch=NO)
tree.column('#2', width=140, stretch=NO)
tree.column('#3', width=200, stretch=NO)
tree.column('#4', width=80, stretch=NO)
tree.column('#5', width=80, stretch=NO)
tree.column('#6', width=80, stretch=NO)
tree.column('#7', width=150, stretch=NO)
tree.place(y=30, relwidth=1, relheight=0.9, relx=0)
display_records()

# Finalizing the GUI window
main.mainloop()
```

SCREENSHOTS

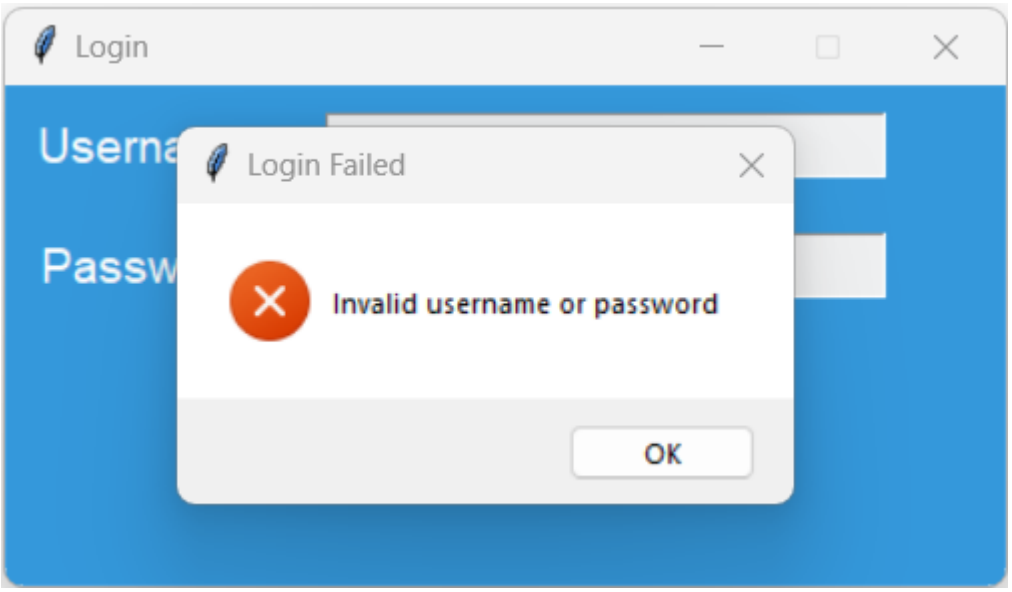
-Login window-



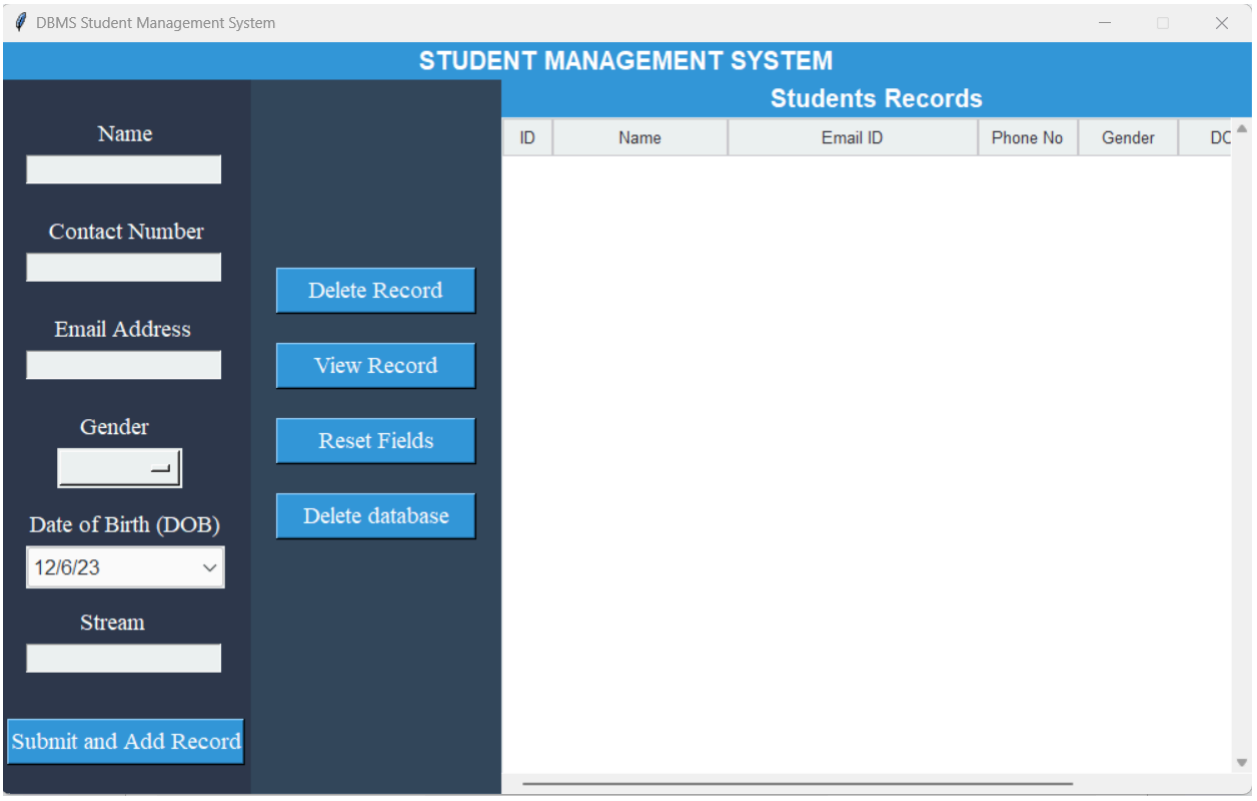
-Successful login-



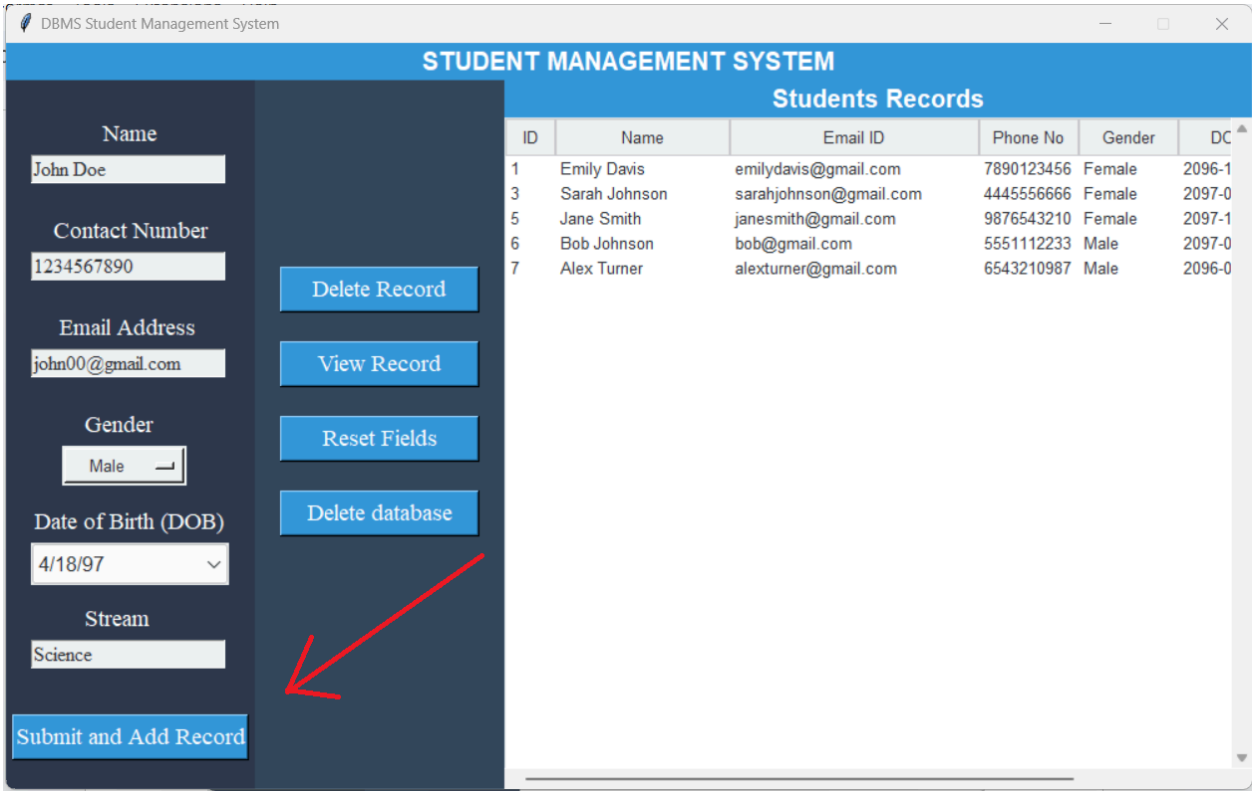
-Unsuccessful login-

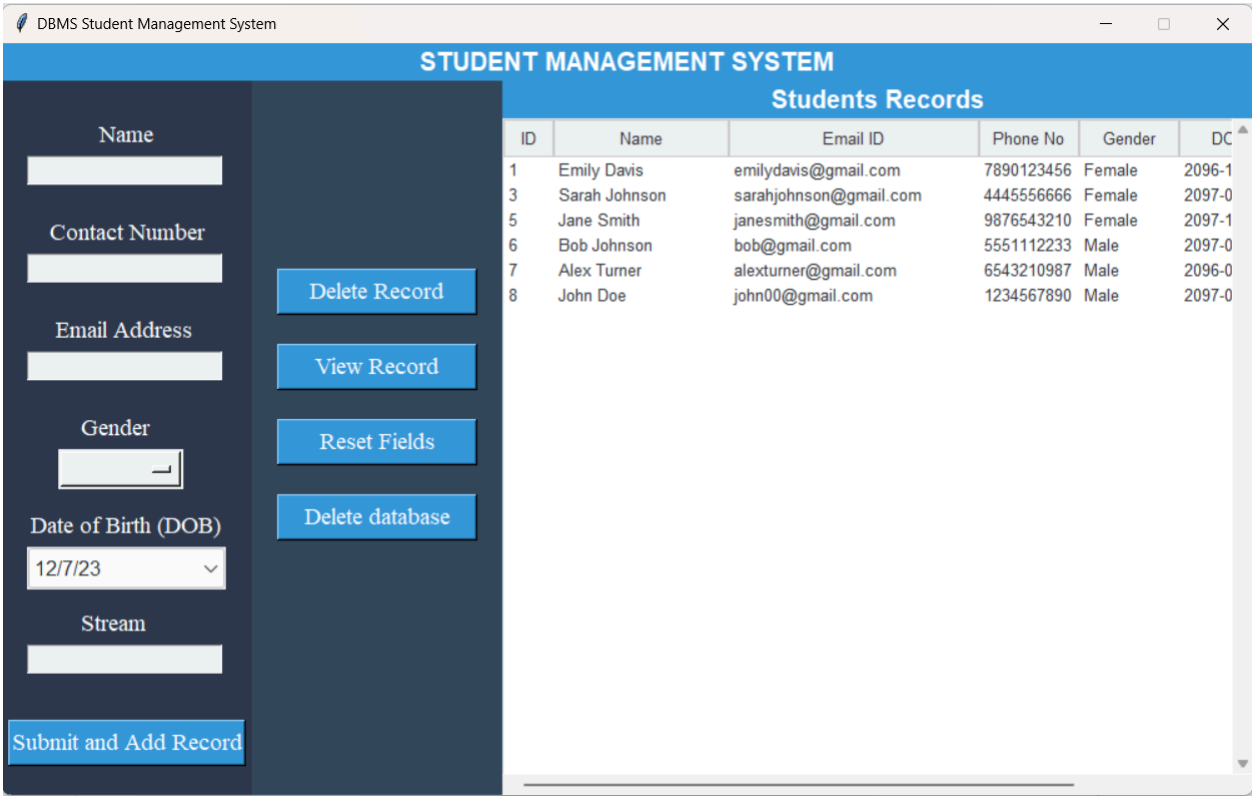


-Main window after successful login-

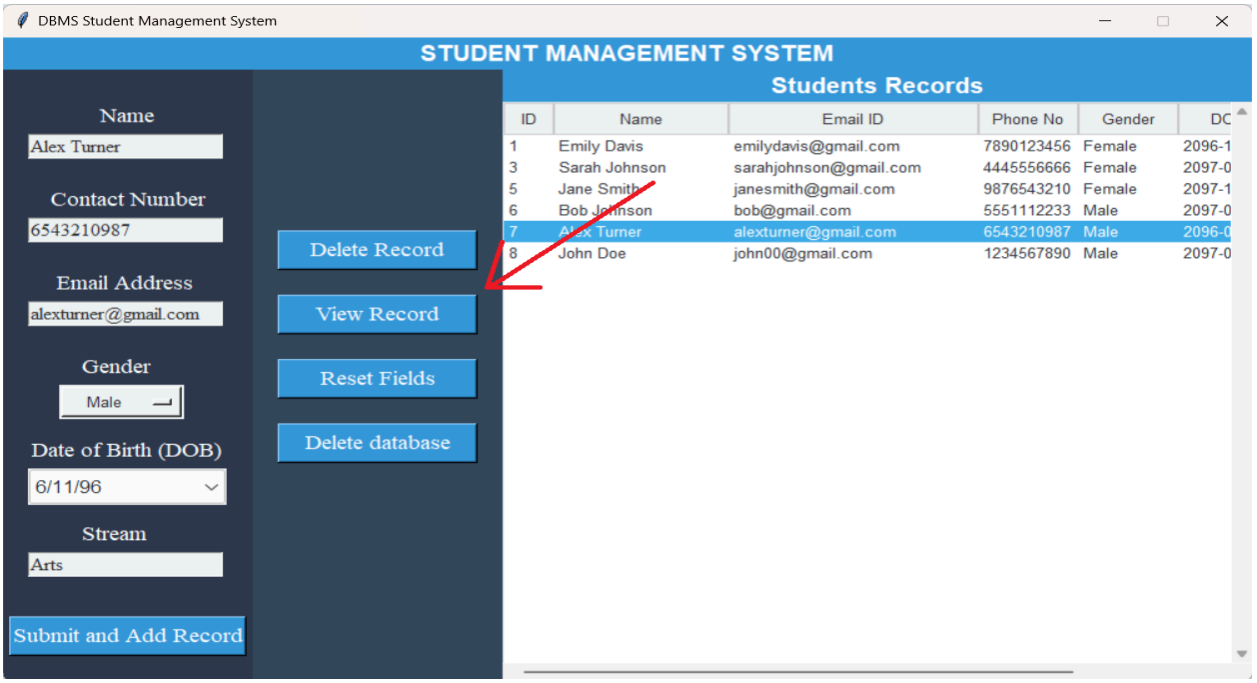


–Adding Student Records to the database–

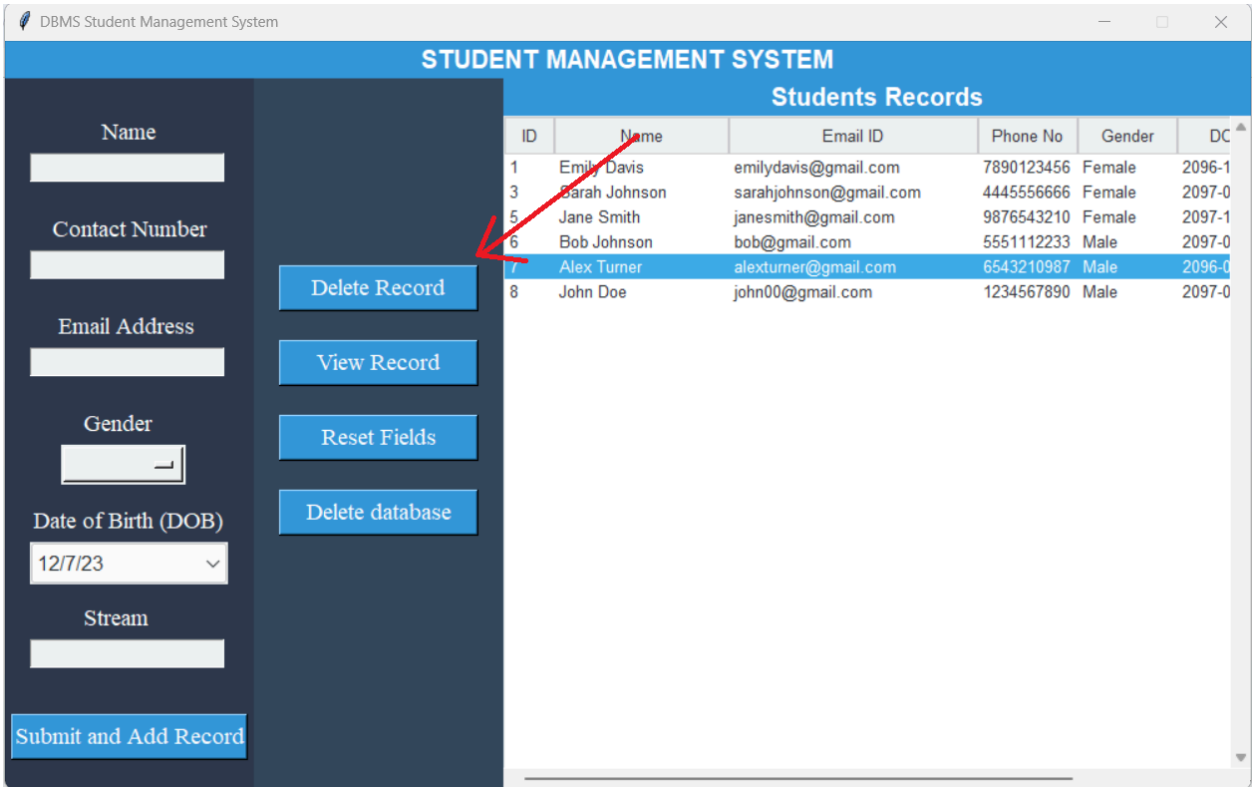


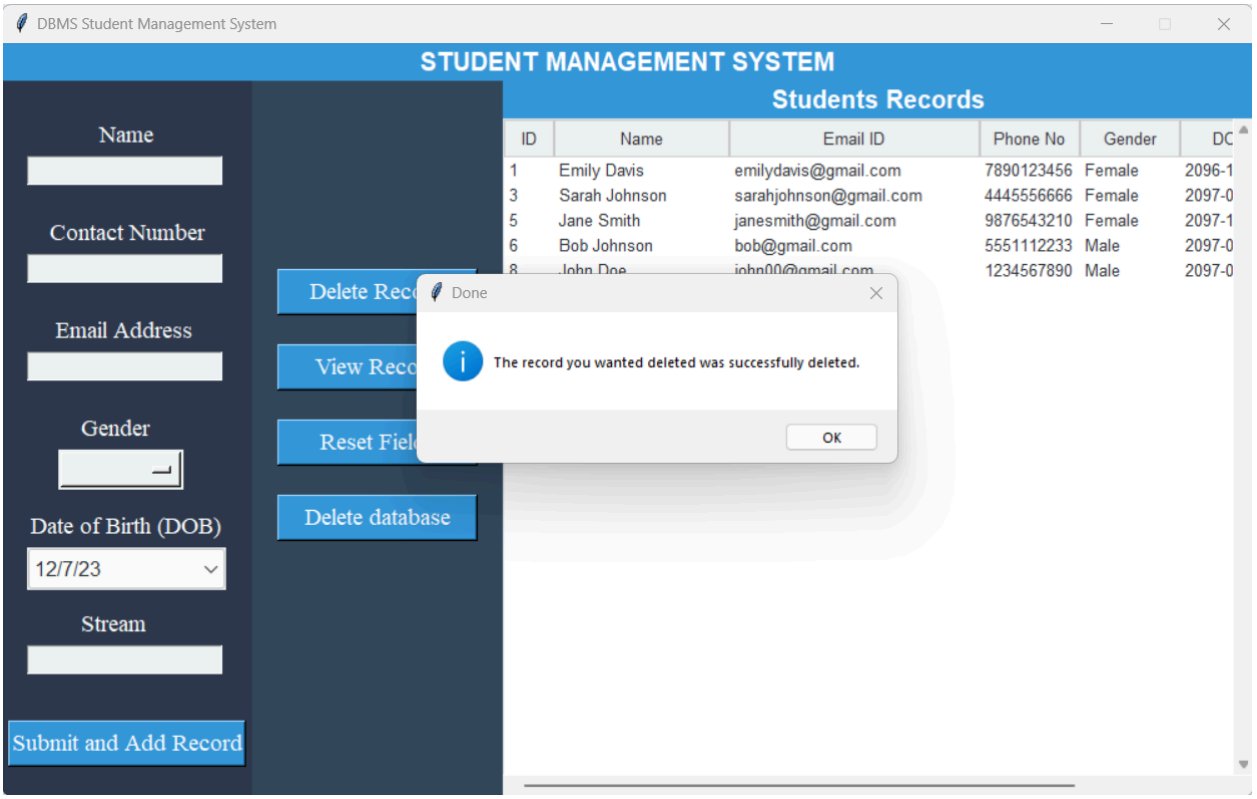


–Viewing a particular record–

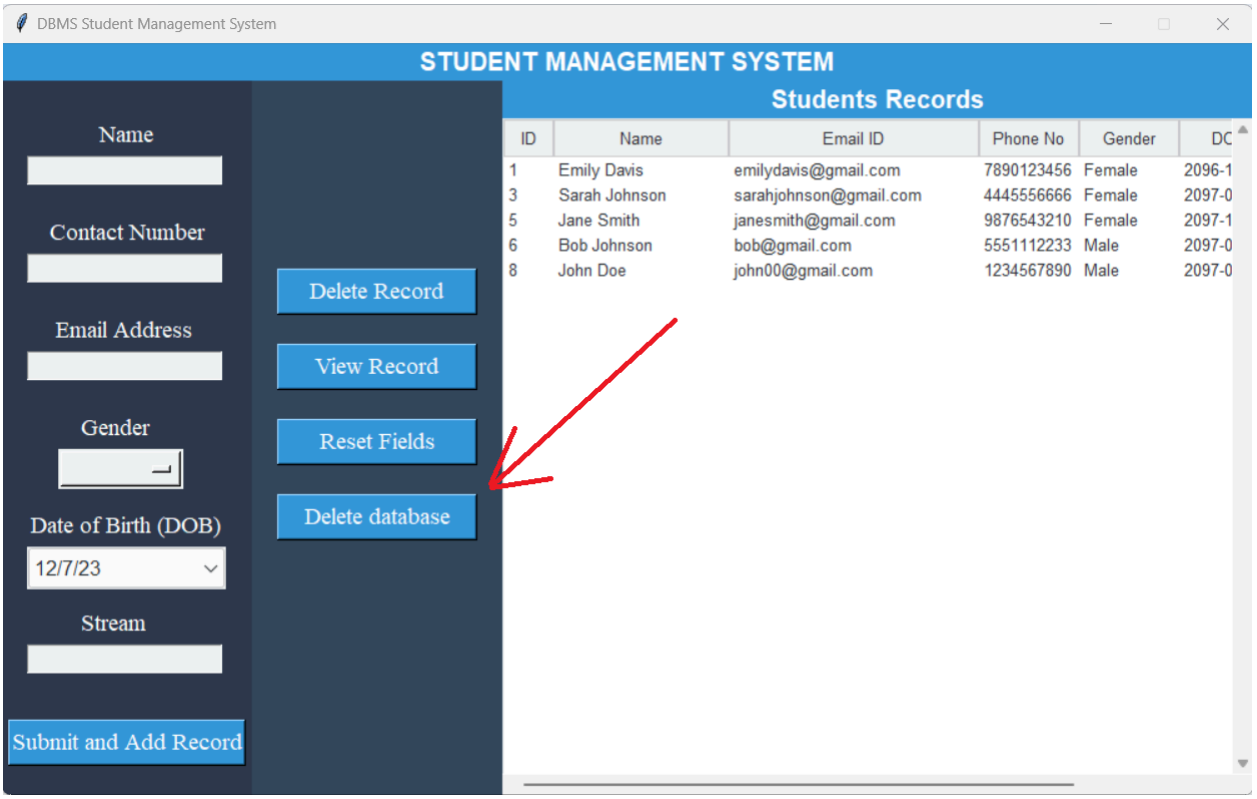


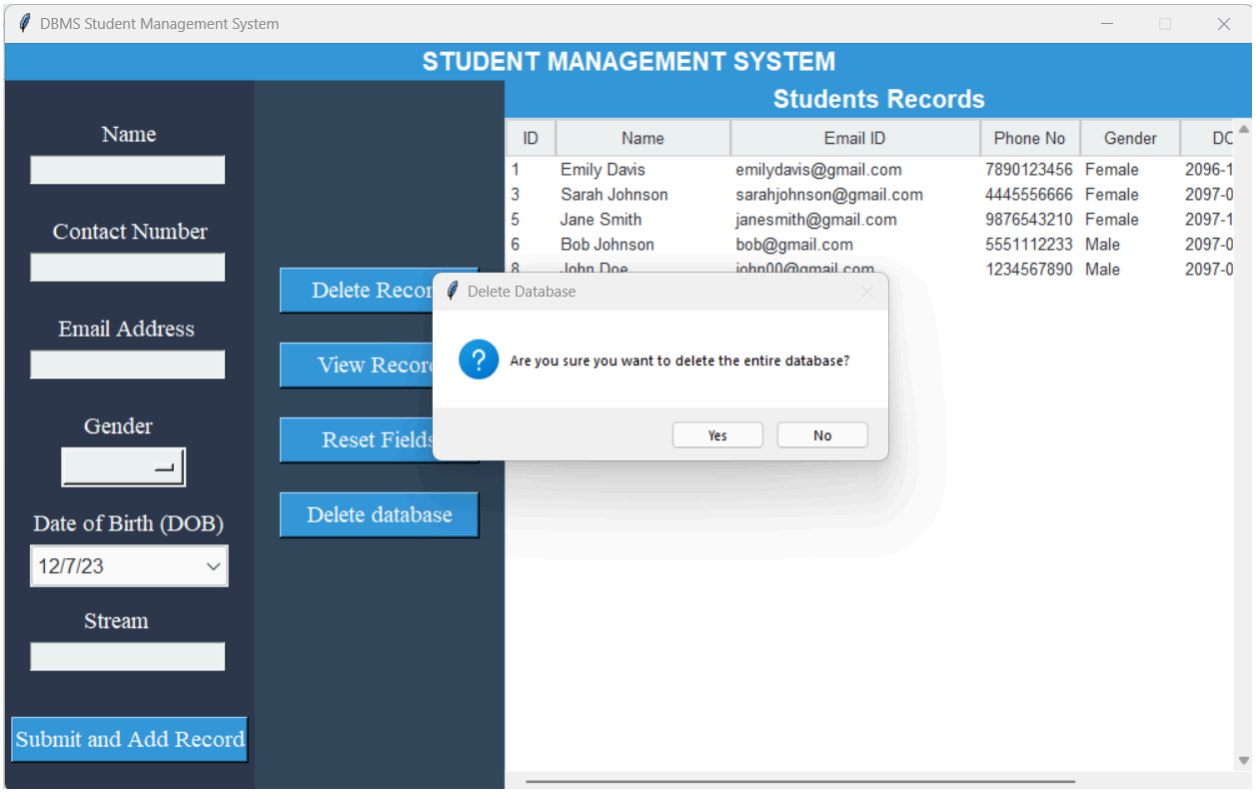
–Deleting a particular record–

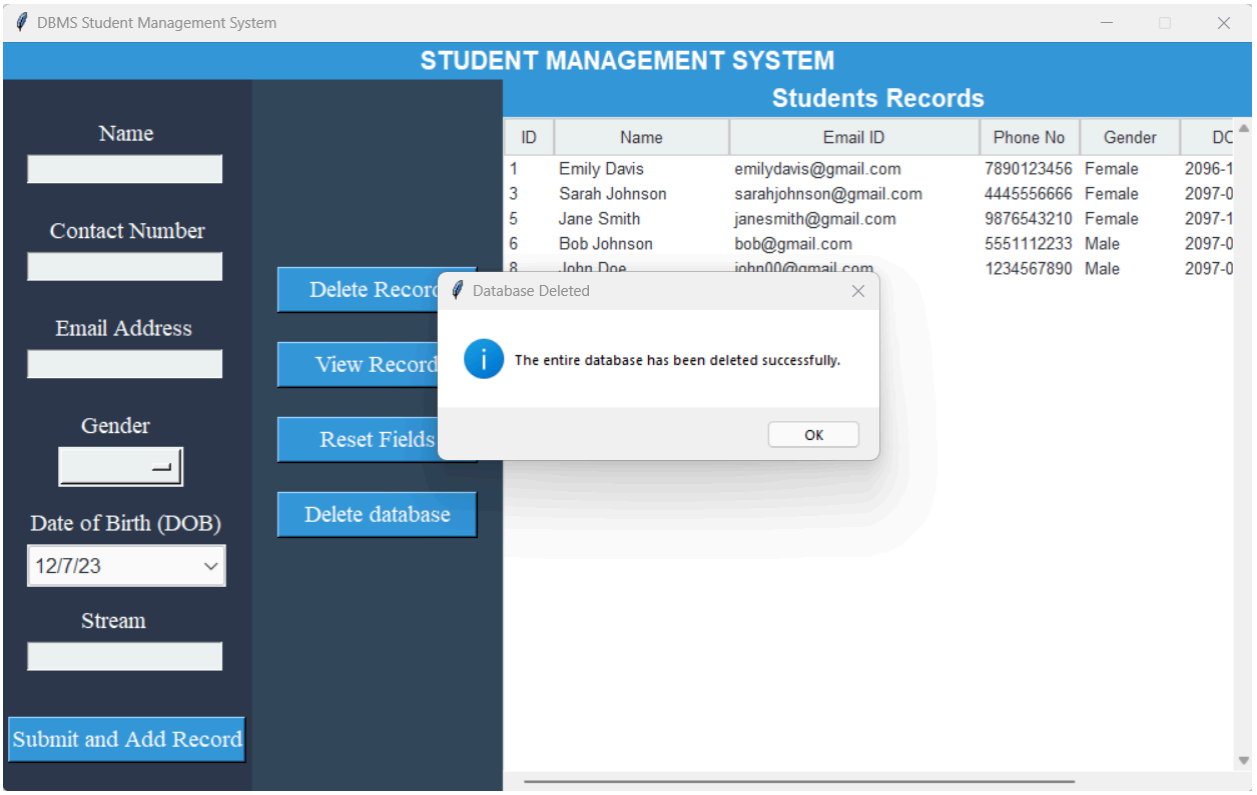




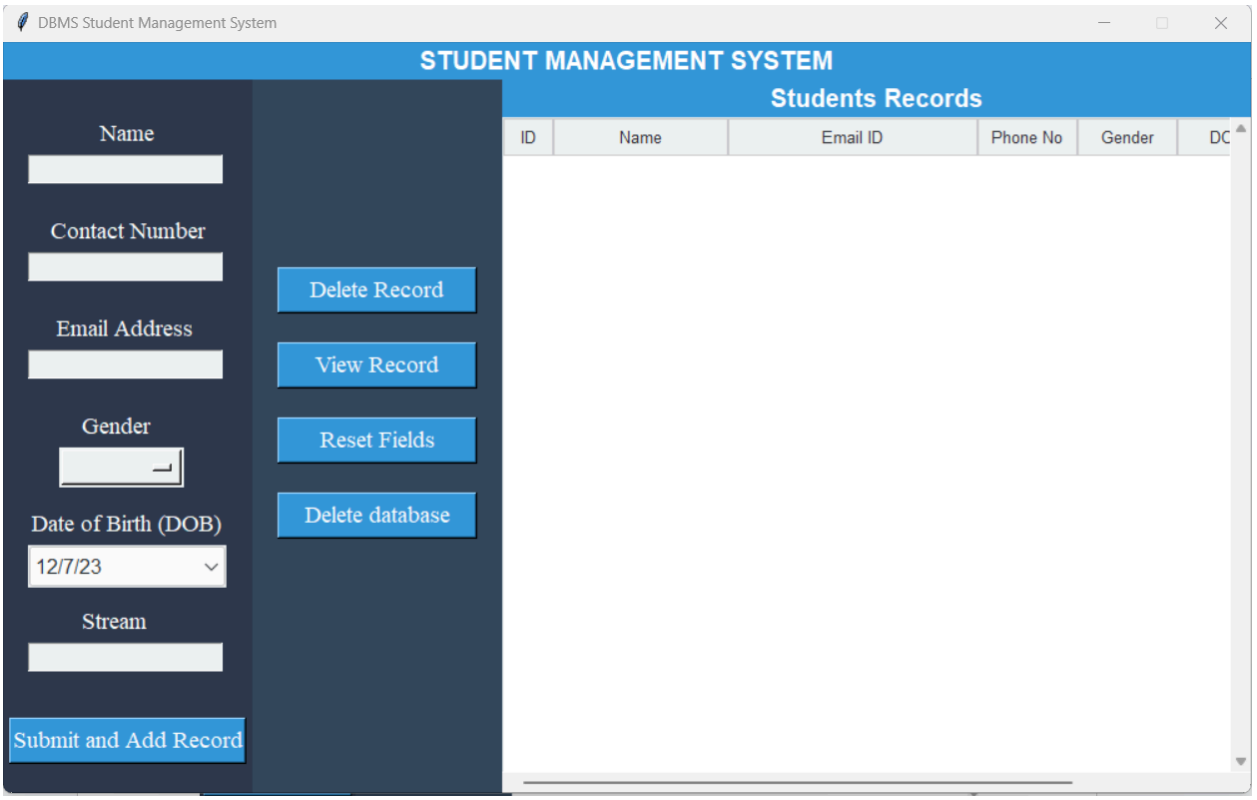
–Deleting the entire database–




DBMS Student Management System
STUDENT MANAGEMENT SYSTEM
Students Records
Name
Contact Number
Email Address
Gender
Date of Birth (DOB)
12/7/23
Stream
Submit and Add Record
Delete Record
View Record
Reset Fields
Delete database
Delete Database
Are you sure you want to delete the entire database?
Yes
No
ID
Name
Email ID
Phone No
Gender
DC
1 Emily Davis emilydavis@gmail.com 7890123456 Female 2096-1
3 Sarah Johnson sarahjohnson@gmail.com 4445556666 Female 2097-0
5 Jane Smith janesmith@gmail.com 9876543210 Female 2097-1
6 Bob Johnson bob@gmail.com 5551112233 Male 2097-0
1234567890 Male 2097-0


DBMS Student Management System
STUDENT MANAGEMENT SYSTEM
Students Records
Name
Contact Number
Email Address
Gender
Date of Birth (DOB)
12/7/23
Stream
Submit and Add Record
Delete Record
View Record
Reset Fields
Delete database
Database Deleted
The entire database has been deleted successfully.
OK
ID
Name
Email ID
Phone No
Gender
DC
1 Emily Davis emilydavis@gmail.com 7890123456 Female 2096-1
3 Sarah Johnson sarahjohnson@gmail.com 4445556666 Female 2097-0
5 Jane Smith janesmith@gmail.com 9876543210 Female 2097-1
6 Bob Johnson bob@gmail.com 5551112233 Male 2097-0
1234567890 Male 2097-0

–After successful deletion–



REFERENCES

1. Tkinter Documentation: [https://docs.python.org/3/library/tkinter.html]SQLite Documentation: [https://www.sqlite.org/docs.html]
2. Tkinter Calendar Widget (tkcalendar): [https://pypi.org/project/tkcalendar/]
3. ttkthemes Documentation: [https://ttkthemes.readthedocs.io/en/latest/]
4. Python Official Documentation: [https://docs.python.org/]
5. Stack Overflow: [https://stackoverflow.com/]

CONCLUSION

The Student Management System presented here offers an efficient and user-friendly solution for managing student records in an educational setting. Developed using Python with the Tkinter library for the graphical interface and SQLite for data storage, the system provides a secure and organized platform for administrators to handle student information.

The system's functionality encompasses key operations, including adding, viewing, and deleting student records, along with a secure login mechanism to control access. The use of themed GUI elements and the tkcalendar module enhances the user experience, providing a visually appealing and intuitive interface.

Additionally, the integration of error handling ensures data integrity, and the inclusion of non-functional requirements such as performance optimization, security measures, and cross-platform compatibility enhances the overall reliability and usability of the application.

The provided references showcase the use of established documentation and community resources that were instrumental in the development process. The structured database schema contributes to efficient data management, allowing for scalability and maintainability.

In conclusion, the Student Management System serves as a robust tool for educational institutions, offering a streamlined approach to student record management while prioritizing security, reliability, and user experience. Future updates and enhancements can build upon this foundation to meet evolving needs in the educational domain.