BLOOD DONATION MANAGEMENT SYSTEM

A MINI PROJECT REPORT

Submitted by

THARUN M **220701301**

THIRUGNANA SAMBANDA MOORHTI R **220701305**

in partial fulfillment of the award of the degree

of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE

An Autonomous Institute
THANDALAM
CHENNAI-602105

2023-2024

BONAFIDE CERTIFICATE

Certified that this Project report "**EVENT MANAGEMENT SYSTEM**" is the bonafide work of "**THARUN M (220701301)**" and "**THIRUGNANA SAMBANDA MOORTHI R (220701305)**" who carried out the project work under my supervision.

SIGNATURE

Dr.R.SABITHA
Professor and Academic Head,
Computer Science and Engineering,
Rajalakshmi Engineering College
(Autonomous),
Thandalam, Chennai - 602 105

SIGNATURE

Dr. G. DHARANI DEVI
Associate Professor,
Computer Science and Engineering,
Rajalakshmi Engineering College,
(Autonomous),
Thandalam, Chennai - 602 105

Submitted for the Practical Examination held on _____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ABSTRACT

The Event Management System is designed using Python to streamline and simplify the process of event registration and management within educational institutions. Leveraging CustomTkinter (CTkinter) for the graphical user interface, the system offers a user-friendly and intuitive experience for both administrators and students. This interface ensures ease of use, minimizing the learning curve and enabling users to quickly navigate and utilize the system's features.

Key functionalities include the ability to register new users, manage events comprehensively, and track student enrollments effectively. The backend infrastructure is powered by a MySQL database, chosen for its robust data management capabilities and efficient data retrieval processes. This choice ensures that the system can handle a large volume of data while maintaining high performance and reliability.

The Event Management System significantly enhances overall operational efficiency by automating various aspects of event management. It reduces the manual workload traditionally associated with these tasks, thereby freeing up valuable administrative resources. Additionally, the system provides real-time access to event information, ensuring that students and administrators are always informed about the latest updates and changes. This immediate access to information fosters better communication and coordination, leading to more successful and well-organized events. Overall, the system represents a significant advancement in the way educational institutions handle event management, promoting a more streamlined and efficient process.

TABLE OF CONTENTS

CHAPTER 1

1. INTRODUCTION

1.1 INTRODUCTION

The event management system is designed to automate and manage the various tasks involved in organizing and tracking events within an educational institution. Users can register new accounts, log in to the system, and navigate through a user-friendly interface to manage events and enrollments. The system, developed using Python and featuring a CustomTkinter (CTkinter) interface, allows staff to efficiently handle event details and student participation with minimal technical expertise. By leveraging a MySQL database for its backend, the system ensures data integrity and efficient retrieval, storing comprehensive information about users, events, and enrollments. This centralized data management solution automates repetitive tasks, enhances overall efficiency, and provides real-time access to event information.

1.2 OBJECTIVES

The main objective of the Event Management System is to manage the details of users, events, and enrollments. The system aims to:

- Facilitate user registration and authentication.
- Allow administrators to add, view, update, and delete event details.
- Enable students to view and enroll in events.
- Maintain a robust database for storing user and event information.

1.3 MODULES

- User Registration
- User Login
- Add Event
- View Events
- Update Event Details
- Delete Event
- View Enrollment Details

2. SURVEY OF TECHNOLOGIES

2.1 SOFTWARE DESCRIPTION

Visual Studio Code:

Visual Studio Code combines the simplicity of a source code editor with powerful developer tooling, such as IntelliSense code completion and debugging. It provides a seamless edit-build-debug cycle, allowing developers to focus on executing their ideas.

2.2 LANGUAGES

2.2.1 Python

Python: A high-level, interpreted programming language known for its readability and versatility.

CustomTkinter (CTkinter): A modern GUI (Graphical User Interface) library in Python that extends Tkinter with additional styling and functionality options. This library provides tools for creating visually appealing and highly interactive interfaces.

2.2.2 SQL

SQL: A standard language for managing and querying relational databases. SQL is widely used by organizations for its performance, scalability, reliability, and ease of use.

3. REQUIREMENTS AND ANALYSIS

3.1 REQUIREMENTS SPECIFICATION

User Requirements:

- The system should allow users to register, log in, and manage event details.
- Administrators should be able to add, view, update, and delete events.
- Students should be able to view and enroll in events.

System Requirements:

- A database backup system should be in place.
- The operating system should be Windows XP or a higher version of Windows.

3.2 HARDWARE AND SOFTWARE REQUIREMENTS

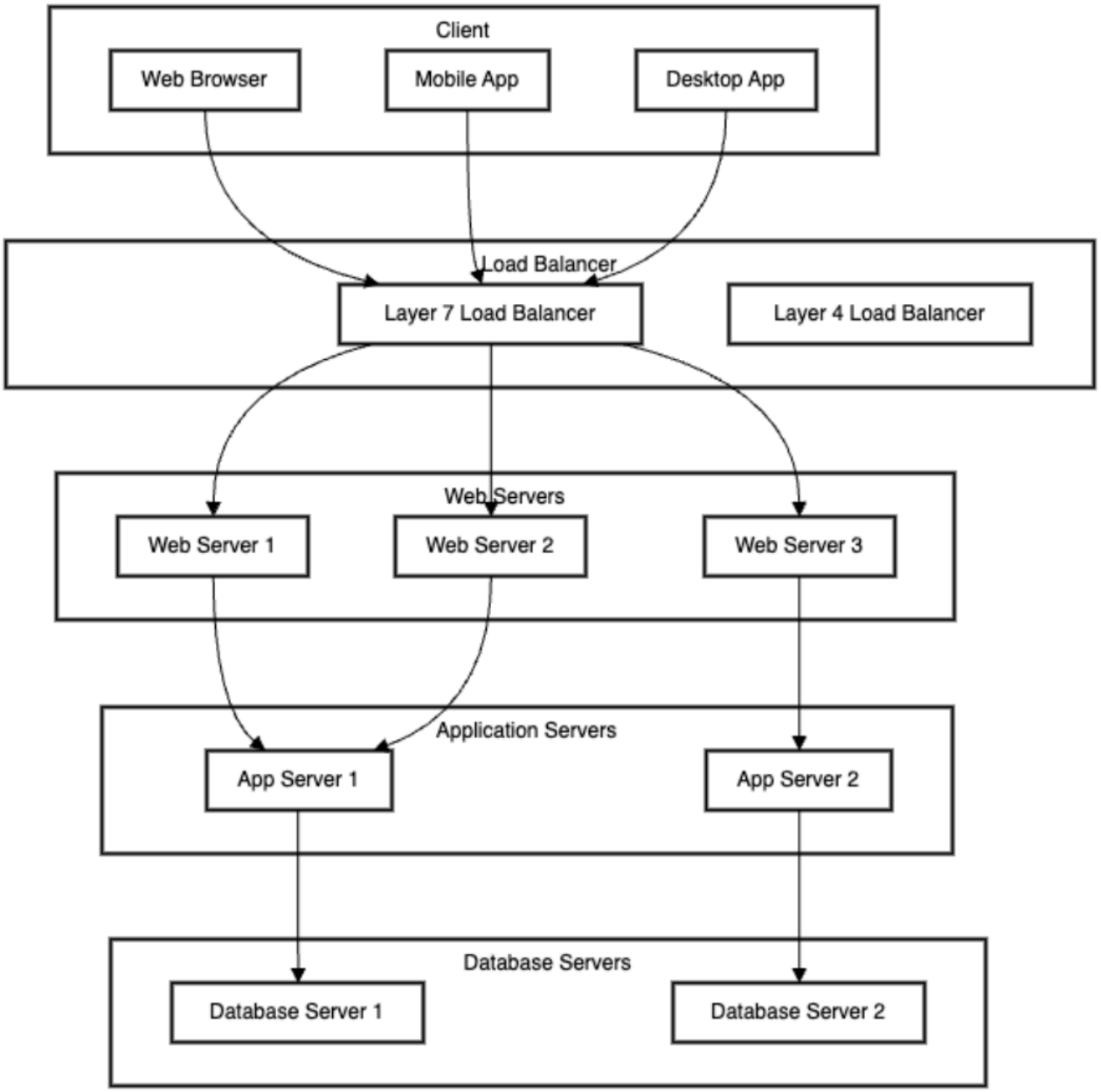
Software Requirements:

- Operating System: Windows 10
- Front End: Python (CustomTkinter)
- Back End: MySQL

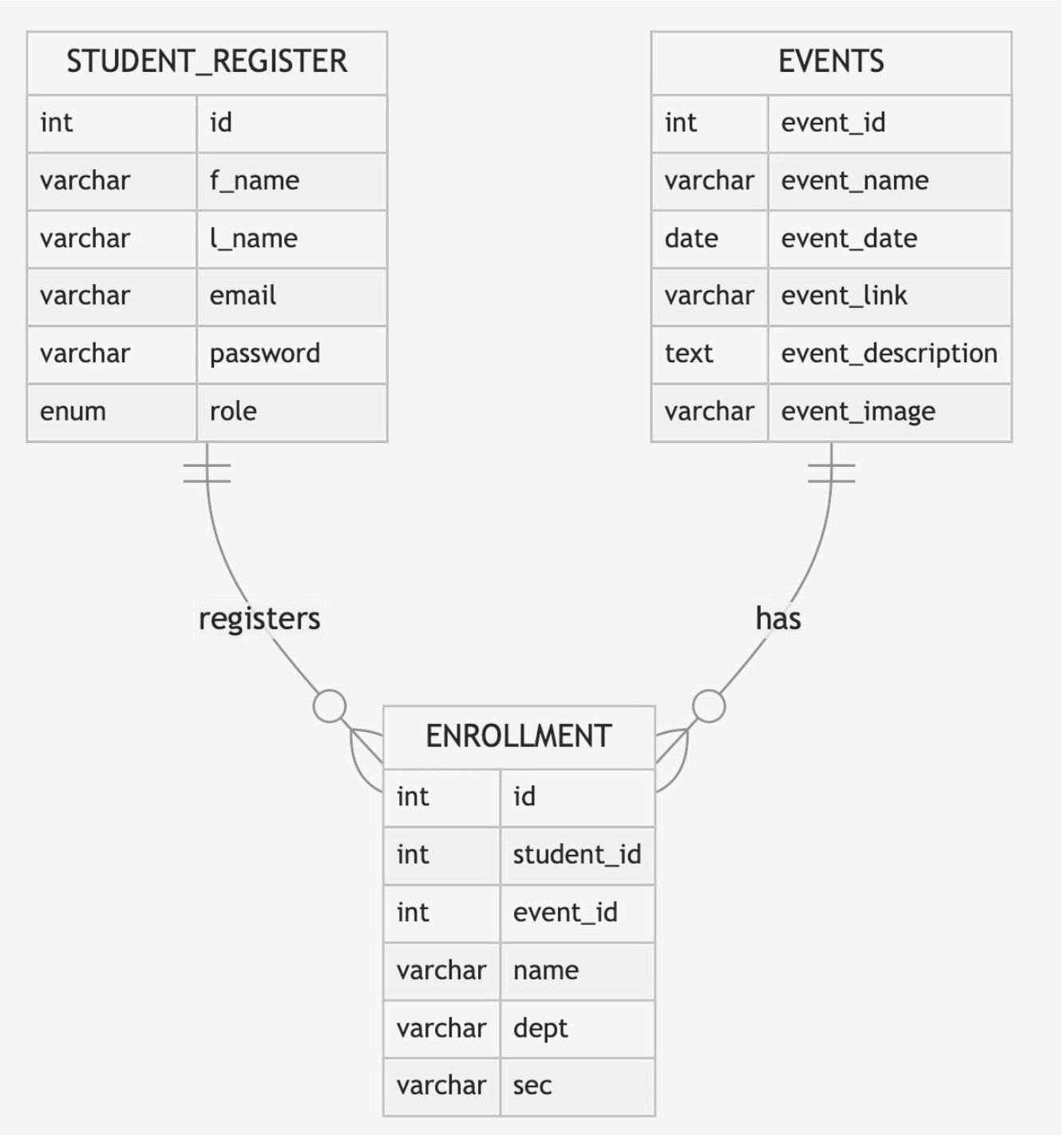
Hardware Requirements:

- Desktop PC or Laptop
- Printer
- minimum Intel® Core™ i3-6006U CPU @ 2.00GHz
- 4.00 GB RAM
- 64-bit operating system, x64-based processor
- 1024 x 768 monitor resolution
- Keyboard and Mouse

3.3 Architecture Diagram



3.4. ER DIAGRAM



3.5 NORMALIZATION

Table: student_register

email	f_name	l_name	password	role
anand@admin.rec.in	anand	k	anand	admin
thiru@student.rec.in	thiru	r	1234	student

Table: events

event_name	event_date	event_link	event_ description	event_image

Table: enrollments

id	event_name	student_email	dept	sec

2nf tables:

Table: student_register

email	f_name	l_name	password	role
anand@admin.rec.in	anand	k	anand	admin
thiru@student.rec.in	thiru	r	1234	student

Table: events

event_name	event_date	event_link	event_ description	event_image
hackathon	2024-05-25	https://www.google.com/	hack	image
techhack	2024-06-10	https://web.whatsapp.com/	tech	image

Table: enrollments

id	event_name	student_email	dept	sec
301	hackathon	thiru@student.rec.in	cse	e
305	techhack	thiru@student.rec.in	cse	e

4.PROGRAM CODE

1. SOURCE CODE:

SQL CODE:

```
-- Create the database
CREATE DATABASE IF NOT EXISTS event_management;
USE event_management;

-- Create the student_register table
CREATE TABLE IF NOT EXISTS student_register (
    id INT AUTO_INCREMENT PRIMARY KEY,
    f_name VARCHAR(255) NOT NULL,
    l_name VARCHAR(255) NOT NULL,
    email VARCHAR(255) NOT NULL UNIQUE,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'student') NOT NULL
);

-- Create the events table
CREATE TABLE IF NOT EXISTS events (
    event_id INT AUTO_INCREMENT PRIMARY KEY,
    event_name VARCHAR(255) NOT NULL,
    event_date DATE NOT NULL,
    event_link VARCHAR(255) NOT NULL,
    event_description TEXT NOT NULL,
    event_image VARCHAR(255) NOT NULL
);

-- Function to create dynamic enrollment tables
DELIMITER //
CREATE PROCEDURE create_enrollment_table(IN event_name VARCHAR(255))
BEGIN
    SET @query = CONCAT('CREATE TABLE IF NOT EXISTS enroll_', REPLACE(event_name, ' ',
'_'), ' (
        id INT AUTO_INCREMENT PRIMARY KEY,
        name VARCHAR(255) NOT NULL,
        dept VARCHAR(255) NOT NULL,
        sec VARCHAR(255) NOT NULL
    )');
    PREPARE stmt FROM @query;
    EXECUTE stmt;
    DEALLOCATE PREPARE stmt;
END //
DELIMITER ;
```

```
-- Sample data insertion
INSERT INTO student_register (f_name, l_name, email, password, role) VALUES
('Admin', 'User', 'admin@example.com', 'admin123', 'admin'),
('John', 'Doe', 'john.doe@example.com', 'password123', 'student'),
('Jane', 'Doe', 'jane.doe@example.com', 'password123', 'student');

INSERT INTO events (event_name, event_date, event_link, event_description, event_image) VALUES
('Tech Talk', '2024-06-20', 'https://example.com/tech-talk', 'An engaging tech talk on AI.',
'/path/to/image1.png'),
('Coding Marathon', '2024-06-25', 'https://example.com/coding-marathon', 'A 24-hour coding event.',
'/path/to/image2.png');
```

PYTHON CODE:

```
import customtkinter as ctk
from tkinter import ttk,messagebox, PhotoImage, Toplevel, Listbox, END, Canvas
import mysql.connector
from PIL import ImageTk, Image
from db import connect_db;
# Database connection
def register():
    firstname = firstname_entry.get()
    lastname = lastname_entry.get()
    email = email_entry.get()
    password = password_entry.get()

    if not firstname or not lastname or not email or not password:
        messagebox.showerror("Error", "Please fill in all fields")
        return
    role = "admin" if "admin" in email else "student"
    try:
        db = connect_db()
        cursor = db.cursor()
        query = "INSERT INTO student_register(f_name, l_name, email, password, role) VALUES (%s,
%s, %s, %s, %s)"
        cursor.execute(query, (firstname, lastname, email, password, role))
        db.commit()
        messagebox.showinfo("Success", "Registration Successful!")
        open_login_page()
    except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Error: {err}")
    finally:
        if 'db' in locals():
            db.close()
```

```
# Function to open the login page
def open_login_page():
    signup_window.destroy()
    login_window = ctk.CTk()
    login_window.title("Login Page")
    login_window.geometry("1000x1000")
    bg_image = Image.open("rec.jpg")
    bg_photo = ImageTk.PhotoImage(bg_image)
    bg_label = ctk.CTkLabel(login_window, image=bg_photo)
    bg_label.place(relx=0.5, rely=0.5, anchor='center')
    form_frame1 = ctk.CTkFrame(login_window)
    form_frame1.place(relx=0.5, rely=0.5, anchor='center')
    email_label = ctk.CTkLabel(form_frame1, text="Email:")
    email_label.grid(row=0, column=0, padx=10, pady=5)
    email_entry = ctk.CTkEntry(form_frame1)
    email_entry.grid(row=0, column=1, padx=10, pady=5)
    password_label = ctk.CTkLabel(form_frame1, text="password:")
    password_label.grid(row=1, column=0, padx=10, pady=5)
    password_entry = ctk.CTkEntry(form_frame1,show='*')
    password_entry.grid(row=1, column=1, padx=10, pady=5)

    def login():
        email = email_entry.get()
        password = password_entry.get()
        try:
            db = connect_db()
            cursor = db.cursor()
            query = "SELECT role FROM student_register WHERE email = %s AND password = %s"
            cursor.execute(query, (email, password))
            user = cursor.fetchone()
            if user:
                role = user[0]
                login_window.destroy()
                if role == 'admin':
                    open_admin_page()
                else:
                    open_student_page()
            else:
                messagebox.showerror("Login Failed", "Invalid email or password")
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Error: {err}")
        finally:
            if 'db' in locals():
                db.close()
    login_button = ctk.CTkButton(form_frame1, text="Login", command=login)
```

```
    login_button.grid(row=2, column=1, padx=10, pady=5)
    login_window.mainloop()
# Function to open the admin page
def open_admin_page():
    admin_window = ctk.CTk()
    admin_window.title("Admin Page")
    admin_window.geometry("1000x1000")
    bg_image4 = Image.open("event1.png")
    bg_photo4 = ImageTk.PhotoImage(bg_image4)
    bg_label4 = ctk.CTkLabel(admin_window, image=bg_photo4)
    bg_label4.place(relx=0.5, rely=0.5, anchor='center')
    form_frame3 = ctk.CTkFrame(admin_window)
    form_frame3.place(relx=0.5, rely=0.5, anchor='center')

    ctk.CTkLabel(form_frame3, text="Event Name:").grid(row=0, column=0, padx=10, pady=5)
    event_name_entry = ctk.CTkEntry(form_frame3)
    event_name_entry.grid(row=0, column=1, padx=10, pady=5)
    ctk.CTkLabel(form_frame3, text="Event Date (YYYY-MM-DD):").grid(row=1, column=0,
padx=10, pady=5)
    event_date_entry = ctk.CTkEntry(form_frame3)
    event_date_entry.grid(row=1, column=1, padx=10, pady=5)
    ctk.CTkLabel(form_frame3, text="Event Link:").grid(row=2, column=0, padx=10, pady=5)
    event_link_entry = ctk.CTkEntry(form_frame3)
    event_link_entry.grid(row=2, column=1, padx=10, pady=5)
    ctk.CTkLabel(form_frame3, text="Event Description:").grid(row=3, column=0, padx=10, pady=5)
    event_description_entry = ctk.CTkEntry(form_frame3)
    event_description_entry.grid(row=3, column=1, padx=10, pady=5)
    ctk.CTkLabel(form_frame3, text="Event Image Path:").grid(row=4, column=0, padx=10, pady=5)
    event_image_entry = ctk.CTkEntry(form_frame3)
    event_image_entry.grid(row=4, column=1, padx=10, pady=5)
    events_listbox = Listbox(form_frame3)
    events_listbox.grid(row=9, column=0, columnspan=2, padx=10, pady=5)
    def add_event():
        event_name = event_name_entry.get()
        event_date = event_date_entry.get()
        event_link = event_link_entry.get()
        event_description = event_description_entry.get()
        event_image = event_image_entry.get()
        if not event_name or not event_date or not event_link or not event_description or not
event_image:
            messagebox.showerror("Error", "Please fill in all fields")
            return
        try:
            db = connect_db()
            cursor = db.cursor()
```

```
        query = "INSERT INTO events (event_name, event_date, event_link, event_description,
event_image) VALUES (%s, %s, %s, %s, %s)"
        cursor.execute(query, (event_name, event_date, event_link, event_description, event_image))
        db.commit()
        messagebox.showinfo("Success", "Event Added Successfully!")
        display_events()
      except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Error: {err}")
      finally:
        if 'db' in locals():
          db.close()
  def display_events():
      try:
        db = connect_db()
        cursor = db.cursor()
        query = "SELECT event_name, event_date FROM events ORDER BY event_date"
        cursor.execute(query)
        events = cursor.fetchall()
        events_listbox.delete(0, END)
        for event in events:
          event_name, event_date = event
          events_listbox.insert(END, f"{event_date} - {event_name}")
        events_listbox.bind('<<ListboxSelect>>', lambda e: show_enrollment_details(e))
      except mysql.connector.Error as err:
        messagebox.showerror("Error", f"Error: {err}")
      finally:
        if 'db' in locals():
          db.close()
  def show_enrollment_details(event):
      event_register_details = ctk.CTk()

      event_register_details.geometry("1200x800")
      def load_student_details():
        selection = events_listbox.curselection()
        if not selection:
          return
        selected_event = events_listbox.get(selection[0])
        event_name = selected_event.split(" - ")[1]
        enroll_table = f"enroll_{event_name.replace(' ', '_')}"
        event_register_details.title(f"{event_name} register details")
        try:
          db = connect_db()
          cursor = db.cursor()
          query = f"SELECT name, dept, sec FROM {enroll_table}"
          cursor.execute(query)
```

```
            students = cursor.fetchall()
            for student in students:
                student_treeview.insert("", "end", values=student)
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Error: {err}")
        finally:
            if 'db' in locals():
                db.close()
    ctk.CTkLabel(event_register_details, text="Student Details").pack(pady=10)

    columns = ("Name", "dept", "sec")
    student_treeview = ttk.Treeview(event_register_details, columns=columns, show="headings")
    for col in columns:
        student_treeview.heading(col, text=col)
        student_treeview.column(col, width=200)
    student_treeview.pack(pady=20, fill="both", expand=True)
    load_student_details()

  display_events()
  def event_entered_details():
    student_window = ctk.CTk()
    student_window.title("Student Page")
    student_window.geometry("1200x800")
    def load_events():
        try:
            db = connect_db()
            cursor = db.cursor()
            query = "SELECT event_name, event_date, event_link, event_description, event_image
FROM events ORDER BY event_date"
            cursor.execute(query)
            events = cursor.fetchall()
            for event in events:
                event_treeview.insert("", "end", values=event)
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Error: {err}")
        finally:
            if 'db' in locals():
                db.close()
    ctk.CTkLabel(student_window, text="Events").pack(pady=10)

    columns = ("Event Name", "Event Date", "Event Link", "Event Description", "Event Image")
    event_treeview = ttk.Treeview(student_window, columns=columns, show="headings")
    for col in columns:
        event_treeview.heading(col, text=col)
        event_treeview.column(col, width=200)
```

```
        event_treeview.pack(pady=20, fill="both", expand=True)
        load_events()
        student_window.mainloop()
    def register_details():
        admin_window = ctk.CTk()
        admin_window.title("Admin Page")
        admin_window.geometry("1200x800")
        def load_student_details():
            try:
                db = connect_db()
                cursor = db.cursor()
                query = "SELECT f_name, l_name, email, role FROM student_register"
                cursor.execute(query)
                students = cursor.fetchall()
                for student in students:
                    student_treeview.insert("", "end", values=student)
            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Error: {err}")
            finally:
                if 'db' in locals():
                    db.close()
        ctk.CTkLabel(admin_window, text="Student Details").pack(pady=10)

        columns = ("First Name", "Last Name", "Email", "Role")
        student_treeview = ttk.Treeview(admin_window, columns=columns, show="headings")
        for col in columns:
            student_treeview.heading(col, text=col)
            student_treeview.column(col, width=200)
        student_treeview.pack(pady=20, fill="both", expand=True)
        load_student_details()
        admin_window.mainloop()
    add_event_button = ctk.CTkButton(form_frame3, text="Add Event", command=add_event)
    add_event_button.grid(row=5, column=0, columnspan=2, pady=10)
    add_event_button = ctk.CTkButton(form_frame3, text="Event details",
command=event_entered_details)
    add_event_button.grid(row=6, column=0, columnspan=2, pady=10)

    add_event_button = ctk.CTkButton(form_frame3, text="Student details ",
command=register_details)
    add_event_button.grid(row=7, column=0, columnspan=2, pady=10)
    admin_window.mainloop()
# Function to open the student page
def open_student_page():
    student_window = ctk.CTk()
    student_window.title("Student Page")
```

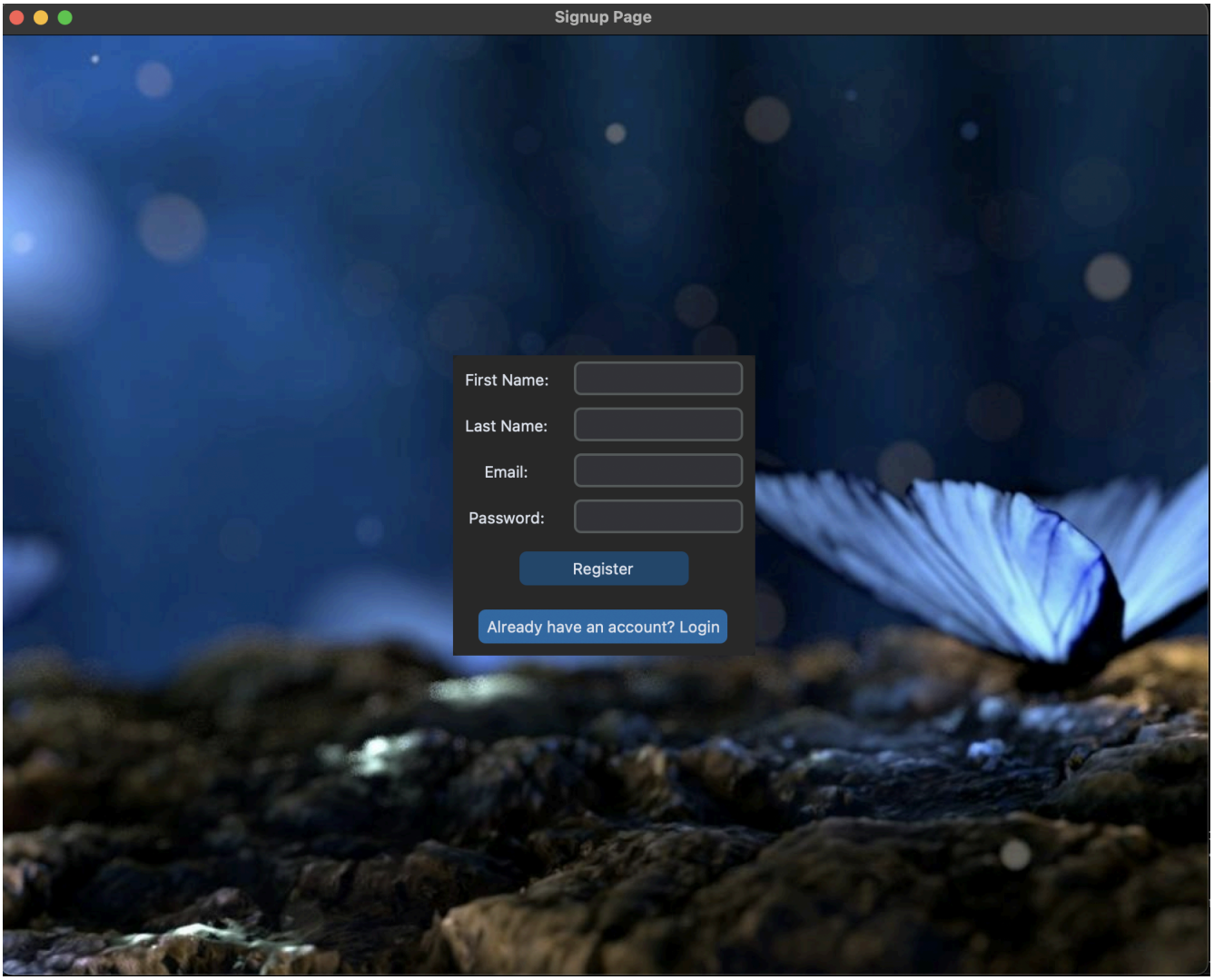
```
    student_window.geometry("1000x1000")
    bg_image = Image.open("event1.png")
    bg_photo = ImageTk.PhotoImage(bg_image)
    bg_label = ctk.CTkLabel(student_window,text="", image=bg_photo)
    bg_label.place(relx=0.5, rely=0.5, anchor='center')
    events_listbox = Listbox(student_window,height=30,width=50)
    events_listbox.place(relx=0.8,rely=0.5,anchor="center")
    def display_events():
        try:
            db = connect_db()
            cursor = db.cursor()
            query = "SELECT event_name, event_date FROM events WHERE event_date >= CURDATE()
ORDER BY event_date"
            cursor.execute(query)
            events = cursor.fetchall()
            events_listbox.delete(0, END)
            for event in events:
                event_name, event_date = event
                events_listbox.insert(END, f"{event_date} - {event_name}")
            events_listbox.bind('<<ListboxSelect>>', lambda e: show_event_details(e))
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Error: {err}")
        finally:
            if 'db' in locals():
                db.close()
    def show_event_details(event):
        selection = events_listbox.curselection()
        if not selection:
            return
        selected_event = events_listbox.get(selection[0])
        event_name = selected_event.split(" - ")[1]
        try:
            db = connect_db()
            cursor = db.cursor()
            query = "SELECT event_name, event_date, event_link, event_description, event_image FROM
events WHERE event_name = %s"
            cursor.execute(query, (event_name,))
            event = cursor.fetchone()
            if event:
                event_name, event_date, event_link, event_description, event_image = event
                details_window = Toplevel(student_window)
                details_window.title(f"Details for {event_name}")
                details_window.geometry("1000x1000")
                ctk.CTkLabel(details_window, text=f"Event Name: {event_name}").pack(pady=5)
                ctk.CTkLabel(details_window, text=f"Event Date: {event_date}").pack(pady=5)
```

```
        ctk.CTkLabel(details_window, text=f"Event Link: {event_link}").pack(pady=5)
        ctk.CTkLabel(details_window, text=f"Event Description:
{event_description}").pack(pady=5)
        try:
            img = Image.open("enroll_3.png")
            img = img.resize((200, 200), Image.ANTIALIAS)
            photo = ImageTk.PhotoImage(img)
            label = ctk.CTkLabel(details_window, image=photo)
            label.image = photo  # Keep a reference to avoid garbage collection
            label.pack(pady=5)
        except Exception as e:
            print(f"Could not open image: {e}")
        def enroll():
            name = name_entry.get()
            dept = dept_entry.get()
            sec = sec_entry.get()
            if not name or not dept or not sec:
                messagebox.showerror("Error", "Please fill in all fields")
                return
            try:
                db = connect_db()
                cursor = db.cursor()
                enrollment_table = f"enroll_{event_name.replace(' ', '_')}"
                cursor.execute(f"CREATE TABLE IF NOT EXISTS {enrollment_table} (name
VARCHAR(255), dept VARCHAR(255), sec VARCHAR(255))")
                cursor.execute(f"INSERT INTO {enrollment_table} (name, dept, sec) VALUES (%s,
%s, %s)", (name, dept, sec))
                db.commit()
                messagebox.showinfo("Success", "Enrollment Successful!")
            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Error: {err}")
            finally:
                if 'db' in locals():
                    db.close()
        ctk.CTkLabel(details_window, text="Name:").pack(pady=5)
        name_entry = ctk.CTkEntry(details_window)
        name_entry.pack(pady=5)
        ctk.CTkLabel(details_window, text="Dept:").pack(pady=5)
        dept_entry = ctk.CTkEntry(details_window)
        dept_entry.pack(pady=5)
        ctk.CTkLabel(details_window, text="Sec:").pack(pady=5)
        sec_entry = ctk.CTkEntry(details_window)
        sec_entry.pack(pady=5)
        enroll_button = ctk.CTkButton(details_window, text="Enroll", command=enroll)
        enroll_button.pack(pady=10)
```

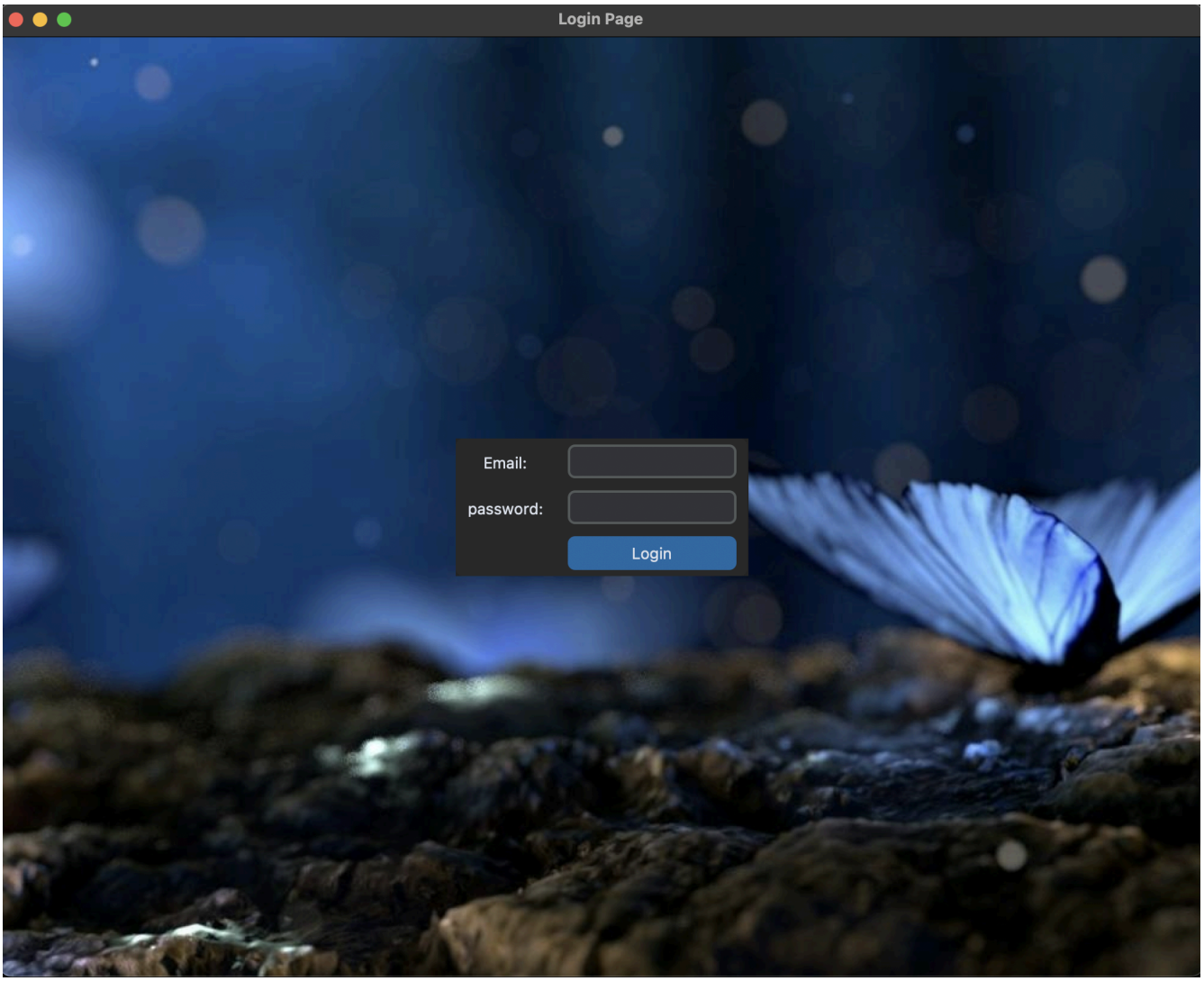
```
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Error: {err}")
        finally:
            if 'db' in locals():
                db.close()
    display_events()
    student_window.mainloop()
# Main signup window
signup_window = ctk.CTk()
signup_window.title("Signup Page")
signup_window.geometry("1000x1000")
bg_image = Image.open("rec.jpg")
bg_photo = ImageTk.PhotoImage(bg_image)
bg_label = ctk.CTkLabel(signup_window, image=bg_photo)
bg_label.place(relx=0.5, rely=0.5, anchor='center')
form_frame = ctk.CTkFrame(signup_window)
form_frame.place(relx=0.5, rely=0.5, anchor='center')
firstname_label = ctk.CTkLabel(form_frame, text="First Name:")
firstname_label.grid(row=0, column=0, padx=10, pady=5)
firstname_entry = ctk.CTkEntry(form_frame)
firstname_entry.grid(row=0, column=1, padx=10, pady=5)
lastname_label = ctk.CTkLabel(form_frame, text="Last Name:")
lastname_label.grid(row=1, column=0, padx=10, pady=5)
lastname_entry = ctk.CTkEntry(form_frame)
lastname_entry.grid(row=1, column=1, padx=10, pady=5)
email_label = ctk.CTkLabel(form_frame, text="Email:")
email_label.grid(row=2, column=0, padx=10, pady=5)
email_entry = ctk.CTkEntry(form_frame)
email_entry.grid(row=2, column=1, padx=10, pady=5)
password_label = ctk.CTkLabel(form_frame, text="Password:")
password_label.grid(row=3, column=0, padx=10, pady=5)
password_entry = ctk.CTkEntry(form_frame, show="*")
password_entry.grid(row=3, column=1, padx=10, pady=5)
register_button = ctk.CTkButton(form_frame, text="Register", command=register)
register_button.grid(row=5, column=0, columnspan=2, pady=10)
login_button = ctk.CTkButton(form_frame, text="Already have an account? Login",
command=open_login_page)
login_button.grid(row=6, column=0, columnspan=2, pady=10)
signup_window.mainloop()
```

2. RESULTS AND DISCUSSION

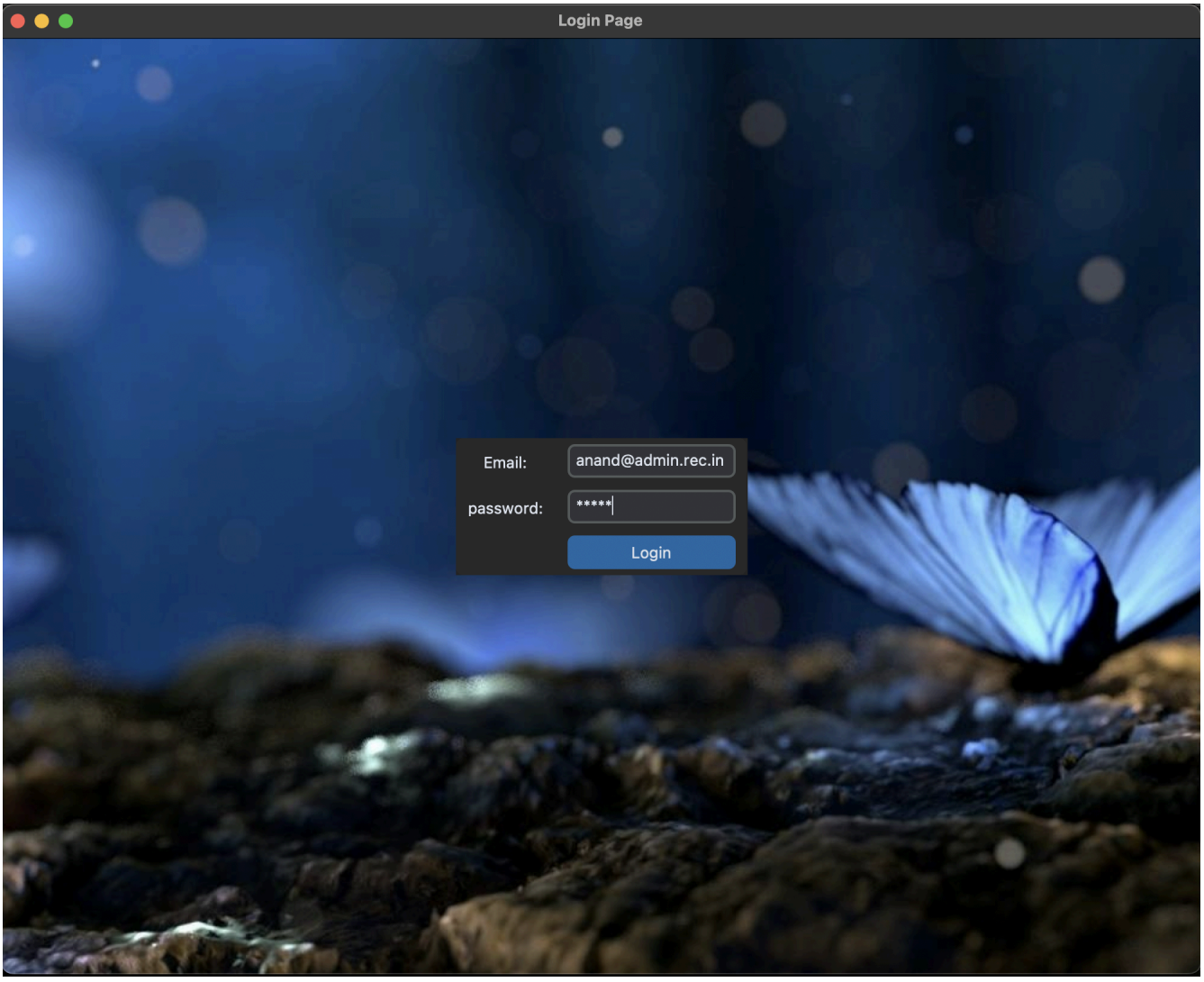
REGISTER PAGE:



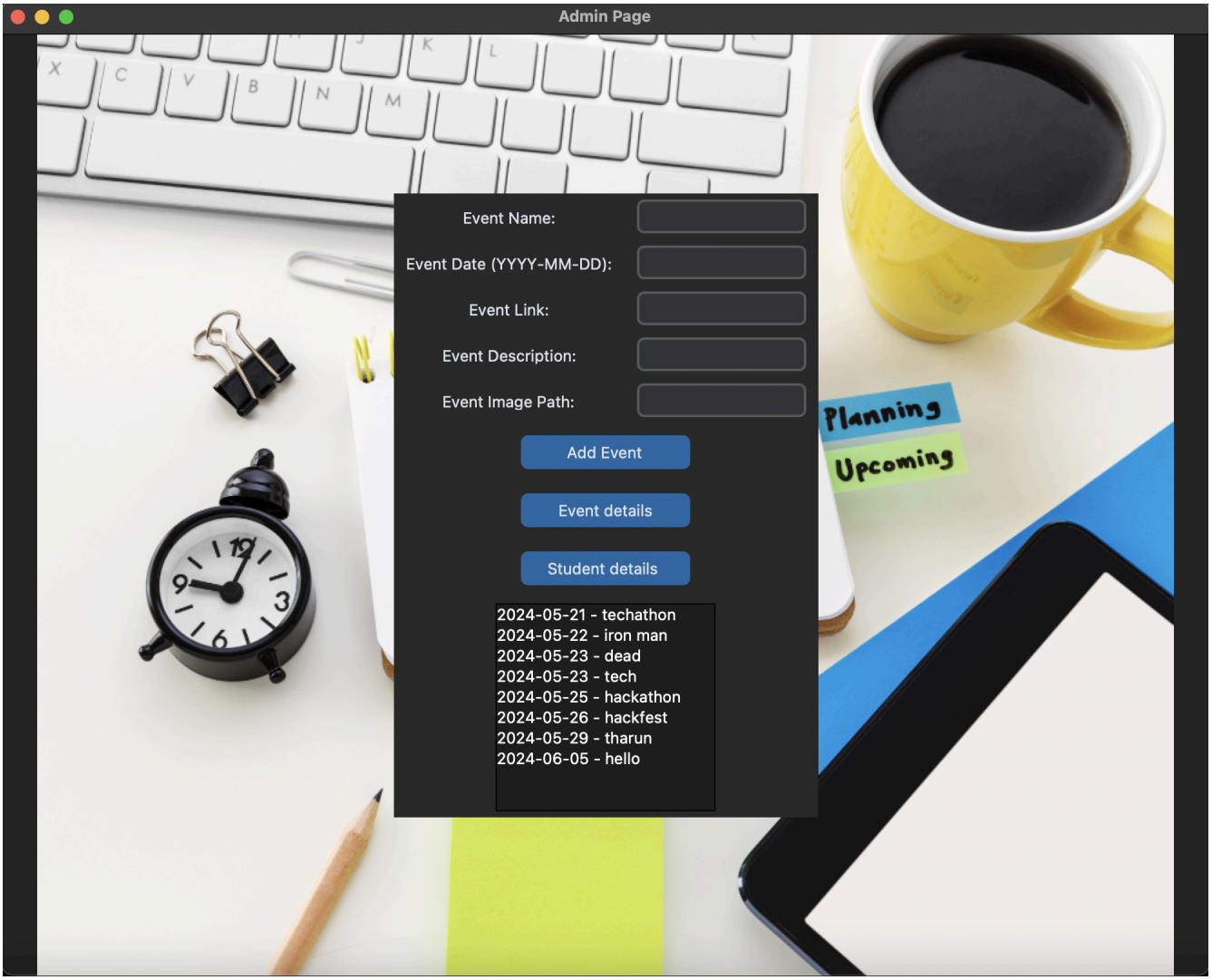
LOGIN PAGE:



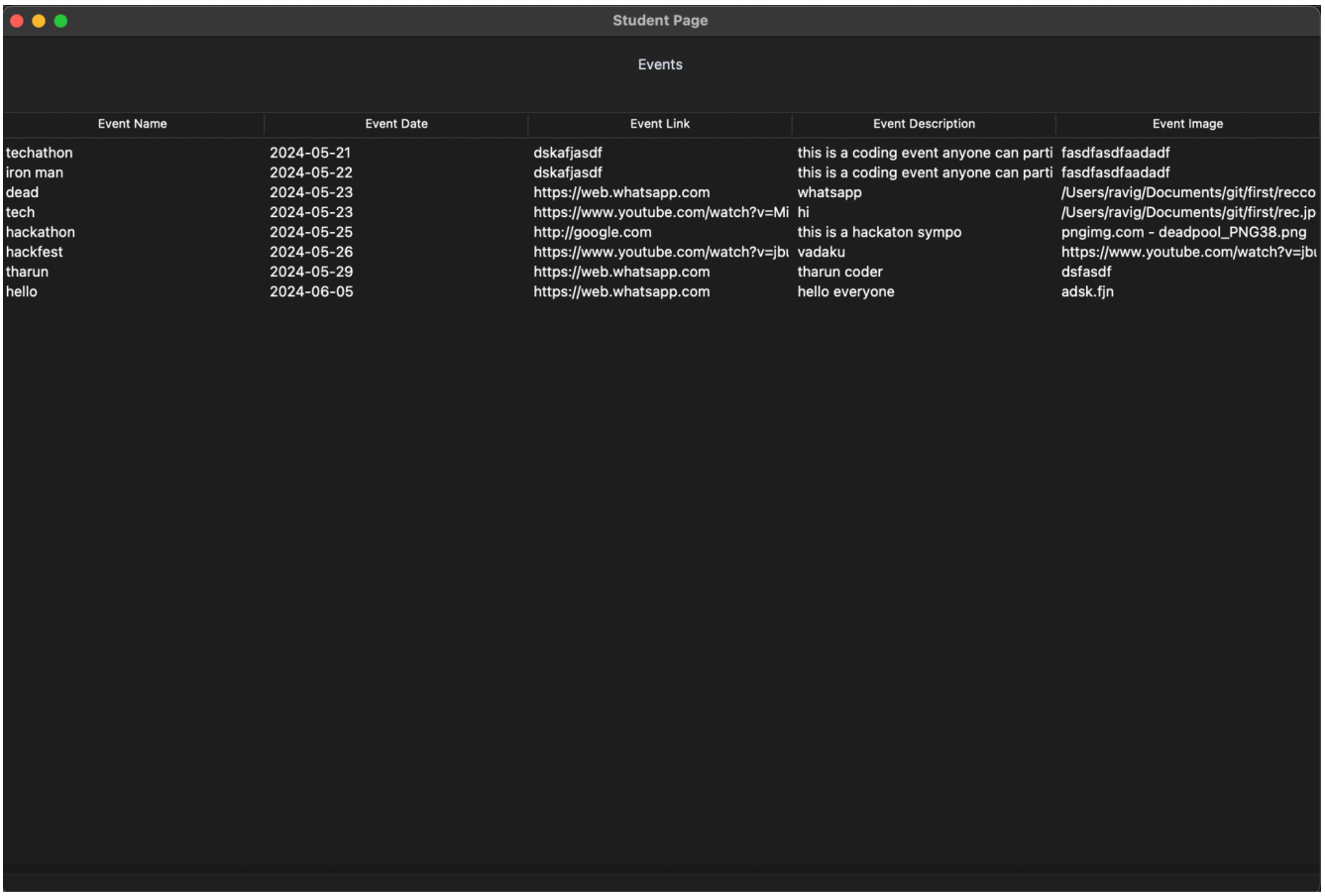
LOGIN PAGE:



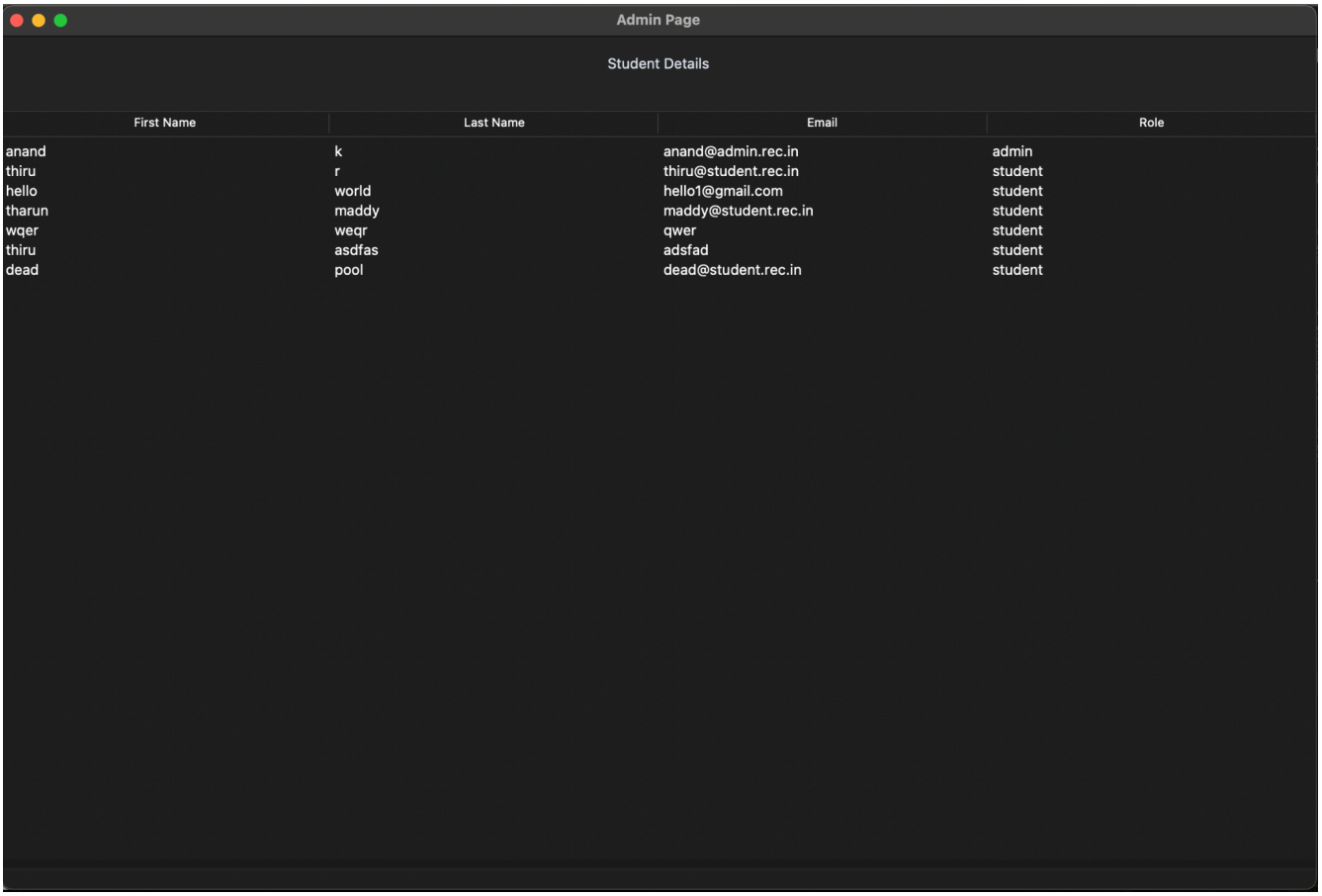
ADMIN LOGIN:


Admin Page
Event Name:
Event Date (YYYY-MM-DD):
Event Link:
Event Description:
Event Image Path:
Add Event
Event details
Student details
2024-05-21 - techathon
2024-05-22 - iron man
2024-05-23 - dead
2024-05-23 - tech
2024-05-25 - hackathon
2024-05-26 - hackfest
2024-05-29 - tharun
2024-06-05 - hello
Planning
Upcoming

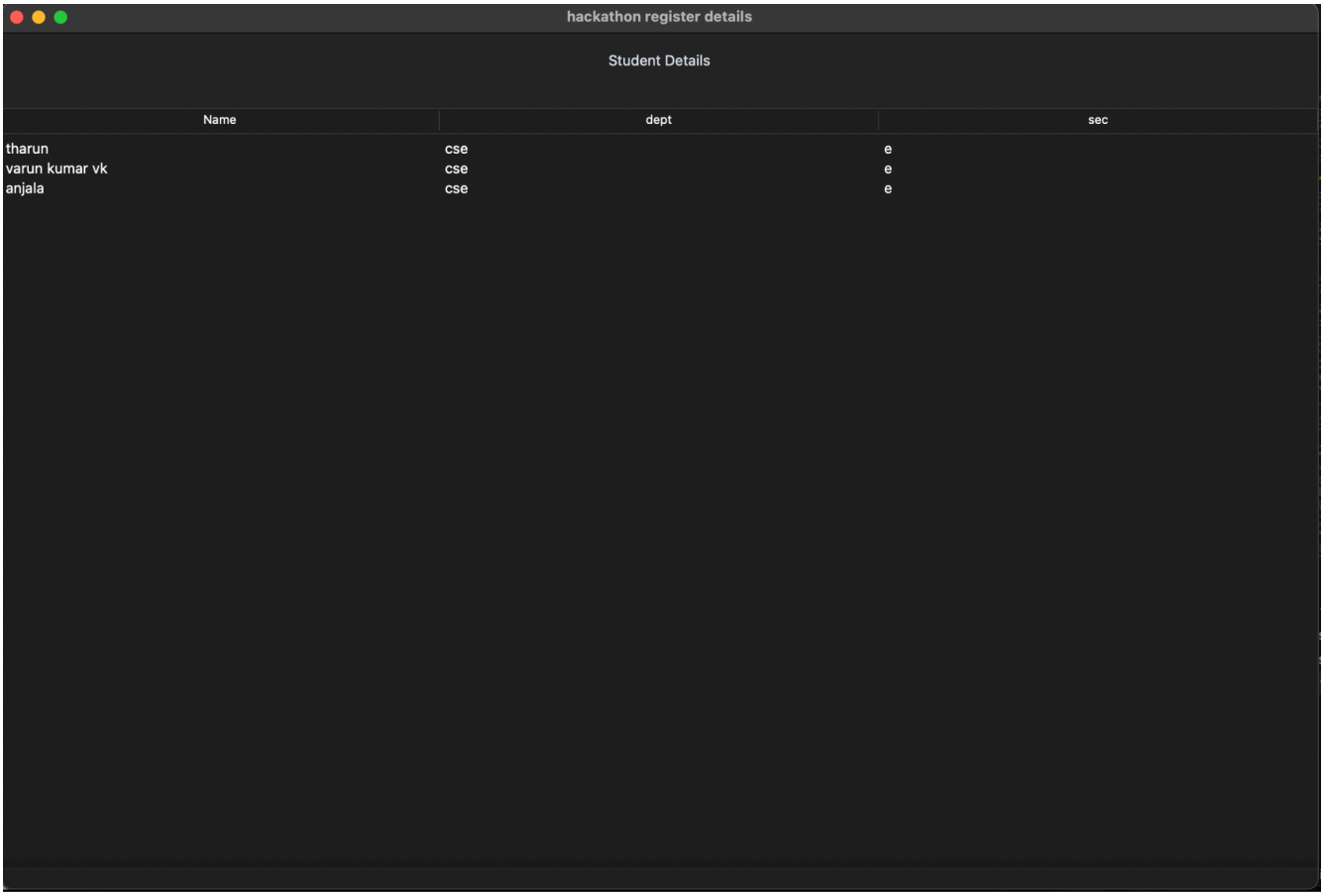
EVENT DETAILS TABLE:



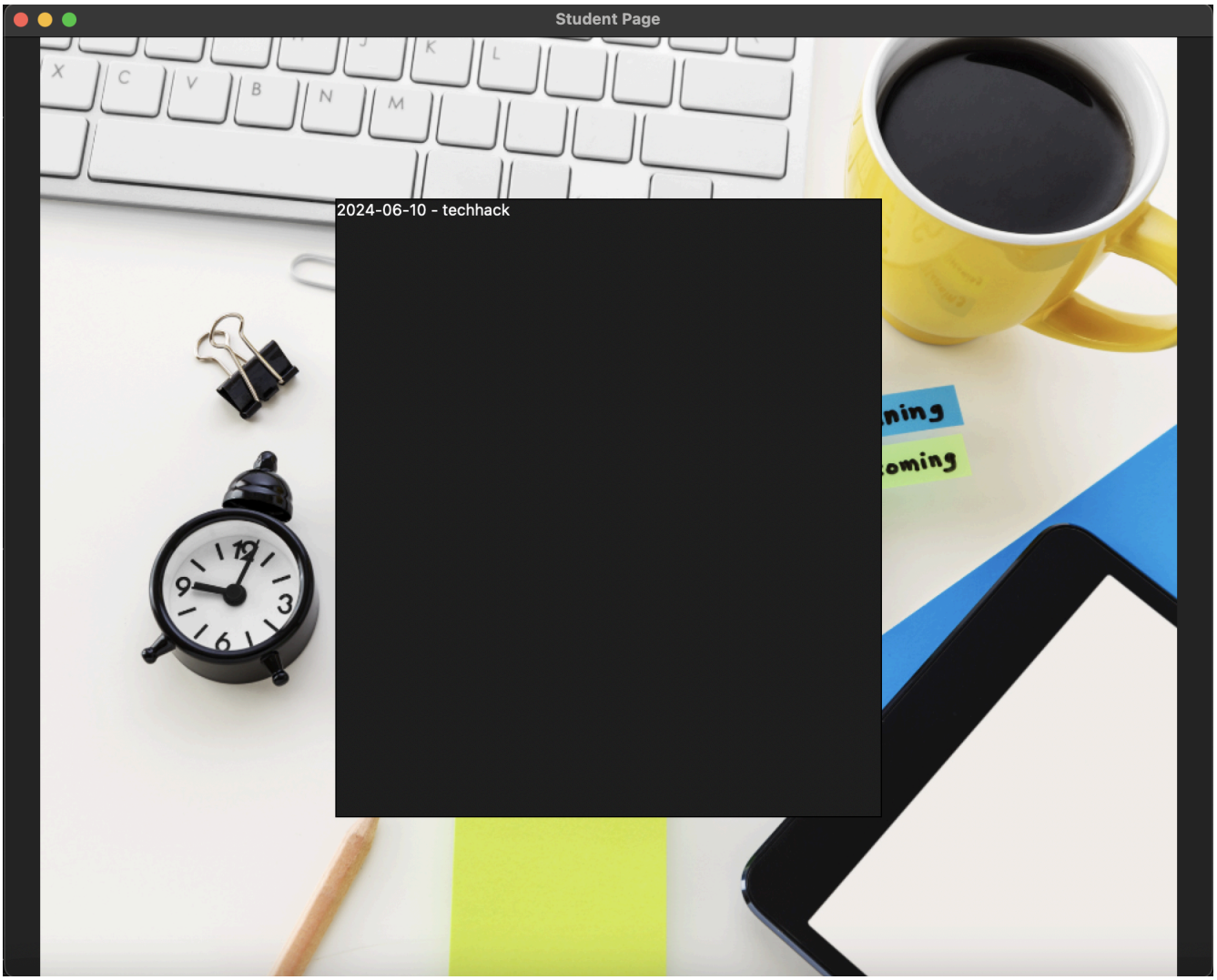
REGISTER DETAILS:



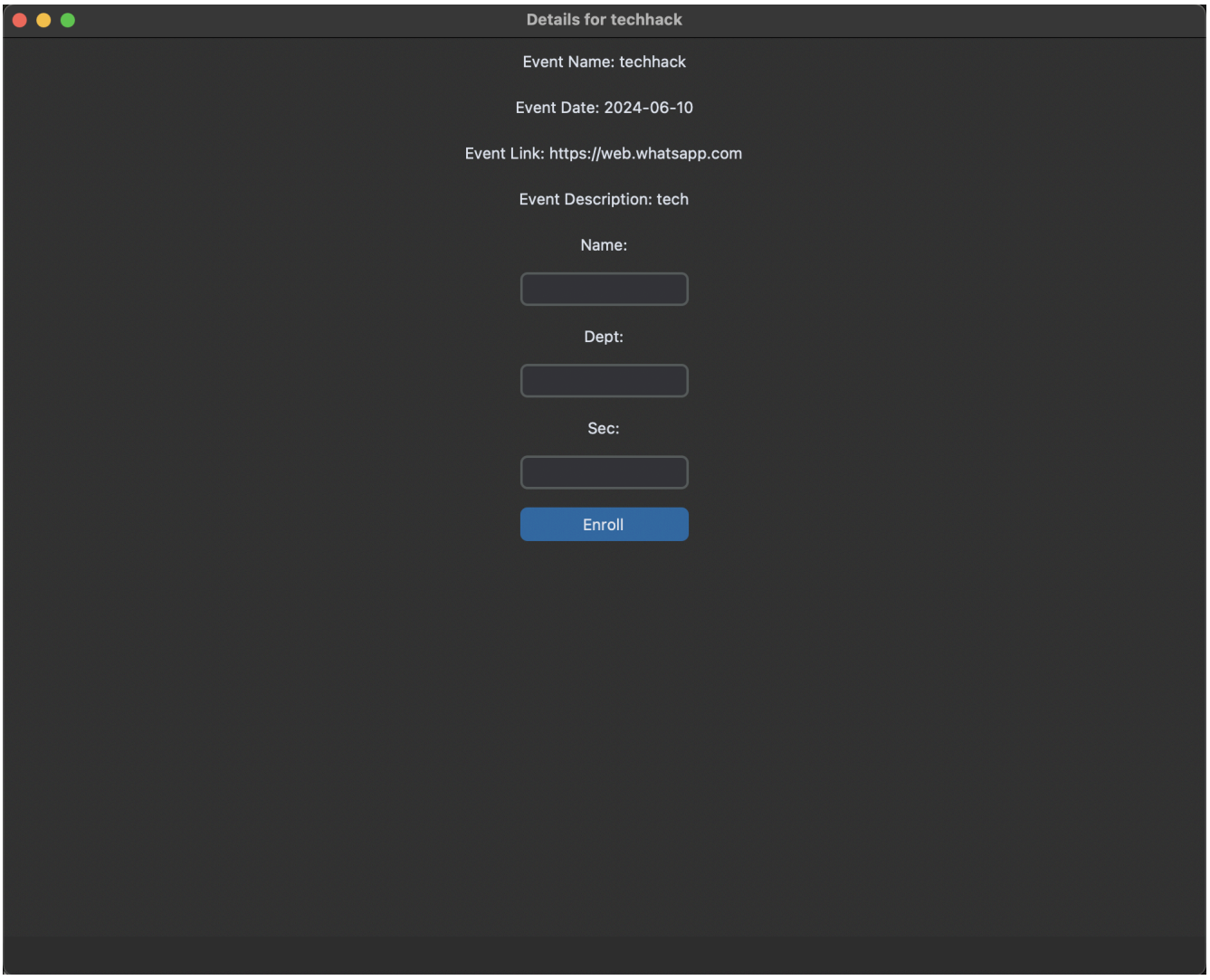
EVENT REGISTER TABLE:



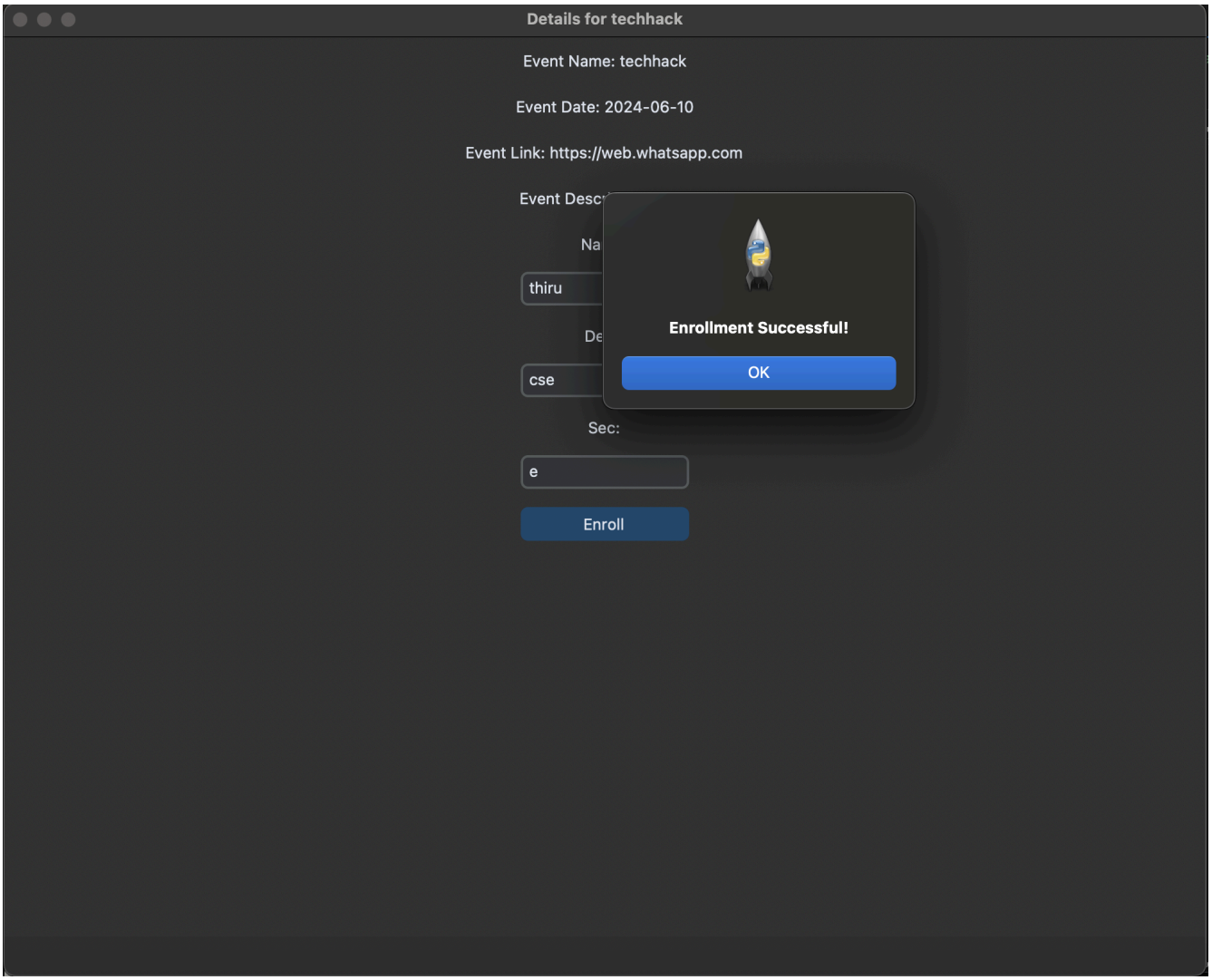
STUDENT LOGIN



ENROLL PAGE:


Details for techhack
Event Name: techhack
Event Date: 2024-06-10
Event Link: https://web.whatsapp.com
Event Description: tech
Name:
Dept:
Sec:
Enroll

ENROLLMENT SUCCESSFUL POPUP:



6. CONCLUSION

Event Management System represents a significant advancement in the way educational institutions handle event management. By streamlining and automating various tasks, such as registration, event creation, and student enrollment, the system enhances operational efficiency and reduces the manual workload for administrators. The user-friendly interface, powered by CustomTkinter (CTkinter) and backed by a MySQL database, ensures ease of use and robust data management. Real-time access to event information promotes better communication and coordination among students and administrators, leading to more successful and well-organized events. Overall, the system's implementation promises a more efficient and effective event management process within educational institutions.

7. REFERENCES

The below websites helped us in gaining more knowledge on the subject and in completing the project

https://www.geeksforgeeks.org/python-gui-tkinter/

https://github.com/demin13/Event-Management-System