Campus Recruitment Management System

A PROJECT REPORT

Submitted by

SATHIYAJITH AASHIM GOVINDARAJ (2303811724321098)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE- 2025

CAMPUS RECRUITMENT MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

SATHIYAJITH AASHIM GOVINDARAJ (2303811724321098)

in partial fulfillment of requirements for the award of the course

CGB1221-DATABASE MANAGEMENT SYSTEMS

in

ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY

(An Autonomous Institution, affiliated to Anna University Chennai and Approved by AICTE, New Delhi)

SAMAYAPURAM – 621 112

JUNE- 2025

K. RAMAKRISHNAN COLLEGE OF TECHNOLOGY (AUTONOMOUS)

SAMAYAPURAM – 621 112

BONAFIDE CERTIFICATE

Certified that this project report on **CAMPUS RECRUITMENT MANAGEMENT SYSTEM** is the bonafide work of **SATHIYAJITH AASHIM GOVINDARAJ (2303811724321098)** who carried out the project work during the academic year 2024 - 2025 under my supervision.

SIGNATURE

Dr.T. AVUDAIAPPAN, M.E.,Ph.D.,

HEAD OF THE DEPARTMENT

ASSOCIATE PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

SIGNATURE

Mrs.S. GEETHA, M.E.,

SUPERVISOR

ASSISTANT PROFESSOR

Department of Artificial Intelligence

K.Ramakrishnan College of Technology (Autonomous)

Samayapuram–621112.

Submitted for the viva-voce examination held on …04.06.2025…….

INTERNAL EXAMINER **EXTERNAL EXAMINER**

DECLARATION

I declare that the project report on **CAMPUS RECRUITMENT MANAGEMENT SYSTEM** is the result of original work done by me and best of my knowledge, similar work has not been submitted to **"ANNA UNIVERSITY CHENNAI "** for the requirement of Degree of **BACHELOR OF TECHNOLOGY**. This project report is submitted on the partial fulfilment of the requirement of the completion of the course **CGB1221 – DATABASE MANAGEMENT SYSTEMS.**

.

Signature

SATHIYAJITH AASHIM GOVINDARAJ

Place: Samayapuram

Date: 04.06.2025

ACKNOWLEDGEMENT

It is with great pride that I express our gratitude and in-debt to our institution "**K.Ramakrishnan College of Technology (Autonomous)**", for providing us with the opportunity to do this project.

I glad to credit honourable chairman **Dr. K. RAMAKRISHNAN**, **B.E.,** for having provided for the facilities during the course of our study in college.

I would like to express our sincere thanks to our beloved Executive Director **Dr. S. KUPPUSAMY, MBA, Ph.D.,** for forwarding to our project and offering adequate duration in completing our project.

I would like to thank **Dr. N. VASUDEVAN, M.Tech., Ph.D.,** Principal, who gave opportunity to frame the project the full satisfaction.

I whole heartily thanks to **Dr. T. AVUDAIAPPAN, M.E.,Ph.D.,** Head of the department, **ARTIFICIAL INTELLIGENCE** for providing his encourage pursuing this project.

I express our deep expression and sincere gratitude to our project supervisor **Mrs.S.GEETHA, M.E.,** Department of **ARTIFICIAL INTELLIGENCE,** for her incalculable suggestions, creativity, assistance and patience which motivated us to carry out this project.

I render our sincere thanks to Course Coordinator and other staff members for providing valuable information during the course.

I wish to express our special thanks to the officials and Lab Technicians of our departments who rendered their help during the period of the work progress.

INSTITUTE

Vision:

- To serve the society by offering top-notch technical education on par with global standards.

Mission:

- Be a center of excellence for technical education in emerging technologies by exceeding the needs of industry and society.
- Be an institute with world class research facilities.
- Be an institute nurturing talent and enhancing competency of students to transform them as all – round personalities respecting moral and ethical values.

DEPARTMENT

Vision:

- To excel in education, innovation, and research in Artificial Intelligence and Data Science to fulfil industrial demands and societal expectations.

Mission

- To educate future engineers with solid fundamentals, continually improving teaching methods using modern tools.
- To collaborate with industry and offer top-notch facilities in a conducive learning environment.
- To foster skilled engineers and ethical innovation in AI and Data Science for global recognition and impactful research.
- To tackle the societal challenge of producing capable professionals by instilling employability skills and human values.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

- **PEO1:** Compete on a global scale for a professional career in Artificial Intelligence and Data Science.
- **PEO2:** Provide industry-specific solutions for the society with effective communication and ethics.
- **PEO3** Enhance their professional skills through research and lifelong learning initiatives.

PROGRAM SPECIFIC OUTCOMES (PSOs)

- **PSO1:** Capable of finding the important factors in large datasets, simplify the data, and improve predictive model accuracy.
- **PSO2:** Capable of analyzing and providing a solution to a given real-world problem by designing an effective program.

PROGRAM OUTCOMES (POs)

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences

3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations

4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions

5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations

6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice

7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development

8. **Ethics:** Apply ethical principles and commit to professional ethics and

responsibilities and norms of the engineering practice.

9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

ABSTRACT

The Campus Recruitment System is an interactive desktop application developed using Python for the backend logic, Tkinter for the graphical user interface (GUI), and MySQL for database management. The application allows two main types of users: administrators (company representatives) and students. Administrators can register and log in using their company email credentials, post job vacancies, view student applications, and send replies directly through the app or via integrated Gmail-based email service using SMTP, making communication seamless and effective. Students can register, log in, view job listings, and apply for positions with just a few clicks. The GUI features a colorful interface with background images for better visual appeal and user engagement. All user actions, including logins, job postings, applications, and admin replies, are stored in a secure MySQL database, ensuring data integrity and traceability. The project also supports in-app messaging and email notifications to ensure timely responses and feedback, enhancing the overall experience for both students and companies. By leveraging technology, the Campus Recruitment System promotes a modernized approach to campus hiring, offering scalability, security, and ease of use for all stakeholders involved.

ABSTRACT WITH POs AND PSOs MAPPING

CO 5 : BUILD DATABASES FOR SOLVING REAL-TIME PROBLEMS.

ABSTRACT	POs MAPPED	PSOs MAPPED
The Campus Recruitment System is a Python-Tkinter desktop application with MySQL backend that streamlines campus hiring. It allows company admins to post jobs, view applications, and send replies via both app and Gmail (SMTP). Students can register, log in, view job posts, and apply with ease. The system ensures efficient communication, secure data storage, and a user-friendly GUI with enhanced visuals. By digitizing the traditional recruitment process, it promotes transparency, reduces manual work, and improves the overall hiring experience on campus.	**PO1 -3** **PO2 -2** **PO3 -3** **PO4 -2** **PO5 -3** **PO6 -3** **PO7 -3** **PO8 -2** **PO9 -2** **PO10 -3** **PO11-3** **PO12 -2**	**PSO1 -3** **PSO2 -3**

Note: 1- Low, 2-Medium, 3- High

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 OBJECTIVE

The primary objective of the Campus Recruitment System is to digitize and streamline the process of campus placements in educational institutions. Traditional campus recruitment involves manual steps such as posting job advertisements on notice boards, collecting physical resumes, arranging interviews offline, and handling communication through email or word-of-mouth. This approach is time-consuming, lacks organization, and is difficult to track or audit. The proposed system aims to eliminate these challenges by offering a centralized platform where administrators (companies) can log in using their credentials to post job opportunities, view student applications, and respond to applicants via email and the application interface. Students, in turn, can register on the system, view job postings, and submit their applications directly through the platform. The system supports sending replies to students via the company's registered Gmail address using SMTP, making communication fast, traceable, and professional. Ultimately, the objective is to provide a secure, user-friendly, and effective environment for managing campus recruitment activities efficiently and transparently.

1.2 OVERVIEW

The Campus Recruitment System is a Python-based desktop application with a graphical user interface developed using Tkinter. It interacts with a MySQL relational database to store and manage all user and job-related data. The system offers two primary roles: Admins (Companies) and Students. Admins can sign in with their registered Gmail accounts and passwords. Upon successful login, they can post job vacancies by providing details like job title, description, salary, and upload relevant images or short videos (up to 30

seconds) to enhance the visual presentation of the job post. Admins can also view a list of students who applied for specific jobs and send individual replies via Gmail (SMTP) that also appear within the application as messages for the students.

Students can register by providing personal and academic details and then log in using a pop-up form. After login, they can browse available job opportunities, view job descriptions and media content, and apply for jobs directly through the interface. Every application is recorded in the backend, and students receive confirmation messages upon applying. Replies from companies are delivered both within the app and to the students' email addresses, ensuring they stay informed even outside the application.

All activities, including registrations, logins, job posts, applications, likes, comments, and replies, are stored in the MySQL database, ensuring complete traceability. The integration of multimedia content, email communication, and database logging makes the system comprehensive, modern, and robust. It transforms the recruitment process into a more dynamic, engaging, and professional experience for both companies and students.

1.3 SQL AND DATABASE CONCEPTS

The Campus Recruitment System relies heavily on structured database concepts using MySQL, a relational database management system known for its scalability, speed, and security. The application uses multiple interrelated tables to organize data efficiently. These include:

- **admin_login:** Stores company login details such as email and password.
- **student_register:** Contains student information like name, email, phone, and academic qualifications.
- **job_posts:** Stores job-related data posted by admins, including descriptions, media file names (images/videos), and timestamps.

- **applications:** Logs student job applications, capturing which student applied for which job and when.
- **replies:** Maintains communication logs where admins send feedback or selection results to applicants.

The system uses primary keys (e.g., student_id, job_id) to uniquely identify records and foreign keys to establish relationships between tables, such as linking applications to students and jobs. This approach ensures referential integrity, meaning that every application corresponds to a valid student and job post.

SQL statements are used for all core data operations:

- **INSERT** statements handle new records like registrations and job posts.
- **SELECT** queries retrieve data for dashboards and job listings.
- **UPDATE** statements manage changes in data, such as updating student profiles or job details.
- **DELETE** statements are used for data cleanup when needed.

The database follows normalization principles to reduce redundancy and improve efficiency. For example, job media files are stored as filenames or paths, not as binary blobs, which optimizes storage and retrieval.

Additionally, the application integrates SMTP (Simple Mail Transfer Protocol) using Python's smtplib to send emails from the admin's Gmail account to student emails. The system uses an App Password for secure authentication. All sent emails are logged in the replies table, ensuring consistent tracking and compliance.

The system embodies key database concepts such as:

- Data integrity (with constraints like NOT NULL and UNIQUE),
- Security (login validation, email verification),
- Concurrency (multiple users interacting simultaneously),

- Backup and recovery (through MySQL tools), and scalability, allowing more features and users to be added without redesigning the system architecture.

By combining SQL's powerful querying capabilities with a well-structured schema and a responsive Python GUI, the Campus Recruitment System ensures both functional depth and ease of use for end-users.

CHAPTER 2

PROJECT METHODOLOGY

2.1 PROPOSED WORK

The proposed work focuses on building a dynamic and interactive desktop application that automates the campus recruitment process using a structured methodology. The methodology follows a modular and incremental approach, breaking down the system into clear functional components such as registration, login, job posting, job application, and message handling. Each component is designed and tested individually to ensure smooth operation and integration.

The application uses Python for front-end development with the Tkinter library for GUI design. Tkinter enables the creation of user-friendly interfaces where both students and admins can interact with the system easily. MySQL is used as the backend to store data such as user credentials, job listings, applications, and replies. The system ensures secure login for both companies and students using validations. Admins can log in using a Gmail account and an app password to securely send replies through SMTP (Simple Mail Transfer Protocol) to students' emails, in addition to in-app replies.

To enhance interactivity, job postings can include images and short 30-second videos that are displayed in the GUI using media widgets or external viewers. Each student can apply to multiple jobs, and the admin can track and respond to each application individually. The system records all activities (logins, job postings, applications, replies) in the database for transparency and audit purposes.

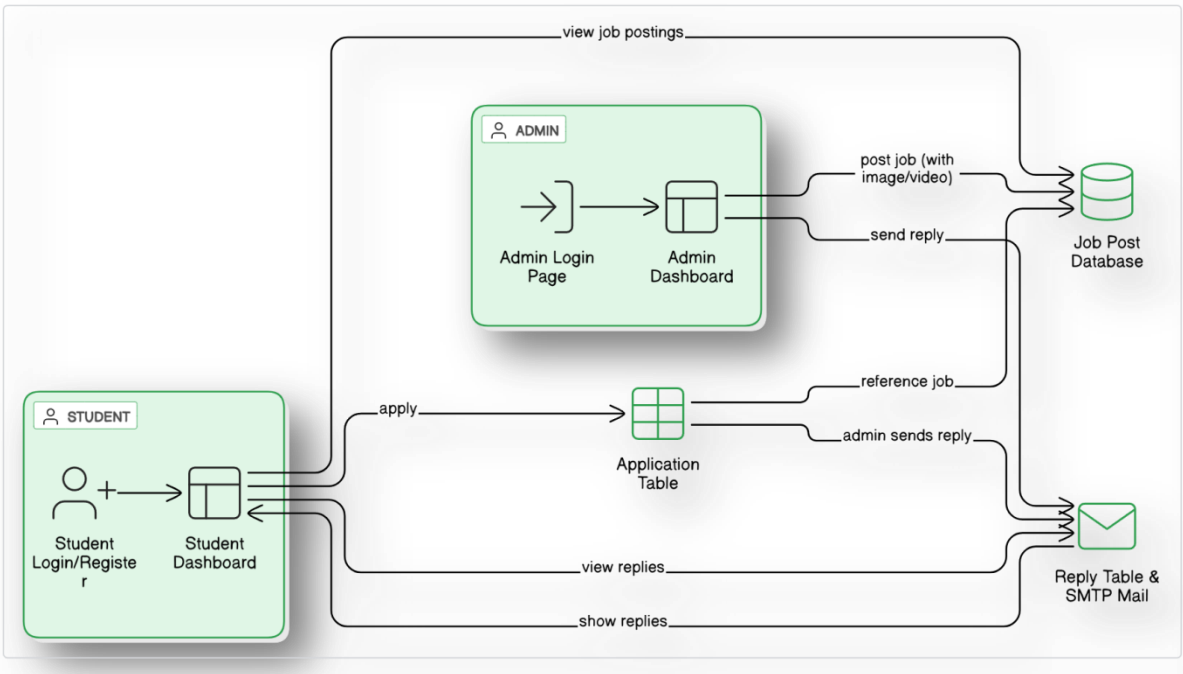
The development methodology follows these key phases:

- **Requirement Analysis** – Understanding and documenting what features are needed for admins and students.

- **System Design** – Designing the user interface layout and database schema.
- **Module Development** – Implementing login, registration, job management, and communication functionalities.
- **Database Integration** – Connecting Python to MySQL and structuring queries for data manipulation.
- **Testing and Debugging** – Ensuring each module performs as expected with valid and invalid inputs.
- **Deployment** – Finalizing the GUI layout, media support, and packaging the application for use.

This systematic methodology ensures that the system is scalable, secure, easy to use, and effectively meets the needs of both students and recruiters.

2.2 BLOCK DIAGRAM



CHAPTER 3

MODULE DESCRIPTION

The Campus Recruitment System is divided into several functional modules, each responsible for a specific set of operations. This modular architecture enhances scalability, maintainability, and simplifies the debugging process. The major modules in this system include:

3.1 ADMIN MODULE

The Admin module is designed for company representatives or recruiters who want to post job opportunities and manage student applications.

Key Functions:

- **Login System:** Authenticates the admin using a registered email and password. It ensures that only authorized company personnel can access the admin dashboard.
- **Job Posting:** Allows admins to add new job opportunities including job title, description, requirements, and salary details. They can also upload related images or 30-second video clips that help in attracting student interest.
- **Application Management:** Enables the admin to view student applications for each job post.
- **Reply Management:** Admins can send replies to students through two methods: (1) storing the message in the database for students to view in the app, and (2) sending an email reply using Gmail SMTP (via app password) to the applicant.
- **Activity Logging:** Records all admin actions such as login time, job postings, replies, and logout events in the MySQL database.

3.2 STUDENT MODULE

This module provides features for students to register, log in, view job postings, and apply for jobs.

Key Functions:

- **Registration System:** New students can register by providing basic information such as name, email, phone number, course, and password. The details are stored in the student table of the database.
- **Login System:** Authenticates the student based on registered credentials.
- **Job Browsing:** Displays job posts in a user-friendly GUI, including any images or videos uploaded by the admin.
- **Application Submission:** Students can apply to available job posts. Their application details are stored in the application table.
- **Reply Viewing:** Students can view responses sent by the admin in the GUI. Each reply corresponds to a job application and is linked to the student's account.
- **Like & Comment Feature:** Allows students to like job posts and comment with their feedback. This interaction is stored in a separate table and visible to the admin.

3.3 JOB MANAGEMENT MODULE

This module connects the admin's actions with student interaction.

Key Functions:

- **Job Creation:** Admins create job entries stored in the database.
- **Job Display:** Retrieves and displays job posts with corresponding media to all logged-in students.
- **Job Details Update:** Admins can update or remove job listings as needed.

3.4 APPLICATION AND COMMUNICATION MODULE

This is a bridge between students and admins for seamless application and response handling.

Key Functions:

- **Application Logging:** Stores which student applied to which job, with timestamps.
- **Reply System:** Admins can send replies stored in the database and simultaneously dispatch email notifications to the student's Gmail account.
- **Email Integration:** Uses Python’s smtplib with Gmail’s SMTP server to send email responses securely via app password.

3.5 DATABASE MODULE

This backend module handles all the data operations and ensures data consistency, integrity, and retrieval.

Key Functions:

- **MySQL Integration:** The system connects to a MySQL database hosted on XAMPP (username: localhost, password: "").
- **Tables Management:** Includes tables for admin credentials, student details, job posts, applications, likes/comments, and replies.
- **Data Retrieval and Manipulation:** Handles queries for login validation, job search, application storage, and communication history.

CHAPTER 4

CONCLUSION & FUTURE SCOPE

CONCLUSION:

The Campus Recruitment System developed using Python with Tkinter for GUI and MySQL for backend storage successfully achieves its primary goal of streamlining the recruitment process within an educational institution. The application provides a user-friendly platform where administrators (companies or recruiters) can post job openings, and students can view and apply for them efficiently. The inclusion of features such as job posting with images and videos, application tracking, admin replies via both in-app messages and email, and the ability for students to like and comment on posts enhances the overall interactivity and engagement of the system.

The login and registration functionalities for both students and admins ensure secure access to the platform, while the integration with Gmail's SMTP server allows real-time communication and acknowledgment. All user activities are logged into a MySQL database, ensuring data consistency and supporting robust reporting capabilities. This modular and scalable system addresses key challenges in campus recruitment and serves as a digital bridge between job providers and students.

FUTURE SCOPE:

Although the current system addresses many fundamental needs, several enhancements can be made to further improve functionality and usability in future versions:

- **Resume Uploading Feature:**
    - Enable students to upload their resumes in PDF or DOC format during job application submission.
- **AI-based Job Recommendations:**

    - Incorporate a recommendation engine that suggests relevant job postings to students based on their course, skill sets, and application history.
- **Video Interview Integration:**
    - Include a scheduling system and video conferencing capability for interviews, reducing the need for in-person interactions.
- **Admin Analytics Dashboard:**
    - Provide visual analytics to the admin, such as the number of applications received per job post, most liked posts, and student engagement trends.
- **Notification System:**
    - Implement real-time push notifications for new job posts, admin replies, and status updates of applications.
- **Mobile App Version:**
    - Extend the system as a mobile application for Android and iOS platforms to increase accessibility and convenience for both students and admins.
- **Enhanced Security:**
    - Add features such as password encryption, two-factor authentication, and user session tracking to strengthen system security.
- **Placement Statistics Module:**
    - Maintain and visualize placement records of previous years to help students and college management analyze placement trends.

By implementing these future enhancements, the Campus Recruitment System can evolve into a comprehensive platform capable of handling end-to-end recruitment and placement needs for academic institution.

APPENDIX A SOURCE CODE

```
import tkinter as tk
from tkinter import messagebox, ttk, simpledialog
from PIL import Image, ImageTk
import mysql.connector
import smtplib
from email.message import EmailMessage

def connect_db():
    return mysql.connector.connect(
        host="localhost", user="root", password="758110",
database="CampusRecruitmentDB"
    )

def log_activity(activity):
    db = connect_db()
    cursor = db.cursor()
    cursor.execute("INSERT INTO logs (activity) VALUES (%s)",
(activity,))
    db.commit()
    db.close()

def send_email(sender, receiver, subject, body):
    app_password = "dtll fqxz nxit sopp"
    try:
        msg = EmailMessage()
        msg.set_content(body)
        msg['Subject'] = subject
        msg['From'] = sender
        msg['To'] = receiver

        with smtplib.SMTP('smtp.gmail.com', 587) as smtp:
            smtp.starttls()
            smtp.login(sender, app_password)
            smtp.send_message(msg)
    except smtplib.SMTPAuthenticationError:
```

```
        messagebox.showerror("Email Error", "Authentication failed. Check
your email and app password.")
    except Exception as e:
        messagebox.showerror("Email Error", str(e))

admin_email = None

def open_login_window():
    login_win = tk.Toplevel(root)
    login_win.title("Login")
    login_win.geometry("400x320")
    login_win.grab_set()

    notebook = ttk.Notebook(login_win)
    notebook.pack(expand=True, fill='both', padx=10, pady=10)

    admin_frame = tk.Frame(notebook)
    notebook.add(admin_frame, text="Admin Login")

    tk.Label(admin_frame, text="Admin Email", font=("Helvetica",
12)).pack(pady=(15,5))
    admin_email_entry = tk.Entry(admin_frame, width=30)
    admin_email_entry.pack()

    tk.Label(admin_frame, text="Admin Password", font=("Helvetica",
12)).pack(pady=(15,5))
    admin_password_entry = tk.Entry(admin_frame, show="*", width=30)
    admin_password_entry.pack()

    def admin_attempt_login():
        global admin_email
        email = admin_email_entry.get()
        password = admin_password_entry.get()

        db = connect_db()
        cursor = db.cursor()
```

```
        cursor.execute("SELECT * FROM admin WHERE email=%s AND
password=%s", (email, password))
        admin = cursor.fetchone()
        db.close()

        if admin:
            admin_email = email
            messagebox.showinfo("Login Success", "Welcome, Admin!")
            login_win.destroy()
            open_admin_dashboard()
        else:
            messagebox.showerror("Login Failed", "Invalid admin
credentials")

    tk.Button(admin_frame, text="Login", command=admin_attempt_login,
bg="#ff6666", fg="white").pack(pady=20)

    student_frame = tk.Frame(notebook)
    notebook.add(student_frame, text="Student Login")

    tk.Label(student_frame, text="Student Email", font=("Helvetica",
12)).pack(pady=(15,5))
    student_email_entry = tk.Entry(student_frame, width=30)
    student_email_entry.pack()

    tk.Label(student_frame, text="Student Password", font=("Helvetica",
12)).pack(pady=(15,5))
    student_password_entry = tk.Entry(student_frame, show="*",
width=30)
    student_password_entry.pack()

    def student_attempt_login():
        email = student_email_entry.get()
        password = student_password_entry.get()

        db = connect_db()
        cursor = db.cursor()
```

```
        cursor.execute("SELECT * FROM students WHERE email=%s AND
password=%s", (email, password))
        student = cursor.fetchone()
        db.close()
        if student:
            log_activity(f"Student {email} logged in")
            messagebox.showinfo("Login Success", f"Welcome, {email}")
            login_win.destroy()
            student_dashboard(email)
        else:
            messagebox.showerror("Login Failed", "Invalid student
credentials")

    tk.Button(student_frame, text="Login",
command=student_attempt_login, bg="#99e699",
fg="white").pack(pady=20)

def open_admin_dashboard():
    win = tk.Toplevel(root)
    win.title("Admin Dashboard")
    win.geometry("850x600")

    notebook = ttk.Notebook(win)
    notebook.pack(expand=True, fill='both')

    post_frame = tk.Frame(notebook)
    notebook.add(post_frame, text="Post Job")

    tk.Label(post_frame, text="Post a Job", font=("Helvetica",
16)).pack(pady=10)

    tk.Label(post_frame, text="Job Title").pack(pady=5)
    title_entry = tk.Entry(post_frame)
    title_entry.pack()

    tk.Label(post_frame, text="Description").pack(pady=5)
    desc_entry = tk.Text(post_frame, height=5)
```

```
desc_entry.pack()

tk.Label(post_frame, text="Salary").pack(pady=5)
salary_entry = tk.Entry(post_frame)
salary_entry.pack()

tk.Label(post_frame, text="Location").pack(pady=5)
location_entry = tk.Entry(post_frame)
location_entry.pack()

def post():
    title = title_entry.get()
    description = desc_entry.get("1.0", tk.END).strip()
    salary = salary_entry.get()
    location = location_entry.get()

    if not title or not description or not salary or not location:
        messagebox.showerror("Error", "All fields are required!")
        return

    try:
        db = connect_db()
        cursor = db.cursor()
        cursor.execute("""
            INSERT INTO posts (title, description, salary, location, email)
            VALUES (%s, %s, %s, %s, %s)
        """, (title, description, salary, location, admin_email))
        db.commit()
        db.close()

        messagebox.showinfo("Success", "Job posted successfully!")
        title_entry.delete(0, tk.END)
        desc_entry.delete("1.0", tk.END)
        salary_entry.delete(0, tk.END)
        location_entry.delete(0, tk.END)

    except mysql.connector.Error as err:
```

```
            messagebox.showerror("Database Error", f"Error: {err}")

    tk.Button(post_frame, text="Post Job", command=post,
bg="#4CAF50", fg="white", font=("Helvetica", 12)).pack(pady=15)

    app_frame = tk.Frame(notebook)
    notebook.add(app_frame, text="View Applications")

    tk.Label(app_frame, text="Applications", font=("Helvetica",
16)).pack(pady=10)

    app_columns = ("Application ID", "Student Email", "Job Title",
"Status/Reply")
    app_tree = ttk.Treeview(app_frame, columns=app_columns,
show='headings')
    for col in app_columns:
        app_tree.heading(col, text=col)
        app_tree.column(col, anchor=tk.W, width=180)
    app_tree.pack(expand=True, fill='both', padx=10, pady=10)

    def refresh_applications():
        for row in app_tree.get_children():
            app_tree.delete(row)

        db = connect_db()
        cursor = db.cursor()
        cursor.execute("""
            SELECT a.id, a.student_email, p.title, a.reply
            FROM applications a
            JOIN posts p ON a.job_id = p.id
            ORDER BY a.id DESC
        """)
        applications = cursor.fetchall()
        db.close()
        for app_row in applications:
            app_tree.insert('', 'end', values=app_row)
```

```
    refresh_applications()

    def update_status():
        selected = app_tree.focus()
        if not selected:
            messagebox.showerror("Error", "No application selected")
            return
        app_id, student_email, job_title, status =
app_tree.item(selected)['values']
        new_status = simpledialog.askstring("Update Status", "Enter new
status (e.g. Pending, Accepted, Rejected):", initialvalue=status)
        if new_status:
            try:
                db = connect_db()
                cursor = db.cursor()
                cursor.execute("UPDATE applications SET reply=%s WHERE
id=%s", (new_status, app_id))
                db.commit()
                db.close()
                log_activity(f"Application {app_id} status updated to
{new_status}")
                refresh_applications()
                messagebox.showinfo("Success", "Application status updated")
            except mysql.connector.Error as err:
                messagebox.showerror("Error", f"Database error: {err}")

    def send_reply_email():
        selected = app_tree.focus()
        if not selected:
            messagebox.showerror("Error", "No application selected")
            return
        app_id, student_email, job_title, status =
app_tree.item(selected)['values']

        subject = f"Application Update for {job_title}"
        body = simpledialog.askstring("Email Reply", "Enter your message to
the applicant:")
```

```
        if not body:
            messagebox.showinfo("Cancelled", "Email sending cancelled.")
            return

        if not admin_email:
            messagebox.showerror("Error", "Admin email is required to send
email.")
            return

        send_email(admin_email, student_email, subject, body)
        messagebox.showinfo("Email Sent", f"Reply sent to {student_email}
successfully.")

        try:
            db = connect_db()
            cursor = db.cursor()
            cursor.execute("UPDATE applications SET reply=%s WHERE
id=%s", (body, app_id))
            db.commit()
            db.close()
            log_activity(f"Sent email reply to {student_email} for application
{app_id} with reply: {body}")
            refresh_applications()
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Database error: {err}")

    btn_frame = tk.Frame(app_frame)
    btn_frame.pack(pady=10)

    tk.Button(btn_frame, text="Update Selected Status",
command=update_status, bg="#2196F3", fg="white", font=("Helvetica",
12)).pack(side=tk.LEFT, padx=10)
    tk.Button(btn_frame, text="Send Email Reply",
command=send_reply_email, bg="#FFA500", fg="black",
font=("Helvetica", 12)).pack(side=tk.LEFT, padx=10)

    student_frame = tk.Frame(notebook)
```

```
    notebook.add(student_frame, text="Manage Students")

    tk.Label(student_frame, text="Registered Students", font=("Helvetica",
16)).pack(pady=10)

    student_columns = ("Student ID", "Name", "Email")
    student_tree = ttk.Treeview(student_frame, columns=student_columns,
show='headings')
    for col in student_columns:
        student_tree.heading(col, text=col)
        student_tree.column(col, anchor=tk.W, width=200)
    student_tree.pack(expand=True, fill='both', padx=10, pady=10)

    def refresh_students():
        for row in student_tree.get_children():
            student_tree.delete(row)

        db = connect_db()
        cursor = db.cursor()
        cursor.execute("SELECT id, name, email FROM students ORDER
BY id DESC")
        students = cursor.fetchall()
        db.close()
        for student in students:
            student_tree.insert('', 'end', values=student)

    refresh_students()

    def delete_student():
        selected = student_tree.focus()
        if not selected:
            messagebox.showerror("Error", "No student selected")
            return
        student_id, name, email = student_tree.item(selected)['values']
        confirm = messagebox.askyesno("Confirm Delete", f"Delete student
{name} ({email})?")
        if confirm:
```

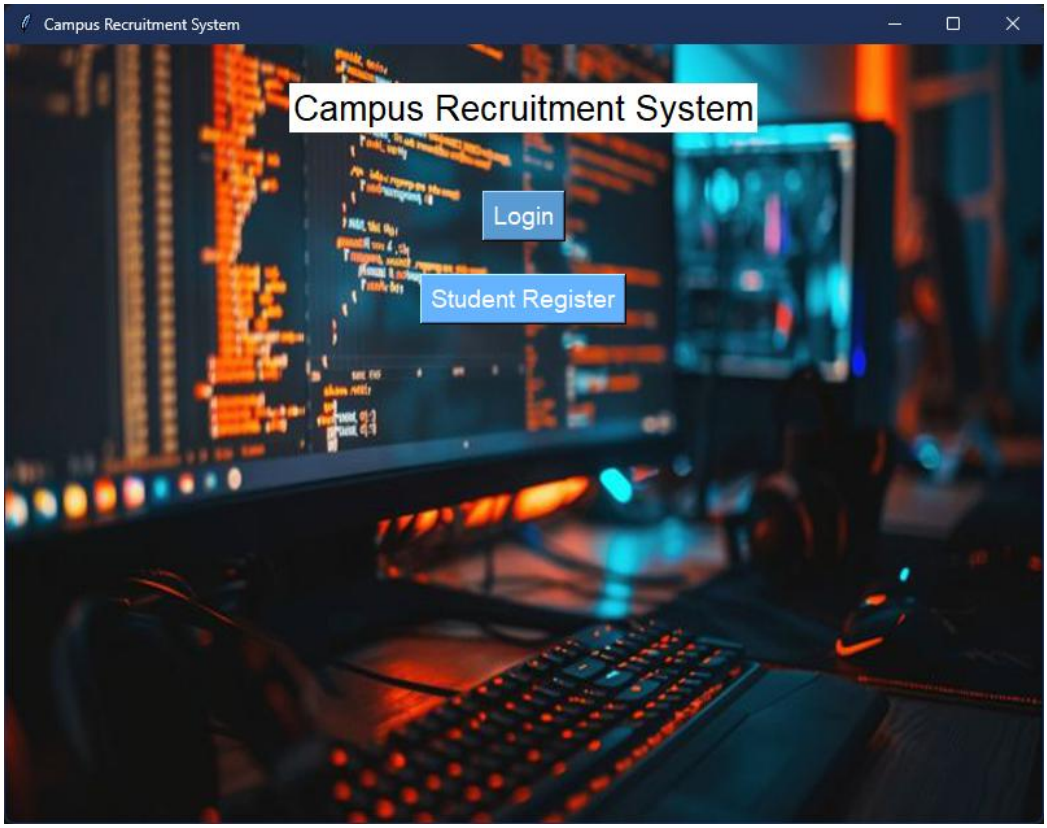
```
        try:
            db = connect_db()
            cursor = db.cursor()
            cursor.execute("DELETE FROM students WHERE id=%s",
(student_id,))
            db.commit()
            db.close()
            log_activity(f"Deleted student {email}")
            refresh_students()
            messagebox.showinfo("Success", "Student deleted")
        except mysql.connector.Error as err:
            messagebox.showerror("Error", f"Database error: {err}")

    tk.Button(student_frame, text="Delete Selected Student",
command=delete_student, bg="#f44336", fg="white", font=("Helvetica",
12)).pack(pady=10)

def student_register():
    name = simpledialog.askstring("Register", "Enter Name")
    email = simpledialog.askstring("Register", "Enter Email")
    password = simpledialog.askstring("Register", "Enter Password",
show='*')
    db = connect_db()
    cursor = db.cursor()
    try:
        cursor.execute("INSERT INTO students (name, email, password)
VALUES (%s, %s, %s)", (name, email, password))
        db.commit()
        log_activity(f"Student {email} registered")
        messagebox.showinfo("Success", "Registration Successful")
    except mysql.connector.errors.IntegrityError:
        messagebox.showerror("Error", "Email already registered")
    db.close()

def student_dashboard(email):
    win = tk.Toplevel(root)
    win.title("Available Jobs & Your Applications")
```

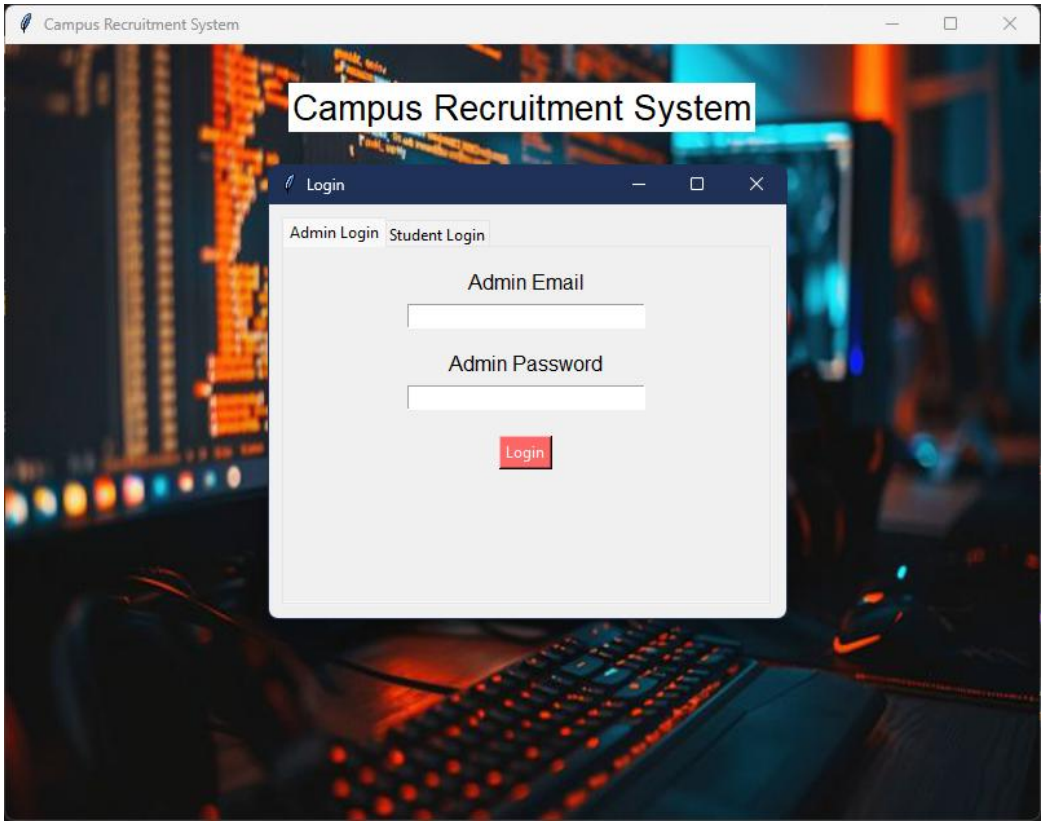
```
    win.geometry("800x600")

    jobs_frame = tk.LabelFrame(win, text="Available Jobs")
    jobs_frame.pack(fill='both', expand=True, padx=10, pady=10)

    db = connect_db()
    cursor = db.cursor()
    cursor.execute("SELECT id, title, email FROM posts")
    jobs = cursor.fetchall()
    db.close()

    tree = ttk.Treeview(jobs_frame, columns=("ID", "Title", "Posted By"),
show='headings')
    tree.heading("ID", text="ID")
    tree.heading("Title", text="Job Title")
    tree.heading("Posted By", text="Posted By")
    tree.pack(expand=True, fill='both', side=tk.LEFT, padx=5, pady=5)

    for job in jobs:
        tree.insert('', 'end', values=job)

    def apply():
        selected = tree.focus()
        if not selected:
            return
        job_id = tree.item(selected)['values'][0]
        db2 = connect_db()
        cursor2 = db2.cursor()
        cursor2.execute("INSERT INTO applications (student_email, job_id,
reply) VALUES (%s, %s, %s)",
                        (email, job_id, 'Pending'))
        db2.commit()
        db2.close()

        subject = f"Application Submitted for Job ID {job_id}"
```

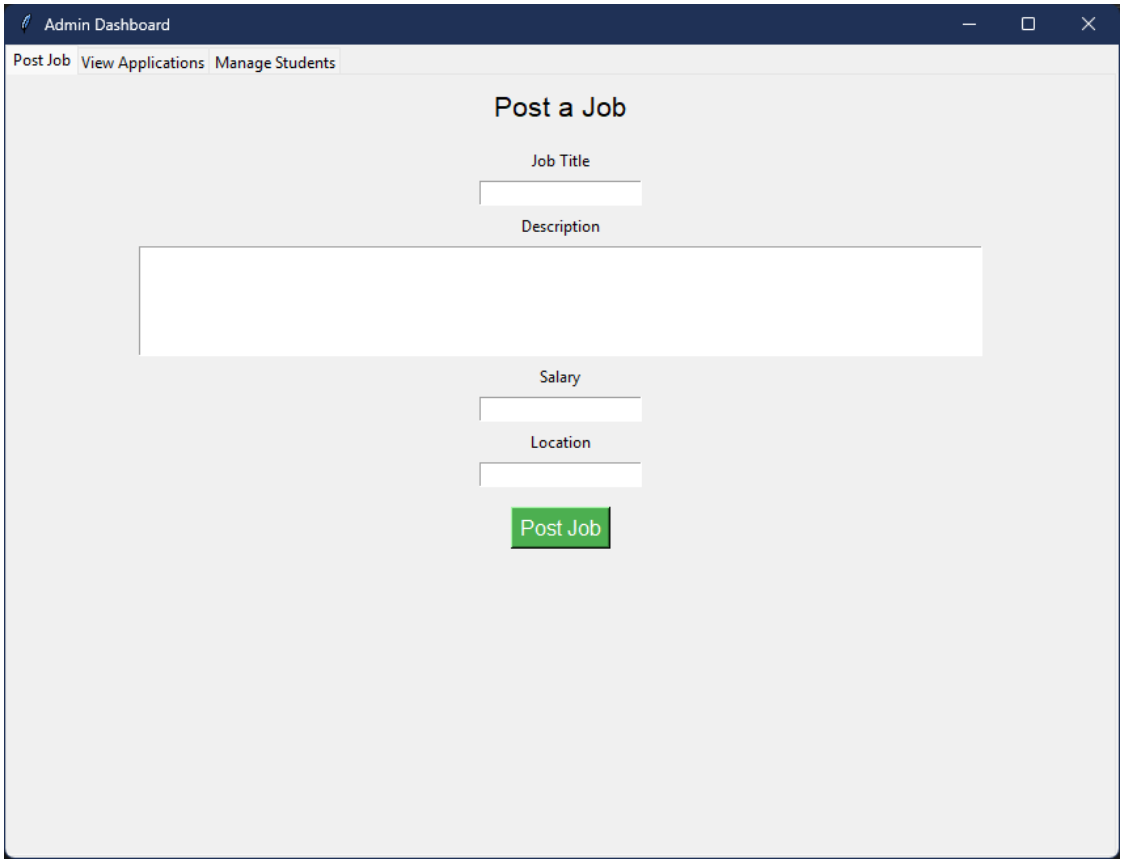
```
        body = f"Dear Student,\n\nYou have successfully applied for the job with ID {job_id}. Your application status is currently 'Pending'.\n\nBest regards,\nCampus Recruitment Team"

        if admin_email:
            send_email(admin_email, email, subject, body)
        else:
            messagebox.showwarning("Warning", "Admin email not set, email not sent.")

        log_activity(f"{email} applied for job {job_id}")
        messagebox.showinfo("Applied", "Application Sent")
        refresh_applications()

    tk.Button(jobs_frame, text="Apply for Selected Job", command=apply).pack(side=tk.RIGHT, padx=10, pady=10)

    apps_frame = tk.LabelFrame(win, text="Your Applications & Status")
    apps_frame.pack(fill='both', expand=True, padx=10, pady=10)

    app_columns = ("Application ID", "Job Title", "Status/Reply")
    app_tree = ttk.Treeview(apps_frame, columns=app_columns, show='headings')
    for col in app_columns:
        app_tree.heading(col, text=col)
        app_tree.column(col, anchor=tk.W, width=200)
    app_tree.pack(expand=True, fill='both', padx=5, pady=5)

    def refresh_applications():
        for row in app_tree.get_children():
            app_tree.delete(row)
        db3 = connect_db()
        cursor3 = db3.cursor()
        cursor3.execute("""
            SELECT a.id, p.title, a.reply
            FROM applications a
            JOIN posts p ON a.job_id = p.id
```

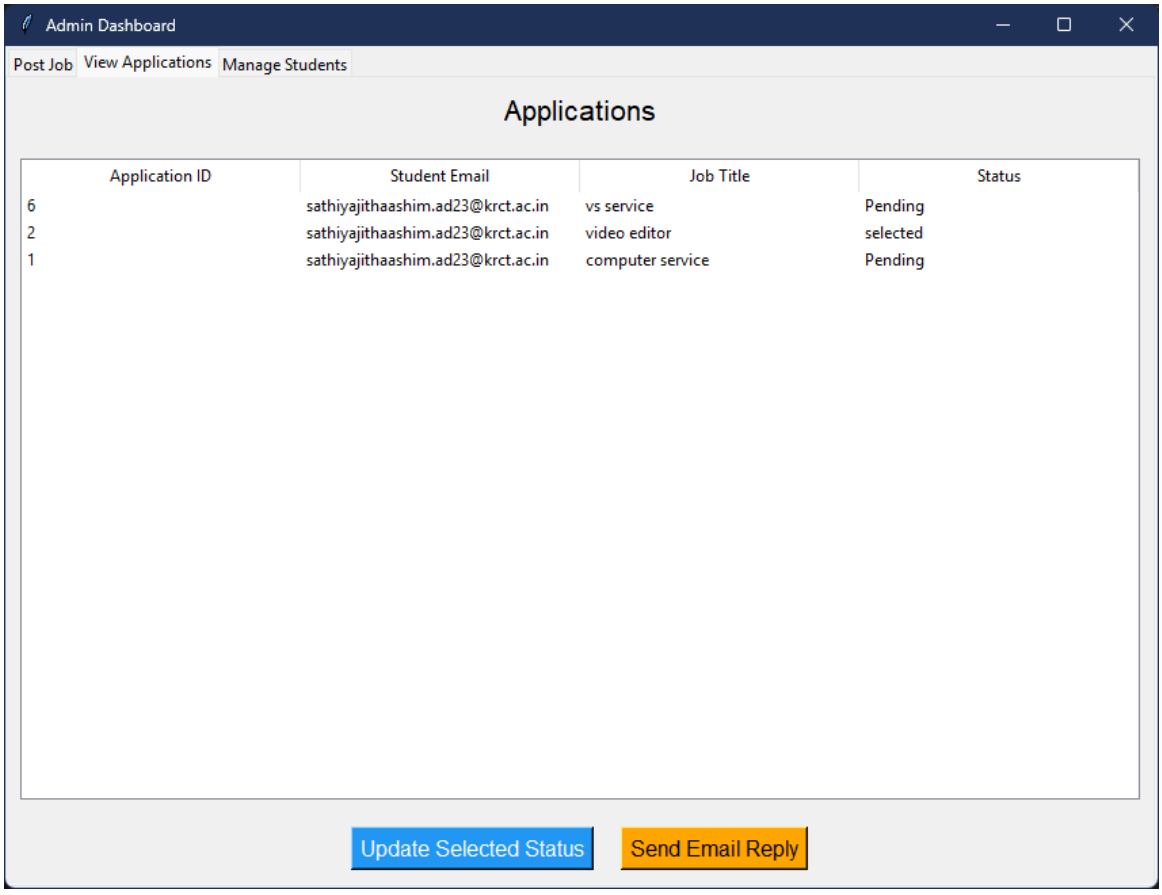
```
            WHERE a.student_email = %s
            ORDER BY a.id DESC
        """, (email,))
        applications = cursor3.fetchall()
        db3.close()
        for app in applications:
            app_tree.insert('', 'end', values=app)

    refresh_applications()

root = tk.Tk()
root.title("Campus Recruitment System")
root.geometry("800x600")

bg_img = Image.open(r"D:\Nothing\COLLEGE\4
SEM\DBMS\project\background.png")
bg_img = bg_img.resize((800, 600))
bg = ImageTk.PhotoImage(bg_img)
bg_label = tk.Label(root, image=bg)
bg_label.place(relwidth=1, relheight=1)

tk.Label(root, text="Campus Recruitment System", font=("Helvetica",
20), bg="#ffffff").pack(pady=30)

tk.Button(root, text="Login", command=open_login_window,
bg="#5a9bd4", fg="white", font=("Helvetica", 14)).pack(pady=15)
tk.Button(root, text="Student Register", command=student_register,
bg="#66b3ff", fg="white", font=("Helvetica", 14)).pack(pady=10)

root.mainloop()
```

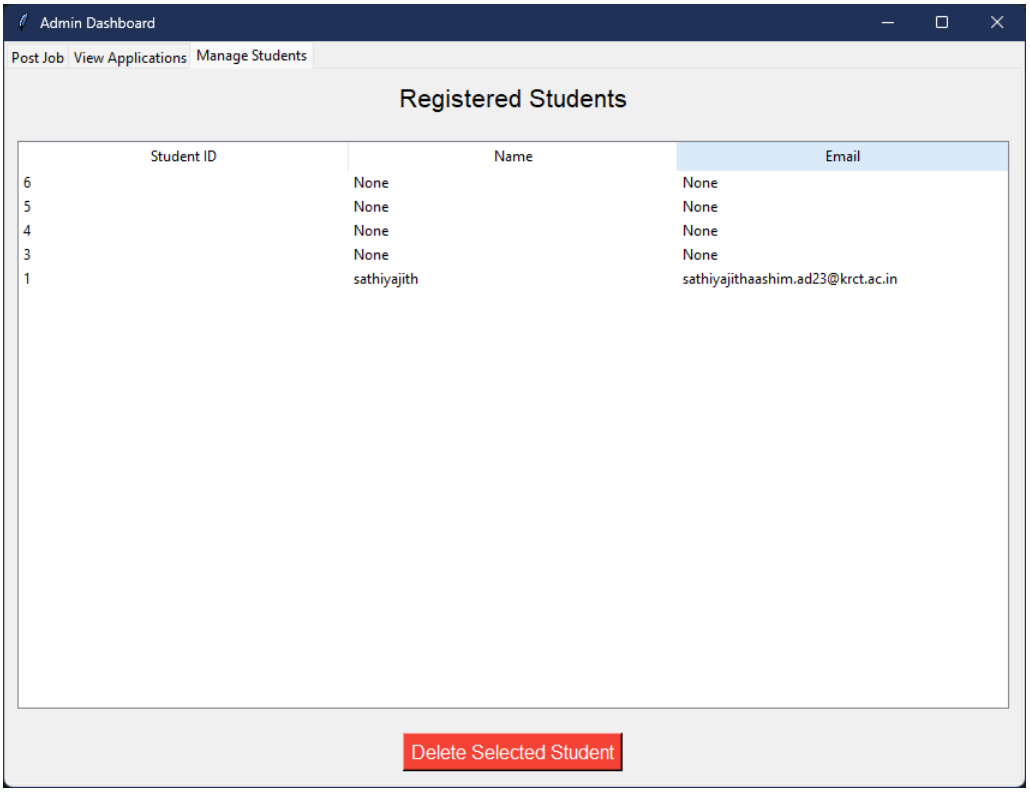
APPENDIX B SCREENSHOTS

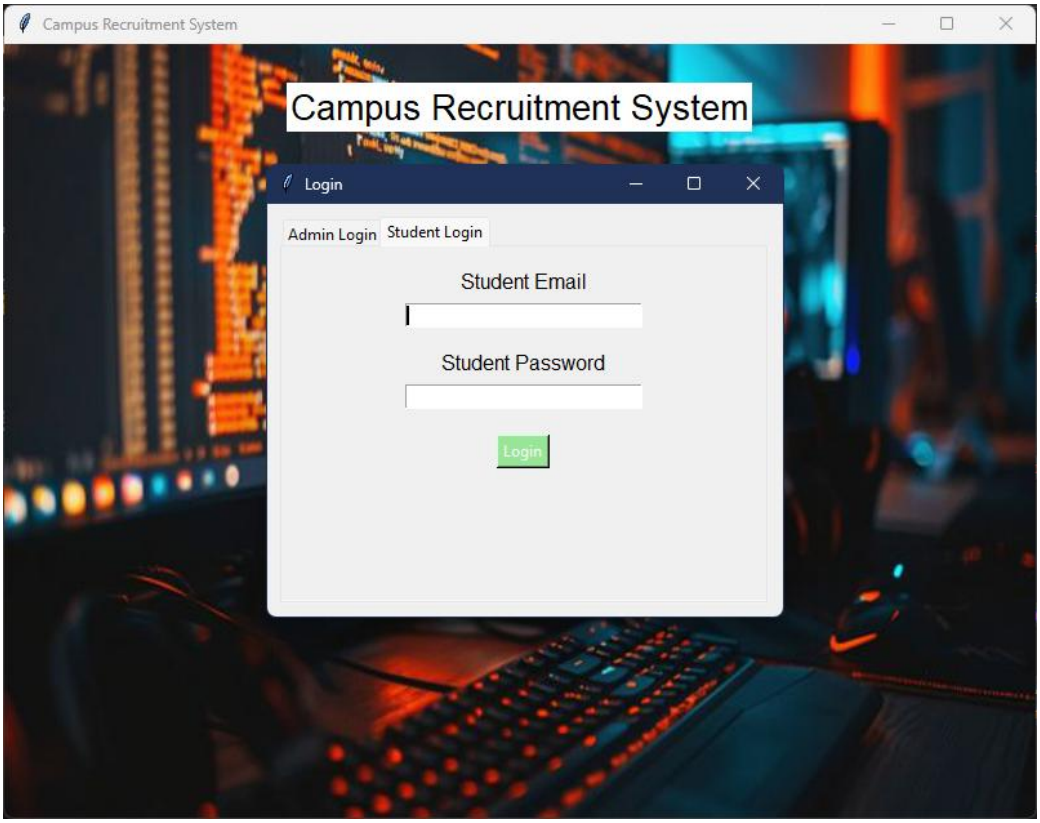


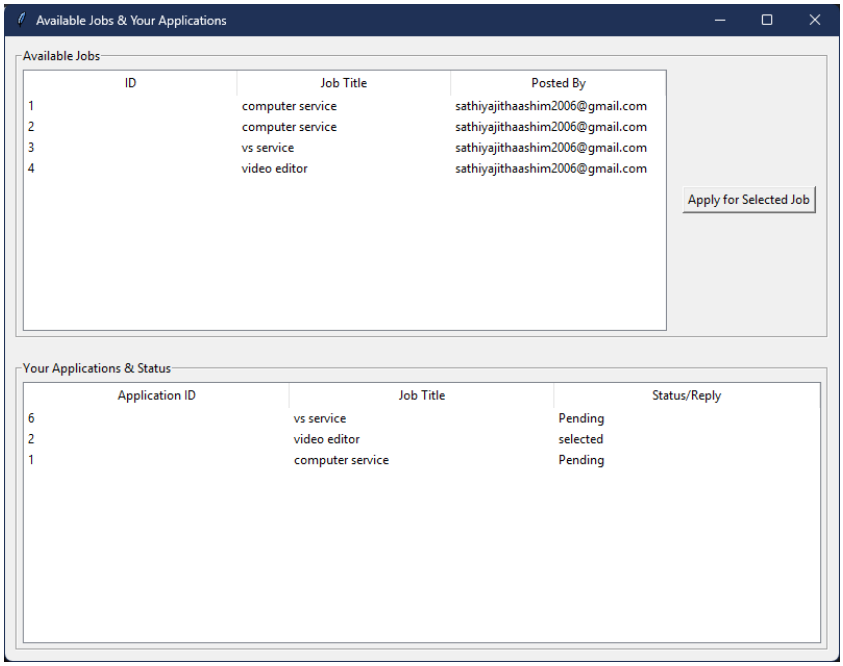


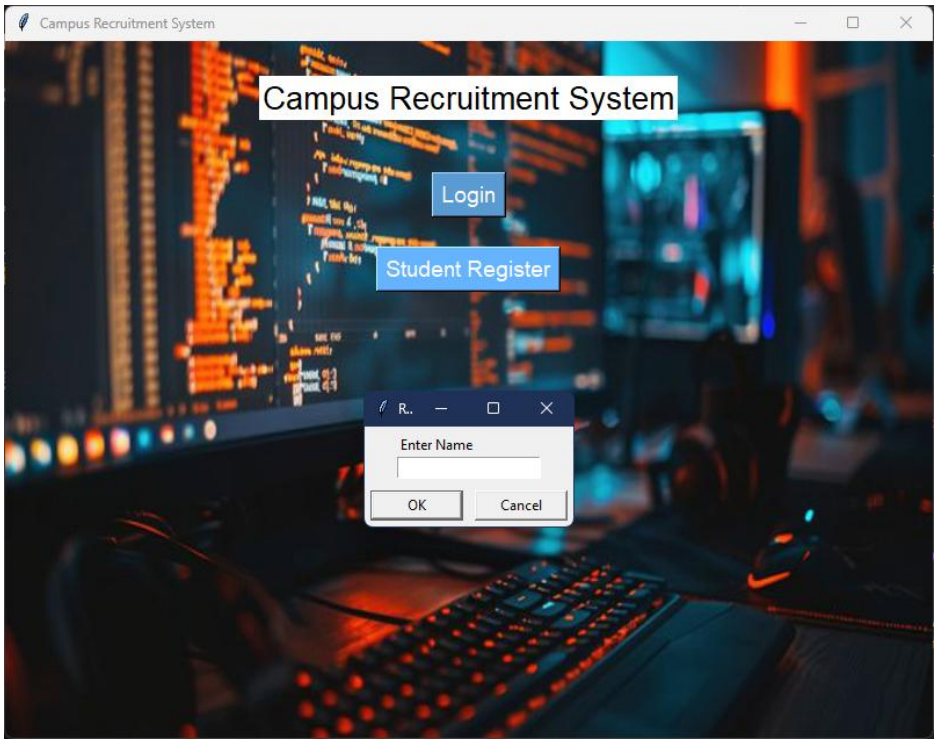

Admin Dashboard
Post Job
View Applications
Manage Students
Post a Job
Job Title
Description
Salary
Location
Post Job

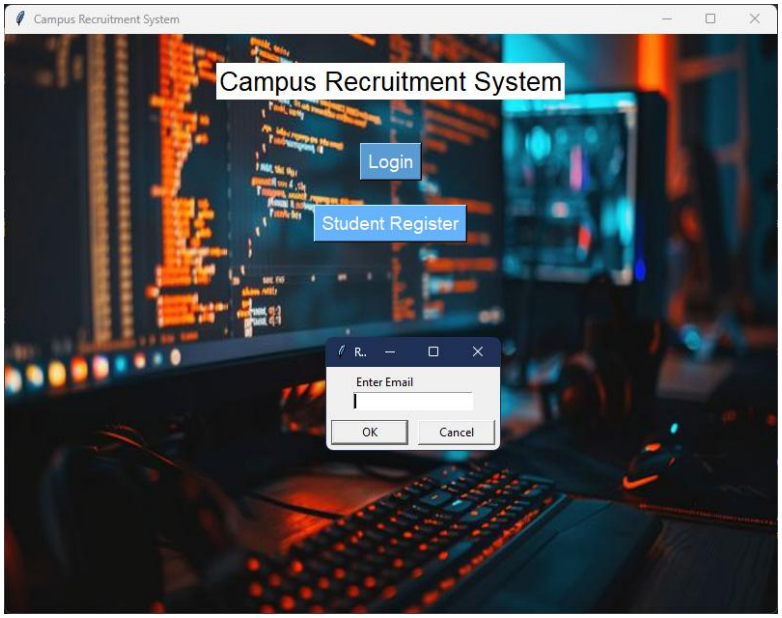

Admin Dashboard
Post Job
View Applications
Manage Students
Applications
Application ID
Student Email
Job Title
Status
6
sathiyajithaashim.ad23@krct.ac.in
vs service
Pending
2
sathiyajithaashim.ad23@krct.ac.in
video editor
selected
1
sathiyajithaashim.ad23@krct.ac.in
computer service
Pending
Update Selected Status
Send Email Reply

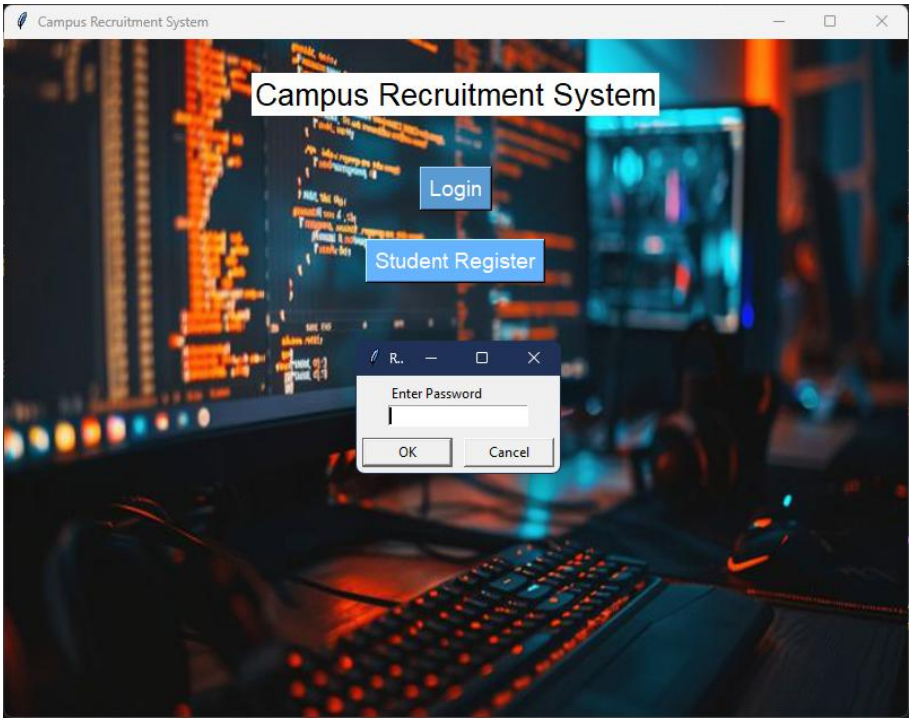

Admin Dashboard
Post Job
View Applications
Manage Students
Registered Students
Student ID
Name
Email
6
None
None
5
None
None
4
None
None
3
None
None
1
sathiyajith
sathiyajithaashim.ad23@krct.ac.in
Delete Selected Student

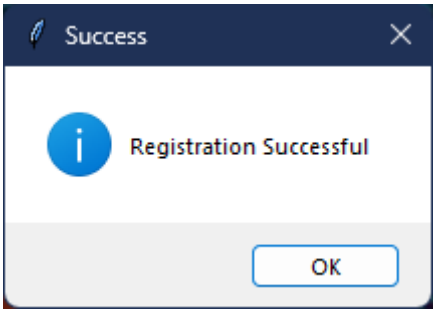

Campus Recruitment System
Campus Recruitment System
Login
Admin Login
Student Login
Student Email
Student Password
Login


Available Jobs & Your Applications
Available Jobs
ID
Job Title
Posted By
1
computer service
sathiyajithaashim2006@gmail.com
2
computer service
sathiyajithaashim2006@gmail.com
3
vs service
sathiyajithaashim2006@gmail.com
4
video editor
sathiyajithaashim2006@gmail.com
Apply for Selected Job
Your Applications & Status
Application ID
Job Title
Status/Reply
6
vs service
Pending
2
video editor
selected
1
computer service
Pending


Campus Recruitment System
Campus Recruitment System
Login
Student Register
R..
Enter Name
OK
Cancel


Campus Recruitment System
Campus Recruitment System
Login
Student Register
R..
Enter Email
OK
Cancel


Campus Recruitment System
Campus Recruitment System
Login
Student Register
R..
Enter Password
OK
Cancel


Success
Registration Successful
OK

REFERENCES

1. Python Official Documentation
   https://docs.python.org/3/
   Used for understanding Python syntax, libraries (Tkinter, smtplib), and programming concepts.
2. Tkinter GUI Programming
   Grasped GUI design principles and widget usage for building user-friendly desktop applications.
   https://docs.python.org/3/library/tkinter.html
3. MySQL Documentation
   https://dev.mysql.com/doc/
   Referred for database creation, SQL queries, data types, and connection setup with Python.
4. W3Schools SQL Tutorial
   https://www.w3schools.com/sql/
   Helped in building and refining SQL commands like SELECT, INSERT, JOIN, and UPDATE.
5. GeeksforGeeks Python Tutorials
   https://www.geeksforgeeks.org/python-programming-language/
   Used for understanding Python concepts, Tkinter examples, and MySQL-Python integration.
6. SMTP and Email Handling in Python
   https://realpython.com/python-send-email/
   Referenced to integrate email functionality for admin replies using SMTP and Gmail.
7. Stack Overflow
   https://stackoverflow.com/

Used to resolve specific programming issues, bugs, and get suggestions from the developer community.

8. XAMPP for MySQL Setup
   https://www.apachefriends.org/index.html
   Utilized for local MySQL server setup and database management.