Mini project report on

University Placement Management System

Submitted in partial fulfilment of the requirements for the award of degree of

Bachelor of Technology

in

Computer Science & Engineering

UE22CS351A – DBMS Project

Submitted by:

Bhavesh Budharaju	**PES2UG22CS129**
Dhanush Suresh Jettipalle	**PES2UG22CS175**

Under the guidance of
Prof. Mannar Mannan
Associate Professor
PES University

AUG - DEC 2024

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

PES UNIVERSITY

(Established under Karnataka Act No. 16 of 2013)
Electronic City, Hosur Road, Bengaluru – 560 100, Karnataka, India

CERTIFICATE

This is to certify that the mini project entitled

University Placement Management System

is a bonafide work carried out by

Bhavesh Budharaju	**PES2UG22CS129**
Dhanush Suresh Jettipalle	**PES2UG22CS175**

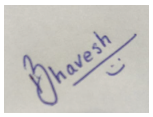
In partial fulfilment for the completion of fifth semester DBMS Project (UE22CS351A) in the Program of Study - Bachelor of Technology in Computer Science and Engineering under rules and regulations of PES University, Bengaluru during the period AUG. 2024 – DEC. 2024. It is certified that all corrections / suggestions indicated for internal assessment have been incorporated in the report. The project has been approved as it satisfies the 5th semester academic requirements in respect of project work.

Signature
Prof. Mannar Mannan
Associate Professor

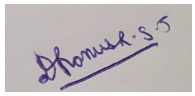
DECLARATION

We hereby declare that the DBMS Project entitled **University Placement Management System** has been carried out by us under the guidance of **Prof. Mannar Mannan, Associate Professor** and submitted in partial fulfilment of the course requirements for the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** of **PES University, Bengaluru** during the academic semester AUG – DEC 2024.

Bhavesh Budharaju **PES2UG22CS129**



Dhanush Suresh Jettipalle **PES2UG22CS175**



ABSTRACT

University Placement Management System is a web-based placement management system, developed using MySQL and Python-Streamlit, that simulates the functionalities of a recruitment platform. Designed as an academic project for a DBMS course, it provides a structured approach to handling student, company, and admin interactions within a placement system. The project implements multi-user authentication, role-specific dashboards, and a real-time email notification feature. By utilizing a modular code structure, this project offers students a hands-on understanding of database operations, web app deployment, and secure data management in a placement context.

TABLE OF CONTENTS

1. INTRODUCTION

University Placement Management System addresses the complexity and demands of managing recruitment activities in educational institutions by replicating the core functionalities of a placement management platform. Current manual processes in campus placements can lead to inefficiencies and errors, especially when handling large volumes of student applications and job listings. This Project offers a scalable and organized system where different user roles—students, companies, and admins—can perform role-specific actions securely and effectively. Students can edit their profiles, track their applications, and manage skill information, while companies can post jobs, shortlist candidates, and manage applications. Administrators play a supervisory role, overseeing system metrics and user management, and they receive real-time insights into the system's functioning. This Project is built using MySQL for data storage and retrieval, ensuring robust data handling, while Python-Streamlit provides a responsive, user-friendly interface. The project uses core database management concepts like triggers, stored procedures, and secure password hashing, integrating these into a real-world, application-based project.

2a . PROBLEM DEFINITION

Managing placement activities in educational institutions can be a challenging task, especially when the process relies heavily on manual coordination. Universities often face difficulties in managing student applications, job postings, and tracking candidate progress due to the complexity and sheer volume of data. These manual processes not only lead to inefficiencies, such as missed opportunities or errors in candidate shortlisting, but also lack transparency, which can create confusion among students, companies, and administrators. As a result, many placement cells struggle to streamline operations, making the recruitment process slower and prone to mistakes.

This Project addresses this issue by automating and simplifying the entire placement management process. This web-based system provides a structured approach where students, companies, and administrators can interact efficiently. By offering role-based functionalities and secure data handling, the system reduces the chances of errors and enhances the overall management of recruitment activities. Moreover, the use of modern technologies like MySQL for database management and Python-Streamlit for the interface ensures that the system is scalable, easy to use, and effective in streamlining the entire placement process.

2b . USER REQUIREMENT SPECIFICATIONS

FUNCTIONAL REQUIREMENTS :

1. The University Placement Management platform is designed to cater to three distinct user roles, each with specific requirements. Students need an easy-to-use interface where they can manage their profiles, update their skill sets, and apply for job postings. They also require the ability to track the status of their applications and receive email notifications about updates, such as whether they have been shortlisted or rejected. Additionally, the system must protect sensitive student data.

2. Companies, on the other hand, need a dashboard that allows them to manage their company profiles, post job openings, and view applications from interested students. They should be able to filter and shortlist candidates based on their qualifications and skills.

3. Administrators are responsible for overseeing the entire system. They need features for managing user accounts (both students and companies) and monitoring the overall placement management. Admins should have access to detailed reports and metrics, such as total number of applications, companies, and students.

NON FUNCTIONAL REQUIREMENTS :

1. The system should be able to handle a large number of concurrent users (students,companies, and admins) without significant performance degradation.

2. Application load times should be minimal, and queries should be optimized for speed.

3. The system should implement secure authentication, using modern password hashing algorithms (e.g., Bcrypt) for user passwords.

4. The user interface should be responsive, ensuring usability on various devices like desktops, tablets, and mobile phones.

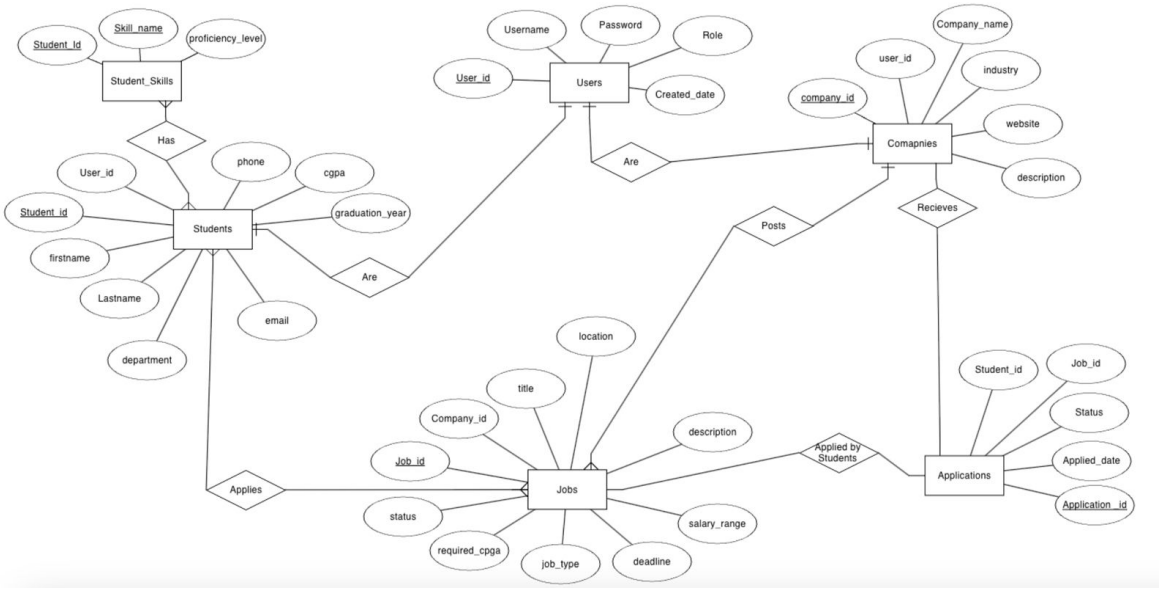
5. The code should be modular and well-documented to facilitate easy maintenance and updates by developers.

6. Regular backups should be made of the database and other critical data to prevent loss in case of a failure.

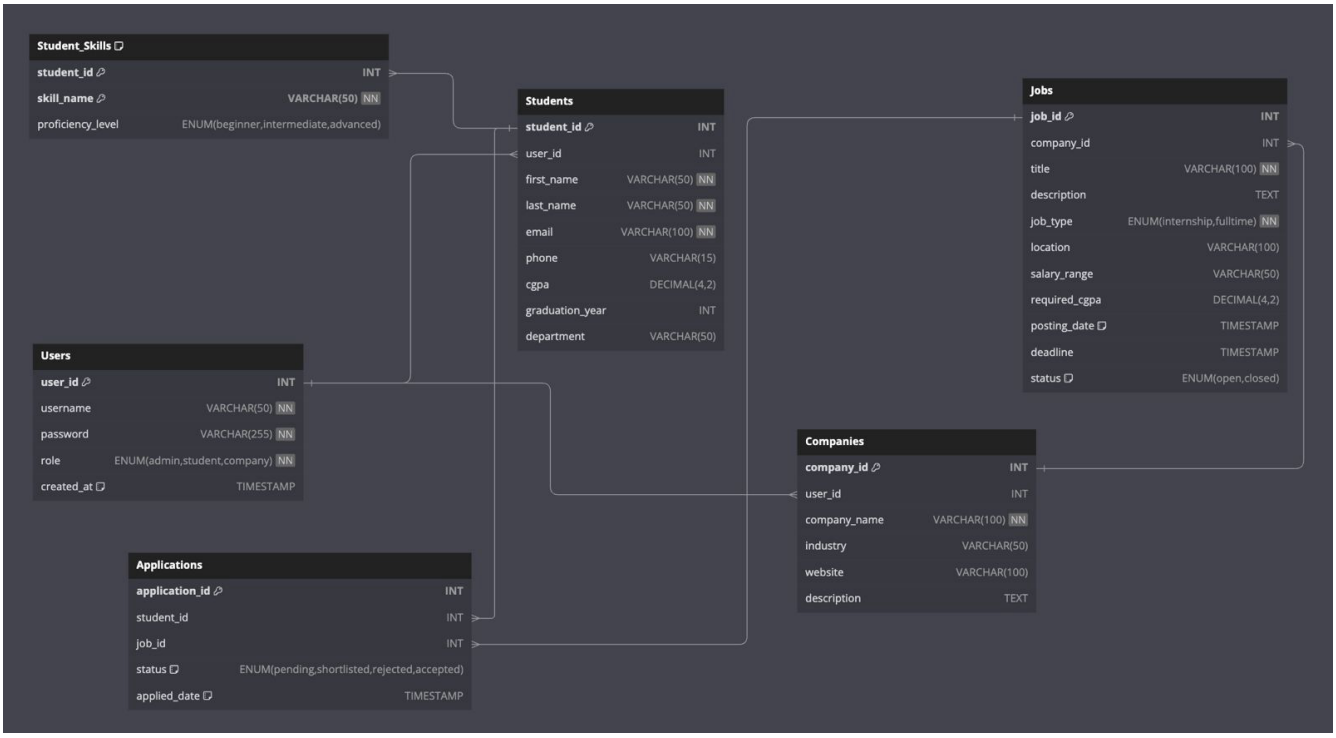
3. LIST OF SOFTWARES/TOOLS/PROGRAMMING LANGUAGES USED

- **Python 3.x**: Core language for application logic
- **Streamlit**: For creating the web-based interface
- **MySQL**: Backend database for data storage and retrieval
- **Bcrypt**: Secure password hashing for user authentication
- **dotenv**: To manage environment variables securely
- **Python MySQL Connector**: To facilitate database interactions from Python
- **SMTPlib and MIME**: Email Notifications
- **Visual Studio Code**: Code Editor
- **Git** : Version Control

4. ER MODEL


Skill_name
Student_Id
proficiency_level
Student_Skills
Has
User_id
phone
cgpa
Student_id
Students
graduation_year
firstname
Lastname
email
department
Are
Username
Password
Role
User_id
Users
Created_date
Are
Company_name
user_id
industry
company_id
website
Comapnies
description
Recieves
Posts
location
title
Company_id
description
Job_id
Jobs
Applies
status
salary_range
required_cpga
job_type
deadline
Applied by Students
Student_id
Job_id
Status
Applied_date
Applications
Application_id

5. ER MODEL TO RELATIONAL MAPPING



6. DDL STATEMENTS

- **Create Commands :**

```
```
CREATE TABLE IF NOT EXISTS Users (
    user_id INT PRIMARY KEY AUTO_INCREMENT,
    username VARCHAR(50) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    role ENUM('admin', 'student', 'company') NOT NULL,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

CREATE TABLE IF NOT EXISTS Students (
    student_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    first_name VARCHAR(50) NOT NULL,
    last_name VARCHAR(50) NOT NULL,
    email VARCHAR(100) UNIQUE NOT NULL,
    phone VARCHAR(15),
    cgpa DECIMAL(4,2),
    graduation_year INT,
    department VARCHAR(50),
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

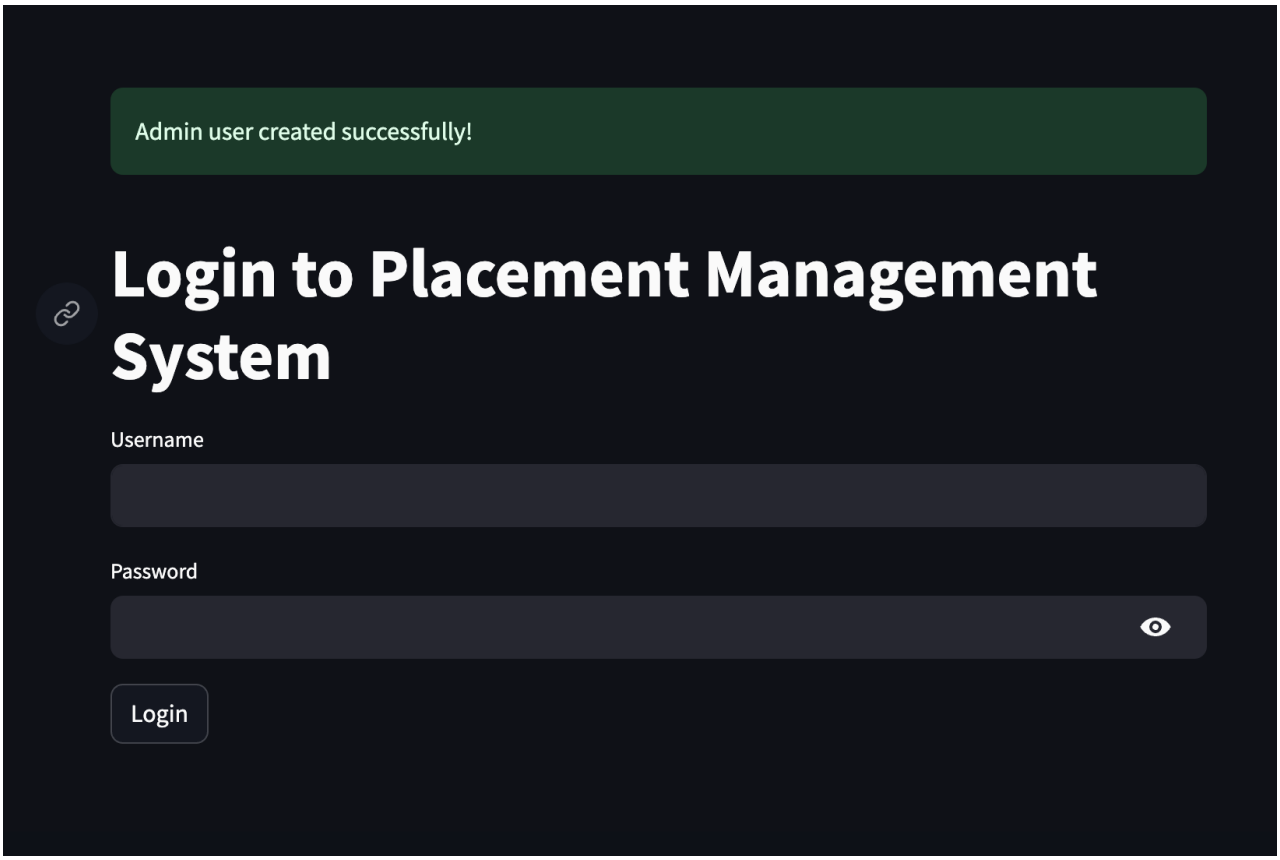
CREATE TABLE IF NOT EXISTS Companies (
    company_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT,
    company_name VARCHAR(100) NOT NULL,
    industry VARCHAR(50),
    website VARCHAR(100),
    description TEXT,
    FOREIGN KEY (user_id) REFERENCES Users(user_id)
);

CREATE TABLE IF NOT EXISTS Jobs (
    job_id INT PRIMARY KEY AUTO_INCREMENT,
    company_id INT,
    title VARCHAR(100) NOT NULL,
    description TEXT,
    job_type ENUM('internship', 'fulltime') NOT NULL,
    location VARCHAR(100),
    salary_range VARCHAR(50),
    required_cgpa DECIMAL(4,2),
    posting_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    deadline TIMESTAMP,
    status ENUM('open', 'closed') DEFAULT 'open',
    FOREIGN KEY (company_id) REFERENCES Companies(company_id)
```
```

```
);

CREATE TABLE IF NOT EXISTS Applications (
    application_id INT PRIMARY KEY AUTO_INCREMENT,
    student_id INT,
    job_id INT,
    status ENUM('pending', 'shortlisted', 'rejected', 'accepted') DEFAULT 'pending',
    applied_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (student_id) REFERENCES Students(student_id),
    FOREIGN KEY (job_id) REFERENCES Jobs(job_id)
);

CREATE TABLE IF NOT EXISTS Student_Skills (
    student_id INT,
    skill_name VARCHAR(50) NOT NULL,
    proficiency_level ENUM('beginner', 'intermediate', 'advanced'),
    PRIMARY KEY (student_id, skill_name),
    FOREIGN KEY (student_id) REFERENCES Students(student_id)
);
```

When The DB is initialized for the the first time , these commands are run and a user with role **admin**
username : admin
Password : admin123 is created.

7. DML STATEMENTS(CRUD)

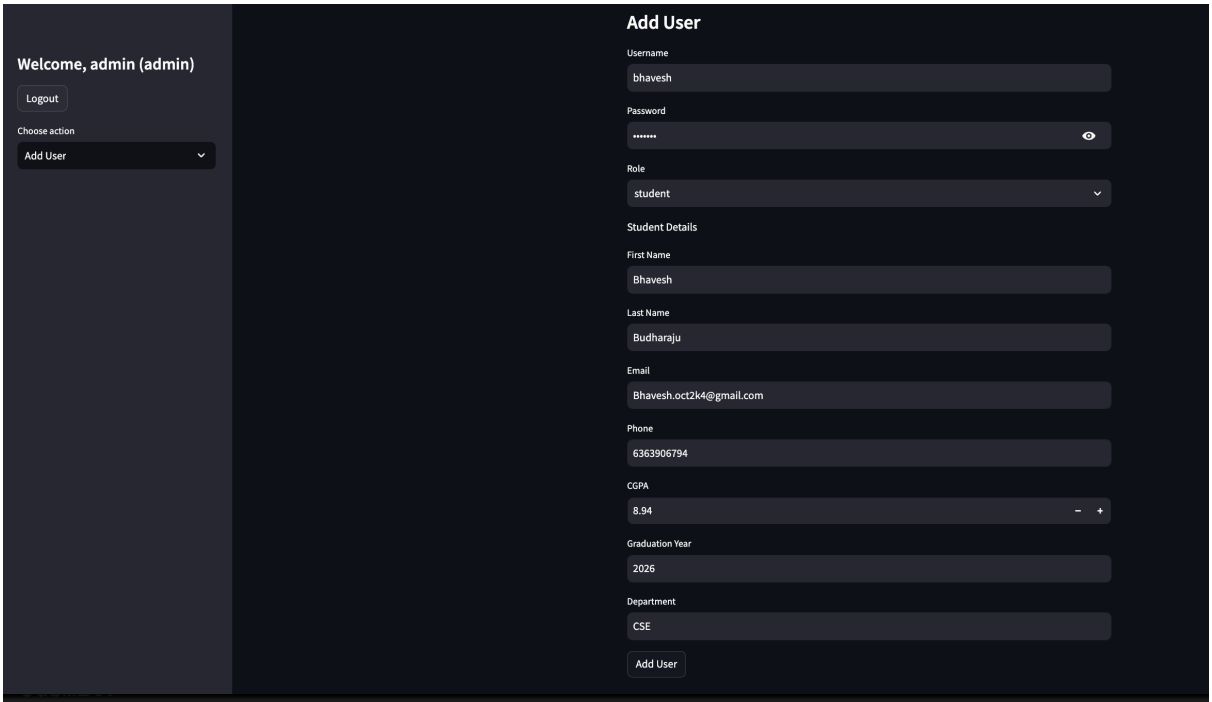


Admin user created on initial initialization of database with credentials
Username : admin
Password : admin123

```
cursor = conn.cursor()

# Check if admin exists
cursor.execute("SELECT COUNT(*) FROM Users WHERE username = 'admin'")
admin_exists = cursor.fetchone()[0]

if not admin_exists:
    # Create admin with hashed password
    hashed_password = bcrypt.hashpw('admin123'.encode('utf-8'), bcrypt.gensalt())

    cursor.execute("""
        INSERT INTO Users (username, password, role)
        VALUES (%s, %s, %s)
    """, ('admin', hashed_password.decode('utf-8'), 'admin'))

    conn.commit()
    st.success("Admin user created successfully!")
```

Insertion :

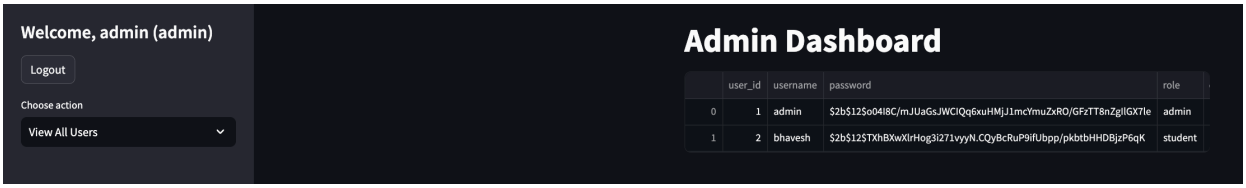
a. Inserting student from admin



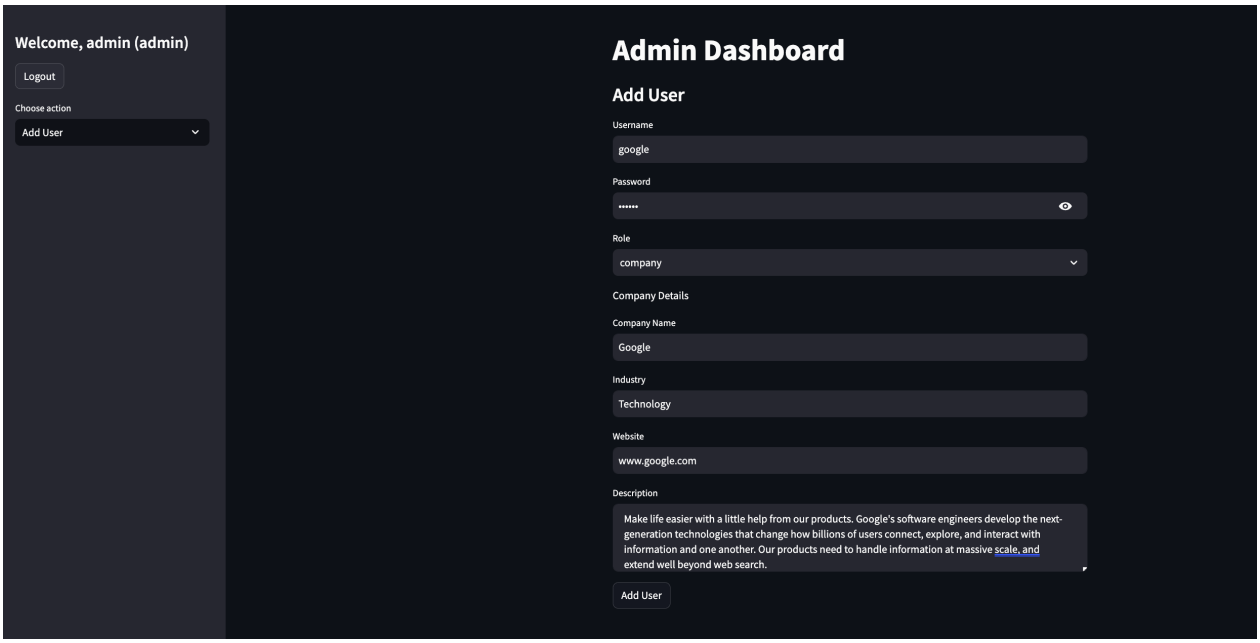

```
hashed_password = hash_password(password)

# Insert into Users table
cursor.execute("INSERT INTO Users (username, password, role) VALUES (%s, %s, %s)",
               (username, hashed_password, role))
conn.commit()
user_id = cursor.lastrowid  # Get the last inserted user_id
```

```
if role == "student":
    cursor.execute("""
        INSERT INTO Students (user_id, first_name, last_name, email, phone, cgpa, graduation_year, department)
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """, (user_id, additional_details['first_name'], additional_details['last_name'],
          additional_details['email'], additional_details['phone'],
          additional_details['cgpa'], additional_details['graduation_year'],
          additional_details['department']))
```

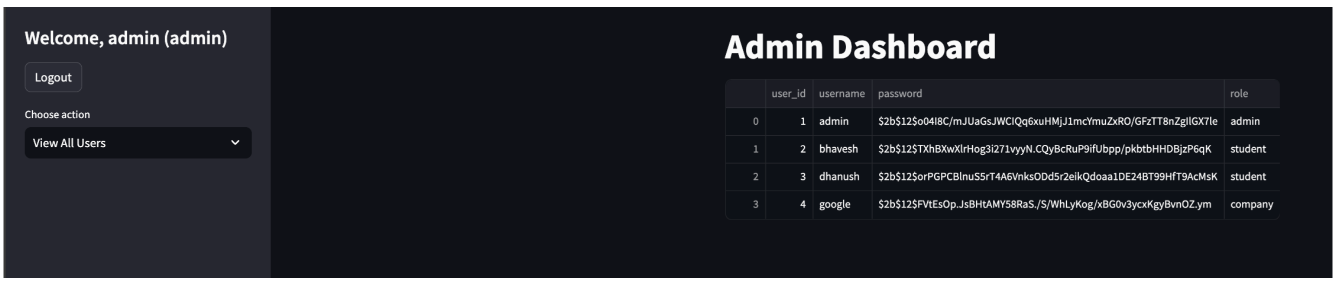
```
mysql> select * from users;
+---------+----------+--------------------------------------------------------------+---------+---------------------+
| user_id | username | password                                                     | role    | created_at          |
+---------+----------+--------------------------------------------------------------+---------+---------------------+
|       1 | admin    | $2b$12$so4I8C/mJUaGsJWCIQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgIlGX7le | admin   | 2024-11-12 09:22:50 |
|       2 | bhavesh  | $2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK | student | 2024-11-12 09:33:22 |
+---------+----------+--------------------------------------------------------------+---------+---------------------+
2 rows in set (0.00 sec)

mysql> select * from students;
+------------+---------+------------+-----------+-----------------------+------------+------+----------------+------------+
| student_id | user_id | first_name | last_name | email                 | phone      | cgpa | graduation_year | department |
+------------+---------+------------+-----------+-----------------------+------------+------+----------------+------------+
|          1 |       2 | Bhavesh    | Budharaju | Bhavesh.oct2k4@gmail.com | 6363906794 | 8.94 |           2026 | CSE        |
+------------+---------+------------+-----------+-----------------------+------------+------+----------------+------------+
1 row in set (0.00 sec)
```



b. Inserting company from admin

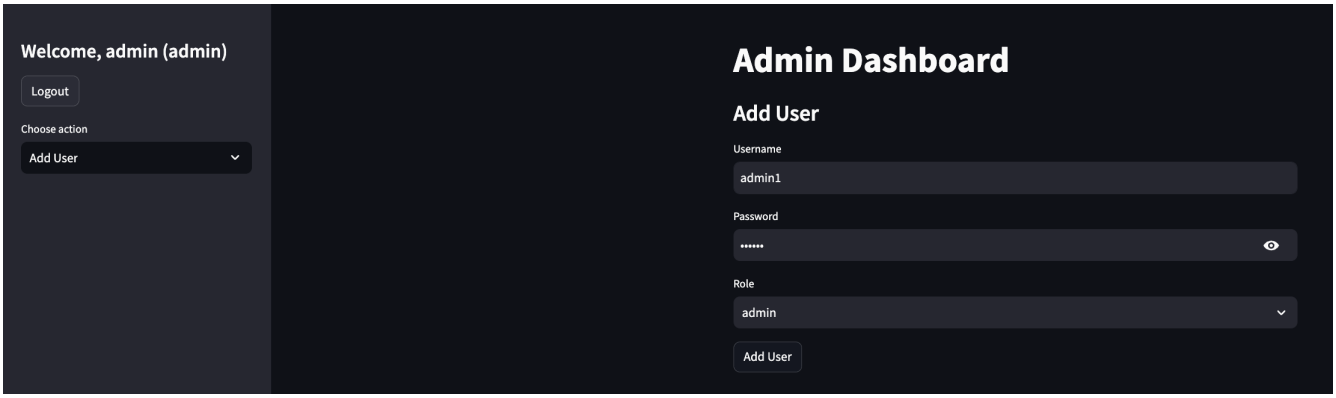


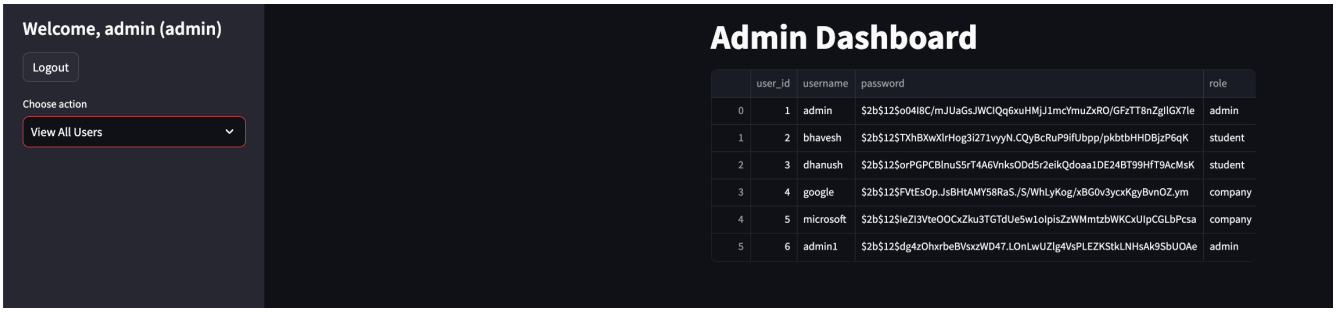
```python
hashed_password = hash_password(password)

# Insert into Users table
cursor.execute("INSERT INTO Users (username, password, role) VALUES (%s, %s, %s)",
               (username, hashed_password, role))
conn.commit()
user_id = cursor.lastrowid  # Get the last inserted user_id
elif role == "company":
    cursor.execute("""
        INSERT INTO Companies (user_id, company_name, industry, website, description)
        VALUES (%s, %s, %s, %s, %s)
    """, (user_id, additional_details['company_name'], additional_details['industry'],
          additional_details['website'], additional_details['description']))

conn.commit()
st.success(f"User {username} added successfully!")
```




```
mysql> select * from users;
+---------+----------+--------------------------------------------------------------+---------+---------------------+
| user_id | username | password                                                     | role    | created_at          |
+---------+----------+--------------------------------------------------------------+---------+---------------------+
|       1 | admin    | $2b$12$o04I8C/mJUaGsJWCIQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgIlGX7le | admin   | 2024-11-12 09:22:50 |
|       2 | bhavesh  | $2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK | student | 2024-11-12 09:33:22 |
|       3 | dhanush  | $2b$12$orPGPCBlnuS5rT4A6VnksODd5r2eikQdoaa1DE24BT99HfT9AcMsK | student | 2024-11-12 09:38:57 |
|       4 | google   | $2b$12$FVtEsOp.JsBHtAMY58RaS./S/WhLyKog/xBG0v3ycxKgyBvnOZ.ym | company | 2024-11-12 09:43:19 |
+---------+----------+--------------------------------------------------------------+---------+---------------------+
4 rows in set (0.00 sec)

mysql> select * from companies;
+------------+---------+--------------+------------+----------------+---------------------------------------------------
| company_id | user_id | company_name | industry   | website        | description
                                  |
+------------+---------+--------------+------------+----------------+---------------------------------------------------
|          1 |       4 | Google       | Technology | www.google.com | Make life easier with a little help from our products. Google's software engineers develop the next-genera
tion technologies that change how billions of users connect, explore, and interact with information and one another. Our products need to handle information at massive scale, a
nd extend well beyond web search. |
+------------+---------+--------------+------------+----------------+---------------------------------------------------
1 row in set (0.00 sec)
```

c. Inserting another admin from admin

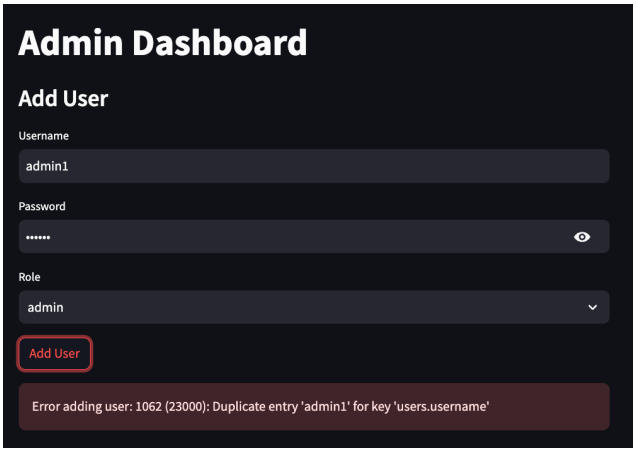




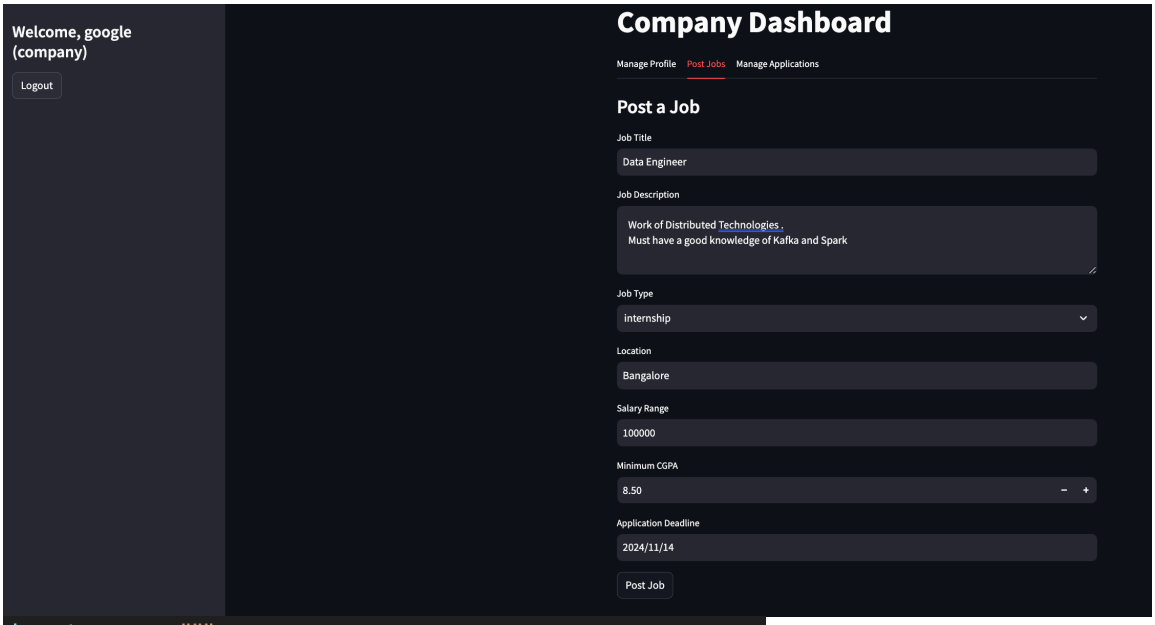

```
mysql> select * from users;
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
| user_id | username  | password                                                     | role    | created_at          |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
|       1 | admin     | $2b$12$o04I8C/mJUaGsJWCIQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgIlGX7le | admin   | 2024-11-12 09:22:50 |
|       2 | bhavesh   | $2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK | student | 2024-11-12 09:33:22 |
|       3 | dhanush   | $2b$12$orPGPCBlnuS5rT4A6VnksODd5r2eikQdoaa1DE24BT99HfT9AcMsK | student | 2024-11-12 09:38:57 |
|       4 | google    | $2b$12$FVtEsOp.JsBHtAMY58RaS./S/WhLyKog/xBG0v3ycxKgyBvnOZ.ym | company | 2024-11-12 09:43:19 |
|       5 | microsoft | $2b$12$IeZI3VteOOCxZku3TGTdUe5w1oIpisZzWMmtzbWKCxUIpCGLbPcsa | company | 2024-11-12 09:48:38 |
|       6 | admin1    | $2b$12$dg4zOhxrbeBVsxzWD47.LOnLwUZlg4VsPLEZKStkLNHsAk9SbUOAe | admin   | 2024-11-12 09:49:56 |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
6 rows in set (0.00 sec)
```

```python
hashed_password = hash_password(password)

# Insert into Users table
cursor.execute("INSERT INTO Users (username, password, role) VALUES (%s, %s, %s)",
               (username, hashed_password, role))
conn.commit()
user_id = cursor.lastrowid  # Get the last inserted user_id
```

What if User is already present in database and redundant user is inserted?

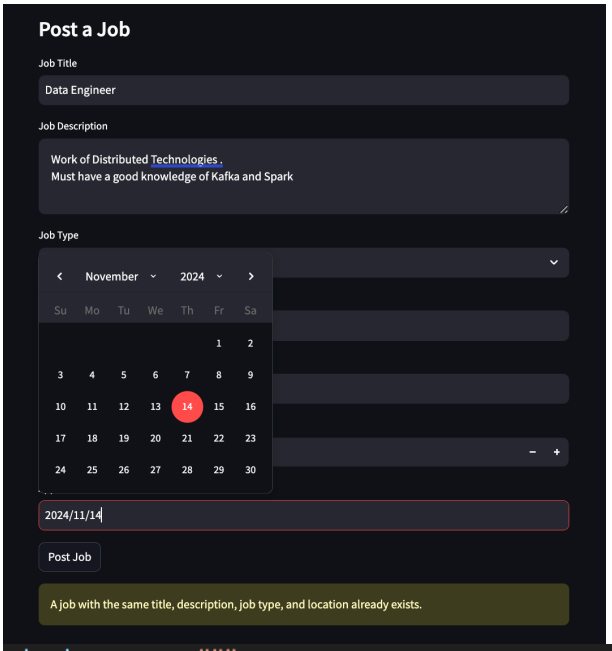


a. Insertion from company – post job




```
insert_query = """
    INSERT INTO Jobs (
        company_id, title, description, job_type,
        location, salary_range, required_cgpa, deadline
    ) VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
"""
cursor.execute(insert_query,
               (company_id, title, description, job_type,
                location, salary_range, required_cgpa, deadline))
conn.commit()
st.success("Job posted successfully!")
```

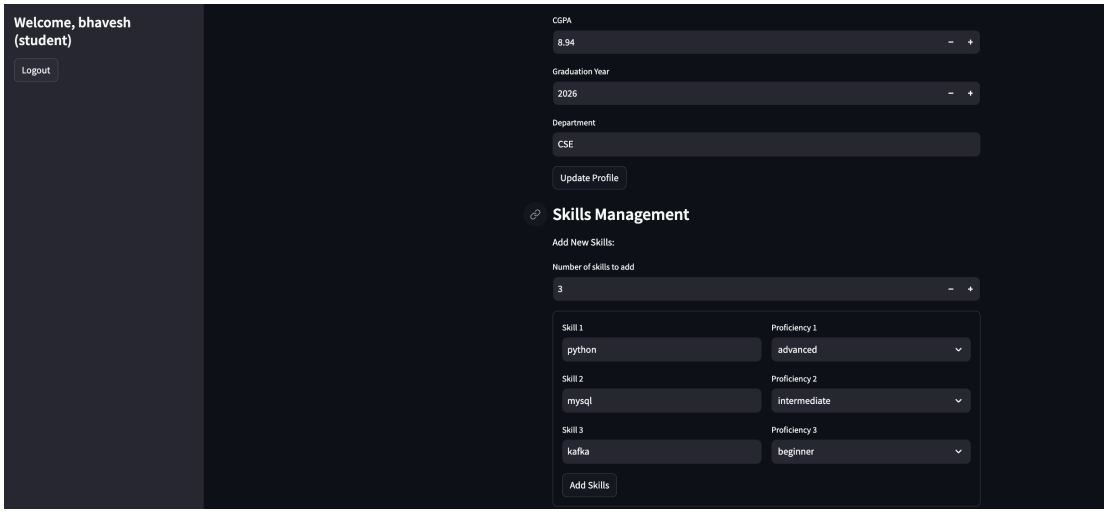
If Job Already posted with the same requirements and description

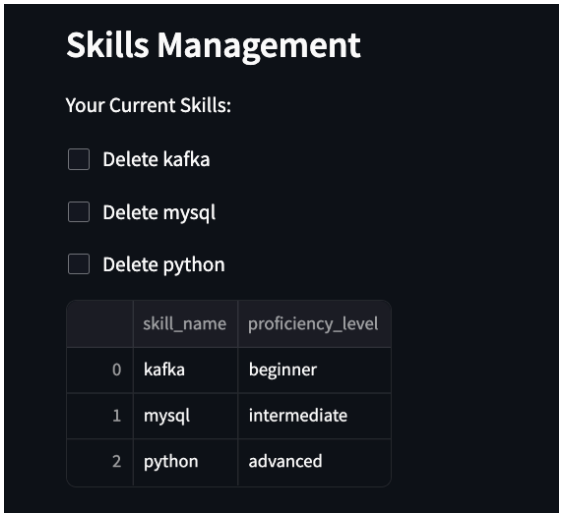



```
check_query = """
    SELECT * FROM Jobs
    WHERE company_id = %s
    AND title = %s
    AND description = %s
    AND job_type = %s
    AND location = %s
"""
cursor.execute(check_query,
               (company_id, title, description, job_type, location))
existing_job = cursor.fetchone()

if existing_job:
    st.warning("A job with the same title, description, job type, and location already exists.")
```

```
mysql> select * from jobs
    -> ;
+--------+------------+---------------+-----------------------------------------------------------------------------------------------+------------+-----------+--------------+--------------
-+---------------------+---------------------+--------+
| job_id | company_id | title         | description                                                                                   | job_type   | location  | salary_range | required_cgpa
 | posting_date        | deadline            | status |
+--------+------------+---------------+-----------------------------------------------------------------------------------------------+------------+-----------+--------------+--------------
-+---------------------+---------------------+--------+
|      1 |          1 | Data Engineer | Work of Distributed Technologies .
Must have a good knowledge of Kafka and Spark | internship | Bangalore | 100000       |         8.50 | 2024-11-12 09:59:17 | 2024-11-14 00:00:00 | open   |
+--------+------------+---------------+-----------------------------------------------------------------------------------------------+------------+-----------+--------------+--------------
-+---------------------+---------------------+--------+
1 row in set (0.00 sec)
```

a. Insertion from student – skills





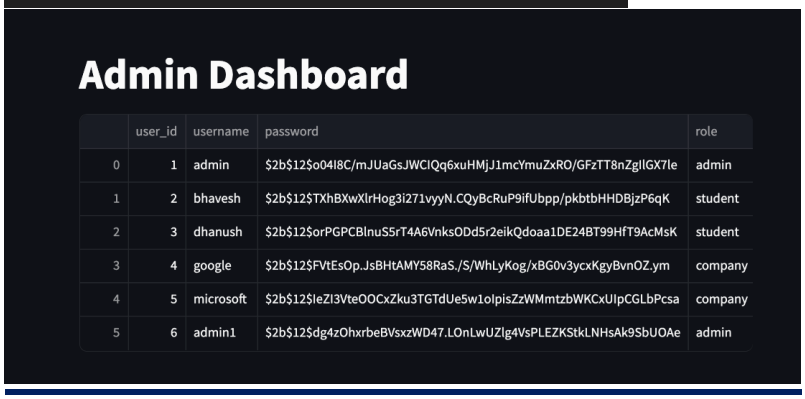

```
mysql> select * from student_skills;
+------------+------------+-------------------+
| student_id | skill_name | proficiency_level |
+------------+------------+-------------------+
|          1 | kafka      | beginner          |
|          1 | mysql      | intermediate      |
|          1 | python     | advanced          |
+------------+------------+-------------------+
3 rows in set (0.00 sec)
```

SELECT (READ) :

ADMIN :

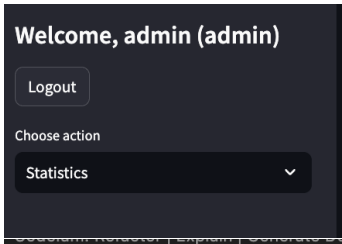
a. View all users

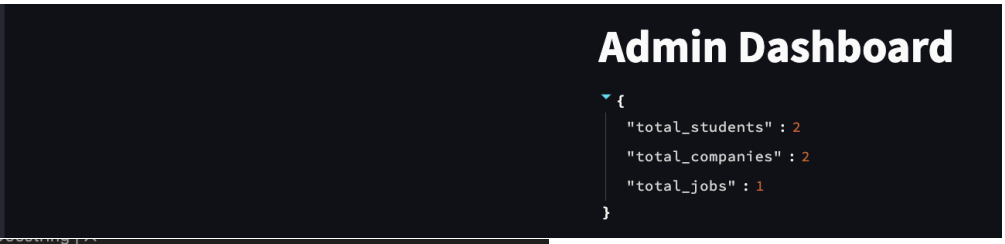
```python
def view_all_users():
    conn = get_database_connection()
    if conn:
        cursor = conn.cursor(dictionary=True)
        cursor.execute("SELECT * FROM Users")
        users = cursor.fetchall()
        cursor.close()
        conn.close()
        if users:
            users_df = pd.DataFrame(users)
            st.dataframe(users_df)
        else:
            st.write("No users found.")
    else:
        st.write("Database connection failed.")
```



	user_id	username	password	role
0	1	admin	$2b$12$o04I8C/mJUaGsJWClQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgllGX7le	admin
1	2	bhavesh	$2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK	student
2	3	dhanush	$2b$12$orPGPCBlnuS5rT4A6VnksODd5r2eikQdoaa1DE24BT99HfT9AcMsK	student
3	4	google	$2b$12$FVtEsOp.JsBHtAMY58RaS./S/WhLyKog/xBG0v3ycxKgyBvnOZ.ym	company
4	5	microsoft	$2b$12$leZI3VteOOCxZku3TGTdUe5w1olpisZzWMmtzbWKCxUIpCGLbPcsa	company
5	6	admin1	$2b$12$dg4zOhxrbeBVsxzWD47.LOnLwUZlg4VsPLEZKStkLNHsAk9SbUOAe	admin

b. Statistics

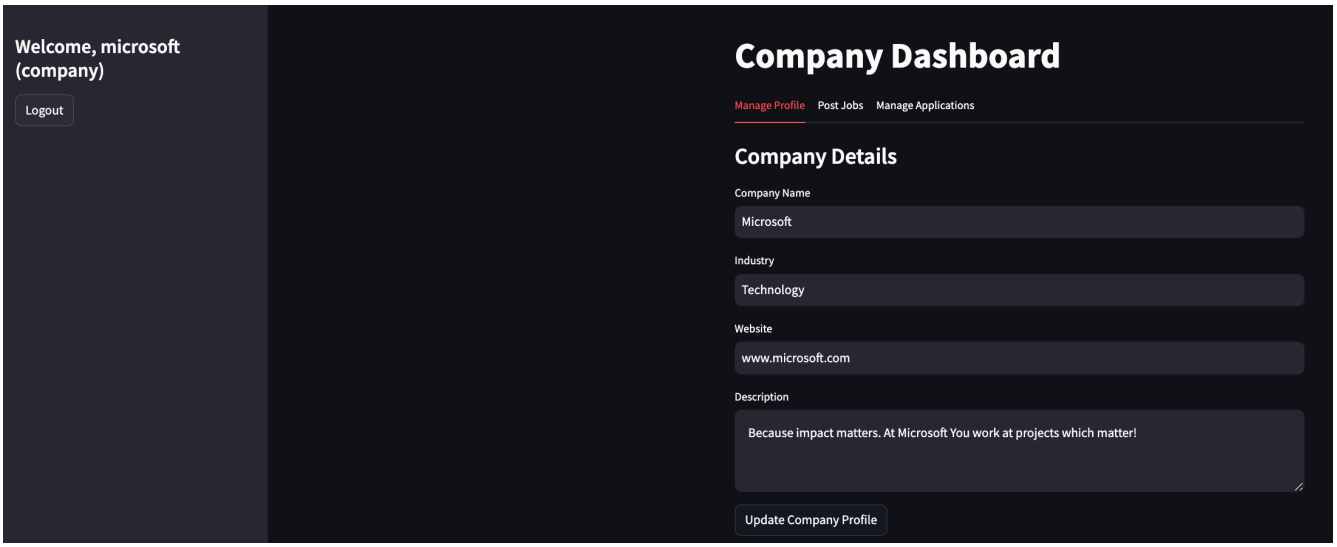




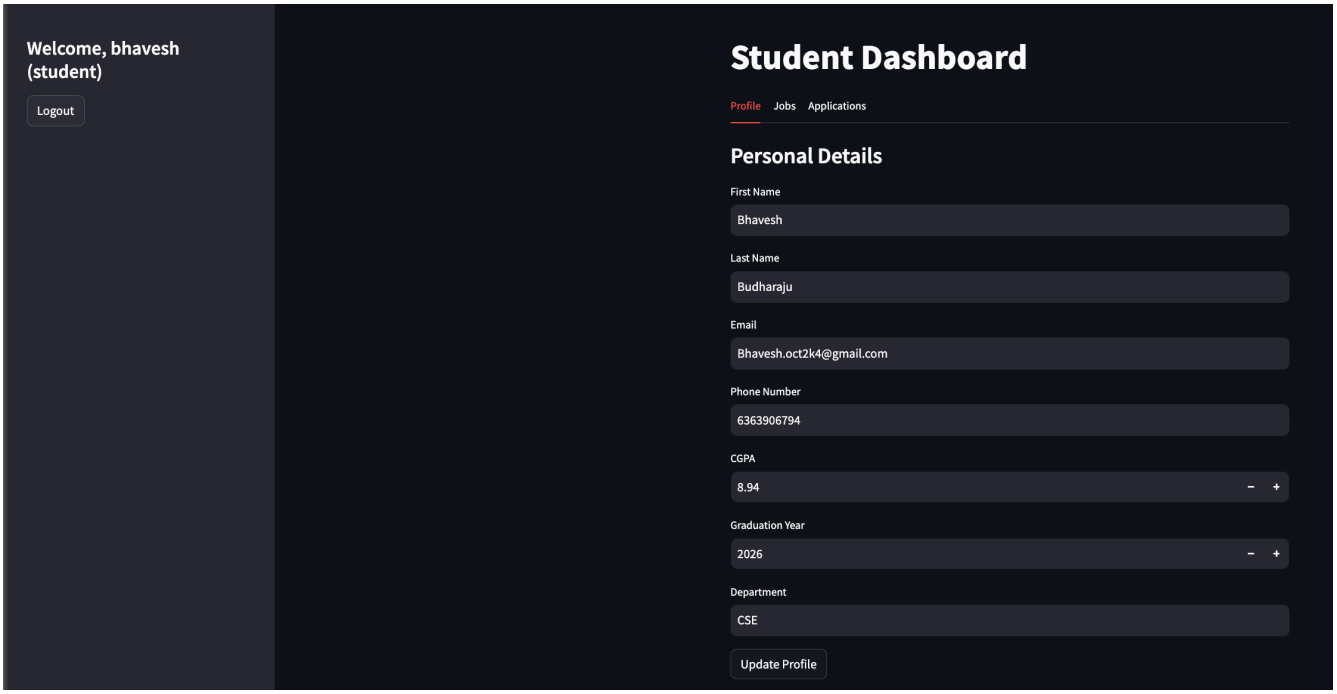

```python
def view_statistics():
    conn = get_database_connection()
    cursor = conn.cursor(dictionary=True)
    cursor.execute("""
        SELECT
            (SELECT COUNT(*) FROM Students) as total_students,
            (SELECT COUNT(*) FROM Companies) as total_companies,
            (SELECT COUNT(*) FROM Jobs) as total_jobs
    """)

    stats = cursor.fetchone()  # fetchone since we only expect one row
    st.write(stats)
```

COMPANY :

```
# Fetch existing details based on the user_id
cursor.execute("SELECT * FROM Companies WHERE user_id = %s", (st.session_state.user_id,))
company = cursor.fetchall()  # Use fetchall() to ensure all results are consumed
```

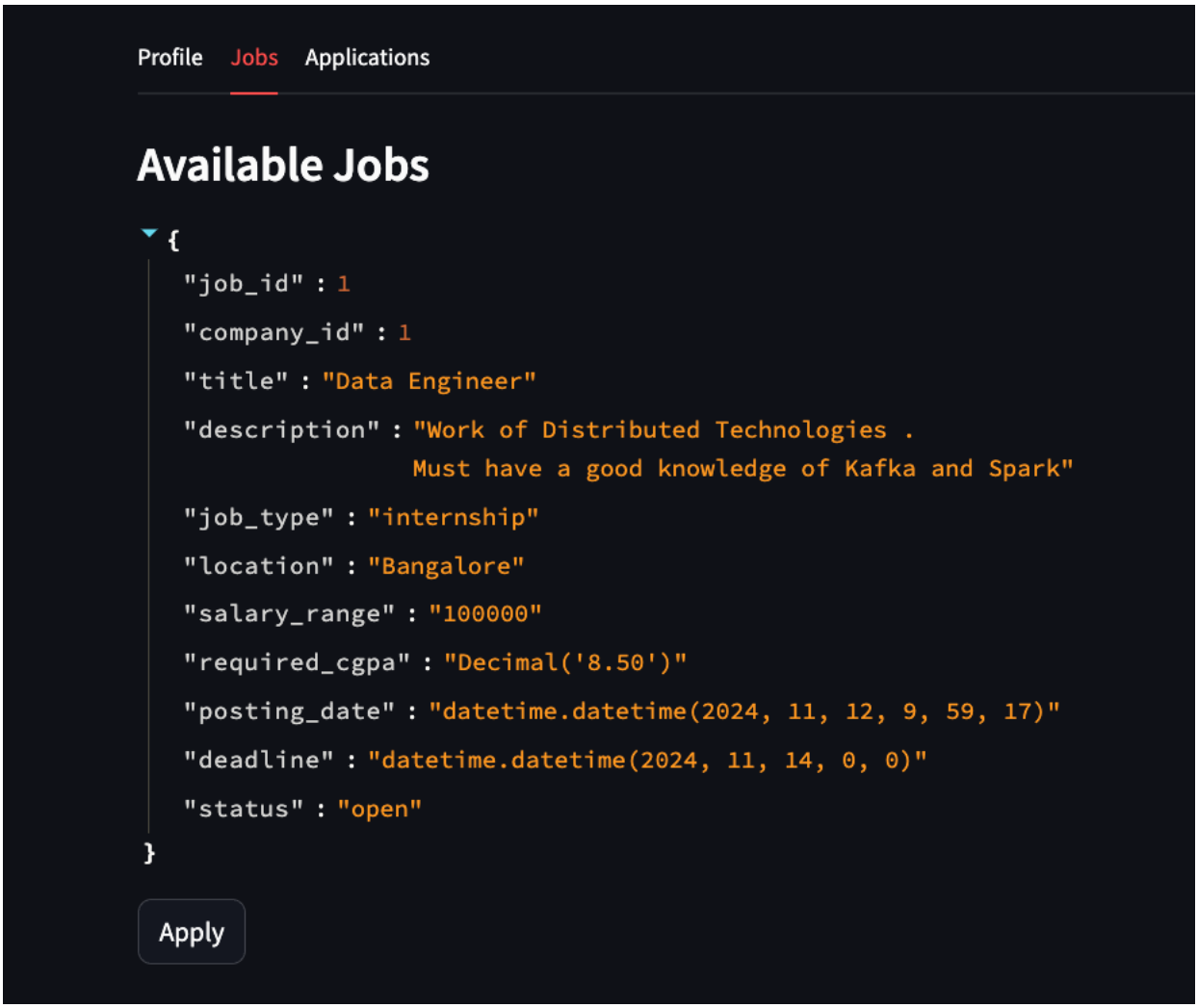


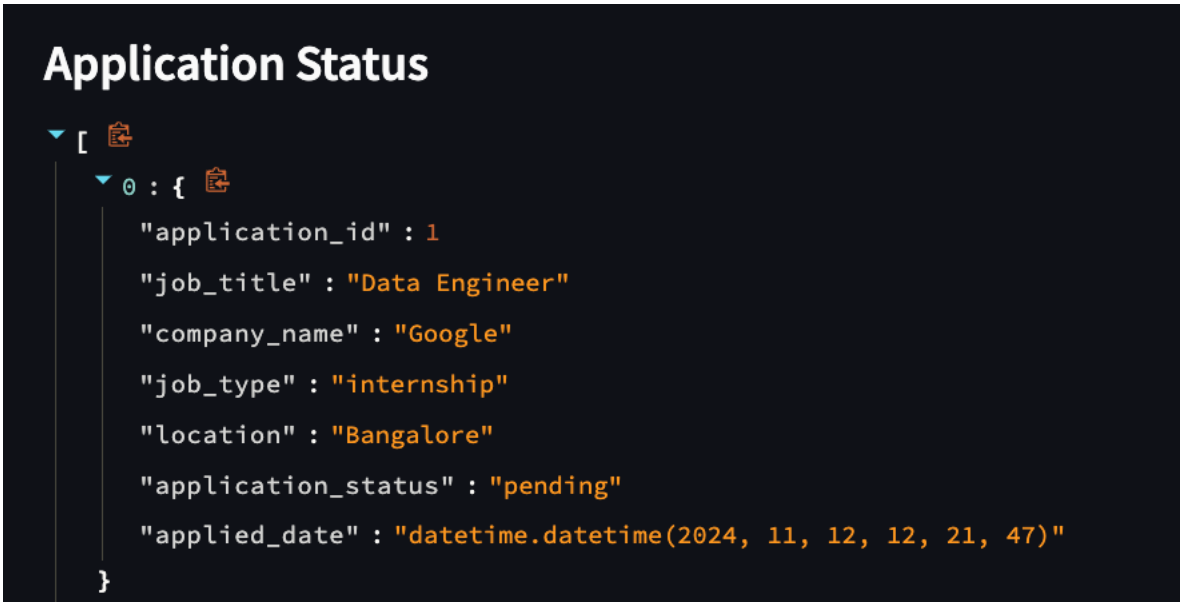
STUDENT :



```python
# Fetch existing details based on the user_id
cursor.execute("SELECT * FROM Students WHERE user_id = %s", (st.session_state.user_id,))
student = cursor.fetchone()
```

```python
student_id = student_data['student_id']
cursor.execute("SELECT * FROM Jobs WHERE status='open'")
jobs = cursor.fetchall()
for job in jobs:
    st.write(job)
    if st.button("Apply", key=job['job_id']):
        cursor.execute("INSERT INTO Applications (student_id, job_id) VALUES (%s, %s)",
                       (student_id, job['job_id']))
        conn.commit()
        st.success(f"Applied for {job['title']}.")
cursor.close()
conn.close()
```

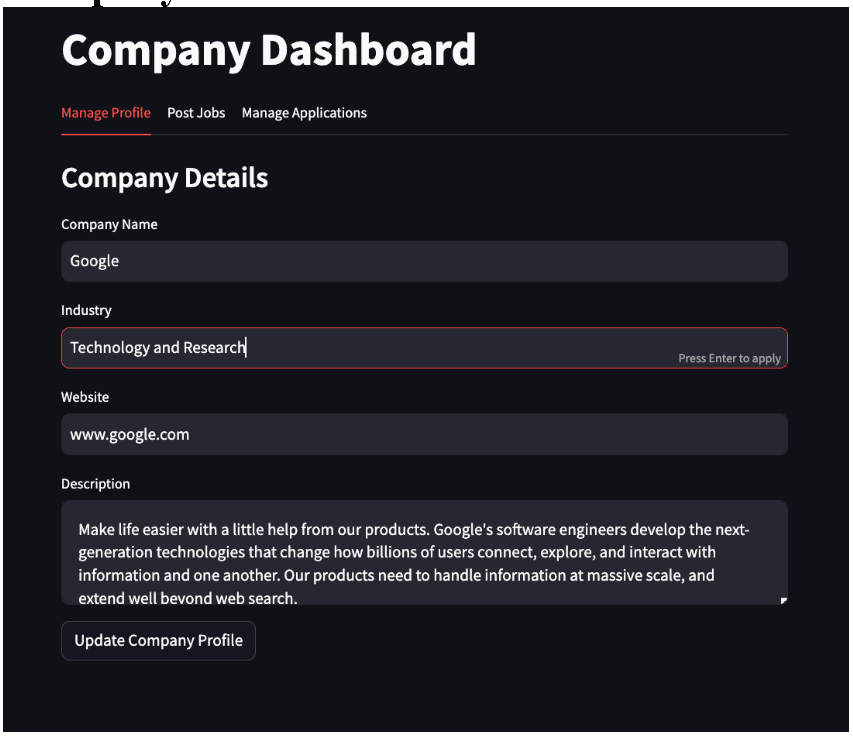




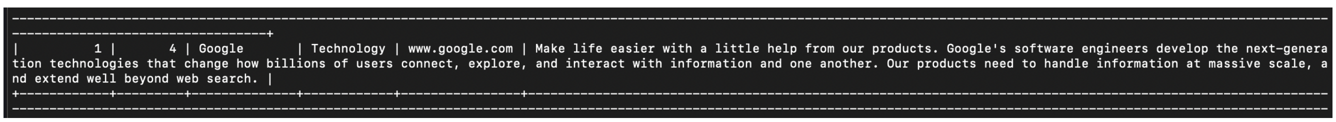

```
cursor.execute("SELECT student_id FROM Students WHERE user_id = %s", (st.session_state.user['user_id'],))
student_data = cursor.fetchone()

if not student_data:
    st.error("Student profile not found.")
    return

student_id = student_data['student_id']
cursor.execute("""
    CALL GetStudentApplicationDetails(%s)
""", (student_id,))
applications = cursor.fetchall()
st.write(applications)
cursor.close()
conn.close()
```

UPDATE :

Company :

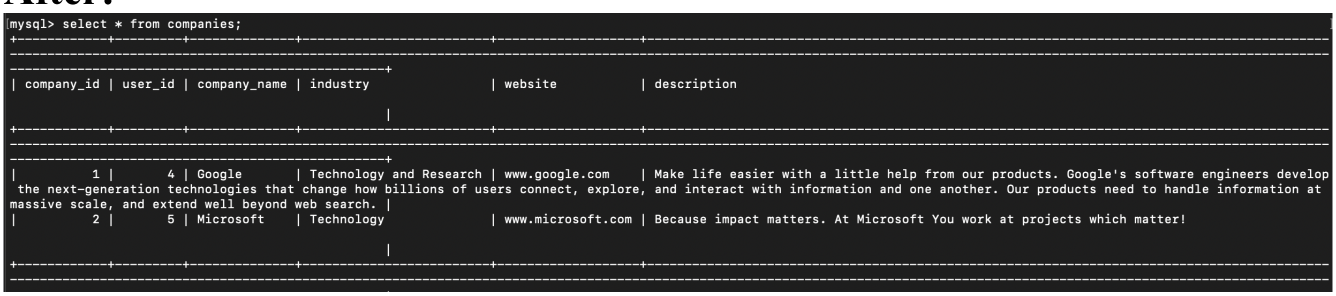


Before :



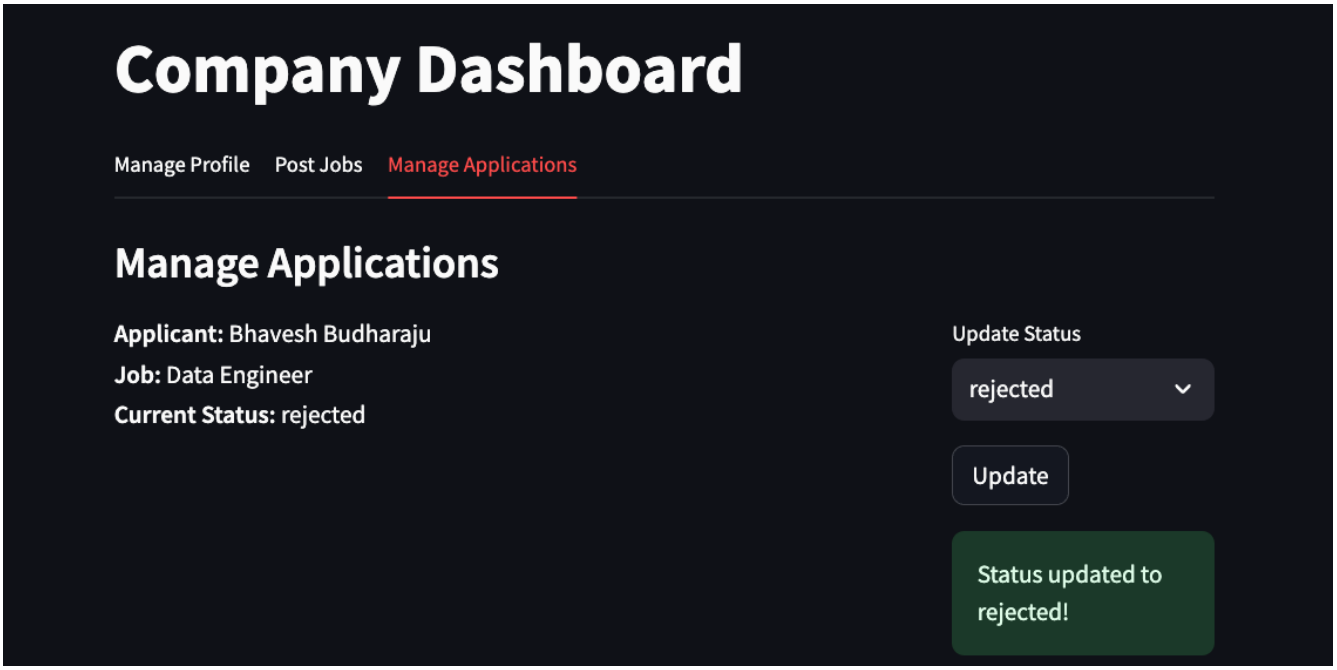
```
|          1 |       4 | Google       | Technology | www.google.com | Make life easier with a little help from our products. Google's software engineers develop the next-genera
tion technologies that change how billions of users connect, explore, and interact with information and one another. Our products need to handle information at massive scale, a
nd extend well beyond web search. |
```

After:



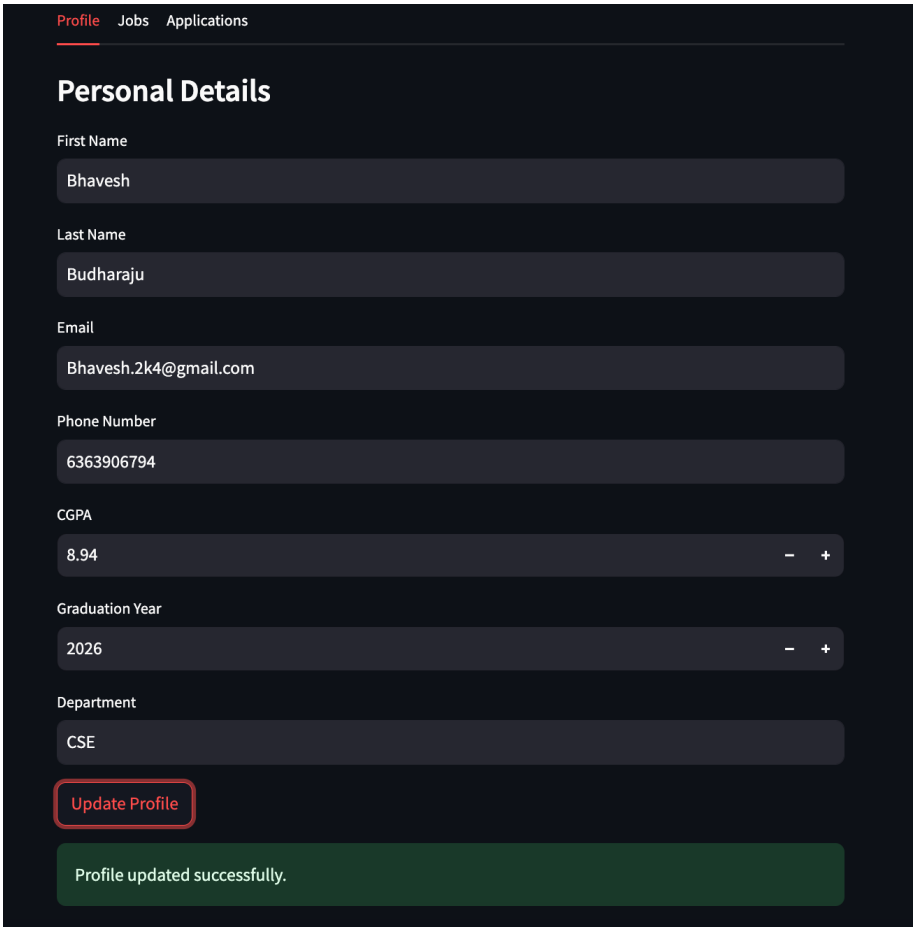
```
mysql> select * from companies;
| company_id | user_id | company_name | industry                 | website           | description
|          1 |       4 | Google       | Technology and Research | www.google.com    | Make life easier with a little help from our products. Google's software engineers develop
 the next-generation technologies that change how billions of users connect, explore, and interact with information and one another. Our products need to handle information at
massive scale, and extend well beyond web search. |
|          2 |       5 | Microsoft    | Technology               | www.microsoft.com | Because impact matters. At Microsoft You work at projects which matter!
```

Google : Industry changed from ‘Technology’ to ‘Technology and Research’




```
if st.button("Update", key=f"update_{app['application_id']}"):
    cursor.execute(
        "UPDATE Applications SET status = %s WHERE application_id = %s",
        (new_status, app['application_id'])
    )
    conn.commit()
```

Student



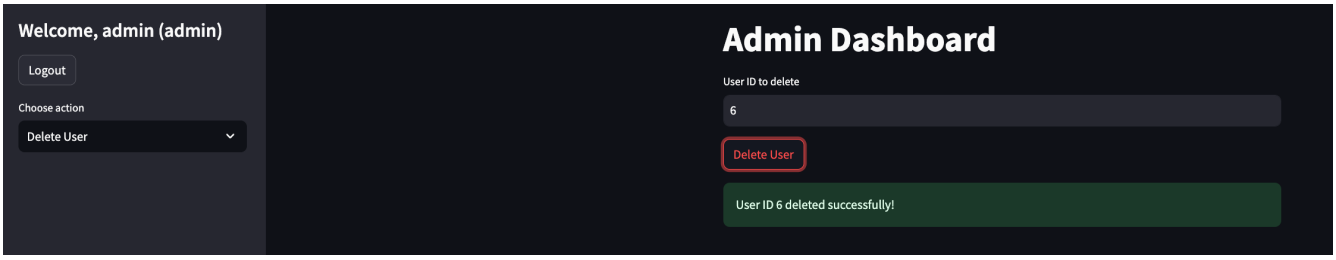

```
mysql> select * from students;
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
| student_id | user_id | first_name | last_name        | email                    | phone      | cgpa | graduation_year | department |
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
|          1 |       2 | Bhavesh    | Budharaju        | Bhavesh.oct2k4@gmail.com | 6363906794 | 8.94 |            2026 | CSE        |
|          2 |       3 | Dhanush    | Suresh Jettipalle | jsdhanush.2004@gmail.com | 6366693334 | 9.30 |            2026 | CSE        |
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
2 rows in set (0.00 sec)
```

```
mysql> select * from students;
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
| student_id | user_id | first_name | last_name        | email                    | phone      | cgpa | graduation_year | department |
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
|          1 |       2 | Bhavesh    | Budharaju        | Bhavesh.2k4@gmail.com    | 6363906794 | 8.94 |            2026 | CSE        |
|          2 |       3 | Dhanush    | Suresh Jettipalle | jsdhanush.2004@gmail.com | 6366693334 | 9.30 |            2026 | CSE        |
+------------+---------+------------+------------------+--------------------------+------------+------+-----------------+------------+
2 rows in set (0.00 sec)
```

```python
if st.button("Update Profile"):
    cursor.execute("""
        UPDATE Students
        SET first_name=%s, last_name=%s, email=%s, phone=%s, cgpa=%s, graduation_year=%s, department=%s
        WHERE user_id=%s
    """, (first_name, last_name, email, phone, cgpa, graduation_year, department, st.session_state.user_id))
    conn.commit()
    st.success("Profile updated successfully.")
```

DELETE

Admin:

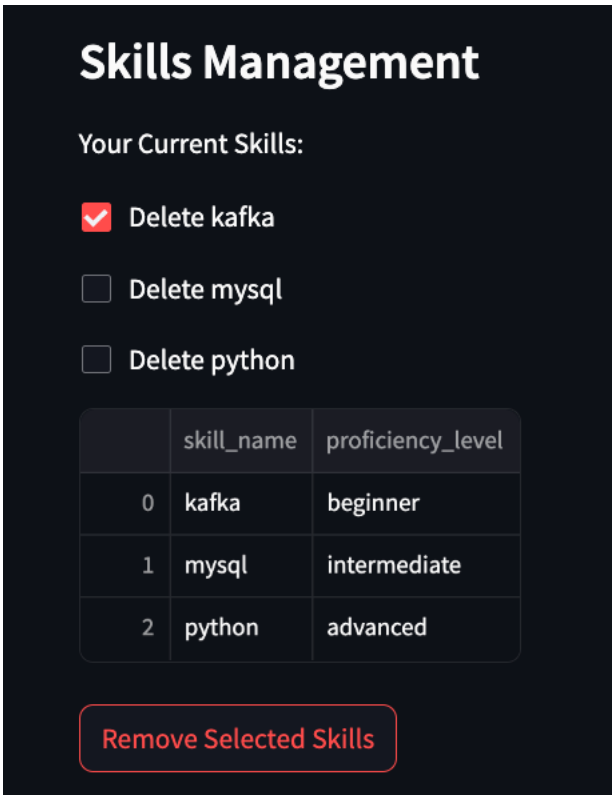


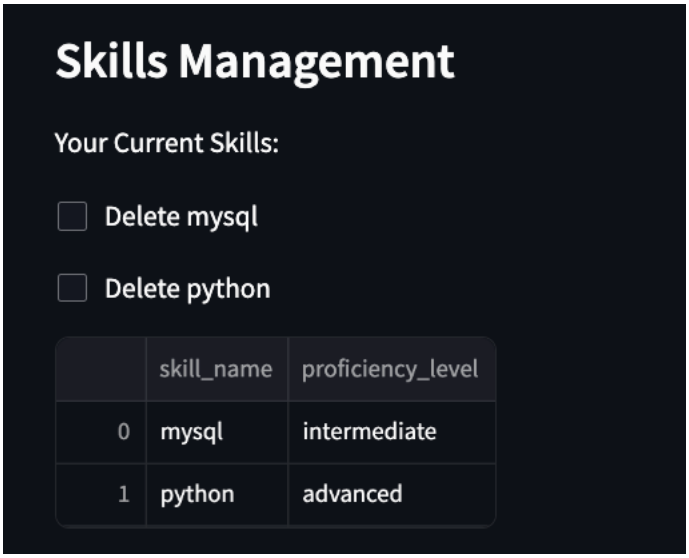

```
mysql> select * from users;
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
| user_id | username  | password                                                     | role    | created_at          |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
|       1 | admin     | $2b$12$o04I8C/mJUaGsJWCIQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgIlGX7le | admin   | 2024-11-12 09:22:50 |
|       2 | bhavesh   | $2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK | student | 2024-11-12 09:33:22 |
|       3 | dhanush   | $2b$12$orPGPCBlnuS5rT4A6VnksODd5r2eikQdoaa1DE24BT99HfT9AcMsK | student | 2024-11-12 09:38:57 |
|       4 | google    | $2b$12$FVtEsOp.JsBHtAMY58RaS./S/WhLyKog/xBG0v3ycxKgyBvnOZ.ym | company | 2024-11-12 09:43:19 |
|       5 | microsoft | $2b$12$IeZI3VteOOCxZku3TGTdUe5w1oIpisZzWMmtzbWKCxUIpCGLbPcsa | company | 2024-11-12 09:48:38 |
|       6 | admin1    | $2b$12$dg4zOhxrbeBVsxzWD47.LOnLwUZlg4VsPLEZKStkLNHsAk9SbUOAe | admin   | 2024-11-12 09:49:56 |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
6 rows in set (0.00 sec)

mysql> select * from users;
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
| user_id | username  | password                                                     | role    | created_at          |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
|       1 | admin     | $2b$12$o04I8C/mJUaGsJWCIQq6xuHMjJ1mcYmuZxRO/GFzTT8nZgIlGX7le | admin   | 2024-11-12 09:22:50 |
|       2 | bhavesh   | $2b$12$TXhBXwXlrHog3i271vyyN.CQyBcRuP9ifUbpp/pkbtbHHDBjzP6qK | student | 2024-11-12 09:33:22 |
|       3 | dhanush   | $2b$12$orPGPCBlnuS5rT4A6VnksODd5r2eikQdoaa1DE24BT99HfT9AcMsK | student | 2024-11-12 09:38:57 |
|       4 | google    | $2b$12$FVtEsOp.JsBHtAMY58RaS./S/WhLyKog/xBG0v3ycxKgyBvnOZ.ym | company | 2024-11-12 09:43:19 |
|       5 | microsoft | $2b$12$IeZI3VteOOCxZku3TGTdUe5w1oIpisZzWMmtzbWKCxUIpCGLbPcsa | company | 2024-11-12 09:48:38 |
+---------+-----------+--------------------------------------------------------------+---------+---------------------+
5 rows in set (0.00 sec)
```

```python
cursor.execute("SELECT role FROM Users WHERE user_id = %s", (user_id,))
user = cursor.fetchone()

if user:
    role = user['role']
    # Delete from Students or Companies based on role
    if role == 'student':
        cursor.execute("DELETE FROM Students WHERE user_id = %s", (user_id,))
    elif role == 'company':
        cursor.execute("DELETE FROM Companies WHERE user_id = %s", (user_id,))

    # Then delete from Users table
    cursor.execute("DELETE FROM Users WHERE user_id = %s", (user_id,))
    conn.commit()
    st.success(f"User ID {user_id} deleted successfully!")
```

Student

```
DELETE FROM Student_Skills
WHERE student_id = %s AND skill_name = %s
```





Before :

```
mysql> select * from student_skills;
+------------+------------+-------------------+
| student_id | skill_name | proficiency_level |
+------------+------------+-------------------+
|          1 | kafka      | beginner          |
|          1 | mysql      | intermediate      |
|          1 | python     | advanced          |
+------------+------------+-------------------+
3 rows in set (0.00 sec)
```

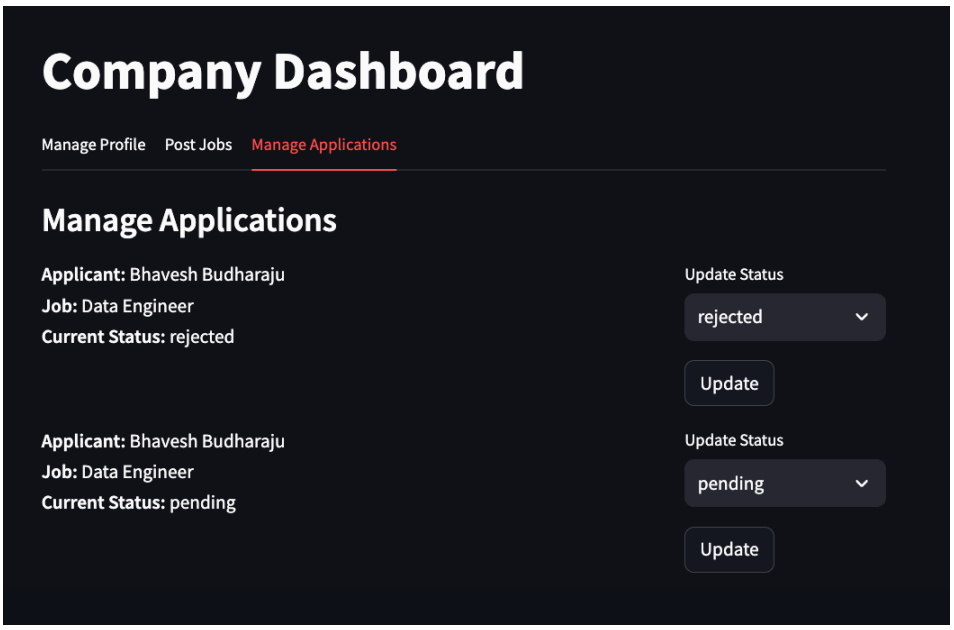
After :

```
mysql> select * from student_skills;
+------------+------------+-------------------+
| student_id | skill_name | proficiency_level |
+------------+------------+-------------------+
|          1 | mysql      | intermediate      |
|          1 | python     | advanced          |
+------------+------------+-------------------+
```

8. QUERIES (JOIN QUERY, AGGREGATE FUNCTION QUERIES AND NESTED QUERY)

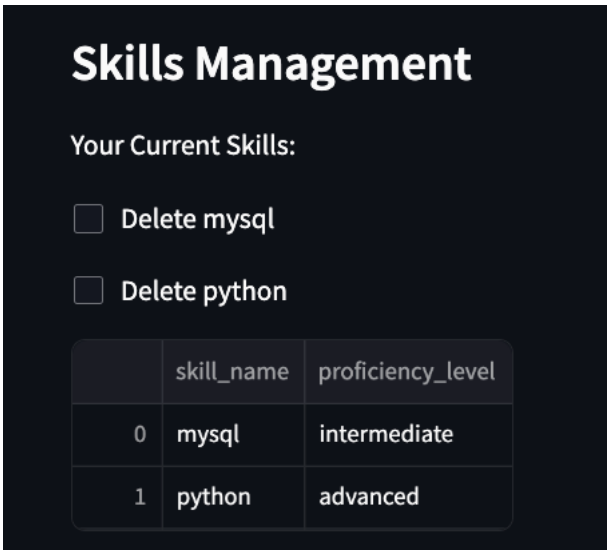
JOIN :

```
# Fetch applications
cursor.execute("""
    SELECT
        a.application_id,
        s.first_name,
        s.last_name,
        s.student_id,
        j.title,
        a.status
    FROM Applications a
    JOIN Students s ON a.student_id = s.student_id
    JOIN Jobs j ON a.job_id = j.job_id
    WHERE j.company_id = %s
""", (company_id,))
```



Aggregate Function Queries:

```
SELECT COUNT(*)
FROM Student_Skills
WHERE student_id = %s AND skill_name = %s
```

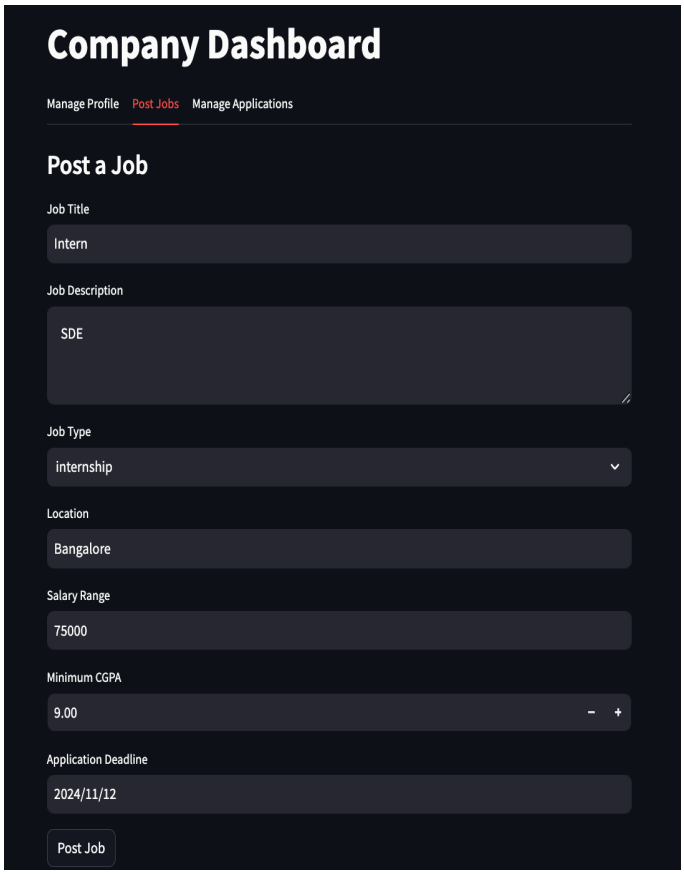


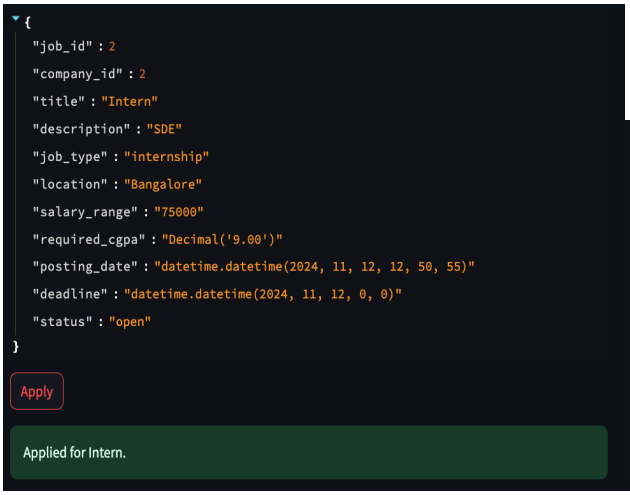
Nested Query :

```
SELECT
            (SELECT COUNT(*) FROM Students) as total_students,
            (SELECT COUNT(*) FROM Companies) as total_companies,
            (SELECT COUNT(*) FROM Jobs) as total_jobs
    """)
```

9. STORED PROCEDURE, FUNCTIONS AND TRIGGERS

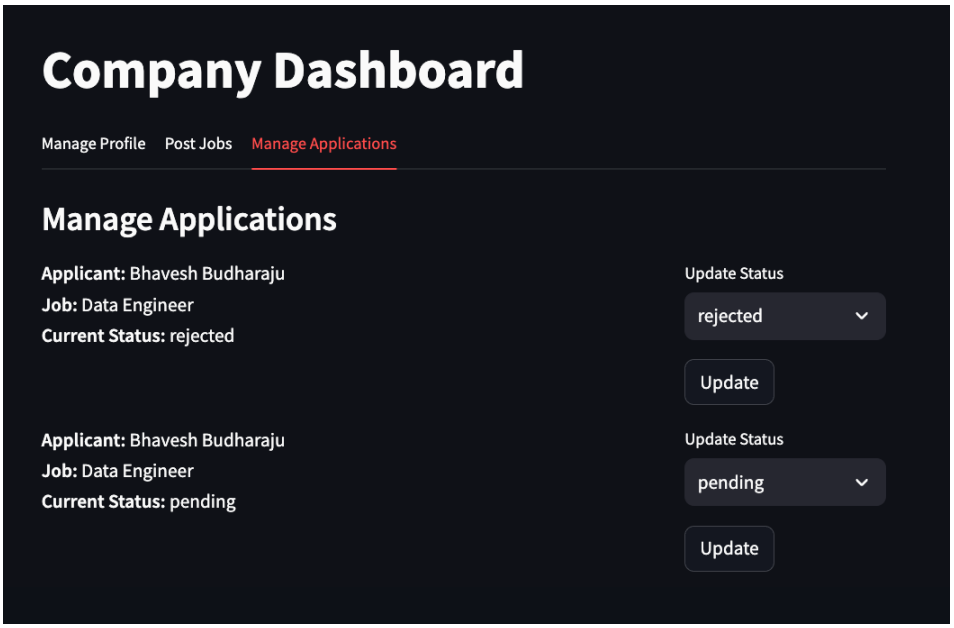
```
TRIGGER_SCRIPT = """
CREATE TRIGGER before_application_insert
BEFORE INSERT ON Applications
FOR EACH ROW
BEGIN
    DECLARE student_cgpa DECIMAL(4,2);
    DECLARE job_min_cgpa DECIMAL(4,2);

    SELECT cgpa INTO student_cgpa
    FROM Students
    WHERE student_id = NEW.student_id;

    SELECT required_cgpa INTO job_min_cgpa
    FROM Jobs
    WHERE job_id = NEW.job_id;

    IF student_cgpa < job_min_cgpa THEN
        SET NEW.status = 'rejected';
    END IF;
END;
"""
```





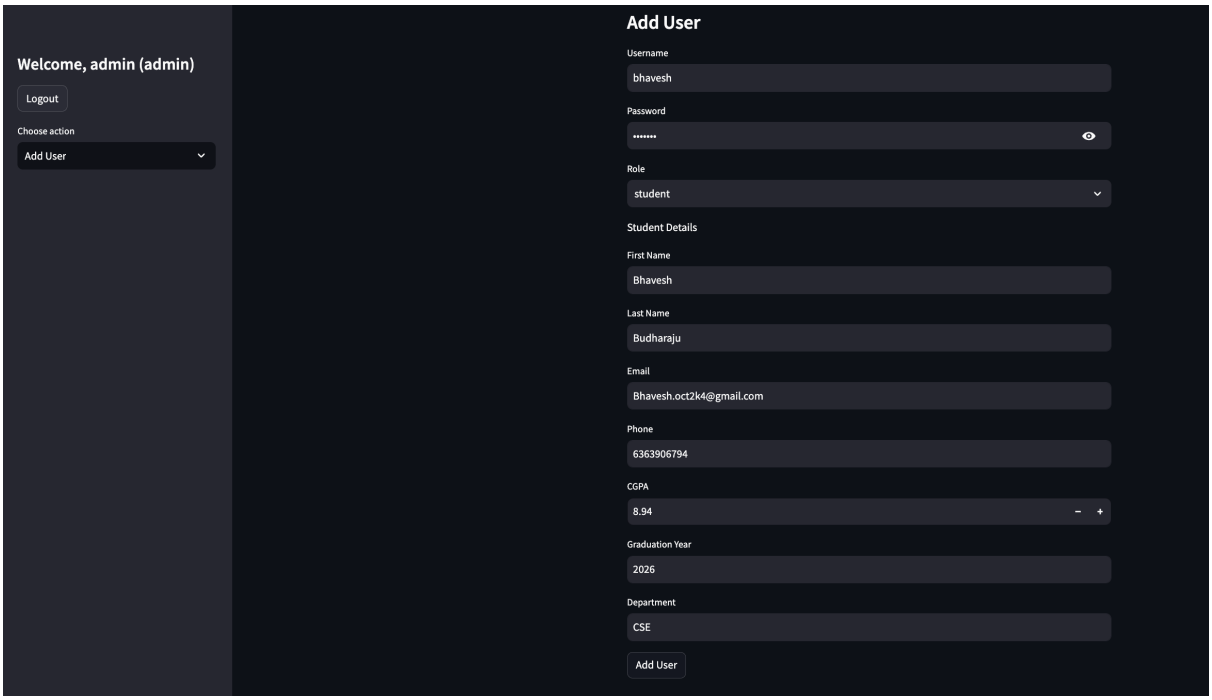
If the students gpa is less than the min gpa required for the job, the status is automatically set to rejected

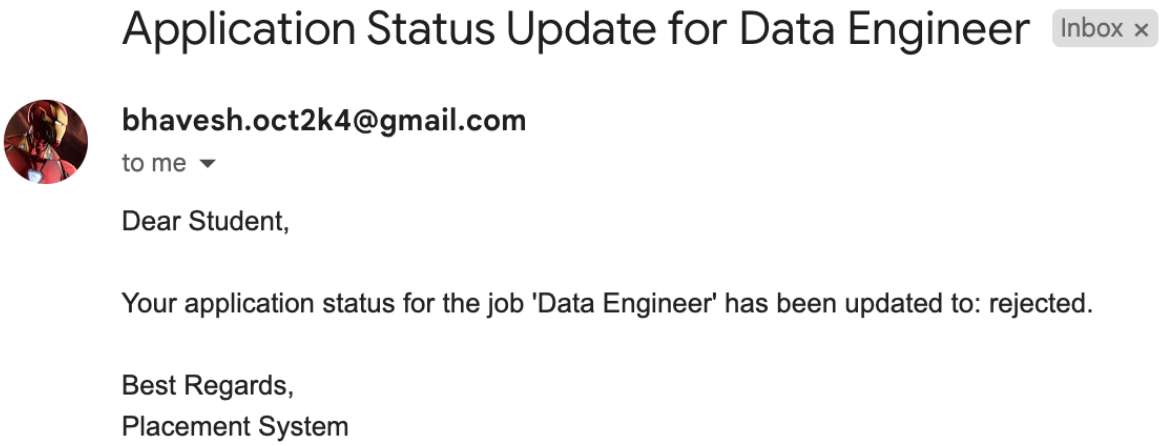
```
STORED_PROC_SCRIPT = """
CREATE PROCEDURE GetStudentApplicationDetails(IN p_student_id INT)
BEGIN
    SELECT
        a.application_id,
        j.title AS job_title,
        c.company_name,
        j.job_type,
        j.location,
        a.status AS application_status,
        a.applied_date
    FROM Applications a
    JOIN Jobs j ON a.job_id = j.job_id
    JOIN Companies c ON j.company_id = c.company_id
    WHERE a.student_id = p_student_id;
END;
"""
```



10. FRONT END DEVELOPMENT (FUNCTIONALITIES/FEATURES OF THE APPLICATION)

Email :





```python
load_dotenv()
EMAIL=os.getenv('EMAIL')
PASSWORD=os.getenv('EMAIL_PASSWORD')
SMTP_SERVER = "smtp.gmail.com"
SMTP_PORT = 587

Codeium: Refactor | Explain | Generate Docstring | ×
def send_application_status_email(student_id, job_title, status):
    to_email = get_student_email(student_id)
    if not to_email:
        return

    subject = f"Application Status Update for {job_title}"
    body = f"Dear Student,\n\nYour application status for the job '{job_title}' has been updated to: {status}.\n\nBest Regards,\nPlacement System"

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

    try:
        with smtplib.SMTP(SMTP_SERVER, SMTP_PORT) as server:
            server.starttls()
            server.login(EMAIL, PASSWORD)
            server.sendmail(EMAIL, to_email, msg.as_string())
    except Exception as e:
        print(f"Failed to send email: {e}")
```

REFERENCES/BIBLIOGRAPHY

- https://dev.mysql.com/doc/
- https://docs.streamlit.io/
- https://docs.python.org/
- https://dev.mysql.com/doc/connector-python/en/
- https://www.npmjs.com/package/bcrypt
- https://docs.python.org/3/library/smtplib.html
- https://github.com/Bhavesh2k4/SQLHire
- https://www.dotenv.org/docs/

APPENDIX A DEFINITIONS, ACRONYMS AND ABBREVIATIONS

Definitions:

● User Authentication: The process of verifying the identity of a user to ensure secure access to the system.

● Stored Procedure: Stored procedures are a set of SQL statements that are stored in a database and can be executed as a single unit.

● Trigger: A database mechanism that automatically executes a predefined action when a specific database event occurs.

Acronyms:

● SQL: Structured Query Language

● DBMS: Database Management System

● CRUD: Create, Read, Update, Delete Operations

● SMTP : Simple Mail Transfer Protocol