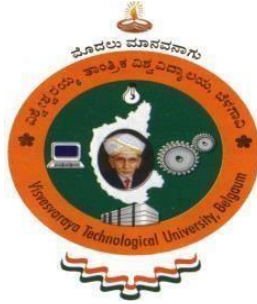


# VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANASANGAMA, BELGAVI-590018



## DBMS LABORATORY WITH MINI PROJECT(18CSL58) ON

### “RECRUITMENT MANAGEMENT SYSTEM”

Submitted in partial fulfillment of the requirements for the 5<sup>th</sup> Semester

INFORMATION SCIENCE AND ENGINEERING

**Submitted by**

**KUMKUM MITTAL** (1BI19IS027)

**ANUSHA G** (1BI20IS401)

Under the guidance of:

**Dr. Hema Jagdish**

Associate Professor  
Dept of ISE, BIT

**Mrs. M Shilpa**

Assistant Professor  
Dept of ISE, BIT



**2021-2022**

**DEPARTMENT OF INFORMATION SCIENCE AND ENGINEERING**

**BANGALORE INSTITUTE OF TECHNOLOGY**

**K. R. Road, V. V. Pura, Bengaluru-560004**

# **BANGALORE INSTITUTE OF TECHNOLOGY**

K.R. Road, V.V. Puram, Bengaluru -560004

## **Department of Information Science and Engineering**



### **CERTIFICATE**

This is to certify that the implementation of **DBMS LABORATORY WITH MINI PROJECT (18CSL58)** entitled “**RECRUITMENT MANGAEMENT SYSTEM**” has been successfully completed by **KUMKUM MITTAL (1BI19IS027)** of V semester B.E. for the partial fulfilment of the requirements for the Bachelor's degree in Information Science & Engineering of the **Visvesvaraya Technological University** during the academic year 2021-2022.

#### **Lab In charge:**

**Mrs. M Shilpa**

Assistant Professor

Dept of ISE, BIT

#### **Head of Department:**

**Dr. Roopa H**

Associate Professor and HOD

Dept of ISE, BIT

#### **Name of Examiners:**

1.

2.

#### **Signature with date**

## **DECLARATION**

**I, KUMKUM MITTAL** bearing USN **1BI19IS027** student of Fifth Semester **Bachelor of Engineering in Information Science Engineering, Department of Information Science Engineering, Bangalore Institute of Technology, Bangalore** declare that the project work has been carried out by me and submitted in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering in Information Science Engineering of Visvesvaraya Technological University, Belagavi** during the year 2021-2022. The matter embodied at this report has not been submitted to any university or institute for the award of any other degree.

**Place: Bangalore**

**Name: Kumkum Mittal**

**Date:**

**USN: 1BI19IS027**

## **ACKNOWLEDGEMENT**

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of the people who made it possible. So, with gratitude, I acknowledge all those whose guidance and encouragement crowned my effort with success.

It's immense pleasure in thanking principal **Dr. Ashwath M.U**, BIT, Bangalore, for providing an opportunity to present this project as a part of my curriculum in the partial fulfilment of the degree.

I express sincere gratitude for **Dr. H. Roopa H**, Head of the Department, Information Science and Engineering for her co-operation and encouragement at all moments of my approach.

It is pleasant duty to place on record my deepest sense of gratitude to my guide **M Chethana**, Asst. Professor, for the constant encouragement, for valuable help and assistance in every possible way.

It is pleasant duty to place on record my deepest sense of gratitude to my guide **Mrs. M Shilpa**, Asst. Professor for the constant encouragement, and **Dr Hema Jagadish**, Associate Professor for valuable help and assistance in every possible way.

I would like to thank all teachers and non-teaching staff of ISE Department for providing their valuable guidance and for being there at all the stages of my world.

**KUMKUM MITTAL (1BI19IS027)**

## **ABSTRACT**

A recruitment management system (RMS) is a set of tools designed to manage the recruiting process. It may be the most important core human resources (HR) system. Recruitment management systems have a range of components and functions. They include applicant tracking systems for managing job postings and applications, and customer relationship management-type functions to keep applicants connected and engaged. Many of these tasks are automated. Recruiting is competitive, especially for candidates with high-demand technical skills, and it is driving interest in sophisticated recruiting systems. These systems are being designed to improve the efficiency of recruiters as well as job seekers.

# **TABLE OF CONTENTS**

<b>Chapter No.</b>	<b>Contents</b>	<b>Page No.</b>
	<b>CERTIFICATE</b>	ii
	<b>DECLARATION</b>	iii
	<b>ACKNOWLEDGEMENT</b>	iv
	<b>ABSTRACT</b>	v
	<b>TABLE OF CONTENTS</b>	vi
	<b>LIST OF FIGURES</b>	vii
	<b>ABBREVIATIONS</b>	viii
<b>1</b>	<b>INTRODUCTION</b>	1
	1.1 Background	1
	1.2 Introduction to Recruitment Management System	2
<b>2</b>	<b>E R DIAGRAM AND RELATIONAL SCHEMA DIAGRAM</b>	4
	2.1 Description of ER Diagram	4
	2.2 Description of Relational Schema Diagram	8
<b>3</b>	<b>SYSTEM DESIGN</b>	11
	3.1 Table Description	11
<b>4</b>	<b>IMPLEMENTATION</b>	14
	4.1 Front-end Code	14
	4.1.1 Modules	14
	4.1.2 Tkinter	38
	4.2 Back-end Code	46
	4.2.1 Main.py	47
	4.2.2 db_inti.py	47
	4.2.3 cv_schema_init.py	47
	4.3 User-flow diagram	49

	4.4 Discussion of code segment	50
	4.5 Discussion of results	59
	4.6 Application of projects	71
<b>5</b>	<b>CONCLUSION AND FUTURE ENHANCEMENTS</b>	72
	5.1 Conclusion	72
	5.2 Future Enhancements	72

## **LIST OF FIGURES**

<b>Figure. No.</b>	<b>Descriptions</b>	<b>Page</b>
Figure. 2.1	E-R Diagram for Recruitment Management System	03
Figure. 2.2	Relational Schema - Recruitment Management System	04
Figure. 3.1	Table: Users	09
Figure. 3.2	Table: Recruiters	09
Figure. 3.3	Table: Clients	09
Figure. 3.4	Table: Job	10
Figure. 3.5	Table: Applications	10
Figure. 4.1	User flow diagram	17
Figure. 4.2	Login segment	17
Figure. 4.3	Register segment	18
Figure. 4.4	Posted job query	18
Figure. 4.5	Submit segment	19
Figure. 4.6	Delete posted job	19
Figure. 4.7	Sort queries	20
Figure. 4.8	Delete job application segment	20
Figure. 4.9	Job application segment	21
Figure. 4.9.1	Homepage with login prompt	21
Figure. 4.10	Registration prompt	22
Figure 4.11	Registration page	22
Figure. 4.12	Registering as recruiter	23
Figure. 4.13	Registering as client	23
Figure. 4.14	Options for recruiter	24
Figure. 4.15	Post a job	24
Figure. 4.16	Posted jobs	25
Figure. 4.17	Applications	25
Figure. 4.18	Client page	26
Figure. 4.19	Available jobs	26
Figure. 4.20	My applications	27



## **ABBREVIATIONS**

API	-	Application Programming Interface
ATS	-	Applicant Tracking System
RMS	-	Recruitment Management System
CRS	-	Computer Reservation System
CSS	-	Cascading style sheets
DBMS	-	Database Management System
ER	-	Entity Relationship
HR	-	Human Resources
HTML	-	Hypertext Markup Language
HTTP	-	Hypertext Transfer Protocol
ID	-	Identification
JS	-	JavaScript
MVD	-	Multi Valued Dependency
PHP	-	PHP Hypertext Preprocessor
SQL	-	Structured Query Language

## Chapter 1

# INTRODUCTION

A **database** is an organized collection of data, generally stored and accessed electronically from a computer system. Where databases are more complex, they are often developed using formal design and modeling techniques.

### 1.1 Background

The database management system (DBMS) is the software that interacts with end users, applications, the database itself to capture and analyze the data and provides facilities to administer the database. The sum total of the database, the DBMS and the associated applications can be referred to as a "database system". Often the term "database" is also used to loosely refer to any of the DBMS, the database system or an application associated with the database. The DBMS manages three important things: the data, the database engine that allows data to be accessed, locked and modified and the database schema, which defines the database's logical structure. These three foundational elements help provide concurrency, security, data integrity and uniform administration procedures. Typical database administration tasks supported by the DBMS include change management, performance monitoring/tuning and backup and recovery. Many database management systems are also responsible for automated rollbacks, restarts and recovery as well as the logging and auditing of activity.

## 1.2 Introduction to Recruitment Management System

A recruitment management system (RMS) is a set of tools designed to manage the recruiting process. It may be the most important core human resources (HR) system. Recruiting is competitive, especially for candidates with high-demand technical skills, and it is the driving interest in sophisticated recruiting systems. These systems are being designed to improve the efficiency of recruiters as well as job seekers.

A Recruitment Management System (RMS) basically helps to automate and manage your organization's recruiting and staffing operations. From posting your jobs to keeping applicants connected and engaged throughout, most RMS lets you manage your entire recruiting process.

Recruitment management systems have a range of components and functions. They include applicant tracking systems for managing job postings and applications, and customer relationship management-type functions to keep applicants connected and engaged. Many of these tasks are automated.

However, for our mini project, we have built an ideal local system RMS, where a recruiter can login and post jobs of various types with necessary qualifications required.

An applicant can set their account up which includes information about their skills and experience, and can thus apply for jobs.

Small companies often migrate from highly manual processes that rely on email and spreadsheets to an ATS. As the recruitment process matures and the organization grows, they may want to take their recruitment to the next level and take advantage of the advanced functions offered by an RMS.

However, an RMS is often best suited to midsize and large organizations, since the software and implementation costs are often higher than with an ATS.

Another reason an RMS is best suited to large organizations is the volume of candidate resumes. As the number of candidates grows, the need for streamlined processes and automation also grows.

Features in our project include using PyMySQL as the connector which basically enables Python programs to access MySQL databases, and custom user interface using images for elements like background buttons and icons.

## Chapter 2

# E R DIAGRAM AND RELATIONAL SCHEMA DIAGRAM

## 2.1 Description of ER Diagram

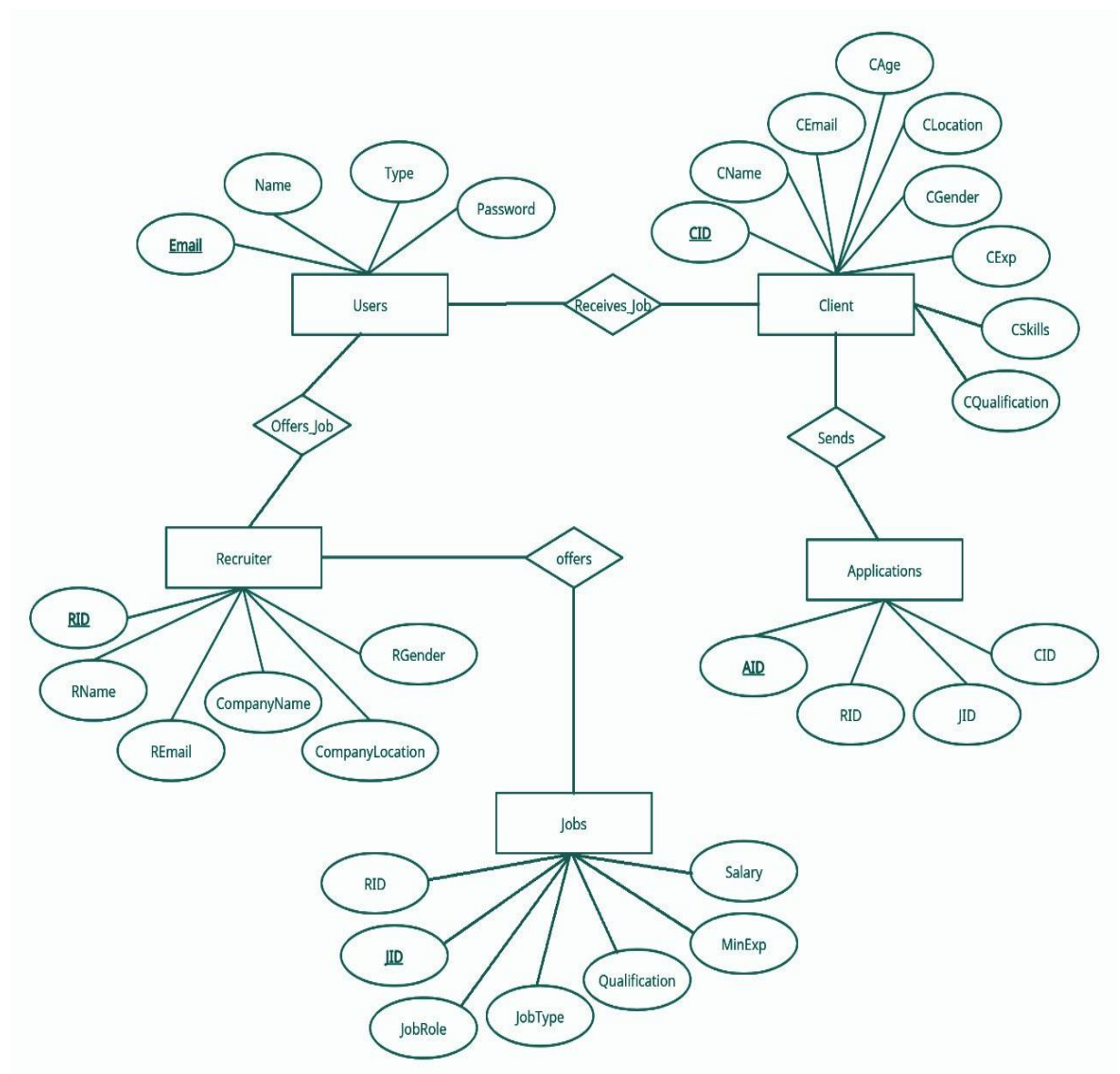


Figure 2.1: E-R Diagram for Recruitment Management System

Entity relationship diagram displays the relationships of entity set stored in a database. In other words, we can say that ER diagrams help you to explain the logical structure of databases. At first look, an ER diagram looks very similar to the flowchart. However, ER Diagram includes many specialized symbols, and its meanings make this model unique.

### 2.1.1 Components of Recruitment Management System, E-R Diagram

Entity types like RECRUITER and JOBS are in rectangular boxes.

1. Relationships like OFFERS and SENDS are in diamond boxes, attached to entity types with straight lines.
2. Attributes are shown in ovals like COMPANYNAME and CGENDER, each attached by a straightline to entity or relationship type.
3. Key attributes like EMAIL and RID are underlined.

### 2.1.2 E-R Diagram Relationships Description

1. **USERS: CLIENT** is of cardinality M: N as one client can possibly have more than one user and one user can have multiple clients. They are connected by RECIEVES\_JOB.
2. **CLIENT: APPLICATIONS** is of cardinality 1:1 as one client can have only one application and an application can be owned by one client only. Connected by SENDS.
3. **RECRUITER:JOBS** is of cardinality M: N. One recruiter can offer multiple jobs, and one job can have multiple recruiters. They're in a relationship called OFFERS.
4. **USERS: RECRUITER** is of cardinality M: N as one user can be connected to multiple recruiters and vice versa. They're in OFFERS\_JOB relationship .

\



## 2.2 Description of Relational Schema Diagram

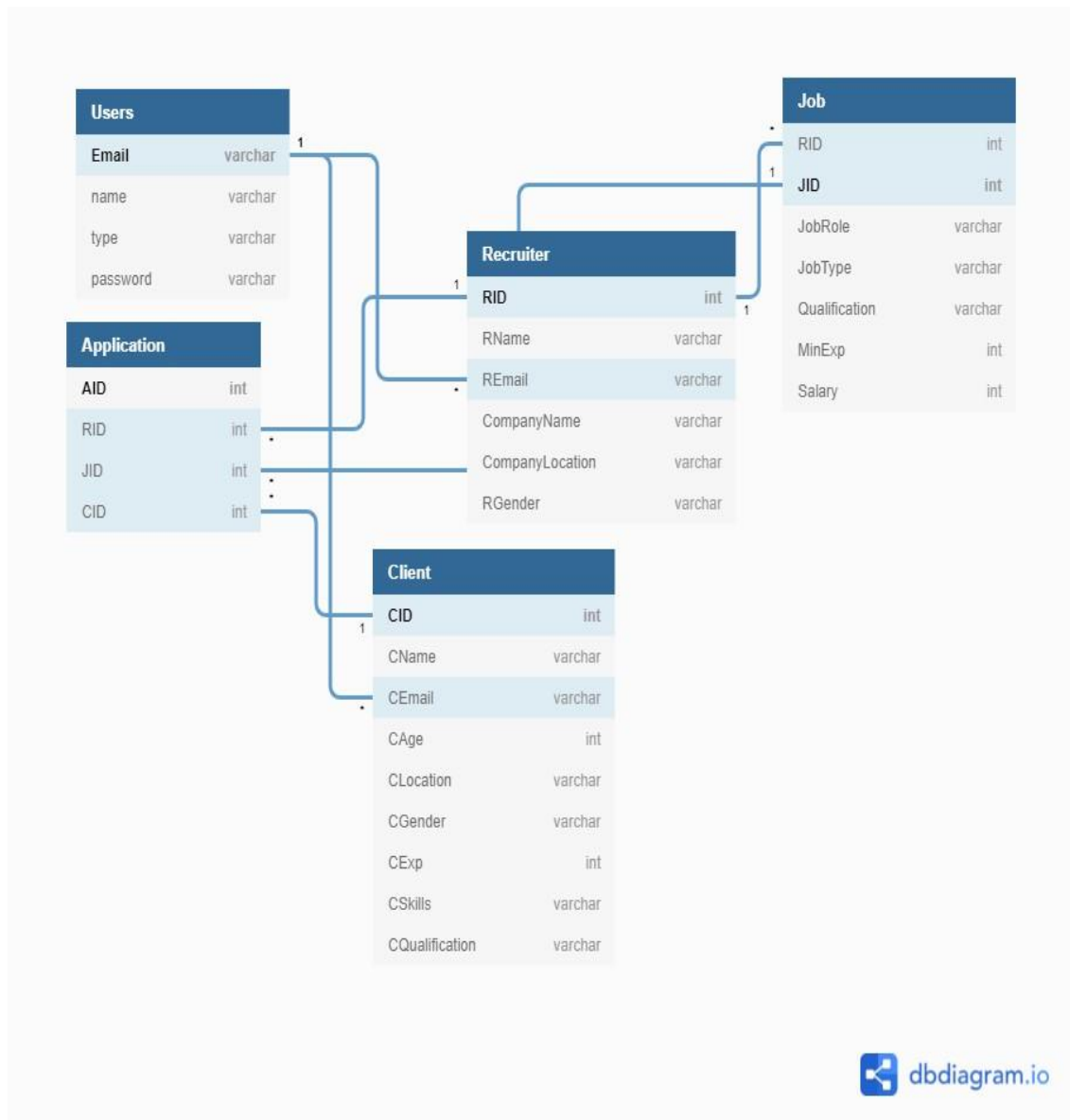


Figure 2.2 Relational Schema Diagram for Recruitment Management System

### 2.2.1 General Constraints

1. **NULL Constraint:** Attributes that are under NOT NULL constraints have to be filled compulsorily. Almost all the attributes in the project are under NOT NULL constraint.
2. **Entity Integrity Constraint:** This constraint makes sure that no primary key can have a NULL value assigned to it. The primary keys involved in the project include:
  - **Code**
  - **AID**
  - **CID**
  - **JID**
  - **RID**
3. **Referential Integrity Constraints:** A table in the back end of the project may have references pointing to an attribute in another table. The various tables are also linked with multiple foreign keys which are all set to cascade any update or delete operation on the attribute in the main table.:

### 2.2.2 Schema Description

The above Figure.2.2 shows the relational schema of Recruitment Management System. It has the following entities.

1. **USERS:** This table contains the details like email, name, type and password.
2. **APPLICATION:** Contains AID, RID JID AND CID.
3. **RECRUITERS:** This table consists RID, RName, Remail, CompanyName, CompanyLocation and RGender.
4. **JOB:** This table consists of all attributes like RID, JID, JobRole, JobType, Qualification, MinExp and Salary.
5. **CLIENT:** This table consists of CID, CName, CEmail. Cage, CLoaction, Cgender, CExp, CSkills, CQualification.

## Chapter 3

### SYSTEM DESIGN

#### 3.1 Table Description

##### USERS

Field	Type	Null	Key	Default	Extra
email	varchar(120)	NO	PRI	NULL	
name	varchar(45)	NO		NULL	
type	varchar(45)	NO		NULL	
password	varchar(45)	YES		NULL	

Table 3.1 Users

##### RECRUITERS

Field	Type	Null	Key	Default	Extra
RID	int	NO	PRI	NULL	auto_increment
RName	varchar(45)	NO		NULL	
REmail	varchar(45)	NO	UNI	NULL	
CompanyName	varchar(45)	NO		NULL	
CompanyLocation	varchar(45)	NO		NULL	
RGender	varchar(2)	NO		NULL	

Table 3.2 Recruiters

**CLIENTS**

Field	Type	Null	Key	Default	Extra
CID	int	NO	PRI	NULL	auto_increment
CName	varchar(45)	NO		NULL	
CEmail	varchar(45)	NO	UNI	NULL	
CAge	int	NO		NULL	
CLocation	varchar(45)	NO		NULL	
CGender	varchar(2)	NO		NULL	
CExp	int	NO		NULL	
CSkills	varchar(45)	NO		NULL	
CQualification	varchar(45)	NO		NULL	

*Table 3.3: Clients***JOB**

Field	Type	Null	Key	Default	Extra
RID	int	NO	MUL	NULL	
JID	int	NO	PRI	NULL	auto_increment
JobRole	varchar(45)	NO		NULL	
JobType	varchar(45)	NO		NULL	
Qualification	varchar(45)	NO		NULL	
MinExp	int	NO		NULL	
Salary	int	NO		NULL	

*Table 3.4: Job*

**APPLICATION**

Field	Type	Null	Key	Default	Extra
AID	int	NO	PRI	NULL	auto_increment
RID	int	NO	MUL	NULL	
JID	int	NO	MUL	NULL	
CID	int	NO	MUL	NULL	

*Table 3.5: Application*

## Chapter 4

# IMPLEMENTATION

### 4.1 Front-end Code

The front-end is built using a combination of Python and Tkinter. Front-end developers design and construct the user experience elements on the web page or app including buttons, menus, pages, links, graphics and more.

#### 4.1.1 Modules

##### Login.py

```
from tkinter import *
from tkinter import messagebox
from tkinter_uix.Entry import Entry
import mysql.connector as sql
from modules.register import *
from modules.recruiter import *
from modules.client import *
from modules.creds import user_pwd

def success(root, email1):
    global f
    f1.destroy()
    try:
        r1.destroy()
    except:
        pass

    s = f'select type from users where email="{email1}"'
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(s)
    q = cur.fetchall()
    mycon.close()
    print(q)

    if q[0][0] == "recruiter":
        rec(root, email1)
    else:
        cli(root, email1)
```

```
def submit(root):
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute('select email,password from users')
    total = cur.fetchall()
    mycon.close()
    email1 = email.get()
    password = pwd.get()
    if email1 and password:
        for i in total:
            if email1 == i[0] and password == i[1]:
                return success(root, email1)
            elif email1 == i[0] and password != i[1]:
                messagebox.showinfo('Alert!', 'Invalid Credentials')
                break
        else:
            messagebox.showinfo(
                'Alert!', 'Email is not registered, Please register')
    else:
        messagebox.showinfo(
            'Alert!', 'Please Enter both Email and Password')

def reg(root):
    try:
        f1.destroy()
    except:
        pass
    mai(root)

def log(root):
    global f1, email, pwd
    try:
        f2.destroy()
    except:
        pass
    f1 = Frame(root, width=1050, height=700, bg='#FFFFFF')
    f1.place(x=0, y=0)

    # Background
    f1.render = PhotoImage(file='elements\\bg.png')
    img = Label(f1, image=f1.render)
    img.place(x=0, y=0)

    # Email
    email_1 = Label(f1, text="Email : ", bg='#FFFFFF',
                    font=('normal', 20, 'bold'), fg="#00B9ED")
    email_1.place(x=620, y=300)
    email = Entry(f1, width=24, placeholder="Enter your Email..")
    email.place(x=720, y=300)
```



```
# Password
pwd_1 = Label(f1, text="Password : ", bg='#FFFFFF',
              font=('normal', 20, 'bold'), fg="#00B9ED")
pwd_1.place(x=565, y=350)
pwd = Entry(f1, show="*", width=24, placeholder="Enter your Password..")
pwd.place(x=720, y=350)

# Buttons
f1.bn = PhotoImage(file="elements\\login2.png")
btn = Button(f1, image=f1.bn, bg='#FFFFFF', bd=0,
             activebackground="#ffffff", command=lambda: submit(root))
btn.place(x=820, y=420)

f1.bn1 = PhotoImage(file="elements\\reg.png")
btn1 = Button(f1, image=f1.bn1, bg='#FFFFFF', bd=0,
              activebackground="#ffffff", command=lambda: reg(root))
btn1.place(x=620, y=420)
```

### Register.py

```
from tkinter import *
from tkinter import ttk
from tkinter import messagebox, Label
from tkinter_uix.Entry import Entry
import mysql.connector as sql
import modules.login as l
from modules.creds import user_pwd

def logi(root):
    try:
        r2.destroy()
        r3.destroy()
    except:
        pass
    l.log(root)

def mai(root):
    try:
        r2.destroy()
    except:
        pass
    global r1
    r1 = Frame(root, height=700, width=1050)
    r1.place(x=0, y=0)
    r1.render = PhotoImage(file="elements/Registration_bg.png")
    img = Label(r1, image=r1.render)
    img.place(x=0, y=0)
    r1.Img1 = PhotoImage(file="elements/recruiter_element.png")
```

```

recruit = Button(r1, image=r1.Img1, border=0, bg="#03DDEE",
                relief="raised", activebackground="#03EAFD", command=lambda:
recruiter_regis(root))
recruit.place(x=140, y=340)
r1.Img2 = PhotoImage(file="elements/client_element.png")
recruit2 = Button(r1, image=r1.Img2, border=0, bg="#05edFC",
                relief="raised", activebackground="#05F6FD", command=lambda:
client_regis(root))
recruit2.place(x=360, y=340)
r1.bn = PhotoImage(file="elements\\backlogin.png")
btn = Button(r1, image=r1.bn, bg='#05e4f6',
            bd=0, activebackground="#05e4f6", command=lambda: logi(root))
btn.place(x=220, y=550)

def recruiter_regis(root):
    global name, email, pwd, cpwd
    print("hello recruiter")
    r1.destroy()
    r2 = Frame(root, height=700, width=1050)
    r2.place(x=0, y=0)
    r2.render = PhotoImage(file="elements/reg_bg.png")
    img = Label(r2, image=r2.render)
    img.place(x=0, y=0)
    name_1 = Label(r2, text="Name : ", bg='FFFFFF', fg="#00B9ED",
                  font=('normal', 20, 'bold'))
    name_1.place(x=100, y=250)
    name = Entry(r2, placeholder='Enter Your Full Name...', width=20)
    name.place(x=290, y=250)

    email_1 = Label(r2, text="Email : ", bg='FFFFFF', fg="#00B9ED",
                    font=('normal', 20, 'bold'))
    email_1.place(x=100, y=300)
    email = Entry(r2, placeholder='Email', width=20)
    email.place(x=290, y=300)

    pwd_1 = Label(r2, text="Password : ", bg='FFFFFF', fg="#00B9ED",
                  font=('normal', 20, 'bold'))
    pwd_1.place(x=100, y=350)
    pwd = Entry(r2, placeholder='Password', show="*", width=20)
    pwd.place(x=290, y=350)

    con_pwd_1 = Label(r2, text="Confirm : ", bg='FFFFFF', fg="#00B9ED",
                      font=('normal', 20, 'bold'))
    con_pwd_1.place(x=100, y=400)
    cpwd = Entry(r2, placeholder='Confirm Password', show="*", width=20)
    cpwd.place(x=290, y=400)

    r2.bn = PhotoImage(file="elements\\next1.png")
    btn = Button(r2, image=r2.bn, bg='FFFFFF', bd=0,
                activebackground="#ffffff", command=lambda: recruiter_check(root))
    btn.place(x=320, y=500)

```

```
r2.back = PhotoImage(file="elements\\back.png")
btn2 = Button(r2, image=r2.back, bg='#FFFFFF', bd=0,
              activebackground="#ffffff", command=lambda: mai(root))
btn2.place(x=120, y=500)

def recruiter_check(root):
    global name1, email1, pwd1, cpwd1
    name1 = name.get()
    email1 = email.get()
    pwd1 = pwd.get()
    cpwd1 = cpwd.get()
    print(name1, email1, pwd1, cpwd1)
    if name1 and email1 and pwd1 and cpwd1:
        mycon = sql.connect(host='localhost', user='root',
                             passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute('select email from users')
        total = cur.fetchall()
        mycon.close()
        exist_email = []
        for i in total:
            exist_email.append(i[0])
        print("existing users:", exist_email)

        if email1 in exist_email:
            messagebox.showinfo('ALERT!', 'EMAIL ALREADY REGISTERED')
            email.delete(0, END)

        else:
            if pwd1 == cpwd1:
                recruit_complete(root)
            else:
                messagebox.showinfo('ALERT!', 'PASSWORDS DO NOT MATCH')

    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS ARE MUST BE FILLED')

def recruit_complete(root):
    print("hello ", name1, ", Let's complete your profile")
    r3 = Frame(root, height=700, width=1050)
    r3.place(x=0, y=0)
    r3.render = PhotoImage(file="elements/reg_bg.png")
    img = Label(r3, image=r3.render)
    img.place(x=0, y=0)

    global gender, company, loc
    gender = StringVar()

    style = ttk.Style(r3)
```

```

style.configure("TRadiobutton", background="white",
                foreground="#696969", font=("arial", 16, "bold"))

gender_1 = Label(r3, text="Gender : ", bg='FFFFFF', fg="#00B9ED",
                font=('normal', 20, 'bold'))
gender_1.place(x=100, y=250)
ttk.Radiobutton(r3, text="Male", value="M", variable=gender).place(
    x=300, y=250)
ttk.Radiobutton(r3, text="Female", value="F", variable=gender).place(
    x=400, y=250)

company_1 = Label(r3, text="Company : ", bg='FFFFFF', fg="#00B9ED",
                font=('normal', 20, 'bold'))
company_1.place(x=100, y=300)
company = Entry(r3, placeholder='Company', width=20)
company.place(x=290, y=300)

loc_1 = Label(r3, text="Location : ", bg='FFFFFF', fg="#00B9ED",
                font=('normal', 20, 'bold'))
loc_1.place(x=100, y=350)
loc = Entry(r3, placeholder='Location', width=20)
loc.place(x=290, y=350)

r3.bn = PhotoImage(file="elements\\reg.png")
btn = Button(r3, image=r3.bn, bg='FFFFFF', bd=0,
            activebackground="ffffff", command=lambda: recruiter_submit(root))
btn.place(x=320, y=500)

def recruiter_submit(root):
    global gender1, company1, loc1
    gender1 = gender.get()
    company1 = company.get()
    loc1 = loc.get()
    print(name1, email1, gender1, company1, loc1)
    if gender1 and company1 and loc1:
        exe = f'insert into users values("{name1}", "{email1}", "recruiter", "{pwd1}")'
        exe1 = f'INSERT INTO mydb.Recruiter(RID, RName, REmail, CompanyName,
        CompanyLocation ,RGender) VALUES
        (NULL, "{name1}", "{email1}", "{company1}", "{loc1}", "{gender1}")'
        try:
            mycon = sql.connect(host='localhost', user='root',
                                passwd=user_pwd, database='mydb')
            cur = mycon.cursor()
            cur.execute(exe)
            cur.execute(exe1)
            name.delete(0, END)
            email.delete(0, END)
            pwd.delete(0, END)
            cpwd.delete(0, END)
            # gender.delete(0, END)
            loc.delete(0, END)

```

```
        company.delete(0, END)
        mycon.commit()
        mycon.close()
        messagebox.showinfo('SUCCESS!', 'Registration Successful')
        logi(root)
    except:
        pass

    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS MUST BE FILLED')

def client_regis(root):
    global name, email, pwd, cpwd
    print("hello client")
    r1.destroy()
    r2 = Frame(root, height=700, width=1050)
    r2.place(x=0, y=0)
    r2.render = PhotoImage(file="elements/reg_bg.png")
    img = Label(r2, image=r2.render)
    img.place(x=0, y=0)

    name_1 = Label(r2, text="Name : ", bg='FFFFFF', fg="#00B9ED",
                    font=('normal', 20, 'bold'))
    name_1.place(x=100, y=250)
    name = Entry(r2, placeholder='Enter Your Full Name...', width=20)
    name.place(x=290, y=250)

    email_1 = Label(r2, text="Email : ", bg='FFFFFF', fg="#00B9ED",
                    font=('normal', 20, 'bold'))
    email_1.place(x=100, y=300)
    email = Entry(r2, placeholder='Email', width=20)
    email.place(x=290, y=300)

    pwd_1 = Label(r2, text="Password : ", bg='FFFFFF', fg="#00B9ED",
                  font=('normal', 20, 'bold'))
    pwd_1.place(x=100, y=350)
    pwd = Entry(r2, placeholder='Password', show="*", width=20)
    pwd.place(x=290, y=350)

    con_pwd_1 = Label(r2, text="Confirm : ", bg='FFFFFF', fg="#00B9ED",
                      font=('normal', 20, 'bold'))
    con_pwd_1.place(x=100, y=400)
    cpwd = Entry(r2, placeholder='Confirm Password', show="*", width=20)
    cpwd.place(x=290, y=400)

    r2.bn = PhotoImage(file="elements\\next1.png")
    btn = Button(r2, image=r2.bn, bg='FFFFFF', bd=0,
                 activebackground="#ffffff", command=lambda: client_check(root))
    btn.place(x=320, y=500)

    r2.back = PhotoImage(file="elements\\back.png")
```

```
btn2 = Button(r2, image=r2.back, bg='#FFFFFF', bd=0,
              activebackground="#ffffff", command=lambda: mai(root))
btn2.place(x=120, y=500)

def client_check(root):
    global name1, email1, pwd1, cpwd1
    name1 = name.get()
    email1 = email.get()
    pwd1 = pwd.get()
    cpwd1 = cpwd.get()
    print(name1, email1, pwd1, cpwd1)
    if name1 and email1 and pwd1 and cpwd1:
        mycon = sql.connect(host='localhost', user='root',
                             passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute('select email from users')
        total = cur.fetchall()
        mycon.close()
        exist_email = []
        for i in total:
            exist_email.append(i[0])
        print("existing users:", exist_email)

        if email1 in exist_email:
            messagebox.showinfo('ALERT!', 'EMAIL ALREADY REGISTERED')
            email.delete(0, END)

        else:
            if pwd1 == cpwd1:
                client_complete(root)
            else:
                messagebox.showinfo('ALERT!', 'PASSWORDS DO NOT MATCH')

    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS MUST BE FILLED')

def client_complete(root):
    print("hello ", name1, ", Let's complete your profile")
    r3 = Frame(root, height=700, width=1050)
    r3.place(x=0, y=0)
    r3.render = PhotoImage(file="elements/reg_bg.png")
    img = Label(r3, image=r3.render)
    img.place(x=0, y=0)

    global gender, age, loc, workxp, qualification, skills
    gender = StringVar()

    style = ttk.Style(r3)
    style.configure("TRadiobutton", background="white",
                    foreground="#696969", font=("arial", 16, "bold"))
```

```
gender_1 = Label(r3, text="Gender : ", bg='FFFFFF', fg="#00B9ED",
                font=('normal', 20, 'bold'))
gender_1.place(x=100, y=200)
ttk.Radiobutton(r3, text="Male", value="M", variable=gender).place(
    x=300, y=200)
ttk.Radiobutton(r3, text="Female", value="F", variable=gender).place(
    x=400, y=200)

age_1 = Label(r3, text="Age : ", bg='FFFFFF', fg="#00B9ED",
              font=('normal', 20, 'bold'))
age_1.place(x=100, y=250)
age = Entry(r3, placeholder='Age', width=20)
age.place(x=290, y=250)

loc_1 = Label(r3, text="Location : ", bg='FFFFFF', fg="#00B9ED",
              font=('normal', 20, 'bold'))
loc_1.place(x=100, y=300)
loc = Entry(r3, placeholder='Location', width=20)
loc.place(x=290, y=300)

workxp_1 = Label(r3, text="Experience : ", bg='FFFFFF', fg="#00B9ED",
                 font=('normal', 20, 'bold'))
workxp_1.place(x=100, y=350)
workxp = Entry(r3, placeholder='Work Experience(yrs)', width=20)
workxp.place(x=290, y=350)

qualification_1 = Label(r3, text="Qualification : ",
                        bg='FFFFFF', fg="#00B9ED", font=('normal', 20, 'bold'))
qualification_1.place(x=100, y=400)
qualification = Entry(r3, placeholder='Btech/BE...', width=20)
qualification.place(x=290, y=400)

skills_1 = Label(r3, text="Skills : ", bg='FFFFFF',
                 fg="#00B9ED", font=('normal', 20, 'bold'))
skills_1.place(x=100, y=450)
skills = Entry(r3, placeholder='separated by comma', width=20)
skills.place(x=290, y=450)

r3.bn = PhotoImage(file="elements\\reg.png")
btn = Button(r3, image=r3.bn, bg='FFFFFF', bd=0,
             activebackground="ffffff", command=lambda: client_submit(root))
btn.place(x=320, y=550)

def client_submit(root):
    global gender1, age1, loc1, workxp1, qualification1, skills1
    gender1 = gender.get()
    age1 = age.get()
    loc1 = loc.get()
    workxp1 = workxp.get()
    qualification1 = qualification.get()
```

```

skills1 = skills.get()
print(name1, email1, gender1, age1, loc1, workxp1, qualification1, skills1)
if gender1 and age1 and loc1 and workxp1:
    exe = f'insert into users values("{name1}", "{email1}", "client", "{pwd1}")'
    exe1 = f'INSERT INTO mydb.Client(CID, CName , CEmail, CAge, CLocation, CGender,
CExp, CSkills, CQualification ) VALUES (NULL, "{name1}", "{email1}", {age1}, "{loc1}",
"{gender1}", {workxp1}, "{skills1}", "{qualification1}")'
    try:
        mycon = sql.connect(host='localhost', user='root',
                             passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute(exe)
        cur.execute(exe1)
        name.delete(0, END)
        email.delete(0, END)
        pwd.delete(0, END)
        cpwd.delete(0, END)
        # gender.delete(0, END)
        loc.delete(0, END)
        age.delete(0, END)
        workxp.delete(0, END)
        qualification.delete(0, END)
        skills.delete(0, END)
        mycon.commit()
        mycon.close()
        messagebox.showinfo('SUCCESS!', 'Registration Successful')
        logi(root)
    except:
        pass

else:
    messagebox.showinfo('ALERT!', 'ALL FIELDS ARE MUST BE FILLED')

```

### Recruiter.py

```

from tkinter import *
from tkinter import ttk
from tkinter import messagebox, Label
from tkinter_uix.Entry import Entry
import mysql.connector as sql
import modules.login as l
from modules.creds import user_pwd

def get_details(email):
    global name, company, gen, recid
    q = f'select RName,CompanyName,RGender,RID from mydb.recruiter where
REmail="{email}"'
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(q)
    d = cur.fetchall()

```



```
mycon.close()
```

```
name = d[0][0]
company = d[0][1]
gen = d[0][2]
recid = d[0][3]
```

```
def logi(root):
```

```
    try:
        bg.destroy()
    except:
        pass
    l.log(root)
```

```
def submit_job():
```

```
    global role1, jtype1, qual1, exp1, sal1
    role1 = role.get()
    jtype1 = jtype.get()
    qual1 = qual.get()
    exp1 = exp.get()
    sal1 = sal.get()
    print(role1, jtype1, qual1, exp1, sal1)
    if role1 and jtype1 and qual1 and exp1 and sal1:
        if jtype1 == "Select":
            messagebox.showinfo('ALERT!', 'Please provide Job Type')
        else:
            exe1 = f'INSERT INTO mydb.Job(RID, JID, JobRole, JobType, Qualification,
MinExp, Salary) VALUES({recid}, NULL, "{role1}", "{jtype1}", "{qual1}", {exp1}, {sal1})'
            try:
                mycon = sql.connect(host='localhost', user='root',
                                passwd=user_pwd, database='mydb')
                cur = mycon.cursor()
                cur.execute(exe1)
                role.delete(0, END)
                jtype.delete(0, END)
                qual.delete(0, END)
                exp.delete(0, END)
                sal.delete(0, END)
                mycon.commit()
                mycon.close()
                messagebox.showinfo('SUCCESS!', 'You have successfully created a Job')
            except:
                pass
    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS ARE MUST BE FILLED')
```

```
# ..... Sort Queries .....
```

```
---
```

```
def sort_all(table):
```

```
    criteria = search_d.get()
```

```

if(criteria == "Select"):
    pass
else:
    table.delete(*table.get_children())
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')

    cur = mycon.cursor()
    cur.execute(
        f'select RID,JID, JobRole, JobType, Qualification, MinExp, Salary FROM mydb.Job
where RID={recid} order by {criteria}')
    all_jobs = cur.fetchall()
    mycon.close()
    i = 0
    for r in all_jobs:
        table.insert("", i, text="", values=(
            r[1], r[2], r[3], r[4], r[5], r[6]))
        i += 1

```

```

def sort_applicants(table):
    criteria = search_d.get()
    if(criteria == "Select"):
        pass
    else:
        table.delete(*table.get_children())
        mycon = sql.connect(host='localhost', user='root',
                            passwd=user_pwd, database='mydb')

        cur = mycon.cursor()
        cur.execute(
            f'SELECT job.JobRole, client.CName, client.CEmail, client.CAge, client.CLocation,
client.CGender, client.CExp, client.CSkills, client.CQualification FROM application JOIN
client ON application.cid=client.CID JOIN job ON job.jid=application.jid where
job.rid={recid} order by {criteria}')
        applicats = cur.fetchall()
        mycon.close()
        print(applicats)
        i = 0
        for x in applicats:
            table.insert("", i, text="", values=(
                x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8]))
            i += 1

```

```

#.....Posted jobs Query.....
---
```

```

def show_all(table):
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')

    cur = mycon.cursor()
    cur.execute(

```

```

        fselect RID,JID, JobRole, JobType, Qualification, MinExp, Salary FROM mydb.Job
where RID={recid}')
    all_jobs = cur.fetchall()
    mycon.close()
    i = 0
    for r in all_jobs:
        table.insert("i, text=", values=(
            r[1], r[2], r[3], r[4], r[5], r[6]))
        i += 1

```

# ..... Applicants .....

```

def show_applicants(table):
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'SELECT job.JobRole, client.CName, client.CEmail, client.CAge, client.CLocation,
client.CGender, client.CExp, client.CSkills, client.CQualification FROM application JOIN
client ON application.cid=client.CID JOIN job ON job.jid=application.jid where
job.rid={recid}')
    applicats = cur.fetchall()
    mycon.close()
    print(applicats)
    i = 0
    for x in applicats:
        table.insert("i, text=", values=(
            x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7], x[8]))
        i += 1

```

# ..... Post a Job .....

```

def create():
    global role, jtype, qual, exp, sal
    for widget in rt.winfo_children():
        widget.destroy()
    for widget in tab.winfo_children():
        widget.destroy()
    bgr.destroy()

    # Create Form
    f1 = Frame(rt, width=520)
    f1.load = PhotoImage(file="elements\\create.png")
    img = Label(rt, image=f1.load, bg="#FFFFFF")
    img.grid(row=0, column=1, padx=150, pady=10)

    # Form
    # Labels
    role_1 = Label(tab, text="Role :", font=(
        'normal', 18, 'bold'), bg="#FFFFFF")
    role_1.grid(row=0, column=0, pady=10, padx=10)

```

```

type_1 = Label(tab, text="Type :", font=(
    'normal', 18, 'bold'), bg="#FFFFFF")
type_1.grid(row=1, column=0, pady=10, padx=10)
qual_1 = Label(tab, text="Qualification :", font=(
    'normal', 18, 'bold'), bg="#FFFFFF")
qual_1.grid(row=2, column=0, pady=10, padx=10)
exp_1 = Label(tab, text="Experience :", font=(
    'normal', 18, 'bold'), bg="#FFFFFF")
exp_1.grid(row=3, column=0, pady=10, padx=10)
sal_1 = Label(tab, text="Salary :", font=(
    'normal', 18, 'bold'), bg="#FFFFFF")
sal_1.grid(row=4, column=0, pady=10, padx=10)

# Entries
style = ttk.Style(tab)
style.configure("TCombobox", background="white",
    foreground="#696969")

role = Entry(tab, placeholder="Enter Job Role")
role.grid(row=0, column=1, pady=10, padx=10)
jtype = ttk.Combobox(tab, font=("normal", 18),
    width=23, state='readonly')
jtype['values'] = ('Select', 'FullTime', 'PartTime', 'Intern')
jtype.current(0)
jtype.grid(row=1, column=1, pady=10, padx=10)
qual = Entry(tab, placeholder="Enter Job Qualifications")
qual.grid(row=2, column=1, pady=10, padx=10)
exp = Entry(tab, placeholder="Enter Minimum Experience")
exp.grid(row=3, column=1, pady=10, padx=10)
sal = Entry(tab, placeholder="Enter Expected salary")
sal.grid(row=4, column=1, pady=10, padx=10)

btn = Button(tab, text="Submit", font=(20), bg="#45CE30",
    fg="#FFFFFF", command=submit_job)
btn.grid(row=5, column=1, pady=15)

```

# ..... Delete A Posted Job .....

```

def deletjob(table):
    selectedindex = table.focus()
    selectedvalues = table.item(selectedindex, 'values')
    ajid = selectedvalues[0]
    mycon = sql.connect(host='localhost', user='root',
        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(f'delete from mydb.application where jid={ajid}')
    cur.execute(f'delete from mydb.job where jid={ajid}')
    mycon.commit()
    mycon.close()
    messagebox.showinfo('Thanks', 'Your Job has been Deleted')

```

posted()

# ..... Posted Jobs by Recruiter .....  
 -----

```
def posted():
    for widget in rt.winfo_children():
        widget.destroy()
    for widget in tab.winfo_children():
        widget.destroy()
    bgr.destroy()

    search_l = Label(rt, text="Order By : ", font=(
        'normal', 18), bg="#ffffff")
    search_l.grid(row=0, column=0, padx=10, pady=10)
    global search_d
    search_d = ttk.Combobox(rt, width=12, font=(
        'normal', 18), state='readonly')
    search_d['values'] = ('Select', 'JobRole', 'JobType')
    search_d.current(0)
    search_d.grid(row=0, column=2, padx=0, pady=10)
    search = Button(rt, text="Sort", font=('normal', 12, 'bold'),
        bg="#00b9ed", fg="#ffffff", command=lambda: sort_all(table))
    search.grid(row=0, column=3, padx=10, pady=10, ipadx=15)
    dlt = Button(rt, text="Delete", font=('normal', 12, 'bold'),
        bg="#00b9ed", fg="#ffffff", command=lambda: deletjob(table))
    dlt.grid(row=0, column=4, padx=10, pady=10, ipadx=5)

    scx = Scrollbar(tab, orient="horizontal")
    scy = Scrollbar(tab, orient="vertical")

    table = ttk.Treeview(tab, columns=('JID', 'JobRole', 'JobType', 'Qualification', 'MinExp',
    'Salary'),
        xscrollcommand=scx.set, yscrollcommand=scy.set)
    scx.pack(side="bottom", fill="x")
    scy.pack(side="right", fill="y")
    table.heading("JID", text="JobID")
    table.heading("JobRole", text="Role")
    table.heading("JobType", text="Type")
    table.heading("Qualification", text='Qualification')
    table.heading("MinExp", text='MinExp')
    table.heading("Salary", text="Salary")

    table['show'] = 'headings'

    scx.config(command=table.xview)
    scy.config(command=table.yview)

    table.column("JID", width=100)
    table.column("JobRole", width=150)
    table.column("JobType", width=150)
```

```

table.column("Qualification", width=100)
table.column("MinExp", width=100)
table.column("Salary", width=150)
show_all(table)
table.pack(fill="both", expand=1)

```

#.....Applications on your recruiters posted jobs.....

```

def app():
    for widget in rt.winfo_children():
        widget.destroy()
    for widget in tab.winfo_children():
        widget.destroy()
    bgr.destroy()

    search_l = Label(rt, text="Order By : ", font=('normal', 18), bg="#ffffff")
    search_l.grid(row=0, column=0, padx=10, pady=10)
    global search_d
    search_d = ttk.Combobox(rt, width=12, font=(
        'normal', 18), state='readonly')
    search_d['values'] = ('Select', 'JobRole', 'CName', 'CLocation')
    search_d.current(0)
    search_d.grid(row=0, column=2, padx=10, pady=10)
    search = Button(rt, text="Sort", font=('normal', 12, 'bold'),
        bg="#00b9ed", fg="#ffffff", command=lambda: sort_applicants(table))
    search.grid(row=0, column=3, padx=45, pady=10, ipadx=30)

    scx = Scrollbar(tab, orient="horizontal")
    scy = Scrollbar(tab, orient="vertical")

    table = ttk.Treeview(tab, columns=('JobRole', 'CName', 'CEmail', 'CAge', 'CLocation',
        'CGender', 'CExp', 'CSkills', 'CQualification'),
        xscrollcommand=scx.set, yscrollcommand=scy.set)
    scx.pack(side="bottom", fill="x")
    scy.pack(side="right", fill="y")

    table.heading("JobRole", text="Job Role")
    table.heading("CName", text='Applicants Name')
    table.heading("CEmail", text='Email')
    table.heading("CAge", text='Age')
    table.heading("CLocation", text='Location')
    table.heading("CGender", text='Gender')
    table.heading("CExp", text='Experience')
    table.heading("CSkills", text='Skills')
    table.heading("CQualification", text='Qualification')

    table['show'] = 'headings'

    scx.config(command=table.xview)
    scy.config(command=table.yview)

```

```

table.column("JobRole", width=150)
table.column("CName", width=200)
table.column("CEmail", width=100)
table.column("CAge", width=50)
table.column("CLocation", width=150)
table.column("CGender", width=100)
table.column("CExp", width=100)
table.column("CSkills", width=200)
table.column("CQualification", width=150)
show_applicants(table)
table.pack(fill="both", expand=1)

```

```

# .....
def rec(root, email1):
    global email
    email = email1
    bg = Frame(root, width=1050, height=700)
    bg.place(x=0, y=0)

    get_details(email)

    bg.load = PhotoImage(file=f'elements\\bg{gen}.png')
    img = Label(root, image=bg.load)
    img.place(x=0, y=0)

    # Navbar
    nm = Label(root, text=f'{name}', font=(
        'normal', 36, 'bold'), bg="ffffff", fg="#0A3D62")
    nm.place(x=300, y=50)
    cp = Label(root, text=f'{company}', font=(
        'normal', 24), bg="ffffff", fg="#0A3D62")
    cp.place(x=300, y=120)
    bn = Button(root, text="LOGOUT", font=(
        'normal', 20), bg="#b32e2e", fg="ffffff", command=lambda: logi(root))
    bn.place(x=800, y=75)

    # Left
    lf = Frame(root, width=330, height=440, bg="ffffff")
    lf.place(x=60, y=220)
    cj = Button(lf, text="Post a Job", font=(
        'normal', 20), bg="#b32e2e", fg="ffffff", command=create)
    cj.grid(row=0, column=0, padx=80, pady=40)
    pj = Button(lf, text="Posted Jobs", font=(
        'normal', 20), bg="#b32e2e", fg="ffffff", command=posted)
    pj.grid(row=1, column=0, padx=80, pady=40)
    ap = Button(lf, text="Applications", font=(
        'normal', 20), bg="#b32e2e", fg="ffffff", command=app)
    ap.grid(row=2, column=0, padx=80, pady=40)

    # Right

```

```

global rt, tab, bgr
rt = Frame(root, width=540, height=420, bg="#ffffff")
rt.place(x=450, y=220)
tab = Frame(root, bg="#FFFFFF")
tab.place(x=460, y=300, width=520, height=350)
bgrf = Frame(root, width=540, height=420)
bgrf.load = PhotoImage(file="elements\\bgr.png")
bgr = Label(root, image=bgrf.load, bg="#00b9ed")
bgr.place(x=440, y=210)

```

### Client.py

```

from tkinter import *
from tkinter import ttk
from tkinter import messagebox, Label
from tkinter_uix.Entry import Entry
import mysql.connector as sql
import modules.login as l
from modules.creds import user_pwd

def get_details(email):
    global name, location, gen, clicid
    q = f'select CName,CLocation,CGender,CID from mydb.client where CEmail="{email}"'
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(q)
    d = cur.fetchall()
    mycon.close()

    name = d[0][0]
    location = d[0][1]
    gen = d[0][2]
    clicid = d[0][3]

def logi(root):
    try:
        bg.destroy()
    except:
        pass
    l.log(root)

#.....Apply a Job.....
def apply(table):
    # fetch cid,jid from treeview that is in available jobs function
    # code
    selectedindex = table.focus() # that will return number index

```



```
# that will return list of values with columns=['JID','JobRole', 'JobType', 'CompanyName',
'CompanyLocation', 'Qualification','MinExp', 'Salary']
selectedvalues = table.item(selectedindex, 'values')
ajid = selectedvalues[0]
chkquery = f'SELECT * from mydb.application where cid={clcid} and jid={ajid}'
mycon = sql.connect(host='localhost', user='root',
                    passwd=user_pwd, database='mydb')
cur = mycon.cursor()
cur.execute(chkquery)
tempbuff = cur.fetchall()
mycon.close()
if(tempbuff):
    messagebox.showinfo(
        'Oops', 'It seems like you have already applied to this job')
else:
    queryapplyjob = f'Insert into application values(NULL,(select rid from mydb.job where
job.jid={ajid}),{ajid},{clcid})'
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(queryapplyjob)
    mycon.commit()
    mycon.close()
    messagebox.showinfo('Thanks', 'Your application has been submitted')
```

# ..... Delete A Job .....

```
def delet(table):
    selectedindex = table.focus()
    selectedvalues = table.item(selectedindex, 'values')
    aaid = selectedvalues[0]
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'delete from mydb.application where aid={aaid}')
    mycon.commit()
    mycon.close()
    messagebox.showinfo('Thanks', 'Your application has been Deleted')
    myapp()
```

# ..... Sort Queries .....

```
def sort_alljobs(table):
    criteria = search_d.get()
    if(criteria == "Select"):
        pass
    else:
        table.delete(*table.get_children())
        mycon = sql.connect(host='localhost', user='root',
```

```

        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'select job.JID,job.JobRole,job.JobType, recruiter.CompanyName,
recruiter.CompanyLocation, job.Qualification, job.MinExp, job.Salary from mydb.job JOIN
mydb.recruiter ON job.rid=recruiter.rid order by {criteria}')
    jobs = cur.fetchall()
    mycon.close()
    i = 0
    for r in jobs:
        table.insert(" i, text="", values=(
            r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7]))
        i += 1

def sort_myapplications(table):
    criteria = search_d.get()
    if(criteria == "Select"):
        pass
    else:
        table.delete(*table.get_children())
        mycon = sql.connect(host='localhost', user='root',
            passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute(
            f'SELECT application.aid,job.JobRole, job.JobType, recruiter.CompanyName,
recruiter.CompanyLocation, job.qualification, job.minexp, job.salary FROM application JOIN
recruiter ON application.rid=recruiter.rid JOIN job ON application.jid=job.jid where
application.CID={clidid} order by {criteria}')
        jobs = cur.fetchall()
        mycon.close()
        i = 0
        for r in jobs:
            table.insert(" i, text="", values=(
                r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7]))
            i += 1

```

# ..... Show all Jobs .....

```

def showalljobs(table):
    mycon = sql.connect(host='localhost', user='root',
        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'select job.JID,job.JobRole,job.JobType, recruiter.CompanyName,
recruiter.CompanyLocation, job.Qualification, job.MinExp, job.Salary from mydb.job JOIN
mydb.recruiter ON job.rid=recruiter.rid')
    jobs = cur.fetchall()
    mycon.close()
    i = 0
    for r in jobs:

```

```

table.insert(" , i, text="", values=(
    r[0], r[1], r[2], r[3], r[4], r[5], r[6], r[7]))
i += 1

```

#.....Show my Applications.....  
-----

```

def show_myapplications(table):
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'SELECT application.aid,job.JobRole, job.JobType, recruiter.CompanyName,
        recruiter.CompanyLocation, job.qualification, job.minexp, job.salary FROM application JOIN
        recruiter ON application.rid=recruiter.rid JOIN job ON application.jid=job.jid where
        application.CID={clcid}')
    applications = cur.fetchall()
    mycon.close()
    print(applications)
    i = 0
    for x in applications:
        table.insert(" , i, text="", values=(
            x[0], x[1], x[2], x[3], x[4], x[5], x[6], x[7]))
        i += 1

```

#.....Available Jobs.....  
---

```

def available():
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    for widget in rt.winfo_children():
        widget.destroy()
    for widget in tab.winfo_children():
        widget.destroy()
    bgr.destroy()

    search_l = Label(rt, text="Order By : ", font=(
        'normal', 18), bg="#ffffff")
    search_l.grid(row=0, column=0, padx=10, pady=10)
    global search_d
    search_d = ttk.Combobox(rt, width=12, font=(
        'normal', 18), state='readonly')
    search_d['values'] = ('Select', 'JobRole', 'JobType', 'CompanyLocation')
    search_d.current(0)
    search_d.grid(row=0, column=2, padx=0, pady=10)
    search = Button(rt, text="Sort", font=('normal', 12, 'bold'),
                    bg="#00b9ed", fg="#ffffff", command=lambda: sort_alljobs(table))
    search.grid(row=0, column=3, padx=10, pady=10, ipadx=15)

```

```

apl = Button(rt, text="Apply", font=('normal', 12, 'bold'),
             bg="#00b9ed", fg="#ffffff", command=lambda: apply(table))
apl.grid(row=0, column=4, padx=10, pady=10, ipadx=5)

scx = Scrollbar(tab, orient="horizontal")
scy = Scrollbar(tab, orient="vertical")

table = ttk.Treeview(tab, columns=('JID', 'JobRole', 'JobType', 'CompanyName',
'CompanyLocation', 'Qualification', 'MinExp', 'Salary'),
                    xscrollcommand=scx.set, yscrollcommand=scy.set)
scx.pack(side="bottom", fill="x")
scy.pack(side="right", fill="y")
table.heading("JID", text="JID")
table.heading("JobRole", text="JobRole")
table.heading("JobType", text="JobType")
table.heading("CompanyName", text='CompanyName')
table.heading("CompanyLocation", text="CompanyLocation")
table.heading("Qualification", text='Qualification')
table.heading("MinExp", text='MinExp')
table.heading("Salary", text="Salary")

table['show'] = 'headings'

scx.config(command=table.xview)
scy.config(command=table.yview)

table.column("JID", width=100)
table.column("JobRole", width=150)
table.column("JobType", width=150)
table.column("CompanyName", width=150)
table.column("CompanyLocation", width=150)
table.column("Qualification", width=100)
table.column("MinExp", width=100)
table.column("Salary", width=150)
showalljobs(table)
table.pack(fill="both", expand=1)
mycon.close()

# ..... My Applications .....
def myapp():
    mycon = sql.connect(host='localhost', user='root',
                       passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    for widget in rt.winfo_children():
        widget.destroy()
    for widget in tab.winfo_children():
        widget.destroy()
    bgr.destroy()

```

```
search_1 = Label(rt, text="Order By : ", font=('normal', 18), bg="#ffffff")
search_1.grid(row=0, column=0, padx=10, pady=10)
global search_d
search_d = ttk.Combobox(rt, width=12, font=(
    'normal', 18), state='readonly')
search_d['values'] = ('Select', 'JobRole', 'JobType', 'CompanyLocation')
search_d.current(0)
search_d.grid(row=0, column=2, padx=0, pady=10)
search = Button(rt, text="Sort", font=('normal', 12, 'bold'), bg="#00b9ed",
    fg="#ffffff", command=lambda: sort_myapplications(table))
search.grid(row=0, column=3, padx=10, pady=10, ipadx=15)

dlt = Button(rt, text="Delete", font=('normal', 12, 'bold'),
    bg="#00b9ed", fg="#ffffff", command=lambda: delet(table))
dlt.grid(row=0, column=4, padx=10, pady=10, ipadx=5)

scx = Scrollbar(tab, orient="horizontal")
scy = Scrollbar(tab, orient="vertical")

table = ttk.Treeview(tab, columns=('AID', 'JobRole', 'JobType', 'CompanyName',
    'CompanyLocation', 'Qualification', 'MinExp', 'Salary'),
    xscrollcommand=scx.set, yscrollcommand=scy.set)
scx.pack(side="bottom", fill="x")
scy.pack(side="right", fill="y")
table.heading("AID", text="AID")
table.heading("JobRole", text="JobRole")
table.heading("JobType", text="JobType")
table.heading("CompanyName", text="CompanyName")
table.heading("CompanyLocation", text="CompanyLocation")
table.heading("Qualification", text="Qualification")
table.heading("MinExp", text="MinExp")
table.heading("Salary", text="Salary")
table['show'] = 'headings'

scx.config(command=table.xview)
scy.config(command=table.yview)

table.column("AID", width=50)
table.column("JobRole", width=150)
table.column("JobType", width=150)
table.column("CompanyName", width=150)
table.column("CompanyLocation", width=150)
table.column("Qualification", width=100)
table.column("MinExp", width=100)
table.column("Salary", width=150)
show_myapplications(table)
table.pack(fill="both", expand=1)
mycon.close()
```

```

# .....
def cli(root, email1):
    global email
    email = email1
    bg = Frame(root, width=1050, height=700)
    bg.place(x=0, y=0)

    get_details(email)

    bg.load = PhotoImage(file=f'elements\\bg{gen}.png')
    img = Label(root, image=bg.load)
    img.place(x=0, y=0)

    # Navbar
    nm = Label(root, text=f'{name}', font=(
        'normal', 36, 'bold'), bg="#ffffff", fg="#0A3D62")
    nm.place(x=300, y=50)
    cp = Label(root, text=f'{location}', font=(
        'normal', 24), bg="#ffffff", fg="#0A3D62")
    cp.place(x=300, y=120)
    bn = Button(root, text="LOGOUT", font=('normal', 20),
        bg="#b32e2e", fg="#ffffff", command=lambda: logi(root))
    bn.place(x=800, y=75)

    # Left
    lf = Frame(root, width=330, height=440, bg="#ffffff")
    lf.place(x=60, y=240)
    pj = Button(lf, text="Available Jobs", font=(
        'normal', 20), bg="#b32e2e", fg="#ffffff", command=available)
    pj.grid(row=0, column=0, padx=60, pady=70)
    ap = Button(lf, text="My Applications", font=(
        'normal', 20), bg="#b32e2e", fg="#ffffff", command=myapp)
    ap.grid(row=1, column=0, padx=60, pady=70)

    # Right
    global rt, tab, bgr
    rt = Frame(root, width=540, height=420, bg="#ffffff")
    rt.place(x=450, y=220)
    tab = Frame(root, bg="#FFFFFF")
    tab.place(x=460, y=300, width=520, height=350)
    bgrf = Frame(root, width=540, height=420)
    bgrf.load = PhotoImage(file="elements\\bgr.png")
    bgr = Label(root, image=bgrf.load, bg="#00b9ed")
    bgr.place(x=440, y=210)
    # root = Tk()
    # root.geometry("1050x700")
    # root.title("Client")
    # root.resizable(0, 0)
    # cli()
    # root.mainloop()

```

**Creds.py**

```
user_pwd = "yourMySQLPassword" # use your MySQL Password
```

**4.1.2 Tkinter****\_\_init\_\_.py**

```
import yaml
```

```
class Theme:
```

```
    def __init__(self, name='default'):
        with open(f'tkinter_uix/themes/{name}.yaml', 'r') as file:
            self.theme = yaml.load(file, Loader=yaml.FullLoader)

        self.app_color = self.theme['App']
        self.btn_color = self.theme['Button']
        self.entry_color = self.theme['Entry']
        self.navbar_color = self.theme['Navbar']
```

**Button.py**

```
from tkinter import Label
from tkinter import FLAT
from typing import NoReturn, Any, Callable
from tkinter_uix import Theme
```

```
theme = Theme()
```

```
class Button(Label):
```

```
    """The Button is a custom button created using a Label widget but has the same
    functionality as a regular button.
    All the Label methods, arguments and keyword arguments are supported as the Button is an
    actual Label widget"""
```

```
    def __init__(self, master: Any, text: str = "", command: (Callable, None) = None, color: str =
    'default',
```

```
        disabled: bool = False, *args: Any, **kwargs: Any):
        """
```

```
        :param Any master: The parent Window where the widget will be placed on.
        :param str text: The text of the button.
        :param command: A function to be trigger when the button is pressed.
        :type command: Callable or None
        :param str color: The available colors are - cloud, info, primary, danger, warning, success,
        elegant and default
        :param bool disabled: If the button is disabled it does not respond to click and it must be a
        bool value.
        :param Any args: Any extra arguments.
        :param Any kwargs: Any extra keyword arguments.
        """
```

```
Label._init_(self, master, text=text, padx=14, pady=6, font=('Verdana', 12),
relief=FLAT, *args, **kwargs)
    self.disabled = disabled
    self.colors = theme.btn_color

    if color not in self.colors:
        color = 'default'

    self.on_hover_color = self.colors[color]['on_hover']
    if not disabled:
        self.background_color = self.colors[color]['background']
        self.foreground_color = self.colors[color]['foreground']
        self.bind('<Enter>', self.on_hover)
        self.bind('<Leave>', self.off_hover)
        self.bind('<Button-1>', lambda event: self.on_click(command))
    elif disabled:
        self.background_color = self.colors[color]['disabled_bg']
        self.foreground_color = self.colors[color]['disabled_fg']

    self.off_hover()

def on_hover(self, *args) -> NoReturn:
    """When mouse hover over the button it changes color and display a hand pointer."""
    if not self.disabled:
        self.configure(bg=self.on_hover_color, cursor="hand2")

def off_hover(self, *args) -> NoReturn:
    """when mouse move away from the button it revert back to main color and default mouse
pointer."""
    self.configure(bg=self.background_color, fg=self.foreground_color)

def on_click(self, command: callable) -> NoReturn:
    """When button is clicked it triggers the function passed on the command argument."""
    if not self.disabled:
        if command:
            command()

@property
def text(self) -> str:
    """Return button text"""
    return self.cget('text')

@text.setter
def text(self, text="") -> NoReturn:
    """Set the button text to the given value"""
    self.configure(text=text)

if __name__ == '__main__':
    pass
```



**Entry.py**

```
from tkinter import Entry as _Entry
from tkinter import Frame as _Frame
from tkinter import YES, FLAT, END, X
from tkinter_uix import Theme

theme = Theme()

class Entry(_Frame):
    def __init__(self, master, bg=theme.entry_color['background'], show=None, width=30,
on_return=None,
                fg=theme.entry_color['foreground'],
                placeholder="", *args, **kwargs):
        _Frame.__init__(self, master, bg=bg, *args, **kwargs)

        self.show = show
        self.width = width
        self.fg = fg
        self.placeholder = placeholder

        self.entry_frame = _Frame(self, borderwidth=2, bg=theme.entry_color['border'])
        self.entry_frame.pack(fill=X, expand=YES)

        self.entry = _Entry(self.entry_frame, borderwidth=6, relief=FLAT,
bg=theme.entry_color['background'],
                fg=theme.entry_color['foreground'], font=('Verdana', 12),
                show=show, width=width)
        self.entry.pack(fill=X, expand=YES)
        self.on_focus_out()

        if on_return:
            self.entry.bind('<Return>', lambda event: on_return(*args, **kwargs))

        # Default bindings to control border on focus and placeholder.
        self.entry.bind('<FocusIn>', self.on_focus_in)
        self.entry.bind('<FocusOut>', self.on_focus_out)

    def on_focus_in(self, *args, **kwargs):
        text = self.entry.get()
        self.configure(borderwidth=2, bg=theme.entry_color['outline'])
        if text and text == self.placeholder:
            self.entry.delete(0, END)

        self.entry.configure(fg=self.fg)
        self.entry.icursor(END)
        self.entry.configure(show=self.show)

    def on_focus_out(self, *args, **kwargs):
        text = self.entry.get()
        if text:
            if text == self.placeholder:
```

```
        self.entry.configure(fg=theme.entry_color['placeholder'])
        self.entry.configure(show="")
    else:
        self.entry.configure(fg=self.fg)
        self.entry.configure(show=self.show)
    else:
        self.entry.insert(0, self.placeholder)
        self.entry.configure(fg=theme.entry_color['placeholder'])
        self.entry.configure(show="")

    self.configure(borderwidth=0)

def get(self, *args, **kwargs):
    text = self.entry.get()
    if text:
        if text == self.placeholder:
            return ""
        else:
            return self.entry.get()
    else:
        return ""

def focus(self, *args, **kwargs):
    self.entry.focus_force()

def delete(self, *args, **kwargs):
    self.entry.delete(0, END)

def insert(self, value="", *args, **kwargs):
    text = self.entry.get()
    if text:
        if text == self.placeholder:
            self.entry.delete(0, END)
            self.entry.insert(0, value)
            self.on_focus_out()
        else:
            self.entry.insert(END, value)
            self.on_focus_out()
    else:
        self.entry.insert(0, value)
        self.on_focus_out()

if __name__ == '__main__':
    pass
```

### **label.py**

```
from tkinter import Label as _Label
from tkinter_uix import Theme
```

```
theme = Theme()
```

```
class Label(_Label):
    def __init__(self, master, *args, **kwargs):
        _Label.__init__(self, master, font=('Verdana', 12), bg=theme.app_color['background'],
                        fg=theme.app_color['foreground'], *args, **kwargs)
```

```
if __name__ == '__main__':
    pass
```

### **Layout.py**

```
import tkinter
from tkinter_uix import Theme
```

```
theme = Theme()
```

```
class BoxLayout(tkinter.Frame):
    def __init__(self, master, bg=theme.app_color['background'], *args, **kwargs):
        tkinter.Frame.__init__(self, master, bg=bg, *args, **kwargs)
        self.show()
```

```
    def show(self):
        self.pack(fill=tkinter.BOTH, expand=tkinter.YES)
```

```
    def hide(self):
        self.pack_forget()
```

```
class AnchorLayout(tkinter.Frame):
    def __init__(self, master, bg=theme.app_color['background'], anchor=None,
        anchor_x='LEFT', anchor_y='TOP',
        *args, **kwargs):
        tkinter.Frame.__init__(self, master, bg=bg, *args, **kwargs)
        self.anchor = anchor
        self.anchor_x = anchor_x
        self.anchor_y = anchor_y
        self.show()

    def show(self):
        anchors = {'LEFT': 'w', 'RIGHT': 'e'}

        if self.anchor == 'CENTER':
            self.pack(side=tkinter.TOP, expand=tkinter.YES, anchor=tkinter.CENTER)
        else:
            x = anchors.get(self.anchor_x)
            y = self.anchor_y
            if x and y in ['TOP', 'BOTTOM']:
                self.pack(side=y.lower(), anchor=x)
            else:
```

```
        raise TypeError('Invalid anchor!')

    def hide(self):
        self.pack_forget()

if __name__ == '__main__':
    pass
```

### Navbar.py

```
import tkinter
from PIL import Image, ImageTk
from tkinter_uix import Theme

theme = Theme()

class Navbar(tkinter.Frame):
    def __init__(self, master, bg=theme.navbar_color['background'], *args, **kwargs):
        tkinter.Frame.__init__(self, master, bg=bg, *args, **kwargs)

        default_image = Image.open('./tkinter_uix/images/python.png')
        self.image = ImageTk.PhotoImage(default_image)
        self.header_title = tkinter.Label(self, image=self.image, text='Title',
        compound=tkinter.LEFT,
                                     font=('Verdana', 12), bg=theme.navbar_color['background'],
                                     fg=theme.navbar_color['foreground'])
        self.header_title.pack(side=tkinter.LEFT)

        self.show()

    def show(self):
        self.pack(side=tkinter.TOP, fill=tkinter.X)

    def hide(self):
        self.pack_forget()

    @staticmethod
    def add_widget(widget):
        widget.pack(side=tkinter.RIGHT, padx=3, pady=3)

    def edit(self, **kwargs):
        self.header_title.configure(**kwargs)

if __name__ == '__main__':
    pass
```

**Notebook.py**

```
from tkinter.ttk import Notebook as NB
from tkinter import ttk
from tkinter_uix import Theme

theme = Theme()

class Notebook(ttk.Notebook):
    def __init__(self, master, **kwargs):
        ttk.Notebook._init(self, master, **kwargs)

        s = ttk.Style()
        s.configure('TNotebook.Tab', font=('Verdana', '12'), padding=(14, 6),
foreground=theme.app_color['foreground'])
        s.configure('TNotebook', background=theme.app_color['background'])
        s.map("TNotebook.Tab", foreground=[('selected', theme.entry_color['outline'])])

if __name__ == '__main__':
    pass
```

**PopupMenu.py**

```
import tkinter
from tkinter_uix import Theme

theme = Theme()

class PopupMenu(tkinter.Menu):
    def __init__(self, master, bg=theme.app_color['background'], *args, **kwargs):
        tkinter.Menu._init(self, master, bg=bg, tearoff=0, *args, **kwargs)

        self.sub_menus = dict()

    def add_item(self, name="", command=None, args=tuple()):
        if command and args:
            self.add_command(label=name, command=lambda: command(args))
        else:
            self.add_command(label=name, command=command)

    def add_sub_menu(self, label="", name="", command=None, args=tuple()):
        sub_menu = PopupMenu(self)
        if label in self.sub_menus:
            pass
        else:
            self.sub_menus[label] = sub_menu
            self.add_cascade(label=label, menu=sub_menu)
            if command and args:
                sub_menu.add_command(label=name, command=lambda: command(args))
```

```
    else:
        sub_menu.add_command(label=name, command=command)

def add_sub_menu_item(self, label="", name="", command=None, args=tuple()):
    sub_menu = self.sub_menus.get(label)
    if sub_menu:
        if command and args:
            sub_menu.add_command(label=name, command=lambda: command(args))
        else:
            sub_menu.add_command(label=name, command=command)
    else:
        raise ValueError(f'Sub menu "{label}" was not found!')

def trigger(self, widget):
    widget.bind('<Button-3>', self.show)

def show(self, event):
    self.post(event.x_root, event.y_root)

if __name__ == '__main__':
    pass
```

### Screen.py

```
import tkinter
from tkinter_uix import Theme

theme = Theme()

class Screen(tkinter.Frame):
    def __init__(self, master, bg=theme.app_color['background'], *args, **kwargs):
        tkinter.Frame.__init__(self, master, bg=bg, *args, **kwargs)

    def show(self):
        self.pack(fill=tkinter.BOTH, expand=tkinter.YES)

    def hide(self):
        self.pack_forget()

class ScreenManager:
    def __init__(self):
        self.active_screen = ""
        self.screens = dict()

    def add_screen(self, master, name, *args, **kwargs):
        screen_widget = Screen(master, *args, **kwargs)
        if name in self.screens:
            pass
        else:
```

```
        self.screens[name] = {'screen': screen_widget, 'state': 'hide'}
    return screen_widget

def switch_screen(self, name):
    if name in self.screens and name != self.active_screen:
        if self.active_screen:
            self.screens[self.active_screen]['screen'].hide()
            self.screens[self.active_screen]['state'] = 'hide'

            self.screens[name]['screen'].show()
            self.screens[name]['state'] = 'show'
            self.active_screen = name
        else:
            self.screens[name]['screen'].show()
            self.screens[name]['state'] = 'show'
            self.active_screen = name

if __name__ == '__main__':
    pass
```

## 4.2 Back-end Code

Backend is server side of the website. It stores and arranges data, and also makes sure everything on the client-side of the website works fine. It is the part of the website that you cannot see and interact with.

It is the portion of software that does not come in direct contact with the users. The parts and characteristics developed by backend designers are indirectly accessed by users through a front-end application.

Activities, like writing APIs, creating libraries, and working with system components without user interfaces or even systems of scientific programming, are also included in the backend.

### 4.2.1 Main.py

```
from tkinter import *
from tkinter import messagebox
from modules.login import *

root = Tk()
root.geometry("1050x700")
root.title("Hire ME")
root.resizable(0, 0)
root.iconbitmap(r'elements\\favicon.ico')
log(root)
root.mainloop()
```

### 4.2.2 Db\_init.py

```
import mysql.connector as sql
from modules.creds import user_pwd
from cv_schema_init import *

mycon = sql.connect(host='localhost', user='root', passwd=user_pwd)
cur = mycon.cursor()
cur.execute(CREATE_SCHEMA)
cur.execute(Create_users_Table)
cur.execute(Create_recruiter_Table)
cur.execute(Create_client_Table)
cur.execute(Create_Job_Table)
cur.execute(Create_Application_Table)
mycon.close()
```

### 4.2.3 cv\_schema\_init.py

```
CREATE_SCHEMA = """
CREATE SCHEMA IF NOT EXISTS `mydb`;
"""

Create_users_Table = """
CREATE TABLE IF NOT EXISTS mydb.Users(
    name VARCHAR(45) NOT NULL,
    email Varchar(120) Not NULL,
    type VARCHAR(45) NOT NULL,
    password VARCHAR(45) NULL,
    UNIQUE INDEX email_UNIQUE (email),
    CHECK (type in ('Recruiter','Client')),
    PRIMARY KEY (email) );
"""
```



```
Create_recruiter_Table = """"
CREATE TABLE IF NOT EXISTS mydb.Recruiter(
  RID INT NOT NULL AUTO_INCREMENT,
  RName VARCHAR(45) NOT NULL,
  REmail VARCHAR(45) NOT NULL,
  CompanyName VARCHAR(45) NOT NULL,
  CompanyLocation VARCHAR(45) NOT NULL,
  RGender VARCHAR(2) NOT NULL,
  PRIMARY KEY (RID),
  UNIQUE (REmail)
);
""""
```

```
Create_client_Table = """"
CREATE TABLE IF NOT EXISTS mydb.Client (
  CID INT NOT NULL AUTO_INCREMENT,
  CName VARCHAR(45) NOT NULL,
  CEmail VARCHAR(45) NOT NULL,
  CAge INT NOT NULL,
  CLocation VARCHAR(45) NOT NULL,
  CGender VARCHAR(2) NOT NULL,
  CExp INT NOT NULL,
  CSkills VARCHAR(45) NOT NULL,
  CQualification VARCHAR(45) NOT NULL,
  UNIQUE (CEmail),
  PRIMARY KEY (CID)
);
""""
```

```
Create_Job_Table = """"
CREATE TABLE IF NOT EXISTS mydb.Job (
  RID INT NOT NULL,
  JID INT NOT NULL AUTO_INCREMENT,
  JobRole VARCHAR(45) NOT NULL,
  JobType VARCHAR(45) NOT NULL,
  Qualification VARCHAR(45) NOT NULL,
  MinExp INT NOT NULL,
  Salary INT NOT NULL,
  FOREIGN KEY (RID) REFERENCES mydb.Recruiter(RID),
  PRIMARY KEY (JID)
);
""""
```

```
Create_Application_Table=""""
CREATE TABLE IF NOT EXISTS mydb.Application(
  AID INT NOT NULL AUTO_INCREMENT,
  RID INT NOT NULL,
  JID INT NOT NULL,
  CID INT NOT NULL,
```

```
PRIMARY KEY(AID),  
FOREIGN KEY(RID) REFERENCES mydb.Recruiter(RID),  
FOREIGN KEY(JID) REFERENCES mydb.Job(JID),  
FOREIGN KEY(CID) REFERENCES mydb.Client(CID)  
);  
""""
```

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics. Its high-level built-in data structures, combined with dynamic typing and dynamic binding, make it very attractive for Rapid Application Development, as well as for use as a scripting or glue language to connect existing components together.

### 4.2.1 Database – MySQL

MySQL is a fast, easy-to-use RDBMS being used for many small and big businesses. It is developed, marketed and supported by MySQL AB, which is a Swedish company. It is released under an open-source license. So, you have nothing to pay to use it.

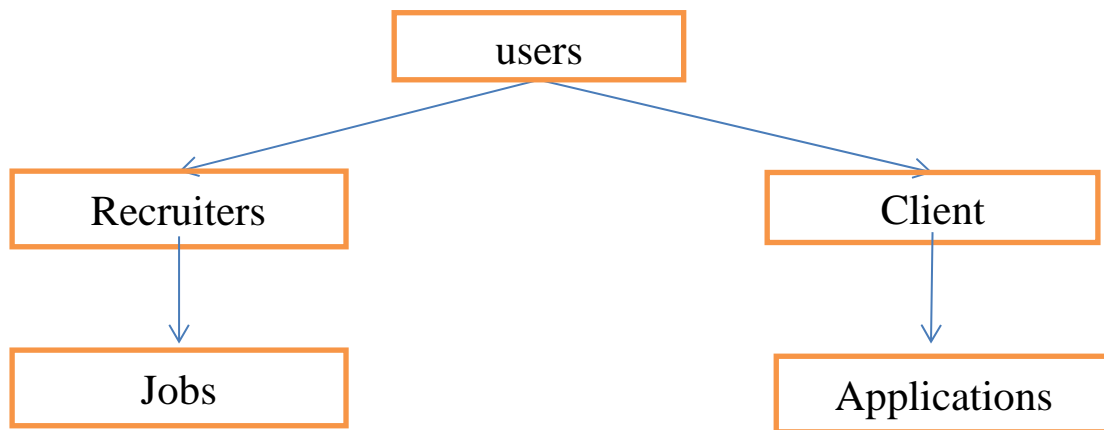
It is a very powerful program in its own right. It handles a large subset of the functionality of the most expensive and powerful database packages. It uses a standard form of the well-known SQL data language.

It works on many operating systems and with many languages including PHP, PERL, C, C++, JAVA, etc. It works very quickly and works well even with large data sets. It is very friendly to PHP, the most appreciated language for web development.

MySQL supports large databases, up to 50 million rows or more in a table. The default file size limit for a table is 4GB, but you can increase this (if your operating system can handle it) to a theoretical limit of 8 million terabytes (TB). It is customizable.

The open-source GPL license allows programmers to modify the MySQL software to fit their own specific environments.

### 4.3 User Flow Diagram



*Figure 4.1 User flow diagram*

The above diagram shows the workflow of our project. We have two options to login one is as a recruiter and another one is as a client. The recruiter issues jobs and clients create applications for jobs.

## 4.4 Discussion of Code Segment

This section talks about the important code sections and modules that are implemented in the Recruitment Management System project.

These modules add logic to the complete system, and make it function the way it is supposed to. It also talks about the integration between the front-end Python code and the back-end MySQL database.

### 4.4.1 Login Segment



```
def submit(root):
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute('select email,password from users')
    total = cur.fetchall()
    mycon.close()
    email1 = email.get()
    password = pwd.get()
    if email1 and password:
        for i in total:
            if email1 == i[0] and password == i[1]:
                return success(root, email1)
            elif email1 == i[0] and password != i[1]:
                messagebox.showinfo('Alert!', 'Invalid Credentials')
                break
        else:
            messagebox.showinfo(
                'Alert!', 'Email is not registered, Please register')
    else:
        messagebox.showinfo(
            'Alert!', 'Please Enter both Email and Password')
```

*Figure 4.2 Login Segment*

### 1.4.2 Register Segment

```
def recruiter_check(root):
    global name1, email1, pwd1, cpwd1
    name1 = name.get()
    email1 = email.get()
    pwd1 = pwd.get()
    cpwd1 = cpwd.get()
    print(name1, email1, pwd1, cpwd1)
    if name1 and email1 and pwd1 and cpwd1:
        mycon = sql.connect(host='localhost', user='root',
                             passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute('select email from users')
        total = cur.fetchall()
        mycon.close()
        exist_email = []
        for i in total:
            exist_email.append(i[0])
        print("existing users:", exist_email)

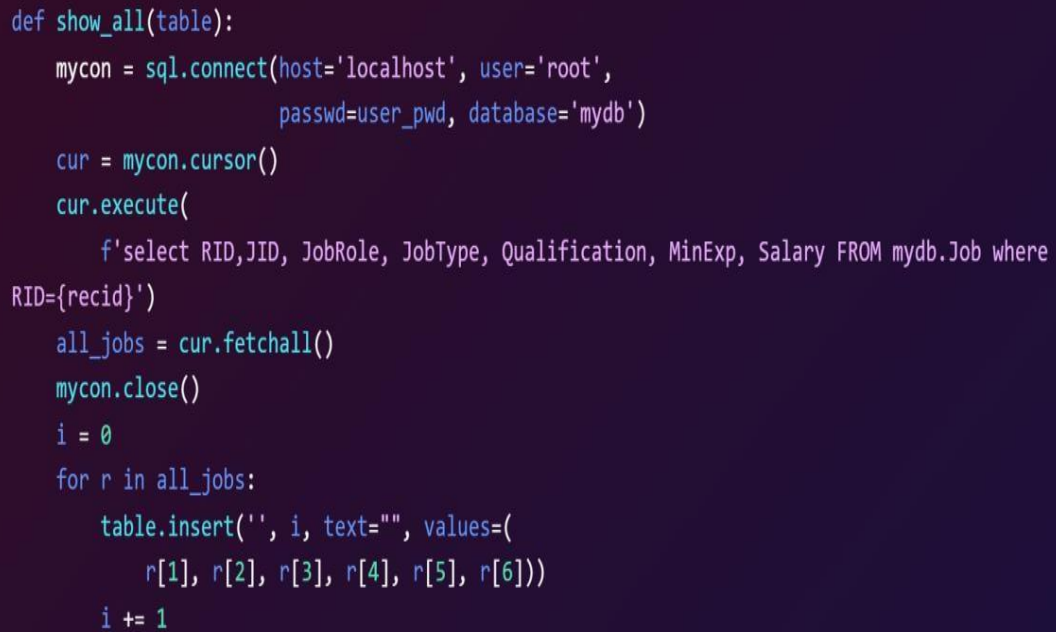
        if email1 in exist_email:
            messagebox.showinfo('ALERT!', 'EMAIL ALREADY REGISTERED')
            email.delete(0, END)

        else:
            if pwd1 == cpwd1:
                recruit_complete(root)
            else:
                messagebox.showinfo('ALERT!', 'PASSWORDS DO NOT MATCH')

    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS ARE MUST BE FILLED')
```

Figure 4.3 Register Segment

#### 4.4.3 Posted Job Query



```
def show_all(table):  
    mycon = sql.connect(host='localhost', user='root',  
                        passwd=user_pwd, database='mydb')  
    cur = mycon.cursor()  
    cur.execute(  
        f'select RID,JID, JobRole, JobType, Qualification, MinExp, Salary FROM mydb.Job where  
RID={recid}')  
    all_jobs = cur.fetchall()  
    mycon.close()  
    i = 0  
    for r in all_jobs:  
        table.insert('', i, text="", values=(  
            r[1], r[2], r[3], r[4], r[5], r[6]))  
        i += 1
```

*Figure 4.4 Posted Job Query*

#### 4.4.4 Submit Segment

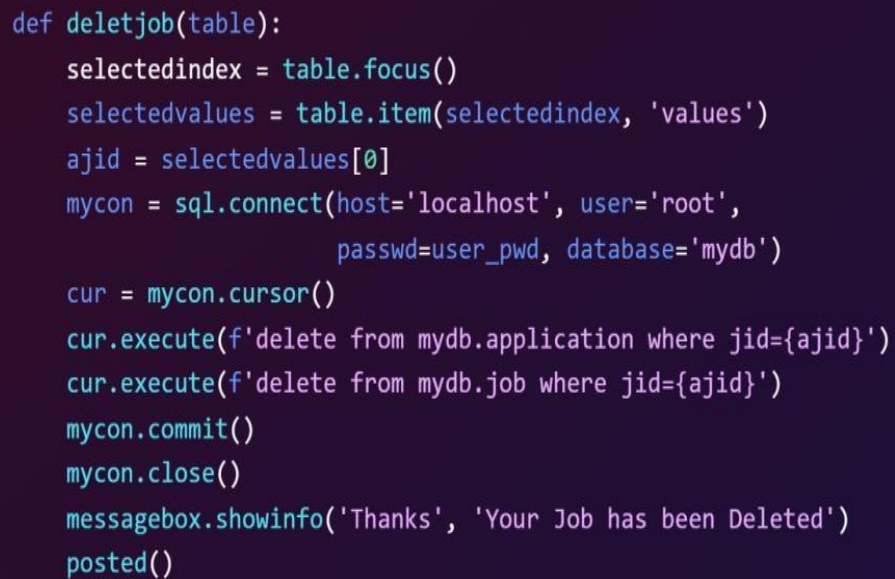
```
def recruiter_submit(root):
    global gender1, company1, loc1
    gender1 = gender.get()
    company1 = company.get()
    loc1 = loc.get()
    print(name1, email1, gender1, company1, loc1)
    if gender1 and company1 and loc1:
        exe = f'insert into users values("{name1}", "{email1}", "recruiter", "{pwd1}")'
        exe1 = f'INSERT INTO mydb.Recruiter(RID, RName, REmail, CompanyName, CompanyLocation
        ,RGender) VALUES (NULL, "{name1}", "{email1}", "{company1}", "{loc1}", "{gender1}")'

        try:
            mycon = sql.connect(host='localhost', user='root',
                                passwd=user_pwd, database='mydb')
            cur = mycon.cursor()
            cur.execute(exe)
            cur.execute(exe1)
            name.delete(0, END)
            email.delete(0, END)
            pwd.delete(0, END)
            cpwd.delete(0, END)
            # gender.delete(0, END)
            loc.delete(0, END)
            company.delete(0, END)
            mycon.commit()
            mycon.close()
            messagebox.showinfo('SUCCESS!', 'Registration Successful')
            logi(root)
        except:
            pass

    else:
        messagebox.showinfo('ALERT!', 'ALL FIELDS MUST BE FILLED')
```

*Figure 4.5 Submit Segment*

#### 4.4.5 Delete Posted Job



```
def deletjob(table):  
    selectedindex = table.focus()  
    selectedvalues = table.item(selectedindex, 'values')  
    ajid = selectedvalues[0]  
    mycon = sql.connect(host='localhost', user='root',  
                        passwd=user_pwd, database='mydb')  
    cur = mycon.cursor()  
    cur.execute(f'delete from mydb.application where jid={ajid}')  
    cur.execute(f'delete from mydb.job where jid={ajid}')  
    mycon.commit()  
    mycon.close()  
    messagebox.showinfo('Thanks', 'Your Job has been Deleted')  
    posted()
```

*Figure 4.6 Delete Posted Job*



#### 4.4.6 Sort Queries

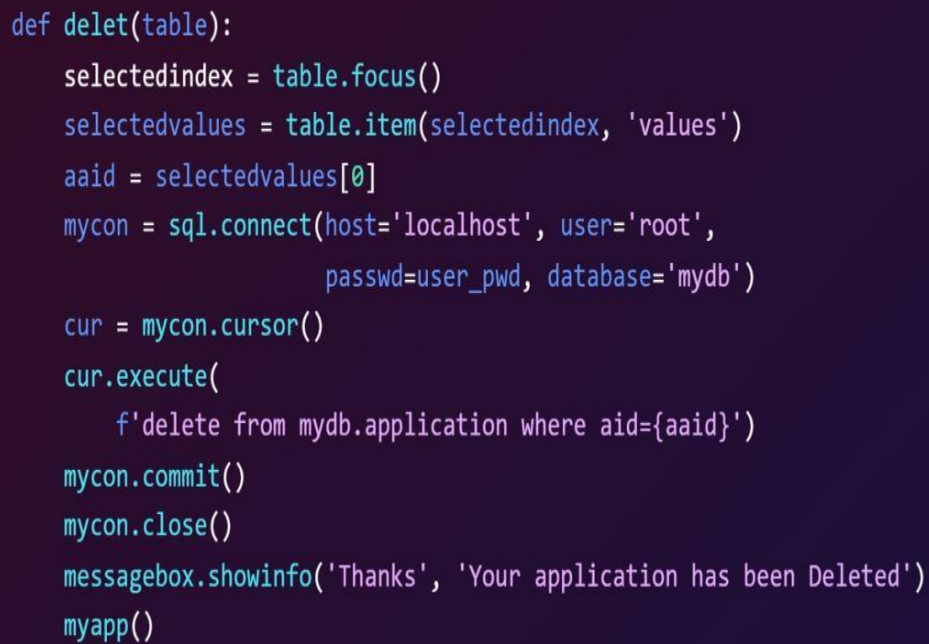
```
def sort_all(table):
    criteria = search_d.get()
    if(criteria == "Select"):
        pass
    else:
        table.delete(*table.get_children())
        mycon = sql.connect(host='localhost', user='root',
                            passwd=user_pwd, database='mydb')

        cur = mycon.cursor()
        cur.execute(
            f'select RID,JID, JobRole, JobType, Qualification, MinExp, Salary FROM mydb.Job where
            RID={recid} order by {criteria}')
        all_jobs = cur.fetchall()
        mycon.close()

        i = 0
        for r in all_jobs:
            table.insert('', i, text="", values=(
                r[1], r[2], r[3], r[4], r[5], r[6]))
            i += 1
```

Figure 4.7 Submit Segment

#### 4.4.7 Delete Job Application Segment



```
def delet(table):
    selectedindex = table.focus()
    selectedvalues = table.item(selectedindex, 'values')
    aaid = selectedvalues[0]
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(
        f'delete from mydb.application where aid={aaid}')
    mycon.commit()
    mycon.close()
    messagebox.showinfo('Thanks', 'Your application has been Deleted')
    myapp()
```

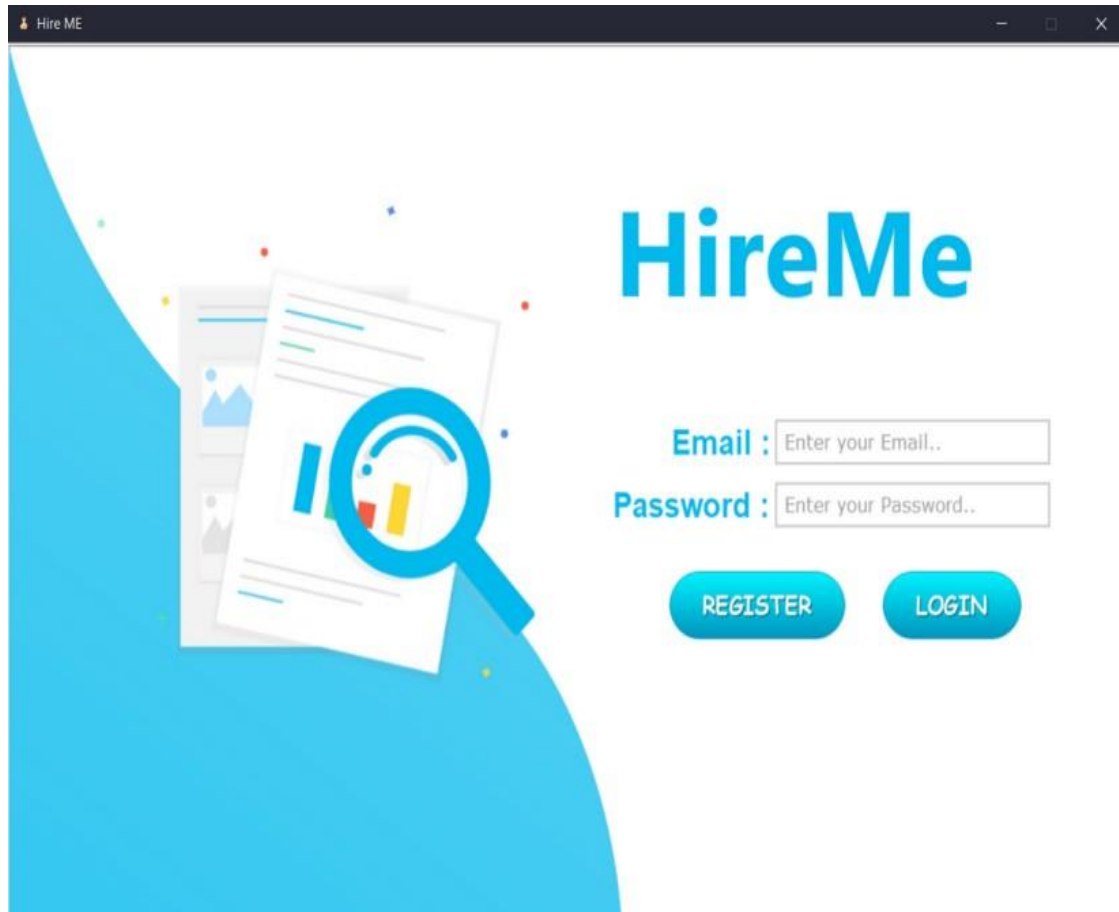
*Figure 4.8 Delete job Segment*

#### 4.4.8 Job Application Segment

```
def apply(table):
    # fetch cid,jid from treeview that is in available jobs function
    # code
    selectedindex = table.focus()    # that will return number index
    # that will return list of values with columns=['JID', 'JobRole', 'JobType', 'CompanyName',
    'CompanyLocation', 'Qualification', 'MinExp', 'Salary']
    selectedvalues = table.item(selectedindex, 'values')
    ajid = selectedvalues[0]
    chkquery = f'SELECT * from mydb.application where cid={clicid} and jid={ajid}'
    mycon = sql.connect(host='localhost', user='root',
                        passwd=user_pwd, database='mydb')
    cur = mycon.cursor()
    cur.execute(chkquery)
    tempbuff = cur.fetchall()
    mycon.close()
    if(tempbuff):
        messagebox.showinfo(
            'Oops', 'It seems like you have already applied to this job')
    else:
        queryapplyjob = f'Insert into application values(NULL,(select rid from mydb.job where
        job.jid={ajid}},{ajid},{clicid})'
        mycon = sql.connect(host='localhost', user='root',
                            passwd=user_pwd, database='mydb')
        cur = mycon.cursor()
        cur.execute(queryapplyjob)
        mycon.commit()
        mycon.close()
        messagebox.showinfo('Thanks', 'Your application has been submitted')
```

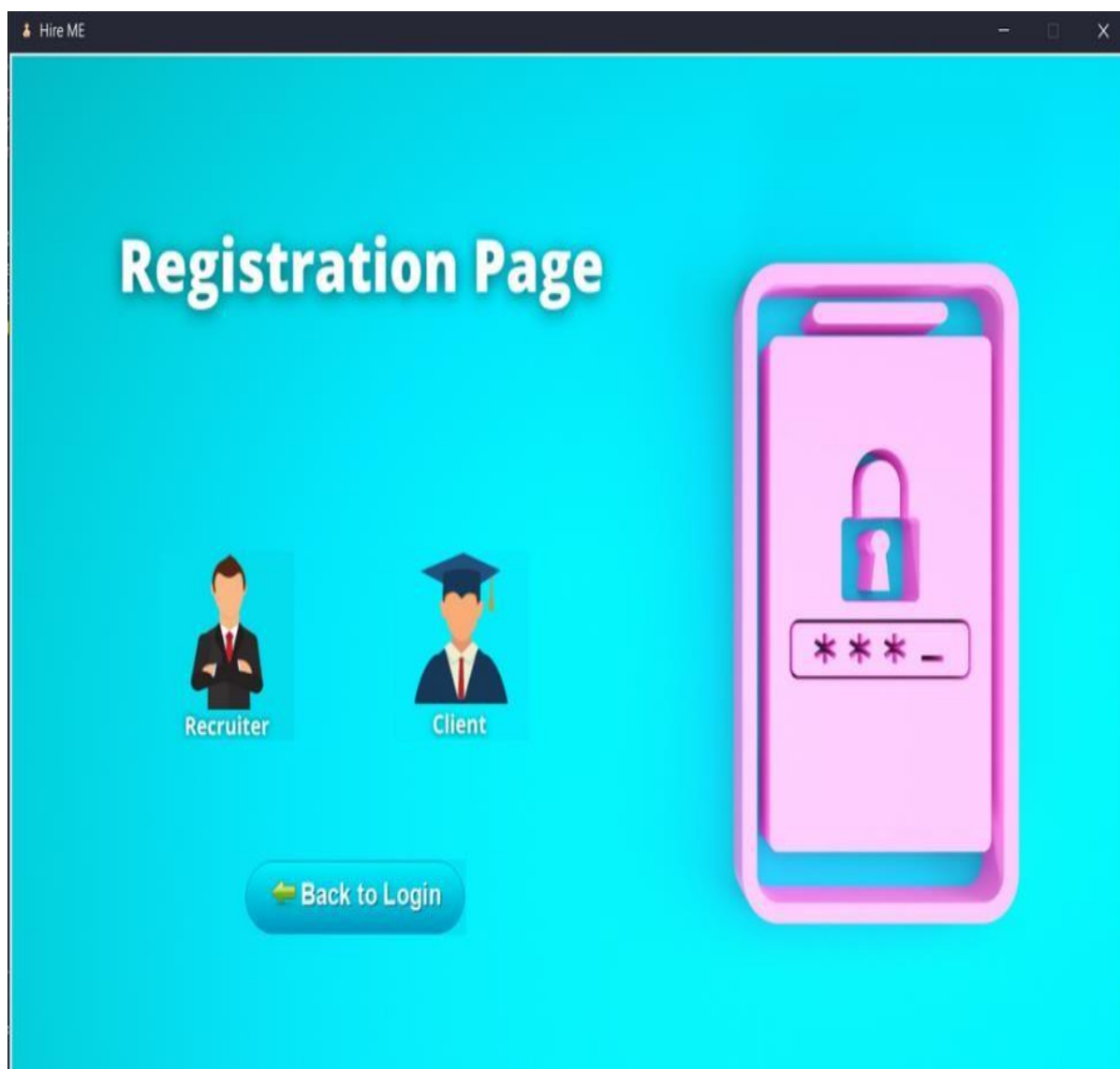
Figure 4.9 Job Application Segment

## 4.5 Discussion of Results



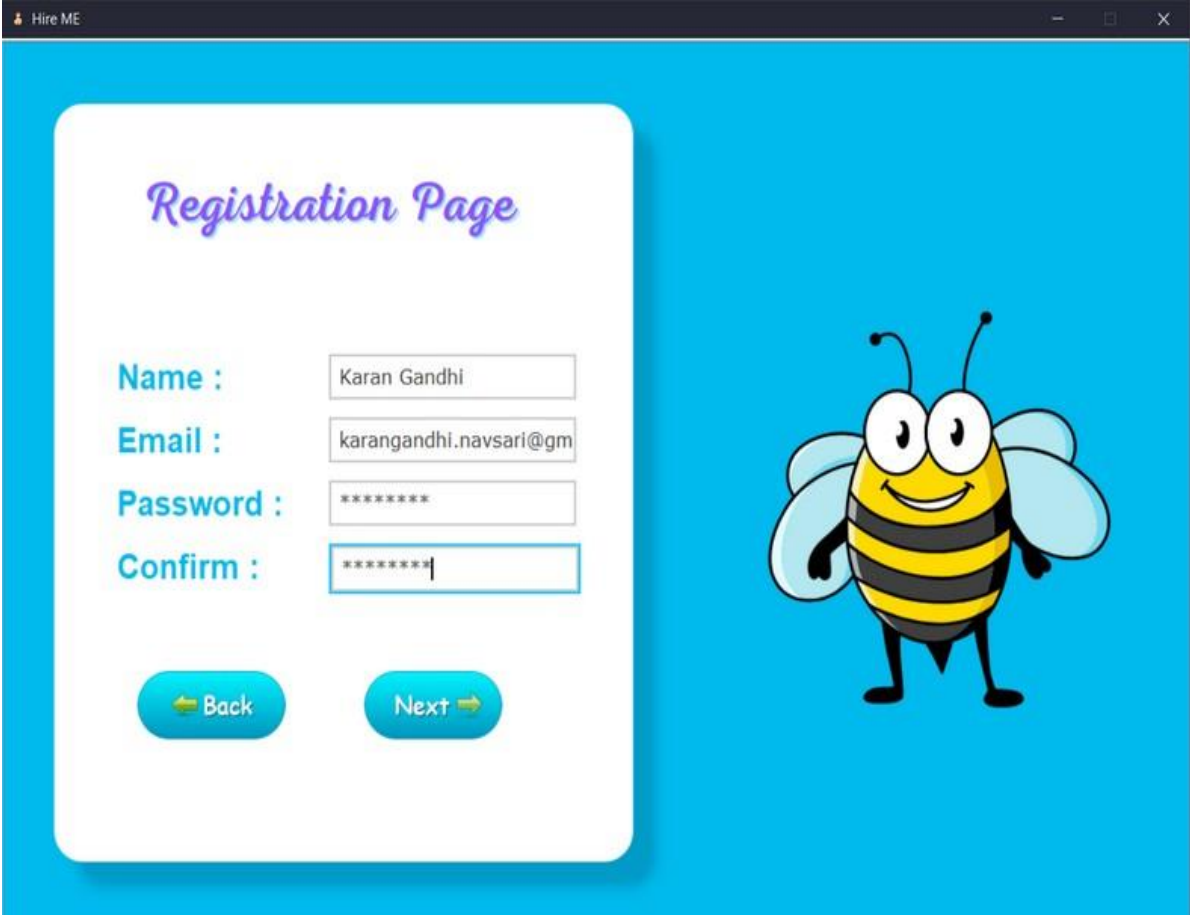
*Figure 4.9 Homepage with login prompt*

The above picture describes about the main screen of our project. Here, the user register or login through the Email id and password, if the typed password and email is correct the user will be taken to registration page.



*Figure 4.10 Registration Prompt*

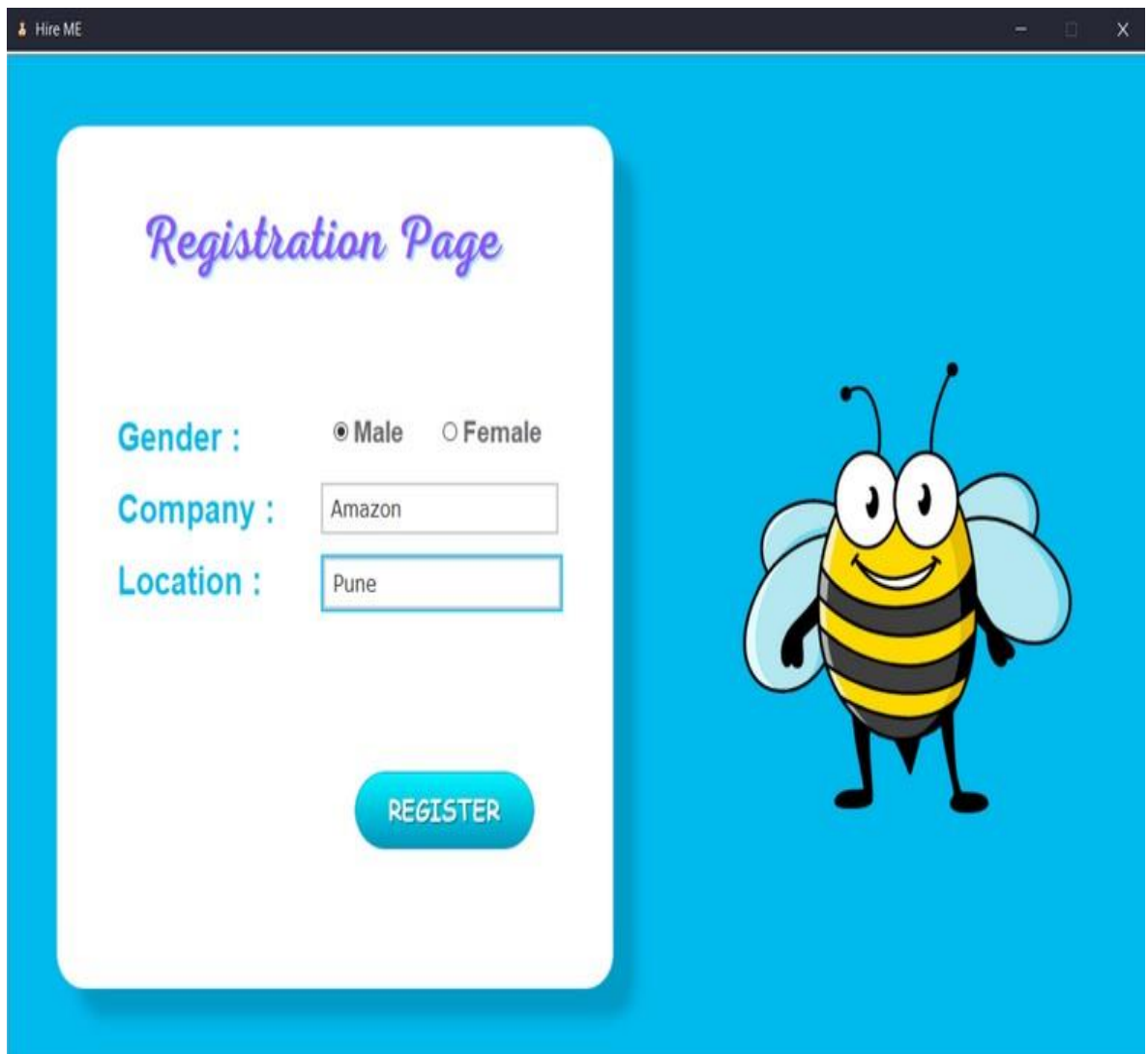
This is the registration page where a user can choose their role whether the user is a recruiter or a client.



The screenshot shows a web browser window with the title "Hire ME". The main content area has a bright blue background. On the left, there is a white rounded rectangle containing the registration form. The form is titled "Registration Page" in a purple, cursive font. It includes four input fields: "Name" (containing "Karan Gandhi"), "Email" (containing "karangandhi.navsari@gm"), "Password" (containing "\*\*\*\*\*"), and "Confirm" (containing "\*\*\*\*\*"). Below the fields are two buttons: "Back" with a left arrow and "Next" with a right arrow. On the right side of the blue background, there is a cartoon bee character with a yellow and black striped body, large eyes, and blue wings.

*Figure 4.11 Registration Page*

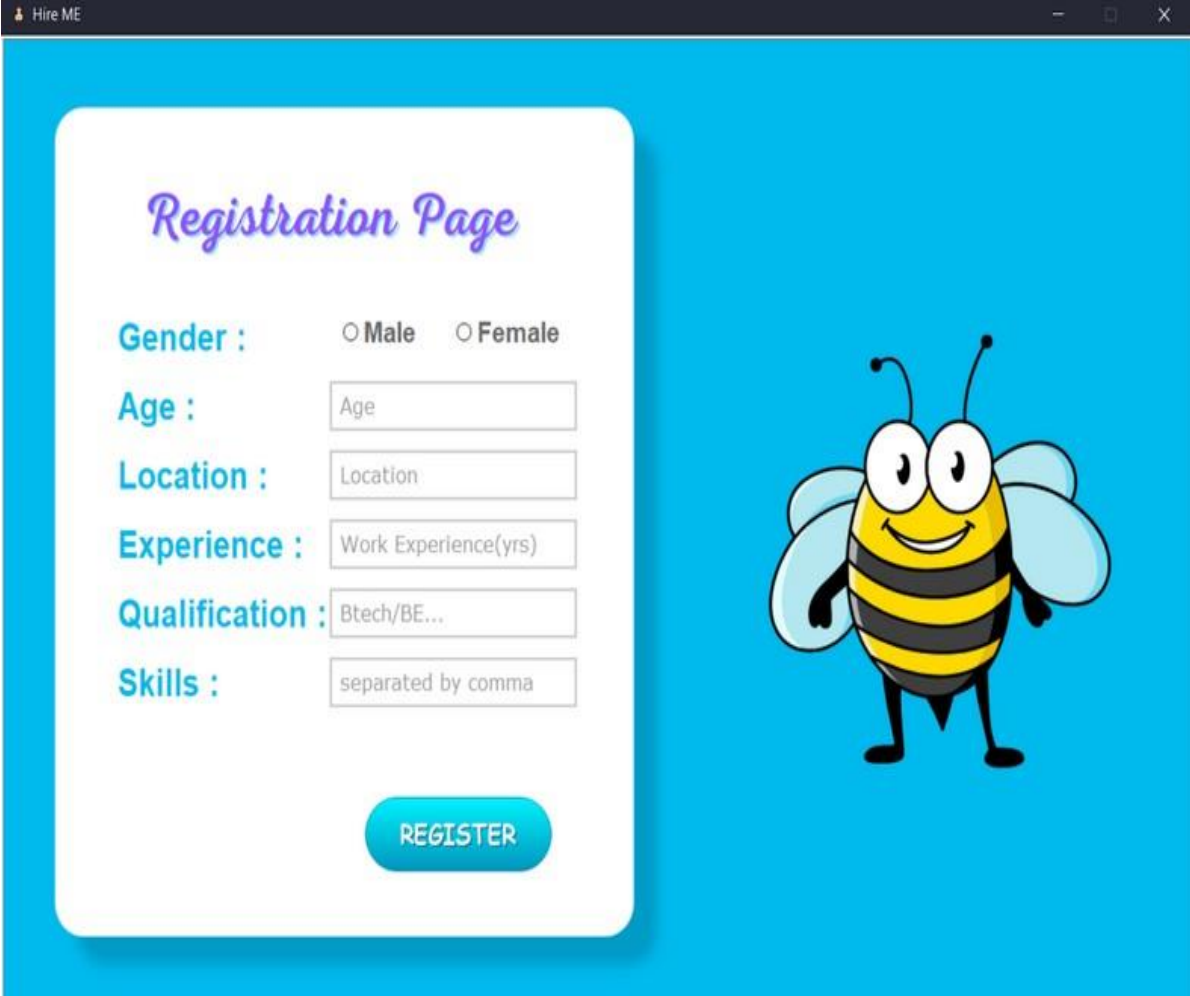
When we click on client or recruiter the first page will be always same where it asks for name, email, password.



The screenshot shows a web browser window titled "Hire ME". The main content area has a bright blue background. On the left, there is a white rounded rectangle containing the "Registration Page" title in purple script. Below the title are three form fields: "Gender :" with radio buttons for "Male" (selected) and "Female"; "Company :" with a text box containing "Amazon"; and "Location :" with a text box containing "Pune". A blue "REGISTER" button is at the bottom of the form. To the right of the form is a cartoon bee character with a yellow and black striped body, blue wings, and a friendly expression.

*Figure 4.12 Registering as recruiter*

When a user register as a recruiter, the user will be forwarded to this page which asks for the details of the company.

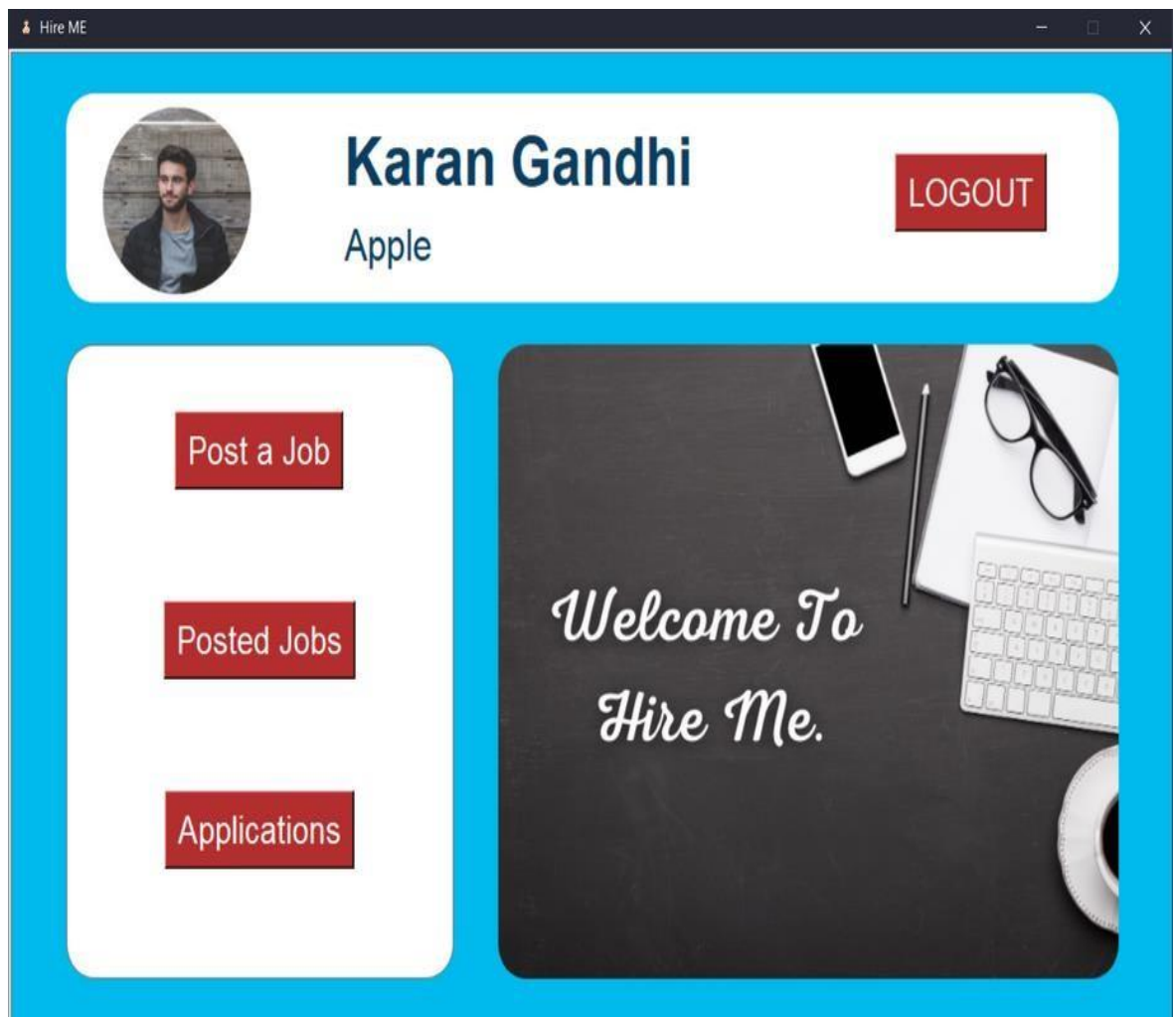


The screenshot shows a web browser window with the title 'Hire ME'. The main content area has a bright blue background. On the left, there is a white rounded rectangle containing the registration form. The form is titled 'Registration Page' in a purple, cursive font. Below the title, there are several fields: 'Gender :' with radio buttons for 'Male' and 'Female'; 'Age :' with a text input field containing the placeholder 'Age'; 'Location :' with a text input field containing the placeholder 'Location'; 'Experience :' with a text input field containing the placeholder 'Work Experience(yrs)'; 'Qualification :' with a text input field containing the placeholder 'Btech/BE...'; and 'Skills :' with a text input field containing the placeholder 'separated by comma'. At the bottom of the form is a blue button with the text 'REGISTER' in white. To the right of the form, there is a cartoon bee character with a yellow and black striped body, large eyes, and a friendly smile.

*Figure 4.13 Registering as client*

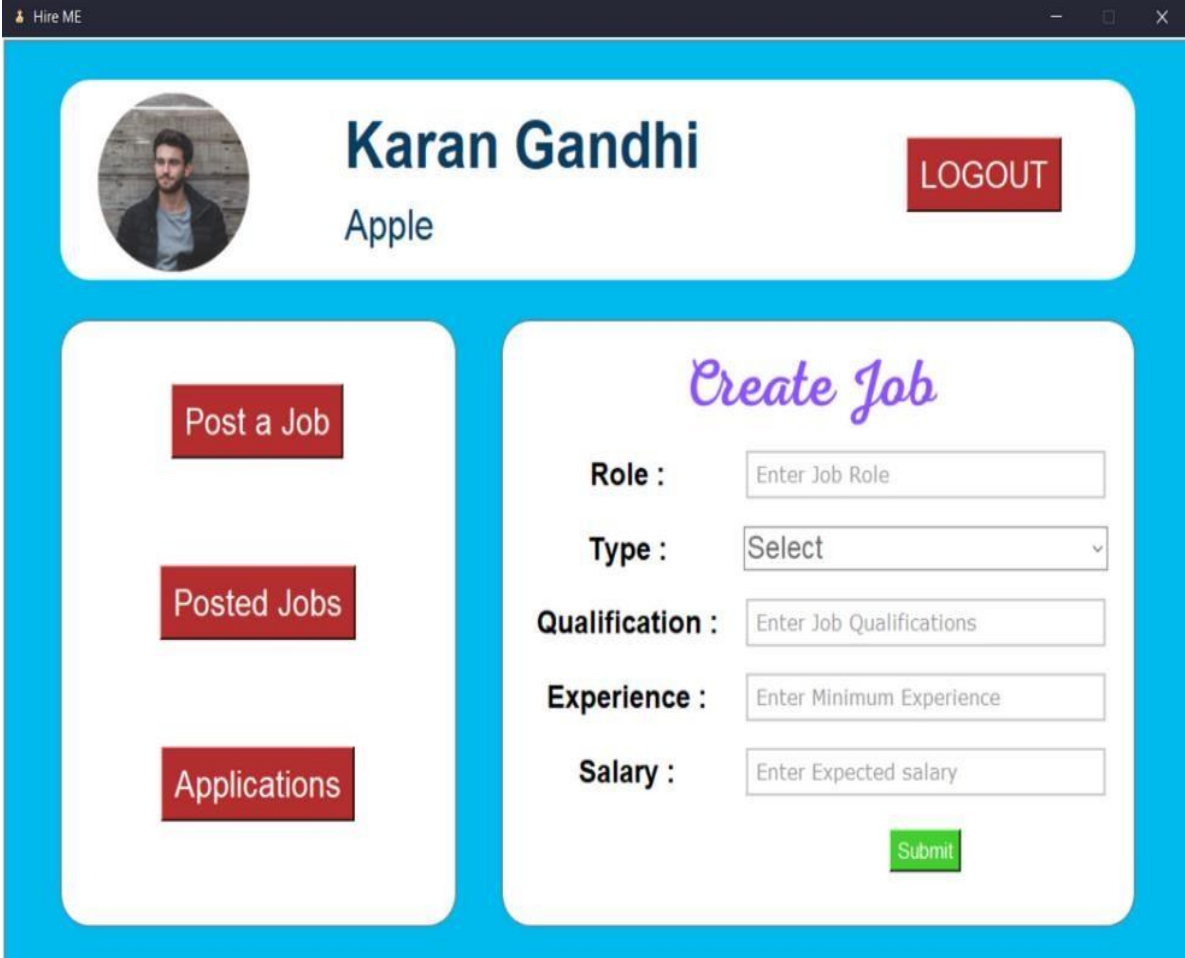
When a user register as a client then this page will open which asks the details of the user and the use's qualification and skills.





*Figure 4.14 Options for Recruiter*

After registration a recruiter will get this page where he will have multiple options to post a job or look into posted jobs or go through the applications of a clients.



The screenshot shows a web application window titled "Hire ME". The user is logged in as "Karan Gandhi" from "Apple", with a "LOGOUT" button. The main interface has a blue background. On the left, there is a sidebar with three red buttons: "Post a Job", "Posted Jobs", and "Applications". The "Post a Job" button is selected. The main content area is titled "Create Job" in purple cursive. It contains five form fields: "Role" (text input), "Type" (dropdown menu), "Qualification" (text input), "Experience" (text input), and "Salary" (text input). Each field has a placeholder text. A green "Submit" button is at the bottom right of the form.

Figure 4.15 Post a Job

If a recruiter selects “post a job” then he will get a form which asks about details of the new creating job.

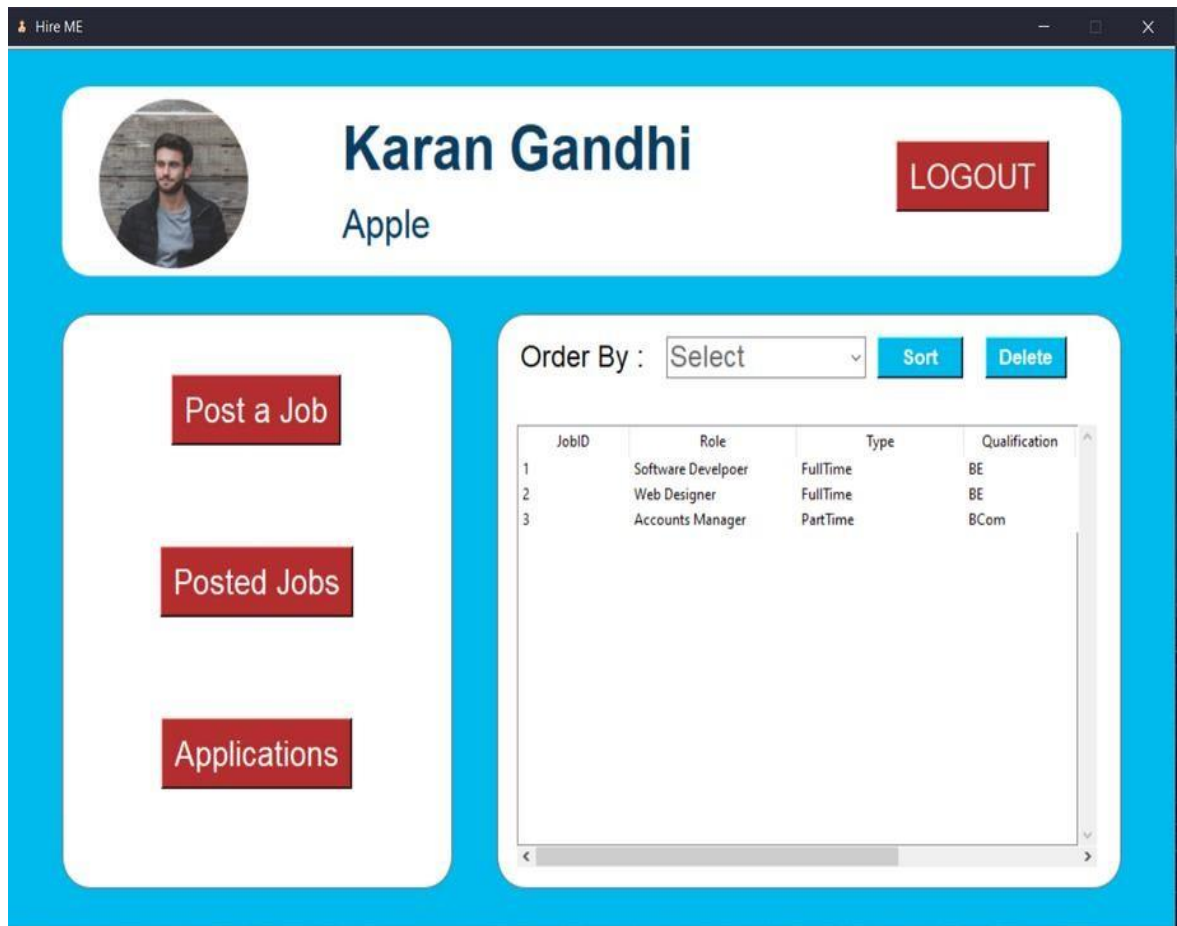


Figure 4.16 Posted Jobs

“Posted Jobs” page looks like the figure above which gives a list of all posted jobs. It also gives options to sort and delete a job.

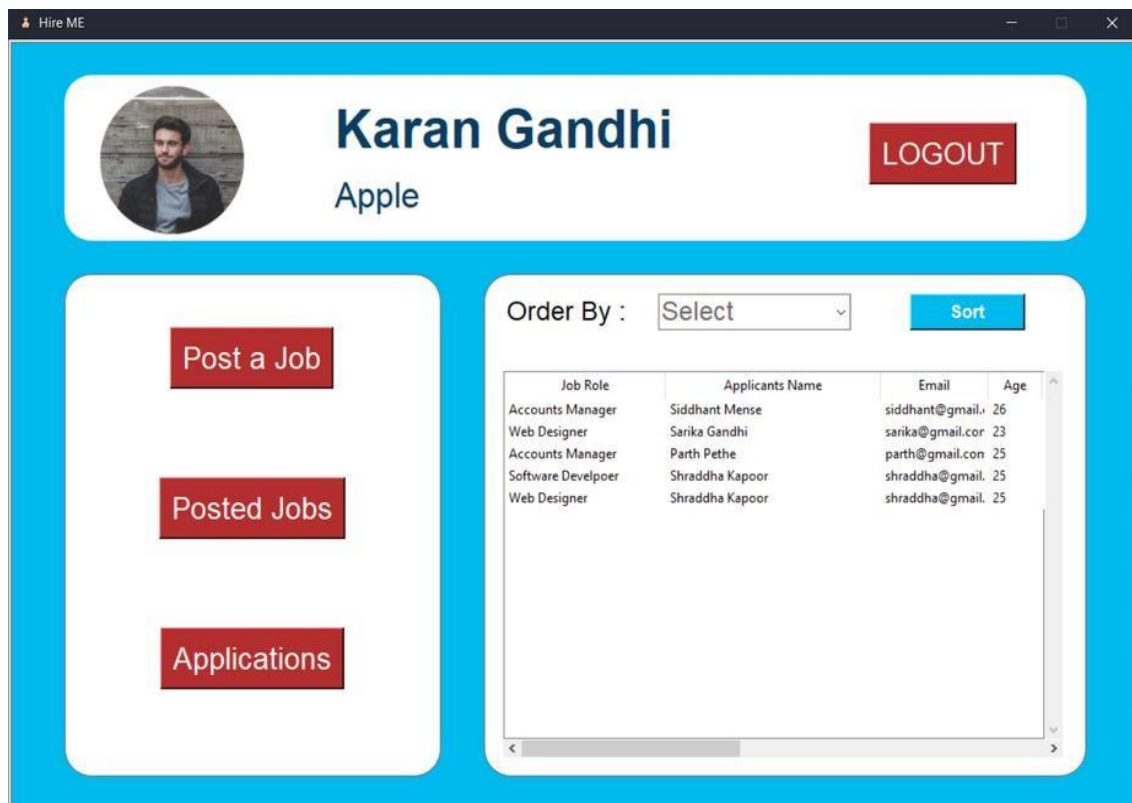
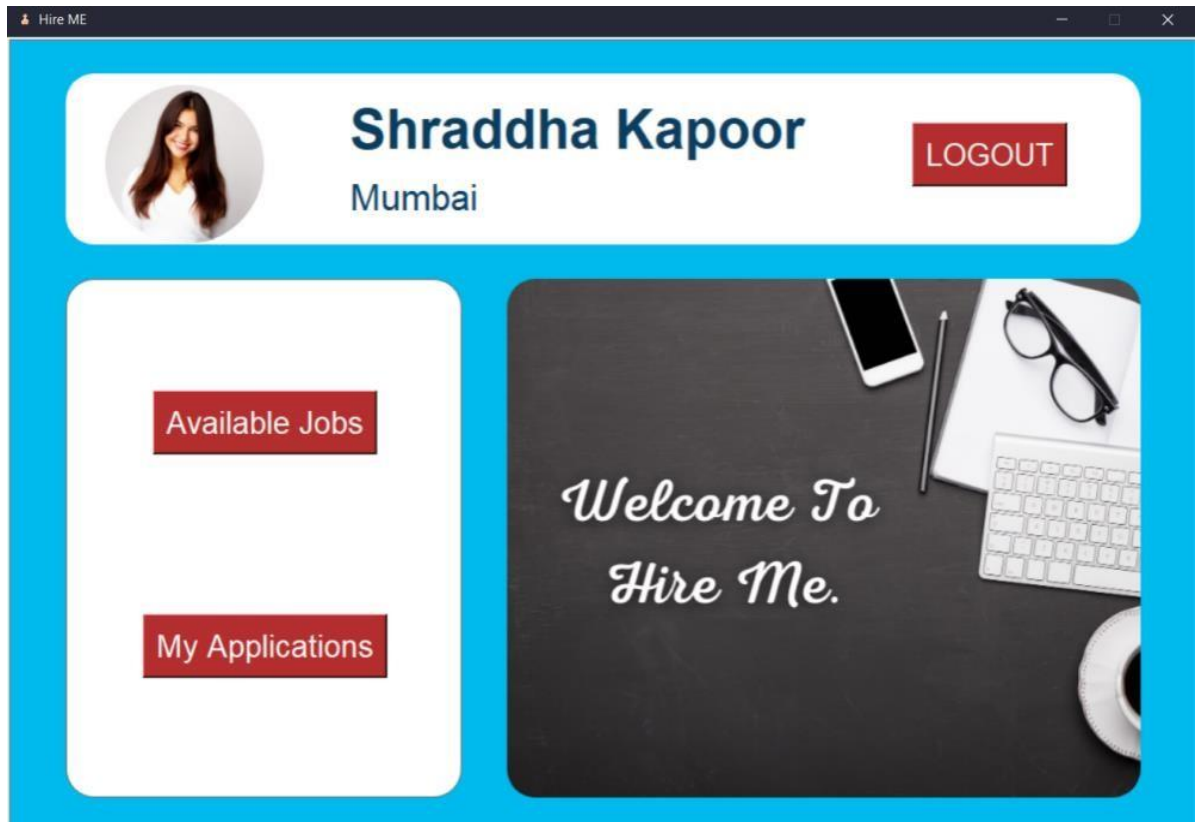


Figure 4.17 Applications

When a recruiter selects “Applications” it shows the list of applications applied by the clients to the recruiter as shown above.



*Figure 4.18 Client page*

When a user registers as a client, then the user will be taken to this page which shows two options, “available jobs” and “my applications”.

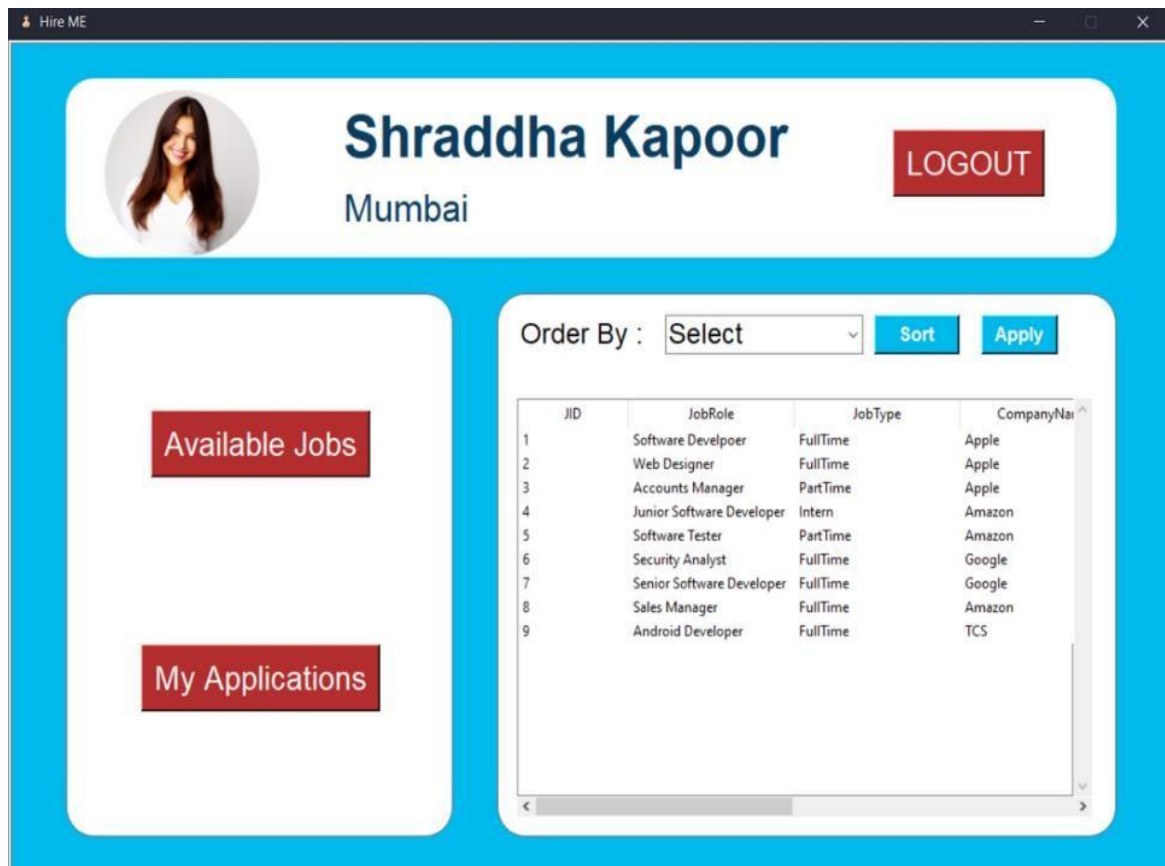


Figure 4.19 Available Jobs

When a client selects “available jobs”, then the user will get a page with list of available job and two options at the top to sort and apply for a job as shown.

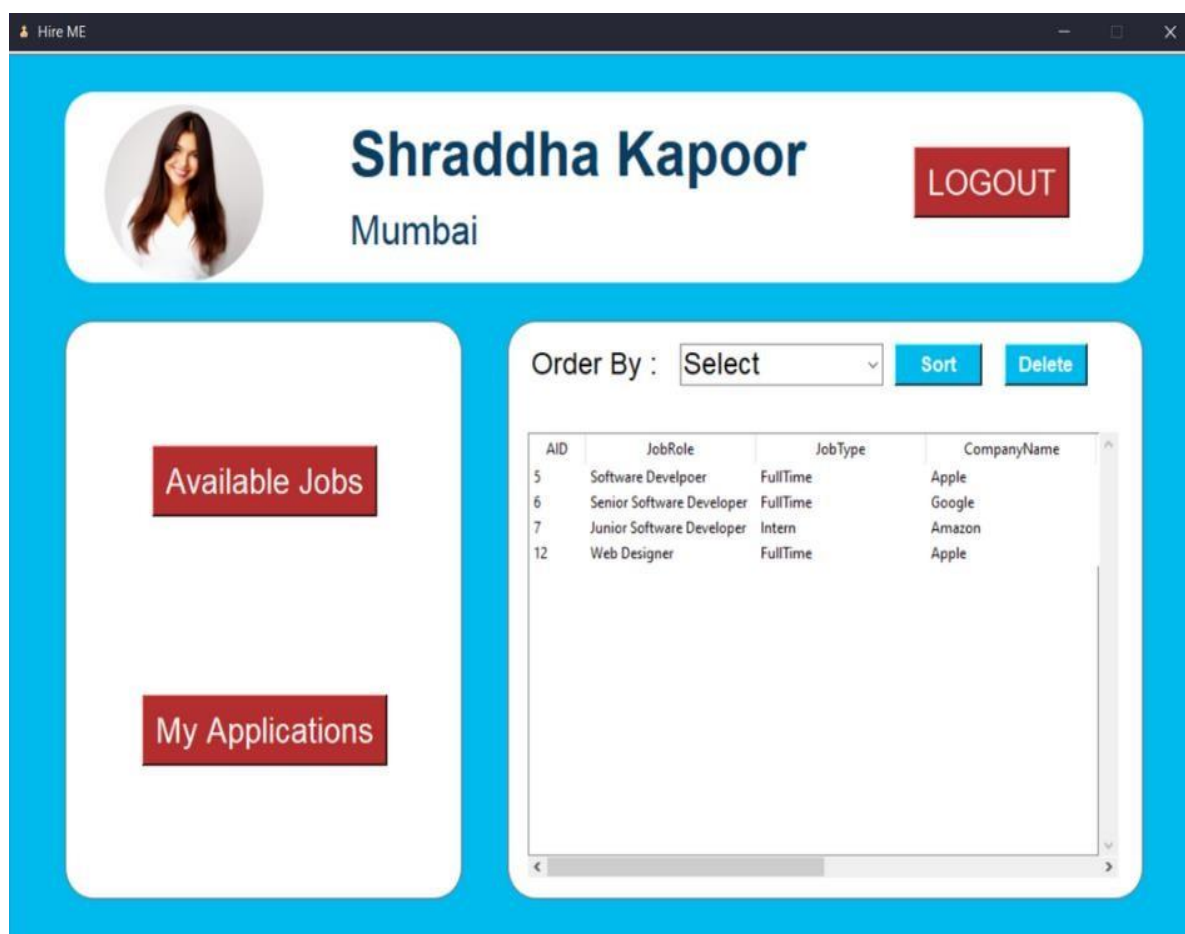


Figure 4.20: My Applications

When a client selects “My Applications”, then the user will get a list of his/her applications applied to different companies as shown. A client can delete his/her own application.

## 4.6 Application of project

These types of job hiring web applications are really useful in the present world as everyone are tending towards online system. A client can apply application to a specific company and check various available jobs available at present scenario under Available Jobs. A huge amount of time will be saved for both recruiters and clients in selecting a client and selecting a job.

A Social Enterprise - Bridging the gap between students and Corporates.

Hire-Me is a Platform providing freshers jobs- linking corporate, students and colleges and forms a link between recruiters and clients who are in search for job.

1. Recruiters from different companies can login with their own email ids and can recruit the clients by verifying the applications.
2. Clients can apply for different jobs according to their own choice and attend for the interviews conducted later.
3. Recruiter can delete its job created when recruitment of that particular job is done.



## Chapter 5

# CONCLUSION AND FUTURE ENHANCEMENT

### 5.1 Conclusion

We have successfully implemented a recruitment management system with backend with Python, tkinter as frontend and mysql for database.

### 5.2 Future Enhancements

- More Features can be added like we can make Resume on it and can share to nearby recruiters.
- It can be implemented on web platform and hosted on web so it can be accessed from any device.
- Super-User Panel For Future Features.
- It is very useful for the clients as well as Recruiter to connect easily and free to search jobs anytime.
- Simple UI and one click functionalities.

## Chapter 6

### REFERENCES

- [1] The GUI we used is referenced from:

*<https://www.geeksforgeeks.org/python-gui-tkinter/>*

- [2] The database connection to our project using python code is done by referring to:

*[https://www.w3schools.com/python/python\\_mysql\\_getstarted.asp](https://www.w3schools.com/python/python_mysql_getstarted.asp)*

*<https://www.javatpoint.com/python-mysql-database-connection>*

- [3] We referred to many web pages like:

*<https://projectsgeek.com/dbms-project-ideas-top-free>*

*<https://instanteduhelp.com/unique-database-project-ideas/>*