NETAJI SUBHAS UNIVERSITY OF TECHNOLOGY

HIRE-ME: Recruitment Management System

SEMESTER-3(BATCH 2024-2028)
BRANCH-MAC

Submitted to:

Mr. Anand Gupta

Presented By:

Raghav Gupta (2024UCM2301)

Tisha Bansal (2024UCM310)

INDEX

1. ABSTRACT

Recruitment is an essential operation in organizations, involving job postings, candidate applications, and applicant evaluation. Manual recruitment processes lead to inefficiencies such as misplaced resumes, poor tracking mechanisms, and inconsistent communication.

HIRE-ME is a desktop-based recruitment management system built using Python Tkinter for the graphical interface and MySQL for database management. The system allows recruiters to post and manage job listings and allows clients (job seekers) to search and apply for jobs. All operations follow CRUD principles and are backed by a normalized relational database.

The project demonstrates key DBMS concepts such as ER modeling, schema design, relational constraints, normalization, SQL queries, and database-driven UI integration.

---

2. INTRODUCTION

Recruitment plays a key role in matching job seekers with appropriate job opportunities. In most traditional settings, recruitment involves:

- Word-of-mouth information
- Offline forms
- Email-based applications
- Paper-based records

These outdated methods result in inefficiencies and data mismanagement.
With the advent of database-driven systems, recruitment processes can be made faster, scalable, and more reliable.
HIRE-ME aims to modernize this process via:

- A structured database
- A user-friendly GUI
- Clear workflows for posting and applying to jobs
- Accurate and consistent data storage

The project integrates DBMS concepts with practical GUI-based CRUD operations, making it ideal for academic DBMS coursework.

---

3. PROBLEM STATEMENT

The current recruitment process has several issues:

- Job postings are scattered across different platforms.
- Applicants struggle to track their applications.
- Recruiters have no unified system to manage applicant data.

- Manual records are prone to errors and data loss.
- Searching, filtering, and organizing candidates is difficult.

Thus, a centralized system is needed that simplifies the recruitment workflow using a clean interface and a well-designed backend database.

---

4. OBJECTIVES

Primary Objectives

- To develop a GUI-based recruitment system using Python Tkinter.
- To design and implement a relational database in MySQL.
- To establish CRUD operations for users, jobs, and applications.
- To enforce authentication with role-specific access.

Secondary Objectives

- To maintain data integrity using foreign key constraints.
- To provide a simplified dashboard for recruiters and clients.
- To ensure normalized, efficient data storage.
- To illustrate complete DBMS documentation for academic evaluation.

---

5. SCOPE OF THE PROJECT

In-Scope Features

- User registration and login

- Recruiter job posting & management
- Client job search & job application
- Application tracking
- Database-backed storage
- Tkinter-based UI

Out-of-Scope Features (Future Enhancements)

- Resume uploads
- Email notifications
- Multi-admin features
- AI-based recommendations
- Mobile/web versions

---

6. EXISTING SYSTEM

Existing/manual recruitment involves:

- Posting jobs on multiple websites
- Receiving applications via email
- Using Excel/PDFs to track applicants
- No centralized database
- Poor data accuracy
- Delayed communication

These inefficiencies lead to high time consumption and mismanagement.

---

7. PROPOSED SYSTEM

The proposed HIRE-ME system provides:

- Centralized storage of all recruitment data
- Systematic workflows for both recruiters and applicants
- Secure login
- Instant job posting
- One-click job application
- Accurate applicant tracking

Benefits:

- Reduced manual efforts
- Better record management
- Improved transparency
- Structured, scalable system

---

8. FEASIBILITY STUDY

Technical Feasibility

- Python and MySQL are efficient, accessible, and widely supported.
- Tkinter requires no external dependencies.

Operational Feasibility

- Simple UI ensures easy adoption.
- Minimal training required.

Economic Feasibility

- Entire system uses open-source software.
- Zero licensing cost.

---

9. SOFTWARE & HARDWARE REQUIREMENTS

Hardware

- CPU: Minimum i3 or equivalent
- RAM: 4GB (recommended 8GB)
- Storage: 500MB

Software

- Windows/Linux/MacOS
- Python 3.x
- Tkinter
- MySQL Server / Workbench
- Python Modules:
    - mysql-connector-python
    - Pillow (optional)

---

10. SYSTEM MODULES & OVERVIEW

Recruiter Module

- Register/Login
- Post job
- Update/delete job

- View applicants

Client Module

- Register/Login
- Search jobs
- Apply for job
- Track applications

Database Module

- Connections
- CRUD operations
- Joins

GUI Module

- Tkinter windows
- Forms
- Buttons, labels, entries

---

11. SYSTEM ANALYSIS

The system consists of:

- Users (Recruiter or Client)
- Jobs posted by recruiters
- Applications submitted by clients

The database ensures:

- Consistent storage

- Accurate relationships
- Referential integrity

---

12. FUNCTIONAL REQUIREMENTS

1. User registration
2. Secure login
3. Recruiter job CRUD
4. Client job search
5. Client job apply
6. Recruiter view applicants
7. Session-based access handling

---

13. NON-FUNCTIONAL REQUIREMENTS

Performance

- CRUD operations complete within seconds

Security

- Password hashing recommended
- SQL injection prevention

Usability

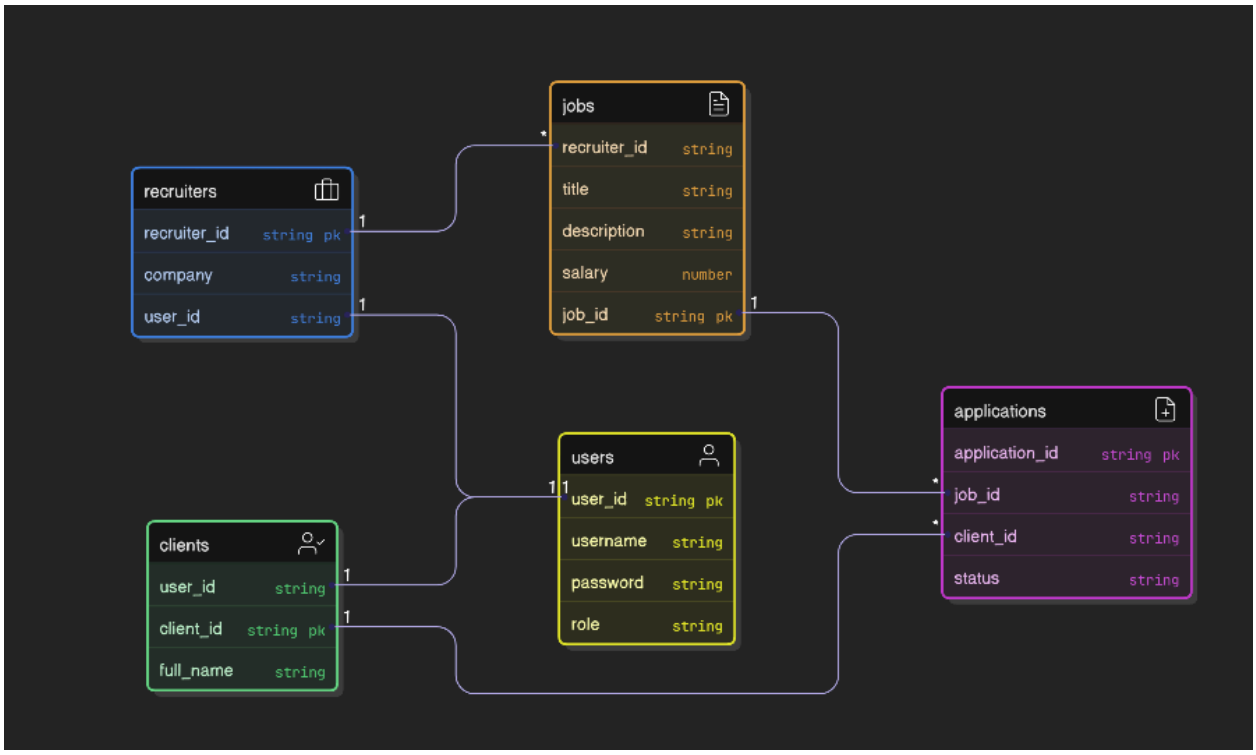
- Simple interface
- Intuitive navigation

Reliability

- MySQL ensures ACID compliance

---

14. ER DIAGRAM



Entities:

1. User(user_id, username, password, role)
2. Recruiter(recruiter_id, user_id, company)
3. Client(client_id, user_id, full_name)
4. Job(job_id, recruiter_id, title, description, salary)
5. Application(application_id, job_id, client_id, status)

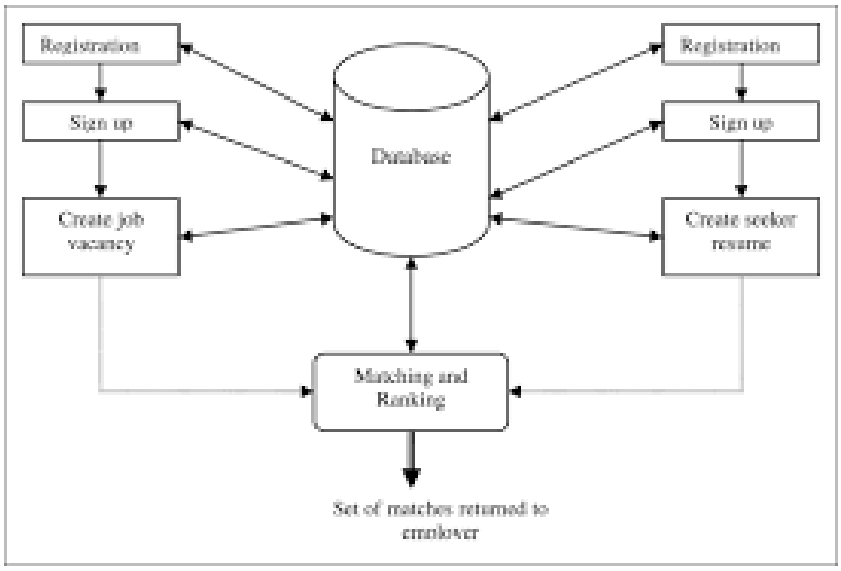
Relationships:

- User 1 — 1 Recruiter
- User 1 — 1 Client
- Recruiter 1 — M Jobs
- Client 1 — M Applications
- Job 1 — M Applications

15. RELATIONAL SCHEMA

```
users(user_id PK, username, password, role)

recruiter(recruiter_id PK, user_id FK, company_name)

client(client_id PK, user_id FK, full_name)

job(job_id PK, recruiter_id FK, title, description, salary)

application(application_id PK, job_id FK, client_id FK, status)
```

16. DATA FLOW DIAGRAMS



DFD Level 0

- Recruiter → System → Database
- Client → System → Database

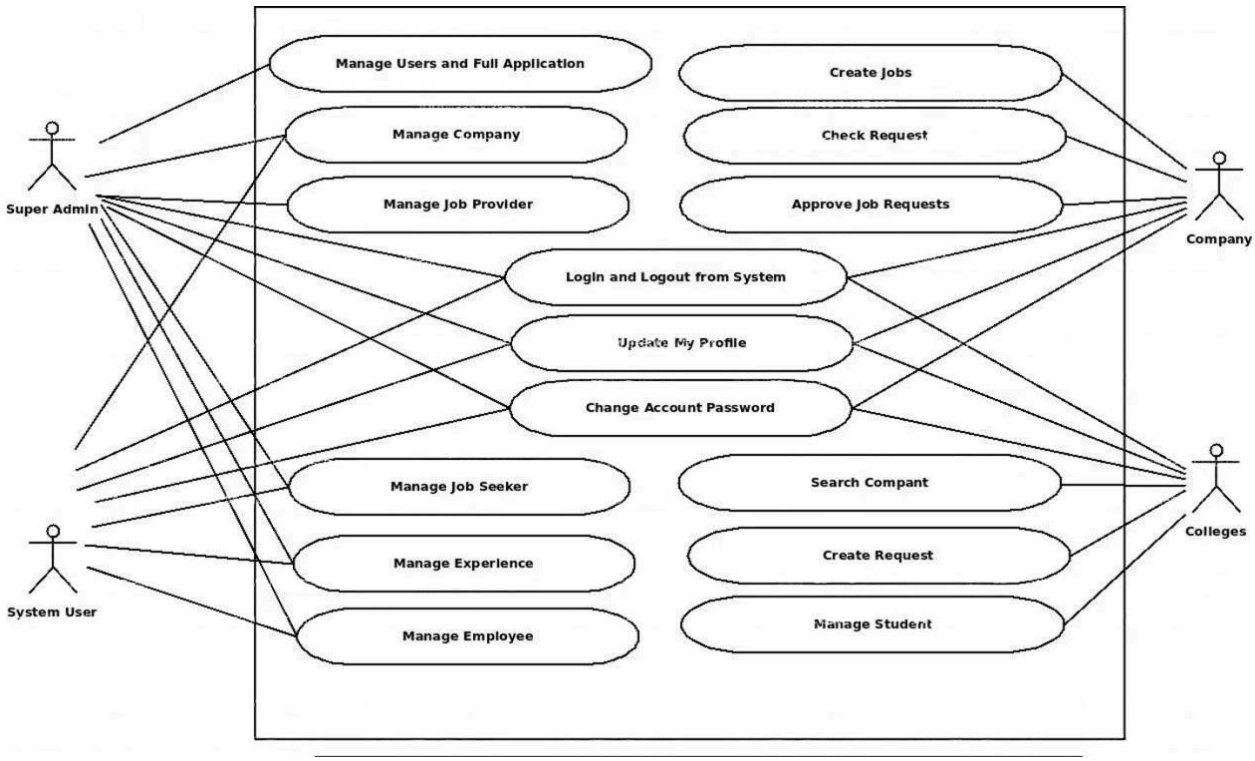
DFD Level 1

Processes:

1. Login & Registration
2. Job Management
3. Job Search
4. Application Management

---

17. USE CASE DIAGRAM (TEXT DESCRIPTION)



Actors:

- Recruiter
- Client

Use Cases:

- Register
- Login
- Post job
- Search job
- Apply job

- View applicants

---

18. NORMALIZATION

1NF

- All tables have atomic values.

2NF

- No partial dependencies since all PKs are single-column.

3NF

- No transitive dependencies.

All tables satisfy 3NF.

---

19. Schemas

1) 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	

2) Recruiter

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	

3) Client

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	

4) 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	

5) 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	

20. IMPLEMENTATION DETAILS

Backend:

- Python
- MySQL Connector
- CRUD operations with exception handling

Frontend:

- Tkinter windows for:
    - Login
    - Register
    - Dashboards
    - Job posting
    - Application view

21. GUI DESIGN

Tkinter components used:

- Labels
- Entry boxes
- Frames
- Buttons
- Listboxes
- Scrollbars

Navigation is menu-driven.

22. SAMPLE CODE SNIPPETS

Hire-Me-Recruitment-Management-System / modules / **login.py**

Code | Blame | 102 lines (88 loc) · 2.83 KB

```python
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')
```

Hire-Me-Recruitment-Management-System / modules / **recruiter.py**

RaghavGupta2910 Add files via upload — 39dbb32 · 9 hours ago — History

Code | Blame | 373 lines (321 loc) · 13.7 KB — Raw

```python
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()
```

Hire-Me-Recruitment-Management-System / modules / register.py

Code Blame 367 lines (314 loc) · 13 KB

```python
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")
```

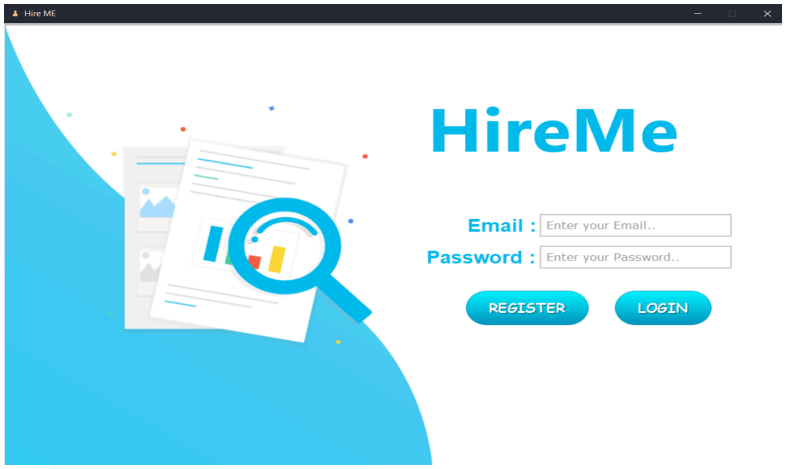
(Add key parts from your GitHub project such as connect.py, login handler, job posting code.)

---

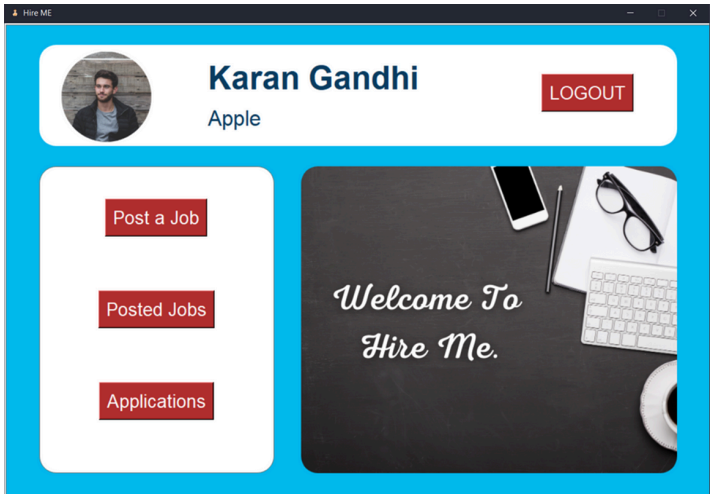
23. SCREENS & SCREENSHOTS

Add screenshots like:

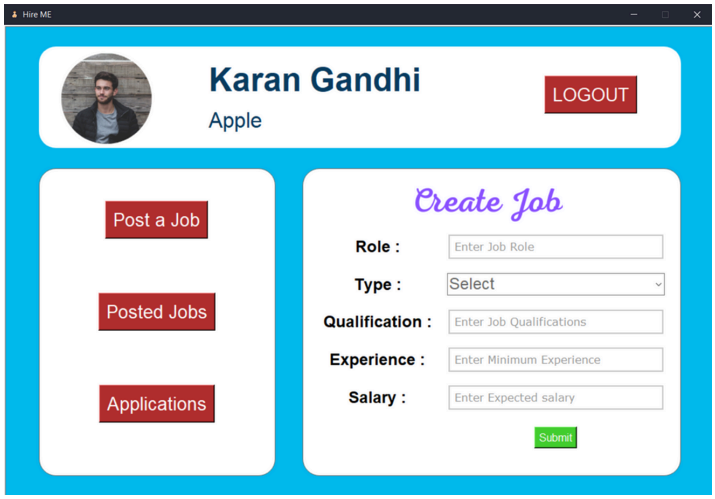
- Login page



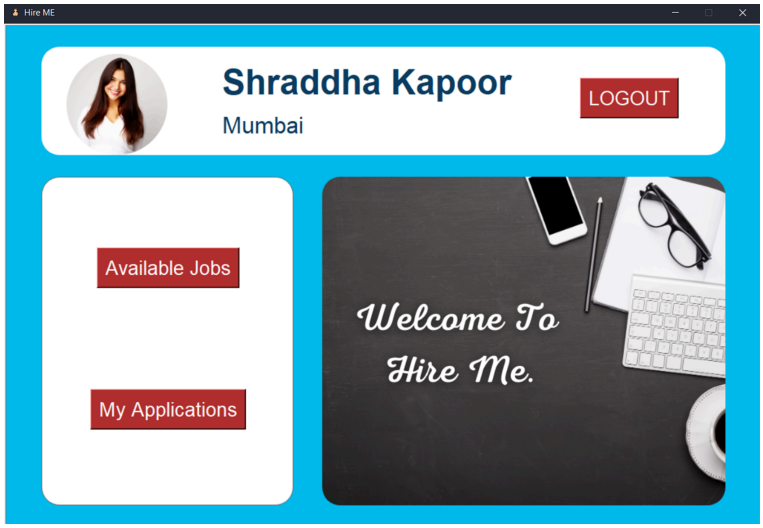
- Recruiter dashboard



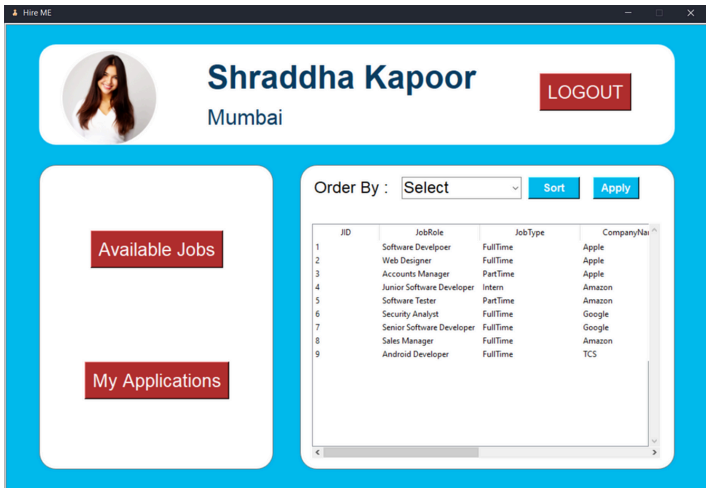
- Job posting



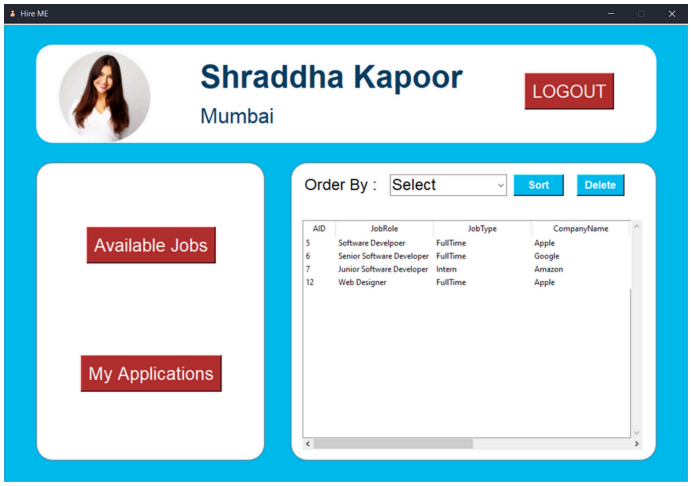
- Client job search



- Available Jobs



- Application list



---

24. TESTING & TEST CASES

Sample test cases:

Test Case ID	Input	Expected Output	Result
TC01	Valid login	Dashboard opens	Pass
TC02	Invalid password	Error message	Pass
TC03	Post job	Job added to DB	Pass
TC04	Apply job	Application stored	Pass

---

25. RESULTS

- All modules work correctly.
- System successfully stores and retrieves recruitment data.

---

26. LIMITATIONS

- No resume upload
- No admin panel
- No email notifications
- Desktop-only

27. CONCLUSION

The HIRE-ME project successfully demonstrates how DBMS principles can be applied to real-world recruitment workflows. It provides both functional features and strong database fundamentals including normalization, schema design, and structured data flow.