SMART HIRING SYSTEM

REPORT OF PROJECT SUBMITTED FOR THE DEGREE OF
MASTER OF COMPUTER APPLICATION

By

ANITESH SAHA
Registration no –231170510006 **University roll no** –11771023006

HRITIK DEY
Registration no –231170510024 **University roll no** –11771023024

TANISHA ROY
Registration no –231170510060 **University roll no** –11771023060

ARPITA SAHA
Registration no –231170510069 **University roll no** –11771023070

UNDER THE SUPERVISION OF

Mr. SUBHASIS CHOWDHURY
Assistant Professor
Department of CA, RCCIIT

AT

RCC INSTITUTE OF INFORMATION TECHNOLOGY
[Affiliated to Maulana Abul Kalam Azad University of Technology, West Bengal]
CANAL SOUTH ROAD, BELIAGHATA, KOLKATA – 700 015

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15, INDIA

CERTIFICATE

The report of the Project titled Smart Hiring System submitted by ANITESH SAHA (Roll No.: 11771023006 of MCA 2nd year 4th Semester of 2025), HRITIK DEY (Roll No.: 11771023024 of MCA 2nd year 4th Semester of 2025), TANISHA ROY (Roll No.: 11771023060 of MCA 2nd year 4th Semester of 2025) and ARPITA SAHA (Roll No.: 11771023070 of MCA 2nd year 4th Semester of 2025) has been prepared under our supervision for the partial fulfillment of the requirements for MCA degree in Maulana Abul Kalam Azad University of Technology.
The report is hereby forwarded.

_____

Mr. SUBHASIS CHOWDHURY
Dept. of Computer Application
RCCIIT, Kolkata
(Internal Supervisor)

Countersigned by

……
Dr. Manas Ghosh
HOD
Department of Computer Application
RCC Institute of Information Technology, Kolkata – 700 015, India

ACKNOWLEDGEMENT

We express our sincere gratitude to Dr. PRAMIT GHOSH of CA Department, RCCIIT and for extending his valuable times for us to take up this problem as a Project.

We are also indebted to Mr. SUBHASIS CHOWDHURY for his unconditional help and inspiration.

Date: ……

ANITESH SAHA
Reg. No.: 231170510006
Roll No.: 11771023006
MCA– 2nd year 4th Semester

HRITIK DEY
Reg. No.: 231170510024
Roll No.: 11771023024
MCA– 2nd year 4th Semester

TANISHA ROY
Reg. No.: 231170510060
Roll No.: 11771023060
MCA– 2nd year 4th Semester

ARPITA SAHA
Reg. No.: 231170510069
Roll No.: 11771023070
MCA– 2nd year 4th Semester

RCC INSTITUTE OF INFORMATION TECHNOLOGY

KOLKATA – 7OOO15, INDIA

CERTIFICATE of ACCEPTANCE

The report of the Project titled Smart Hiring System submitted by ANITESH SAHA (Roll No.: 11771023006 of MCA 2nd year 4th Semester of 2025), HRITIK DEY (Roll No.: 11771023024 of MCA 2nd year 4th Semester of 2025), TANISHA ROY (Roll No.: 11771023060 of MCA 2nd year 4th Semester of 2025) and ARPITA SAHA (Roll No.: 11771023070 of MCA 2nd year 4th Semester of 2025) is hereby recommended to be accepted for the partial fulfillment of the requirements for MCA degree in Maulana Abul Kalam Azad University of Technology.

Name of the Examiners	**Signature with Date**
1. …….	…….
2. ……..	……..
3. ……	……
4. ……..	……

TABLE OF CONTENTS

INTRODUCTION

The Smart Hiring System is a web-based recruitment platform developed using Python, Natural Language Processing (NLP), and Streamlit. The primary objective of this project is to streamline and modernize the hiring process by providing a user-friendly interface that automates key recruitment stages such as resume parsing, candidate shortlisting, and virtual interview scheduling. This system aims to reduce manual effort, improve decision-making accuracy, and enhance the overall efficiency of hiring for both companies and candidates.

The application serves three primary user classes Job Applicants, Company HR, System Administrators. Company HRs initiate the process by posting their job requirements. Only then job seekers can apply by uploading their resumes, which are automatically parsed to extract key information and matched against job criteria. Resumes are scored based on relevance (Matching Skills & Experience), and top candidates appear first. Company HRs can download resumes, accept or reject applicants, and schedule interviews. Accepted candidates will receive an email of interview details and get the meeting link in the application. System administrators manage users, companies, and data integrity.

This application is especially useful for companies that want to expedite their recruitment process by immediately finding top prospects. The domain of the Smart Hiring System (ATS or Application Tracking System) revolves around streamlining the recruitment process by connecting job seekers, companies through a centralized platform. This system is designed to automate and enhance the hiring workflow by leveraging advanced data management and also skill-based shortlisting and resume parsing techniques.

PROBLEM ANALYSIS

The Smart Hiring System is designed to solve common issues in traditional recruitment methods using automated and intelligent solutions. Below is a discussion of major recruitment challenges and how the system addresses them:

1. Inefficient Resume Screening

Problem: Time-consuming and more susceptible to human error, hence resulting in lost opportunities and delays in hiring.

Solution: The system uses NLP-based resume parsing to automatically extract candidate information such as name, email, experience, and skills. The candidates are ranked based on predefined criteria like skill match and experience level which significantly reduces manual effort.

2. Matching Skills and Job Requirements

Problem: Matching candidates' skills to job requirements is challenging, which often leads to irrelevant applications or missed qualified candidates.

Solution: It implements dynamic skill-matching algorithms to evaluate resumes against job specifications. The ranked list of applicants is provided to the hiring managers with the highest relevance of skill and experience.

3. Lack of Application Transparency

Problem: Applicants often face uncertainty after submitting job applications due to the absence of real-time updates or communication regarding their status.

Solution: The Smart Hiring System provides instant confirmation upon resume submission along with a downloadable reference PDF. Additionally, shortlisted candidates automatically receive email notifications with interview details and links, ensuring clear and timely communication throughout the hiring process.

4. Disconnected Recruitment Workflow

Problem: Separate systems for job posting, application tracking, and interview scheduling cause inefficiencies.

Solution: This platform integrates all processes—job posting, resume review, candidate shortlisting, interview scheduling, and communication—into a single interface.

5. Manual Interview Scheduling

Problem: Scheduling interviews manually is time-consuming.

Solution: Hiring managers can schedule interviews within the system, which auto-generates and sends Jitsi meeting links to both parties.

6. Data security and privacy concerns

Problem: In traditional systems, sensitive information like resumes, contact information, and company details is vulnerable to breaches.

Solution: Uses secure encryption methods to safeguard data stored and transmitted. Meets privacy regulations, such as GDPR (General Data Protection Regulation), and gives users control over their data, including the ability to delete or update their profiles.

7. Algorithmic Bias and Inclusivity

Problem: Automated systems may perpetuate biases that exclude certain demographic groups or favour certain profiles.

Solution: Designs the system to avoid over-reliance on specific keywords or factors that may introduce bias.

8. Scalability and Performance Issues

Problem: As the number of users and job postings increases, system performance may degrade, leading to slower processing times.

Solution: The system is designed to be scalable, ensuring consistent performance even with high user demand. Optimized database queries and efficient back-end architecture maintain responsiveness.

REVIEW OF LITERATURE

The **Smart Hiring System** uses modern technologies and recruitment methodologies which make it an efficient and user-friendly hiring platform. Many studies and existing literature on applicant tracking systems (ATS), artificial intelligence in hiring processes, and recruitment automation have contributed to the development and design of this system. An overview of the key works of literature that influenced the idea for the project and execution is mentioned below:

Applicant Tracking Systems (ATS): A study by *Smith et al. (2019)* shows that ATS improves hiring efficiency by reducing time and manual errors in resume screening. Research by *Johnson and Miller (2020)* emphasized that ATS solutions with NLP significantly enhance candidate matching accuracy.

Resume Parsing and Natural Language Processing (NLP): Research of *Patel et al. (2018)* and *Kumar et al. (2021)* shows that NLP increase the accuracy in resume parsing and improves candidate ranking.

Virtual Interviews and Scheduling: *Davis and Carter (2020)* examined the effectiveness of integrated scheduling tools like Jitsi in ATS platforms. It improves user satisfaction and streamlines interview process. Research by *Lee et al. (2022)* highlighted the significance of providing seamless virtual meeting integrations in modern recruitment systems.

Recruitment Bias and Fairness: A study by *Chen et al. (2019)* discovered that ATS systems reduced hiring bias by focusing on skill-based evaluations and standardizing candidate assessments. *Martin et al. (2021)* suggested that fair and transparent algorithms encourage diversity and inclusivity in hiring.

User-Centric Design in Recruitment Platforms: *Baker and Taylor (2019)* highlighted the role of user-centric design in increasing adoption rates among hiring managers and job seekers. Research by *Harris et al. (2021)* introduce that features like real-time feedback, application tracking, and automated notifications enhance the user experience.

Scalability and Data Security: *Singh et al. (2020)* introduce the need of robust database management and encryption to protect user data in ATS platforms. The significance of scalable architectures for ATS systems to manage expanding user bases without experiencing performance degradation was covered by *Garcia and Nelson (2021*).

The Smart Hiring System ensures a strong, equitable, and effective hiring platform by integrating best practices from all aspects of existing literature. It introduces features like automated email notifications, Jitsi integration, and skill-based candidate ranking that are suited for modern recruitment needs while developing upon established technologies and methodologies. This strategy establishes the system as a modern remedy for modern difficulties in hiring.

SDLC MODEL

For our Smart Hiring System, the most appropriate SDLC (Software Development Life Cycle) model is the Incremental Model.

SDLC Model Used: Incremental Model

The Incremental Software Development Life Cycle (SDLC) model builds a system in small sections (increments), adding functionality gradually.

The Smart Hiring System followed the incremental SDLC model to allow for flexibility in feature enhancement, and gradual development. It was developed step-by-step, with each module built and tested individually. Every completed module added to the working system increased overall functionality.

1. Progressive Development:

Our project was developed in phases—starting with core functionalities like user login/signup, job posting, resume upload, and interview scheduling. More features like password reset, admin view, email notifications, etc., were added incrementally.

2. Modular Feature Addition:

The system was developed in distinct functional chunks (increments):

- User registration/login module
- Resume upload and parsing
- Job posting by companies
- Shortlisting logic
- Interview scheduling
- Admin dashboard

3. Feature Enhancements Over Time:

We've extended the system beyond its original version like improved resume updating, job posting. This reflects continuous development which is a core principle of the Incremental Model.

4. Testing at Each Stage:

Each module (like resume parsing, login system, or Jitsi scheduling) can be developed, tested, and deployed independently.

IMPLEMENTATION DETAILS

The **Smart Hiring System** is a web-based application designed to be accessible from any device with an internet connection and a modern browser. The system is developed using **Python** and **Streamlit**, which provide a lightweight and efficient framework for building interactive web applications. Its compatibility with multiple operating systems, including Windows, macOS, and Linux, ensures accessibility for a diverse user base.

The application operates on a server using a **MySQL** database for backend data management. The setup offers a stable, secure, and scalable environment for handling user data, job postings, recruitment criteria and interview scheduling. The database schema is optimized for efficient storage and retrieval, supporting operations like resume parsing, skill matching, and generating ranked candidate lists. The system leverages **Streamlit** for its web interface, providing an intuitive and interactive platform for all users.

Specific Requirements:

- **Functional Requirements:**

  1: User can register and log in.

  2: Candidate uploads resume in PDF format.

  3: Resume is parsed using spaCy and details are extracted.

  4: Company can post jobs with required skills and deadline.

  5: System compares candidate skills with job requirements.

  6: Company shortlists candidates and schedules interviews.

  7: Emails are sent to candidates after shortlisting.

  8: Candidate receives real time notification and reference pdf of his/her application.

  9: Admin can monitor all system.

  10: Users cannot apply to the same job multiple times.

- **Non-Functional Requirements:**

  1: The system must handle concurrent users without significant delays.

  2: Ensure secure login, password encryption, and data protection.

  3: System must support concurrent users.

  4: All sensitive actions should require confirmation (e.g., delete).

  5: Logs should be maintained for all transactions.

Software Requirements:

- **Programming Language:** Python (Version 3.9 or later)
- **Framework:** Streamlit
- **Database:** MySQL
- **Web Server:** Streamlit Cloud
- **Video Conferencing Integration:** Jitsi (for embedded and automated interview scheduling)
- **Supported Browsers:** Google Chrome, Mozilla Firefox, Safari, and Microsoft Edge
- **Operating Systems**: Windows, macOS, Linux

Hardware Requirements:

- **Server Specifications:**
    - **Processor:** Intel Core i3 or higher
    - **RAM:** 8GB or more
    - **Storage:** 256GB SSD or higher for faster performance
    - **Network:** Stable internet connection with a minimum of 10 Mbps speed
- **Client Device Specifications:**
    - Any device (laptop, desktop, tablet, or smartphone)
    - **Processor:** Dual-core CPU (minimum)
    - **RAM:** 4GB or more
    - **Internet:** Stable internet connection

This combination of software and hardware ensures the system's seamless performance, scalability, and compatibility across a wide range of user environments.

COCOMO MODEL

The COCOMO (Constructive Cost Model) is a procedural software cost estimation model. It helps estimate the time, effort, and cost of a software project based on its size in terms of KLOC (Kilo Lines of Code). Our Smart Hiring System project most appropriately follows the Basic COCOMO (Constructive Cost Model) – specifically in the Organic Mode.

➤ **Project Type**: Organic

Our Smart Hiring System is considered Organic because:

- It is a relatively small-scale system.
- Developed in a familiar environment.

Why Basic COCOMO – Organic Mode?:

The **Basic COCOMO model** is used for small to medium-sized software projects where:

- The project is relatively straightforward.
- Teams have good experience in similar applications.
- Requirements are fairly well understood at the start.

Basic COCOMO Formula:

- **Effort (person-months):**
  Effort=a×(KLOC)^b
- **Development Time (months):**
  Time=c×(Effort)^d
- **Constants for Organic Mode:**
  a = 2.4, b = 1.05, c = 2.5, d = 0.38

Estimations Using COCOMO:

1. **Effort:**

   E=2.4×(1.3)^1.05≈3.16 person-months

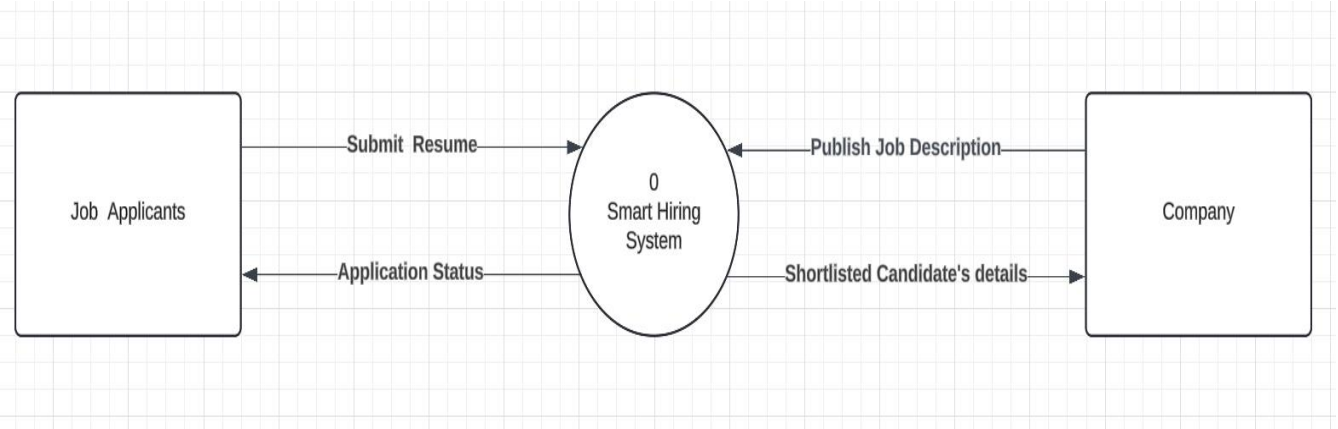
2. **Development Time:**

   T=2.5×(3.16)^0.38≈3.87 months

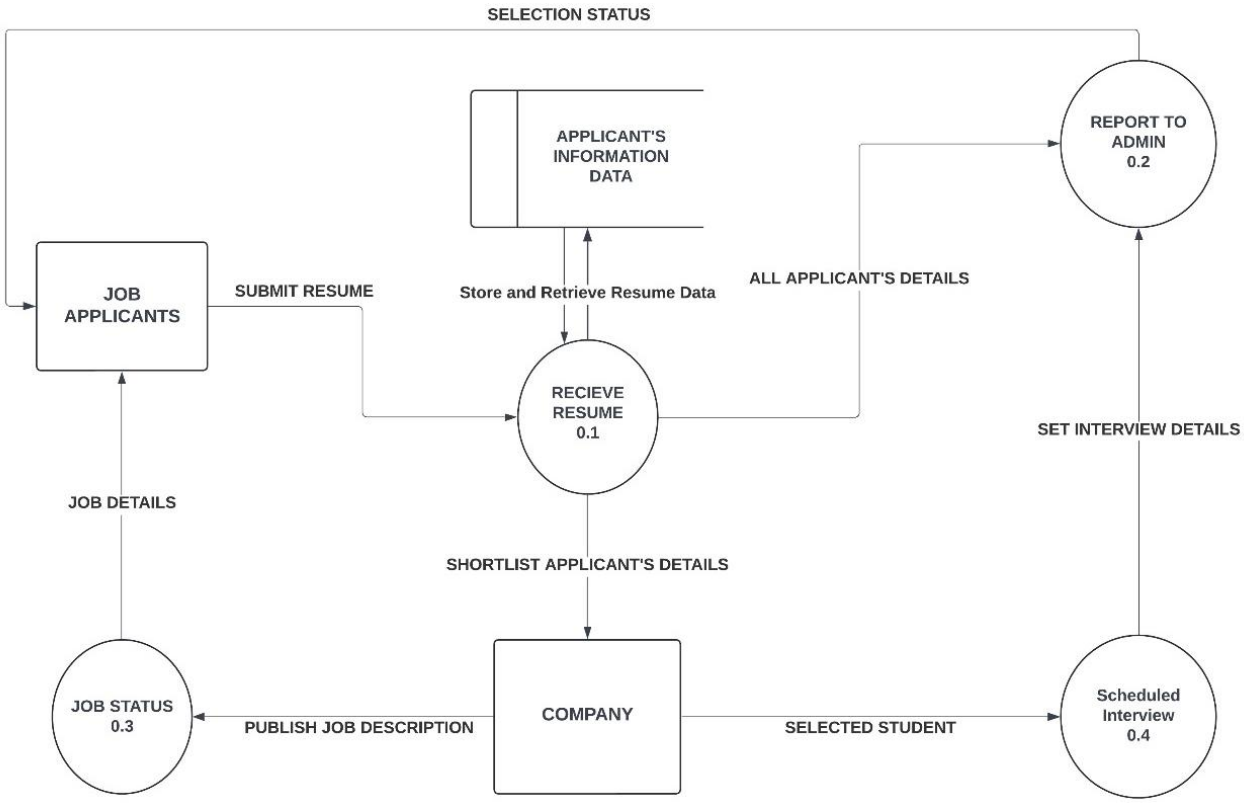
Actual effort may vary due to additional features like resume parsing (SpaCy), scheduling, and mailing. External integrations (like Jitsi/Zoom) might add more complexity. UI/UX enhancements and deployment issues are not included in the estimation.

DFD (DATA FLOW DIAGRAM)

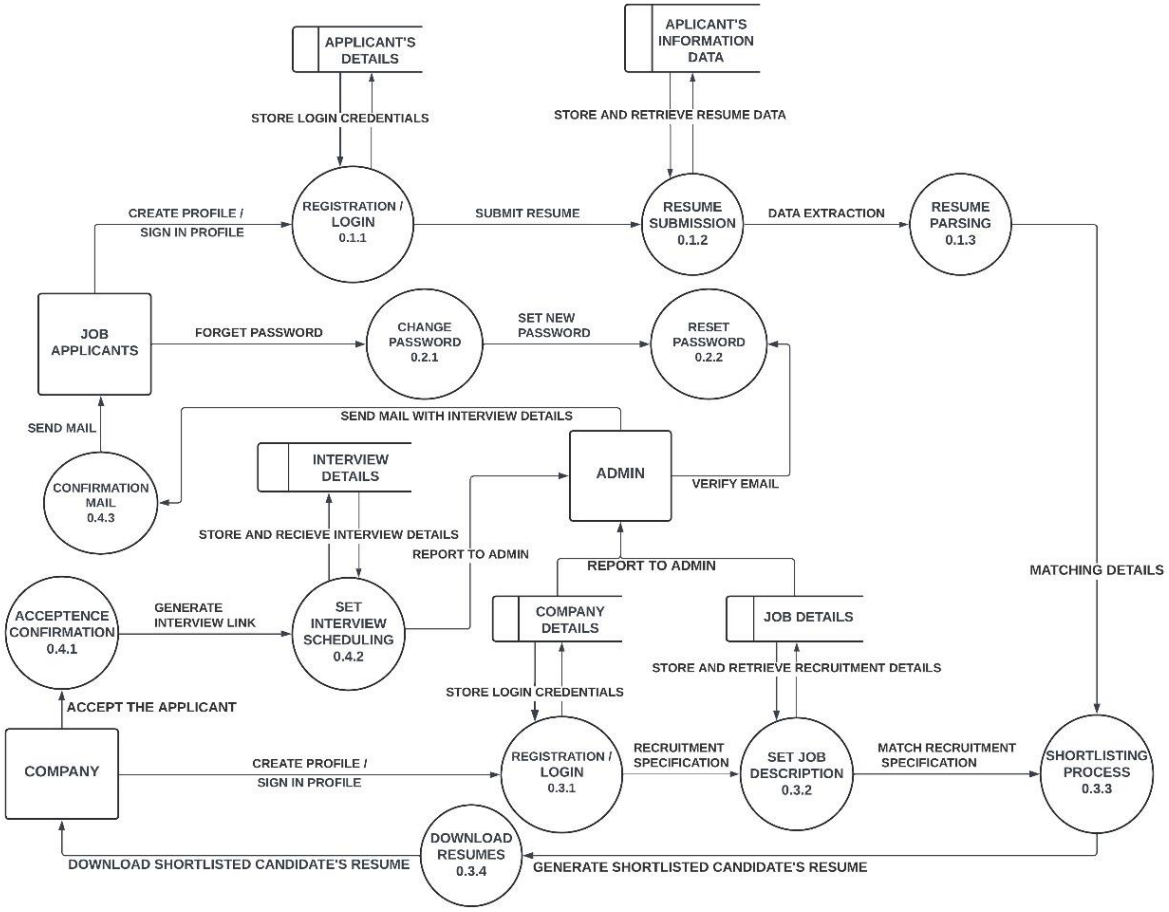
- **Level 0 DFD:**



- **Level 1 DFD:**



- **Level 2 DFD:**


APPLICANT'S DETAILS
APLICANT'S INFORMATION DATA
STORE LOGIN CREDENTIALS
STORE AND RETRIEVE RESUME DATA
CREATE PROFILE / SIGN IN PROFILE
REGISTRATION / LOGIN 0.1.1
SUBMIT RESUME
RESUME SUBMISSION 0.1.2
DATA EXTRACTION
RESUME PARSING 0.1.3
JOB APPLICANTS
FORGET PASSWORD
CHANGE PASSWORD 0.2.1
SET NEW PASSWORD
RESET PASSWORD 0.2.2
SEND MAIL
SEND MAIL WITH INTERVIEW DETAILS
CONFIRMATION MAIL 0.4.3
INTERVIEW DETAILS
ADMIN
VERIFY EMAIL
STORE AND RECIEVE INTERVIEW DETAILS
REPORT TO ADMIN
REPORT TO ADMIN
MATCHING DETAILS
ACCEPTENCE CONFIRMATION 0.4.1
GENERATE INTERVIEW LINK
SET INTERVIEW SCHEDULING 0.4.2
COMPANY DETAILS
JOB DETAILS
STORE AND RETRIEVE RECRUITMENT DETAILS
STORE LOGIN CREDENTIALS
ACCEPT THE APPLICANT
COMPANY
CREATE PROFILE / SIGN IN PROFILE
REGISTRATION / LOGIN 0.3.1
RECRUITMENT SPECIFICATION
SET JOB DESCRIPTION 0.3.2
MATCH RECRUITMENT SPECIFICATION
SHORTLISTING PROCESS 0.3.3
DOWNLOAD RESUMES 0.3.4
DOWNLOAD SHORTLISTED CANDIDATE'S RESUME
GENERATE SHORTLISTED CANDIDATE'S RESUME

ERD (ENTITY RELATIONSHIP DIAGRAM)

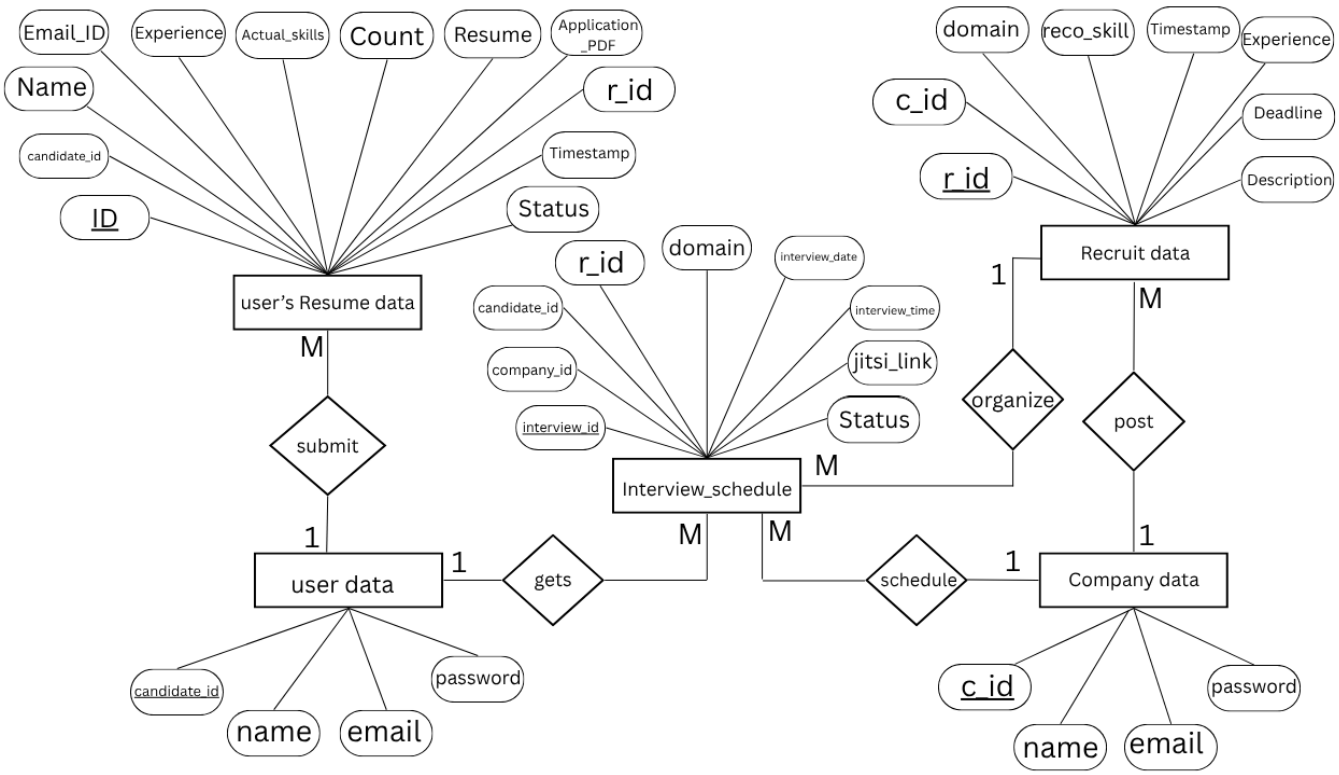
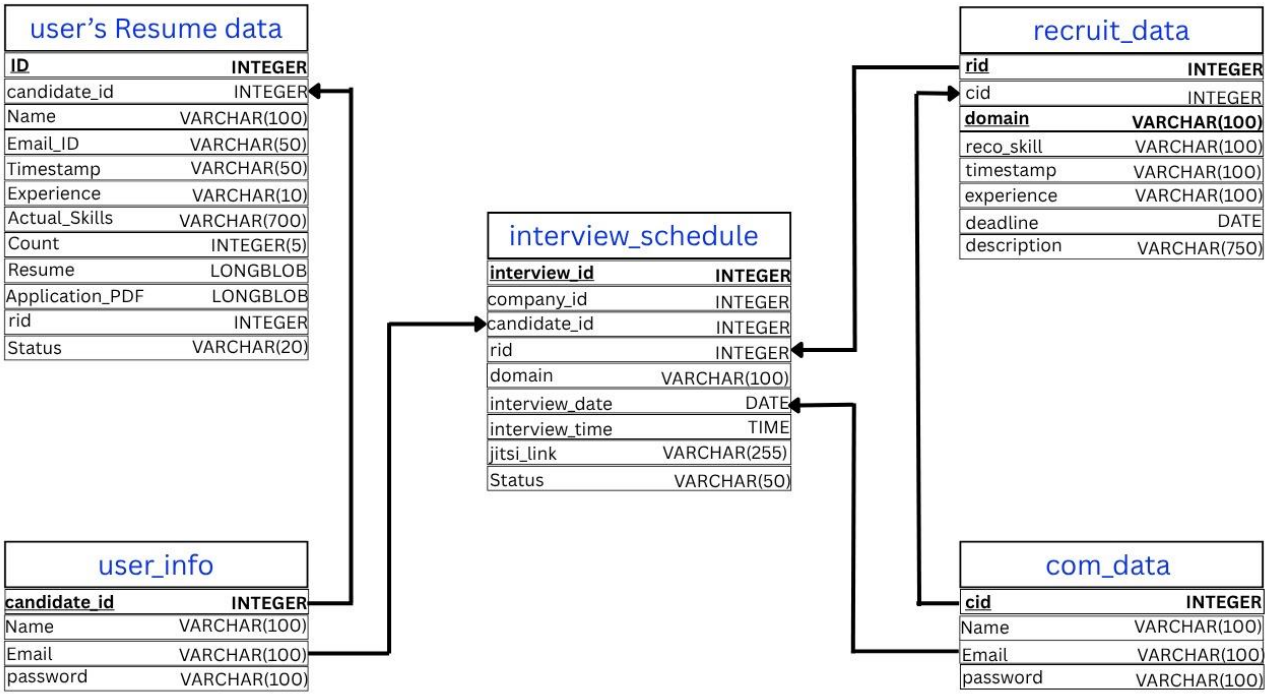


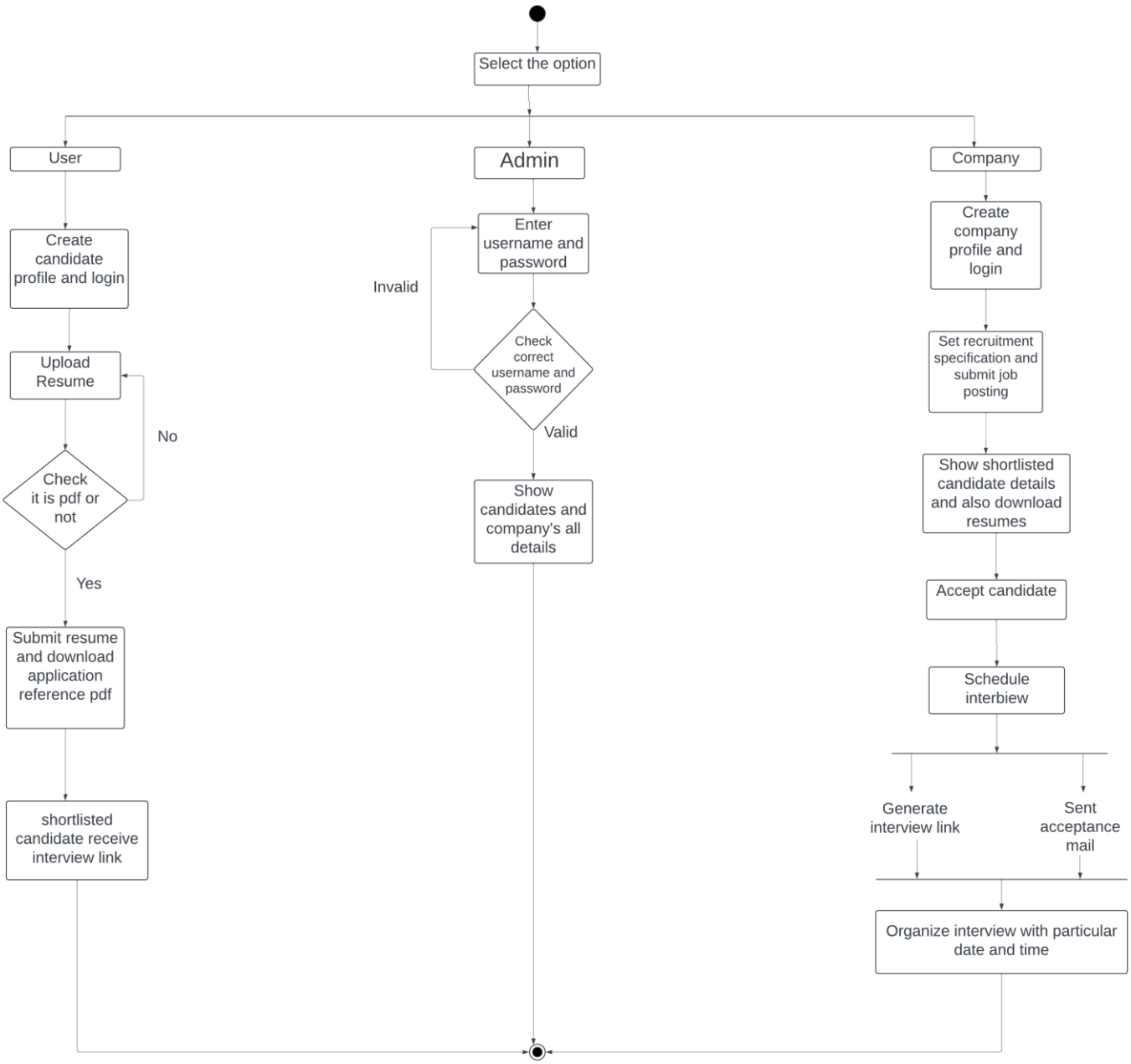
TABLE SCHEMA:



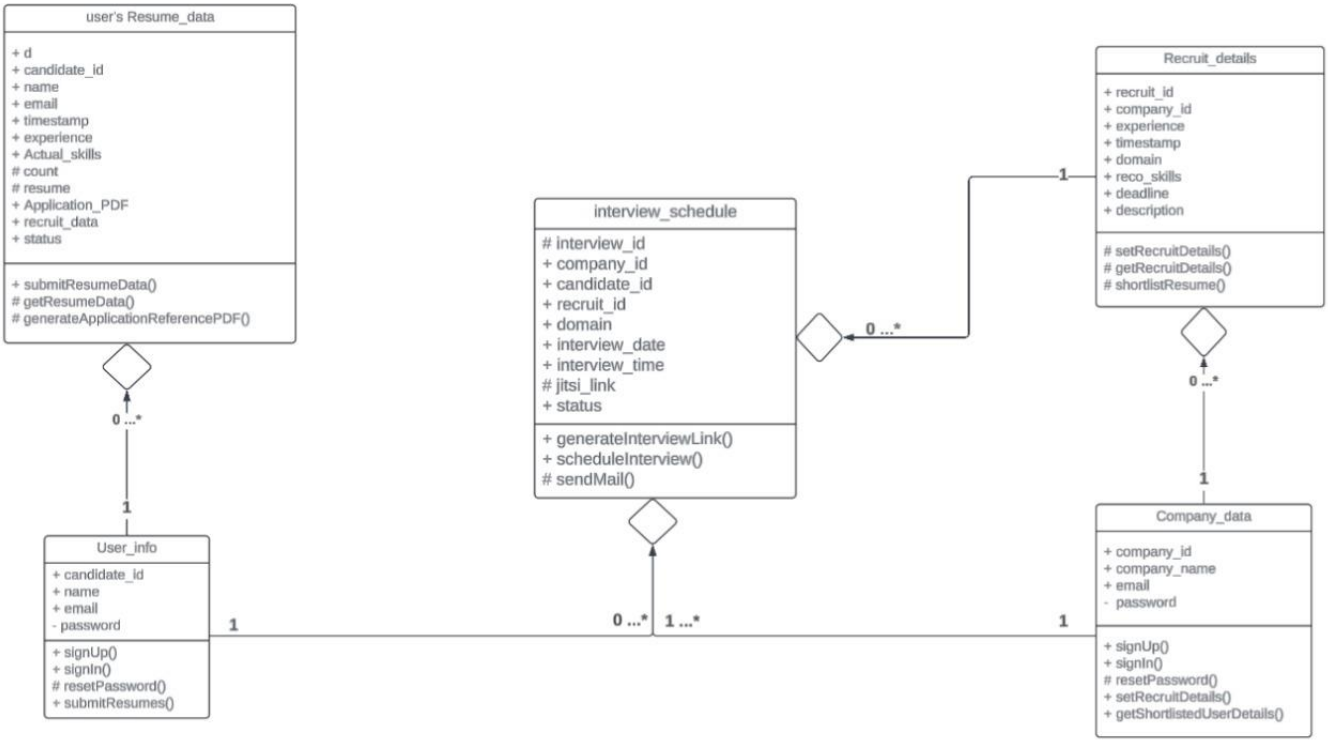
UML DIAGRAM

ACTIVITY DIAGRAM:

For better understanding, here is the representation of the workflow of the proposed system using **Activity diagram.**



CLASS DIAGRAM:



IMPLEMENTATION OF PROBLEM

The implementation of the Smart Hiring System integrates different technologies and components into the recruitment process to automate and streamline it. Below is a detailed breakdown of the program's implementation, including technologies, system architecture, functionalities, and processes.

1. Technologies Used:

The Smart Hiring System uses **Streamlit** for frontend development, which is a Python-based framework that creates an interactive and user-friendly interface for job applicants, hiring managers, and administrators. The backend is powered by **Python**, which handles business logic, resume parsing, and database operations. Libraries like **spacy** and **pyresparser** enable natural language processing (NLP) and structured data extraction from resumes, while **Pandas** helps manage and display tabular data. For email notifications, **smtplib** is used and also **FPDF** is used for generating application reference pdf.

Data is stored using a **MySQL database**, which gives a relational structure to store the user profile, company details, and recruitment records securely. Other tools used are **Jitsi** for handling virtual interview feature, **hashlib** python library for encryption, **Plotly** for optional use as an additional data visualization tool, and **Streamlit Tags** provide the interactivity as tag inputs.

2. System Architecture:

The system architecture is divided into three major layers: Presentation Layer (Frontend), Application Layer (Backend) and Database Layer. The frontend provides an interactive and user-friendly interface to job applicants for applying for specific job just by uploading resumes and receiving feedback, hiring managers for posting jobs requirements, selecting candidates, scheduling interviews and administrators with monitoring and managing system data. The customization of Streamlit along with HTML/CSS ensures that the user interface is seamless.

The backend will handle core functionality, including parsing resumes with `pyresparser`, matching skills and ranking candidates through predefined criteria, dynamic creation of reference letters using ‘FPDF’ and sending automated email notifications via ‘smtplib’ which includes interview details. It would process database queries efficiently in retrieving and updating data.

The database module contains a MySQL database with interconnected tables for user data, company profiles, recruitment records and interview records to provide seamless management and integration of data with encryption using ‘hashlib’.

3. Key Functionalities:

Post Job Openings:

- **Sign up /sign in:** A company can sign up with company name and password and an auto generated company ID is provided. They can sign in securely using that company ID and password. Also, can reset password with email verification.

- **Set requirements specification:** Choose required domain and create new job postings with required skills, experience, deadlines and other descriptions. Create job postings very easily using interactive components.
- **Display previous activity**: A company can see their previous recruitment post on the site.

Apply in Jobs:

- **User Sign up/Sign in:** Users can sign up with name, email and password and an auto generated user ID is provided. They can sign in securely using that user ID, email and password. Also, user can reset password with email verification.
- **Upload Feature:** Users can directly apply in a job by submitting resumes in the Job Posting section where list of all active job openings are shown.
- **Accepted Formats:** Supports PDF formats.
- **Confirmation and Generate Reference PDF:** After applying, the system confirms submission and generates an Application Reference PDF for the applicants.

Resume Analysis:

- **Resume parsing data:** Extract the data from resumes like email, name, phone no, work experience, skills etc.
- **Data insert:** The extracted data are stored in the database in table format.

Candidate Shortlisting:

- **Evaluation Criteria**: Compares extracted resume data against predefined criteria reflecting company priorities (e.g., work experience relevance, skill-job match).
- **Numeric Count:** Generates a count based on the matching skills i.e. managers will easily understand of how many users have the same skills which was provided by them.
- **Efficient Processing**: Ensures timely results, even with large volumes of resumes.
- **Custom Search**: Allows companies to search for specific roles (e.g., Java Developer) and highlights the highest-ranked resumes matching the job requirements.

Candidate Selection:

- **Skill match:** Applicant's skills that matches with company's requirements are to be counted and HR or manager can see the count which is the number of matching skills.
- **Download resume:** HR or manager can download the resume of the applicants from the shortlisted candidate's list.
- **Download basic data:** HR or manager can download the csv file which contains the basic information like name, email, phone no, actual skills, experience of the shortlisted candidates.
- **Automatic mail:** HR or manager can select from the shortlisted candidates and by clicking accept button the applicant will get an auto generated mail from the company.

Interview Management:

- **Schedule interview by company:** Company can schedule interviews with date, time, and auto-generate Jitsi meeting links. When applicant get shortlisted mail from the company, Jitsi link will be create automatically.
- **Get interview details by applicant:** Company send interview details to candidates via email.
- **Join interview link:** On particular scheduled date interview link will be shown in the application interface of that particular applicant and then applicant & HR can join those links and complete the interview process.

Data Management:

- **User and Company Management:** Admin can monitor all recruitment activities and application statuses.
- **Database Integrity:** System ensures security by storing encrypted data of users. System manages expired job posts and cleanup outdated applications and give role-based access control to segregate functionalities for applicants, managers, and admins.

4. Database Design:

The database has five major tables: Job Applicants Data, Company Data, and Recruitment Data, User's Resume Data, Interview Data. The User Data table maintains candidate profiles, including a unique candidate ID, name, email and encrypted password. Whenever an applicant applies for a job, those data are store in User' Resume Data table. It stores information like candidate ID name, email, timestamp, experience, skills which is extracted from the candidates resumes and the resumes, the auto generated application reference pdf and status of the applicant. The Company Data table maintains company profiles, including a unique company ID, name, and encrypted password. The Recruitment Data table tracks job postings with details like recruitment ID, required skills, experience level, and timestamps, deadline, job description. The Interview Data table tracks interview details like interview ID, company ID, candidate ID, interview date, interview time, interview link, status. Relationships are established between these tables to connect user's resume data with recruitment posts and identify job posts based on specific companies and also interview data is connected with company data & user data and user data is connected with user' resume data based on job post on specific companies.

5. Algorithms and Logic:

The resume parsing feature extracts text from uploaded resumes using NLP techniques, identifying key sections like name, skills, and experience. The skill-matching algorithm compares job requirements with candidate skills, calculates a match score based on their overlap, and ranks candidates based on their scores and years of experience. Email automation is implemented using Gmail's SMTP server, enabling the system to send personalized acceptance emails to candidates. When company accept an candidate it will get an autogenerated email with interview details and the system generates a unique interview link using Jitsi Api which is available for both the applicant and the company in their profile.

6. Security Measures:

The system has incorporated various security measures to ensure the protection of data. Passwords and resumes are encrypted, and access to data is restricted based on roles to the users who have permission to view it. The system adheres to privacy laws like GDPR to handle user data with maximum security and transparency.

7. System Workflow:

Job applicants and hiring managers initiate the process by registering and logging into the system. Hiring managers can post job openings by specifying domains, skills, experience levels, and deadlines only after which the applicants are avail to see the job postings. Applicants upload resumes, which are parsed using NLP algorithms to extract details like skills and experience, and the data is stored in the database. Resumes are scored based on criteria such as skill matching and experience relevance, generating a ranked list for hiring managers. Managers can shortlist candidates, download their resumes, and notify accepted applicants via automated emails containing interview details. Interviews are scheduled with date, time, and links displayed on both applicant's and company's profile. Applicants can track their application status, while expired job posts and outdated applications are automatically cleaned up to maintain data integrity. This workflow ensures a seamless, transparent, and user-friendly recruitment process.

8. PROGRAM SOURCE CODE:

```
import time
import streamlit as st
import spacy
spacy.load('en_core_web_sm')
import pandas as pd
import io
import time, datetime
from pyresparser import ResumeParser
from streamlit_tags import st_tags
from PIL import Image
import mysql.connector
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from fpdf import FPDF
import hashlib
import random
import string
from streamlit_option_menu import option_menu
from streamlit_navigation_bar import st_navbar
def delete_old_user_resume_data():
    try:
        cursor.execute("""
            DELETE FROM user
            WHERE STR_TO_DATE(Timestamp, '%Y-%m-%d %H:%i:%s') < DATE_SUB(CURDATE(), INTERVAL 30 DAY)
        """)
        connection.commit()
    except Exception as e:
        st.error(f"⚠ Error while deleting old data: {e}")
def generate_jitsi_link():
    meeting_code = ''.join(random.choices(string.ascii_letters + string.digits, k=10))
    return f"https://meet.jit.si/{meeting_code}"
def get_meeting_link():
    candidate_id = st.session_state.get("candidate_id", None)
    if not candidate_id:
        return []
    cursor.execute("""
        SELECT interview_date, interview_time, jitsi_link
        FROM interview_schedule
        WHERE candidate_id = %s AND interview_date = CURDATE()
```

```
    """, (candidate_id,))
    return cursor.fetchall()
def com_get_meeting_link():
    cursor.execute("""
        SELECT candidate_id, interview_date, interview_time, jitsi_link, rid, domain
        FROM interview_schedule
        WHERE interview_date=CURDATE()
    """)
    result=cursor.fetchall()
    if result:
        for id, date, time, link, rid, domain in result:
            st.subheader("Job Post: "+domain)
            st.write(f"**Post ID:** {rid}")
            st.write(f"**Candidate ID:** {id}")
            st.write(f"🗓️ **Date:** {date} 🕒 **Time:** {time}")
            st.markdown(f"🔗 *[Join Jitsi Meeting]({link})*")
            st.markdown("---")
    else:
        st.info("No Interview Scheduled")
def view_interview_schedule():
    st.subheader("Your Scheduled Interviews")
    if "candidate_id" not in st.session_state:
        st.warning("You must be logged in to view interviews.")
        return
    if st.button("View Interview Link",type="primary"):
        interviews = get_meeting_link()
        if interviews:
            for date, time, link in interviews:
                st.write(f"🗓️ **Date:** {date} 🕒 **Time:** {time}")
                st.markdown(
                    f'<iframe src="{link}" width="800" height="600" allow="camera; microphone; fullscreen" style="border:0;"></iframe>',
                    unsafe_allow_html=True
                )
                st.markdown("---")
        else:
            st.info("No interviews scheduled yet.")
def generate_pdf(name, email, com, role):
    pdf = FPDF()
    pdf.add_page()
    pdf.set_font("Arial", size=12)
    pdf.set_font("Arial", "B", 15)  # Bold font for header
    pdf.cell(200, 10, txt="Smart Hiring System", ln=True, align='C')
    pdf.ln(5)
    pdf.set_font("Arial", "B", 14)
    pdf.cell(200, 10, txt="Application Reference Letter", ln=True, align='C')
    pdf.ln(10)
    pdf.set_font("Arial", size=11)
    pdf.cell(0, 10, txt="Date: " + datetime.datetime.now().strftime('%Y-%m-%d'), ln=True)
    pdf.ln(5)
    pdf.multi_cell(0, 10, txt=f"Dear {name},\n\n  Thank you for your interest in joining {com}. We are pleased to confirm the receipt of your application
for the {role} role.\n\n")
    pdf.set_font("Arial", size=11)  # Regular font for the rest of the text
    pdf.multi_cell(0, 10, txt=(
        "We value the time and effort you have invested in applying for this opportunity. Our team will carefully review your application "
        "to assess its alignment with the role's requirements. If shortlisted, you will be contacted shortly with the next steps in the "
        "selection process.\n\n"
        "We appreciate your interest in contributing to our team and wish you all the best in your career endeavors.\n\n"
        "Best regards,\n"
        "Smart Hiring System Team"
    ))
    pdf_output = pdf.output(dest='S').encode('latin1')  # Return binary data
    return pdf_output
def get_pdf_from_db(email):
    query = "SELECT Application_PDF FROM user WHERE Email_ID = %s"
    cursor.execute(query, (email,))
    result = cursor.fetchone()
    if result:
        return result[0]  # Binary PDF data
    else:
        return None
def send_email(to_email, subject, message):
    try:
        sender_email = "smarthiringsystem2024@gmail.com"    # "20hritikdey@gmail.com"
        sender_password = "ngpx aerc otlw vxxj"  # Store this securely in practice "ewfb zfri yufy mzfv"
        msg = MIMEMultipart()
```

```
        msg['From'] = sender_email
        msg['To'] = to_email
        msg['Subject'] = subject
        msg.attach(MIMEText(message, 'plain'))
        server = smtplib.SMTP('smtp.gmail.com', 587)
        server.starttls()  # Enable security
        server.login(sender_email, sender_password)
        text = msg.as_string()
        server.sendmail(sender_email, to_email, text)
        server.quit()
    except Exception as e:
        print(f'Error sending email: {e}')
connection = mysql.connector.connect(host='localhost', user='root', password='',database='my5')
cursor = connection.cursor()
def insert_data(candidate_id, name, email, timestamp, exp, skills, count, Resume, Application_pdf, rid):
    DB_table_name = 'user'
    insert_sql = f"""
        INSERT INTO {DB_table_name}
        (ID, candidate_id, Name, Email_ID, Timestamp, Experience, Actual_skills, Count, Resume, Application_PDF, rid, status)
        VALUES (0, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s, %s)
    """
    rec_values = (candidate_id, name, email, timestamp, str(exp), skills, count, Resume, Application_pdf, rid, 'Applied')
    cursor.execute(insert_sql, rec_values)
    connection.commit()
def insert_com_data(name, email, password):
    DB_table_name = 'com_data'
    insert_sql = "insert into " + DB_table_name + """
    (Name, email, password) values (%s,%s,%s)"""
    rec_values = (name, email, password)
    cursor.execute(insert_sql, rec_values)
    connection.commit()
    return cursor.lastrowid  # Return the auto-generated cid
def insert_recruit_data(cid,domain,reco_skill,timestamp,experience,deadline,description):
    DB_table_name = 'recruit_data'
    insert_sql = "insert into " + DB_table_name + """
    values (0,%s,%s,%s,%s,%s,%s,%s)"""
    rec_values = (cid,domain,reco_skill,timestamp,experience,deadline,description)
    cursor.execute(insert_sql, rec_values)
    connection.commit()
    return cursor.lastrowid
def fetch_previous_recruitments(cid):
    query = "SELECT rid,domain,reco_skill,timestamp,experience,deadline,description FROM recruit_data WHERE cid = %s"
    cursor.execute(query, (cid,))
    return cursor.fetchall()
def update_data(email, rid, timestamp, exp, skills, resume, application_pdf):
    DB_table_name = 'user'
    update_sql = f"""
    UPDATE {DB_table_name}
    SET timestamp = %s, experience = %s, actual_skills = %s, resume = %s, Application_PDF = %s
    WHERE email_id = %s AND rid = %s
    """
    cursor.execute(update_sql, (timestamp, exp, skills, resume, application_pdf, email, rid))
    connection.commit()
def delete_expired_jobs():
    cursor.execute("DELETE FROM recruit_data WHERE deadline < CURDATE()")
    connection.commit()
def hash_password(password):
    return hashlib.sha256(password.encode()).hexdigest()
def signup(name, email, password):
    hashed_pwd = hash_password(password)
    cursor.execute("SELECT * FROM user_info WHERE email = %s", (email,))
    result = cursor.fetchone()
    if result:
        return "⚠ Email already registered. Try logging in."
    else:
        # Insert new user
        insert_sql = "INSERT INTO user_info (name, email, password) VALUES (%s, %s, %s)"
        cursor.execute(insert_sql, (name, email, hashed_pwd))
        connection.commit()
        return cursor.lastrowid
def signin(candidate_id, email, password):
    hashed_pwd = hash_password(password)
    cursor.execute("SELECT * FROM user_info WHERE candidate_id = %s AND email=%s AND password = %s", (candidate_id, email,
hashed_pwd))
    result = cursor.fetchone()
```

```
    if result:
        st.session_state.logged_in = True
        st.session_state.candidate_id = result[0]
        st.session_state.candidate_name = result[1]
        return result
    else:
        st.error("Invalid username or password.")
        return None
def reset_password(email, new_password):
    cursor.execute("SELECT * FROM user_info WHERE email = %s", (email,))
    result = cursor.fetchone()
    if result:
        cursor.execute("UPDATE user_info SET password = %s WHERE email = %s", (new_password, email))
        connection.commit()
        st.success("Password reset successful! You can now sign in.")
        st.session_state.reset_mode = False
    else:
        st.error("Username not found!")
def reset_password_company(email, new_password):
    cursor.execute("SELECT * FROM com_data WHERE email = %s", (email,))
    result = cursor.fetchone()
    if result:
        cursor.execute("UPDATE com_data SET password = %s WHERE email = %s", (new_password, email))
        connection.commit()
        st.success("Password reset successful! You can now sign in.")
        st.session_state.reset_mode = False
    else:
        st.error("Username not found!")
def company_signin(cid, password):
    has_pass=hash_password(password)
    cursor.execute("SELECT * FROM com_data WHERE cid = %s AND password = %s", (cid, has_pass))
    result = cursor.fetchone()
    st.write("Entered (hashed):", has_pass)
    if result:
        st.session_state.company_logged_in = True
        st.session_state.company_id = result[0]
        st.session_state.company_name = result[1]
        return result
    else:
        st.error("⚠ Incorrect Company ID or Password.")
        return None
def run():
    if "reset_mode" not in st.session_state:
        st.session_state.reset_mode = False
    st.sidebar.markdown("### **Choose User**")
    activities = ["User 👨‍💼", "Admin 👤","Company 🏢"]
    choice = st.sidebar.selectbox("**Choose among the given options:**", activities)
    DB_table_name3 = 'user_info'
    table_sql3 = "CREATE TABLE IF NOT EXISTS " + DB_table_name3 + """
            (candidate_id INT NOT NULL AUTO_INCREMENT,
            name varchar(100) NOT NULL,
            email varchar(100) NOT NULL,
            password varchar(100) NOT NULL,
            PRIMARY KEY (candidate_id)) AUTO_INCREMENT=1001;
            """
    cursor.execute(table_sql3)
    DB_table_name1 = 'com_data'
    table_sql1 = "CREATE TABLE IF NOT EXISTS " + DB_table_name1 + """
                (cid INT NOT NULL AUTO_INCREMENT,
                Name varchar(100) NOT NULL UNIQUE,
                email VARCHAR(50) NOT NULL,
                password VARCHAR(100) NOT NULL,
                PRIMARY KEY (cid)) AUTO_INCREMENT=101;
                """
    cursor.execute(table_sql1)
    DB_table_name2 = 'recruit_data'
    table_sql2 = "CREATE TABLE IF NOT EXISTS " + DB_table_name2 + """
            (rid INT NOT NULL AUTO_INCREMENT,
            cid INT NOT NULL,
            domain varchar(100) NOT NULL,
            reco_skill varchar(100) NOT NULL,
            timestamp varchar(100) NOT NULL,
            experience varchar(100) NOT NULL,
            deadline date NOT NULL,
            description varchar(750) NOT NULL,
```

```
        PRIMARY KEY (rid,domain),
        FOREIGN KEY(cid) REFERENCES com_data(cid) ON DELETE CASCADE)AUTO_INCREMENT=5001;
        """
    cursor.execute(table_sql2)
    connection.commit()
    DB_table_name = 'user'
    table_sql = "CREATE TABLE IF NOT EXISTS " + DB_table_name + """
                (ID INT NOT NULL AUTO_INCREMENT,
                 candidate_id INT NOT NULL,
                 Name varchar(100) NOT NULL,
                 Email_ID VARCHAR(50) NOT NULL,
                 Timestamp VARCHAR(50) NOT NULL,
                 Experience VARCHAR(10) NOT NULL,
                 Actual_skills VARCHAR(700) NOT NULL,
                 Count INT(5) NOT NULL,
                 Resume LONGBLOB NOT NULL,
                 Application_PDF LONGBLOB NOT NULL,
                 rid INT NOT NULL,
                 status VARCHAR(20) DEFAULT 'Applied',
                 PRIMARY KEY (ID),
                 FOREIGN KEY(candidate_id) REFERENCES user_info(candidate_id) ON DELETE CASCADE);
                """
    cursor.execute(table_sql)
    DB_table_name4 = 'interview_schedule'
    table_sql4 = "CREATE TABLE IF NOT EXISTS " + DB_table_name4 + """
            (
               interview_id INT AUTO_INCREMENT PRIMARY KEY,
               company_id INT NOT NULL,
               candidate_id INT NOT NULL,
               rid INT NOT NULL,
               domain varchar(100) NOT NULL,
               interview_date DATE NOT NULL,
               interview_time TIME NOT NULL,
               jitsi_link VARCHAR(255) NOT NULL,
               status VARCHAR(50) DEFAULT 'Scheduled',
               FOREIGN KEY (company_id) REFERENCES com_data(cid) ON DELETE CASCADE,
               FOREIGN KEY (rid, domain) REFERENCES recruit_data(rid, domain) ON DELETE CASCADE,
               FOREIGN KEY (candidate_id) REFERENCES user_info(candidate_id) ON DELETE CASCADE
            );
            """
    cursor.execute(table_sql4)
    delete_old_user_resume_data()
    if choice == 'User 👨🏻 ':
            menu = ["Sign In", "Sign Up"]
            choice = st.sidebar.selectbox("Select an option", menu)
            if choice == "Sign Up":
               original_title ='"<p style="font-size: 45px; font-weight: bold;color: #333;
               text-shadow:
               1px 1px 0px #eab,  /* Top-left shadow (light) */
               2px 2px 0px #ccc,  /* Middle-left shadow */
               3px 3px 0px #999;  /* Bottom-left shadow (dark) */
               font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>'"
               st.markdown(original_title, unsafe_allow_html=True)
               st.subheader("Create a New Account")
               with st.form("signup_form"):
                  name = st.text_input("Full Name")
                  email = st.text_input("Email")
                  password = st.text_input("Password", type="password")
                  if st.form_submit_button("Sign Up",type="primary"):
                     cursor.execute(f"SELECT * FROM {DB_table_name3} WHERE email = %s", (email,))
                     user_exists = cursor.fetchone()
                     if user_exists:
                        st.error(f"**_A user with the name '{name}' already exists. Please use a different name or log in._**")
                     else:
                        try:
                           message = signup(name, email, password)
                           st.success(f"**_:blue[Your profile is created. Your candidate ID is {message}]_**")
                        except Exception as e:
                           st.error(f"**_An unexpected error occurred: {e}_**")
            elif choice == "Sign In":
               if st.session_state.reset_mode:
                  st.subheader("🔁 Reset Password")
                  with st.form("reset_form"):
                     email = st.text_input("Registered Email")
                     new_password = st.text_input("New Password", type="password")
```

```
            confirm_password = st.text_input("Confirm Password", type="password")
            reset_btn = st.form_submit_button("Reset Password",type="primary")
            if reset_btn:
                if new_password != confirm_password:
                    st.error("❌ Passwords do not match.")
                else:
                    hashed = hash_password(new_password)
                    reset_password(email, hashed)
        if st.button("⬅ Back to Sign In"):
            st.session_state.reset_mode = False
        return  # ✅ Important: exit to prevent running login/signup below
    if not st.session_state.get("logged_in", False):
        original_title ="<p style="font-size: 50px; font-weight: bold;color: #333;
        text-shadow:
        1px 1px 0px #eab,  /* Top-left shadow (light) */
        2px 2px 0px #ccc,  /* Middle-left shadow */
        3px 3px 0px #999;  /* Bottom-left shadow (dark) */
        font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>"
        st.markdown(original_title, unsafe_allow_html=True)
        st.subheader("Login to Your Account")
        with st.form("signin_form"):
            candidate_id = st.text_input("Candidate ID")
            email = st.text_input("Email")
            password = st.text_input("Password", type="password")
            if st.form_submit_button("Sign In",type="primary"):
                user = signin(candidate_id, email, password)
                if user:
                    st.rerun()
            forgot = st.form_submit_button("Forgot Password?")
            if forgot:
                st.session_state.reset_mode = True
                st.rerun()
    else:
            pages = {
                "Profile": "🏠 Home",
                "Jobs": "💼 Jobs",
                "Applied Jobs":"💼 Applied Jobs",
                "View Interview Schedule": "🗓️ Interviews",
                "About":"💡 About",
                "Logout": "🔐 Logout"
            }
            selected_page = st_navbar(list(pages.values()))
            label_to_key = {v: k for k, v in pages.items()}
            page = label_to_key[selected_page]
            st.toast(f"**:green[Welcome back, {st.session_state['candidate_name']}! 🎉]**")
            page_bg_img = f"""
            <style>
            [data-testid="stAppViewContainer"] > .main {{
            background-image: url("https://i.postimg.cc/LXgf9ZvP/Untitled-design-43.png");
            background-position: center center;
            /* Make image fixed */
            background-attachment: fixed;
            /* Not repeat images */
            background-repeat: no-repeat;
            /* Set background size auto */
            background-size: 100%;
            }}
            [data-testid="stHeader"] {{
            background: rgba(0,0,0,0);
            }}
            </style>
            """
            st.markdown(page_bg_img, unsafe_allow_html=True)
            if page == "Jobs":
                candidate_id = st.session_state.get("candidate_id", None)
                if candidate_id:
                    st.header("Browse Job Listings")
                    cursor.execute("""
                        SELECT r.rid, c.Name, r.domain, r.reco_skill, r.timestamp, r.experience, r.deadline, r.description, r.cid FROM recruit_data r,
com_data c
                        WHERE r.cid=c.cid and deadline >= CURDATE()
                        AND rid NOT IN (
                            SELECT rid FROM user WHERE candidate_id = %s
                        )
```

```
                    ORDER BY timestamp DESC
                """, (candidate_id,))
                job_posts = cursor.fetchall()
                if job_posts:
                    for job in job_posts:
                        job_id = job[0]
                        job_com = job[1]
                        job_title = job[2]
                        job_exp = job[5]
                        job_deadline = job[6]
                        Job_desc=job[7]
                        st.subheader("Job Post: "+job_title)
                        st.write(f"**Company:** {job_com.upper()}")
                        st.write(f"**Experience Required:** {job_exp}")
                        st.write(f"**Deadline:** {job_deadline}")
                        st.write(f"**Description:** {Job_desc}")
                        pdf_file = st.file_uploader("Upload Your Resume", type=["pdf"], key=f"resume_{job_id}")
                        post = st.button(f"Submit Resume", key=f"submit_{job_id}", type="primary")
                        st.markdown("---")
                        if post:
                            if pdf_file is not None:
                                resume_binary = pdf_file.read()
                                resume_data = ResumeParser(pdf_file).get_extracted_data()
                                sql = "SELECT email_id FROM user WHERE email_id = %s AND rid = %s"
                                cursor.execute(sql, (resume_data['email'], job_id))
                                result = cursor.fetchone()
                                if result:
                                    st.warning("**⚠ Something went wrong. You CV is already submitted for this post.**")
                                else:
                                    application_pdf = generate_pdf(resume_data["name"], resume_data["email"], job_com, job_title)

                                    insert_data(candidate_id,resume_data['name'], resume_data['email'], datetime.datetime.now(),
                                                str(resume_data['total_experience']), str(resume_data['skills']),
                                                0, resume_binary, application_pdf,job_id)
                                    if resume_data:
                                        st.subheader("**Resume Analysis:**")
                                        st.success(f"**_ ✅ Congratulations {resume_data['name']} 🎉. Your application has been Submitted._**")
                                        st.text(f"Name: {resume_data['name']}")
                                        st.text(f"Email: {resume_data['email']}")
                                        st.text(f"Experience: {resume_data['total_experience']} years")
                                    stored_pdf = get_pdf_from_db(resume_data['email'])
                                    if stored_pdf:
                                        st.download_button(
                                            label="📥 Download Application Reference PDF",
                                            data=stored_pdf,
                                            file_name=f"Application_{resume_data['name'].replace(' ', '_')}.pdf",
                                            mime="application/pdf",
                                            type="primary"
                                        )
                                        st.markdown("---")
                                    else:
                                        st.error("**⚠ An unexpected error occurred while processing your resume.**")
                            else:
                                st.error("**⚠ Please upload your resume before submitting.**")
                else:
                    st.write("No job posts available yet.")
            elif page =="Applied Jobs":
                candidate_id = st.session_state.get("candidate_id", None)
                if candidate_id:
                    st.header("Applied Job List")
                    cursor.execute("""
                        SELECT r.rid, c.Name, r.domain, r.reco_skill, u.Timestamp, r.experience, r.deadline, r.description, r.cid, u.status FROM
recruit_data r, com_data c, user u
                        WHERE r.cid=c.cid and u.rid=r.rid and u.candidate_id = %s
                        ORDER BY u.Timestamp DESC
                    """, (candidate_id,))
                    job_posts = cursor.fetchall()
                    if job_posts:
                        for job in job_posts:
                            job_id = job[0]
                            job_com = job[1]
                            job_title = job[2]
                            job_exp = job[5]
                            job_deadline = job[6]
                            Job_desc=job[7]
```

```
                        Job_status=job[9]
                        st.subheader("Job Post: "+job_title)
                        st.write(f"**Company:** {job_com.upper()}")
                        st.write(f"**Experience Required:** {job_exp}")
                        st.write(f"**Deadline:** {job_deadline}")
                        st.write(f"**Description:** {Job_desc}")
                        st.write(f"**Status:** {Job_status}")
                        st.markdown("---")
                    else:
                        st.write("You have not applied to any Job posts yet.")
                elif page == "View Interview Schedule":
                    view_interview_schedule()
                elif page == "Profile":
                    st.header(" Your Profile")
                    candidate_id = st.session_state.get("candidate_id", None)
                    if candidate_id:
                        st.subheader(" Basic Information")
                        cursor.execute("SELECT name, email FROM user_info WHERE candidate_id = %s", (candidate_id,))
                        user_data = cursor.fetchone()
                        if user_data:
                            st.write(f"**Name:** {user_data[0]}")
                            st.write(f"**Email:** {user_data[1]}")
                            st.write(f"**Candidate ID:** {candidate_id}")
                        else:
                            st.warning("**Unable to fetch profile information.**")
                        st.subheader(" Applications Summary")
                        cursor.execute("SELECT COUNT(*) FROM user WHERE candidate_id = %s", (candidate_id,))
                        apps_count = cursor.fetchone()[0]
                        st.write(f"**Total Applications Submitted:** {apps_count}")
                        st.subheader(" Interview Calls")
                        cursor.execute("SELECT COUNT(*) FROM interview_schedule WHERE candidate_id = %s", (candidate_id,))
                        interview_count = cursor.fetchone()[0]
                        st.write(f"**Total Interview Calls Received:** {interview_count}")
                elif page == "About":
                    st.header("About Smart Hiring System")
                    st.write("Welcome to the Smart Hiring System, a cutting-edge recruitment platform designed to connect talented candidates with top-tier organizations. Our platform streamlines the hiring process by offering an intuitive interface for job seekers, recruiters, and administrators alike.")
                    st.subheader("Key Features:")
                    columns = st.columns(3)
                    with columns[0]:
                        with st.container(height=250):
                            st.write("**For Candidates:**")
                            st.write("Effortlessly apply for jobs, upload resumes, and track your applications. Get instant notifications about interview schedules and stay ahead in your job search.")
                    with columns[1]:
                        with st.container(height=250):
                            st.write("**For Companies:**")
                            st.write(" Post job openings, review candidate applications, and schedule interviews seamlessly. Utilize intelligent filters to shortlist the best talent for your organization.")
                    with columns[2]:
                        with st.container(height=250):
                            st.write("**For Admins:**")
                            st.write("Oversee platform activities, manage user data, and ensure smooth operations across all stakeholders.")
                elif page=="Logout":
                    st.session_state.clear()
                    st.rerun()
    elif choice=='Admin ':
        try:
            original_title ='''<p style="font-size: 50px; font-weight: bold;color: #333;
            text-shadow:
            1px 1px 0px #eab,  /* Top-left shadow (light) */
            2px 2px 0px #ccc,  /* Middle-left shadow */
            3px 3px 0px #999;  /* Bottom-left shadow (dark) */
            font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>'''
            st.markdown(original_title, unsafe_allow_html=True)
            if "admin_logged_in" not in st.session_state:
                st.session_state.admin_logged_in = False
            if not st.session_state.get("admin_logged_in", False):
                st.subheader("Admin Login Page")
                with st.form("signin_form"):
                    admin_user=st.text_input("**Username**")
                    admin_password=st.text_input("**Password**",type='password')
                    if st.form_submit_button("Sign In",type="primary"):
                        with st.spinner(':blue[Authenticating...]'):
```

```
                    time.sleep(2)
                    if admin_user == 'admin' and admin_password == 'admin123':
                        st.session_state.admin_logged_in = True
                        st.rerun()
                    else:
                        st.error("**Invalid credentials. Please try again.**")
            else:
                st.toast(f"**:green[Welcome back Admin 🎉]**")
                page_bg_img = f"""
                <style>
                [data-testid="stAppViewContainer"] > .main {{
                background-image: url("https://i.postimg.cc/LXgf9ZvP/Untitled-design-43.png");
                background-position: center center;
                background-attachment: fixed;
                background-repeat: no-repeat;
                background-size: 100%;
                }}
                [data-testid="stHeader"] {{
                background: rgba(0,0,0,0);
                }}
                </style>
                """
                st.markdown(page_bg_img, unsafe_allow_html=True)
                st.markdown("### 👤 Admin Dashboard")
                cursor.execute('"SELECT candidate_id,name,email FROM user_info"')
                data = cursor.fetchall()
                st.header("**User's Data**")
                df = pd.DataFrame(data, columns=['Candidate_ID', 'Name', 'Email'])
                st.dataframe(df)
                cursor.execute('"SELECT u.candidate_id, u.Name, u.Email_ID, u.Timestamp, u.Experience, u.Actual_skills, u.rid, c.Name, u.status FROM
user u, recruit_data r, com_data c where u.rid=r.rid and r.cid=c.cid"')
                data = cursor.fetchall()
                st.header("**Job Application Data**")
                df = pd.DataFrame(data, columns=['Candidate ID', 'Name', 'Email', 'Timestamp', 'Experience', 'Actual Skills','Recruit Id','Company','Status'])
                st.dataframe(df)
                cursor.execute('"SELECT interview_id, company_id, candidate_id,interview_date,interview_time,status FROM interview_schedule"')
                data1 = cursor.fetchall()
                st.header("**Interview Details**")
                df = pd.DataFrame(data1, columns=['interview_id', 'company_id', 'candidate_id','interview_date','interview_time','status'])
                st.dataframe(df)
                cursor.execute('"SELECT cid, Name, email FROM com_data"')
                data1 = cursor.fetchall()
                st.header("**Company's Data**")
                df = pd.DataFrame(data1, columns=['Company ID', 'Company Name', 'email'])
                st.dataframe(df)
                c_id=st.text_input("**Enter Company ID for Show company's Previous Posts:**")
                loadnow1=st.button("Show Previous Posts",type="primary")
                if "loadnow1_state" not in st.session_state:
                    st.session_state.loadnow1_state= False
                if loadnow1 or st.session_state.loadnow1_state:
                    st.session_state.loadnow1_state=True
                    if c_id.isdigit():
                        c_id=int(c_id)
                        cursor.execute(f"SELECT * FROM com_data WHERE cid = %s", (c_id,))
                        company_exists1 = cursor.fetchone()
                        if company_exists1:
                            st.subheader(f"Previous recruitments of Company ID- {c_id}:")
                            recruitments = fetch_previous_recruitments(c_id)
                            if recruitments:
                                recruit_df = pd.DataFrame(recruitments, columns=['RID', 'Domain', 'Reco_Skills', 'Timestamp', 'Experience','Deadline','Job
Description'])
                                st.dataframe(recruit_df)
                            else:
                                st.info(f"**No previous recruitment posts for {c_id}.**")
                        else:
                            st.error("**There is no such ID exists.**")
                    else:
                        st.error("**⚠ Nothing to show**")
        except Exception as main_error:
            st.error(f"**⚠ An unexpected error occurred: {main_error}**")
    else:
        try:
            activities1 = ["Sign in", "Sign up"]
            choice2 = st.sidebar.selectbox("Select an option:", activities1)
```

```
if choice2=='Sign up':
    original_title ='"<p style="font-size: 50px; font-weight: bold;color: #333;
    text-shadow:
    1px 1px 0px #eab,  /* Top-left shadow (light) */
    2px 2px 0px #ccc,  /* Middle-left shadow */
    3px 3px 0px #999;  /* Bottom-left shadow (dark) */
    font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>'"
    st.markdown(original_title, unsafe_allow_html=True)
    st.subheader("Create a New Account")
    with st.form("signup_form"):
        name=st.text_input("**Company_name**")
        email=st.text_input("**Company_email**")
        password=st.text_input("**Company_password**",type="password")
        if st.form_submit_button("Sign Up",type="primary"):
            hsd_pass=hash_password(password)
            cursor.execute(f"SELECT * FROM {DB_table_name1} WHERE Name = %s", (name,))
            company_exists = cursor.fetchone()
            if company_exists:
                st.error(f"**_A company with the name '{name}' already exists. Please use a different name or log in._**")
            else:
                try:
                    new_cid = insert_com_data(name, email, hsd_pass)
                    st.success(f"**_:blue[Your profile is created. Your company ID is {new_cid}]_**")
                    st.balloons()
                except Exception as e:
                    st.error(f"**_An unexpected error occurred: {e}_**")
else:
    if st.session_state.reset_mode:
        original_title ='"<p style="font-size: 50px; font-weight: bold;color: #333;
        text-shadow:
        1px 1px 0px #eab,  /* Top-left shadow (light) */
        2px 2px 0px #ccc,  /* Middle-left shadow */
        3px 3px 0px #999;  /* Bottom-left shadow (dark) */
        font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>'"
        st.markdown(original_title, unsafe_allow_html=True)
        st.subheader("🔁 Reset Password")
        with st.form("reset_form"):
            email = st.text_input("Registered Email")
            new_password = st.text_input("New Password", type="password")
            confirm_password = st.text_input("Confirm Password", type="password")
            reset_btn = st.form_submit_button("Reset Password",type="primary")
            if reset_btn:
                if new_password != confirm_password:
                    st.error("❌ Passwords do not match.")
                else:
                    hashed = hash_password(new_password)
                    reset_password_company(email, hashed)
        if st.button("⬅ Back to Sign In"):
            st.session_state.reset_mode = False
            st.rerun()
        return
    if not st.session_state.get("company_logged_in", False):
        original_title ='"<p style="font-size: 50px; font-weight: bold;color: #333;
        text-shadow:
        1px 1px 0px #eab,  /* Top-left shadow (light) */
        2px 2px 0px #ccc,  /* Middle-left shadow */
        3px 3px 0px #999;  /* Bottom-left shadow (dark) */
        font-family: Times new roman;"><b>SMART HIRING SYSTEM</b></p>'"
        st.markdown(original_title, unsafe_allow_html=True)
        st.subheader("Company Sign In")
        if "company_cid" not in st.session_state:
            st.session_state.company_user = ""
        if "company_password" not in st.session_state:
            st.session_state.company_password = ""
        with st.form("signin_form"):
            st.session_state.company_user = st.text_input("Company ID", value=st.session_state.company_user)
            st.session_state.company_password = st.text_input("Password", type="password", value=st.session_state.company_password)
            if st.form_submit_button("Sign In",type="primary"):
                with st.spinner(':blue[Authenticating...]'):
                    time.sleep(2)
                    company = company_signin(st.session_state.company_user, st.session_state.company_password)
                    if company:
                        st.rerun()
            forgot = st.form_submit_button("Forgot Password?")
            if forgot:
```

```
                st.session_state.reset_mode = True
                st.rerun()
        else:
                pages = {
                "Home": "🏠 Home",
                "Display Shortlist Candidates": "🗓️ Shortlisted Candidates",
                "Scheduled Interview": "🗓️ Scheduled Interview",
                "About":" 💡 About",
                "Logout": "🔐 Logout"
                }
                selected_page = st_navbar(list(pages.values()))
                label_to_key = {v: k for k, v in pages.items()}
                page = label_to_key[selected_page]
                st.toast(f"**:green[Welcome back, {st.session_state['company_name']}! 🎉]**")
                page_bg_img = f"""
                <style>
                [data-testid="stAppViewContainer"] > .main {{
                background-image: url("https://i.postimg.cc/LXgf9ZvP/Untitled-design-43.png");
                background-position: center center;
                background-attachment: fixed;
                background-repeat: no-repeat;
                background-size: 100%;
                }}
                [data-testid="stHeader"] {{
                background: rgba(0,0,0,0);
                }}
                </style>
                """
                st.markdown(page_bg_img, unsafe_allow_html=True)
                if page=="Home":
                    activities = ["Web Development", "Python Development", "Java Development", "Data Scientist", "Full Stack
Development","Android Development"]
                    choice1 = st.selectbox("**Choose Required Domain:**", activities)
                    st.session_state.c1=choice1
                    st.subheader("You Selected: " + choice1)
                    cursor.execute('"SELECT Actual_skills FROM user"')
                    data = cursor.fetchall()
                    cursor.execute('"SELECT Email_ID FROM user"')
                    data2 = cursor.fetchall()
                    if choice1=="Web Development":
                        options = st.multiselect(
                            "Choose the required fields",
                            ["JavaScript", "HTML", "CSS","React","PHP","Node.js","Next.js","Express.js"],
                            ["HTML"],
                        )
                        li=options
                        l=[]
                        for x in data:
                            for y in x:
                                converted_list = eval(y)
                                c=0
                                lowercase_list = [item.lower() for item in converted_list]
                                for i in range(len(li)):
                                    if li[i].lower() in lowercase_list:
                                        c=c+1
                                l.append(c)
                        emails_list = [email[0] for email in data2]
                        result = dict(zip(emails_list, l))
                        for email, value in result.items():
                            delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
                            cursor.execute(delete_sql, (value,email,))
                            connection.commit()
                        age = st.slider("**select required experience year?**", 0, 40, 5)
                        st.subheader("**Experience level set to: **"+ str(age) + "** years**")
                        job_deadline = st.date_input("Application Deadline", min_value=datetime.date.today())
                        des=st.text_area("Description","Write here something")
                    elif choice1=="Python Development":
                        options = st.multiselect(
                            "Choose the required fields",
                            ["Python", "Django", "Flask","Tkinter","CherryPy","WEB2PY","FastAPI","TensorFlow"],
                            ["Python"],
                        )
                        li=options
                        l=[]
```

```
    for x in data:
        for y in x:
            converted_list = eval(y)
            c=0
            lowercase_list = [item.lower() for item in converted_list]
            for i in range(len(li)):
                if li[i].lower() in lowercase_list:
                    c=c+1
            l.append(c)
    emails_list = [email[0] for email in data2]
    result = dict(zip(emails_list, l))
    for email, value in result.items():
        delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
        cursor.execute(delete_sql, (value,email,))
        connection.commit()
    age = st.slider("**select required experience year?**", 0, 40, 5)
    st.subheader("**Experience level set to: **"+ str(age) + "** years**")
    job_deadline = st.date_input("Application Deadline",min_value=datetime.date.today())
    des=st.text_area("Description","Write here something")
elif choice1=="Java Development":
    options = st.multiselect(
        "Choose the required fields",
        ["java", "JSP", "Servlet","Spring boot","JavaScript","angular"],
        ["java"],
    )
    li=options
    l=[]
    for x in data:
        for y in x:
            converted_list = eval(y)
            c=0
            lowercase_list = [item.lower() for item in converted_list]
            for i in range(len(li)):
                if li[i].lower() in lowercase_list:
                    c=c+1
            l.append(c)
    emails_list = [email[0] for email in data2]
    result = dict(zip(emails_list, l))
    for email, value in result.items():
        delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
        cursor.execute(delete_sql, (value,email,))
        connection.commit()
    age = st.slider("**select required experience year?**", 0, 40, 5)
    st.subheader("**Experience level set to: **"+ str(age) + "** years**")
    job_deadline = st.date_input("Application Deadline",min_value=datetime.date.today())
    des=st.text_area("Description","Write here something")
elif choice1=="Data Scientist":
    options = st.multiselect(
        "Choose the required fields",
        ["Mechine Learning", "Python", "AI","NLP","Deep Learning","Pandas","TensorFlow","Power BI","Pytorch","Excel"],
        ["AI"],
    )
    li=options
    l=[]
    for x in data:
        for y in x:
            converted_list = eval(y)
            c=0
            lowercase_list = [item.lower() for item in converted_list]
            for i in range(len(li)):
                if li[i].lower() in lowercase_list:
                    c=c+1
            l.append(c)
    emails_list = [email[0] for email in data2]
    result = dict(zip(emails_list, l))
    for email, value in result.items():
        delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
        cursor.execute(delete_sql, (value,email,))
        connection.commit()
    age = st.slider("**select required experience year?**", 0, 40, 5)
    st.subheader("**Experience level set to: **"+ str(age) + "** years**")
    job_deadline = st.date_input("Application Deadline",min_value=datetime.date.today())
    des=st.text_area("Description","Write here something")
elif choice1=="Full Stack Development":
    options = st.multiselect(
```

```
                "Choose the required fields",
                ["Python", "Java", "R", "Ruby", "Node.js", "PHP", "React", "Angular",
"Express.js","C++","MongoDB","MySQL","PostgreSQL"],
                ["PHP"],
            )
            li=options
            l=[]
            for x in data:
                for y in x:
                    converted_list = eval(y)
                    c=0
                    lowercase_list = [item.lower() for item in converted_list]
                    for i in range(len(li)):
                        if li[i].lower() in lowercase_list:
                            c=c+1
                    l.append(c)
            emails_list = [email[0] for email in data2]
            result = dict(zip(emails_list, l))
            for email, value in result.items():
                delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
                cursor.execute(delete_sql, (value,email,))
                connection.commit()
            age = st.slider("**select required experience year?**", 0, 40, 5)
            st.subheader("**Experience level set to: **"+ str(age) + "** years**")
            job_deadline = st.date_input("Application Deadline",min_value=datetime.date.today())
            des=st.text_area("Description","Write here something")
        elif choice1=="Android Development":
            options = st.multiselect(
                "Choose the required fields",
                ["Java","Kotlin","Android UI","C++","Python"],
                ["Java"],
            )
            li=options
            l=[]
            for x in data:
                for y in x:
                    converted_list = eval(y)
                    c=0
                    lowercase_list = [item.lower() for item in converted_list]
                    for i in range(len(li)):
                        if li[i].lower() in lowercase_list:
                            c=c+1
                    l.append(c)

            emails_list = [email[0] for email in data2]
            result = dict(zip(emails_list, l))
            for email, value in result.items():
                delete_sql = f"UPDATE {DB_table_name} SET Count = %s WHERE Email_ID = %s"
                cursor.execute(delete_sql, (value,email,))
                connection.commit()
            age = st.slider("**select required experience year?**", 0, 40, 5)
            st.subheader("**Experience level set to: **"+ str(age) + "** years**")
            job_deadline = st.date_input("Application Deadline",min_value=datetime.date.today())
            des=st.text_area("Description","Write here something")
        if "load1_state" not in st.session_state:
            st.session_state.load1_state = False  # For "Submit new recruitment posts"
        if "load12_state" not in st.session_state:
            st.session_state.load12_state = False  # For "View Previous Recruitment Posts"
        if "load120_state" not in st.session_state:
            st.session_state.load120_state = False
        load1 = st.button('Submit new recruitment posts',type="primary")
        if load1:
            st.session_state.load12_state = False
            st.session_state.load1_state = True  # Set flag to indicate button click
        if st.session_state.load1_state:
            ts = time.time()
            cur_date = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d')
            cur_time = datetime.datetime.fromtimestamp(ts).strftime('%H:%M:%S')
            timestamp = str(cur_date + '_' + cur_time)
            jid=insert_recruit_data(st.session_state.company_user, choice1, ', '.join(options), timestamp, age, job_deadline,des)
            st.success(f"**New recruitment post for {choice1} submitted sucessfully**")
            st.success(f"**Post ID is: {jid}**")
            st.session_state.load1_state = False
        load12 = st.button("View Previous Recruitment Posts")
```

```
                    if load12:
                        st.session_state.load1_state = False
                        st.session_state.load12_state = True  # Set flag for this button
                    if st.session_state.load12_state:
                        previous_posts = fetch_previous_recruitments(st.session_state.company_user)
                        if previous_posts:
                            st.header("**Previous Recruitment Posts**")
                            df = pd.DataFrame(previous_posts, columns=['RID', 'Domain', 'Recommended Skills', 'Timestamp',
'Experience','Deadline','Job Description'])
                            st.dataframe(df)
                        else:
                            st.info("No previous recruitment posts found.")
                elif page=="Display Shortlist Candidates":
                    st.subheader("Select a Job Posting to View Candidates")
                    cursor.execute("SELECT rid, domain FROM recruit_data WHERE cid = %s", (st.session_state.company_user,))
                    job_posts = cursor.fetchall()
                    if job_posts:
                        job_options = {f"{domain} (ID: {rid})": rid for rid, domain in job_posts}
                        selected_label = st.selectbox("Select a Job Posting", options=list(job_options.keys()))
                        selected_rid = job_options[selected_label] # Get rid from selected label
                        load120 = st.button("Show Candidates for Selected Job",type="primary")
                        if load120:
                            st.session_state.load12_state = False  # Reset previous button states
                            st.session_state.load120_state = True  # Set flag for this button
                        if st.session_state.get("load120_state", False):
                            cursor.execute(f"""
                                SELECT candidate_id, Name, Email_ID, Experience, Actual_skills, Resume
                                FROM user
                                WHERE rid = %s AND Count >= 1 AND Experience >= (SELECT experience FROM recruit_data WHERE rid = %s)
                                ORDER BY Count DESC
                            """, (selected_rid, selected_rid))
                            data10 = cursor.fetchall()
                            cursor.execute(f"""
                                SELECT candidate_id, Name, Email_ID, Experience, Actual_skills
                                FROM user
                                WHERE rid = %s AND Count >= 1 AND Experience >= (SELECT experience FROM recruit_data WHERE rid = %s)
                                ORDER BY Count DESC
                            """, (selected_rid, selected_rid))
                            data_10 = cursor.fetchall()
                            if data10:
                                df2 = pd.DataFrame(data_10, columns=['Candidate_ID', 'Name', 'Email_ID', 'Experience', 'Actual Skills'])
                                st.dataframe(df2)
                                df1 = pd.DataFrame(data10, columns=['Candidate_ID', 'Name', 'Email_ID', 'Experience', 'Actual Skills','Resume'])
                                st.write("## Shortlist Candidates")
                                for index, row in df1.iterrows():
                                    cols = st.columns([1, 2, 2, 2])  # Adjust column width ratio as needed
                                    cols[0].write(row["Candidate_ID"])
                                    cols[1].write(row["Name"])
                                    if row["Resume"]:
                                        cols[2].download_button(
                                            label="Download Resume",
                                            data=row["Resume"],  # Resume binary data
                                            file_name=f"{row['Name']}_resume.pdf",
                                            mime="application/pdf"
                                        )
                                    with cols[3]:
                                            accept_key = f"accept_state_{row['Candidate_ID']}"
                                            accept = st.button(f"Accept {row['Name']}", type="primary", key=f"interview_{row['Candidate_ID']}")
                                            if accept:
                                                st.session_state[accept_key] = True
                                            if st.session_state.get(accept_key, False):
                                                cursor.execute(f"SELECT Name FROM {DB_table_name1} WHERE cid = %s",
(st.session_state.company_user,))
                                                r = cursor.fetchone()
                                                cursor.execute("UPDATE user SET status = 'Shortlisted' WHERE candidate_id = %s AND rid = %s",
(row["Candidate_ID"], selected_rid))
                                                connection.commit()
                                                today = datetime.date.today()
                                                company_id = st.session_state.company_user
                                                company_name = st.session_state.get("company_name", "Your Company")
                                                cursor.execute("SELECT email FROM user_info WHERE candidate_id = %s", (row["Candidate_ID"],))
                                                candidate_email = cursor.fetchone()[0]
                                                cursor.execute("SELECT domain FROM recruit_data WHERE rid = %s", (selected_rid,))
                                                domain = cursor.fetchone()[0]
```

```
                                        interview_date = st.date_input("Interview Date", min_value=today,
key=f"interview_date_{row['Candidate_ID']}")
                                        interview_time = st.time_input("Interview Time",key=f"interview_time_{row['Candidate_ID']}")
                                        jitsi_link = generate_jitsi_link()
                                        if st.button("✅ Schedule Interview",type="primary",key=f"interview_set_{row['Candidate_ID']}"):
                                            cursor.execute("""
                                               INSERT INTO interview_schedule (company_id, Candidate_ID, rid, domain, interview_date,
interview_time, jitsi_link)
                                               VALUES (%s, %s, %s, %s, %s, %s, %s)
                                            """, (company_id, row["Candidate_ID"], selected_rid, domain, interview_date, interview_time, jitsi_link))
                                            connection.commit()
                                            subject = f"{company_name} - Virtual Interview Invitation"
                                            message = f"""Dear Candidate,
Thank you for applying to the role at {company_name}. We are pleased to inform you that you have been shortlisted for the next stage of our hiring
process.

Interview Details:
📅 Date: {interview_date}
🕒 Time: {interview_time}
🔗 Link: Please join via your Smart Hiring System profile.

NOTE:
1. Please ensure you have a stable internet connection if the interview is virtual.
2. Keep a copy of your resume and any relevant documents.

We look forward to speaking with you and learning more about your qualifications.

Regards,
{company_name} Recruitment Team"""

                                            send_email(candidate_email, subject, message)
                                            st.success("✅ Interview scheduled successfully.")
                                            st.markdown(f"🔗 **[Join Jitsi Meeting]({jitsi_link})**")

                                else:
                                    st.info("No candidates have applied for this job yet.")
                            else:
                                st.info("No job post available yet.")
                        elif page=="Scheduled Interview":
                            com_get_meeting_link()


                        elif page == "About":
                            st.header("About Smart Hiring System")
                            st.write("Welcome to the Smart Hiring System, a cutting-edge recruitment platform designed to connect talented candidates with
top-tier organizations. Our platform streamlines the hiring process by offering an intuitive interface for job seekers, recruiters, and administrators
alike.")
                            st.subheader("Key Features:")
                            columns = st.columns(3)
                            # Add unique content in each column using containers
                            with columns[0]:
                                with st.container(height=250):
                                    st.write("**For Candidates:**")
                                    st.write("Effortlessly apply for jobs, upload resumes, and track your applications. Get instant notifications about interview
schedules and stay ahead in your job search.")


                            with columns[1]:
                                with st.container(height=250):
                                    st.write("**For Companies:**")
                                    st.write(" Post job openings, review candidate applications, and schedule interviews seamlessly. Utilize intelligent filters to
shortlist the best talent for your organization.")
                            with columns[2]:
                                with st.container(height=250):
                                    st.write("**For Admins:**")
                                    st.write("Oversee platform activities, manage user data, and ensure smooth operations across all stakeholders.")
                        elif page=="Logout":
                            st.session_state.clear()
                            st.rerun()
        except Exception as main_error:
            st.error(f"**⚠ An unexpected error occurred: {main_error}**")
run()
```

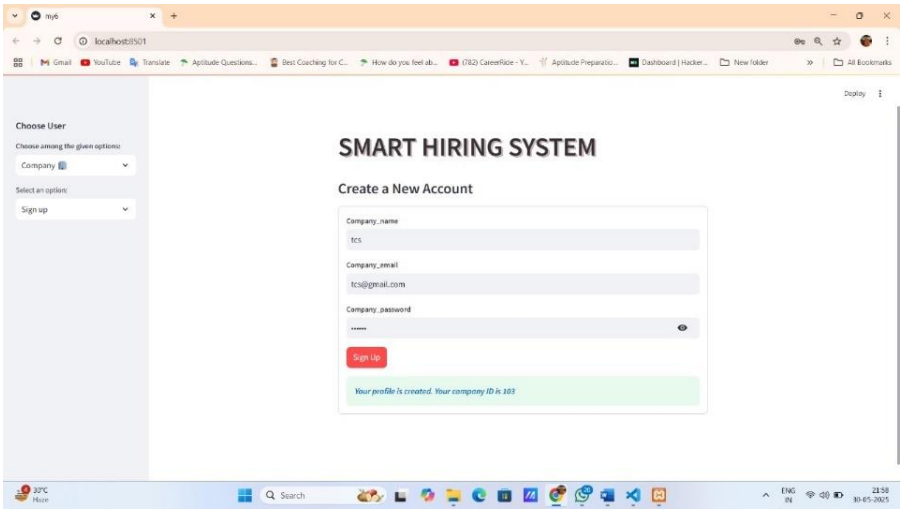
ADVANTAGES OF THE SYSTEM:

- **Efficiency**: Automates time-consuming tasks such as resume screening, and candidate ranking. Streamlines the entire hiring process, from job posting to interview scheduling, ensuring timely and organized recruitment.
- **Accuracy**: Reduces human errors in candidate evaluation through algorithm-based matching.
- **User-Friendly**: Offers a frictionless experience for applicants, recruiters, and administrators.
- **Skill-Based Shortlisting**: Uses count-based matching algorithms to rank candidates based on relevant skills and experience.
- **Transparent Communication**: Notifies users at every stage of their application with autogenerated emails and downloadable application reference PDF.
- **Role-Based Interfaces**: Distinct portals for users, companies, and administrators enhance usability and control.
- **Integrated Interview Scheduling**: Embeds Jitsi links and schedules interviews directly within the app.
- **Scalability**: Handles growing user and data volumes efficiently.
- **Security**: Incorporates secure data handling and compliance with privacy regulations.
- **Cost-Effective:** Offers an affordable hiring solution which is beneficial for startups and small businesses.
- **Customizable Job Criteria**: Hiring managers can set requirements very easily and fast by using pre-define components for specific skills, experience levels, and other requirements.

LIMITATIONS OF SYSTEM:

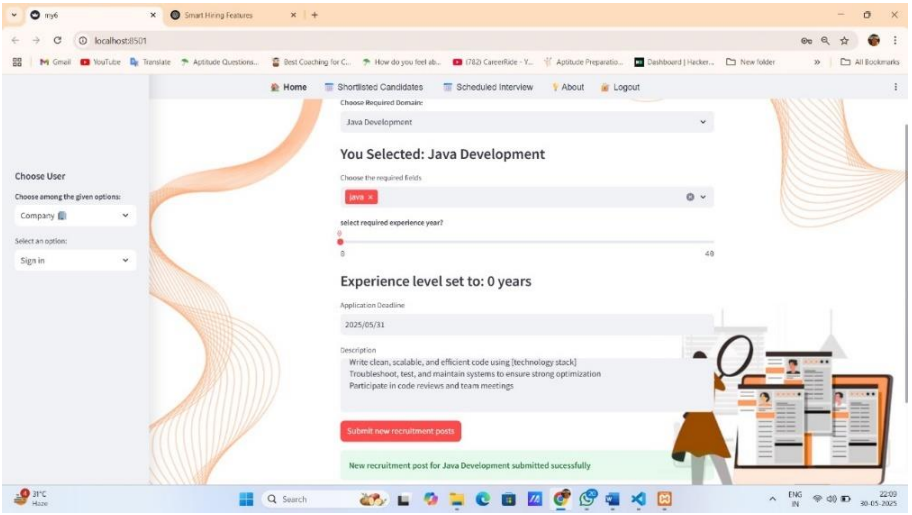
- **Qualitative Assessment**: Difficulty in evaluating non-quantifiable attributes like cultural fit or soft skills.
- **Dependency on Internet**: Requires continuous internet connectivity for real-time functionalities.
- **Resume Parsing Errors**: Issues with extracting information accurately from resumes with non-standard formats.
- **Keyword-Based Matching**: The resume shortlisting process relies on keywords for skill matching, which may overlook candidates with relevant but differently phrased skills.
- **Limited Interview Flexibility**: The system integrates Jitsi for virtual interviews, which may lack advanced features compared to premium video conferencing tools.
- **Limited Customization**: While the system supports customizable job criteria but it is rely on few dynamic requirements and may not cater to all unique requirements or workflows of different companies.

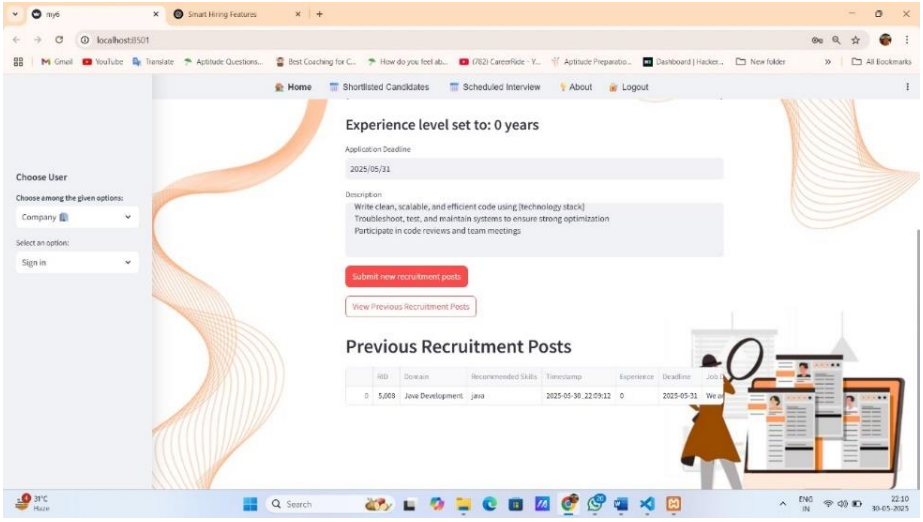
SAMPLE OUTPUT



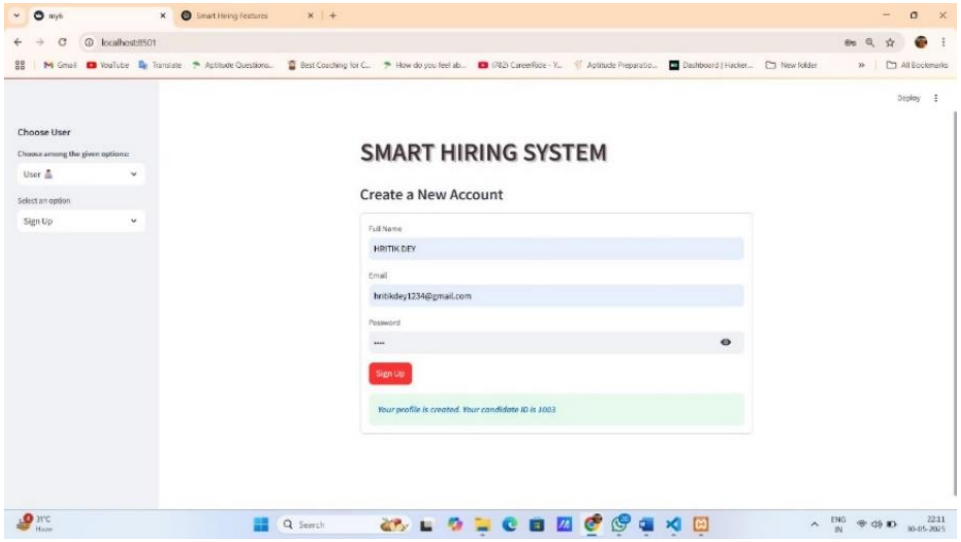
: Register Company

Set Recruitment Specification:



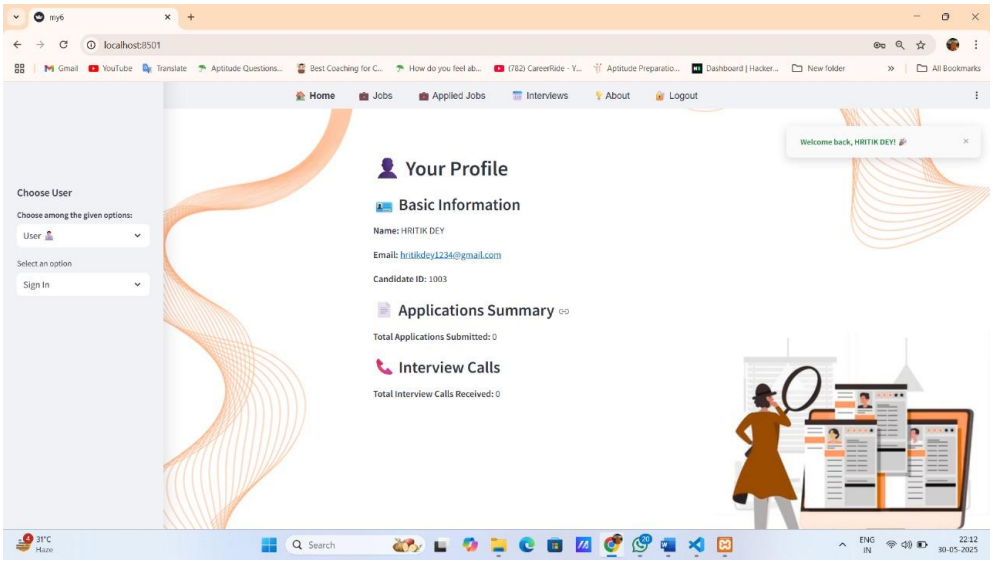


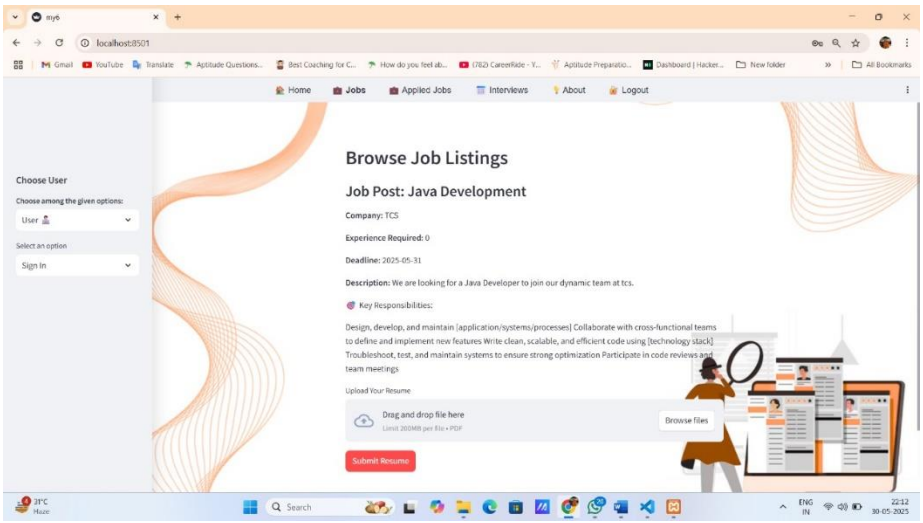
: View Previous Recruitment Post



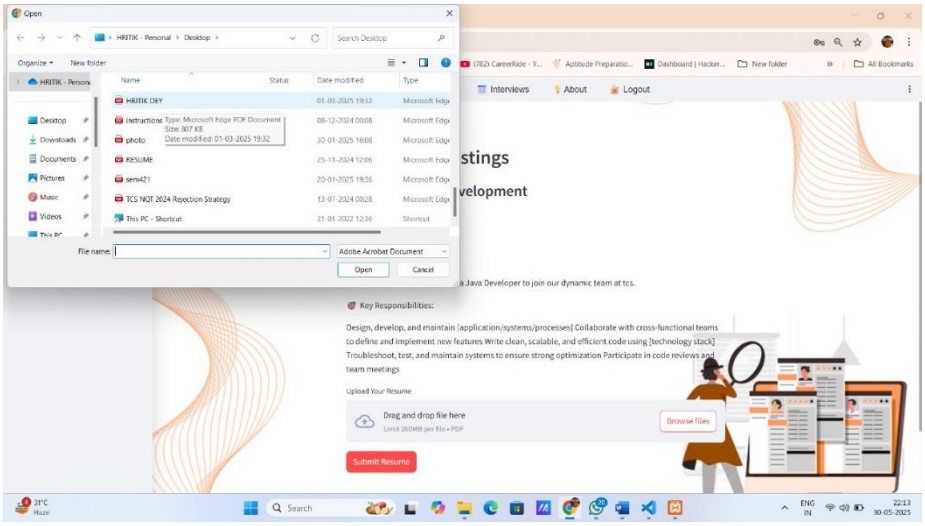
: Create Account by User

User Profile:



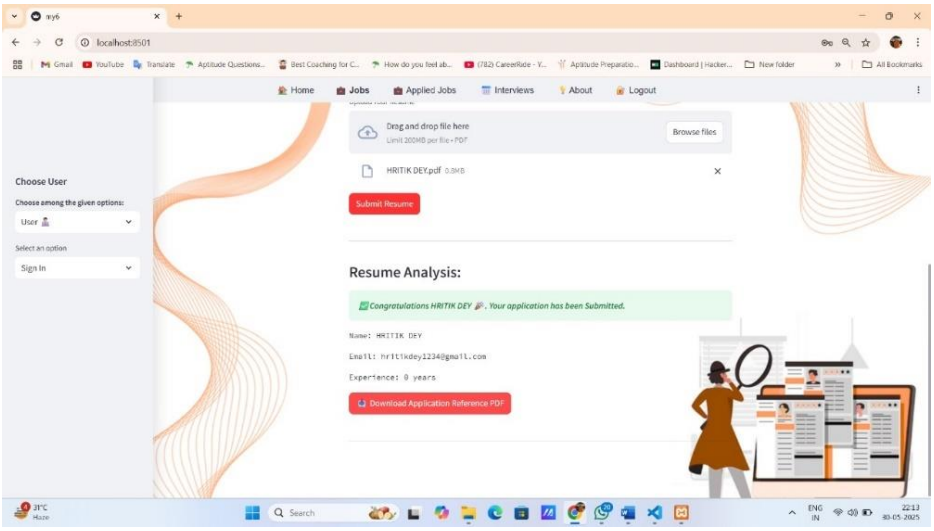


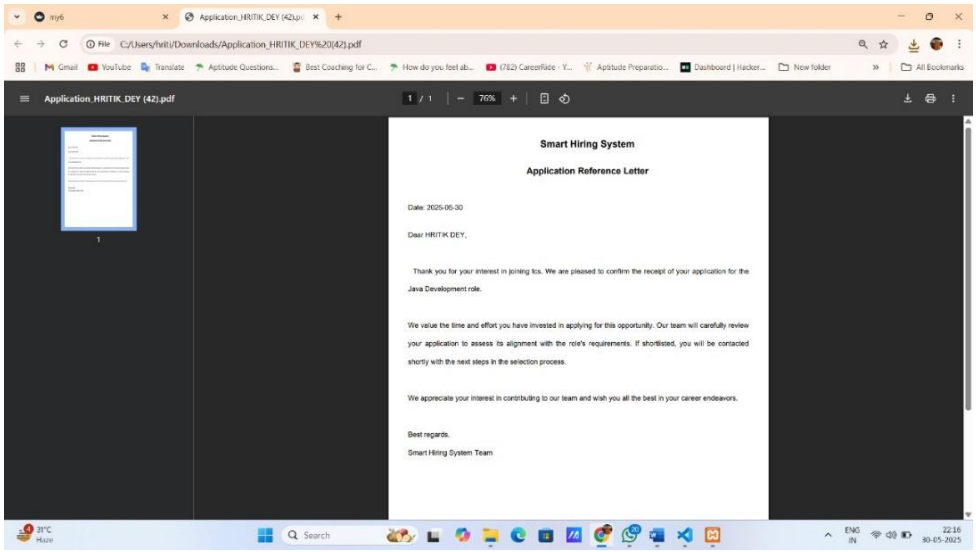
: Job Openings



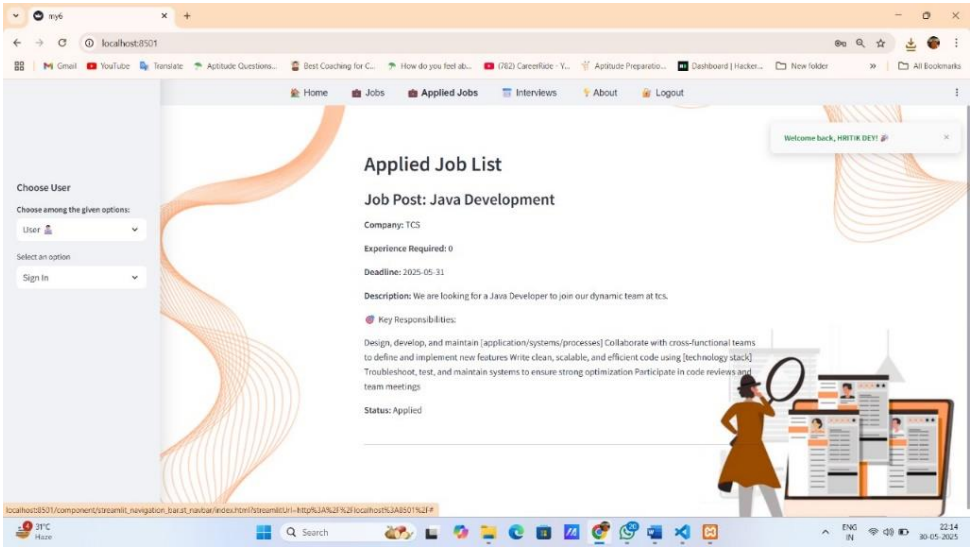
: Resume Submission

Resume Submission Confirmation:



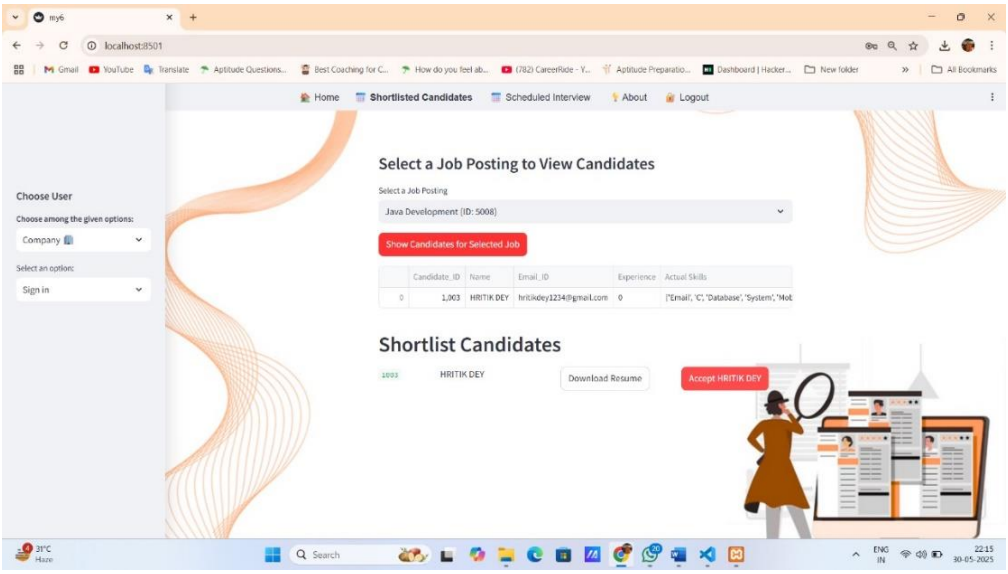


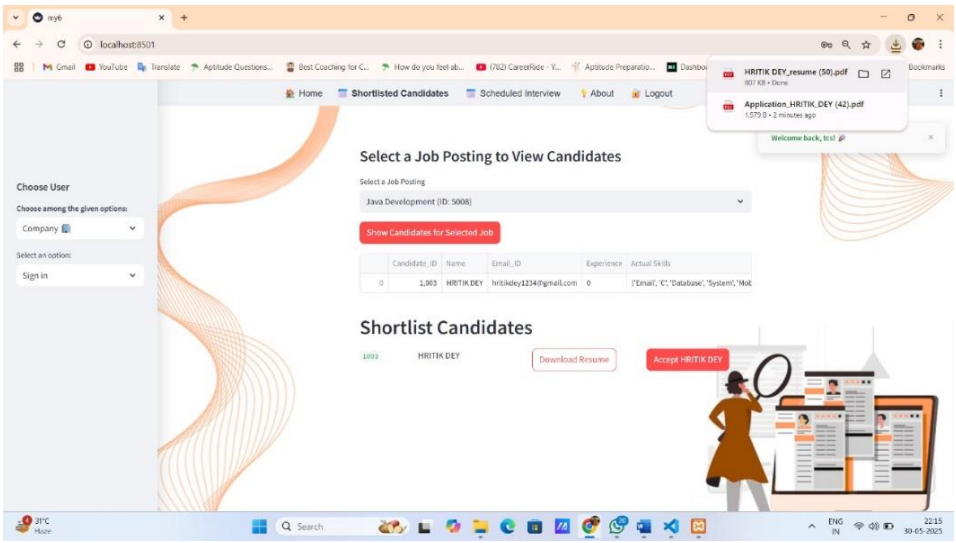
: Application Reference PDF



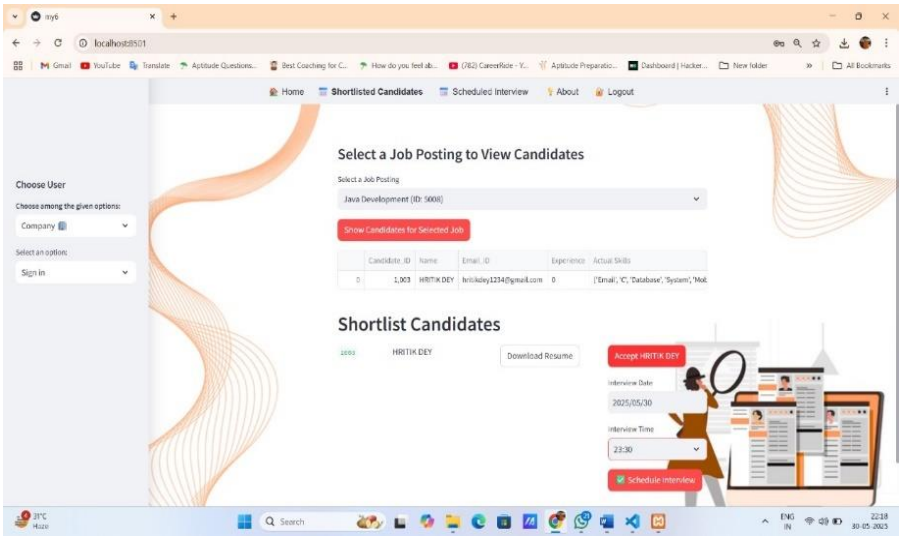
: Applied Job List

Shortlisted Candidate's Data in Company side:



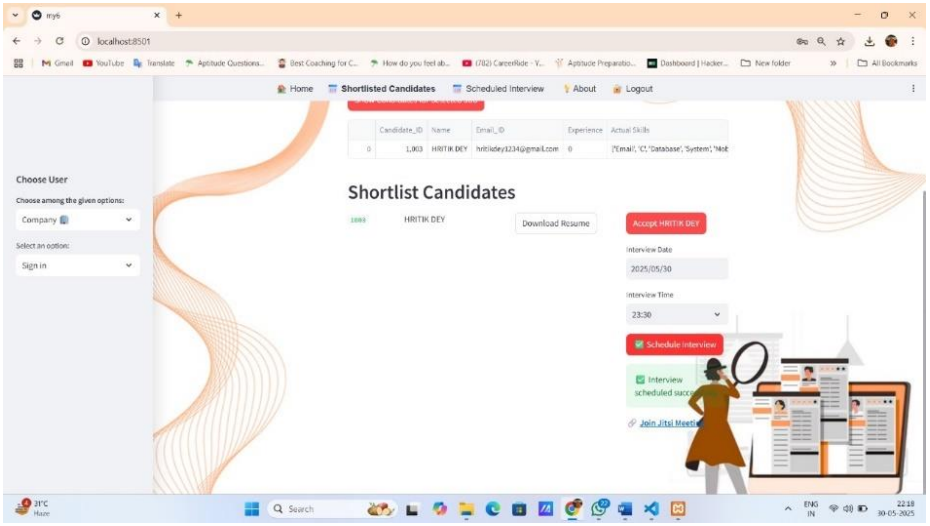


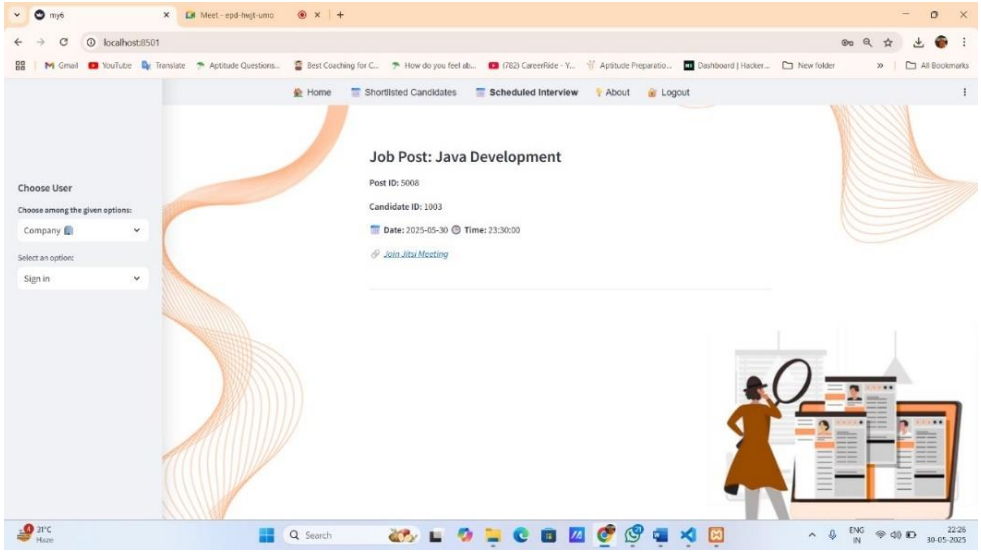
: Download Candidate's Resume



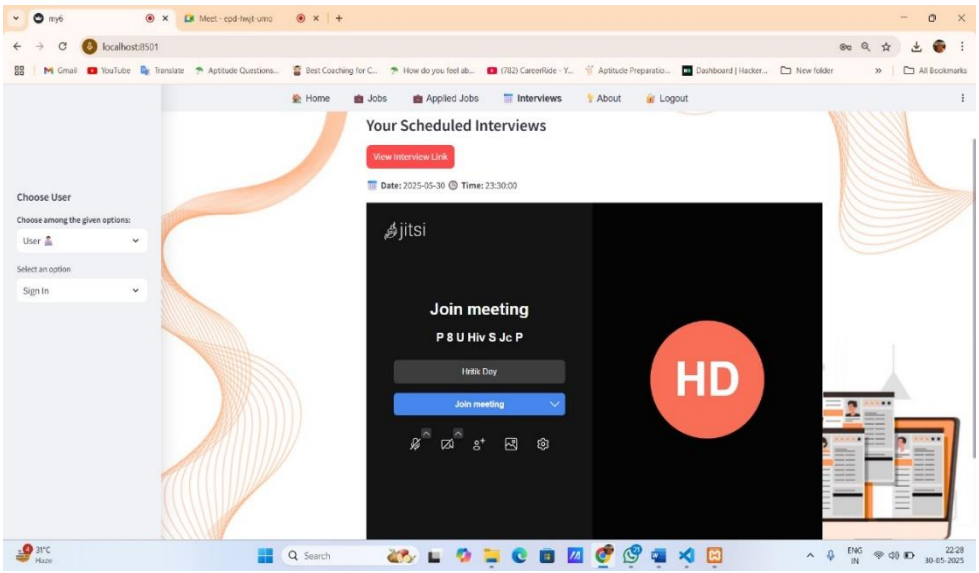
: Accept & Schedule Interview

Confirmation message with an interview meeting link:



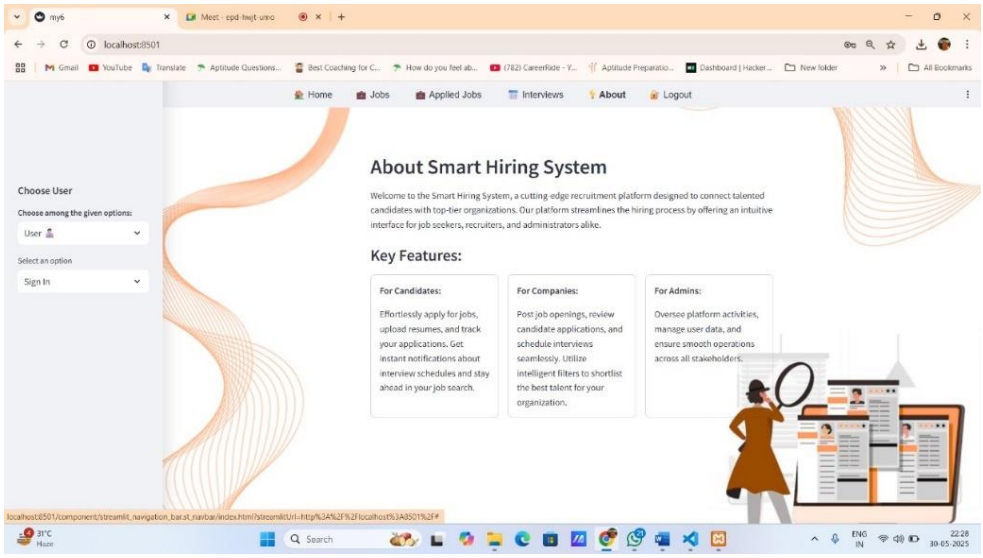


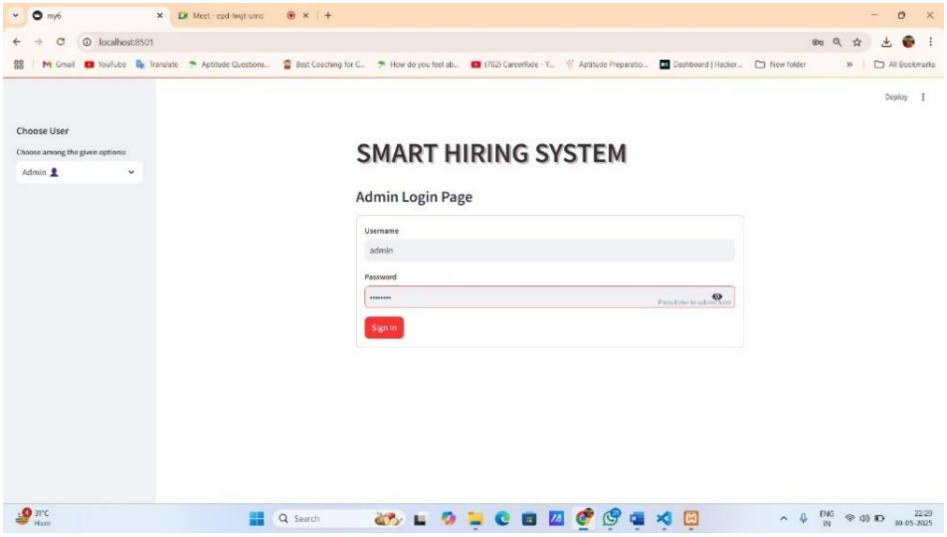
: Get Meeting Link at company side



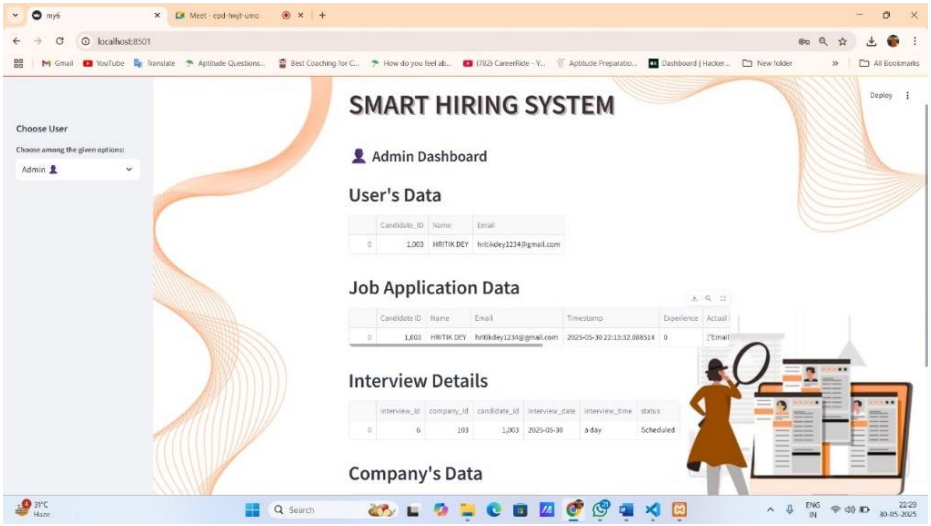
: Join Meeting from applicant's side

About Our Application:



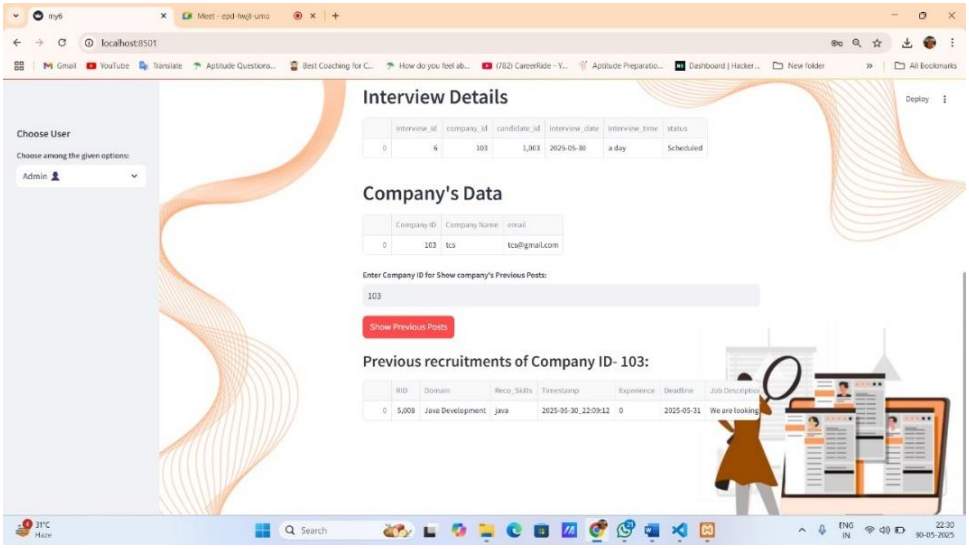


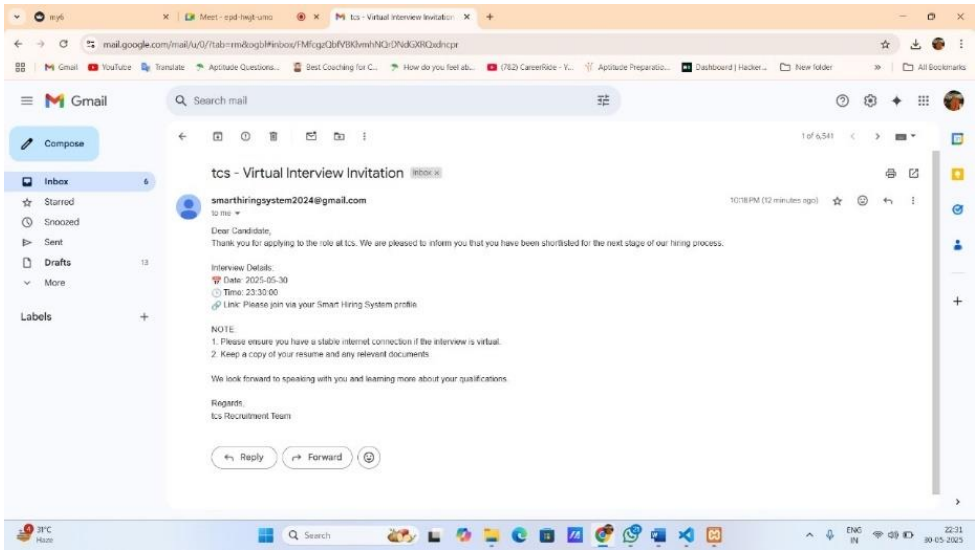
: Admin Login Page



: View all Details

Check Previous Recruitments by a Company:





: Interview Mail

CONCLUSION

The Smart Hiring System revolutionizes recruitment by offering an intelligent, automated, and user-friendly platform that streamlines the hiring process for candidates, companies, and administrators. Built with technologies like Python, Streamlit, NLP, and MySQL, it efficiently handles tasks such as resume parsing, skill-based shortlisting, and interview scheduling. Its integration of real-time tracking and virtual interviews ensures transparency and ease of use. While it faces limitations related to resume formatting and cloud scalability, it remains a cost-effective alternative to traditional hiring methods. The system emphasizes fairness by evaluating candidates based on skills rather than bias-prone filters. Its modular architecture allows for continuous enhancements and scalability. By minimizing manual effort, it significantly reduces recruitment time and administrative burden. The platform supports seamless interaction and fosters collaboration between stakeholders. Overall, the Smart Hiring System blends innovation with accessibility to modernize the hiring process. It stands as a reliable, adaptable solution for today's evolving recruitment needs.

REFERENCES

1. Project - How to build a Resume Parser using Python - GeeksforGeeks
2. GitHub - OmkarPathak/pyresparser: A simple resume parser used for extracting information from resumes
3. Streamlit documentation
4. #73 Python Database Connection | MySQL
5. Sending EMAILS with PYTHON: 5 minute tutorial
6. Download PDF option - Using Streamlit - Streamlit
7. Jitsi URL Generator
8. Architecture | Jitsi Meet
9. GitHub - jitsi/jitsi-meet: Jitsi Meet - Secure, Simple and Scalable Video Conferences that you use as a standalone app or embed in your web application.
10. https://en.wikipedia.org/wiki/R%C3%A9sum%C3%A9_parsing
11. https://pmc.ncbi.nlm.nih.gov/articles/PMC9516509

Video links:

1. **NLP Project** (Introduction) : Building AI Resume App
2. Resume Parser Using Python | Extract Data from Resume Python | Satyajit Pattnaik
3. How to Deploy Your App to Streamlit Community Cloud
4. Deploy Streamlit Python Application for FREE
5. STREAMLIT TRICKS - Web App RERUNS on every WIDGET CLICKS ? Here's WHAT-TO-DO |SessionState| PYTHON
6. How to make NESTED buttons in Streamlit with Session State