PROJECT REPORT

Employee Recommendation System

PROFESSOR

Dr Arif Ur Rahman

SUBJECT

Natural Language Processing

SUBMITTED BY

Muhammad Arslan Shaukat
01-134211-053
Muhammad Moeez
01-134211-104

Department of Computer Science,
Bahria University, Islamabad.

19-05-2024

Table of Contents

Abstract

The proposed system utilizes modern NLP techniques to extract key information from resumes, including skills, experience, and qualifications. By employing advanced keyword extraction and similarity algorithms, the system identifies relevant terms and concepts associated with specific job positions.

The core functionality of the system involves the input of a desired job position, such as "web developer," and the subsequent retrieval of resumes that closely match the specified criteria. To achieve this, the system parses through a corpus of resumes, extracting pertinent information and evaluating the degree of relevance to the target position.

Key features of the system include:

1. **Keyword Matching:** Employing a comprehensive library of industry-specific keywords and phrases to identify relevant resumes.
2. **Similarity:** Utilizing similarity techniques to gauge the matching of resumes.
3. **Regular Expression:** For Processing the cv data
4. **User Interface:** Providing an intuitive interface for users to input job positions and review retrieved resumes, facilitating efficient recruitment processes.

The system's effectiveness is evaluated through rigorous testing on CVs, encompassing a wide range of job positions and industries.

1. Introduction

The recruitment process is a critical aspect of organizational operations, significantly impacting the efficiency and success of businesses. However, traditional recruitment methods often suffer from inefficiencies and limitations, primarily stemming from manual screening processes. These processes are time-consuming, prone to biases, and may overlook potentially suitable candidates.

In response to these challenges, the Employee Recommendation System aims to revolutionize the recruitment process by automating candidate screening and matching. By harnessing the power of NLP techniques and machine learning algorithms, the system offers a data-driven approach to identify the most suitable candidates for specific job roles. This project holds immense potential for organizations seeking to optimize their recruitment processes and secure top talent efficiently.

2. Objective

The primary objective of this project is to design and develop an Employee Recommendation System that can accurately match candidates with job openings based on their skills, experience, and education.

Specifically, the system aims to:

- Automate the candidate screening process to reduce manual effort and time spent by recruiters.
- Improve the accuracy and efficiency of candidate-job matching by leveraging NLP techniques.
- Enhance the overall recruitment experience for both recruiters and candidates.

3. Problem Statement

The traditional recruitment process is plagued by inefficiencies, including time-consuming manual screening, subjective candidate evaluation, and missed opportunities due to limited candidate visibility. These inefficiencies not only hinder the recruitment process but also result in increased costs and delays in filling crucial job vacancies.

The Employee Recommendation System addresses these challenges by offering an automated solution that efficiently identifies and matches suitable candidates with job openings. By leveraging advanced NLP techniques and machine learning algorithms, the system streamlines the recruitment process, mitigating biases and ensuring a more objective candidate evaluation.

4. Methodology

The methodology employed in this project encompasses various NLP techniques and machine learning algorithms to achieve the desired objectives.

Key components of the methodology include:

- **Data Preprocessing:** Raw resume data is preprocessed to extract relevant information, including skills, experience, and education, using techniques such as Regular Expression
- **Feature Extraction:** Regular expressions are used to extract dates and calculate tenure duration, Regex is also used for processing and removing unnecessary numerical values from the text.
- **Storage:** Each section is then stored in corresponding column of the candidate table in database.
- **Query:** User is able to enter a query from the frontend, query is in format [[Education: Bahria University],[Experience: XYZ],[Skills: python]]. The query is then processed and broken down in individual header like Education, Experience, skills etc.
- **Similarity Calculation:** Each header is converted to numerical vector. Records are retrieved from the database and relevant headers are converted to numerical vectors. Cosine Similarity is calculated between query and retrieved record vector for each column and are added up and stored in a list.
- **Retrieval:** ID for record with highest similarity is used to retrieve the pdf Url which is sent over to the frontend.

5. Project Scope

The scope of this project includes the development of the Employee Recommendation System, encompassing data preprocessing using Regular expression, storing and retrieving data from database and calculating similarity. The development of a user interface, focusing solely on the core functionality of candidate-job matching.

6. Implementation and Technology Stacks

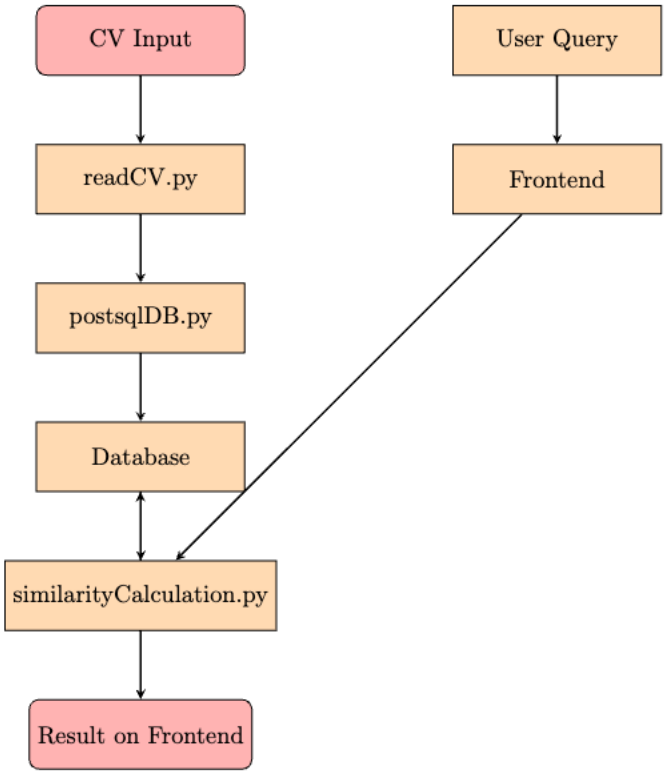
The implementation of the project will be carried out using Python programming language, leveraging libraries such as tempfile, flask, PyDPF2,RE, and scikit-learn for NLP tasks. Additionally it will use datetime library for further assistance.

The choice of these technologies is driven by their widespread adoption in the NLP and machine learning community, as well as their robustness and flexibility in handling diverse tasks.

For Frontend we have used Reactjs. User can enter the query on the webpage and the resultant pdf is displayed on the webpage.

We have used PostgreSQL as database for this project because of it ability to handle structured document effectively.

7. Workflow Diagram



8. Code

8.1 ReadCV.py

Reading PDF

This function reads the content of a PDF file and extracts text from it using the PyPDF2 library.

```
def extract_text_from_pdf(pdf_path):
    text = ""
    with open(pdf_path, 'rb') as file:
```

```
        reader = PyPDF2.PdfReader(file)
        num_pages = len(reader.pages)
        for page_number in range(num_pages):
            page = reader.pages[page_number]
            text += page.extract_text()
    return text
```

REGULAR EXPRESSION

This function uses regular expressions to find specific sections like Education, Experience, Skills, Projects, and Interests in the resume text.

```
def extract_sections(text):
    # Define regular expressions for each section header
    section_patterns = {
        "Education": r"(?i)education(?=.*BS)(.*?)(?=Experience)",
        "Experience": r"(?i)experience(.*?)(?=Skills)",
        "Skills": r"(?i)skills(.*?)(?=Projects)",
        "Projects": r"(?i)projects(.*?)(?=Interests)",
        "Interests and Extracurriculars": r"(?i)interests and extracurriculars(.*)"
        # Add more section patterns as needed
    }
    sections = {}

    for section, pattern in section_patterns.items():
        matches = re.search(pattern, text, re.IGNORECASE | re.DOTALL)
        if matches:
            sections[section] = matches.group(1).strip()

    return sections
```

SPLITING RESUME SECTIONS

This function splits the text of each section into individual items based on different criteria depending on the section type.

```
ef process_section(section_text, section_type):
    # Split the text based on different criteria depending on the section type
    if section_type == "Education":
        items = [item.strip() for item in section_text.split('. ') if item.strip()]
    elif section_type == "Experience":
        items = [item.strip() for item in section_text.split('.') if item.strip()]
    elif section_type == "Skills":
```

```
        items = [item.strip() for item in section_text.split(',') if item.strip()]
    elif section_type in ["Projects", "Interests and Extracurriculars"]:
        items = []
        for item in section_text.split('•'):
            item = item.strip()
            if item:
                items.append(item)
    else:
        items = [item.strip() for item in section_text.split('\n') if item.strip()]

    return items
```

EXTRACTING DATES

This function extracts date ranges from the text sets using regular expressions.

```
def extract_dates_with_regex(text_set):
    # Define the regular expression pattern to match date ranges
    date_pattern = r"(\d{1,2})/(\d{4}) - (\d{1,2})/(\d{4})|\d{1,2}/(\d{2}) -
present|(\d{1,2})/(\d{4}) - Present"

    extracted_dates = []

    # Get the current month and year
    current_month = datetime.now().strftime("%m")
    current_year = datetime.now().strftime("%Y")

    # Iterate over each text in the set
    for text in text_set:
        # Use findall to search for date ranges
        matches = re.findall(date_pattern, text, re.IGNORECASE)

        # Process each match
        for match in matches:
            # Check if the match contains "Present"
            if match[3].lower() == "present":
                # If the second date is "Present", replace it with the current month
and year
                date_range = f"{match[0]}/{match[1]} - {current_month}/{current_year}"
                extracted_dates.append(date_range)
            elif match[5]:  # If match contains MM/YY - present format
                start_month = match[5]
                start_year = match[6]
                date_range = f"{start_month}/{start_year} -
{current_month}/{current_year}"
```

```
                extracted_dates.append(date_range)
            else:
                # Otherwise, add the date range as it is
                date_range = f"{match[0]}/{match[1]} - {match[2]}/{match[3]}"
                extracted_dates.append(date_range)

    return extracted_dates
```

REMOVING DATES

This function removes date patterns from text sets using regular expressions.

```
def remove_dates_with_regex(text_set):
    # Define the regular expression pattern to match date ranges
    date_pattern = r"(\d{1,2})/(\d{4}) – (\d{1,2})/(\d{4})|\d{1,2}/(\d{2}) –
present|(\d{1,2})/(\d{4}) – Present"

    # Initialize an empty list to store the text without dates
    text_list_without_dates = []

    # Iterate over each text in the set
    for text in text_set:
        # Use sub to replace date patterns with an empty string
        text_without_dates = re.sub(date_pattern, "", text, flags=re.IGNORECASE)

        # Append the modified text to the list
        text_list_without_dates.append(text_without_dates.strip())

    return text_list_without_dates
```

TENURE CALCULATIONS

This function calculates tenure durations in months based on the extracted date ranges.

```
def calculate_tenure_durations(date_ranges):
    tenure_durations = []
    for date_range in date_ranges:
        start_date, end_date = date_range.split(" - ")
        start_month, start_year = map(int, start_date.split("/"))
        end_month, end_year = map(int, end_date.split("/"))

        # Calculate tenure in months
        if start_year == end_year:
            duration = end_month - start_month + 1
        else:
            duration = (end_year - start_year) * 12 + (end_month - start_month) + 1
        tenure_durations.append(duration)

    return tenure_durations
```

8.2 PostSQLIDB.py

Following function takes the file path of a PDF document as input, reads the binary data of the PDF file, and returns it.

```
def read_pdf_file(pdf_path):
    with open(pdf_path, 'rb') as file:
        pdf_data = file.read()
    return pdf_data
```

This code connects to a PostgreSQL database using the psycopg2.connect() function. It requires the database name, username, password, host, and port number. This SQL command creates a table named candidates if it doesn't already exist in the database. The table has columns for ID, Education, Education_dates, Experience, Experience_dates, Skills, and PDF_data.

```
conn = psycopg2.connect(
    dbname="postgres",
    user="admin1",
    password="123",
    host="localhost",
    port="5432"
)
cur = conn.cursor()

# Create table if not exists
cur.execute("""
    CREATE TABLE IF NOT EXISTS candidates (
        ID SERIAL PRIMARY KEY,
        Education TEXT,
        Education_dates TEXT,
        Experience TEXT,
        Experience_dates TEXT,
        Skills TEXT,
        PDF_data BYTEA
    )
""")
```

The data extracted from the resume and stored in variables like education_list_without_dates, experience_list_without_dates, skills_set, edu_tenure_durations_str, and exp_tenure_durations_str are prepared for insertion into the database. Strings are joined where necessary.

```
education_text = " | ".join(education_list_without_dates)
experience_text = " | ".join(experience_list_without_dates)
skills_text = ", ".join(skills_set)
edu_tenure_durations = " | ".join(edu_tenure_durations_str)
exp_tenure_durations = " , ".join(exp_tenure_durations_str)
```

This SQL command inserts the extracted data and PDF binary data into the candidates table.

```
cur.execute("""
    INSERT INTO candidates (Education, Education_dates, Experience, Experience_dates,
Skills, PDF_data)
    VALUES (%s, %s, %s, %s, %s, %s)
""", (education_text, edu_tenure_durations, experience_text, exp_tenure_durations,
skills_text, pdf_data))
```

8.3 CalculateSimilarity.py

This function connects to a PostgreSQL database, executes a SELECT query to fetch data from a table called candidates, and returns all the rows fetched from the query.

```
def fetch_data_from_db():
    # Connect to PostgreSQL database
    conn = psycopg2.connect(
        dbname="postgres",
        user="admin1",
        password="123",
        host="localhost",
        port="5432"
    )
    cur = conn.cursor()

    # Execute query to fetch data
    cur.execute("SELECT id, Education, Education_dates, Experience, Experience_dates,
Skills, PDF_data FROM candidates")

    # Fetch all rows from the executed query
    rows = cur.fetchall()

    # Close the cursor and connection
    cur.close()
    conn.close()

    return rows
```

This function processes a single row of data fetched from the database. It extracts different vectors (like education, experience, skills) and the PDF data from the row and returns them.

```
def process_data(row):
    education_vector = row[1]
    education_dates_vector = row[2]
    experience_vector = row[3]
    experience_dates_vector = row[4]
    skills_vector = row[5]
    pdf_data = row[6]  # Retrieve PDF data
    return education_vector, education_dates_vector, experience_vector,
experience_dates_vector, skills_vector, pdf_data
```

```
def process_query():
    data = request.get_json()
    query_list = data['resultList']  # Extract the list of queries directly without
the 'resultList' key
    rows = fetch_data_from_db()
```

- It receives JSON data from the frontend, containing a list of queries.
- Extracts the list of queries directly without accessing the 'resultList' key.
- Fetches candidate data from the database using the fetch_data_from_db function.
- Initializes an empty list response_data to store the response.

```
# Iterate through each parameter in the query
        for param in query:
            category, value = param.split(':')
            if category.strip().lower() == 'skills':
                skills.extend(value.split(','))
            elif category.strip().lower() == 'experience':
                experience.extend(value.split(','))
            elif category.strip().lower() == 'education':
                education.extend(value.split(','))
```

- Splits each parameter in the query into category and value based on the colon (:) separator.
- Extracts skills, experience, and education values from the query and populates the corresponding lists.

```
candidate_texts = [(row[1], row[2], row[3]) for row in rows]  # Assuming education at
index 0, experience at index 1, and skills at index 2

        # Vectorize the query skills, experience, and education
        vectorizer = CountVectorizer().fit(skills + experience + education)
        skills_vector = vectorizer.transform(skills)
        experience_vector = vectorizer.transform(experience)
        education_vector = vectorizer.transform(education)
```

- Uses CountVectorizer from scikit-learn to vectorize the skills, experience, and education lists.
- fit method learns the vocabulary from the combined lists of skills, experience, and education.
- transform method converts each list into a vector representation based on the learned vocabulary.

```
similarities = []
        for candidate_education, candidate_experience, candidate_skills in
candidate_texts:
            candidate_education_similarity = cosine_similarity(education_vector,
vectorizer.transform([candidate_education]))
            candidate_experience_similarity = cosine_similarity(experience_vector,
vectorizer.transform([candidate_experience]))
            candidate_skills_similarity = cosine_similarity(skills_vector,
vectorizer.transform([candidate_skills]))

            total_similarity = (candidate_education_similarity.mean() +
candidate_experience_similarity.mean() + candidate_skills_similarity.mean()) / 3
            similarities.append(total_similarity)

        # Get top 5 candidate indices with highest similarity
        top_5_indices = sorted(range(len(similarities)), key=lambda i:
similarities[i], reverse=True)[:5]
```

- Iterates through each candidate's education, experience, and skills text data.
- Calculates cosine similarity between each candidate's data and the query vectors.
- Computes the total similarity as the mean of similarities across education, experience, and skills.

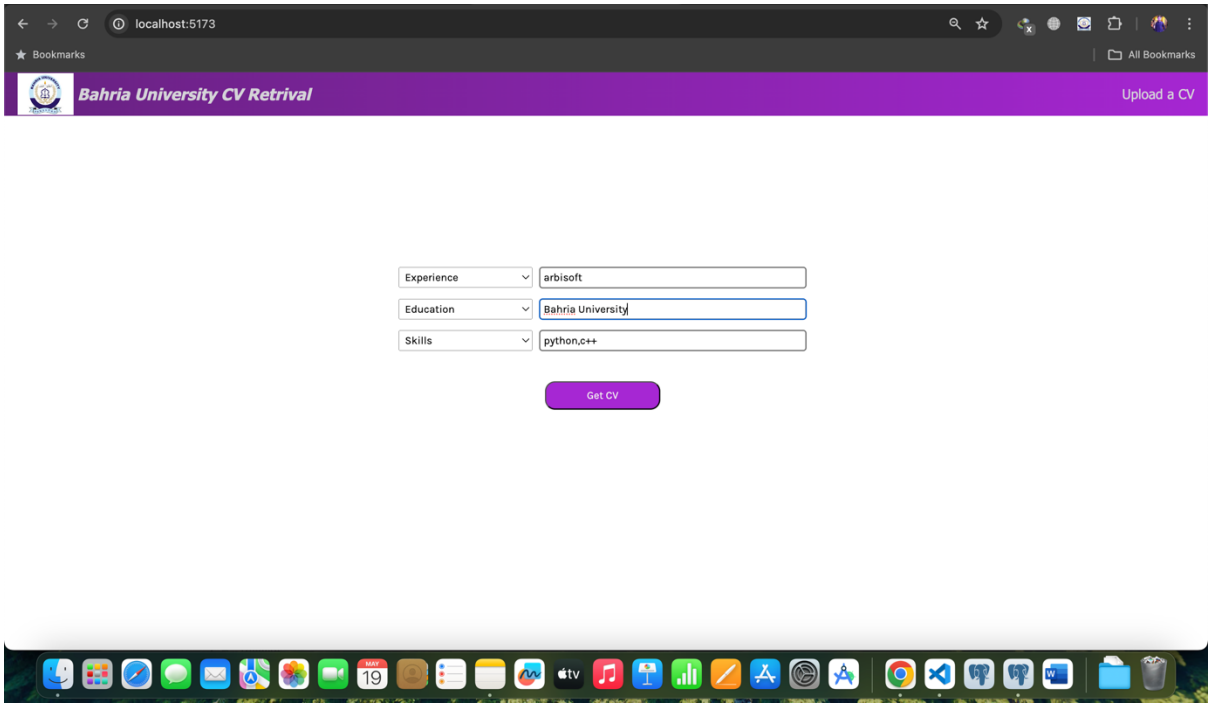
```
for idx in top_5_indices:
            candidate = rows[idx]
            response_data.append({
                "id": candidate[3],  # Assuming candidate ID at index 3
                "pdf_link": f"http://127.0.0.1:5000/get_cv/{candidate[3]}",  #
Assuming candidate ID at index 3
                "total_similarity": similarities[idx]
            })

    return jsonify(response_data)
```

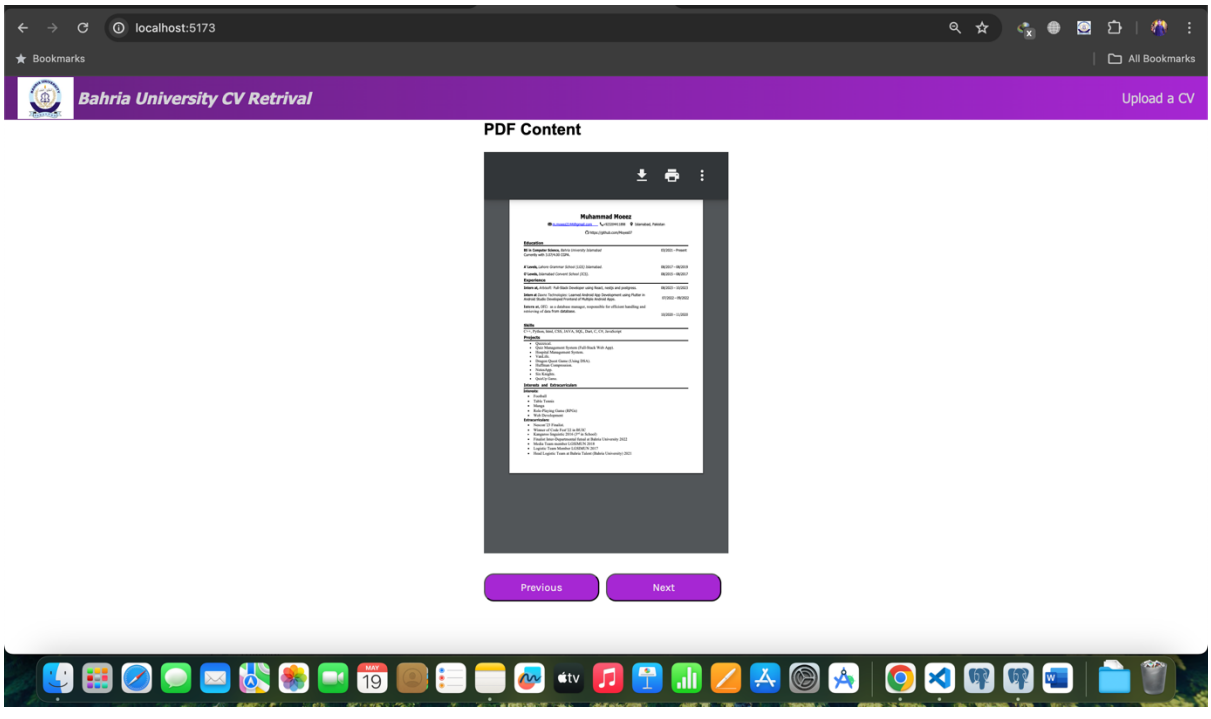
- Identifies the indices of the top 5 candidates with the highest similarity.
- Retrieves candidate data for each top candidate from the rows**.**
- Constructs the response data containing candidate ID, PDF link, and total similarity.
- Appends the response data for each top candidate to the response_data list.

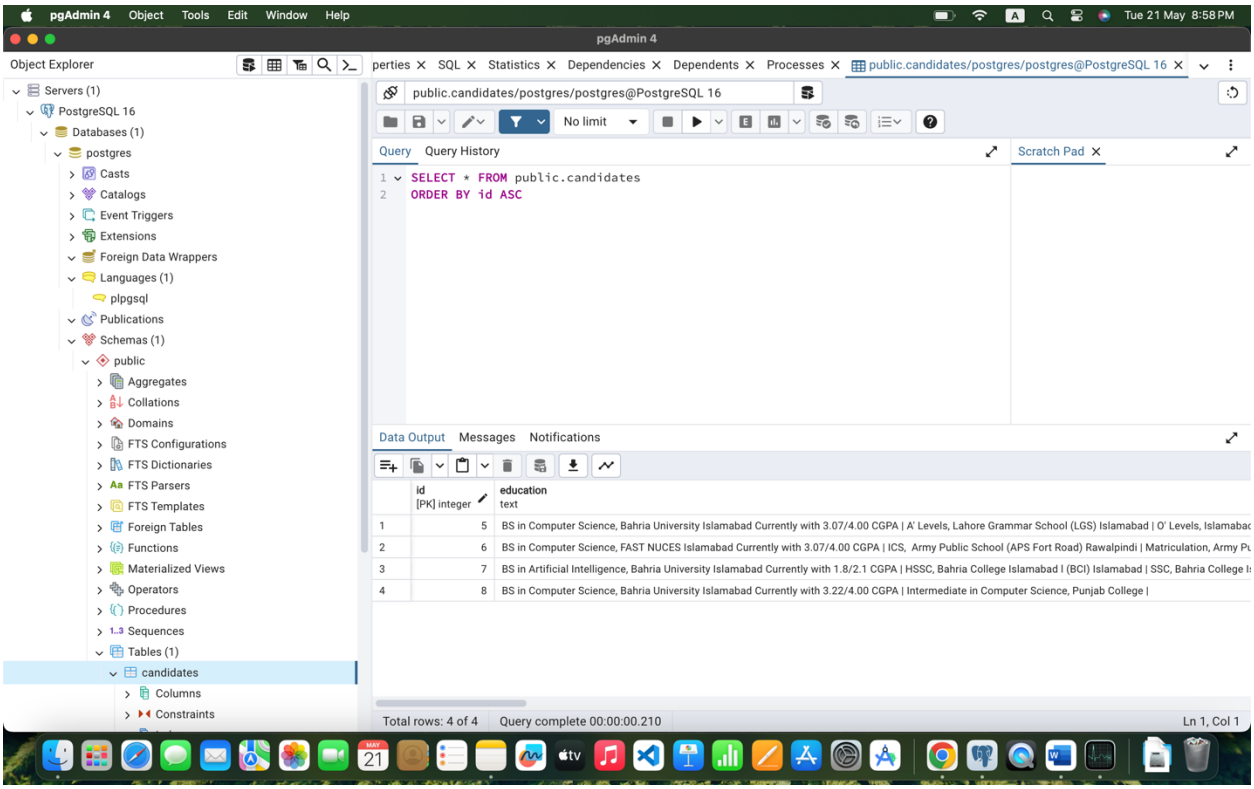
```
@app.route('/get_cv/<int:candidate_id>', methods=['GET'])
def get_cv(candidate_id):
    # Fetch data for the candidate from the database
    conn = psycopg2.connect(
        dbname="postgres",
        user="admin1",
        password="123",
        host="localhost",
        port="5432"
    )
    cur = conn.cursor()
    cur.execute("SELECT PDF_data FROM candidates WHERE id=%s", (candidate_id,))
    pdf_data = cur.fetchone()[0]
    cur.close()
    conn.close()

    # Create a temporary file to store the PDF data
    with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
        tmp_file.write(pdf_data)
        tmp_file.seek(0)

        # Send the PDF file to the frontend for rendering
        return send_file(tmp_file.name, as_attachment=False,
mimetype='application/pdf')
```

This route retrieves the PDF data for a specific candidate from the database, creates a temporary file to store the PDF data, and then sends the PDF file to the frontend for rendering.

9. Sample Screenshots






pgAdmin 4
Object Explorer
Servers (1)
PostgreSQL 16
Databases (1)
postgres
Casts
Catalogs
Event Triggers
Extensions
Foreign Data Wrappers
Languages (1)
plpgsql
Publications
Schemas (1)
public
Aggregates
Collations
Domains
FTS Configurations
FTS Dictionaries
FTS Parsers
FTS Templates
Foreign Tables
Functions
Materialized Views
Operators
Procedures
Sequences
Tables (1)
candidates
Columns
Constraints
public.candidates/postgres/postgres@PostgreSQL 16
No limit
Query
Query History
Scratch Pad
SELECT * FROM public.candidates
ORDER BY id ASC
Data Output
Messages
Notifications
id
[PK] integer
education
text
1 5 BS in Computer Science, Bahria University Islamabad Currently with 3.07/4.00 CGPA | A' Levels, Lahore Grammar School (LGS) Islamabad | O' Levels, Islamabac
2 6 BS in Computer Science, FAST NUCES Islamabad Currently with 3.07/4.00 CGPA | ICS, Army Public School (APS Fort Road) Rawalpindi | Matriculation, Army Pu
3 7 BS in Artificial Intelligence, Bahria University Islamabad Currently with 1.8/2.1 CGPA | HSSC, Bahria College Islamabad I (BCI) Islamabad | SSC, Bahria College Is
4 8 BS in Computer Science, Bahria University Islamabad Currently with 3.22/4.00 CGPA | Intermediate in Computer Science, Punjab College |
Total rows: 4 of 4
Query complete 00:00:00.210
Ln 1, Col 1

10. GitHub & LinkedIn

Arslan Shaukat

GitHub: github.com/arslan-sb
LinkedIn: https://www.linkedin.com/in/arslan-shaukat-12503a155/

Muhammad Moeez

GitHub: https://github.com/Moyes07/CV-ResumeRetrival
LinkedIn: https://www.linkedin.com/in/muhammad-moeez-8613072ba/

11. Drive link

Google Drive:
https://drive.google.com/drive/u/1/folders/13SZcEI-0GrJgMLqiY-8aseEUy0zmcmr8

12. REFERENCES

https://spacy.io/usage/models

https://pypi.org/project/PyPDF2/

https://scikit-learn.org/stable/

https://docs.python.org/3/library/re.html

https://flask.palletsprojects.com/en/3.0.x/

https://www.postgresql.org/docs/