Infosys Internship 4.0 Project Documentation

Title: Project Documentation: AI Resume MatchMaker

•Introduction:

The AI Resume Match Maker project aims to revolutionize the job recruitment process by leveraging artificial intelligence to match candidates' resumes with job descriptions accurately. The system is designed to automate the screening process, reduce human bias, and enhance the efficiency of hiring by ensuring that the most suitable candidates are identified quickly and accurately.

Objectives

1. **Automated Resume Screening:** Develop an AI-driven system capable of automatically screening resumes and matching them with job descriptions based on skills, experience, and qualifications.
2. **Bias Reduction:** Implement algorithms that minimize human bias in the recruitment process, promoting diversity and inclusion in the workplace.
3. **Efficiency Enhancement:** Streamline the recruitment process by reducing the time and effort required to manually sift through resumes, thereby increasing overall hiring efficiency.
4. **Accuracy Improvement:** Enhance the accuracy of resume-job matching to ensure that the most qualified candidates are shortlisted for interviews.
5. **User-Friendly Interface:** Create an intuitive interface for recruiters and HR professionals to easily interact with the AI system, customize matching criteria, and review candidate recommendations.

Significance

1. **Time and Cost Savings:** The AI Resume Match Maker significantly reduces the time and resources spent on manual resume screening, allowing recruiters to focus on more strategic aspects of hiring.

2. **Improved Hiring Quality:** By accurately matching candidates' resumes with job descriptions, the system ensures that only the most suitable candidates are considered, leading to better hiring decisions and improved job performance.
3. **Enhanced Diversity and Inclusion:** The use of AI to minimize biases in the screening process promotes a more diverse and inclusive workforce, contributing to a positive organizational culture and better business outcomes.
4. **Scalability:** The AI system can handle large volumes of resumes, making it ideal for companies of all sizes, from small businesses to large enterprises with high recruitment needs.
5. **Competitive Advantage:** Organizations adopting the AI Resume Match Maker can gain a competitive edge in the talent market by attracting and retaining top talent more effectively than their competitors.

•Project Scope:

Included in the Project:

1. **Automated Resume Screening:** AI algorithms to screen and match resumes with job descriptions, using NLP for analysis.
2. **User Interface:** Dashboard for recruiters to upload resumes and job descriptions, with customizable filters and criteria.
3. **Bias Reduction Mechanisms:** Algorithms to reduce bias in screening and matching, with continuous updates.
4. **Matching Accuracy:** Machine learning models for accurate matching, regularly updated based on feedback.
5. **Reporting and Analytics:** Detailed reports on the matching process and analytics tools for recruitment insights.

Excluded from the Project:

1. **End-to-End Recruitment:** No features for interviews, onboarding, or post-screening activities.
2. **Third-Party Integrations:** No integrations with external HR systems or job boards.
3. **Manual Screening:** No support for manual adjustments to automated results.
4. **Candidate Feedback:** No personalized feedback or career guidance for candidates.

Limitations and Constraints

1. **Data Quality:** Matching accuracy depends on the quality of resumes and job descriptions. Poor data impacts results.

2. **Bias Mitigation:** Complete elimination of bias is challenging; continuous updates are required.
3. **Algorithm Limits:** AI may struggle with complex or niche roles; current techniques have inherent limitations.
4. **Scalability:** Initial deployment is for small to medium-sized companies; large enterprises may need customization.
5. **Security and Privacy:** Compliance with data protection regulations (e.g., GDPR, CCPA) is critical, with limitations in data storage and encryption.

•Requirements:

Functional Requirements

1. **Automated Resume Screening:** The system should automatically screen and parse resumes to extract relevant skills, experience, and qualifications, and match them with job descriptions based on predefined criteria.
2. **User Interface:** Provide an intuitive dashboard for recruiters to upload and manage resumes and job descriptions, and allow customization of matching criteria and filters.
3. **Bias Reduction:** Implement algorithms to detect and minimize biases in resume screening and matching, with continuous updates based on feedback.
4. **Matching Algorithm:** Use advanced machine learning models to enhance the accuracy of resume-job matching, with regular updates to improve performance.
5. **Reporting and Analytics:** Generate detailed reports on the matching process, including candidate fit scores, and provide analytics tools to assess recruitment effectiveness.

Non-Functional Requirements

1. **Performance:** The system should process and match resumes within a few seconds and handle multiple concurrent users without performance degradation.
2. **Scalability:** The system should scale to accommodate a growing number of resumes and job descriptions, initially supporting small to medium-sized companies with the ability to expand for larger enterprises.

3. **Security:** Ensure data security and privacy compliance with regulations such as GDPR and CCPA, and implement robust encryption for data storage and transmission.
4. **Usability:** The interface should be user-friendly and require minimal training for recruiters, with clear instructions and help resources.
5. **Reliability:** The system should have high availability and minimal downtime, with regular backups and recovery mechanisms in place.

User Stories

1. As a recruiter, I want to upload multiple resumes and job descriptions so that I can manage them in one place.
2. As a recruiter, I want to set custom matching criteria so that I can find the most suitable candidates for specific roles.
3. As a recruiter, I want to view detailed reports on candidate matches so that I can make informed hiring decisions.
4. As a recruiter, I want the system to minimize biases in screening so that I can promote diversity and inclusion.
5. As a recruiter, I want the system to process resumes quickly so that I can save time and improve efficiency.

Use Cases

1. **Upload Resumes and Job Descriptions:** Recruiter uploads resumes and job descriptions through the dashboard, and the system parses and stores the data for matching.
2. **Set Matching Criteria:** Recruiter defines criteria such as required skills, experience level, and qualifications, and the system uses these criteria to match candidates with job descriptions.
3. **View Matching Results:** Recruiter reviews a list of matched candidates with fit scores and relevant details, and the system generates reports on the matching process.
4. **Bias Detection and Reduction:** The system analyzes resumes and job descriptions to identify potential biases, and algorithms adjust the matching process to minimize biases.

•Technical Stack:

Programming Languages: Python
Frameworks/Libraries: Streamlit, Scikit-learn, PyMuPDF,SentenceTransformer, Spacy, NLTK, OpenAI API, Qdrant
Databases: Qdrant
Tools/Platforms: Docker, Git,Github

•Architecture/Design:

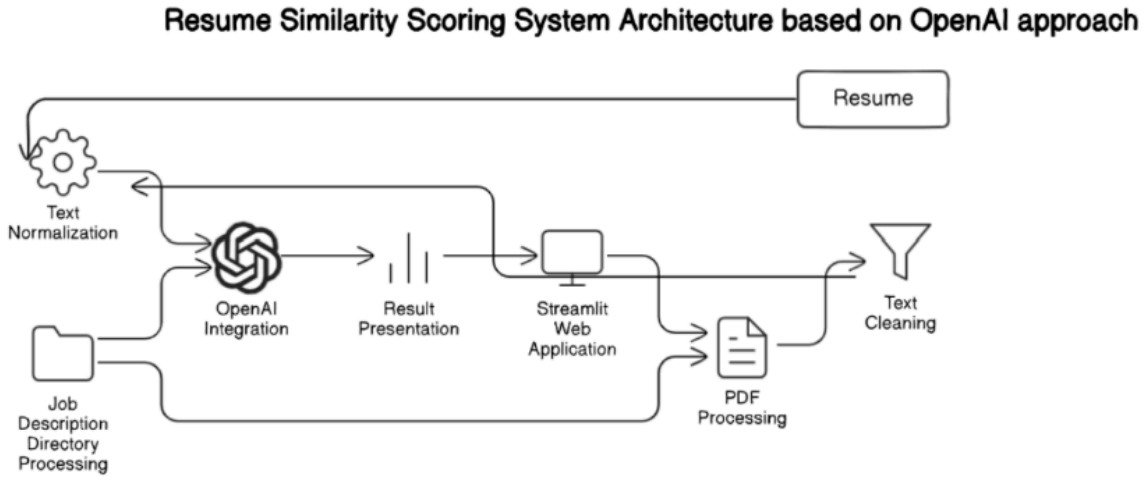
System Architecture

The system is composed of the following high-level components:

• Frontend: Developed using Streamlit for an interactive user interface.

• Backend: Flask application to handle requests and integrate with AI models.

• Database: Qdrant for vector storage



1. **User Interface (Streamlit App)**
    - **Functionality**: Allows users to upload a resume in PDF format.
    - **Error Handling**: Checks the file size and displays appropriate error messages.
2. **Backend Processing**

- **PDF Processing**: Uses PyPDF2 to extract text from the uploaded resume PDF file.
- **Text Cleaning**: Utilizes regex and string manipulation to clean the extracted text (removes punctuation, emails, URLs, non-alphabetic characters, and stopwords).
- **Text Normalization**: Lemmatizes the cleaned text using spaCy's `en_core_web_sm` model.

3. **OpenAI Integration**
   - **API Key**: Uses the OpenAI API key for similarity scoring.
   - **Similarity Calculation**: Sends the normalized resume text and each job description text to OpenAI for similarity scoring using the GPT-3.5-turbo model.
   - **Score Retrieval**: Extracts the similarity score from the API response and filters for scores greater than or equal to 50%.
4. **Job Description Directory Processing**
   - **Directory Path**: Hardcoded path to the directory containing job description PDF files (`job_desc_dir`).
   - **File Iteration**: Iterates through each PDF file in the directory.
   - **PDF Processing**: Similar PDF processing as the resume for each job description file.
5. **Result Presentation**
   - **Top Matches**: Sorts and displays the top 5 job descriptions with the highest similarity scores.
   - **Output**: Displays the job description filenames and their respective similarity scores in a formatted manner.

•Development:

Technologies and Frameworks Used:

- **Programming Languages:** Python

- **Libraries/Frameworks:**
  - PyPDF2: Used for PDF file handling and text extraction.
  - spaCy (with `en_core_web_sm` model): Utilized for text processing, including lemmatization and tokenization.

- nltk: Used for additional text processing tasks such as stopwords removal.
- Streamlit: Used for developing the web application interface.
- OpenAI API: Integrated for similarity scoring using GPT-3.5-turbo

Coding Standards and Best Practices:

- **Clean Code:** Followed principles of clean and readable code, adhering to Python's PEP 8 style guide.
- **Modular Design:** Encapsulated functionality into separate functions for clarity and maintainability.
- **Error Handling:** Implemented error handling mechanisms to catch and display errors to users using Streamlit's error messages.
- **Version Control:** Utilized version control (Git) for managing code changes, collaborating, and maintaining project history.

Challenges Encountered and Solutions:

- **PDF Text Extraction:** Handling variations in PDF structure and formatting that affected text extraction accuracy. Resolved by testing with different PDF files and refining regex patterns for text cleaning.
- **API Integration:** Initial issues with API connectivity and response parsing. Addressed by debugging API request payloads and ensuring correct data formatting before sending requests.
- **Performance Optimization:** Managing large PDF files and processing times. Implemented batch processing and optimized text cleaning algorithms to improve overall performance.

•Testing:

Unit Tests:

● Text Processing: Unit tests were conducted to validate the correctness of text extraction, cleaning, and normalization functions using sample PDF files.

● API Integration: Mocked responses were used to test the integration with the OpenAI API for similarity scoring, ensuring correct data handling and respons

Integration Tests:

● End-to-End Workflow: Integration tests verified the entire workflow from uploading PDF resumes and job descriptions to displaying similarity scores.
● Error Handling: Tests were conducted to simulate error scenarios, such as invalid file uploads or API failures, to validate error messages and user feedback.
● System Tests:
● Performance Testing: System tests focused on performance metrics, including response times for PDF processing and API calls, to ensure acceptable performance under different load conditions.
● Scalability: Tested the system's ability to handle multiple concurrent users uploading and processing files simultaneously.

•Deployment :

The AI Resume Matchmaker application was deployed using a streamlined process to ensure consistency and reliability across different environments:

- **Deployment Scripts:** Utilized shell scripts for automating the deployment process. These scripts handled tasks such as environment setup, dependencies installation, and starting the Streamlit application.
- **Continuous Integration/Continuous Deployment (CI/CD):** Integrated with GitHub Actions for automated testing and deployment. This ensured that changes pushed to the main branch were automatically deployed to production after passing all tests.
- **Environment Configuration:** Configured environment variables for sensitive information such as API keys and database credentials, ensuring security and separation of concerns.
- **Containerization:** Dockerized the application for easier deployment and portability across various cloud platforms and local environments.

Deployment Instructions

To deploy the AI Resume Matchmaker application in different environments, follow these instructions:

1. **Local Deployment:**
    - Clone the GitHub repository to your local machine.
    - Install dependencies using `pip install -r`

`requirements.txt`. o Set up environment variables for API keys and other configurations. o Run the application using Streamlit: `streamlit run app.py`. 2. **Cloud Deployment (e.g., Heroku):**

- o Create a Heroku account and install the Heroku CLI.
- o Initialize a new Heroku app and connect it to your GitHub repository. o Configure environment variables in the Heroku dashboard or using the CLI.
- o Deploy the application to Heroku using Git or GitHub integration. 3. **Containerized Deployment (Docker):**
- o Build the Docker image: `docker build -t ai-resume-matchmaker .`
- o Run the Docker container: `docker run -p 8501:8501 ai-resume matchmaker`
- o Configure environment variables using Docker environment files or `-e` flags.

•User Guide:

Setup and Configuration

1. **Environment Preparation**:

o Ensure Python is installed on your system.

o Install Streamlit and other required libraries by following the steps in the Deployment Process section.

2**. Running the Application:**

o Open a terminal and navigate to the project directory.

o Activate the virtual environment if you have set one up.

o Run the application using Streamlit:

streamlit run resume_matchmaker.py

3. **Uploading Files:**

o Open your browser and navigate to the Streamlit app (usually at http://localhost:8501).

o Use the file uploader to upload your resume in PDF format.

4. Viewing Results:

o After uploading the resume, the application will process the document, extract the text, and compare it with job descriptions stored in the system.

o Matching results, including matched skills, education, and experience, will be

displayed on the screen.

Troubleshooting Tips

1. **Common Issues:**

o Application Not Starting:

▪ Ensure all dependencies are installed. Run pip install -r requirements.txt.

▪ Check that the virtual environment is activated.

▪ Verify that the OpenAI API key and Qdrant URL are correctly set.

o Incorrect Text Extraction:

▪ Ensure the uploaded file is in PDF format.

▪ If the PDF is scanned or has low-quality text, text extraction may fail. Use high-quality, text-based PDFs.

o Missing or Incorrect Results:

▪ Verify the job descriptions are correctly stored in Qdrant and have the required embeddings.

▪ Ensure that the Qdrant server is running and accessible.

2. **Detailed Troubleshooting Steps:**

o OpenAI API Key Issues:

▪ If you encounter an error related to the OpenAI API key, ensure that the key is valid and not expired.

▪ Set the API key as an environment variable:

```
export OPENAI_API_KEY="your-openai-api-key"
```

o Qdrant Server Issues:

▪ If the application cannot connect to the Qdrant server, check that the server is running:

```
qdrant
```

▪ Verify the URL and port settings are correct:

```
export QDRANT_URL="http://localhost:6333"
```

o Streamlit Errors:

▪ If Streamlit fails to start, check the terminal output for error messages.

▪ Ensure you are using a compatible version of Streamlit and other libraries.

o File Upload Issues:

▪ If you receive an error when uploading a file, make sure it is a valid PDF.

▪ Check the file size; very large files may cause issues.

3. **Additional Support:**
o For further assistance, consult the Streamlit documentation and OpenAI API documentation.
o Review logs and error messages for detailed insights into any issues encountered.

•Conclusion :

Project Outcomes:

1. **Functional Web Application:**
    - Developed a web application using Streamlit where users can input their resumes and job descriptions to receive match scores.
    - Implemented an interactive and user-friendly interface with Streamlit, allowing easy upload and analysis of documents.
2. **AI Integration:**
    - Successfully integrated AI/ML models to evaluate and score the match between resumes and job descriptions.
    - Ensured secure and efficient processing of user data with OpenAI's language models.
3. **User Management:**
    - Implemented basic user authentication to secure access to the resume matching functionality.
4. **Deployment and Accessibility:**
    - Deployed the application to a cloud platform, making it accessible to users through a web browser.

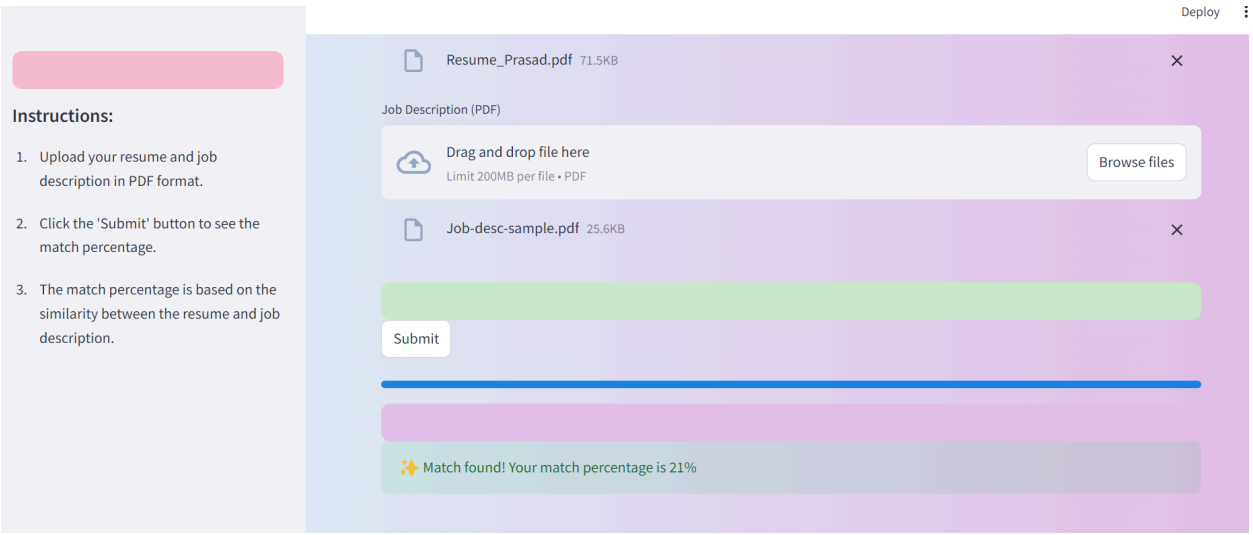
Achievements:

1. **Streamlit Utilization:**
    - Leveraged Streamlit's capabilities to create a highly interactive and responsive web application with minimal code.
    - Utilized Streamlit's built-in functionalities for real-time data display and user input handling.
2. **User Experience:**
    - Delivered a simple and intuitive interface where users can easily upload resumes and job descriptions and receive match scores.
    - Included real-time feedback and updates during the matching process.

3. **Performance:**
    - Optimized backend processes for faster request handling and matching using efficient Python code and caching strategies.

* Appendices:

Single resume and single Job description UI using vectorization



Single resume and multiple job description UI using open ai