Hiring Assistant
Project Documentation

Prepared By:
Sarah Asad

Overview

The Hiring Assistant project is designed to streamline the process of handling resumes, extracting relevant information using Named Entity Recognition (NER) and storing the data in a Neo4j graph database for efficient querying and analysis. This project comprises of both a frontend and a backend application.

This project involves building an application with the following key components:

1. **Frontend:** A user-friendly interface built with React to upload resumes and download processed results.

2. **Backend:** A FastAPI-based backend that handles file uploads, processes resumes to extract entities, and interacts with the Neo4j database.

3. **Database:** A Neo4j graph database to store and manage extracted entities and their relationships.

Key Features

1. Resume Upload and Processing

- Users can upload resume in PDF format.
- The system parses the resumes, pulling out raw data from resumes stored in PDF format.

2. Entity Recognition

- Utilizes spaCy's pre-trained model to perform Named Entity Recognition (NER) on the parsed resumes extracting key entities such as names, skills and experiences using Natural Language Processing (NLP) techniques.

- Extracted entities are organized and categorized for better analysis.

3. Data Storage in Neo4j

- Extracted entities are stored in a Neo4j graph database.
- This allows for efficient querying and relationship mapping between different entities.

4. Download Processed Data

- Users can download the processed resume data, including the extracted entities in a CSV format.

5. **User-Friendly Interface**

- A React-based frontend application for ease of interaction.
- Simple and intuitive design, ensuring a smooth user experience.

Technologies Used

1. **Backend: FastAPI**

- Provides a robust and high-performance web framework for building APIs.
- Handles various endpoints for uploading, parsing and storing resume data.

2. **Frontend: React**

- It provides a user-friendly interface for interacting with Hiring Assistant application. Users can upload resumes and view search results.

- It connects with the backend (built with FastAPI and other technologies like Neo4j) to send and receive data. For instance, it might send resume data for processing and receive processed results back from the backend.

- Ensures a responsive design, real time updates using HMR (Hot Module Replacement) from Vite and provides dynamic user experience.

3. **Natural Language Processing**: spaCy

- A powerful NLP library in Python used for extracting entities from resumes.
- Provides pre-trained models for accurate entity recognition.

4. **Database**: Neo4j

- A graph database used for storing and managing extracted entities.
- Facilitates complex queries and relationships between entities.

5. **Containerization**: Docker

- Ensures the application is easy to deploy and manage.
- Docker Compose is used to orchestrate the different services (frontend, backend and database).

Application Flow

1. Upload Resume:

- User uploads a resume through the frontend application.
- The resume file is sent to the backend for processing.

2. Resume Processing:

- The backend extracts entities from the resume using spaCy.
- Extracted entities are stored in the Neo4j database.

3. Data Retrieval and Download:

- Users can query the database to retrieve specific data.
- Processed data can be downloaded in CSV format.

File Descriptions

- **main.py**
  The main.py file serves as the main entry point for the backend application.

- **README.md**
  The README.md file provides guidance and instructions for setting up and running the application.

- **requirements.txt**
  The requirements.txt file lists all the dependencies required to run the application.

- **Dockerfile**
  The Dockerfile contains a set of instructions to create a docker image for the application. It specifies the base image, copies application files, installs dependencies and sets up the environment to run the application within a docker container. Steps for building and running a docker image using a dockerfile:

Step 1. Build the Docker image

```
docker build -t myapp .
```

Step 2: Run the Docker container based on the built image

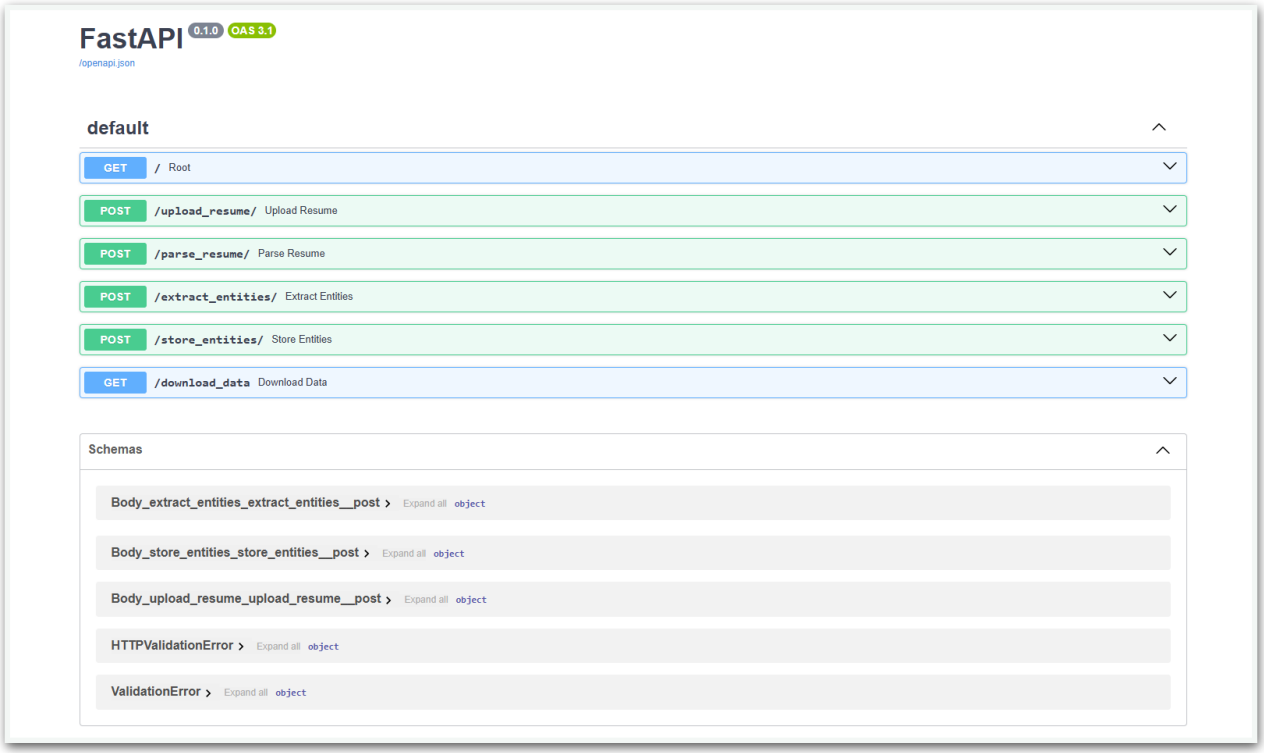
```
docker run -p 8000:8000 myapp
```

Project Structure

1. Backend (FastAPI)

The backend of our application serves as the core processing unit responsible for handling resume uploads, parsing and extracting entities using Natural Language Processing (NLP) techniques and storing this information in a Neo4j graph database. We have used following endpoints to accomplish our task.

1. /upload_resume/: Endpoint for uploading a resume file.
2. /parse_resume/: Endpoint for parsing and processing uploaded resumes to extract information from pdf files.
3. /extract_entities/: Endpoint for extracting entities from parsed resume files using NER.
4. /store_entities/: Endpoint for storing entities in the Neo4j database.
5. /download_data: Endpoint for extracting and downloading data from the Neo4j database.



Backend application

Backend Setup:

- Navigate to the server directory and install dependencies:

  pip install -r requirements.txt

- Run the FastAPI server:

  uvicorn main:app —host 0.0.0.0 --port 13001

2. Frontend (React)

Role of Frontend in application:

User Interface:

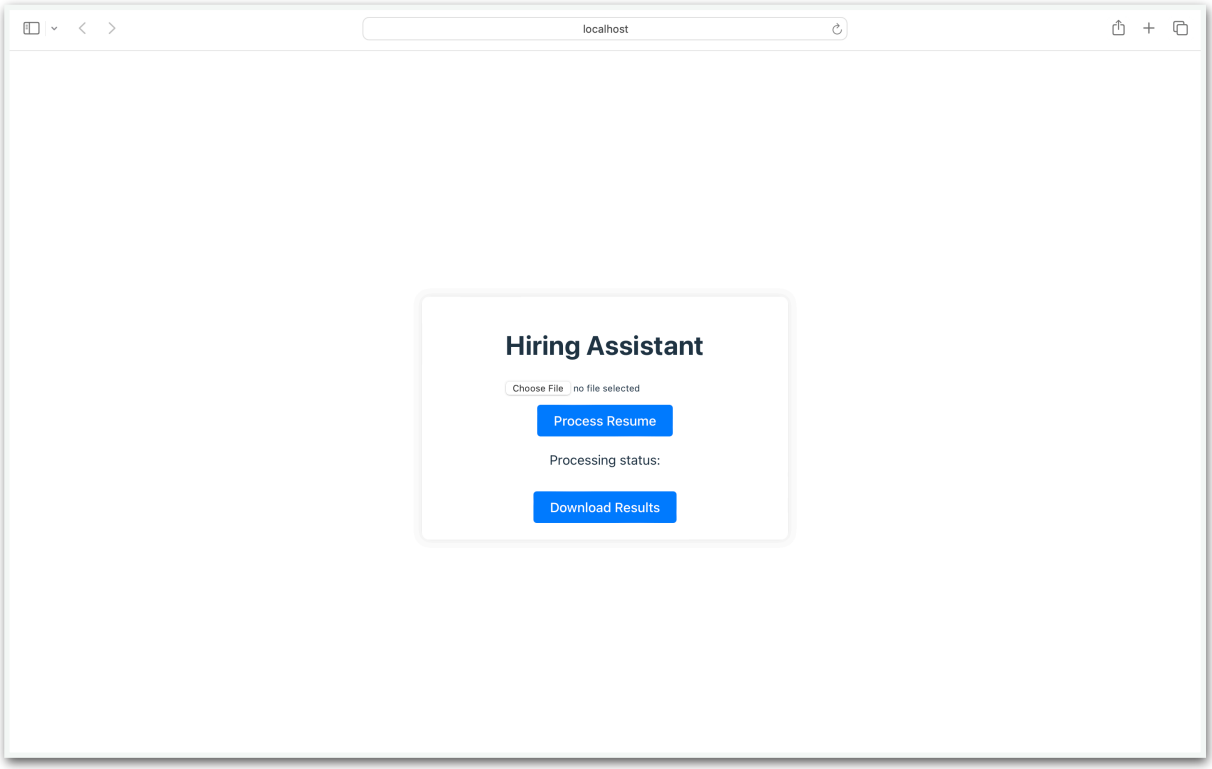
- Resume Upload: Provides a form for users to upload their resumes in PDF format.
- Feedback: Displays messages and statuses regarding the upload and processing of resumes.
- Results Display: Shows the extracted entities from the resumes and any relevant information retrieved from the Neo4j database.

Integration with Backend:

- API Calls: Makes HTTP requests to the FastAPI backend to upload resumes, parse the uploaded files, extract entities and store or retrieve data.

- Data Handling: Manages the data flow between the user inputs and the backend responses, ensuring a smooth interaction.

Note: Vite's Hot Module Replacement (HMR) enables fast updates to our frontend application during development without reloading the entire page, ensuring quick feedback on code changes and preserving application state for smoother user interactions.



Frontend application

Frontend Setup:

- Ensure you have Node.js installed.

- Navigate to the src directory in FrontEnd directory and install dependencies:
  npm install

Note: This command installs all the dependencies listed in your package.json file.

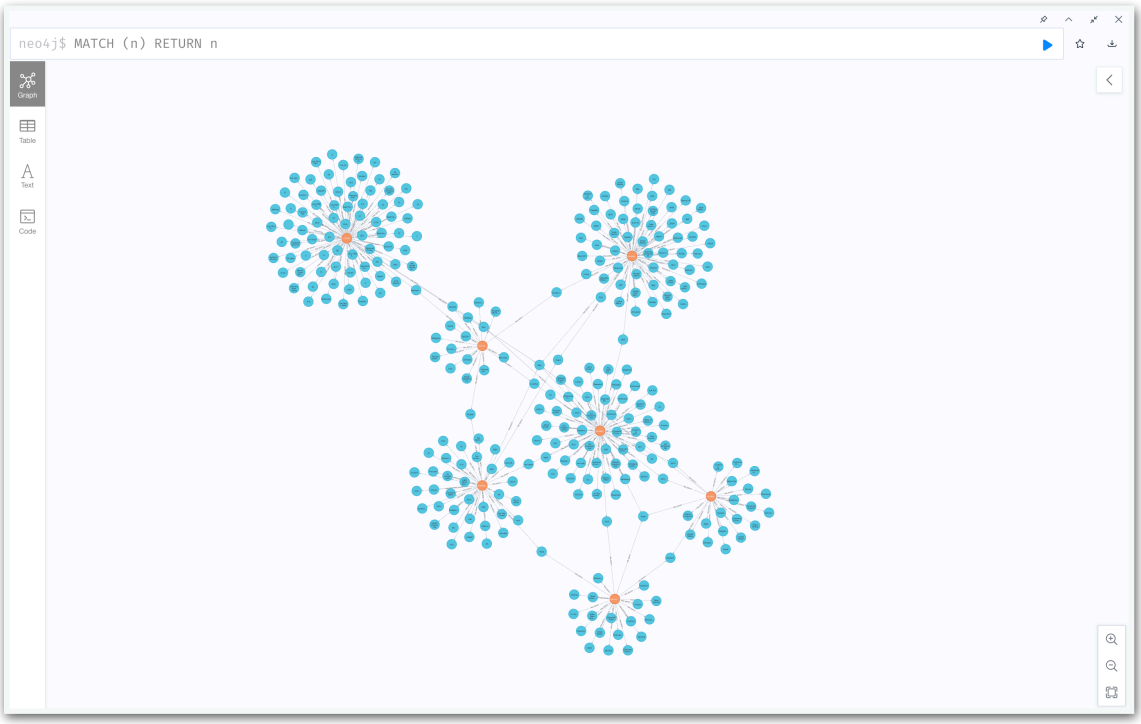
- Run the React development server:
  npm run dev

Note: This command starts the development server. Typically, it will print a message saying where your development server is running (e.g., http://localhost:5173). Open this URL in your browser to view your application.

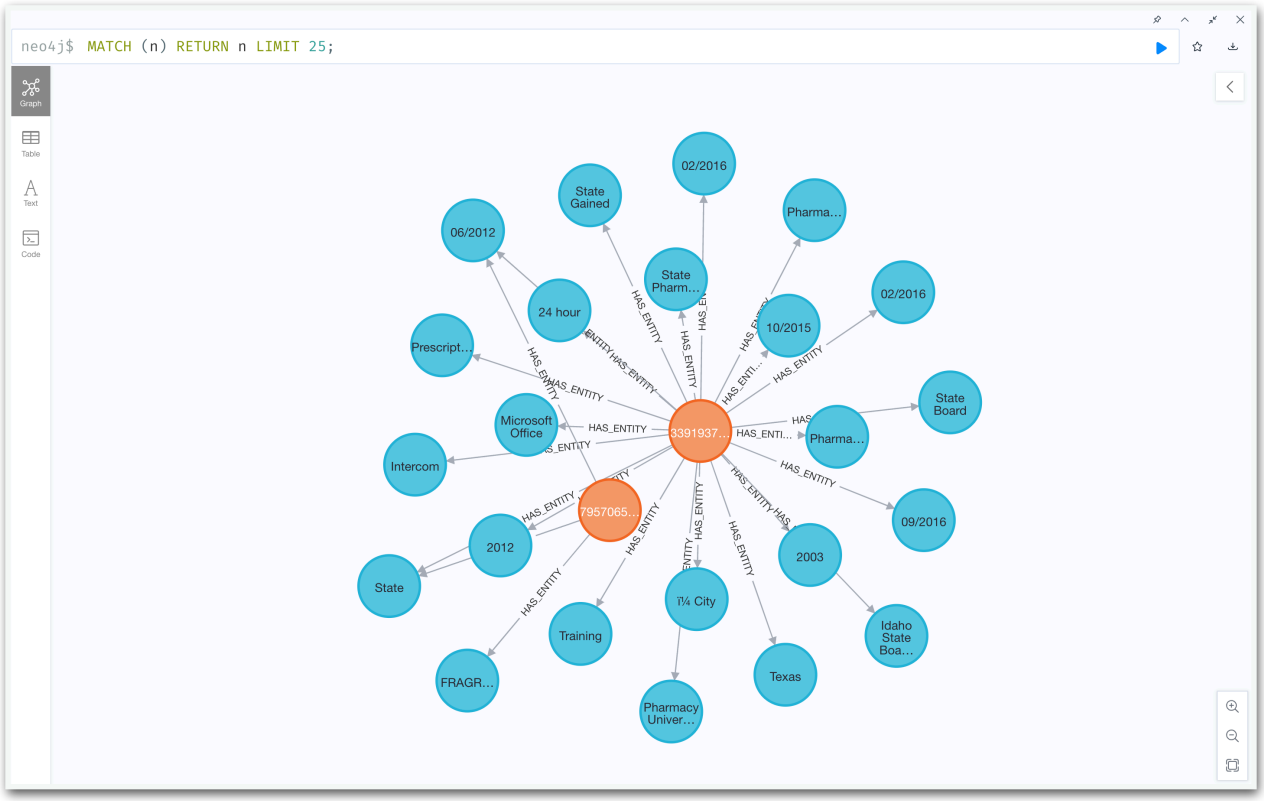
3. Neo4j Database

Neo4j is a highly scalable, native graph database that is optimized for handling connected data and complex queries with ease. It stores data in nodes, relationships between nodes, and properties associated with both. Neo4j uses a property graph model where nodes represent entities (like people or products), relationships represent connections between these entities (like friendships or purchases), and properties provide additional details about nodes and relationships.

Neo4j's query language, Cypher, is specifically designed for graph traversals and pattern matching, making it powerful for querying and analyzing graph data efficiently.



Graph showing nodes and relationships



Top 25 nodes and entities displayed

4. Docker Compose

Docker Compose is a tool that allows you to define and manage multi-container Docker applications. In our project, we've used Docker Compose to integrate multiple components: the frontend, backend (FastAPI application), and Neo4j database. Docker Compose simplifies the process of running, scaling and managing our application stack by specifying the configuration in a single YAML file (docker-compose.yml), which includes all necessary services, networks, and volumes. This approach streamlines development, testing and deployment workflows, enhancing overall efficiency and consistency across different deployment environments.

To run application using docker-compose file, navigate to the directory containing docker-compose.yml and run:

“docker-compose up —build"

Frontend: Access your frontend at http://localhost:13002
Backend (FastAPI): Access the Swagger UI at http://localhost:13001/docs

Project Components Explained:

1. PDF Parser

This component utilizes the PDFMiner library from the Python Package Index (PyPI). It is designed to extract text and data from PDF documents, preserving formatting and layout information. This capability is crucial for applications needing to programmatically access content within PDF files, such as processing resumes or other PDF-based documents. The PDF parser ensures efficient handling of PDF documents.

```python
from pdfminer.high_level import extract_text

def parse_resume(pdf_file):
    text = extract_text(pdf_file)
    return text
```

An excerpt from pdf_parser.py

2. NER (Named Entity Recognition)

This component leverages the **spaCy** library of Python available on Python Package Index (PyPI). NER is a natural language processing technique that identifies and classifies named entities (such as names of persons, organizations, locations etc.) within text.

Specifically, this project utilizes the en_core_web_sm model from spaCy, which is pre-trained and optimized for various NLP tasks, including entity recognition. The en_core_web_sm model provides efficient and accurate processing of text data, enabling the extraction of relevant entities from resumes or other textual documents.

```python
import spacy

# Load the language model
nlp = spacy.load("en_core_web_sm")

# Function to extract entities using spaCy NER
def extract_entities(text):
    doc = nlp(text)
    entities = [(ent.text, ent.label_) for ent in doc.ents]
    return entities
```

An excerpt from NER.py

Conclusion

In conclusion, the Hiring Assistant project represents a significant advancement in modern recruitment processes by leveraging cutting-edge technologies. By integrating PDF parsing, Named Entity Recognition (NER), and a Neo4j graph database, the project streamlines the extraction, analysis, and storage of candidate data from resumes. This innovative approach not only automates tedious manual tasks but also enhances the accuracy and efficiency of matching candidates to job descriptions.

Looking ahead, the project has the potential for further enhancements, such as real-time matching algorithms, machine learning-driven insights and seamless integration with existing HR systems. These improvements aim to empower recruiters with actionable data, expedite decision-making and ultimately contribute to more effective and informed hiring practices.

Overall, the Hiring Assistant project underscores the transformative impact of technology in optimizing recruitment processes, paving the way for more intelligent, data-driven approaches to talent acquisition in organizations.

Future Recommendations

Future recommendations could focus on developing a comprehensive platform that streamlines the entire recruitment process. This platform would automate resume processing and analysis, integrating seamlessly with job descriptions (JDs) to assess candidate suitability efficiently. Key enhancements could include:

1. **Automated Matching Algorithms**: Implementing advanced algorithms that automatically match candidate resumes with job descriptions based on skillsets, experience, and qualifications. This would provide instant feedback on candidate suitability for specific roles.

2. **Real-time Updates**: Enabling real-time updates where recuiters receive notifications or alerts when a new JD is added, and the system automatically evaluates existing resumes to recommend suitable candidates.

3. **Machine Learning Enhancements**: Leveraging machine learning models to continuously improve matching accuracy by learning from historical data and recruiter feedback. This would enhance the platform's ability to predict successful candidate-job matches.

4. **User Interface Refinement**: Enhancing the frontend user interface (UI) to improve usability, accessibility, and user experience, making it more intuitive for recruiters and HR professionals to interact with the application.

5. **Security Enhancements**: Implementing robust security measures to protect sensitive candidate data stored in the Neo4j database and ensuring compliance with data privacy regulations.

Running the application

1. **Backend**: Navigate to the server directory and build the docker image.

   docker build -t fastapi-app .

   docker run -p 13001:13001 fastapi-app

2. **Frontend**: Navigate to the src directory and build the docker image.

   docker build -t frontend .

   docker run -p 13002:13002 frontend

3. **Using docker compose**: Navigate to the root directory where docker compose.yml is placed, execute the command:

   docker-compose up --build

Installing and setting up a local database in Neo4j Desktop

1. **Download Neo4j Desktop**: Go to the Neo4j Download link and download Neo4j Desktop for your operating system (Windows, macOS, or Linux).

2. **Install Neo4j Desktop**: Run the installer and follow the on-screen instructions to complete the installation.

3. **Open Neo4j Desktop**: Launch Neo4j Desktop after installation.

4. **Create a New Project**: When Neo4j Desktop opens, click on the "Add Graph" button.

5. **Set Up a New Database**: Click on "Create a Local DBMS" to set up a new local database management system (DBMS).

6. **Configure Database Settings**: Choose a name for your database. Set a password for the default `neo4j` user (you'll need this later to connect to the database).

7. **Start the Database**: Once configured, click on "Start" to start the local database instance.

8. **Access Neo4j Browser**: After the database starts, you can access the Neo4j Browser by clicking on the "Open" button next to your database instance in Neo4j Desktop.

9. **Neo4j Browser:** Open your web browser and go to http://localhost:7474/. This interface allows you to write and execute Cypher queries, visualize your graph data and manage your Neo4j database through a graphical interface.