# Job Recommendation System Documentation

## Overview

This document outlines the implementation of a job matching system that uses natural language processing and machine learning techniques to match job postings with potential candidates.

## System Components

### 1. Data Retrieval

* **Database**: SQL Server (MPOWER\_TEST)
* **Connection**: PyODBC
* **Data Types**: Job postings and member (candidate) profiles

### 2. Text Processing

* **Function**: create\_job\_text() and create\_member\_text()
* **Purpose**: Convert structured data into a single text representation for each job and candidate

### 3. Embedding Generation

* **Model**: BERT (bert-base-uncased)
* **Library**: Transformers
* **Function**: get\_bert\_embedding()
* **Process**: Tokenization, BERT encoding, mean pooling

### 4. Similarity Calculation

* **Method**: Cosine Similarity
* **Library**: Scikit-learn
* **Function**: calculate\_similarity()

### 5. Matching and Ranking

* **Process**:
  1. Calculate similarity between each job and all candidates
  2. Filter matches with >60% similarity
  3. Rank top 150 matches per job

### 6. Explanation Generation

* **Model**: GPT-4 (gpt-4o-mini-2024-07-18)
* **Library**: OpenAI's Python client
* **Function**: explain\_similarity()
* **Purpose**: Generate human-readable explanations for each match

### 7. API Endpoint

* **Framework**: Flask
* **Route**: /get\_recommended\_profiles/<job\_id>
* **Method**: GET
* **Response**: JSON with job details, matched candidates, and explanations

## Workflow

1. Fetch job and candidate data from the database
2. Create text representations for jobs and candidates
3. Generate BERT embeddings for all texts
4. For each job:  
   a. Calculate similarity with all candidates  
   b. Filter and rank top matches  
   c. Generate explanations for top matches using GPT-4
5. Format and return results via API

## Key Libraries and Dependencies

* PyODBC
* Transformers (Hugging Face)
* PyTorch
* Scikit-learn
* Flask
* OpenAI
* Asyncio

## Notes

* The system uses asynchronous processing to handle multiple API requests efficiently.
* BERT embeddings provide a semantic understanding of job and candidate descriptions.
* GPT-4 generates nuanced explanations of why a candidate matches a job.
* The 60% similarity threshold and top 150 limit are configurable based on needs.

## Future Improvements

* Implement caching for embeddings to improve performance
* Add more advanced filtering options (e.g., by location, salary range)
* Integrate a feedback loop to improve matching over time
* Enhance error handling and logging for production environments