Project: Question Answering (QA) System Technical Documentation

Task 0: Directory Structure

The directory structure is the foundational backbone of any project. For this QA system, a coherent and straightforward structure was designed:
- docs/: For storing essential data files, like the passages and other relevant datasets.
- corpus/: A folder which contains files of case judgments and their corresponding metadata (each pair has the same filename with the only difference being the endings; _Technical.txt
- app/: Source directory containing all Python scripts and modules.
- Docker/: Contains the dockerfile, requirements.txt(Specifies all Python dependencies) and .dockerignore (to ignore copying my venv)

Task 1: Parsing

The XML dataset, consisting of legal passages, was parsed with Python's built-in xml module. Each passage, along with its metadata, was meticulously extracted. This ensures streamlined indexing and searching in the stages that follow.

Task 2: Passage Embeddings Generation

The generation of passage embeddings offers numerical representations for textual data, thereby enhancing search efficiency.
- The acclaimed sentence-transformers library was harnessed for this task.
- The model paraphrase-MiniLM-L6-v2 was specifically chosen for its adeptness in similar undertakings. This model is a lightweight and efficient alternative, which provides good quality embeddings especially in a setup where computational efficiency is desired.

Task 3: ElasticSearch Integration

ElasticSearch, a robust search engine, was seamlessly integrated into the system to efficiently retrieve passages:
- Upon generating their embeddings, each passage was indexed into ElasticSearch, paired with its metadata.
- The Python client for ElasticSearch was instrumental in ensuring a seamless integration.
- The specific steps covered in the class are:

Initialization: It loads the SentenceTransformer model, sets up the Elasticsearch connection, and defines the path to your CSV file.
Elasticsearch Setup: It connects to the Elasticsearch instance using the provided credentials and endpoint.
Index Creation or Update: It either creates a new index with the specified mappings or updates an existing one.
Data Indexing: It reads the data from the CSV file and indexes each row to Elasticsearch.
Execution: It runs the indexing process

Task 4: Document Retrieval

The retrieval of pertinent passages is crucial:
- A dedicated class named Retrievals was established within the retrieval.py file.

- This class interfaces with the ElasticSearch server, effectively fetching relevant passages for any proffered query.
- overview of what the code does:
The Retrievals class is initialized, setting up the Elasticsearch connection and loading the Sentence Transformer model.
The compute_embedding function calculates embeddings using the Sentence Transformer model.
The search_similar_documents function constructs a query to Elasticsearch using the given embeddings and returns the top 3 most similar passages.
The retrieve_answers function accepts a list of questions, computes embeddings for each question, retrieves relevant passages from Elasticsearch, and then writes the results to a CSV file.

Task 5: Containerization
Containerization ensures the application's functionality across varying environments:
- Docker was utilized for containerizing the application.
- A Dockerfile was crafted, detailing the necessary instructions to encapsulate the application.

Task 6: API Deployment with Flask
The Flask framework, known for its agility, was employed for API creation:
- The app.py script functions as the heart of the Flask app.
- Both the retrieval and indexing parts have been integrated into the Flask API, and this setup allows users to search for relevant passages based on their questions and to upload new documents to be indexed.
- Two endpoints were set up, facilitating the addition of passages and retrieval of answers.
- Through the config.py file and its Config class system, managing different environments (production, staging, development, and Testing.) becomes a breeze.

Task 7: Evaluation
The importance of a comprehensive evaluation mechanism cannot be overstated:
- An evaluation.csv file was generated, which records the top 3 answers for ten distinct queries taken from user_queries.txt.
- A separate script (manual_rating_and_performace_evaluation.py) aids in the manual rating process, culminating in the evaluation_rated.csv file.
- Both Top-1 and Top-3 accuracy metrics are computed, with the results diligently stored in performance.csv.

Task 9 and 10
In Progress

Conclusion
This project epitomizes a comprehensive approach to architecting a QA system. By melding state-of-the-art embedding techniques with the capabilities of

ElasticSearch, and ensuring versatility and scalability through Docker and Flask, a resilient and efficient system is born.