MCQ GENERATOR DOCUMETATION:

Overview

The **MCQ Generator** is a Python-based application designed to generate, manage, and import multiple-choice questions (MCQs) into the Examly platform for two domains: **LTI** and **Neowise**. It leverages a Streamlit interface for user interaction, integrates with Elasticsearch for duplicate detection, and uses the Azure OpenAI API for generating high-quality MCQs. The application supports various question types, difficulty levels, and problem-solving filters, ensuring diverse and relevant question sets.

WORK FLOW :

ARCHITRCTURE DIAGRAM:

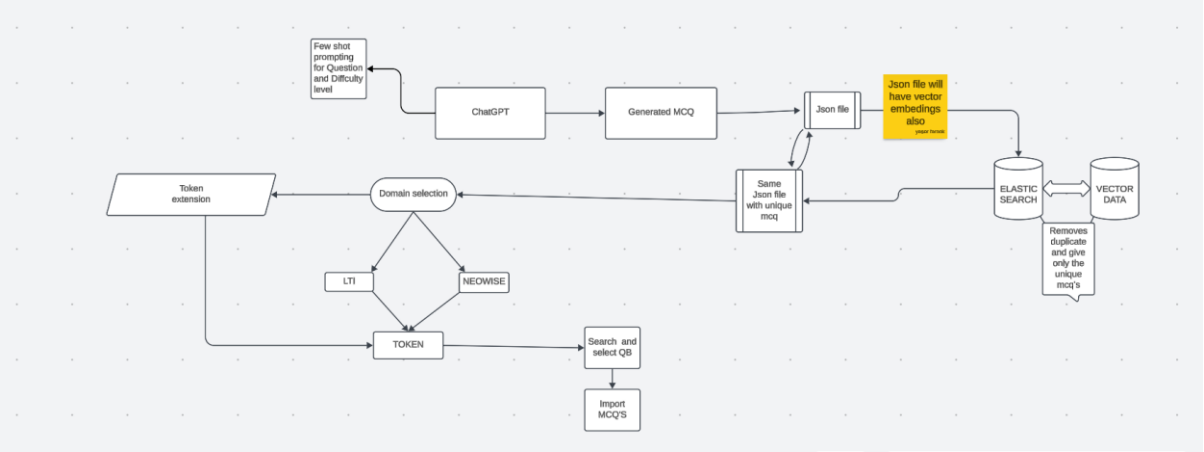

Table of Contents

- [MCQ Generation]
- [Question Bank Management]
- [Elasticsearch Integration]
- [API Integration]

7. [File Structure]
8. [Error Handling and Logging]
9. [Limitations]
10. [Future Enhancements]

Features

- MCQ Generation: Generate MCQs based on user-specified topic, number of questions, difficulty level (Easy, Medium, Hard), and question type (Conceptual, Factual, Problem-solving, Scenario-based).

- Problem-solving Filters: For Problem-solving questions, users can select specific subtypes (e.g., Output prediction, Error identification, Debugging).

- Duplicate Detection: Uses Elasticsearch and sentence transformers to identify and filter out duplicate questions.

- Question Bank Fetching: Retrieve question banks from Examly API for LTI or Neowise domains.

- MCQ Import: Import generated MCQs into selected question banks on the Examly platform.

- Streamlit UI: Intuitive web interface for user input and interaction.

- Logging: Comprehensive logging for debugging and error tracking.

System Architecture

The MCQ Generator is composed of several interconnected modules:

1. Streamlit Frontend: Provides a user-friendly interface for inputting parameters, generating MCQs, fetching question banks, and importing questions.

2. Azure OpenAI API: Generates MCQs using the `gpt-4o-mini` model, guided by meta-sorting plans and few-shot examples.

3. Elasticsearch Backend: Stores questions and checks for duplicates using sentence embeddings from the `all-MiniLM-L6-v2` model.

4. Examly API Client: Handles communication with the Examly platform to fetch question banks and import MCQs.

5. File-based Storage: Saves generated MCQs and unique questions to files (`question_prompt.txt`, `unique_mcqs.json`).

Dependencies

The application relies on the following Python libraries:

- `streamlit`: For the web-based user interface.

- `requests`: For making HTTP requests to the Examly API.

- `python-dotenv`: For loading environment variables from a `.env` file.

- `openai`: For interacting with the Azure OpenAI API.

- `sentence-transformers`: For generating sentence embeddings to detect duplicates.

- `elasticsearch==7.17.9`: For storing and querying questions.

Setup and Installation

1. Clone the Repository:

```bash
git clone <repository-url>
cd mcq-generator
```

2. Install Dependencies:

```bash
pip install -r requirements.txt
```

Create a `requirements.txt` file with the following content:

```
streamlit
requests
python-dotenv
openai
sentence-transformers
elasticsearch==7.17.9
```

3. Set Up Environment Variables:

Create a `.env` file in the project root with the following variables:

```env
AZURE_OPENAI_ENDPOINT=<your-azure-openai-endpoint>
AZURE_OPENAI_API_KEY=<your-azure-openai-api-key>
ELASTICSEARCH_HOST=<your-elasticsearch-host>
ELASTICSEARCH_PORT=<your-elasticsearch-port>
```

4. Set Up Elasticsearch:

Ensure an Elasticsearch instance is running (default: `http://localhost:9200`). The application automatically creates an index (`mcq_questions`) if it does not exist.

5. Run the Application:

```bash
streamlit run app.py
```

Replace `app.py` with the name of your main Python file containing the Streamlit code.

Usage

Generating MCQs

1. Access the Streamlit Interface: Open the application in a browser (typically `http://localhost:8501`).

2. Enter Parameters:

- Topic: Specify the subject or topic (e.g., "Python Programming").

- Number of Questions: Choose between 1 and 100 questions.

- Difficulty Level: Select Easy, Medium, or Hard.

- Question Type: Choose Conceptual, Factual, Problem-solving, or Scenario-based.

- Problem-solving Filters (if Problem-solving is selected): Select specific subtypes (e.g., Debugging, Time complexity).

3. Generate MCQs: Click the "Generate MCQs" button.

- The system generates MCQs, saves them to `question_prompt.txt`, converts them to JSON, checks for duplicates using Elasticsearch, and saves unique questions to `unique_mcqs.json`.

- Success or error messages are displayed.

Fetching Question Banks

1. Select Domain: Choose between LTI or Neowise.

2. Enter Token: Provide the Examly API authorization token.

3. Search Query(optional): Enter a search term to filter question banks.

4. Search Question Banks: Click the "Search Question Banks" button.

- The system fetches question banks from the Examly API and displays them as radio buttons.

- Select a question bank to store its ID for importing.

Importing MCQs

1. Ensure a Question Bank is Selected: Either select a question bank from the fetched list or manually enter a Question Bank ID.

2. Import MCQs: Click the "Import MCQs to {domain}" button.

- The system imports questions from `unique_mcqs.json` to the selected question bank via the Examly API.

- Displays the number of successful and failed uploads.

Key Components

MCQ Generation

- Function: `generate_mcqs(topic, num_questions, difficulty, question_type, selected_filters=None, max_retries=3)`

- Description: Uses the Azure OpenAI API to generate MCQs based on user inputs. It creates a meta-sorting plan to ensure diverse questions and uses few-shot examples for consistency.

- Process:

1. Validates inputs (difficulty, question type).

2. Loads instructions and examples from JSON files (`question_type_instructions.json`, `difficulty_definitions.json`, `few_shot_examples.json`).

3. Generates a meta-sorting plan to outline sub-topics and question structures.

4. Constructs an enhanced prompt with guidelines, examples, and the meta-sorting plan.

5. Sends the prompt to the Azure OpenAI API and returns formatted MCQs.

- Output: A string containing MCQs in the specified format, separated by `---`.

Question Bank Management

- Functions:

  - `get_all_qbs(token, search=None, page=1, limit=100)`: Fetches question banks for LTI.

  - `get_all_qbs_neowise(token, search=None, page=1, limit=25)`: Fetches question banks for Neowise.

- Description: Retrieves question banks from the Examly API using a POST request with predefined department IDs and headers.

- Output: JSON response containing question bank details or `None` on failure.

Elasticsearch Integration

- Class: `QuestionBank`

- Description: Manages question storage and duplicate detection using Elasticsearch and sentence embeddings.

- Key Methods:

  - `__init__`: Initializes the Elasticsearch client and sentence transformer model.

  - `_create_index_if_not_exists`: Creates the `mcq_questions` index with mappings for question data and vectors.

  - `add_unique_questions(questions)`: Adds questions to Elasticsearch, skipping duplicates based on sentence similarity.

  - `question_exists(question_data, options)`: Checks if a question exists using a phrase match query.

  - `find_similar_questions(query, num_results=5)`: Finds similar questions using cosine similarity.

- Output: Unique questions and the number of duplicates skipped.

API Integration

- Functions:

  - `import_mcqs_to_examly(input_file, qb_id, created_by, token)`: Imports MCQs to LTI.

  - `import_mcqs_to_neowise(input_file, qb_id, created_by, token)`: Imports MCQs to Neowise.

- Description: Sends MCQs from a JSON file to the Examly API's MCQ creation endpoint.

- Process:

  1. Reads questions from the input file (`unique_mcqs.json`).

  2. Removes unnecessary fields (e.g., `question_vector`).

  3. Posts each question to the API with appropriate headers.

  4. Tracks successful and failed uploads.

- Output: Tuple of successful and failed upload counts.

File Structure

- `app.py`: Main Streamlit application file (assumed name).

- `question_prompt.txt`: Stores generated MCQs in text format.

- `unique_mcqs.json`: Stores unique MCQs in JSON format.

- `.env`: Environment variables for API keys and Elasticsearch settings.

- `question_type_instructions.json`: Instructions for different question types.

- `difficulty_definitions.json`: Definitions for difficulty levels per question type.

- `few_shot_examples.json`: Example MCQs for each question type and difficulty.

Error Handling and Logging

- Logging: Configured with `logging.basicConfig(level=logging.DEBUG)` to capture detailed logs for debugging.

- Error Handling:

  - API requests include try-except blocks to handle `requests.exceptions.RequestException`.

  - MCQ generation retries up to three times on failure.

  - Elasticsearch operations log errors and raise exceptions for critical failures.

  - Streamlit displays user-friendly error messages for invalid inputs or failed operations.

- Log Output: Includes errors, warnings, and info messages for API calls, file operations, and question processing.

Limitations

- Hardcoded Values: Department IDs and `created_by` UUID are hardcoded, limiting flexibility.

- API Dependency: Relies on Examly API availability and correct token authentication.

- Elasticsearch Setup: Requires a running Elasticsearch instance, which may be complex to configure.

- Question Quality: Dependent on the Azure OpenAI API's ability to generate relevant and accurate MCQs.

- File-based Storage: Uses local files, which may not scale for large question sets.

Future Enhancements

- Dynamic Department IDs: Allow users to input or fetch department IDs dynamically.

- Customizable Created By: Enable users to specify the `created_by` UUID.

- Database Storage: Replace file-based storage with a database for scalability.

- Advanced Duplicate Detection: Incorporate more sophisticated similarity metrics.

- Batch Import: Support batch API calls to improve import performance.

- Question Preview: Add a preview feature for generated MCQs before importing.