**Project Overview and Documentation**

**Overview:**

This project consists of a set of Python scripts designed to facilitate intelligent interactions with a GitHub repository. The primary functionalities include cloning a repository, tokenizing its codebase, and leveraging tokenized information for contextualized search and question-answering. The project utilizes external libraries for natural language processing, document indexing, and machine learning model interactions.

**File\_processing Documentation:**

**Purpose:**

The `file\_processing.py` script serves as the backbone for repository interaction. It encompasses functionalities for cloning a GitHub repository, loading and tokenizing its codebase, creating a BM25Okapi index, and searching for relevant documents based on user queries.

**Dependencies**:

- **`os`:** Operating system interactions.

- `**uuid`:** Generates universally unique identifiers.

- `**subprocess`:** Allows interaction with the command line to clone GitHub repositories.

- `**sklearn`:** Provides TF-IDF vectorization and cosine similarity calculation.

- **`rank\_bm25`:** Implements the BM25 ranking algorithm.

- `**langchain`:** Contains document loaders and text splitters for various file types.

- `**nltk`:** Natural Language Toolkit for text processing.

- **`re`:** Regular expressions for text cleaning.

**Installation:**

Ensure dependencies are installed using `pip install [dependency]`. Git is required for GitHub cloning functionality.

**Configuration:**

Minimal configuration is needed. Ensure paths and configurations are correct, especially for `utils.py`.

Usage:

```python

# Example Usage

github\_url = "https://github.com/example/repository.git"

local\_path = "/path/to/local/repository"

# Clone GitHub repository

clone\_success = clone\_github\_repo(github\_url, local\_path)

if clone\_success:

# Load and index files

index, split\_documents, file\_type\_counts, source\_paths = load\_and\_index\_files(local\_path)

# Search for documents

query = "your search query"

search\_results = search\_documents(query, index, split\_documents, n\_results=5)

# Process search results as needed

for result in search\_results:

print(f"File ID: {result.metadata['file\_id']}, Source: {result.metadata['source']}")

```

**Workflow Description:**

1. **Clone GitHub Repository:**

- The `clone\_github\_repo` function clones a specified GitHub repository to a local path.

**2. Load and Index Files:**

- The `load\_and\_index\_files` function loads and indexes files from the local repository.

- Tokenization is performed, and the BM25Okapi index is created.

**3. \*\*Search Documents:\*\***

- The `search\_documents` function takes a query, searches for relevant documents using a combination of BM25 and TF-IDF with Cosine Similarity scores, and returns the top results.

**Code Structure:**

**- \*\*GitHub Cloning:\*\***

- `clone\_github\_repo`: Clones a GitHub repository using the `git clone` command.

**- \*\*File Loading and Indexing:\*\***

- `load\_and\_index\_files`: Loads and indexes files from a local repository, creating a BM25Okapi index.

**- \*\*Document Searching:\*\***

- `search\_documents`: Performs a search based on a query using BM25 and TF-IDF with Cosine Similarity, returning the top results.

**Important Code Snippets:**

**- Tokenization and BM25 indexing of documents:**

```tokenized\_documents = [clean\_and\_tokenize(doc.page\_content) for doc in split\_documents]

index = BM25Okapi(tokenized\_documents```

**- Combining BM25 and Cosine Similarity scores:**

```python

combined\_scores = bm25\_scores \* 0.5 + cosine\_sim\_scores \* 0.5

```

**`questions.py` Documentation:**

**Overview:**

questions.py` handles user questions, maintains conversation context, and interfaces with language models for question-answering.

Classes and Functions:

**QuestionContext:**

- A class representing the context for asking questions, including the index, documents, and relevant information about the GitHub repository.

**ask\_question:**

- A function that takes a user question and a `QuestionContext` and returns an answer using the provided LLM chain.

`utils.py` Documentation:

**Overview:**

`utils.py` contains utility functions for text processing and document formatting.

**Functions:**

**clean\_and\_tokenize:**

- Cleans and tokenizes text for further processing.

**format\_documents:**

- Formats a list of documents into a numbered list.

**format\_user\_question:**

- Cleans and formats a user's question for processing.

**`config.py` Documentation**:

**Overview:**

`config.py` contains configuration variables for the script.

**Variables:**

**WHITE, GREEN, RESET\_COLOR:**

- ANSI escape codes for terminal text color.

**model\_name:**

- The name of the language model to be used in the script.

`main.py` Documentation:

**Overview:**

`main.py` is the main entry point of the script.

**Functions:**

**- \*\*main():\*\***

- The main function that orchestrates the interaction with the GitHub repository, user questions, and document processing.

**Configuration:**

- The script requires an OpenAI API key, which can be set as an environment variable.

**Usage:**

- Run `main()` to start the interactive question-answering session. The script will prompt for the GitHub repository URL, clone the repository, and initiate a conversation loop.

**Dependencies:**

- Ensure all dependencies from `file\_processing.py`, `questions.py`, and `utils.py` are installed.

**License:**

**This script is released under the [MIT License](LICENSE).**

**Contact Information:**

For questions, feedback, or assistance, please contact me at [chinmayadas2003@rediffmail.com].