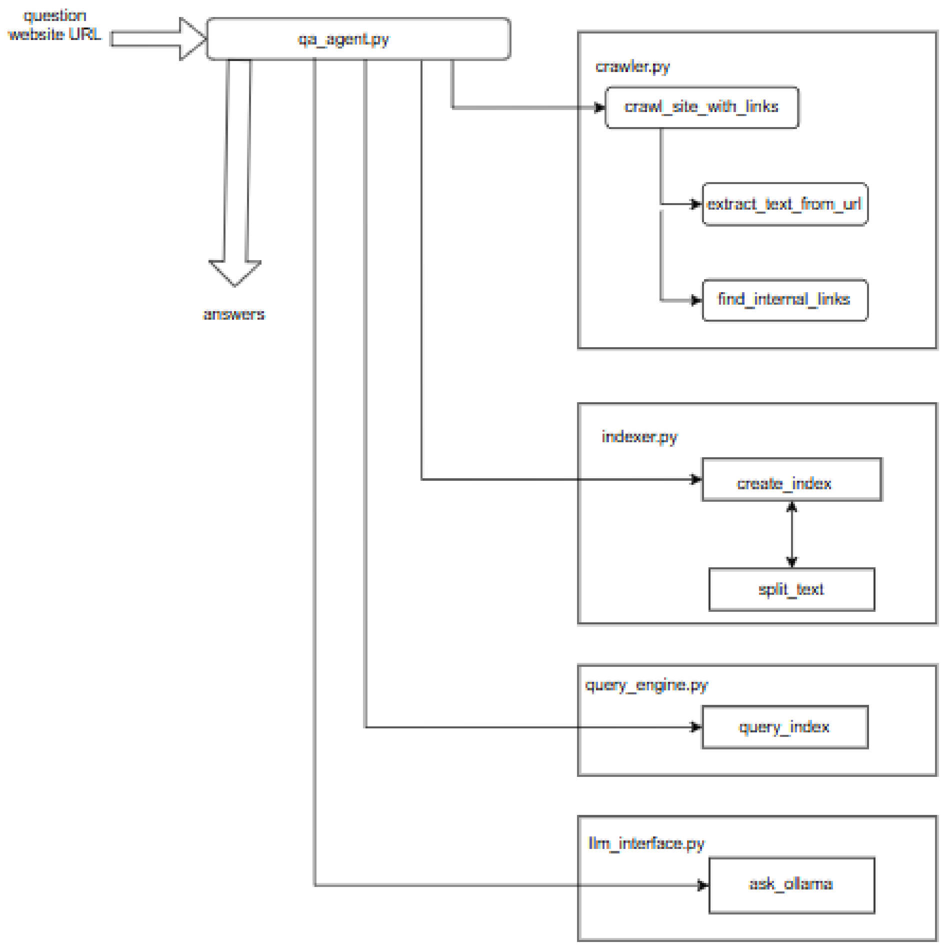
Documentation

Technical architecture overview



1. **Crawler**: Crawls and extracts content from a help website.
2. **indexer**: Embeds chunks into vectors using SentenceTransformer and Splits text into meaningful chunks (200 words) so we have two functions here
3. Query Engine: This queries our question from the embedded model
4. **LLM Answering**: Uses Ollama- mistral to answer questions with context.
5. **QA Agent**: takes the website as input, Calls the necessary function, receives the input question and call LLM answering to output the answer

Implementation Approach

1. **Crawler**: Crawls and extracts content from a help website. This has 3 function inside.
   a) The crawl_site_with_link function crawls across the base_url website and then calls the find_internal_link function.
   b) The find internal link function,finds upto 50 internal urls (because max pages was set to 50) and returns it to the crawl_site_with_link function.
   c) The crawl_site_with_link function then runs a loop for all the 50 urls and call the extract_text_from url function to extract text for each url which results in a dictionary in the form of {url:content}

2. **indexer**: Embeds chunks into vectors using SentenceTransformer and Splits text into meaningful chunks (200 words) so we have two functions here
   The split_text function to split the text into meaningful chunks and the create_index function to embed these chunks and build the vector using FAISS
3. **Query Engine**: This queries our question from the embedded model (all-MiniLM-L6-v2)
4. **LLM Answering**: Uses Ollama- mistral to answer questions with context. We need to give a well-structured prompt so that the LLM can fetch the correct contexctual data and cite the document with the URL where ever required.
   We have also set a default answer that the LLM need to return in case the document does not contain the information for the question that was asked.
6. **QA Agent**: takes the website as input, Calls the crawler.py and indexer,py, receives the input question and call LLM which interns call the query engine.py and that returns the answering according to the prompt to output the answer.

Future improvement suggestions:

1. To use a more persistent vector storage (e.g., SQLite + FAISS), instead of just FAISS.
2. Support for OpenAI or Claude APIs as alternatives to Ollama, as those model are better in Accuracy.
3. Add web-based interface using Flask or Fast API for better interface.
4. Support for more HTML tags (tables, code blocks) to read info from tables as well.
5. Implement async crawling for high speed