Wikipedia Question Answering System Development Documentation

# Development Notes

This document provides detailed explanations of the various modules and their design decisions in the Wikipedia Question Answering System.

# Overview

The Wikipedia Question Answering System is designed to answer user queries by fetching relevant information from Wikipedia. The system is divided into several core modules, each of which plays a specific role in processing user input, querying Wikipedia, and handling feedback. Here's an overview of how the modules work together:

- **Database Initialization**: This module sets up the SQLite database, which is used to store user queries and their feedback. The database allows us to track the history of user interactions and analyze performance metrics over time.

- **Concept Extraction**: This module extracts the main concept from the user's query. The system uses OpenAI's language model to identify the core topic of the query, ensuring that the search focuses on the most relevant information.

- **Wikipedia Querying**: Once the main concept is extracted, the system queries Wikipedia using the Wikipedia API. If the query is ambiguous (i.e., multiple relevant Wikipedia pages exist), the system prompts the user to select the most relevant topic from a list.

- **Handling Ambiguity**: If multiple Wikipedia pages match the user's query (e.g., "Java" could refer to the programming language or the island), this module presents the possible options to the user, allowing them to disambiguate their query.

- **Relevance Scoring**: In cases where multiple potential answers exist, the system calculates the relevance of each option using similarity scoring. This ensures that the most relevant page is presented to the user first.

- **User Feedback Mechanism**: After providing an answer, the system collects feedback from the user to determine whether the provided information was helpful. This feedback is stored in the database and used to calculate accuracy metrics such as Precision, Recall, and F1 Score.

- **Displaying Performance Metrics**: The system computes and displays key performance metrics based on user feedback, which helps track how well the system is performing over time.

- **Downloading Feedback Data**: Users can download the feedback data as a CSV file for further analysis, allowing for long-term performance tracking.

By combining these modules, the system delivers accurate and relevant answers from Wikipedia, handles ambiguous queries effectively, and continuously improves based on user feedback.

## 1. Database Initialization

| **def** **initialize\_db**():  # Initialize SQLite database |
| --- |

**Reasoning**: We use an SQLite database to store user queries and feedback. SQLite is a lightweight, serverless database that requires no additional setup, making it an ideal choice for small-scale projects like this. Storing user feedback allows the system to track performance over time and improve based on user interactions.

* **Why SQLite?**: It is simple, integrated with Python, and doesn’t need a separate server, which reduces the project’s complexity.
* **Future Consideration**: In a production-level system with heavy traffic, we might switch to a more robust database like PostgreSQL or MySQL.

## 2. Configure Wikipedia settings

| wikipedia.set\_lang('en') wikipedia.set\_user\_agent('WikiQuestionAnsweringBot/1.0 (your\_email@example.com)') |
| --- |

**Reasoning:** The reason for using **WikiQuestionAnsweringBot** is that it allows direct use of Wikipedia's built-in QA system, which helps better understand the user's questions.

## 3. Concept Extraction Using OpenAI

| **def** **get\_concept\_from\_llm**(question: str) -> str:  # Extract main concept using OpenAI |
| --- |

**Reasoning:** To improve the accuracy of user queries, we use OpenAI to extract the main concept. However, if an API Key is not provided, since we are already using the WikiQuestionAnsweringBot, it should theoretically be able to provide answers that align with user needs. This is simply an attempt to use LLM to enhance the accuracy of the query.

In fact, I also considered using spaCy's NER recognition feature to extract the Named Entities from the user's question, thereby improving the accuracy of the QA system. However, considering that if the user's question is more complex, the number of Named Entities might be large, which could make it harder for the QA system to identify the main focus. Therefore, I decided not to use it. In this small project, I don't have much time to delve into this issue, but perhaps it could be a direction for future research.

## 4. Handling Ambiguity

| **def** **present\_multiple\_entities**(entities):  # Present multiple possible entities to the user for selection |
| --- |

**Reasoning**: In cases where a user query is ambiguous (e.g., “Java”), the system may extract multiple entities from the question. These entities could relate to different Wikipedia pages. Instead of guessing the correct interpretation, we ask the user to select the most relevant entity.

* **User Interaction**: By letting the user choose between multiple entities (like "Java Island" or "Java Programming Language"), the system ensures that the final result is accurate and aligned with user expectations.
* **Disambiguation Design**: This strategy reduces errors in cases where multiple valid answers exist and shifts some control to the user to clarify their intent.

## 5. Searching Wikipedia and Handling Disambiguation Errors

| **def** **search\_wikipedia**(query: str) -> list:  # Use Wikipedia API to search for the query and return a list of relevant pages |
| --- |

**Reasoning:** The wikipedia.search() function is used to find relevant pages based on the user’s query. However, Wikipedia can return disambiguation pages if the query is too broad.

| **def** **get\_page\_summary\_cached**(page\_title: str) -> str:  # Retrieve Wikipedia page summary for the selected page, with caching |
| --- |

**Handling Disambiguation**: If the search leads to a disambiguation page (multiple possible meanings), the system will prompt the user to choose the right interpretation. This helps avoid returning incorrect or vague answers.

**Why Caching?**: To avoid repeated Wikipedia API calls for the same query, we use caching to store previously fetched results. This reduces API load and improves response time for frequent queries.

## 6. Relevance Scoring and Sorting

| **def** **calculate\_similarity**(question: str, options: list) -> dict:  # Use cosine similarity to compare the user query with possible Wikipedia options |
| --- |

**Reasoning**: When there are multiple potential answers, the system needs a way to rank the relevance of each option. Using SentenceTransformer, the system calculates cosine similarity between the user’s query and the possible Wikipedia options.

* **Why SentenceTransformer?**: It’s a powerful library that converts text into embeddings, allowing for semantic similarity comparisons. This helps the system return the most relevant Wikipedia page, even if the exact keywords don’t match perfectly.

| **def** **calculate\_relevance\_batch\_llm**(question: str, options: list) -> dict:  # Use OpenAI to assign relevance scores to Wikipedia options |
| --- |

**Fallback to OpenAI**: If available, OpenAI’s GPT model is used to directly assess the relevance of each option. This provides even better accuracy in understanding nuanced queries.

## 7. User Feedback Mechanism

| **def** **collect\_feedback**(query\_id: int):  # Collect user feedback on the usefulness of the system's answer |
| --- |

**Reasoning**: The system allows users to provide feedback on the answers it generates. This feedback is stored in the database and used to compute accuracy metrics like Precision, Recall, and F1 Score.

* **Why Collect Feedback?**: Feedback is essential to improving the system’s performance. By knowing which answers were helpful and which were not, the system can learn and adjust its behavior over time.

| **def** **update\_feedback**(query\_id: int, feedback: str):  # Update the feedback status for a given query in the database |
| --- |

**Feedback Storage**: Each user query and the corresponding feedback are stored in SQLite. This allows the system to track how well it performs and adjust future responses based on user input.

## 8. Displaying Performance Metrics

| **def** **display\_metrics**():  # Calculate and display Precision, Recall, and F1 Score based on feedback data |
| --- |

**Reasoning**: To evaluate the system's performance, we calculate three key metrics: Precision, Recall, and F1 Score. These metrics are computed based on the user feedback stored in the database.

* **Precision**: Measures the proportion of correct answers out of the total number of provided answers.
* **Recall**: Measures the proportion of correct answers out of the total number of queries.
* **F1 Score**: The harmonic mean of Precision and Recall, providing a balanced view of the system’s overall performance.
* **Why Metrics Matter?**: These metrics help quantify the system’s effectiveness and highlight areas that need improvement. Regularly calculating these scores ensures the system can track its progress and maintain high-quality responses.

## 9. Downloading Feedback Data

| **if** st.button("💾 Export Feedback Data"):  # Export feedback data as a CSV file for analysis |
| --- |

**Reasoning**: The system allows users to export all stored feedback data as a CSV file. This feature is useful for analyzing the system’s performance over time, particularly if multiple users or long-term data collection is involved.

* **Why Export Data?**: Allowing data export ensures transparency and flexibility. Users can analyze the feedback in external tools and track performance outside the system.

# Conclusion:

These development notes explain the core design and rationale behind the system’s features. The primary goal is to ensure flexibility, accuracy, and the ability to scale or adapt the system for different use cases. Each module is designed with the user's needs and feedback in mind, ensuring a robust and responsive question-answering experience based on Wikipedia.