INFS2044 Assignment 2 Case Study

In this assignment you will be developing a simple question answering system. In this system, users can enter questions in natural language, and the system will attempt to determine what the question was and answer it. We will consider a simplistic version of such a software that is restricted to retrieving answers based on the similarity between the entered question and the questions stored in a catalogue of question-answer pairs.

Use Cases

The system supports the following use cases:

- UC1 Welcome: The system starts and displays a welcome message to the user.
- UC2 Ask Question: The user enters a question as a single sentence in plain English, and the system stores the question, looks up the most similar question in its question bank and returns the question and its associated answer to the user.
- UC3 Quit: The user enters a blank line instead of a question. The system displays a goodbye notice and terminates execution.

UC2 repeats until UC3 terminates the system.

Example Transcript

The following is an example transcript of a user's interaction with the system:

Hello, I am a question answering bot.

Please enter a question, and press the ENTER key:

which day is it?

I think that you asked 'What day is today?' and conclude that the answer is 'Monday'.

Please enter a question, and press the ENTER key:

How is the weather like?

I think that you asked 'What is the weather like today?' and conclude that the answer is 'Same as yesterday.'.

Please enter a question, and press the ENTER key:

Goodbye!

Other Requirements

- R01: The system shall interact with the user through a text console (Terminal on MacOS/Linux and Command Prompt on MS Windows).
- R02: The system shall store logged question phrases in a plain text format in the file named “asked_questions_log.txt”, one question per line.
- R03: The system shall read its catalogue of know questions and corresponding answers from the file named “faq.json”. The format of this file is described in section “Question Catalogue File Format” below.
- R04: The system shall determine similarity among question phrases as described in section “Similarity Calculation” below.

Similarity Calculation

The most similar question is determined as follows. Suppose the user enters question text, E, and the system considers a stored question text, Q. The similarity between E and Q is determined by the number of words that appear jointly in E and Q, relative to the total number of words that occur in E or Q. Let W_E and W_Q denote the set of distinct words in E and Q, respectively. Then, the similarity between E and Q can be calculated as $\frac{|W_E \cap W_Q|}{|W_E \cup W_Q|}$; that is, divide the number of distinct words that appear in both E and Q by the number of distinct words that appear in either E or Q. This yields a number in the range [0..1], where the similarity is zero if E and Q have no words in common, and where the similarity is one if E and Q share the same set of words. For example, consider the phrases E=“are red apples also apples?” and Q=“are apples and raspberries red?”. The set of words W_E is {also, apples, are, red}, and W_Q is {and, apples, are, raspberries, red}. There are three words that appear in both sets: apples, are, red. There are six unique words in total. Hence, the similarity between E and Q is $\frac{3}{6} = 0.5$. This similarity measure is known as the *Jaccard Similarity Score*.

In your program, use the question that maximises the Jaccard similarity with the question phrase entered by the user. If there are multiple questions that are equally similar, use any one of these questions to generate the answer.

Word boundaries are determined by whitespace and punctuation. For example, there are four words in the phrase “this simple-looking program?”: this, simple, looking, program.

Words are to be compared in *case-insensitive* manner. That is, “Are”, “are”, and “aRe” etc are all considered to be the same word. Moreover, all punctuation is to be omitted when comparing words.

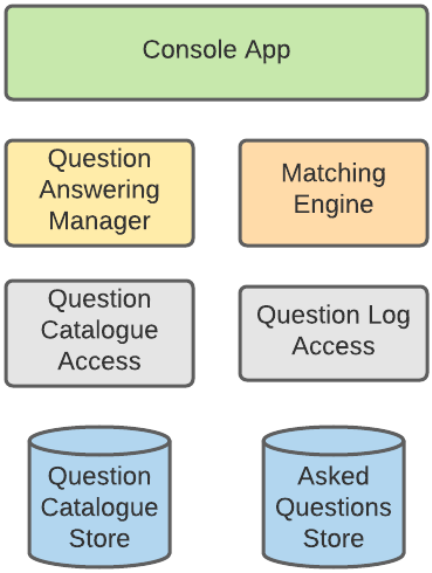
Question Catalogue File Format

The catalogue of questions and corresponding answers is stored in JSON format containing a list of dictionary objects, each corresponding to a candidate question-answer pair. The question phrase is stored under key “question”, and the corresponding answer is stored under key “answer” in the dictionary object for each entry. The following example illustrates a catalogue comprising two entries:

```
[
  {
    "question": "What day is today?",
    "answer": "Monday"
  },
  {
    "question": "What is the weather like today?",
    "answer": "Same as yesterday."
  }
]
```

Decomposition

You must use the following component decomposition as the basis for your implementation design:



The responsibilities of the components are as follows:

Component	Responsibilities
Console App	Interact with the user (acquire user input, and display the system response)
Question Answering Manager	Orchestrates the use case UC02
Matching Engine	Computes the most similar question-answer pair for a given question text
Question Catalogue Access	Retrieves questions that include one or more of a given set of words from the question catalogue store
Question Log Access	Stores the asked questions in the Asked Questions Store
Question Catalogue Store	Holds the known questions-answer pairs
Asked Questions Store	Holds all the questions entered by the user

Assumptions

This decomposition has been created based on the volatility assumptions that in future:

- more sophisticated matching engines may be developed, and
- question catalogue store and asked questions store may be implemented using (different) database technologies.

Scope

Your implementation must respect the boundaries defined by the above decomposition and include classes for each of the components in this decomposition (except the two store components, which are realised as files as per requirements R02 and R03).

The implementation must:

- run on python 3, and
- correctly implement the given use cases, and
- it must function correctly with any given faq.json in the correct format (you can assume that the entire content of this file fits into main memory); and
- it must include a comprehensive unit test suite using pytest.

Focus your attention on the *quality* of your code.
Speed and memory efficiency are not primary concerns for purposes of this assignment.

To give an indication about the amount of code that you will write: a comprehensive and largely self-documenting implementation of the system described here can be achieved in less than 200 lines of python code (including whitespace and the code that is given to you). This number does not include the code for the unit tests.