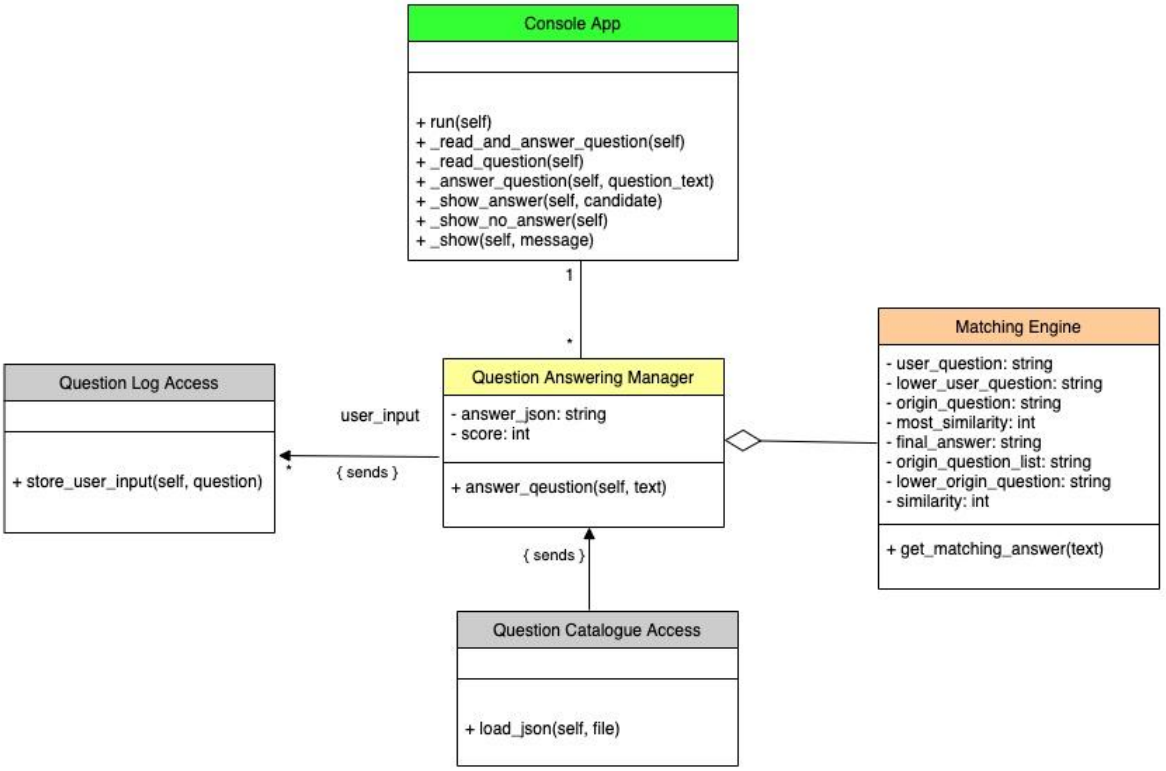
Assignment 2 Case Study

INFS 2044 – Software Implementation and design

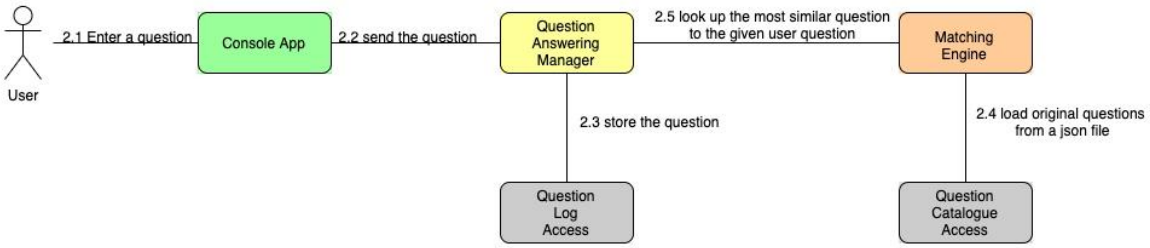
Name: Erika Hosokawa
Student ID: 110308922
Email address: hosey001@mymail.unisa.edu.au

Class-Level Design

- UML Class Diagram



- Decomposition Validation



The console app is the interface between a user and the system. The question answering manager receives user inputs from the console app. The user inputs are stored through the question log access into a text file (asked_question.txt). The matching engine looks for an appropriate question and answer pair to the given user question. The engine returns the pair based on the user input and through the question catalogue access from a json file (faq.json) where the system stores original question and answer pairs.

- Assumption

My design has consisted of five classes and two storages based on decompositions. Classes have to be small pieces as much as we consider because it makes us change easily when we have got issues. Thus, I have created those classes and storages for the system.
I have assumed that the question answering manager communicates with other classes (Console App, Question Log Access and Matching Engine) such as receiving user inputs, storing user questions and finding a similar question and the answer from a json file to the given question from a user.

Quality Assurance

- Approach to quality assurance

I have approached my design and source code to maintain the loose coupling and high cohesion. I have decided the Question Answering Manager should work in the middle of the console app and other classes. Thus, the manager is located in the centre in the diagram. In addition, two access classes have each storage as they store different elements and support the high cohesion. My classes do not conflict with each other as my design is created based on loose coupling and high cohesion. I have described more about two principles in the design principles and patterns section.

- Approach to test-driven development of the program

The test should confirm if all functions I have created work. I have tried to make the test code to cover errors as much as I can expect.
test_get_matching_answer(): The method looks into if the system has a similar question and answer pair to the user input. If so, they provide the pair to the Question Answering Manager. And if not, they provide empty value. I have tested two different expected types.
test_load_json(): The system holds some questions. The system expects what questions they would be asked by a user. And they store the questions in the faq.json file. The Matching Engine needs to know what questions the system has. Thus, this method loads the json file to let the Matching Engine know. The test_load_json has the test.json file just for the test.
test_store_user_input(): When the Question Answering Manager gets user inputs, the Question Log Access receives the input and stores it into the Asked Questions Store. It does not return anything. It just holds the input in a text file. Thus, assert expects it is None.
test_manager(manager): It tests the answer_question method in InteractiveConsoleClient class.

Reflection

- Major changes

I have not changed much for my design and my code. However, I had created some functions in Matching Engine to have a look at the json file. My first code tried to get values (question and answer) from the array in the json file. And I created two loops for that. I realised that it is not reusable and readable codes at all as I had repeated the same codes for that. Therefore, I have changed that part to adopt the loose coupling. The function for Matching Engine I have created is just one which is get_matching_answer(text) eventually. In addition, the question catalogue access and the question log access had a specific file name in each method (load_json and store_user_input) to store or extract data from there. However, in the test drive, it is not met with scalability if the file names are changed at some point. For that, I have changed those methods to be able to accept the change. They have the argument which is “file” to manage different files.

- Design principles and patterns

As I mentioned before, I have created my design and source code (Classes I have created such as Question Answering Manager, Matching Engine, Question Log Access and Question Catalogue Access) based on the loose coupling and the high cohesion.
In the loose coupling case, if the json file in the Question Catalogue Access class is changed to a text file, it does not influence Matching Engine. I can add or recreate the load_json function to be able to return the same format as the json file. Then, Matching Engine receives the same format from the access. On the other hand, if my code is tight coupling, I have to change a lot because each class is strongly linked.
In the high cohesion case, I have considered that classes should not have included a lot of functions in one class. Thus, my design is composed of five classes that do not have uncommon functions. It means that each class just has included the functions related to the responsibilities.