Programming Assignment 1
Inf-8320

Abul Ahsan Md Mahmudul Haque
Department of Computer Science
University of Tromsø

Introduction

This assignment consists of few parts, which includes: building a distributed quiz object, adding persistency to that object, and finally implementing a restful version of the object. The remote object provides interfaces with various methods like start(), question(id), result(id), etc. Figure 1, represents the available interfaces and class variables of the quiz object. The quiz application has been built with the help of "Pyro (Python Remote Object)" library, which takes care of locating the right object to execute the remote methods. Besides, "web.py" framework is used to make the quiz object as a restful service.

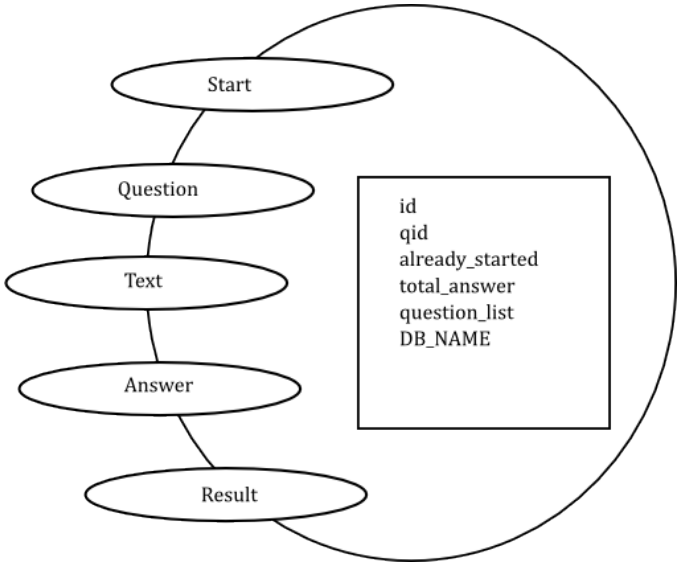

Figure 1: Class interface diagram of *Quiz* object

Design

This quiz application has two parts: one should be executed at the client site and another should be running at server site. All of the methods of the remote object have been implemented at the server site. Server module maintains one database to store the quiz questions and answers. When first time the object is invoked then server module executes the sql commands to create and store the questions and answers into the database if it is not already there. The definition of the database is below:

```
Create table qbank (ques_no integer primary key auto increment not null,
                    statement text,
                    option_one text,
                    option_two text,
                    option_three text,
                    option_four text,
                    right_answer text
                    )
```

From the client site, when it invokes the quiz object then server module checks that any quiz object has already been initiated and incomplete or not. If it's already been there then server module starts with the same set of questions and returns with the rest of the questions. Otherwise, it randomly selects a set of questions and returns questions; finally it sends the overall score to the client.

For keeping the persistency, the server site maintains one log file that keeps track of the quiz object and starts the quiz game from the remaining set of questions if it remains as incomplete.

In case of restful implementation, the quiz object uses the "web.py" framework. There the quiz object has the following URL mapping:

```
urls = ('/quiz', 'quiz',
    '/quiz/(.*)', 'question',
        …
        …
    '/lastquiz', 'lastquiz',
    )
```

So from the client browser user may provide the address of the "quiz" resource then it either may select a random set of questions or return with the set that has already been started and incomplete. It has also an interface where the user can give input like "/quiz/set number"; if the given set number is valid then the server will return that particular set of questions, otherwise it returns appropriate error message. Moreover, user can invoke "lastquiz", which redirects to an incomplete quiz or simply replies that there are no remaining questions in the last attempted quiz.

Implementation

This assignment is accomplished using different python data structures and several other modules and platforms. Client invokes the methods of the remote object using the Pyro4 middleware. Initially at server site, "Quiz" object has been registered with the Pyro´s name server and client automatically gets a proxy object by resolving with the appropriate object name. The server site has predefined sets of questions with set numbers and the question numbers from the question database. While a client starts quiz then the server site choose a random set number from the set of questions. For simplicity, here I have used only three sets of questions where each set contains three questions.

The log file appends the set number of the questions and the answers of the earlier questions. If the server site requires to resume from an earlier session then at first server module checks the set number, then it checks the number of the questions from that set has already been answered; finally it determines the rest of the questions from the question set and send those question to the clients.

In restful version, it mostly follows the same application logic. Initially by default the server starts at "0.0.0.0:8080" location and some url mappings are used to provide the right pages to the client. The source file is placed in the src folder and templates for the html files are placed in the templates folder. When the client starts a new quiz, then it checks for any incomplete objects and invokes the post

method of the question class to create a form with the questions and possible answers. When the client submits answer then immediately server site checks the answer and let the client know that answer is right or wrong and the at the end the client receives the number of right answer as the final result. The question class also has a get method and it is invoked when the client send a request for a particular set.

Discussion

One of the good things about this assignment is that there were few options and we had the liberty of choosing any of those platforms/mechanisms. Thus we have at least looked at different alternatives and tried one of those. I use “Pyro4” as an object-oriented middleware platform as its powerful, fast, easy to use, and it has comparatively low overhead. In case of adding persistency, I use the file system approach for adding persistency as its also simple and serve the purpose in this context. However, shelve module could also be an alternative here. In case of implementing restful version, I use the web.py framework as its nice and powerful way of letting every http verbs to execute accordingly. However, “django” could also be a good alternative in this aspect. Since here this application does not directly deal with any database; thus, the delete and update verbs are not used with the http requests.

Here, persistency is only added to the quiz object. It could have been more realistic to add persistency on client perspective too, which indicates that adding persistency to users on every quiz they participate. So if a user left a quiz at some point of time then only next time the user will resume from that particular state. For simplicity we only maintain the persistency on the quiz object.

There might be few bugs in the implementation of the quiz game; one obvious is in restful version while the client checks multiple options from the checkboxes, the server object cannot distinguish it from the single selection and fails to generate the appropriate error message.