**COMP 2005 – Final Project**

Weicai Zhao

Vladimir Andrade

Uvwie Omafume

Clifford Bowe

*A description (up to two pages plus diagrams) of the project, describing main functionality and software architecture, depicting each module and its responsibilities.*

## *Project Description*

*The client is the instructor for COMP2005 course at the Computer Science Department, Memorial University of Newfoundland*

*The project description presented by the client is as follows:*

*The COMP2005 course makes use of a proprietary online quiz application integrated into the Brightspace product and supported by Memorial's CITL unit. However, the COMP2005 instructor would like to develop his own online quiz program, which could later be extended to include software code testing and live classroom quiz features. The initial release of this project may be limited to MCQ questions and online asynchronous (i.e. not live classroom) quiz.*

*Each project team is a group of up to 5 students registered in COMP2005 for the Winter 2019 semester.*

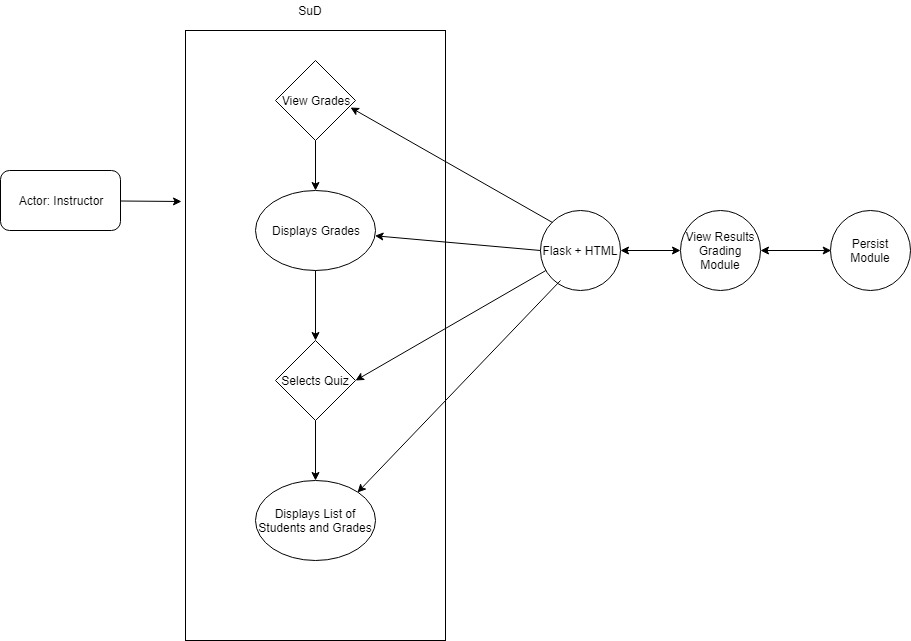
*Project architecture and functional description of each module is outlined in*[*lecture note 14*](http://www.cs.mun.ca/~brown/cs2005/Notes/notes-14.html)*. An initial*[*functional requirements list*](http://www.cs.mun.ca/~brown/cs2005/Notes/project/projrequirements.html)*will also be provided for reference; your team should modify this list to your own use cases and needs.*

*A description (up to two pages plus diagrams) of the project, describing main functionality and software architecture, depicting each module and its responsibilities.*

**Main Functionality**

The project implements an alternative to the Brightspace provided online quiz application. It supports authentication of various levels of access (student and instructor) and gives different options to the user, based on their current level of access. Instructors will have the option to create a quiz, which will take them through the process of designing their own, multiple choice quiz to be stored and accessed later. The instructor can also view results (specifically, the results that are for the instructor only to view, such as class participation or the grade histogram for a given quiz). The student will have the option to view quizzes, which will take them to a list of quizzes that are within the system and then allow them to take a quiz, if they are permitted to. They will also have the option to view results, specific to them.

**Use Case UML Diagram**



**Clear Connection between Module and Functionality**

The basis of my implementation is going to be the “\_quiz\_dict” dictionary member variable of my QuizResult object. When called, it will take the dictionary from Persist, supplied by the Take Quiz module, and iterate over it, adding each attempt’s information to the \_quiz\_dict dictionary, with the quiz names as keys and a list with the attempt data (student email, list of questions, list of choices, list of correct answers and list of student responses) for each individual attempt

Per the requirements list (<http://www.cs.mun.ca/~brown/cs2005/Notes/project/projrequirements.html>) each of my functions is tailored to satisfy a specific requirement, which should be clear from the naming of the function. For example, the ins\_class\_part function is meant to return the class participation as a percentage value to the instructor. The stu\_quiz\_grades function is meant to return a list of strings representing a summary of the student’s grades for a specific quiz. The naming convention makes the connection clear. The flask front end will then use these functions to display the appropriate data as needed.

**Description of Module Design**

The module is designed such that the information detailed in the mandatory requirements list is all linked to an individual function. For example, for the instructor to view the class average of a quiz, the ins\_class\_avg function exists and will be called to return the data for the histogram as a list. It will then be displayed through the flask and HTML based front end as required.

For the purposes of implementation, ToD would simply need to verify that the dictionary being received from the Persist module is of the proper format below:

Take\_Quiz\_Export\_Dict

Keys: Student Emails

Values: Dictionaries

Keys: Quiz Names

Values: Quiz Attempt Objects

The quiz attempt object structure is laid out in the Take Quiz documentation.

The flask front end, once fully constructed, will simply call the functions as needed to display the required data for the given web pages.