Programmer 3: TAKE QUIZ COMPONENT

*Note: Deliverables 2, 4, 6 and 7 have been adjusted*

2.A description (up to one page plus diagrams) of your assigned programming component for the project, per lecture note 14. Describe the functionality of your component in terms of the overall project.

**MAIN FUNCTIONALITY IN RELATION TO OTHER COMPONENETS**:

(quiz is created by instructor and sent to persist module by the *Create Quiz module*)

The **Take Quiz** module does the following:

-Gets the quiz selected by the student from the persist interface.

-Allow student to take the quiz.

-Returns the students attempt on the quiz back to the persist interface.

(the students quiz-attempts await retrieval for grading by the *Automatic Grading Module*)

NOTE FOR DIAGRAM:

*Diagram shows the functionality of Take Quiz module in relation to the other components. Curved lines return data to persistence. Straight lines collect data from persistence*

**CARRIYING OUT FUNCTIONALITY AS PER LECTURE NOTE 14**:

1. Open the quiz chosen by the student:

* + - Verifies that the student has access to chosen quiz.
    - Retrieve chosen quiz from persistence.
    - Send quiz content to flask code to display on website (quiz name, quiz questions, quiz choices)

2. Record all answers chosen by the student:

* + - Save student's response to each question he/she answers.
    - Save this answered/submitted quiz as a quiz-attempt for this student

3. Send quiz-attempt to persist for storage (to await grading):

* Store student quiz attempt in a dictionary
  + - * store this quiz attempt in a temporary list
      * add temporary list as a value to dictionary1 with the quiz-name as key.
      * add dictonary1 as value to dictionary2 with student user-name as key
* Call persist to store dictionary2

3.One use case describing the principal user interaction with your project component.

**Title:** Take Quiz

**Primary Actor:** Student

**Stakeholders and Interests:**

Student: wants the quiz recorded

Instructor: wants quiz submission

**Precondition**: Student logged in to quiz selection page.

**Main Scenario**:

1. SuD presents quizzes available to Student
2. Student selects quiz
3. SuD presents any quiz special instructions
4. Student starts quiz
5. SuD presents quiz question
6. Student enters answer
7. SuD saves and advances to next question
8. Student repeats 5-8 as desired
9. Student submits quiz solution
10. SuD confirms quiz submission

**Extensions**:

6a. Suspend Quiz

6a1. Student suspends quiz session

6a2. SuD maintains answers for subsequent quiz session

10a. Submission rejected

10a1. SuD says quiz incomplete

10a2. Student forces incomplete submission

Variation: 6b. Student selects any other quiz question

Variation: 6c. Student changes previous answer

4. Clear connection between your module and its responsibilities from a project requirements list.

**Requirement Header 1.3**: User can take a Quiz

**Link to requirements List used**: <http://www.cs.mun.ca/~brown/cs2005/Notes/project/projrequirements.html>

**Requirement1**: Users can take permitted Quiz, Access Limitations are checked

**Method that solves requirement**: checkAccess()

Check Access:

* Student must be logged in.
* Student must be listed as part of student to take the quiz.
* The max attempts for the quiz must not be already exceeded by student.
* The deadline for the quiz must not be exceeded.

**Requirement2**: Users can have multiple attempts.

**Method that solves requirement**: Take quiz constructor\_\_init\_\_()

Constructor:

* Creates a new attempt for that quiz when someone tries to take a quiz again.

**Requirement3**: Users can pause quiz.

**Method that solves requirement**: stopQuiz(), resumeQuiz()

Pause Quiz and Resume Quiz:

Student presses the pause quiz button to attend to an urgent situation

* stopQuiz() adds the students incomplete quiz-attempt to the student quiz-attempt dictionary.
* The dictionary is then persisted
* resumeQuiz() resets the present quiz-attempt to the incomplete quiz-attempt when the user decides to continue the quiz.

5.  pydoc and PEP257 compatible docstrings for your module

* Modules have been documented with their docstrings.

6. Clearly defined interfaces/exports for your component suitable for its project functionality responsibilities

1. Design documentation clearly illustrating appropriate decoupling and separation of concerns

Design Choice to aid project functionality

The Take-Quiz module imports the persist. The persist class instantiates a persist object, *storage*, which is used by all other components/module to access the methods in persist. Storage helps to pull and push information to the shelve.

The persist class has a class variable *classList*.

* Class list: A pool of students that the instructor can grant access to a quiz.

The Take-Quiz module also imports *quiz-attempt* class.

* Quiz-attempt class: It holds information of a quiz taken by a student.

Design choice to aid module responsibilities

The Take Quiz Module uses three (3) class methods*, resumequiz(), store\_studentsQA(), get\_studentsQA()*. They are class methods because they specifically handle the class-variable dictionary, *studentsQA*. They respectively help to retrieve an incomplete attempt, persist the dictionary to storage, and retrieve the persisted dictionary from storage.

The *get\_studentsQA()* method was made mainly for testing the *store\_studentsQA()* method in the *test\_take\_quiz* test file.

All other methods are clearly documented with their docstrings.

The Take Quiz Module has a function *quizzesInStorage()*. It returns a list of all the names of the available quiz in the persist. It is sent to the flask front end code to display the quizzes that are available for taken.

7. A description of your module design (up to one page plus diagrams) providing any additional needed implementation details suitable for a programmer that may need to maintain the module.

**IMPLEMENTATION DETAIL TO NOTE BY TOD:**

1. In the *\_\_init\_\_()* method of the TakeQuiz class it retrieves a tuple containing (create-quiz object, quiz-object). It then separates them into two separate instance variables for the module’s usage. The create quiz object is used only once to access the list of the students who can take that quiz.
2. In the *\_\_init\_\_()* method of the TakeQuiz class the self.\_presentAttempt variable is overwritten if the user has an incomplete attempt.
3. The *studentsQA* (student Quiz Attempts) is a dictionary within a dictionary. It holds the information of every student, their quizzes, and all their attempts on those quiz.

{Key: username

Value: dictionary (Key: quizName,

Value: List[] of quizAttempt objects} }

1. Whenever TakeQuiz is called it will create a new quizAttempt object to hold students attempt information. if there is an incomplete quiz attempt, It will reset this quizAttempt object to the incomplete quizAttempt object so the information can continued to be recorded on the incomplete attempt until it is completed.

8. Instructions to install and run your submitted test code in a venv environment workable on the CS lab configuration; create a README file for this purpose.

* README file in folder

9. [20 points] unit tests for functionality of your component interface

1. a minimum of two-unit tests for each exported function or method; one test for correct success mode, and one test for correct failure mode
2. a minimum of ten tests overall
3. use python standard unit-test module
4. tests must run but do not have to pass for this assignment

Test Files

* test\_take\_quiz: This file is the unit-test for the TakeQuiz class. It has 10-unit test.
* test\_quiz\_attempt: This is the unit-test for the quiz-attempt class in the common\_classes. It has 5-unit test.

10.[5 points] stubs: exported/interface methods do not have to be fully implemented but must provide stubs which accept correct parameters and return correct data types according to your component interface. Stubs must be consistent with your unit tests and project components of teammates.

* Stubs are in place