Technical Guide

Quiz Manager

By:
Jonathan Sadighian
Rhea Moubarak

February 20, 2019

1. Introduction

1.1. Project Background

This project is a console application and GUI for managing digital quizzes - preparation and execution. This document is prepared by Rhea Moubarak and Jonathan Sadighian for their Fundamental Java Project in the Computer Science Master's Program at L'École Pour l'Informatique et les Techniques Avancées (EPITA).

1.2. Project Overview

In this project, user can login as a student, take a test and get their grades at the end of the quiz. In addition, user can login as a teacher, create questions and quizzes.

1.3. Project Scope

In Scope :

1. Automatically assembles quizzes using open questions and multiple-choice questions
2. Auto-grading for multiple choice questions
3. Creating data access object with CRUD methods
4. Create a configurable properties file for the application
5. Export quizzes to plain text

Out of Scope :

6. Incorporating Associative questions into the GUI
7. Exporting quizzes to PDF
8. Using an algorithm to arrange quizzes
9. Stylizing the GUI
10. Creating different user accounts other than Student
11. Creating user accounts with logging in
12. Providing real questions for the Quiz Manager application

1.4. Project Dependencies

In order to run the program, the following steps are required:

- Install JavaFx
- Install h2 JDBC
- Install Java 8
- Create tables in database
- Add questions

1.5. Acronyms and abbreviations

Acronyms	Meaning
CRUD	(Create, Read, Update, Delete) Functions that are implemented in relational database applications
GUI	(Graphical User Interface) Interface presented to the user of an application
DAO	(Data Access Object) Service class for the application to communicate with the H2 database
UML	(Unified Modeling Language) a standard way to visualize the design of a system
MCQ	(Multiple Choice Question) Question with enumerated possible answers, implemented with radial buttons in our application
OQ	(Open question) Question with a free-text field response and no predefined structured answer
JDBC	(Java Database Connectivity) application programming interface (API) for the programming language Java

2. Business Requirements

BR#	Requirement Description
1	Create a quiz based on user's specifications (topics)
2	Use questions stored in database (using h2 database)
3	Allow a user to take a quiz
4	Correct MCQ questions and display result at the end of the evaluation
5	Export this quiz under a plain text format
6	Search questions by topic or difficulty
7	Use CRUD operations on a question (create, read, update, or delete a question)

8	Use existing default credentials to take a quiz
9	3 types of questions: associative, multiple choice, and open questions.
10	GUI is available with the application (using JavaFx)

3. System Specifications

3.1. Functional Requirements

BR#	Implementation
1	**Users can customize their quiz when instantiating the Quiz object, which takes a list of topics as one of the parameters.** public Quiz(Student student, List<String> topics, int totalQuestions, Boolean includeOpenQuestions) { this.student = student; this.topics = topics; this.totalQuestions = totalQuestions; this.dao = new QuestionJDBCDAO(); this.random = new Random(); String temp = ""; for (String topic : this.topics) { temp += " " + topic; } this.title = this.student.getName() + " |" + temp; this.usedQuestions = new LinkedList<Question>(); this.availableQuestions = new LinkedList<Question>(); this.includeOpenQuestions = includeOpenQuestions; }
2	**Users can use the questions that are already used and stored in the database.** public MultipleChoice getNewQuestion() { // extract a specified difficulty List<MultipleChoice> temp = this.availableMCQuestions.stream().collect(Collectors.*toList*()); int size = temp.size(); if (size < 1) { System.***out***.println("getNewQuestion --> ran out of questions");

	return null; } MultipleChoice newQuestion = temp.remove(this.random.nextInt(size)); if (!this.availableMCQuestions.remove(newQuestion)) { System.***out***.println("getNewQuestion --> Warning! Question not found in available list."); } return newQuestion; }
3	**Users can take a quiz by inserting their own credentials; name, id, topics, number of questions, with a choice of enabling/disabling open questions.** Student student = new Student(textName.getText(), textId.getText()); List<String> topics = new LinkedList<String>(Arrays.asList(textTopics.getText().split(","))); int numberOfQuestions = Math.max(Integer.parseInt(textQuizSize.getText()), 1); this.quiz = new Quiz(student, topics, numberOfQuestions); this.quiz.loadNewQuiz();
4	**Users can have his quiz evaluated and results displayed after submission.** this.buttonSubmitAnswer = new Button(); this.buttonSubmitAnswer.setText("Submit Answer"); this.buttonSubmitAnswer.setPadding(new Insets(5, 5, 5, 5)); this.buttonSubmitAnswer.setOnAction(a -> { System.out.println("Submit button clicked"); this.currentQuestion.setChoice(getUserChoice(labelOp1, labelOp2, labelOp3, labelOp4)); this.quiz.processNewQuestion(this.currentQuestion); updateQuizUI(labelQuestionNumber, labelQuestion, labelOp1, labelOp2, labelOp3, labelOp4); this.progressBar.setProgress(this.quiz.getProgress());

	if (this.quiz.getProgress() >= 1.0) { name.setText("Student: " + this.quiz.getStudent().getName()); grade.setText("Correct: " + Integer.toString(this.quiz.getGrade())); questionCount.setText("Total: " + Integer.toString(this.quiz.getUsedMCQuestions().size())); this.window.setScene(sceneSummary); } }); public void processNewQuestion(Question question) { question.setNumber(this.usedQuestions.size() + 1); question.gradeAnswer(); this.usedQuestions.add(question); if (question.getIsCorrect()) { this.grade += 1; } this.progress = (double)this.usedQuestions.size() / (double)this.totalQuestions; System.out.println("this.progress is now : " + this.progress); }
5	**Users can export their quiz as a plain text.** public void exportAll(Quiz quiz, Boolean includeSummary) { if (includeSummary.booleanValue()) { this.write("Student", quiz.getStudent().getName()); this.write("Correct", quiz.getGrade()); this.write("Total Questions", quiz.getTotalQuestions()); this.write("Topics", quiz.getTopics().toString()); } for (Question question : quiz.getUsedQuestions()) { this.writeQuestion(question); }

	}
6	**Users can search questions by difficulty and topics.** public MultipleChoice getNewQuestion(int difficulty) { // extract a specified difficulty List<MultipleChoice> temp = this.availableMCQuestions.stream() .filter(q -> q.getDifficulty() == difficulty) .collect(Collectors.*toList*()); int size = temp.size(); if (size < 1) { System.*out*.println("getNewQuestion --> ran out of questions"); return null; } MultipleChoice newQuestion = temp.remove(this.random.nextInt(size)); if (!this.availableMCQuestions.remove(newQuestion)) { System.*out*.println("getNewQuestion --> Warning! Question not found in available list."); } return newQuestion; } public List<MultipleChoice> search(List<String> topicSearch) { List<MultipleChoice> resultList = new LinkedList<MultipleChoice>(); String SEARCH_STATEMENT_TOPICS = "SELECT * from MCQ WHERE"; int numberOfTopics = topicSearch.size(); if (numberOfTopics < 1) { System.out.println("Search incomplete. Please add topics."); return resultList; } for (int i=0; i<numberOfTopics; i++) { SEARCH_STATEMENT_TOPICS += " ARRAY_CONTAINS(TOPICS, '" + topicSearch.get(i) + "')";

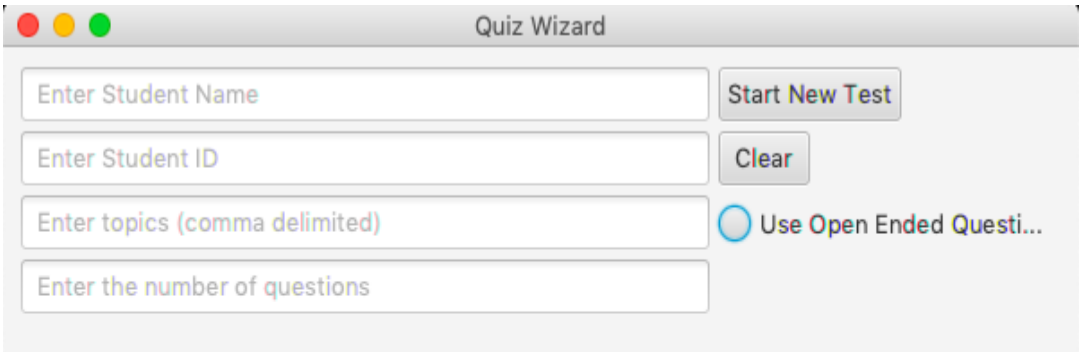
```
                    if (i + 1 < numberOfTopics) {
                        SEARCH_STATEMENT_TOPICS += " OR";
                    }
                }

                try (Connection connection = getConnection();
                    PreparedStatement preparedStatement =
connection.prepareStatement(SEARCH_STATEMENT_TOPICS); ) {
                    ResultSet results = preparedStatement.executeQuery();

                    while (results.next()) {
                        MultipleChoice mc = new MultipleChoice();
                        mc.setQuestion(results.getString("QUESTION"));
                        mc.setDifficulty(results.getInt("DIFFICULTY"));
                        Array array = results.getArray("TOPICS");
                        Object[] topics = (Object[]) array.getArray();
                        List<String> topicsList = new
LinkedList<String>();
                        for (int i=0; i<topics.length; i++) {
                            topicsList.add((String)topics[i]);
                        }
                        mc.setTopics(topicsList);
                        mc.addOption(results.getString("OP_1"));
                        mc.addOption(results.getString("OP_2"));
                        mc.addOption(results.getString("OP_3"));
                        mc.addOption(results.getString("OP_4"));
                        mc.setAnswer(results.getInt("ANSWER"));
                        mc.setId(results.getInt("ID"));
                        resultList.add(mc);
                    }
                    results.close();
                    System.out.println("Searched for TOPICS=" +
topicSearch.toString() + ".  Found " + resultList.size());
                }
                catch (Exception e) {
                    e.printStackTrace();
                }

                return resultList;
```

	}
7	**Users can create, read, update, and delete a question.** **//Create** public void create(MultipleChoice question) { try (Connection connection = getConnection(); PreparedStatement insertStatement = connection.prepareStatement(*INSERT_STATEMENT*);) { insertStatement.setString(1, question.getQuestion()); insertStatement.setInt(2, question.getDifficulty()); Array array = connection.createArrayOf("VARCHAR", question.getTopics().toArray()); insertStatement.setArray(3, array); insertStatement.setString(4, question.getOptions().get(0)); insertStatement.setString(5, question.getOptions().get(1)); insertStatement.setString(6, question.getOptions().get(2)); insertStatement.setString(7, question.getOptions().get(3)); insertStatement.setInt(8, question.getAnswer()); insertStatement.execute(); array.free(); System.*out*.println("Created question : " + question.toString()); } catch (SQLException e) { e.printStackTrace(); } } **//Read** public List<MultipleChoice> read() { List<MultipleChoice> resultList = new LinkedList<MultipleChoice>(); try (Connection connection = getConnection(); PreparedStatement readStatement = connection.prepareStatement(*READ_STATEMENT*)){ ResultSet results = readStatement.executeQuery(); while (results.next()) { MultipleChoice mc = new MultipleChoice(); mc.setQuestion(results.getString("QUESTION"));

```
                        mc.setDifficulty(results.getInt("DIFFICULTY"));

                        Array array = results.getArray("TOPICS");
                        Object[] topics = (Object[]) array.getArray();
                        List<String> topicsList = new LinkedList<String>();
                        for (int i=0; i<topics.length; i++) {
        topicsList.add((String)topics[i]);
                        }
                        mc.setTopics(topicsList);
                        mc.addOption(results.getString("OP_1"));
                        mc.addOption(results.getString("OP_2"));
                        mc.addOption(results.getString("OP_3"));
                        mc.addOption(results.getString("OP_4"));
                        mc.setAnswer(results.getInt("ANSWER"));
                        mc.setId(results.getInt("ID"));
                         resultList.add(mc);
                 }
                results.close();

                System.out.println("Read questions. Found " +
resultList.size());
        }
        catch (SQLException e) {
                e.printStackTrace();
        }
        return resultList;
}

//Update
public void update(MultipleChoice question) {
        try (Connection connection = getConnection();
                PreparedStatement updateStatement =
connection.prepareStatement(UPDATE_STATEMENT)){
                updateStatement.setString(1, question.getQuestion());
                updateStatement.setInt(2, question.getDifficulty());
                Array array = connection.createArrayOf("VARCHAR",
question.getTopics().toArray());
                updateStatement.setArray(3, array);
                updateStatement.setString(4, question.getOptions().get(0));
                updateStatement.setString(5, question.getOptions().get(1));
```

	updateStatement.setString(6, question.getOptions().get(2)); updateStatement.setString(7, question.getOptions().get(3)); updateStatement.setInt(8, question.getAnswer()); updateStatement.setLong(9, question.getId()); updateStatement.execute(); System.*out*.println("Updated question : " + question.toString()); } catch (SQLException e) { e.printStackTrace(); } } **//Delete** public void delete(MultipleChoice question) { try (Connection connection = getConnection(); PreparedStatement deleteStatement = connection.prepareStatement(*DELETE_STATEMENT*)){ deleteStatement.setInt(1, question.getId()); deleteStatement.executeUpdate(); System.*out*.println("Deleted question : " + question.toString()); } catch (SQLException e) { e.printStackTrace(); } }
8	**Users can use default credentials to take a quiz by clicking on start new test.** this.buttonCreateTest = new Button(); this.buttonCreateTest.setText("Start New Test"); this.buttonCreateTest.setPadding(new Insets(5, 5, 5, 5)); this.buttonCreateTest.setOnAction(a -> { System.out.println("Start Button clicked"); if (textName.getText().equals("")) { textName.setText("Bob Smith"); } if (textId.getText().equals("")) { textId.setText("12345");

	} if (textTopics.getText().equals("")) { textTopics.setText("France,Italy"); } if (textQuizSize.getText().equals("")) { textQuizSize.setText("5"); }
9	**Quiz can have three different types of questions; associative, MCQ, and open questions.** **//Associative** private int answer; private int choice; private Map<Integer, String> choices; public Associative() { super(); this.setChoices(new HashMap<Integer, String>()); } **//MCQ** private int answer; private int choice; private List<String> options; public MultipleChoice() { super(); this.setOptions(new LinkedList<String>()); } **//Open** private String response; public Open() { }
10	**User is able to run the application using desktop GUI.** Run QuestionManagerApplication.jar



3.2. Non-functional Requirements

3.2.1. The program shall be able to run on GUI using JavaFx library.

3.2.2. The program shall be able to read a configuration property set from a file on the filesystem which will avoid hardcoded parameters.

3.2.3. The data is stored in h2 databases.

4. User Interface Design

Refer to the User Guide.

5. Hardware Requirements

#	Hardware	Requirement
1	Operating System	Compatible with Windows, Mac OS X, Linux
2	RAM	Minimum required 124MB
3	Disk Space	Minimum required 124MB
4	Processor	64-bit, four-core, 2.5 GHz minimum per core

6. Appendix

UML Class Diagram