QUIZ APPLICTION TECHNICAL SPECIFICATION

1. Project Overview
2. Technical Requirements
3. Activity Diagrams
4. Database Schema
5. Structure of the program
6. General flow of the core methods
7. Code snippets and explanation

PROJECT OVERVIEW:

The quiz program has two users – a teacher and a student.

A teacher can create quizzes and questions. Two types of questions can be created – open questions and MCQ Questions.

A student can take a quiz based on topics or choose a quiz from a list of quizzes. At the end of the quiz, the MCQ Score is displayed.

The answers for the open questions are recorded in the answer table.

The quiz is exported to a file QuizExport.txt

TECHNICAL REQUIREMENTS:

To use this program these are the requirements:

Java JRE 1.8 and JDK 1.8

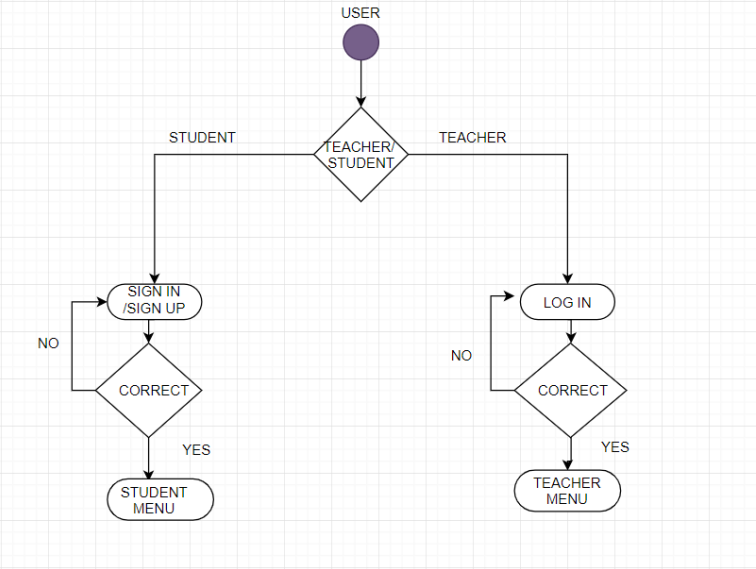
An IDE (Eclipse) as it is a console application

H2 database

ACTIVITY DIAGRAMS

User login/Sign up.

The application has 2 types of users – Student and Teacher



An existing student logs in. A new student signs up.

A teacher logs in. The teacher credentials are:

Login : ADM

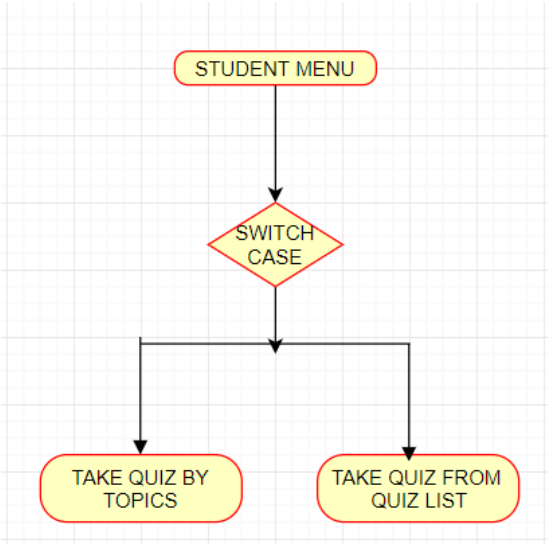
Password : ADM

These values are stored in the conf.properties file.

STUDENT ACTIVITIES:

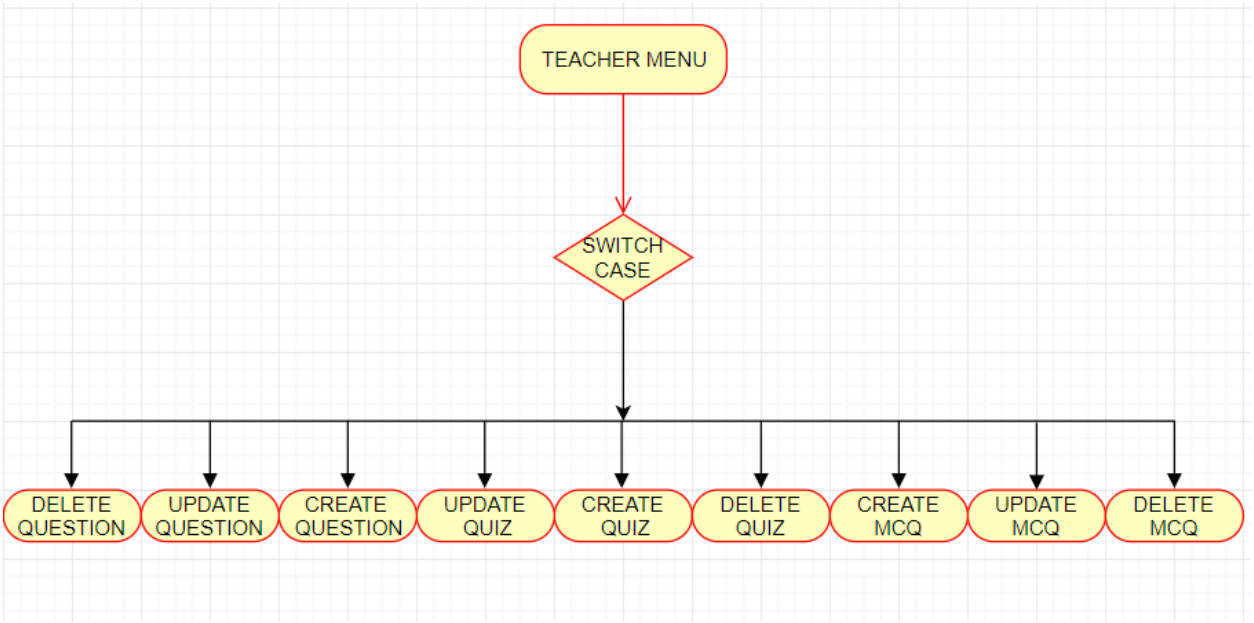
After a student is successfully logged in, the student menu is displayed.

A can take a quiz based on topics or choose a quiz from a list of quizzes.

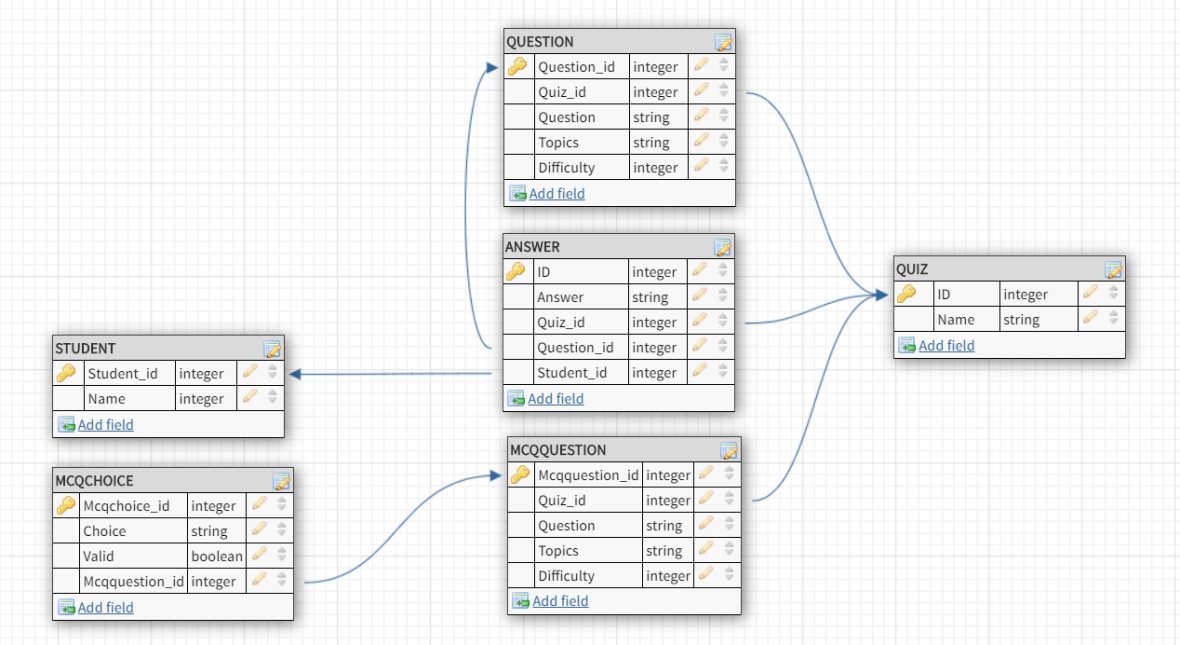


TEACHER ACTIVITIES:

After a teacher successfully logged in, the teacher menu is displayed.
A teacher can create, update or delete a quiz/question (both open and MCQ)



DATABASE SCHEMA



The application uses 6 tables
All the tables have a primary that is auto incremented.

TABLES:

Quiz table – The quiz table stores the ID and the name of the quizzes.
Primary Key : ID

Question Table – The question table stores the Question Id, question, topics and the difficulty.
Primary Key : Question_id
Foreign Key : Quiz_id (References Quiz)

Answer Table – The answer table stores the ID, answer
Primary Key – ID
Foreign Keys :
Quiz_id (References Quiz)
Question_id (References Question)
Student_id (References Student)

MCQQuestion Table – The MCQQuestion table stores the Mcqquestion_id, question, topics and the difficulty.
Primary Key – Mcqquestion_id
Foreign Keys:
Quiz_id (References Quiz)

MCQChoice Table – The MCQChice table stores the MCQChoice_id, choice,valid
Primary_key – MCQChoice_id

Student Table – The student table stores the student_id , name
Primary Key – student_id

STRUCTURE OF PROGRAM

There are 7 packages under the src folder:

DATAMODEL:

The datamodel package has the classes for the entities – Quiz, Question, Answer, MCQQuestion, MCQQuiz and Student.

It has has an Enum Difficulty.java
Difficulty has the accepted levels of difficulty for the open and MCQ Questions.

LAUNCHER:

The launcher package contains the welcomePage.java class.
It has the main method.

On successful authentication,
It calls the studentMenu() for a student
I calls the teacherMenu() for a teacher

USERPAGES:

The userpages package contains two classes –
StudentDisplay.java for student
TeacherDisplay.java for teacher

TeacherDisplay.java :
Displays the teacher menu
This class also contains the methods to do the teacher activities like create/update/delete. The read function is used where needed.
After the teacher enters the choice, it calls the corresponding method to perform the function.

StudentDisplay.java :
Displays the student menu
This class also contains the methods to do the student activities like take quiz based on topics or quiz list.
After the student enters the choice, it calls the corresponding method to perform the function.

SERVICE:

The service package contains the JDBCDAO classes for the entities to connect to the database. It also has an Enum configEntry that has the keys for the values in the conf.properties file.

The configurationService class has the methods to get the value.
These files ensure everything is configurable and prevent hardcoding.

It contains the FilesUtil.java class that has the methods to export the quiz to a file.

TESTS:

The tests package contains the test classes to test the JDBCDAO connections.

EXCEPTION:

This package contains the exception classes like DataAcessException, CreateFailedException, etc.

FILES:

This package contains the QuizExport.txt file. This file is used to export a quiz in plain text format.

FOLDER SQL:

It contains a file with the SQL queries. These queries have to be executed in the same order to create the tables.

CONF.PROPERTIES FILE:

The conf.properties file has the parameters and the sql queries used.

FLOW OF THE METHODS

Teacher activities:

Create, Delete and Update a Quiz/ Question (both Open and MCQ)

For creating a quiz/question, the general flow of the method is:

1. Gets the quiz/ question from the user.
2. Checks if it already exists.
3. If yes, it displays a message “Already exists” and terminates
4. If no, gets the additional information if necessary (topics, difficulty for questions) and creates the record in the table.
5. For an MCQ question, the MCQ choices are created. There must atleast 2 choices and one valid choice.

For Updating a quiz/ question, the general flow of the method is:

1. Displays the existing quizzes / questions.
2. User enters the ID of the quiz/question to be updated.
3. If the ID is not valid, it displays “Invalid ID” and terminates.
4. If the ID is valid, the user enters the updated quiz / question.
5. The quiz / question gets updated in the table.

For Deleting a quiz / question, the general flow of the method is:

1. Displays the existing quizzes / questions
2. User enters the ID of the quiz / question to be deleted
3. If the ID is not valid, it displays “Invalid ID” and terminates.
4. If the ID is valid, the quiz / question is deleted from the table.
5. For deletion, to preserve referential integrity :

   For a quiz :

   The open questions and the answers that belong to the quiz are also deleted.
   The MCQ questions and their choices that belong to the quiz are also deleted.
   Then the quiz is deleted.

   For an open question :

   The answers that belong to the question are deleted.

   For an MCQ Question:

   The choices that belong to the MCQ Question are deleted.

Student Activities:

After successful login, a student can:

1. Take a quiz based on the topics
2. Take a quiz from a list of quizzes.

To take a quiz based on the topics:

1. The user enters the topics separated by commas.
2. These topics are split by the comma delimiter.
3. Relevant questions are fetched by comparing the topics field of the question table and the user entered topic.
4. Sometimes, a question can be fetched twice as it might satisfy two topics.
   Example : Id multiple inheritance supported in java?
   This question satisfies both java, inheritance topics.
5. To prevent a question from being fetched twice, the question IDs are stored in a hash set. Every time a new question is fetched, it is checked if it already exists. If, no it is added to the question list.
6. The questions are displayed one at a time to the user. The user types in the answer for the open question.
7. The user enters the choice number for the MCQ Question.
8. The answers for the MCQ Questions are evaluated and the score is tracked using a counter.
9. The MCQ score is displayed at the end of the quiz.
10. The quiz is also export to the QuizExport.txt file in the files package.

To take a quiz from a list of quizzes:

1. The list of the available quizzes is displayed.
2. The user chooses the quiz
3. If the quiz number is valid, the questions from that quiz are displayed one at at time.
4. The procedure for answering the open question and the MCQ questions are same as above.

These core activities use other methods like generate(), checkIfExists(), setTopics(), setDifficulty() methods.

EXCEPTIONS:

The package Exception has exception classes. In case of a failure to connect to the database, an error message is displayed and the program is terminated.

```
1
Enter name:
Hagrid
Problem connecting to the database
Please check connections and try again
```

CODE SNIPPETS

TEACHER ACTIVITIES

1. **Create a quiz:**

   To create a quiz, the createQuiz() and checkIfTitle() methods are used.

   When the user chooses to create a quiz, it calls the createQuiz() method.

```
public void createQuiz() throws DataAccessException
{
  boolean titleExists;
  String title;
  Quiz quiz = new Quiz();
  System.out.println("Enter the quiz title");
    title = scanner.nextLine();
    quiz.setTitle(title);
    titleExists = checkIfQuizTitleExists(quiz);
    if (!titleExists) {
      quizdao.createQuiz(quiz);
    }
    else {
    System.out.println("Already exists");
    }
}
```

The createQuiz() method takes in the user input. It checks if the quiz title already exists using the checkIfTitleExists() method.If it exists, it displays " Already exists". Else, it creates the quiz using the createQuiz() method.

2. **Update a quiz:**

```
public void updateQuiz( ) throws SQLException, DataAccessException {
  boolean quizIDExists;
  String newTitle;
  Quiz quiz = new Quiz();
  //displays the available quizzes
  displaymenus.generateQuizList();
  System.out.println("Enter the quiz ID to be updated");
  try {
  int quizID = Integer.parseInt(scanner.nextLine());
  quiz.setId(quizID);
  quizIDExists = checkIfQuizIDExists(quiz);
  if ( quizIDExists)
  {
    System.out.println("Enter the new title");
    newTitle = scanner.nextLine();
    quiz.setTitle(newTitle);
    quizdao.updateQuiz(quiz);
  }
  else {
    System.out.println(invalidID);
  }
  }
  catch (NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
}
```

The updateQuiz() calls the generateQuizList() method to display the list of existing quizzes. The user enters the ID of the quiz to be generated. Using the checkIfQuizIDExists() method, it checks if the entered ID is valid. If yes, the user enters the new title. The quiz is updated using the updateQuiz() method.

3. **Delete a Quiz**

```
public void deleteQuiz() throws SQLException, DataAccessException
{
  boolean quizIDExists;
  Quiz quiz = new Quiz();
  displaymenus.generateQuizList();
  try {
  System.out.println("Enter the Quiz ID to be deleted");
  int quizID = Integer.parseInt(scanner.nextLine());
  quiz.setId(quizID);
  quizIDExists = checkIfQuizIDExists(quiz);
  if( quizIDExists)
  {
    answerdao.deleteAnswerByQuizId(quiz);
    questiondao.deleteQuestionByQuizId(quiz);
    List<MCQQuestion> mcqquestionlist = mcqquestiondao.readMCQQuestionsByQuizID(quiz);
    if(!mcqquestionlist.isEmpty()) {
    for(MCQQuestion mcqquestion : mcqquestionlist)
    {
      mcqchoicedao.deleteMCQChoiceByQuestionID(mcqquestion);
      System.out.println("MCQ choices deleted");
    }
    mcqquestiondao.deleteMCQQuestionByQuizId(quiz);
  }
    quizdao.deleteQuiz(quiz);
  }
  else {
    System.out.println(invalidID);
  }
  }
  catch(NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
}
```

The deleteQuiz() method calls the generateQuizList() to display the list of available quizzes. The user enters the ID of the quiz to be deleted. Using the checkIfQuizIDExists(), it hecks if the entered ID is valid. If yes, the answers, questions, mcqquestionsand its choices are deleted first. Then the quiz is deleted. This is done to preserve the referential integrity.

4. **Create an open question for a quiz**

```
public void createOpenQuestion() throws SQLException, DataAccessException,NumberFormatException {
  boolean quizIDExists;
  boolean questionExists;
  int quizId;
  String text;
  Quiz quiz = new Quiz();
  Question question = new Question();
  List<String> topics = new ArrayList<>();
  displaymenus.generateQuizList();
  try {
  System.out.println("Enter the quiz ID for the question");
  quizId = Integer.parseInt(scanner.nextLine());
  quiz.setId(quizId);
  quizIDExists = checkIfQuizIDExists(quiz);
  if(quizIDExists)
  {
  System.out.println("Enter the open question");
  text = scanner.nextLine();
  question.setContent(text);
  questionExists = checkIfQuestionExists(question);
  if(!questionExists)
  {
  //topics
  topics = setTopics();
  //difficulty
  Difficulty diff = setDifficulty();
  question.setDifficulty(diff.toInteger());
  question.setContent(text);
  question.setTopics(topics);
  questiondao.createQuestion(question, quiz);
  }
  else
  {
    System.out.println("This open question already exists");
  }
  }
```

The createOpenQuestion() calls the generateQuizList() to display the list of available quizzes. The user chooses the quiz in which the question is to added. After checking if the entered ID is valid, it prompts the user to enter the open question. If the question already exists, it displays “Already exists” ; if it doesn’t it prompts the user to enter the topics, difficulty. Using the createQuestion() method, it creates the question.

setDifficulty()

```
public Difficulty setDifficulty()
{
  Difficulty diff = Difficulty.valueOf("VERY_EASY");
  List < String > difficulty = new ArrayList <>(Arrays.asList("VERY_EASY","EASY","MEDIUM","HARD","VERY_HARD","EXTREMELY_HARD"));
  System.out.println("The Difficulty levels are:");
  for ( String s : difficulty)
  {System.out.println(s); }
  boolean difficultySet = false;
  do {
  System.out.println("Enter the difficulty");
  String d = scanner.nextLine();
  if(difficulty.contains(d.toUpperCase())) {
  diff = Difficulty.valueOf(d.toUpperCase());
  difficultySet = true;
  }
  else { System.out.println("Enter correct value"); }
  }while(!difficultySet);
  return diff;
}
```

The setDifficulty() method is used to set the difficulty for a question. It uses the Enum Difficulty. The accepted levels of difficulty are listed for the user to choose. When the user enters a valid difficulty level, the method returns the difficulty level.

5. **Update an open Question**

```
public void updateOpenQuestion() throws ReadFailedException, SQLException, UpdateFailedException {
  boolean openQuestionIDExists;
  int id;
  String updatedQuestion;
  Question question = new Question();
  displaymenus.generateQuestionList();
  try
  {
  System.out.println("Enter the question ID to be updated");
  id = Integer.parseInt(scanner.nextLine());
  question.setId(id);
  openQuestionIDExists = checkIfQuestionIDExists(question);
  if(openQuestionIDExists) {
    System.out.println("Enter the new question");
    updatedQuestion = scanner.nextLine();
    question.setContent(updatedQuestion);
    questiondao.updateQuestion(question);
  }
  else {
    System.out.println(invalidID);
  }
  }
  catch(NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
}
```

The updateOpenQuestion() method calls the generateQuestionList() display the list of available open Questions. The user enters the ID of the question to be updated. Using the checkIfQuestionIDExists() method, it checks if the ID is valid. If valid, the user enters the updated question. The question is updated using the updateQuestion() method.

6. **Delete an open question**

```
public void deleteOpenQuestion() throws ReadFailedException, SQLException, DeleteFailedException {
  boolean openQuestionIDExists;
  int id;
  Question question = new Question();
  displaymenus.generateQuestionList();
  System.out.println("Enter the question ID to be deleted");
  try {
  id = Integer.parseInt(scanner.nextLine());
  question.setId(id);
  openQuestionIDExists = checkIfQuestionIDExists(question);
  if(openQuestionIDExists) {
    answerdao.deleteAnswerByQuestionId(question);
    questiondao.deleteQuestionByQuestionID(question);
  }
  else {
    System.out.println(invalidID);
  }
}
  catch(NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
  }
```

The deleteOpenQuestion() method calls the generateQuestionList() to display the list of available questions. The user enters the ID of the question to be deleted. Using the checkIfQuestionIDExists() method, it checks if the ID is valid. If it is valid, the answers associated with the question is deleted using the deleteAnswerByQuestionID() method. Then the question is deleted using the deleteQuestionByQuestionID() method. The answers associated with the question are deleted to preserve the referential instegrity.

7. **Create an MCQ Question and its choices for a quiz**

Code snippet of the createMCQQuestion() method

```
public void createMCQQuestion() throws SQLException, DataAccessException {
  boolean quizIDExists;
  boolean mcqquestionExists;
  int quizID;
  String question;
  Quiz quiz = new Quiz();
  MCQQuestion mcqquestion = new MCQQuestion();
  displaymenus.generateQuizList();
  System.out.println("Enter the Quiz ID to add the MCQ Question in");
  try {
  quizID = Integer.parseInt(scanner.nextLine());
  quiz.setId(quizID);
  quizIDExists = checkIfQuizIDExists(quiz);
  if(quizIDExists) {
    System.out.println("Enter the question");
    question  = scanner.nextLine();
    mcqquestion.setContent(question);
    mcqquestionExists = checkIfMCQQuestionExists(mcqquestion);
    if(!mcqquestionExists)
    {
    //topics
    List <String> topics = setTopics();
    //difficulty
    Difficulty diff = setDifficulty();

    mcqquestion.setDifficulty(diff.toInteger());
    mcqquestion.setContent(question);
    mcqquestion.setTopics(topics);
    mcqquestiondao.createMCQQuestion(mcqquestion, quiz);

    MCQQuestion  mcqquestionID = mcqquestiondao.searchMCQQuestion(mcqquestion);
    int x=1;
    int validCounter = 0;
    int choiceCounter = 0;
    List<MCQChoice> mcqChoiceList = new ArrayList<>();
```

The createMCQQuestion() method calls the generateQuizList() to display the list of available quizzes. Using the checkIfQuizIDExists(), it checks if the ID the user entered is valid. If yes, the user enters the question. Using the checkIfMCQQuestionExists() method, it checks if the MCQQuestion already exists. If it does, it displays a message “Already exists..” and terminates. If it doesn’t, the user is prompted for the related topics. The difficulty level is set using the setDifficulty() method. TheMCQQuestion is created using the createMCQQuestion() method. The user is prompted to enter the choices.

```
int x=1;
int validCounter = 0;
int choiceCounter = 0;
List<MCQChoice> mcqChoiceList = new ArrayList<>();
do
{
  System.out.println("Enter the choice");
  String choiceOption  = scanner.nextLine();
  MCQChoice mcqChoice = new  MCQChoice();
  mcqChoice.setChoice(choiceOption);
  System.out.println("Enter the validity");
  Boolean validity  = Boolean.valueOf(scanner.nextLine());
  if(validity) { validCounter ++; }
  if ((validity) && validCounter > 1)
  {
    System.out.println("Enter only one valid choice");
    continue;
  }
  mcqChoice.setValid(validity);
  mcqChoiceList.add(mcqChoice);
  choiceCounter ++;
  System.out.println("More Choices? yes/no");
  String option  = scanner.nextLine();
  if  ( (option.trim().equals("no") ) &&  (validCounter <  1) )
  {
    System.out.println("Enter  one valid option");
    x = 1;
  }
  else if ((option.equals("no"))  && (choiceCounter < 2))
  {
    System.out.println("Enter atleast 2 choices");
    x =1;
  }
  else if (option.equals("yes"))
  {
```

The MCQ choices for the MCQ question are created. The user is prompted to enter a choice and its validity. The validity is a Boolean value. Using the validCounter and choiceCounter, it is ensured that there are atleast 2 choices and one valid choice.

8. Update an MCQ Question

```
public void updateMCQQuestion() throws ReadFailedException, SQLException, UpdateFailedException {
  boolean mcqQuestionIDExists;
  int id;
  String updatedQuestion;
  MCQQuestion mcqquestion = new MCQQuestion();
  displaymenus.generateMCQQuestionList();
  System.out.println("Enter the MCQ question ID to be updated");
  try {
  id = Integer.parseInt(scanner.nextLine());
  mcqquestion.setId(id);
  mcqQuestionIDExists = checkIfMCQQuestionIDExists(mcqquestion);
  if(mcqQuestionIDExists) {
    System.out.println("Enter the updated question");
    updatedQuestion = scanner.nextLine();
    mcqquestion.setContent(updatedQuestion);
    mcqquestiondao.updateMCQQuestion(mcqquestion);
  }
  else {
    System.out.println(invalidID);
  }
}
  catch(NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
}
```

The updateMCQQuestion() calls the generateMCQQuestionList() to display the list of existing MCQ Questions. The user enters the ID of the MCQ Question to be updated. Using the checkIfMCQQuestionExists() the validity of the MCQ Question is checked. If valid, the user enters the updated question. The MCQ Question is updated using the updateMCQQuestion() method.

9. Delete an MCQ Question

```
public void deleteMCQQuestion() throws ReadFailedException, SQLException, DeleteFailedException
{
  boolean mcqQuestionIDExists;
  int id;
  MCQQuestion mcqquestion = new MCQQuestion();
  displaymenus.generateMCQQuestionList();
  try
  {
  System.out.println("Enter the MCQ question ID to be deleted");
  id = Integer.parseInt(scanner.nextLine());
  mcqquestion.setId(id);
  mcqQuestionIDExists = checkIfMCQQuestionIDExists(mcqquestion);
  if(mcqQuestionIDExists) {
    mcqchoicedao.deleteMCQChoiceByQuestionID(mcqquestion);
    mcqquestiondao.deleteMCQQuestionByID(mcqquestion);
  }
  else {
    System.out.println(invalidID);
  }
}
  catch(NumberFormatException nfe)
  {
    System.out.println(invalidID);
  }
}
```

The deleteMCQuestion calls the generateMCQQuestionList() to dislay the list of available quizzes. The user enters the ID of the question to be deleted. Using the checkIfMCQQuestionExists(), it checks if the entered ID is valid. If yes, the MCQ choices associated with the question are deleted using the deleteMCQChoiceByQuestionID(). The MCQ question is deleted using the deleteMCQQuestionByID(). This is done to preserve the referential integrity.

STUDENT ACTIVITIES:

An existing user has to login. A new user has to sign up.

StudentLogin():

```
public int studentLogin() throws SQLException, IOException, SearchFailedException{
    int init = 0;
    System.out.println("Enter name:");
    String name = scanner.nextLine();
    student.setName(name);
    try {
    student = studentdao.searchStudent(student);
    if (student.getName() == null)
    {
      System.out.println("User not found. Please sign up");
    }
    else
    {
    System.out.println("Welcome  " + student.getName());
    fileContent = "Name: " + student.getName();
    fileWriter.writeToTextFile(fileContent);
    init = 1;
    }
  } catch(SQLException sqle) {
    System.out.println("Problem in logging in");
  }
    return init;
}
```

When a student tries to login, the name entered is checked if it exists. If yes, the student is logged in. If not found, the student has to sign up.

studentSignUp()

```
public void studentSignUp() throws SQLException, IOException, SearchFailedException, CreateFailedException {

    System.out.println("Enter name:");
    String name = scanner.nextLine();
    student.setName(name);
    Student studentSearchResult = studentdao.searchStudent(student);
    if ( studentSearchResult.getName()  != null)
    {
      System.out.println("Already Exists");
    }
    else {
    studentdao.createStudent(student);
    }
    student = studentdao.searchStudent(student);
    System.out.println("Welcome" + student.getName());
    fileContent = "Name: " + student.getName();
    fileWriter.writeToTextFile(fileContent);
}
```

The signup() method prompts the user to enter the name. After ensuring the name is not already present using the searchStudent() method, the student record is created in the student table.

A student can take a Quiz by:

1. Entering a list of topics

```
public void takeQuizByTopics() throws SQLException, DataAccessException, IOException, BadInputException {
  boolean questionsExist = false;
  System.out.println("Enter the topics separated by commas");
  String topics  = scanner.nextLine();
  List<Question> questionlist = questiondao.readQuestionsByTopics(topics);

  if(!questionlist.isEmpty()) {
    questionsExist = true;
  answerOpenQuestion(questionlist);
  }

  List<MCQQuestion> mcqquestionlist = mcqquestiondao.readMCQQuestionsByTopics(topics);
  if(!mcqquestionlist.isEmpty())
  {
    questionsExist = true;
    answerMCQQuestion(mcqquestionlist);
  }
  if (!questionsExist)
  {
    System.out.println("There are no questions for this topic! ");
  }
  System.out.println("END OF QUIZ");
  choiceMenu();
}
```

The student enters the topics separated by commas. The takeQuizByTopics() method calls the readQuestionsByTopics() to fetch a list of the open questions relevant to the topics. The questions are displayed one at a time. The user enters the answers. The answers are written to the table using the answerOpenQuestion() method. It then calls the readMCQQuestionsByTopics() method to fetch a list of the relevant questions. It displays them one at a time. It calls the answerMCQQuestion() method to answer the MCQ Questions. If there are no questions related to the given topics, it displays an appropriate message. This is checked using a Boolean variable 'questionsExist'. If there are questions, it is true else false.

2. Choosing a quiz from a quiz list

```
public void takeQuizByQuizList() throws SQLException, DataAccessException, IOException, BadInputException {
  Quiz quiz = new Quiz();
  System.out.println("QUIZ LIST");
  displaymenus.generateQuizList();
  try
  {
  System.out.println("Which quiz would you like to take today?");
  int quizChoice = Integer.parseInt(scanner.nextLine());
  quiz.setId(quizChoice);
  if( teacherdisplay.checkIfQuizIDExists(quiz)) {
  List<Question>questionList = questiondao.readQuestionByQuizID(quiz);
  if(!questionList.isEmpty()) {
    answerOpenQuestion(questionList);
  }

    List<MCQQuestion> mcqquestionlist = mcqquestiondao.readMCQQuestionsByQuizID(quiz);
    if(!mcqquestionlist.isEmpty()) {
      answerMCQQuestion(mcqquestionlist);
  }
    System.out.println("END OF QUIZ");
  }
  else
  {
    System.out.println("Enter Valid Quiz ID");
    takeQuizByQuizList();
  }
  }
  catch(Exception e)
  {
    throw new BadInputException("Enter the correct value");
  }
  finally {
    choiceMenu();
  }
}
```

The takeQuizByQuizList() method calls the generateQuizList() to display a list of quiz topics. The student enters the ID of the quiz. It calls the checkIfQuizIDExists() method to verify if the ID exists. If yes, the open questions are fetched using the readQuestionsByQuizID() method. The questions are displayed one at a time. The user enters the answers. The answers are written to the table using the answerOpenQuestion() method. It then calls the readMCQQuestionsByQuizID to fetch the MCQ Questions. . It calls the answerMCQQuestion() method to answer the MCQ Questions.

SERVICE PACKAGE:

The service package has the JDBCDAO classes for the entities, config entry and the ConfigurationService classes, displaymenus and filesUtil

AnswerJDBCDAO.java

createAnswer(Answer answer, Question question, Student student, Quiz quiz)
Creates an answer with the answer, question_id, student_id, quiz_id in the answer table

deleteAnswerByQuizId(Quiz quiz)
Deletes an answer based on a quiz.
This method is called when a quiz is deleted.

deleteAnswerByQuestionid(Question question)
Deletes an answer based on a question_id.
This method is called when a question is deleted.

MCQCHOICEJDBCDAO.java

createMCQChoice(List<MCQChoice>MCQChoiceList,MCQQuestion question)
Creates MCQ Choices in the MCQChoiceList in the MCQ Choice table for the MCQ Question passed in the argument.
This method is called when an MCQ Question is created.

List<MCQChoice> getMCQChoiceByID(MCQQuestion question)
Returns a list of the MCQ Choices for the MCQ Question in the argument.
This method is called when a student answers an MCQ Question.

deleteMCQChoiceByQuestionID(MCQQuestion question)
Deletes the MCQ Choices for the MCQQuestion in the argument.

MCQQUESTIONJDBCDAO.java

createMCQQuestion(MCQQuestion question,Quiz quiz)
Creates an MCQ Question for the quiz passed in the argument.

updateMCQQuestion(MCQQuestion mcqquestion)
Updates the MCQ Question passed in the argument

List<MCQQuestion> readMCQQuestionsByQuizID(Quiz quiz)
Returns a list of MCQQuestions in the quiz passed in the argument.

List<MCQQuestion> readMCQQuestionsByTopics(String topics)
Returns a list of MCQ Questions based on the topics based in the argument.
The user enters a string of topics separated by commas.

Example:

```
Enter the topics separated by commas
java,programming
```

Here, each word is a topic and so is a search criteria.

The string is split by the delimiter(‘,’) to get each topic. Then each word is used as a search criterion to search the question table to retrieve relevant questions.

Sometimes the same question may be retrieved more than once as it can satisfy more than one topic. To avoid repeating the same question, the question_ids are stored in a hashset. Everytime a new question is fetched, it checks if the id is already in the set, if not, it adds the question to the question list.

```
public List<MCQQuestion>  readMCQQuestionsByTopics(String topics) throws SQLException, ReadFailedException {
  String[] tempArray;
  String delimiter = ",";
  tempArray = topics.split(delimiter);
  String readMCQQuestionsByTopics =ConfigurationService.getInstance()
      .getConfigurationValue("db.queries.mcqquestion.readQuestionsByTopic","");
  HashSet<Integer> id_list=new HashSet<Integer>();
  List<MCQQuestion> mcqquestionlist = new ArrayList<MCQQuestion>();
  for (int i = 0; i < tempArray.length; i++) {
    String search_word = "%" + tempArray[i] + "%";
    try(Connection connection = getConnection();
    PreparedStatement pstmt = connection.prepareStatement(readMCQQuestionsByTopics);){
    pstmt.setString(1,search_word.toLowerCase());
    ResultSet rs = pstmt.executeQuery();
    while(rs.next())
    {
      int id = rs.getInt("mcqquestion_id");
      String text = rs.getString("question");
      if(!id_list.contains(id)) {
      MCQQuestion question = new MCQQuestion();
      question.setId(id);
      question.setContent(text);
      mcqquestionlist.add(question);
      id_list.add(id);
      }
    }
```

Splits the topic list by commas

Adds the question only if is not present in the has set (to avoid duplicates)

deleteMCQQuestionByQuizId(Quiz quiz)
Deletes an MCQ Question by the quiz_id passed in the argument.
This method is called when a quiz is deleted.

deleteMCQQuestionByID(MCQQuestion mcqquestion)
Deletes an MCQ Question by the mcqquestion_id passed in the argument.

searchMCQQuestion(MCQQuestion questioncriterion)
Searches for an MCQ Question based on the search criterion.

QUESTIONJDBCDAO.java

createQuestion(Question question,Quiz quiz)
Creates a question for the quiz passed in the argument.

updateQuestion(Question question)
Updates the question passed in the argument

readQuestionByQuizID(Quiz quiz)
Returns the questions in the quiz passed in the argument.

readQuestionsByTopics(String topics)
Returns questions based on topics given by the user. Uses the same logic as retrieving MCQ questions based on topics.

deleteQuestionByQuizId(Quiz quiz)
Deletes the questions in the quiz passed in the argument.

deleteQuestionByQuestionID(Question question)
Deletes the question passed in the argument.

QUIZJDBCDAO.java

createQuiz(Quiz quiz)
Creates the quiz passed in the argument in the quiz table.

updateQuiz(Quiz quiz)
Updates the quiz passed in the argument in the table.

List<Quiz> readQuiz()
Returns the list of quizzes from the quiz table.

deleteQuiz(Quiz quiz)
Deletes the quiz passed in argument.

STUDENTJDBCDAO.java

createStudent(Student student)
Creates the student passed in the argument in the student table.