Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Estado de México

Desarrollo e implantación de sistemas de software

Project 1: Quiz Design

Grupo 502

Gilberto André García Gaytán - A01753176

Profesor:

Mikaela Rae Grace

Index

Assignment 1: Modeling Quiz App 3
Problem Statement 3
Your Task 3
Deliverables 3
Feature list 3
Detailed class diagram 4
Full Diagram 8
Behavioral diagram for the following use case 8
Design Pattern Review 9
References 9

Assignment 1: Modeling Quiz App

Problem Statement:

You are a freelance software engineer, and you have a new customer who wants you to make a quiz app! The customer will provide you with a list of questions and answers. Here are their specifications:

1) The application must allow the user to select how many different questions (between one and ten) the user wants to answer.
2) Each question must appear all by itself in the browser window.
3) Once the user answers a question, the application must give the user the corresponding feedback, indicating if the answer was right or wrong, and displaying the correct answer if the user's choice was wrong. After this, the user can proceed to see and answer the next question.
4) Once all questions have been answered, the application must display the final score. Also, a score table with the initials and scores (in descending order) of **all previous users** should be displayed.
5) Finally, the user should be given the option to restart the quiz application.

Your Task:

Your job is to design this system, following the SOLID and OOP principles we have discussed. Specifically, you will design a set of classes that work together to provide the needed functionality.

You will have to think through how the product will work, and what classes you might need.

Deliverables:

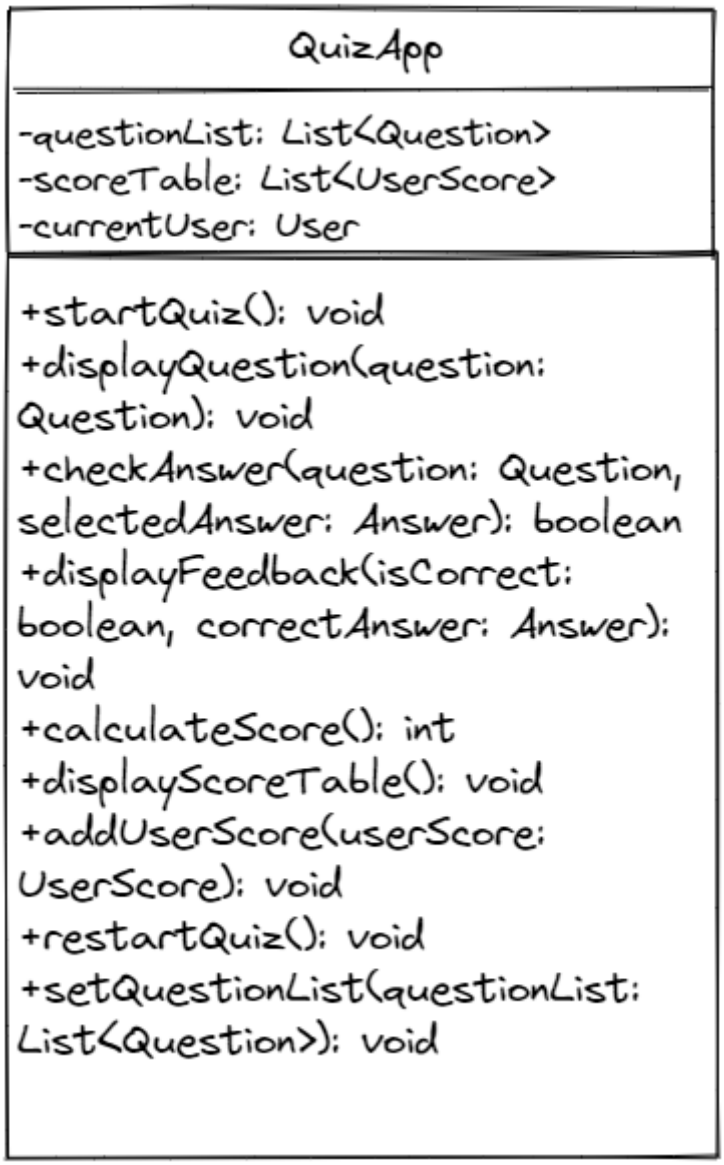
- **Feature list:**
    - A clear and detailed feature list. What features are provided by this app? Feel free to group them or structure the list however you like – the point here is to get really specific and clear on what you are trying to build.
    - Be aware that some requirements are not stated explicitly above, e.g. in order to display previous scores, some information will need to be collected from the customer.
- **Question List:** The app will allow the customer to provide a list of questions and answers.

- **Select Number of Questions:** The app will allow the user to select how many different questions they want to answer (between one and ten).
- **Present Questions:** The app will present each question one by one to the user.
- **Possible Answers:** The app will show a list of possible answers to each question.
- **Feedback:** The app will provide feedback to the user after each answer, indicating if the answer was right or wrong.
- **Correct Answer:** The app will display the correct answer if the user's choice was wrong.
- **User Progress:** The app will track the user's progress through the quiz.
- **Final Score:** The app will calculate the user's final score after all questions have been answered.
- **Score Table:** The app will display a score table with the initials and scores (in descending order) of all previous users who have taken the quiz.
- **Restart Quiz:** The app will allow the user to restart the quiz after completing it.
- **User Name:** The app will allow the user to enter their name before taking the quiz.
- **User Score:** The app will store the user's score and display it in the score table.
- **Data Persistence:** The app will save user scores between sessions so they can be displayed in the score table.
- **Score Table Limits:** The app will limit the score table to display only the top 10 scores.
- **Time Limit:** The app will provide an optional time limit for answering each question.
- **Difficulty Level:** The app will allow the customer to assign a difficulty level to each question.
- **Multiple Choice:** The app will allow the customer to provide multiple-choice questions.
- **True/False Questions:** The app will allow the customer to provide true/false questions.
- **Short Answer Questions:** The app will allow the customer to provide short answer questions.
- **Randomized Questions:** The app will randomize the order of questions presented to the user.
- **Detailed class diagram:**
    - Diagram showing all essential classes needed, associations with navigability.
        - Feel free to include things like "MongoDatabase" that will be third-party imports or services, as well as the classes you designed.
        - Feel free to group your classes into sections like "frontend" and "backend" or other groupings, as you like.
        - Add constraints or comments as appropriate. You do not have to model low-level classes like simple lists, vectors, user interface elements (buttons, menu, menu item, dialog boxes).
    - For each class: attributes with visibility and type, operations with full signature (input parameters and result values), descriptions ONLY where necessary.
        - For example, do not need to explain operations where the name says it all like in 'set_name(name)').

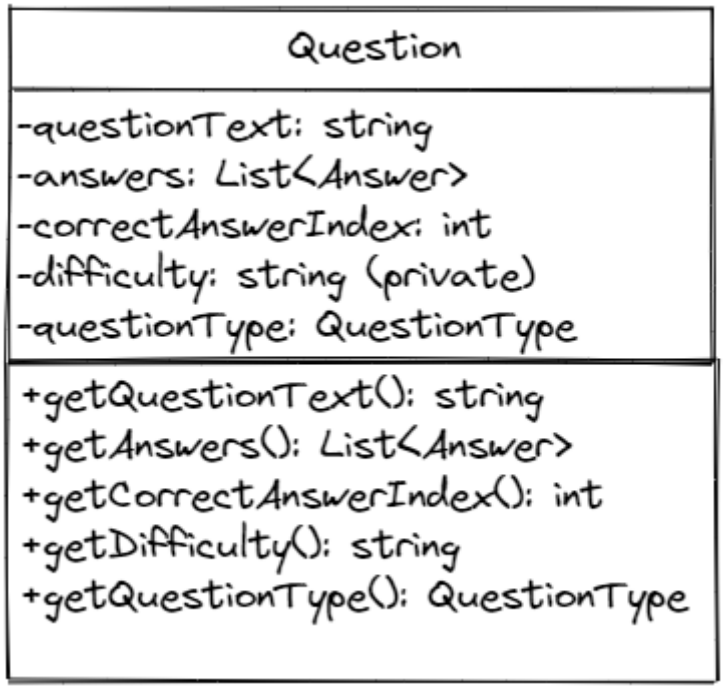
- This can be either in the class diagram or separately in text form.

QuizApp Class:

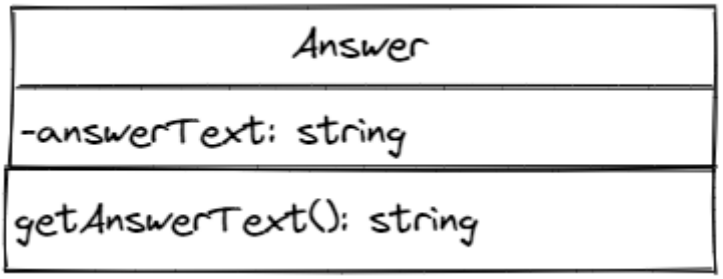
This is the main class that handles the overall functionality of the quiz app.



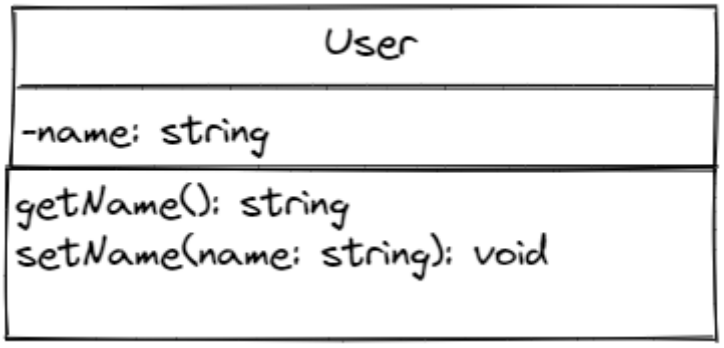
Question Class:
This class represents a single question in the quiz.



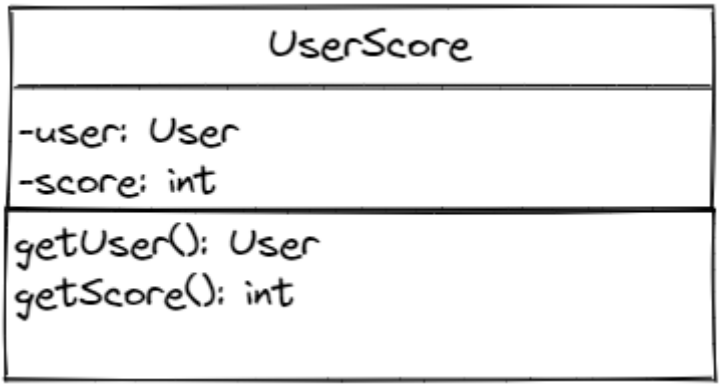
Answer Class:
This class represents a single answer to a question.



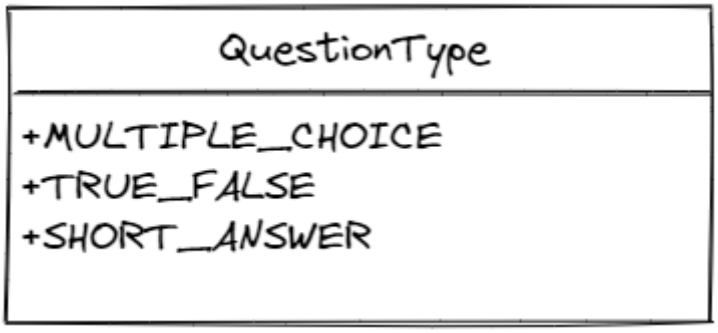
User Class:
This class represents a user of the quiz app.



UserScore Class:
This class represents a single user's score for the quiz.

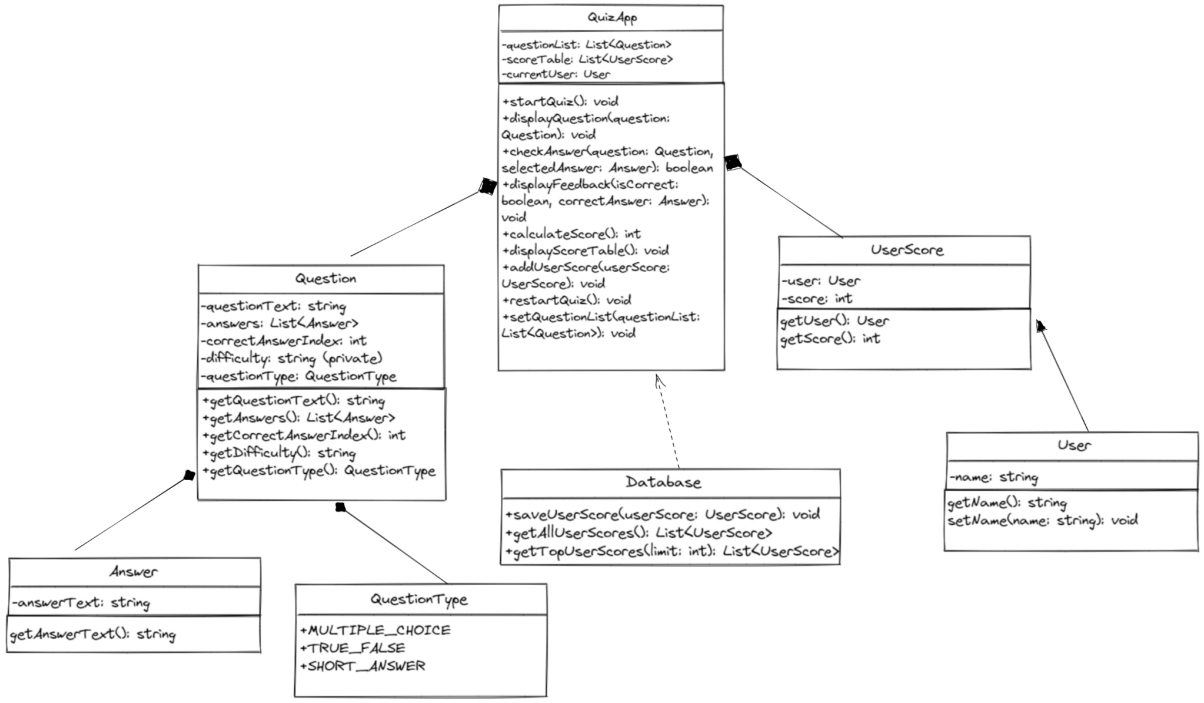


QuestionType Enum Class:
This enum represents the different types of questions that can be in the quiz (multiple choice, true/false, short answer).

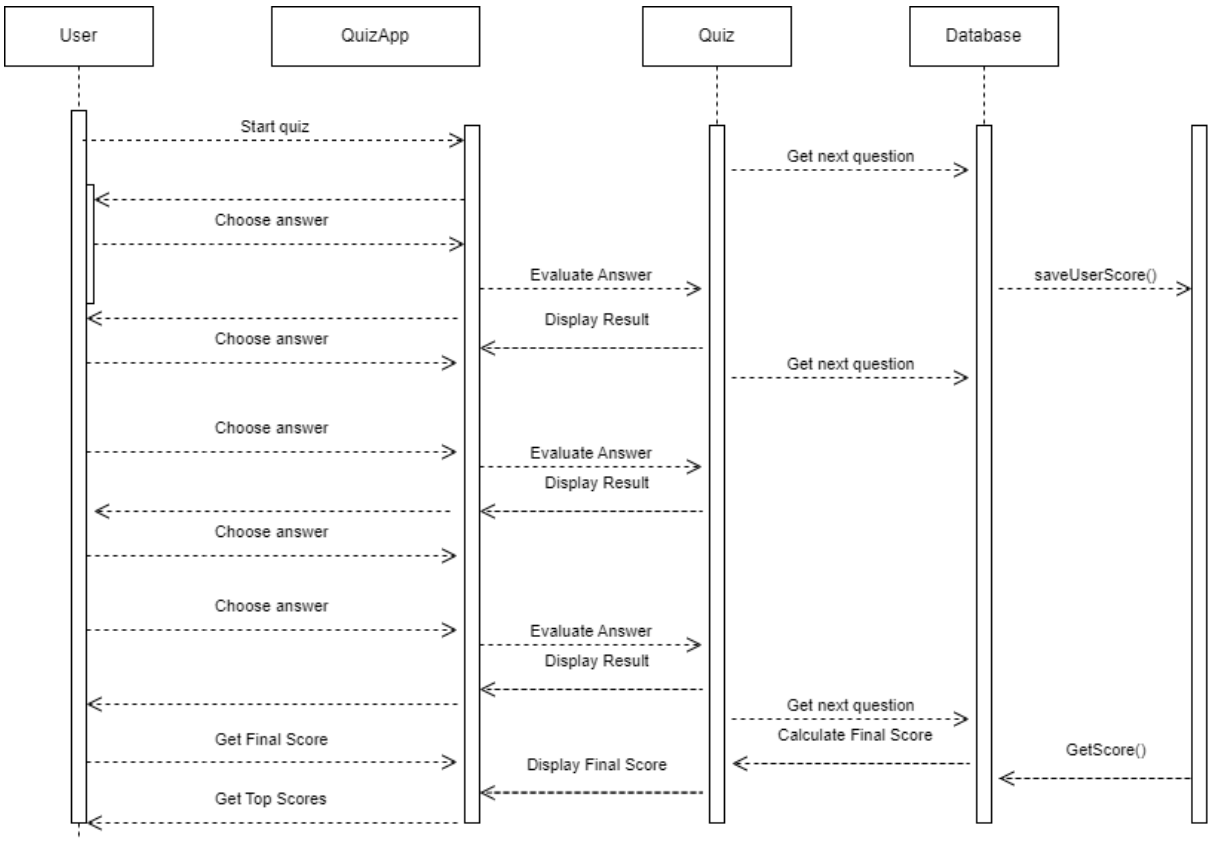


Database Class:
This class handles the data persistence of the quiz app.

Full Diagram:



- **Behavioral diagram for the following use case:**
  - (From the beginning of the quiz process). A user answers all questions correctly and gets the top score, which is a tie for the top score with another previous user.



- **Design Pattern Review:**
    - A short (1-2 paragraph) writeup of which of the design patterns, architecture concepts, or SOLID concepts you are using, and why.

In the design of the quiz app, several SOLID principles have been used to ensure a robust and maintainable design. The Single Responsibility Principle (SRP) has been applied to ensure that each class has only one responsibility, such as the Question class which is responsible for storing and managing a single question. The Open-Closed Principle (OCP) has also been applied by designing classes that are open for extension but closed for modification, such as the Quiz class which can be extended to include new types of questions without modifying the existing code. Finally, the Dependency Inversion Principle (DIP) has been applied to ensure that classes depend on abstractions rather than concrete implementations, making it easier to replace and test different implementations. Overall, these SOLID principles help us to create a flexible and maintainable design for the quiz app.

References

Freeman, A., & Pryce, N. (2009). Growing Object-Oriented Software, Guided by Tests. Addison-Wesley Professional. https://www.growing-object-oriented-software.com/

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley Professional. https://www.informit.com/store/design-patterns-elements-of-reusable-object-oriented-9780201633610

Martin, R. C. (2000). Design Principles and Design Patterns. Object Mentor. https://objectmentor.com/resources/articles/Principles_and_Patterns.pdf