**Logical design**

For the logical design I structured a plan in the following way:

1. Question Structure

* Each question has :
  + - An ID for identification
    - The question itself
    - Multiple choices
    - The index for the correct answer

1. User progress tracking

* Keep track of:
  + - Score, based on the number of the correct answers.
    - Current displayed question(the question that is displayed)
    - Quiz completion(a flag that identifies if the quiz is completed)

1. Code organization:

* 3 files(HTML, CSS & JavaScript):
  + JavaScript:
    - Initialize variables for score, current question and completion status.
    - Create functions for getting random question from the array and displaying it.
    - Add functions to handle the answers submission and user’s progression through the quiz.
    - Function to display a question with its answers.
    - Function to submit answers and handle correctness.
    - Going through each question in the array until all questions have been answered.
    - Display user’s result.
  + HTML:
    - Structure the quiz container.
    - Elements to display questions, choices and result.
    - Link the CSS and JavaScript file.
  + CSS:
    - Style the layout, font and size for the user interface.
    - Apply hover effects to choices and buttons.
    - Hide or show elements based on quiz progression.

**Pseudocode and Algorithm implementation**

In the second part, I will present the pseudocode/the logic behind the code:

Firstly, in the JavaScript file, we find the most important part of the code for the quiz app. Here I developed the algorithm for random selection of the questions, the answer handler and the scoring system.

In the beginning of the file, we’re going to find the array of the questions. After that we’re going to see the initialization of the variables(score, currentQuestion, quizCompleted). The first function, seen as the most important one in the text of the problem, is the function that gets a random question from the pool of 50. The “getRandomQuestion()” function handles this problem. Here is the pseudocode:

***Function getRandomQuestion():***

***Randomly select a question from the questions array.***

***Remove the selected question from the array.***

***Return the selected question.***

Basically, every time this function(**line 130, quiz.js**) is called, it is randomly generating an index within the valid range of the questions array, using the multiplication between a random floating-point number (Math.random() - function) and the lenght of the array( questions.lenght) and then ensure that the index generated is an integer by rounding the result of the multiplication to the nearest whole number using the Math.floor() function. After generating the random index, the variable selectedQuestion gets the value of the question[randomIndex], then this question is removed from the array using the “splice()” function, in the end our function returns the value of the variable “selectedQuestions”. That’s how we’re sure a random question from the array is selected without repeating until all the questions are answered.

O imagine care conține text, captură de ecran, Font

Descriere generată automat

**Class and Database Representation**

**Class Diagram**

In the end, I will explain how I would represent the structure of the application in a class diagram and a database diagram. I must add that I didn’t created classes in the code I provided, because I understood from the tasks that it wouldn’t be necessary to have the entire code, so I tried just to have a short app that would simulate the requirements of the problem so I could have an user interface and easily describe how I would solve the task. And added only 20 questions for testing easily the application.

So, in a class diagram I would had two classes: QuizManager class and the Question class. In the QuizManager class I would have all the important attributes like the array of questions, the score, the flag of the quiz completion and the currentQuestion attribute that represent the question which is displayed and the all the functions: getRandomQuestion(), displayQuestion(), showSubmitButton(), handleAnswerSubmission(), handleAnswer(), displayNextQuestion() and showResult(). This class orchestrates the entire flow of the quiz, managing the question selection, user interactions and display the result. The second class, Question, is basically a blueprint to represent the individual quiz questions. It would have attributes like id, question, choices and correct answers. It encapsulates all the essential components of a quiz question into one piece.

**Database Diagram**

In a simple database scheme, I would create two tables, one for the questions and one for the choices. In the question table I would have a question ID with a primary key, the question text and the correct choice id. In the choice table, I would have the choice ID(primary key as well), question ID with a foreign key referencing the question table and the choice text. This structure would allow me to connect questions to their choices of answers and link the correct answer to each question.

**Relationship and dataflow**

The QuizManager class is instantiated when the quiz application loads. QuizManager owns an array attribute called questions that holds instances of the Question class. QuizManager owns an array attribute called questions that holds instances of the Question class. QuizManager calls its getRandomQuestion() method to select a random Question object from the questions array. This method returns a selected Question object to be displayed. QuizManager calls the displayQuestion method, passing the selected Question object. The displayQuestion() method extracts data from the Question object and presents it in the user interface. When the user selects an answer and submits it, the QuizManager triggers the handleAnswerSubmission() method. This method validates and processes the user's answer by calling the handleAnswer() method, passing the selected answer index. handleAnswer() compares the user's selected answer index with the correct answer index stored in the currentQuestion attribute. If the answer is correct, it updates the score attribute in the QuizManager. After handling the user's answer, QuizManager proceeds to the next question by calling displayNextQuestion() to continue the quiz flow. The process continues until all questions are answered. Upon completion, QuizManager calls showResult() to display the final score.