Assignment 2: Quiz app server APIs

***IMPORTANT**: This assignment must be done individually.*

Read Section A to understand the programming requirements, Section B to understand the programming tasks that you need to carry out, Section C gives you some more tips and Section D to know what you need to submit as the result.

A. Description

In the assignment 1, you created your quiz app which fetch data from my provided server APIs on https://wpr-quiz-api.herokuapp.com/. In this assignment, you will create your own server APIs with use of ExpressJS and database MongoDB.

API end-points

Recall, the Quiz is an app that provides a way to make a simple, login-less attempt on predefined content (HTML, CSS, JS or anything else).

Attempt: Start a new attempt

`POST: /attempts`

- **Request:**
  - **Data:** none
- **Response:**
  - **Status:** 201 (Created)
  - **Data:** the just created attempt (contains attempt ID & an array of questions)

Use-case: after reading some information about the quiz, user click "*Start the X quiz*"

Client: send request to create a new attempt

Server: → create a new **Attempt** with 10 random questions from the `questions` collection → save into the database & returns

The data returned follows the structure below:

- **_id**: attempt id (auto generated value by the Mongo DBMS)
- **questions**: array of 10 random questions from the `questions` collection, each question is an object with:
    - **_id:** question id (auto generated value by the Mongo DBMS)
    - **text:** question content
    - **answers:** array of options for user to choose

  For example,

```
{
    "_id": "5f6bb58597518f0bc82e4231",
    "text": "Who is making the Web standards?"
    "answers": [
        "Mozilla",
        "Microsoft",
        "Google",
        "The World Wide Web Consortium"
    ],
}
```

- **startedAt**: current time
- **completed**: false (by default)

IMPORTANT NOTE: the correct answer should NOT be returned with questions.

Attempt: Submit user's answers & finish attempt

POST: `/attempts/:id/submit`

- **Request:**
    - **Route param:**
        - `:id` id of the attempt
        - **JSON body:** an object of user selected answers with format `{questionID: userSelectedAnswerIndex, …}`, where
            - `questionID`: _id of the question
            - `userSelectedAnswerIndex`: corresponding index of the user selected answer in the array of all answers (options) for that question

For example: the below question has _id=`5f6bb58597518f0bc82e4231` with the selected answer: `The World Wide Web Consortium` (index 3 in the array of answers) will be represented as:
{"5f6bb58597518f0bc82e4231": 3, …}

```
{
    "_id": "5f6bb58597518f0bc82e4231",
    "text": "Who is making the Web standards?"
    "answers": [
        "Mozilla",
        "Microsoft",
        "Google",
        "The World Wide Web Consortium"
    ],
}
```

- **`Response:`**
    - **`Status:`** `200 (OK)`
    - **`Data:`** `The attempt (contains questions, user selected answers, correct answers of the same format, score, scoreText)`

Use-case: after answering questions in the quiz, user click "*Submit your answers*"

Quiz app: send request with user's answers to score & complete the **`Attempt`** specified by `:id`

Server:

- Get the attempt with specified id
- Save user's answers for later process
- Compute score by comparing user's answers with the corresponding correct answers for each question. (max 10 since we have only 10 questions)
- Choose scoreText corresponding to the computed score using this rule:
    - score < **5** → "Practice more to improve it :D"
    - **5** <= score < **7** → "Good, keep up!"
    - **7** <= score < **9** → "Well done!"
    - **9** <= score <= **10** → "Perfect!!"

IMPORTANT: Don't score an attempt again if it is completed

The data returned follows the structure below:

- `_id, questions, correctAnswers, startedAt` are similar to what described in the API ***Attempt: Start a new attempt***

But,

- `answers`: user's answers
- `score`: user's score
- `scoreText`: feedback to user
- `completed`: true

Data Model

- You can think of your Quiz app as having two main "entities":
    - **Questions:** A question contains the text content, an array of options for user to select, also the correct answer for this question (identified by its index in the options array)

      Note: in this app, we create dynamic attempts by randomizing a list of questions but not to mix the answers
    - **Attempts:** An attempt records user's attempt on our designed quiz, containing a list of random questions with their correct answers, the timestamp when user started the attempt, an indicator if user completed this attempt; and of course, user's answers, the score & corresponding feedback to user.

  IMPORTANT: the unique `_id` is added automatically by MongoDB.
- You don't have to model your data (attempts) this way, but it should make querying your data simpler.

IMPORTANT: In this assignment, you HAVE TO

1. Import & use the provided ***questions*** collection (with sample data for testing). We will use this file for marking also.
2. Serve ExpressJS server at port **3000**
3. Use MongoDB with default configuration (default port **27017**, **no username/password** required)

B. Task requirements

In this assignment, you are free to organize your app

1. **Structure your web application**

   Note: `package.json` must be included.

2. **Backend (NodeJS)**

   - Create api routes to manipulate with database to serve functions as mentioned in *(Section A)*

3. **Weekly plan**

   You have to schedule yourself a weekly plan to complete your tasks, an example is given below.

Week	Database	Backend
1 **Check point 1**	- Setup MongoDB database with the provided ***questions*** - Decide the structure of the database for your app (***attempts***)	- Setting up project with *package.json* - API end-point: *Attempt: Start a new attempt* - Test the API end-point: *Attempt: Start a new attempt* with your Quiz app (created in previous assignment)
2 **Final submission**	- Any modification (if needed)	- API end-point: *Attempt: Start a new attempt* - Test the API end-point: *Attempt: Start a new attempt* with your Quiz app (created in previous assignment)

NOTE:
In case you haven't completed the previous assignment yet, use Postman app with the provided ***postman_collection*** & sample request/ response from https://wpr-quiz-api.herokuapp.com/ for testing.

C. Tips

Here are some specific tips for coding the Quiz app APIs:

Date class

- You will need to use JavaScript's `Date` class to get the current date and time.
    - Date example in JavaScript & MongoDB: Please check this out!

Sample request & response

- See: https://wpr-quiz-api.herokuapp.com/

D. Submission

You must submit a single zip file containing the application to the portal by the due date. The zip file name must be of the form `a2_Sid.zip`, where *Sid* is your student identifier (the remaining bits of the file name must not be changed!). For example, if your student id is 1801040001 then your zip file must be named a2_1801040001.zip.

Note: Do not include `node_modules` folder into your submission

IMPORTANT: failure to name the file as shown will result in no marks being given!

NO PLAGIARISM: if plagiarism is detected, 0 mark will be given!