# Project 3 : Build Front-end SPA application

## Assignment 4 : Build Front-end SPA application

As part of this project, you need to create the front-end for a quiz delivery applications. You will implement a mini-version of a quiz app from the ixl website . In particular, you will implement the `Take Quiz Use-case`. You need to implement the the logic of your application using `javascript`. This must be a single-page application with a **static back-end REST API** to deliver the data and you must use `HTML`, `CSS` and bootstrap. Your implementation should comply with all the requirements described below.

## Use-case Description

At the starting page of your application, the user will be asked to provide their name (via an input form) and select from a menu one of two possible quizzes to take. Once a quiz is selected, the student should be able to begin the quiz. The quiz should appear in a **separate view within your application** (i.e. not a separate page; your solution must be a SPA). You **must** use the handlebars template engine to handle the transitions between the various views of your application, and you are not allow to use bootstrap modal plugins or other similar modal libraries).

Once the student selects a quiz, the quiz information should be loaded asynchronously from a static REST API endpoint (more on this below), and the quiz begins. Your asynchronous call should only download the information needed. Each quiz includes different types of questions and each question has a unique correct answer. During the quiz, a student is promoted with a question (one question at a time) and an appropriate interface for the student to provide an answer to the question (i.e. type the answer, or select correct answer from multiple options etc). After each response, the application should evaluate the user's response and provide a feedback to the student. If the student answered the question correctly, then an encouraging message, such as "Brilliant!", "Awesome!" or "Good work!, should appear on the screen for one second (1000 milliseconds). After the one second has passed, the application should automatically hide the message and display the next question in the test. If the user answers the question wrongly, then a new feedback view (i.e. a new HTML DOM ELEMENT) should providing a short explanation as to what the correct answer is. The view should also provide an HTML button with the text "Got it"; when this button is clicked by the user, the "feedback view" should disappear and the next question in the test should appears. This process should continue until all questions in the quiz are shown. Questions for the quiz **must be loaded asynchronously, one at a time** (i.e. **do not** load all the questions at the beginning of the quiz)

During the quiz, a scoreboard should keep track and display the number of questions answered so far, as well as, the elapsed time and the total score. At the end of the quiz, the student should receive a message, in a **separate view**, saying "Congratulations <student name>! You pass the quiz" if the students scores above 80\% or "Sorry <Student Name>, you fail the quiz" otherwise. Where you replace the <Student Name> with the actual name the student provided at the starting view of you application. Moreover, at the end of the quiz, the student should be given the option to either re-take the quiz or return to the main page.

**Static back-end API**: You application should retrieve all quiz information and quiz questions from a remote API. For simplicity, in this milestone of the project you are expected to use a static API to serve the data. For that, you must use the J**SONPlaceholder** server which allows you to build fake online REST API for testing and prototyping. The fake API provides RESTFUL API endpoints that serve responses based on static JSON files that are stored in your GitHub repository. You can read how to set the fake api for your code at "**https://my-json-server.typicode.com/**"; the process is straight forward.

## Project requirements and additional specifications

- Your submission should provide a static REST API access points that serve any data your applications needs using the JSONPlaceholder service. Thus, all the data for your application (i.e. questions details, correct answer) should be store on a GitHub Repository appropriately configured to be used with JSONPlaceholder service.
- Your application should load all data from the fake API asynchronous using network request. You can use either **fetch** command with **Promises** or the **'Async/Await'** syntax. Each request should retrieve only the data it needs to perform the underlying operation. For example, when you display a question on the screen, you are expected to retrieve the current question information only using asynchronous network request.
- Your `use-case` should implement at least five different `question types` for the quizzes based on the question types implemented on ixl website. Some examples, of `type of questions` are linked below, but feel free to navigate the ixl website for more options: Multiple Choice Text Buttons ; Narrative question with text response ; Question with Image selection among others.
- Each online quiz instance should include **at least twenty unique** questions. The question must be create by you. They must be real meaningful question and appropriate correct answer and illustrations (i.e. do not use lorem ipsum or question placeholders). The question must relate to computer science or a programming quiz; for example, you can have questions about a programming language such as java, javascript, C/C++, python, rust, php; or a programming framework etc.
- Your application should implement all features specified in the use-case description.

- Your application must be a single-page applications (SPA).
- You must style your applications using Bootstrap and CSS. Your design must be unique, responsive and visually appealing.
- Your application must use the **handlebars template engine** to handle all dynamic view changes in the page. Documentation for the handlebars can be found here;
- You application must use asynchronous request using the fetch method and promises (use can use the async/await syntax or the .then syntax). All data must be retrieved, at the time they are needed by the application, from the static back-end API.