

W1 PRACTICE

QUIZ APP

💡 Learning objectives

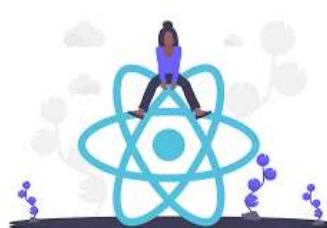
- ✓ Understand **separation of concerns**
 - Data (quiz, answers, score)
 - View (DOM rendering)
- ✓ **State management** using JS variables
- ✓ **Navigation** using URL or JS variables
- ✓ **CRUD logic** (Create / Read / Update)

⚠️ Important

- ✓ The **reflection part** will be done in **teams of 2 (designing) and 4 (sharing)**
- ✓ The **coding part** needs to be submitted **individually**

⌚ How to submit?

- ✓ **Push** your final code on **your GitHub repository**
- ✓ Then **attach the GitHub path** to the MS Team assignment and **turn it in**



THE QUIZ APP

During this practice you will implement a **QUIZ APP**, following the bellow requirements:

Quiz Player

- Start a quiz
- Answer multiple questions
- Navigate through questions
- See the final score at the end

Quiz Editor

- View all quiz questions
- Edit existing questions
- Add new questions
- (Optional) Delete questions

Navigation

- Switch between views

The image displays four screenshots of a mobile application interface for a "Quiz App".

- Screenshot 1:** Home screen. The title "Quiz App" is at the top. Below it is a large, friendly cartoon brain wearing glasses and holding books, with a small speech bubble above it.
- Screenshot 2:** Quiz player screen. The title "Quiz App" is at the top. Below it is a question: "What does CSS stand for?". Four options are shown in purple boxes: "Cisco and Super Star!", "CI So Sa.", "Cascading Style Sheets", and "I don't know!".
- Screenshot 3:** Quiz editor screen. The title "Quiz App" is at the top. A modal dialog box is open with the heading "Create new question". It has fields for "Title" (containing "What does HTML stand for?") and "Answers" (containing "Hi Thierry More Laugh!", "How To move Left", "Ho Theory Missed the Laundry!", and "Hypertext Markup Language"). At the bottom are "Cancel" and "EDIT" buttons.
- Screenshot 4:** Quiz player screen. The title "Quiz App" is at the top. Three questions are listed with checkboxes:
 - "What does HTML stand for?" (checkbox checked, red dot)
 - "What does CSS stand for?" (checkbox checked, red dot)
 - "What does JS stand for?" (checkbox checked, red dot)

STEP-1 – Let's start

 The starter code is provided; **you need to complete it.**

The HTML is composed of 3 views: start, quiz, score.

We want to **dynamically display only 1 view**

```
<body>
  <div id="start">Start Quiz!</div>

  <div id="quiz" style="display: none">
    <p id="question"></p>
    <div id="choices">
      <div class="choice" id="A" onclick="checkAnswer('A')">AAAA</div>
      <div class="choice" id="B" onclick="checkAnswer('B')">BBBB</div>
      <div class="choice" id="C" onclick="checkAnswer('C')">CCCC</div>
      <div class="choice" id="D" onclick="checkAnswer('C')">DD</div>
    </div>
  </div>

  <div id="score" style="display: none"></div>
</body>
</html>
```

- ✓ Complete the show and hide functions

```
const dom_start = document.querySelector("#start");
const dom_quiz = document.querySelector("#quiz");
const dom_score = document.querySelector("#scoreContainer");

function hide(element) {
  // TODO
}

function show(element) {
  // TODO
}

// TEST (display the quiz view)
show(dom_quiz);
hide(dom_start);
hide(dom_score);
```

- ✓ Make sure you can dynamically display 1 view

STEP -2 – The *player view*

Implement the JS code to comply with the player requirements:

- ✓ The game shall **start on the start view**
- ✓ When clicking on start quiz, **the quiz view** is displayed
- ✓ When player clicks on any answer button, the **next question** shall be displayed
- ✓ When the last question is finished **the score view** is displayed with the score



SCORE	EMOJI
<20	:(
Between 20 and 40	(:(
Between 40 and 60	:(=
Between 60 and 80	:)
>80	:D

💡 You can use the **bellow variables** to handle your data:

```
let questions = [...] // all questions
let runningQuestionIndex = 0; // index of the current question
let score = 0; // current score
```

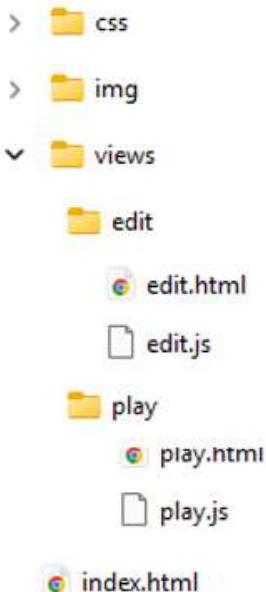
💡 You can **divide your code using the bellow functions**:

Function	Description
onStarted	Display the quiz view, render the current question
renderQuestion(questionIndex)	Render a question on the quiz view
onPlayerSubmit(answerId)	Update the score, display the next question or the score view
renderScore	Display the score Display an emoji related to the score value

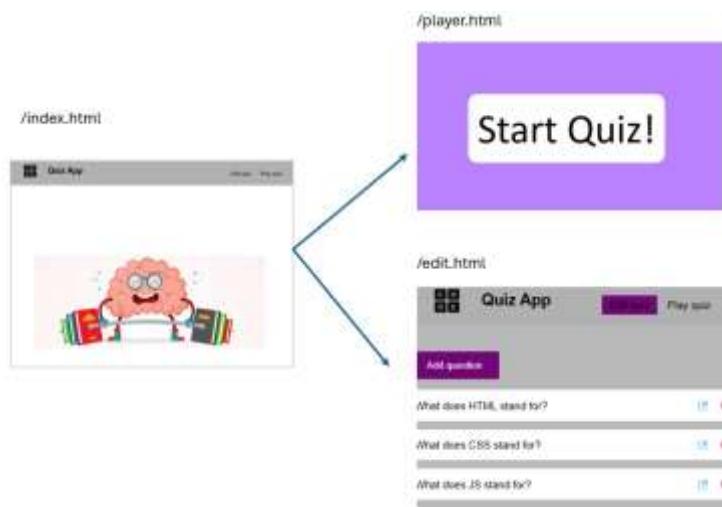
STEP -3 – The menu and separated view

Now we need to separate the app html and js into different modules, to handle the new features easily:

- ✓ Refactor the **project structure** as follows



- ✓ Add the navigation bar to navigate between the 2 menu items



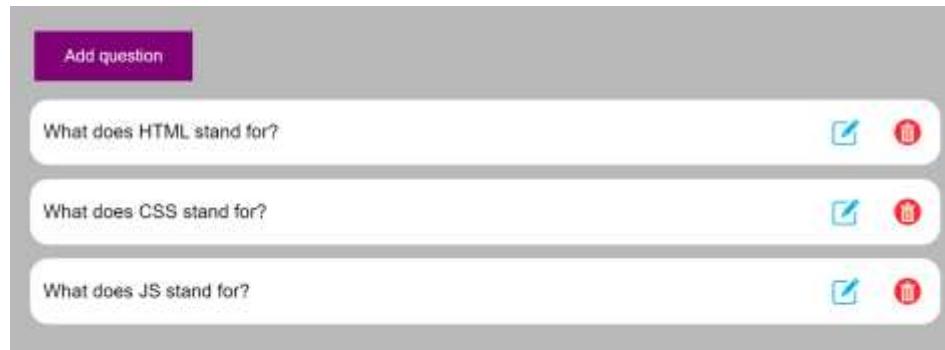
Don't forget to include the menu in the 2 sub views, and to display the selected menu item

- 💡 Use the CSS class `active` to style the active menu item

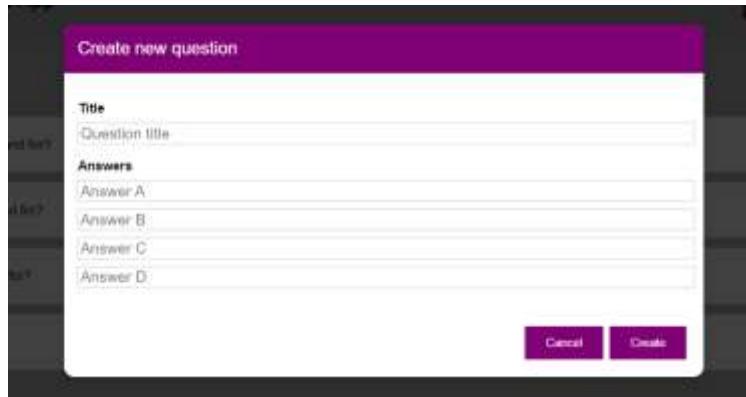
STEP -4 – The *edition* view

Implement the JS code to comply with the edition requirements:

- ✓ The edition view shows all questions, and allows them to add or edit them

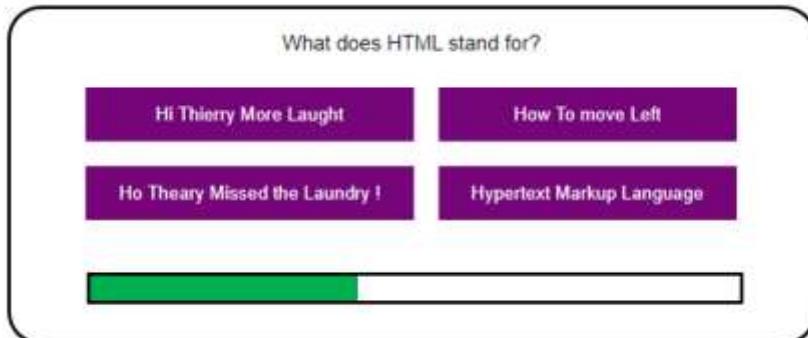


- ✓ When clicking on edit or add, a dialog is displayed to edit the question



BONUS

- ✓ Add a **progress bar** on the PLAY view to see the progress in the quiz



- ✓ On the DIALOG, add a way to select the GOOD answer

Create new question

Title

Question title

Answers

Answer A

Answer B

Answer C

Answer D