CADT IDT

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**

Lab1.1:

The Selector check

```
<h1 id="main-title">Hello World</h1>
<button id="change-btn">Change Title</button>
```

Todo:

1. Select the `h1` and the `button` using JavaScript.
2. Add a click event listener to the button.
3. When clicked, change the text of the `h1` to "Javascript is fun!".

Lab1.2:

Style Manipulation

```
<div id="box" style="width: 100px; height: 100px; background-color: blue;"></div>
<button id="color-btn">Turn Red</button>
```

Todo:

1. When the button is clicked, change the background color of the box to "red".
2. Make it toggle. If it is red, turn it back to blue, and vice versa.

Lab1.3

Input Handling

```
<input type="text" id="username" placeholder="Enter name">
<button id="greet-btn">Greet</button>
<p id="message"></p>
```

Todo:

1. When the button is clicked, get the value typed into the input field.
2. Display "Hello, [Name]!" inside the `<p id="message">`.
3. If the input is empty, display "Please enter a name" in red color.

Lab1.4

The Counter (Data Driven)

```
<h2 id="counter-view">0</h2>
<button id="inc-btn">+1</button>
```

Todo:

1. Create a Javascript variable `let count = 0;` (This is your **Data**).
2. Create a function `render()` that updates `counter-view` with the value of `count`.
3. When the button is clicked:
    - Update the variable (`count++`).
    - Call `render()` to update the UI.
    - *Do NOT change the DOM directly inside the event listener.*

Lab1.5

The Light Switch (Boolean Logic)

```
<div id="room" style="width: 200px; height: 200px;
border: 1px solid ■black; background-color: □white;"
></div>

<button id="switch-btn">Light Switch</button>
<p id="status-text">Lights are ON</p>
```

Todo:

1. Create a variable `let isLightOn = true;`.
2. Create a `render()` function:
    - If `isLightOn` is true: set box background to white, text to "Lights are ON".
    - If `isLightOn` is false: set box background to black, text to "Lights are OFF".
3. On button click: toggle the variable (`isLightOn = !isLightOn`) and call `render()`.

Lab1.6

Simple Object Rendering

```html
<div id="card">
    <h3 id="user-name"></h3>
    <p id="user-role"></p>
</div>
<button id="promote-btn">Promote to Admin</button>
```

Todo:

1. Create a data object: `let user = { name: "Sok", role: "Student" };`
2. Create a `render()` function that puts the user's name and role into the HTML elements.
3. When the "Promote" button is clicked:
    - Change `user.role` to "Admin".
    - Call `render()`.

Lab1.7

Rendering a List (Static)

```html
<ul id="fruit-list"></ul>
```

Todo:

1. Start with this data: `let fruits = ["Apple", "Banana", "Orange", "Mango"];`
2. Write a function `renderFruits()` that:
    - Clears the current innerHTML of `fruit-list`.
    - Loops through the `fruits` array.
    - For each fruit, creates an `<li>` element.
    - Appends the `<li>` to the `<ul>`.
3. Call the function once at the start to display the list.

Lab1.8

Adding to a List (Dynamic)

```
<input type="text" id="new-fruit">
<button id="add-btn">Add Fruit</button>
<ul id="fruit-list"></ul>
```

Todo:

1. Use the code from Exercise 7.
2. When the "Add Fruit" button is clicked:
    - Get the text from the input.
    - `.push()` the new text into the `fruits` array.
    - Call `renderFruits()`.
    - *Observation: Notice how you don't need to manually create the new DOM element in the click event? The render function handles the whole list.*

Lab1.9

Smart List (Objects + Style)

```
<div id="tasks-container"></div>
```

Task:

1. Data:

   JavaScript

   ```
   let tasks = [
     { title: "Do Homework", isUrgent: true },
     { title: "Wash dishes", isUrgent: false }
   ];
   ```

2. Create a `renderTasks()` function.
3. Loop through tasks and create `<div>` elements for each.
4. **Logic:** If `isUrgent` is `true`, set the text color to **red**. If `false`, set it to **black**.
5. Append them to the container.

Lab1.10

The Search Filter

```
<input type="text" id="search-input" placeholder="Search item...">
<ul id="items-list"></ul>
```

Task:

1. Data: `let items = ["Book", "Pen", "Pencil", "Paper", "Backpack"];`
2. Write a function `render(filterText)` that receives a filter argument.
3. Inside the function:
   - Filter the `items` array so it only contains items that include the `filterText`.
   - Render the filtered list to the `<ul>`.
4. Add an event listener to the input (event type: `input` or `keyup`).
5. On event: Call `render(input.value)`.

======== THE QUIZ APP ======

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

Quiz Player

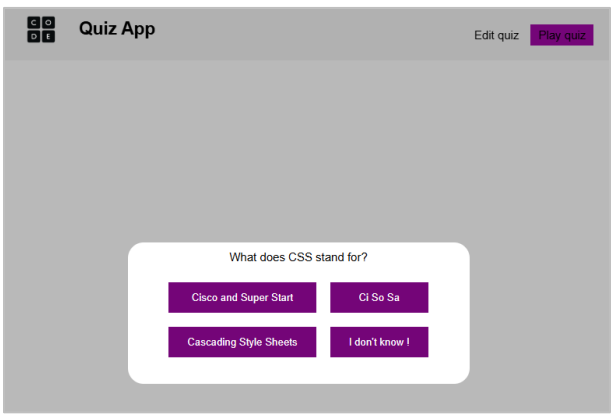
- Start a quiz
- Answer multiple questions
- Navigate through questions
- See the final score at the end

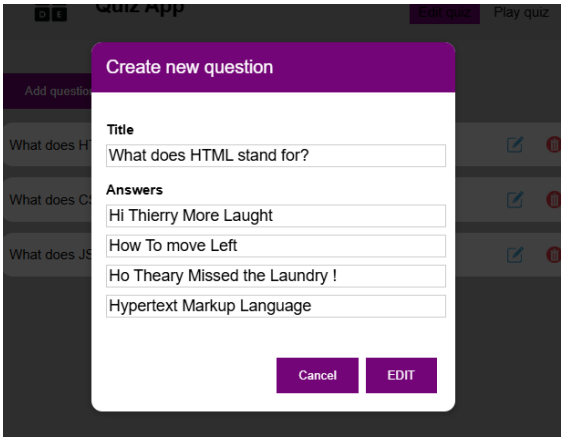
Quiz Editor

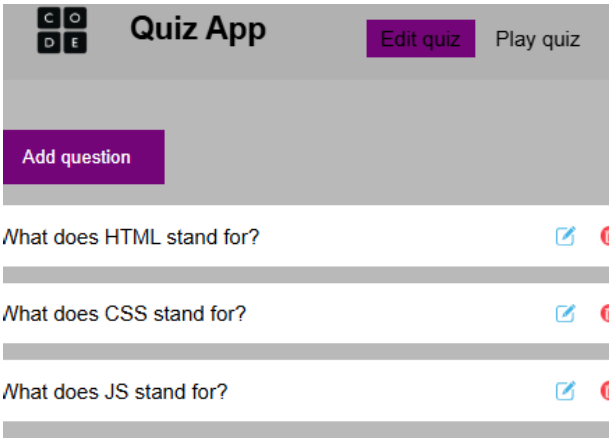
- View all quiz questions
- Edit existing questions
- Add new questions
- (Optional) Delete questions

Navigation

- Switch between views




Create new question
Title
What does HTML stand for?
Answers
Hi Thierry More Laught
How To move Left
Ho Theary Missed the Laundry !
Hypertext Markup Language
Cancel
EDIT


Quiz App
Edit quiz
Play quiz
Add question
What does HTML stand for?
What does CSS stand for?
What does JS stand for?

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**

```html
<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

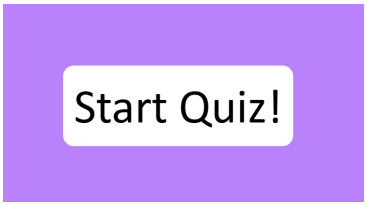
```
const dom_start = document.querySelector("#start");
const dom_quiz = document.querySelector("#quiz");
const dom_score = document.querySelector("#scoreContainer");

function hide(element) {
  // TODO
}


function show(element) {
  // TODO
}

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

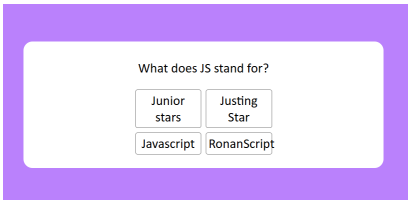
- ✓ Make sure you can dynamically display 1 view

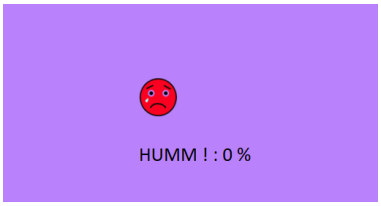
*STEP -2 – The **player** view*

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	EMOIJ
<20	
Between 20 and 40	
Between 40 and 60	
Between 60 and 80	
>80	

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

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

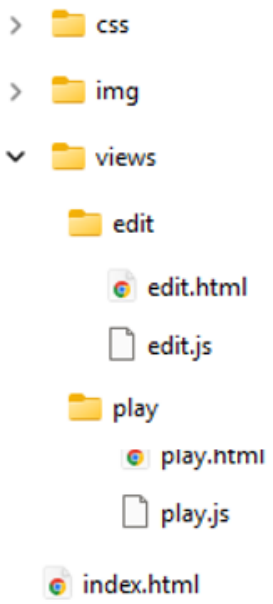
💡 You can d**ivide your code using the bellow functions**:

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

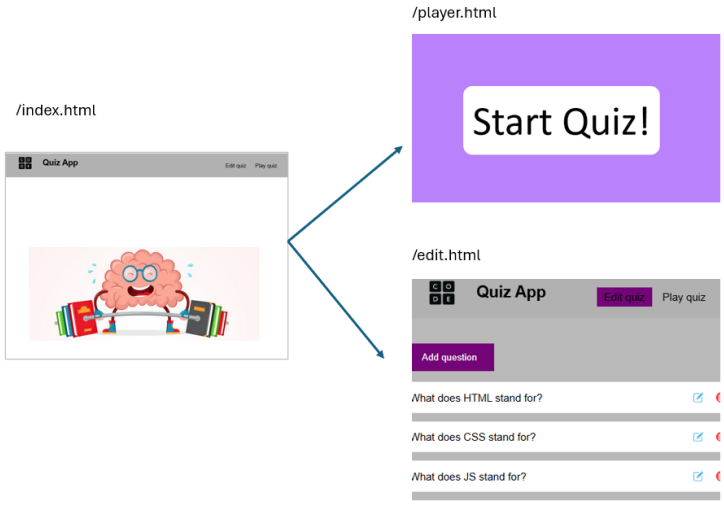
*STEP -3 – The **menu** and **separated view***

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



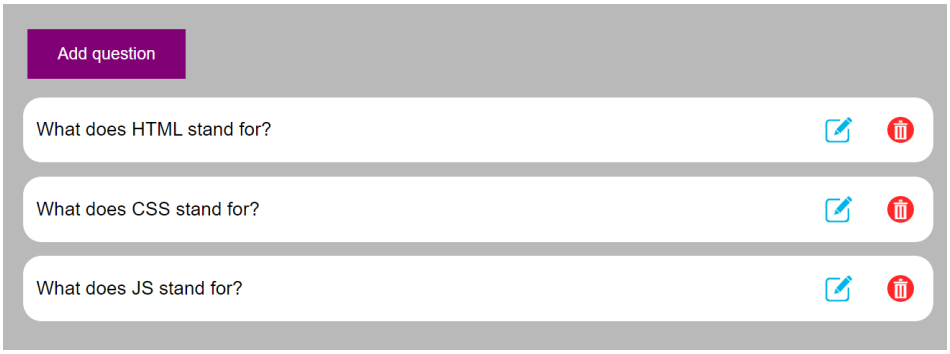
Don't forget to include the menu in the 2 sub views, and to display the selected menu item

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

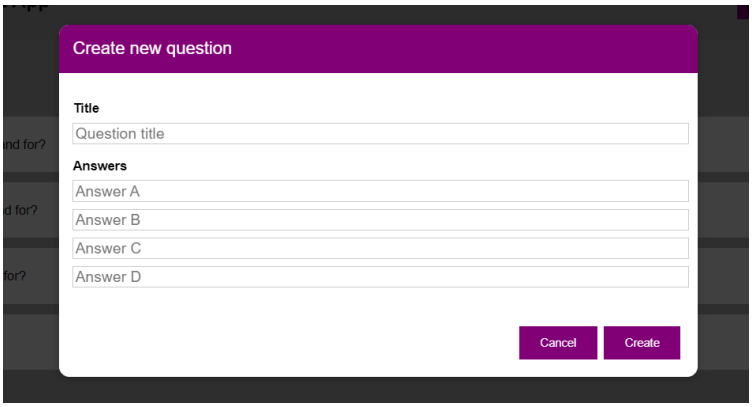
*STEP -4 – The **edition** view*

Implement the JS code to comply with the edition requirements:

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

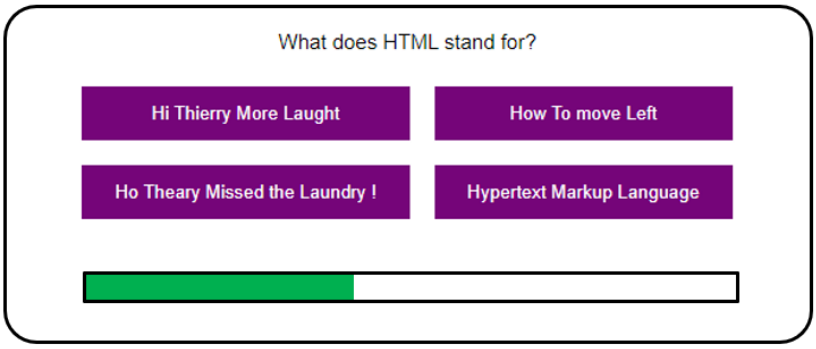


- ✓ When clicking on edit or add, a dialog is displayed to edit the question



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