**Assignment 2 – Vue.js: Who Wants to be a Millionaire Game**

# Purpose:

Using Vue.js, create the game ‘Who Wants to be a Millionaire’. This assignment requires you to use vue.js, async/await, audio API, speech synthesis API, speech recognition API, keypress detection, and to integrate client-side scripts with server-side scripts. Your mark counts for 13% of your final grade.

Due Date:  **Sunday, Dec. 9 @ 11:59 pm**

**Submission Requirements via Blackboard:**

* A link to your private Github repo
* Ensure you invite me as a collaborator to your repo so I can see your code
* A hosted url where you’ve uploaded your code (Azure or other, NOT codepen)

All work must be your own. Failure to submit an independent assignment will result in a grade of zero.

# What you’ll need:

* Vue.js : <script src="https://cdn.jsdelivr.net/npm/vue/dist/vue.js"></script>
* An open trivia API: <https://opentdb.com/api_config.php>
* You may want to research on youtube how the gameshow’s behaviour/sounds are done.
* Create your own HTML/CSS game page similar to the following screenshot. You can choose a different background from images.google.com if you wish. The level of detail of your recreation is up to you, but there should be an area on the right hand side to keep score, a bottom area for the question/answers, a left area for the life-lines, and a top area for ask an audience.  
  

# Application Requirements:

1. Use of vue.js throughout the entire app
   1. Inclusion of vue.js library on the html page
   2. Instantiation of new Vue({}) is present
   3. Vue’s el, data, methods, and mounted properties are used appropriately
2. Display trivia question and answers using async/await
   1. Use of async/await/fetch to get json data from opentdb.com’s API.
   2. Use of v-for to display the scoreboard.
   3. Use of vue’s mustache/handlebar templates to display question/answers
3. Respond to click events and keypress events
   1. Use of v-on:click or @click to respond if the user clicks on A, B, C, D.
   2. Use of vue’s methods which will run upon the user clicking on one of the options above
   3. Use of v-on:keydown or @keydown to detect keypresses on those same options above that will respond appropriately
4. Make browser speak as if it was the host asking the question
   1. Use of speech Synthesis API to speak question and answers
   2. Make it speak every new question/answers at the appropriate time
5. Play background game music and sound effects
   1. Use of audio API to play background game music
   2. Vary the game music when user reaches level 6 and level 11 and play appropriate sound effect(s) for correct answers, and the appropriate sound effect for the wrong answer.
   3. Create functional buttons to make the music louder, softer, and pause/play
6. Use speech recognition so that if you speak to the game, it responds appropriately
   1. For example, if you say “A, final answer” the browser will respond as if you clicked A.
7. Document your code throughout. If you decide to attempt any bonus challenges, add comments near the top of your code what I’m to look out for so as not to miss it.

**Evaluation Method**

Your work will be evaluated based on how your script performs when I load and try your web page on my local Chrome browser.

# Evaluation Criteria (max 16 marks + 4 possible bonus marks)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Criteria** | **0** | **1** | **1** | **1** | **Marks** |
| **Use of vue.js throughout entire app** | - no vue.js library is included | - included vue.js library | - instantiation of new Vue({}) is present | - Vue’s el, data, methods, mounted properties, etc are used appropriately | **3** |
| **Display trivia question and answers using async/await** | - no async/await found OR  - No question is displayed | - used async/await to fetch JSON data from API. | - used v-for to display scoreboard | - Use of vue’s declarative rendering via mustache/handlebar tempaltes to display question/answers | **3** |
| Respond to click events and keypress detection | - No click events found | - Use of v-on:click or @click if user clicks A,B,C, or D. | - use of vue’s methods which will run upon the user clicking A, B, C, or D. | - Use of v-on:keydown or @keydown to detect keypresses on those same options above that will respond appropriately | **3** |
| Make browser speak as if it was the gameshow host | - No code implemented for speech synthesis API | - Use of speech Synthesis API to speak question and answers | - each time the user gets to the next level, make it speak the new question and answers |  | **2** |
| **Play background game music and sound effects** | - didn’t use audio API | - use of audio API to play background game music | - vary the game music when user reaches level 6 and level 11  - play appropriate sound effects for correct and incorrect answers. | - create functional buttons to make music louder, softer, and pause/play | **3** |
| **Use speech recognition API** | - no speech recognition API used | - if you say “A, final answer” the game will respond appropriately | n/a | n/a | **1** |
| **Comments + Github commits** | No comments | Comments present | Tip: “A comment doesn't say what, a comment says why. Or alternatively, if you've had to write a line of code that could be confusing, you could explain how.”  – Kyle Simpson, author of YDKJS | | **1** |
| **BONUS** | Make it so that the user can optionally pause before saying “final answer” such that if the user says, “A” the browser will highlight the A button. Then when the user says “final answer” your code will respond as if the user clicked the A button. | | | | **1** |
| **BONUS** | Create the 50:50 lifeline by creating your own algorithm using Math.random or by using a method in the lodash library. Clicking the 50:50 button removes 2 incorrect answers. | | | | **1** |
| **BONUS** | Create the ask an audience lifeline by using a server to collect other people’s responses, and then using a graph/chart library to display the audience’s responses | | | | **1** |
| **BONUS** | Create the phone a friend lifeline by either using some kind of chat service via socket.io library, webRTC, or other API of your choice. The chat service could either be text-only, audio-only which uses the microphone, or both visual/audio which uses the webcam. | | | | **1** |