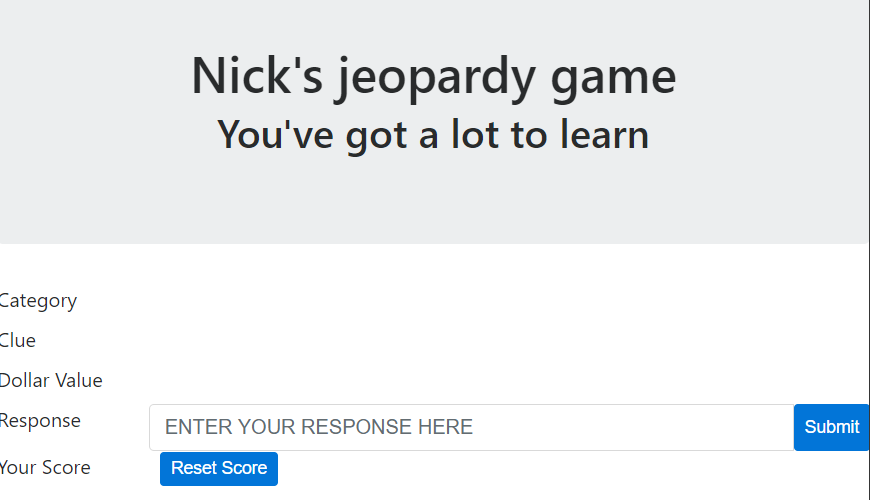
**Instructions**: Complete the following tasks to build a trivia website. You may use your course textbook, old sprints or code-alongs from class. You must do this project **individually**. You can use the internet but you can’t get help from anyone else. Any copying, plagiarism or cheating of any kind will result in a **grade of 0**.

**Task #1-Setup the database**

Open MySQL Workbench and run the DB Script **\_\_TRIVIA\_\_.sql**. It will create a new schema called “Trivia” with two tables called “categories” and “questions”. The tables will contain no records initially.

**Task #2-Setup a development sandbox**

Create a folder called **finalproject** in your **htdocs** directory. Unzip all the start files to your new finalproject directory. Open the files in VS Code and run the index.php file in a web browser. It should be like Figure #1 below:



**Figure #1**: index.php

**Task #3-Import some trivia questions from an API**

Open a web browser and go to: <http://cluebase.lukelav.in/clues/random>

This is a website that contains a database of several thousand trivia questions that have appeared on the TV game show “JEOPARDY”. By using the URL above, you can randomly get one question from this database. Notice the data comes back from the API in JSON format.

1. Create a new PHP file called **ImportQuestions.php** in your **finalproject** folder. End users will never interact with ImportQuestions.php, it will only be used to import questions from the API into our database. So ImportQuestions.php doesn’t need any HTML, CSS or JavaScript.
2. In your new ImportQuestions.php file, use the PHP curl functions to create an HTTP request to the cluebase url above and store the returning JSON in a PHP variable.
3. Write some PHP that will parse the JSON that contains the random question and insert into your categories and questions DB tables. You will need to get this data from the JSON (and ignore all other data):
4. Category
5. Question (or clue)
6. Answer (or response)
7. Value (dollar value of the question in the game)

Echo out these details to the screen and debug as necessary until you are sure you get the random question data correctly.

1. For the category, insert a record into the **categories** table. Store the **category\_id** that you just inserted in a variable, because you will need it to insert into the questions table. **HINT**: This article shows you how to grab the id of a record you just inserted, so you use that ID as a foreign key in the next insert: <https://www.w3schools.com/php/php_mysql_insert_lastid.asp>
2. Now use the category ID from the previous step and insert a record into the **questions** DB table.
3. CHECKPOINT: so far, your page should insert a new category and question every time the user hits the page. Make sure that part works before moving on.
4. So, importing 1 question at a time would take forever, right? Put your existing code in a loop of, let’s say 100, so that it will process 100 questions every time the user hits the page.
5. It’s possible if you run this PHP page enough times, you might get the same category again. We need to make sure we don’t have duplicate categories in our database. Add code in your loop to query the categories table to see if the category exists before it inserts a new category. If the category doesn’t exists in your DB yet, then go ahead and insert it. If the category does already exist, get the category\_id and insert the new question with that existing category\_id.
6. CHECKPOINT: If everything works so far, you should have several hundred questions and many categories in your database. If we had more time, we could add security to this page to make sure not just ANY user could hit this webpage, but don’t worry about that for this project.

**Task #4-Fetch a question to display**

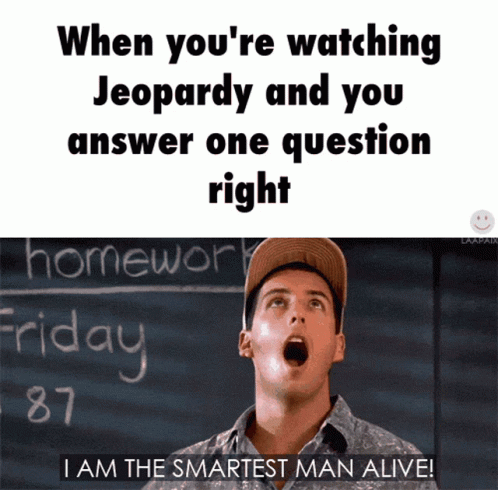
Create a new php file called **GetQuestion.php** in your **finalproject** folder. This page will not be accessed directly by the user, it will only be accessed through AJAX. Therefore, this page doesn’t need any HTML, CSS or JavaScript.

1. Add code to GetQuestion.php that will select one **random** question from the questions table (HINT: you will need to join with the categories tables).
2. You must return the question in JSON format so it can be processed by index.php (HINT: try putting the question data -- category, title, question, answer into an array and then use json\_encode to encode the array)
3. Echo the result of the json\_encode so that index.php will get it when the AJAX fires.

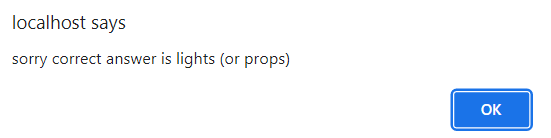
**Task #5-Display the question in index.php**

The index.php already calls the JavaScript function getQuestion() which doesn’t do anything yet.

1. Add some JS code to the getQuestion() in jeopardy.js that will call getQuestion.php and get the details of a question.
2. Once you get the question data, use jQuery to display the relevant details in the form elements. HINT: You can use JSON.parse() to convert the json data into an array. Then use the array to populate the form elements like this: $("#lblClue").text(json[0].question); )
3. Obviously, don’t display the correct answer to the user. Store the correct answer in a JS variable so that your code can check the user’s answer later.

**Task #6**-Check the user’s answer

1. After the user enters their response and clicks “Submit”, you can write some JS code in the CheckAnswer() function to compare the answer and let them know if they are correct or not using an alert box as in Figure #2. If the user is wrong, include the correct answer in the alert.



**Figure #2**-alert box after user get question correct & wrong

1. Adjust the users score and display it so they can see it as in Figure #3.



**Figure #3**-Score displayed

1. After the user clicks “OK” on the alert box, the checkQuestion() function should automatically call getQuestion() again to get a new question to display.
2. The “**Reset** to 0 using a little JS.

**Task #7**- Validating answer **Score**” button will simply set the scores

What if the user has a minor typo in their response, should they still get credit? I think so! You can write some JS code to validate their answer with the correct answer but allow for minor spelling mistakes. You can use this same logic to make sure the answer doesn’t have to be case-sensitive. You can research the algorithm to see how you can write some JS code that will check if a given string is “close” to another string. That way you can still give the user credit if their answer is close to the correct answer as in figure #4 below where the user misspelled “Jefferson Davis”.

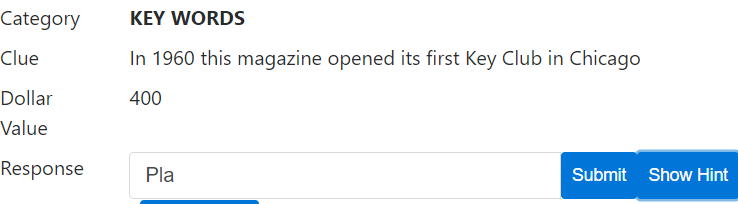
A screenshot of a computer

Description automatically generated

**Figure #4**-User has a typo but still gets credit for a correct answer

**Task #8**-Extra Credit-giving a “Hint” to the user

To make your website a little more interactive, you could allow the user to request a “Hint”. Just add a button that says “Show Hint”. The “Show Hint” button will call a little JS code that will display the first letter of the correct answer in the response textbox. The user can click the button multiple times to show multiple letters of the correct response.



**Figure #5**-The user selects “Show Hint” to see the first letter

RUBRIC (30pts):

Questions are importing properly from ImportQuestions.php (Task 3)– 9pts

GetQuestion.php retrieves a question and returns it according to the requirements (Task 4) – 5 pts

Display a question in index.php (Task 5) – 9pts

User answer is verified properly and score is updated accordingly (Task #6)-5pts

Logic is implemented to allow for minor spelling mistakes in user answer (Task #7) – 2pts

BONUS-“Hint” logic implemented to show 1 character of the correct answer at a time (Task #8) – 1pt