
JAVASCRIPT

EXERCISES

home / exercises / javascript

#exercises


#javascript

#web



JAVASCRIPT EXERCISES

PART I - DOM


1. INTRODUCTION

- Download the following file:  [products.zip](#)
 - Extract it into a folder inside your *public_html* folder.
 - Inside that folder, you will find an HTML document that references a CSS and a Javascript file.
 - The HTML document contains a *form*, with a line where the user can input a product description and its desired quantity.
-

2. HELLO WORLD

- Open the *javascript* file (*script.js*).
- Use the [alert](#)  function to open a dialog saying 'Hello World!'.
- Reload the document to see if it works.
- Now try using the [console.log](#)  function to do the same thing.
- Reload and inspect the *javascript* console to check if it is working.

3. SELECTING ELEMENTS

- Clear the *javascript* file.
- Use the `getElementById`  function to select the table with id 'products' and assign it to a variable called *products*.
- Then try writing that variable to the console:

```
let products = document.getElementById('products');  
console.log(products);
```




- Notice that it doesn't work. This happens because the *javascript* code is being loaded and run before the DOM is completely loaded and ready.
- Try fixing it by running the code inside the *load* event of the *window* object:

```
window.addEventListener('load', function() {  
    let products = document.getElementById('products');  
    console.log(products);  
});
```

- Another way to fix this problem is to add the *defer* attribute to the *script* element in the *HTML* file.
 - This makes the browser wait for the DOM to be completely loaded before running the script.
 - Try it.
-

4. SELECTING ELEMENTS

- Clear the *javascript* file.
- Select and print to the console the following elements:

- The form (use `getElementsByTagName` )
 - The second input inside the form (use `querySelector` )
 - All the inputs inside the form (use `querySelectorAll` ) and a *for loop* to print each one of them.
 - Repeat but this time use the *innerHTML* attribute to print the HTML code of each element.
-




5. EVENTS

- Add a *submit* event listener to the form so that it opens a dialog saying 'Submitted!' when the user presses the *Add* button.

```
let form = document.getElementsByTagName('form')[0];
form.addEventListener('submit', function() {
  alert('Submitted!');
});
```

- Use the `preventDefault`  function to prevent the form from being submitted.
-

6. ADD PRODUCT



- Alter the function that is called when the form is submitted so that it adds a new line to the table.
- Use the following functions and attributes:
 - `createElement`  to create a new line.
 - `innerHTML`  to add the data as HTML code to the new line.
 - `append`  to add the new line to the table.
- Each product line should have the following format:

```
<tr><td>Apple</td><td><input value="10"></td><td><input type="button" value="Remove"></td></tr>
```

7. REMOVE PRODUCT

- Make it so that when the 'Remove' button is pressed, the line the button belongs to is removed.

8. TOTAL

- Make the text inside the *span* with *id total* update whenever a line is added, removed or a quantity in a line is changed.
- The value of the span should be the sum of all the quantities in the form.
- Use the `change`  event of the *input* object.
- Also try using the `keydown`  event of the *input* object instead.

PART II - AJAX2

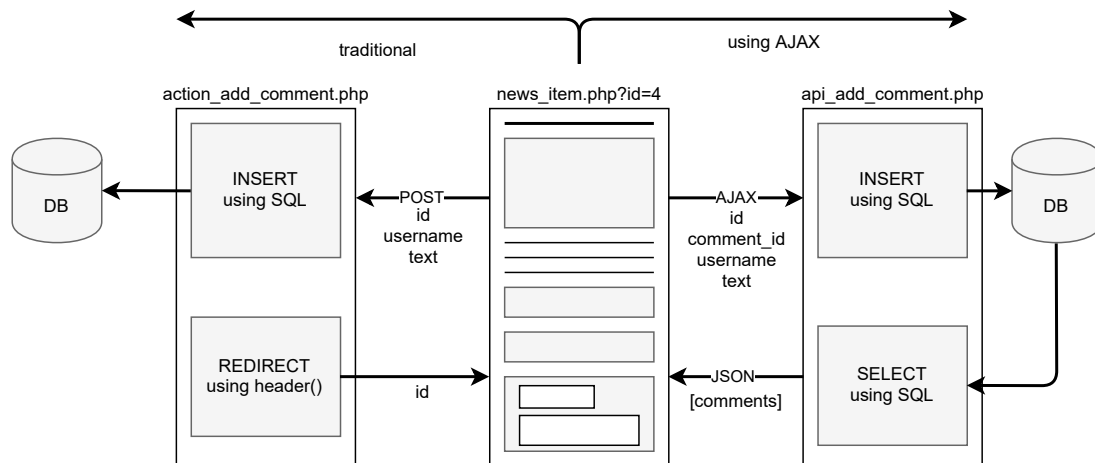
1. INTRODUCTION

In this exercise we will add the ability to add new comments in the newspaper application we developed in previous exercises.

In traditional web development, data would be sent to an *action* page using HTTP's POST method. That page would then insert the data into the database and redirect back to the page containing the form triggering a full reload of that page.

In this exercise we will implement this same functionality using *AJAX* (Asynchronous JavaScript And XML). When a user tries to insert a new comment, a HTTP request will be done in the background using *Javascript*;

the result of that request will then be used to update the page without forcing a complete refresh.



The following steps will guide you into developing your first AJAX based functionality:

2. DOWNLOAD BASE CODE

- Download the solution to step 7 of the PHP exercises: [solution_php_7.zip](#)
- Extract it into a folder inside your *public_html* folder.

3. COMMENT IDS

- Change the *list_comments* template so that it includes a field containing the id of each comment.
- Make sure that comments now look like this (use “view source code”):

```
<article class="comment">
  <span class="id">2</span> <!-- the comment id -->
  <span class="user">Abril Cooley</span>
  <span class="date">2017-10-17 15:40:32</span>
  <p>Duis scelerisque purus fermentum turpis euismod congue.
  Phasellus sit amet sem mollis, imperdiet quam porta, volutpat purus. In
  et sodales urna, sed cursus lectus. Vivamus a massa vitae nisl lobortis
```

laoreet nec tristique magna. Mauris egestas ipsum eu sem lacinia.</p>
</article>

- Also change the CSS code so that this field remains hidden from the user.

4. CREATE JAVASCRIPT FILE

- Create a new file called *script.js* that will contain all your *javascript* code.
- Link that file to your HTML code in the *header* template. Add the *defer* attribute to the *script* tag.

5. CAPTURE THE SUBMIT EVENT

- In your *Javascript* file start by creating a variable called *commentForm* that contains the *form* element (use the *querySelector* method).
- Using the *addEventListener* method, make it so that when the user submits the form (*submit* event) a function called *submitComment* is called.
- In this function, use the *preventDefault* method so that the page isn't reloaded when the user submits a new comment.
- Use *console.log* to write something to the console to test if the function is being called properly.

5. COLLECT DATA

- In the *submitComment* function, create 4 variables that will contain the data that will be sent to the server: the *id* of the news item, the *id* of the last comment (*comment_id*), the *username* and the comment *text*.

- Use the *querySelector* method together with the *value* and *textContent* attributes to get these values:
 - The *id* and *username* values can be fetched from the *form input* fields (using *value*).
 - The *text* can be fetched from the *form textarea* field (using *value*).
 - The *comment_id* can be fetched from the *span* you created in **step 3**. Get the last comment and use *textContent*.
 - Print these values to the console to verify if they are getting correctly collected.
-

6. SEND AJAX REQUEST

- In the *submitComment* function, create an *XMLHttpRequest* object ([AJAX slides](#)).
 - Use the *addEventListener* function so that when the AJAX request loads, a function called *receiveComments* is called.
 - Use the *open* function so that the AJAX request is done using POST, *asynchronously* and to a PHP file called *api_add_comment.php*.
 - Use the *setRequestHeader* function to set the correct *Content-Type* header for POST requests (see the slides).
 - Use the *send* function to send the data (*id*, *comment_id*, *username* and *text*). Use the *encodeForAjax* function described in the slides to help in this step.
 - Open the developer tools in your browser and check the network tab to see if the request is being sent correctly.
-

7. COMMENTS API

- Create a new file called *api_add_comment.php* that will:

- Receive the news item **id**, a **username**, some **text** and the id of the last comment (**comment_id**) present in the news item page.
 - Insert the new comment into the database.
 - Fetch all comments after the received **comment_id**.
 - Print those comments in JSON format (using *json_encode*).
- Open the developer tools in your browser and check the network tab to see if the response is being received correctly.

8. INSERT NEW COMMENTS

- Back in the *javascript* file, create the *receiveComments* function.
- Use the *JSON.parse* function to parse the response received.
- Loop over the received comments and add them to the HTML DOM (using *createElement* and *insertBefore*).

Copyright © André Restivo