HOW I IMPLEMENTED RANDOM DOGS API

INTRODUCTION

I chose to implement the Random Dogs API (https://dog.ceo/dog-api/documentation/random) on a side project solely for the use of the API. In this document, I will be explaining the steps I took to create this project and how you can replicate this easily.

IMPLEMENTATION

Step 1: Set up the HTML Structure: I started by creating a HTML file and setting up the basic structure. Include the necessary HTML tags such as **<!DOCTYPE html>**, **<html>**, **<head>**, and **<body>**. Add the required meta tags and title for the web page. You can also link external stylesheets for styling.

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Random Dog API</title>
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">
    <link rel="stylesheet" href="/css/dog.css">
</head>
```

Step 2: Create the Button and Image Elements. Inside the **<body>** section of your HTML file, create a button element with an appropriate ID, such as "get-dog". This button will trigger the API request. Also, add an **<img>** element with an ID, such as "dog", where the fetched dog image will be displayed.

```
<body>
    <div class="columns is-centered">
        <button class="button is-success" id="get-dog">Get a random dog image</button>
    </div><br>
    <div id="dogDiv" class="columns is-centered">
        <img id="dog">
    </div>
```

Step 3: Add Some JavaScript Code: Create a new JS file such as “dog.js” and link it to your HTML file using the ‘<script>’ tag.

```
    <script src="js/dog.js"></script>
</body>
</html>
```

In my dog.js file, I first of all use **document.getElementById('get-dog');** to find the HTML element with the ID “**get-dog**” and assign it to the variable ‘**dogButton’.** This element represents the button on the page that we want to listen for clicks on. We then add a click event listener to the ‘dogButton’ element using ‘**addEventListener()**’. So when the button is clicked, the function is executed.

```
const dogButton = document.getElementById('get-dog');

dogButton.addEventListener('click', function () {
  getData()
    .then((response) => {
      const dogImg = document.getElementById('dog');
      dogImg.src = response.message;
    })
    .catch((err) => {
      console.error(err);
    });
});
```

Inside the event listener function, we call the '**getData()**' function to fetch a random dog image from the Dog API. It returns a promise, allowing us to handle the result asynchronously using '**.then()**' and '**.catch()**'.

We then use the '**getData()**' function to set up a new promise. Inside the promise, we create a XMLHttpRequest object using '**new XMLHttpRequest()**'

```
// Function to fetch data from the Dog API
function getData() {
  return new Promise(function (resolve, reject) {
    // Create a new XMLHttpRequest object
    const xhr = new XMLHttpRequest();
```

Next, we open a GET request to the Dog API endpoint URL using '`xhr.open('GET', 'https://dog.ceo/api/breeds/image/random', true)`' and then send the request to the server using '**xhr.send()**'

```
    // Open a GET request to the Dog API endpoint
    xhr.open('GET', 'https://dog.ceo/api/breeds/image/random', true);

    // Send the request
    xhr.send();

```

We then write a code to listen for changes in the request state using '**xhr.onreadystatechange**'. When the state changes, the provided function is executed.

```
33      // Listen for changes in the request state
34      xhr.onreadystatechange = function () {
35        // When the request is complete
36        if (xhr.readyState === 4) {
37          // If the request is successful
38          if (xhr.status === 200) {
39            // Parse the response as JSON
40            const respJSON = JSON.parse(xhr.responseText);
41
42            // Resolve the promise with the parsed JSON response
43            resolve(respJSON);
44          } else {
45            // If the request fails, reject the promise with the HTTP status code
46            reject(xhr.status);
47          }
48        }
49      };
50    });
51  }
```

Inside the '**onreadystatechange**' function in line 34 we check if the request is complete ('**xhr.readyState === 4**'). If it is complete, we check if the request was successful ('**xhr.status === 200**').

If the request was successful, we parse the response text as JSON using '**JSON.parse(xhr.responseText)**' and assign it to the '**respJSON**' variable as seen as on line 38-40.

After which, we then resolve the promise with the parsed JSON response using '**resolve(respJSON)**' and if the request fails, we reject the promise with the HTTP status code using '**reject(xhr.status)**' as seen as from line 43-46

Finally, when the promise is resolved or rejected, we handle the result using '.**then()**'and '**.catch()**' in the click event listener.

In summary, the JavaScript code sets up an event listener on a button element. When the button is clicked, it triggers an HTTP request to the Dog API to fetch a random dog image. The fetched image URL is then updated in the HTML page. The code uses XMLHttpRequest and promises to handle the asynchronous nature of the HTTP request and provide a way to handle both successful and failed responses.

Step 4: Styling. You can choose to style the project however you like. I personally didn't style it as much because I was looking for a simple design, but if you really want to, create a css file and code your css there. DO NOT FORGET to link your stylesheet to your html file as seen here;

```
8       <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bulma@0.9.4/css/bulma.min.css">
9       <link rel="stylesheet" href="/css/dog.css">
```