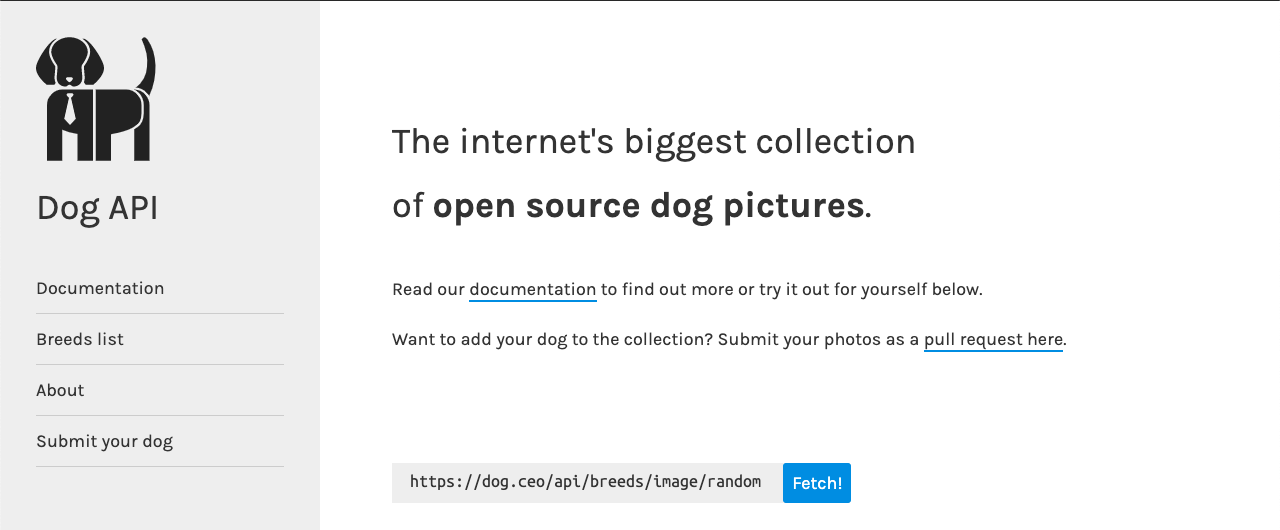
AJAX, AJAX request and fetch

AJAX and fetch to get data from API and build a dog therapy project:

**JSON Data and AJAX Requests**

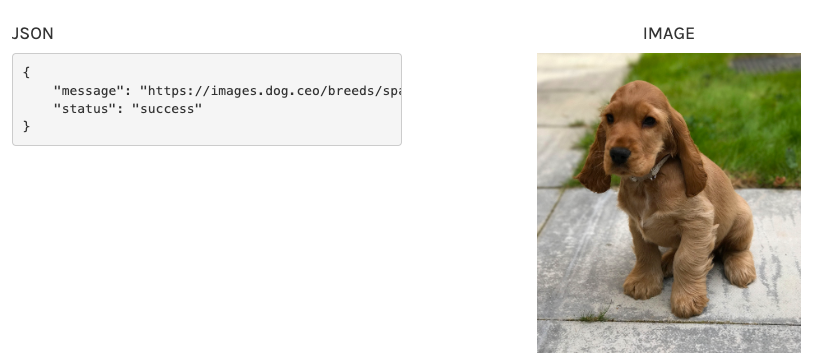
In this lesson, we are going to walk through interacting with the DOG API in the console before we try to build out our page. I'll walk through this in the next video if you prefer to watch me do this rather than just read about it.

First, let's head over to <https://dog.ceo/dog-api/>and you should see the following:



Notice the URL https://dog.ceo/api/breeds/image/random  and the button "Fetch"!

When we make an AJAX request to this URL we see that this should be our expected response:

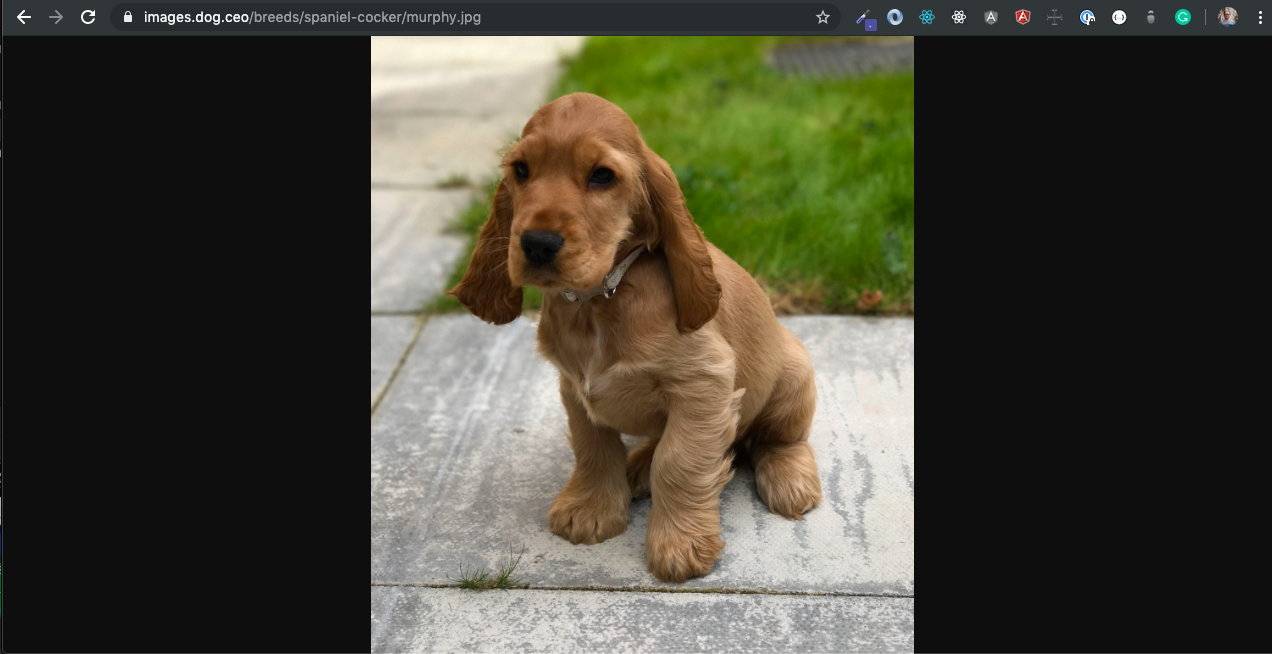


We get this JSON data. **JSON** stands for **JavaScript Object Notation**. We have looked at JSON data in our Todo App with Local Storage in the past. The most basic syntax required for a JSON object are these opening and closing { } and all the properties and values must be surrounded in " " quotes.

Pay special attention to this. They must be " " quotes not ' ' quotes!

When we make an AJAX request to the endpoint  https://dog.ceo/api/breeds/image/random we get a JSON object as a response with a property message with a value of https://images.dog.ceo/breeds/spaniel-cocker/murphy.jpg and a property of status with a value of success.

You can now take this url https://images.dog.ceo/breeds/spaniel-cocker/murphy.jpg and paste it into a browser and see an image.



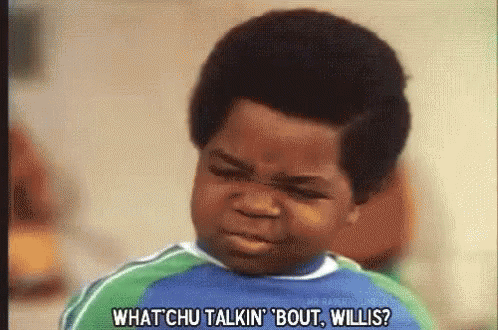
But how do I make an AJAX request to the endpoint https://dog.ceo/api/breeds/image/random ?

Wait! Before that, you may be still be wondering, "What is an AJAX request?!!"

Good question. AJAX stands for **A**synchronous **J**avaScript **A**nd **X**ML.

According to MDN:

"In a nutshell, it is the use of the [XMLHttpRequest](https://developer.mozilla.org/en/DOM/XMLHttpRequest) object to communicate with servers. It can send and receive information in various formats, including JSON, XML, HTML, and text files. AJAX’s most appealing characteristic is its "asynchronous" nature, which means it can communicate with the server, exchange data, and update the page without having to refresh the page."

[](https://www.google.com/url?sa=i&source=images&cd=&ved=2ahUKEwi1od-49ZvkAhUJheAKHePWDH0QjRx6BAgBEAQ&url=https%3A%2F%2Ftenor.com%2Fsearch%2Fwhatchu-talkin-bout-willis-gifs&psig=AOvVaw36Faef6qPzsf5w9Zw5g5R3&ust=1566750193562666)

Ok, that was probably too much information. Luckily, MDN breaks down the to major features that happen with an AJAX request:

* Make requests to the server without reloading the page
* Receive and work with data from the server

Ok, that still may be a bit confusing. It's ok if it is!

You may feel the complexity building up. How do we make an AJAX request from our program to get this JSON data and display it on our page?

That's a big task!

Fortunately, JavaScript reduces the complexity by giving us access to something called the fetch() method that we'll take a look at in our next lesson!

**How to Make an AJAX Request with the fetch() Method**

We left off the last lesson asking the question, **"How do we make an AJAX request from our program to get JSON data to display on our page?"**

The fetch() method will allow us to do this. Before we look at the fetch() method let's take a brief look at two alternatives. There is the [XMLHttpRequest](https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest) and [jQuery.ajax()](https://api.jquery.com/jquery.ajax/)approach.

If you take a look at the XMLHttpRequest documentation you will see that this can be a bit of a steep learning curve. I found this to make AJAX requests very unapproachable for me. However, over time you may find some benefits of this approach.

I'm just glad we do not have to take this approach in learning to make AJAX requests.

The next approach comes from the very user friendly jQuery library. jQuery.ajax(). It significantly lowers the barrier to entry. It requires using the jQuery library so we won't address it at this time.

The great news is that vanilla JS, what we all have access to without any special libraries, gives us the very user friendly fetch() method.

Remember our goal:

* We want to hit the endpoint: https://dog.ceo/api/breeds/image/random
* We want to get back this JSON:
  1. {
  2. "message": "https://images.dog.ceo/breeds/coonhound/n02089078\_4065.jpg",
  3. "status": "success"
  4. }

First, I'll show you the code it takes to pull this off then we'll walk through it step by step:

1. fetch('https://dog.ceo/api/breeds/image/random')
2. .then(function(response) {
3. return response.json();
4. })
5. .then(function(myJson) {
6. // do something with our JSON data ;
7. });

The most straightforward use of the fetch() method accepts one argument, the resource path you want to fetch, which returns a promise containing a Response object. Let's take a look at this in the console:

A computer screen shot of a program

Description automatically generated

This response is not the actual JSON we want. It is just the HTTP response. We will need to use the json() method to extract the JSON body content from the response. We'll also need another tool to help us when our promise is resolved or rejected, *then()*. This method takes two functions as parameters.

1. First function is executed if promise is resolved and a result is received. - We will act on this in our next step.
2. Second function is executed if promise is rejected and an error is received. - We will skip over implementing this for now.

A screen shot of a computer

Description automatically generated

Awesome! We have access to the message and status values from the JSON object.

So we will attach another *then() and pass in another function to execute if the promise is resolved and a result is achieved. This parameter name that we choose will be the value that is returned. Instead of just a response that we were working with we are now working with a JSON object so I'll name this parameter myJson. So, let's do a console log and print out our message and status values.*

A screen shot of a computer program

Description automatically generated

Great job! Now, with this information, we are ready to start building our pet therapy app!

**Build and Style our Page with HTML & CSS**

Getting Started

If you are familiar with HTML and CSS you will probably just want to copy and paste the following content into HTML and CSS files in the same repository. If you are still shaky on these concepts I cover HTML & CSS in the following video.

Getting Started

You may want to build this app out in a [Codepen](http://codepen.io/), however, because we want to deploy this to Netlify, I will show you in the videos the files I create and how I link them together in a text editor.

You will want to have a folder and include three files. I will call my folder dog-therapy and add three files: index.html, main.js, styles.css

I would encourage you to use the same names as this if you are learning some of this for the first time. However, if you are comfortable with creating folders and files and linking your index.html file to your js and css files then feel free to use whatever names you would like.

In your index.js file add the following code:

1. <div class='main'>
2. <div class="card">
4. <img src='https://images.dog.ceo/breeds/spaniel-cocker/murphy.jpg'/>
5. <div class="container">
6. <h1>Dog Therapy</h1>
7. <button>
8. <b>Fetch Dog</b></button>
9. </div>
10. </div>
11. </div>

The result should look something like this:

A puppy sitting on a sidewalk

Description automatically generated

Great, now that we have the structure of our page set up let's go ahead and add some styling. In your styles.css file let's add the following code:

1. .main {
2. display: grid;
3. place-items: center;
4. height: 90vh;
5. }
7. .card {
8. box-shadow: 0 4px 8px 0 rgba(0,0,0,0.2);
9. transition: 0.3s;
10. background: white;
11. max-width: 350px;
12. }
14. .card:hover {
15. box-shadow: 0 8px 16px 0 rgba(0,0,0,0.2);
16. }
18. img {
19. width: 100%;
20. }
22. h1 {
23. text-align: center;
24. }
26. button {
27. padding: 15px 0;
28. background: #FFE600;
29. width: 100%
30. }

The result of our styling should resemble the following:

A dog sitting on a sidewalk

Description automatically generated

If you would like more help you can watch the next video. In the first, I will build out the HTML & CSS.

Many courses on JavaScript just tell you to paste in the HTML and CSS. But what I have found is that often times when someone is learning JavaScript they are also at the same time still learning the foundations of HTML and CSS. So I want to give you a quick video in which you can code along to make sure you understand the codebase we are working with when you begin to implement your JavaScript.

If the HTML and CSS are straightforward for you then please jump ahead to the section on JavaScript!

**Dog Therapy - JavaScript**

Now that we have our foundation of HTML and styling of CSS, let's build the functionality of our image generator.

Let's implement our goal of displaying a random dog image. If we look at our current page we will see that all we have is a static page with a default dog image that doesn't respond to user interaction. We can click on a button and it doesn't do anything.

The first task specifies that when the “Fetch dog” button is clicked, a new random image is displayed to the user and replaces the default image.

To accomplish this, the first thing to do is to store a reference to the image tag in our JavaScript code.

If you peer into the HTML pane, you will see the markup for the default image as follows:

<img src='https://images.dog.ceo/breeds/spaniel-cocker/murphy.jpg' />

If we want to target this image tag we could do this in a couple of ways. Because there is only one image on our page we could simply target the img tag. However, if more images are added to our project in the future that didn't relate to an image that we wanted to change then that could be a problem that we would have to refactor in the future.

So I'm going to add a class dog-image to the img tag:

<img class="dog-image" src='https://images.dog.ceo/breeds/spaniel-cocker/murphy.jpg' />

Great, now I want to add our functionality to the "Fetch dog" button.

We are going to take a different approach from our Quote Machine project. In that project, we put an event listener on the generate quote button when clicked that made the AJAX request to grab a random quote.

In this project, we will take a different approach. The prior approach for our quote machine wasn't wrong but there are a number of ways of solving the same task in JavaScript. I will mention that some approaches are better than others for the reason of performance, readability, easability of bug fixes, etc...

So let's look at our button that contains the text "Fetch dog":

<button>Fetch Dog</button>

We are going to use an onclick() event. This means that when a click event happens on this HTML element it will trigger some sort of response. The response we want to trigger will be a function that we will build out in our JS. So let's add an onclick and set it equal to a function fetchDog().

<button onclick='fetchDog()'><b>Fetch Dog</b></button>

Awesome, to recap what we have done:

1. We have added a class to our default image.
2. We have added an onclick event to our "Fetch dog" button.

Now, let's work on our JS to target that image and build out a function that makes an AJAX request when the "Fetch dog" button is clicked.

In our main.js file we are going to target our image with the class of dog-image.

var image = document.querySelector('.dog-image');

Our next step is the bulk of our project. We want to make an AJAX request to the Dog CEO API to get back a random image and replace the default image.

Below is the code for the fetchDog() function definition then we'll walk through the code in a moment:

1. function fetchDog() {
2. console.log("click registerd")
3. console.log(image);
4. fetch('https://dog.ceo/api/breeds/image/random').then(response => {
5. return response.json();
6. console.log(response.json())
7. }).then(jsonResponse => {
8. console.log("before", image.src); console.log(jsonResponse.message);
9. image.src=jsonResponse.message;
10. console.log("after", image.src);
11. })
12. }

If you paste this code in your text editor the application should run as we would expect. However, if you are like me then you don't learn well when you just copy and paste someone's solution. You need help seeing how to build out something step by step even if it seems like it takes a long time.

The first thing I would do is define the function and put in a couple console.log's to make sure that things are connected as expected.

In the main.js file I would start with:

1. function fetchDog() {
2. console.log("click registered")
3. console.log(image);
4. }

Now, when I click the "Fetch dog" button and open up the console I should see the following:

A black screen with white text

Description automatically generated

You should see the text "click registered" which means that we have successfully linked our method to our onclick event on our "Fetch dog" button.

Next, we should see the HTML for your img tag so we can be certain that we have targeted this correctly in our main.js file.

Now, we are ready to make our AJAX request to get our random image. We already walked through this process and how to do it in the console so please refer back to that lesson if any of this gets confusing!

We want to use the fetch() method on the endpoint and when we resolve that promise we want to convert it to JSON with the json() method. We'll then log to the console what we get to make sure that we are on the right track:

1. function fetchDog() {
2. console.log("click registerd")
3. console.log(image);
4. fetch('https://dog.ceo/api/breeds/image/random').then(response => {
5. return response.json();
6. }).then(jsonResponse => {
7. console.log(jsonResponse)
8. })
9. }

We should see the following in the console:

A screen shot of a computer

Description automatically generated

We are all set! We will have access to our message property and its value set to a random image with jsonResponse.message.

But how do we change the default image to this jsonResponse.message image provided when we make a fetch request with JavaScript?

We can do that by using JavaScript to target the src property on our image with the class of dog-image. Remember, we have targeted this with JS as the image variable.

1. function fetchDog() {
2. console.log("click registerd")
3. console.log(image);
4. fetch('https://dog.ceo/api/breeds/image/random').then(response => {
5. return response.json();
6. }).then(jsonResponse => {
7. console.log(jsonResponse.message);
8. console.log(image.src);
9. })
10. }

You should get the following in the console:

A screen shot of a computer program

Description automatically generated

We logged to the console the value of which will always be a random value.

We also logged to the console the default image src property value.

So the only thing we have to do is change the default value with the new random value that is fetched!

1. var image = document.querySelector('.dog-image');
3. function fetchDog() {
4. console.log("click registerd")
5. console.log(image);
6. fetch('https://dog.ceo/api/breeds/image/random').then(response => {
7. return response.json();
9. }).then(jsonResponse => {
10. image.src=jsonResponse.message;
11. })
12. }

And that is it! Every time you hit the "Fetch dog" button we will replace the src attribute value with our new image!

We'll go ahead and deploy our Dog Therapy project in the next page. In the meantime, enjoy this GIF:

[](https://giphy.com/explore/dogs)