**Introduction to Requests**

Have you ever wondered what happens after you click a "Submit" button on a web page? For instance, if you are submitting information, where does the information go? How is the information processed? The answer to the previous questions revolves around *HTTP requests*.

There are many types of HTTP requests. The four most commonly used types of HTTP requests are GET, POST, PUT, and DELETE. In this lesson, we'll cover GET and POST requests. If you want to learn more about the different HTTP requests, we recommend the following documentation:

* [Mozilla Developer Network: HTTP methods](https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods)

With a GET request, we're retrieving, or *getting*, information from some source (usually a website). For a POST request, we're *posting*information to a source that will process the information and send it back.

In this lesson, we will explain how to make GET and POST requests by using JavaScript's **XHR**object. We'll also incorporate query strings into our requests.

We'll use the Datamuse API for GET requests and the Rebrandly URL Shortener API for POST requests. To complete the exercise on POST, make sure you create a Rebrandly API Key by following the instructions in the article below:

* [Codecademy Articles: Rebrandly URL Shortener API](https://www.codecademy.com/articles/rebrandly-signup) .

**HTTP Requests**

One of JavaScript's greatest assets is its non-blocking properties, or that it is an *asynchronous* language.

Websites, like newspaper websites, take advantage of these non-blocking properties to provide a better user experience. Generally, a site's code is written so that users don't have to wait for a giant image to load before being allowed to read the actual article—rather, that text is rendered first and then the image can load in the background.

JavaScript uses an *event loop* to handle asynchronous function calls. When a program is run, function calls are made and added to a stack. The functions that make requests that need to wait for servers to respond then get sent to a separate queue. Once the stack has cleared, then the functions in the queue are executed.

Web developers use the event loop to create a smoother browsing experience by deciding when to call functions and how to handle asynchronous events. We'll be exploring one system of technologies called Asynchronous JavaScript and XML, or AJAX.

To read more about the event loop, read the MDN documentation:

* [MDN Documentation: Event Loop](https://developer.mozilla.org/en-US/docs/Web/JavaScript/EventLoop)

# XHR GET Requests I

Asynchronous JavaScript and XML (AJAX), enables requests to be made after the initial page load. Initially, AJAX was used only for XML formatted data, now it can be used to make requests that have many different formats.

[MDN Documentation: Extensible Markup Language (XML)](https://developer.mozilla.org/en-US/docs/XML_introduction).

Similarly, the XMLHttpRequest (XHR) API, named for XML, can be used to make many kinds of requests and supports other forms of data.

Remember, we use GET to retrieve data from a source. Take a look at the boilerplate code in the diagram to see how to make an XHR GET request.

We’ll construct this template from scratch in a different exercise and walk through what each step does.

<https://s3.amazonaws.com/codecademy-content/courses/intermediate-javascript-requests/diagrams/XHR+GET+transparent.svg>

**XHR GET Requests II**

We are going to reconstruct XHR GET request boilerplate code step-by-step until we have written a complete GET request.

Feel free to refer to the XHR GET diagram at any point while completing this exercise:

* [XHR GET diagram](https://s3.amazonaws.com/codecademy-content/courses/intermediate-javascript-requests/diagrams/XHR+GET+diagram.svg)

**XHR GET Requests III**

By this point, you have an idea of how to write the boilerplate code for an AJAX request using an XHR object.

In this exercise, you will incorporate that boilerplate code to make a GET request to the Datamuse API to search for words that rhyme!

* [Datamuse API Documentation](https://www.datamuse.com/api/)

**XHR GET Requests IV**

In the previous exercise, you made a GET request to the Datamuse API to find words that rhyme. In this exercise, we will create a request to set a topic and find adjectives that describe the input word using query strings.

A query string contains additional information to be sent with a request. The Datamuse API allows us to retrieve more specific data with query strings attached to the request URL.

* [Wiki: query string](https://en.wikipedia.org/wiki/Query_string)

A query string is separated from the URL using a ? character. After ?, you can then create a parameter which is a key value pair joined by a =. Examine the example below:

'https://api.datamuse.com/words?key=value'

If you want to add an additional parameter you will have to use the & character to separate your parameters. Like so:

'https://api.datamuse.com/words?key=value&anotherKey=anotherValue'

Let's incorporate this into our code!

**XHR POST Requests I**

Reminder: If you haven’t already signed up for an API Key from Rebrandly, please read this Rebrandly sign up guide.

* [Codecademy Articles: Rebrandly URL Shortener API](https://www.codecademy.com/articles/rebrandly-signup) .

Great! By this point you've signed up for an API key, and you know the essence of making a GET request. We will be making a POST request using the Rebrandly API.

The major difference between a GET request and POST request is that a POST request requires additional information to be sent through the request. This additional information is sent in the *body* of the post request.

We’ll walk through the code from the diagram and construct our own POST request in the next lesson.

<https://s3.amazonaws.com/codecademy-content/courses/intermediate-javascript-requests/diagrams/XHR+POST+transparent.svg>

**XHR POST Requests II**

We are going to reconstruct the code from the previous exercise step-by-step until we have written a complete AJAX POST request.

Feel free to refer to the XHR POST diagram at any point while completing this exercise:

* [XHR POST diagram](https://s3.amazonaws.com/codecademy-content/courses/intermediate-javascript-requests/diagrams/XHR+POST+diagram.svg)

**XHR Post Requests III**

Reminder: If you haven’t already signed up for an API Key from Rebrandly, please read the article:

* [Codecademy Articles: Rebrandly URL Shortener API](https://www.codecademy.com/articles/rebrandly-signup).

In this exercise, you'll be making a POST request to the Rebrandly API to shorten a URL.

Get ready! You’re now going to incorporate the previous lesson’s boilerplate code into making an actual POST request!

If you reset the exercise at any point, you will have to paste in your API key again at the top!

# Review Requests I

You’ve done an amazing job navigating through making XHR GET and POST requests! Take some time to review the core concepts before moving on to the next lesson.

1. JavaScript is the language of the web because of its asynchronous capabilities. AJAX, which stands for Asynchronous JavaScript and XML, is a set of tools that are used together to take advantage of JavaScript's asynchronous capabilities.
2. There are many HTTP request methods, two of which are GET and POST.
3. GET requests only request information from other sources.
4. POST methods can introduce new information to other sources in addition to requesting it.
5. GET requests can be written using an XMLHttpRequest object and vanilla JavaScript.
6. POST requests can also be written using an XMLHttpRequest object and vanilla JavaScript.
7. Writing GET and POST requests with XHR objects and vanilla JavaScript requires constructing the XHR object using new, setting the responseType, creating a function that will handle the response object, and opening and sending the request.
8. To add a query string to a URL endpoint you can use ? and include a parameter.
9. To provide additional parameters, use &and then include a key-value pair, joined by =.
10. Determining how to correctly write the requests and how to properly implement them requires carefully reading the documentation of the API with which you're working.