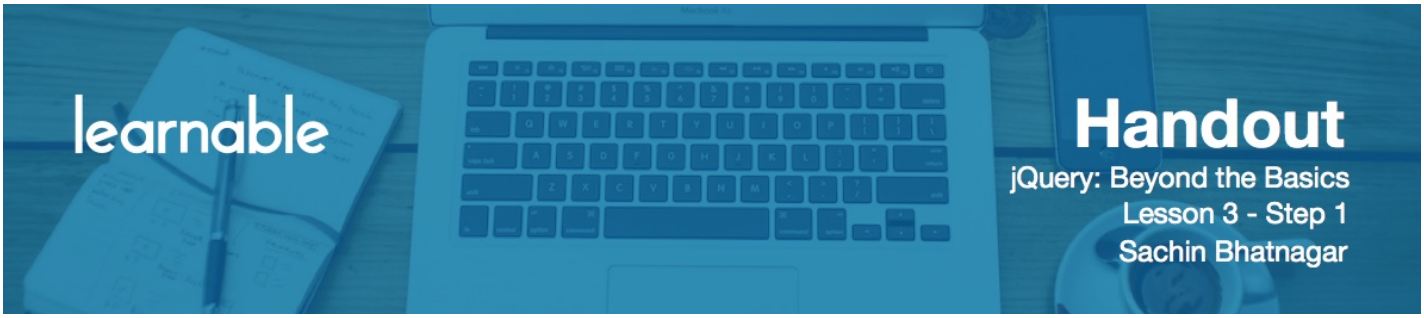

Introduction to AJAX

In this lesson, we will discuss **AJAX**.

At the end of 1990s a typical website's front-end was mostly limited to displaying static content. If a certain feature on the site had to be dynamic, then its processing had to be handled by the server. For example, if you submitted a form on the site, the contents of the form, and the resultant page would then be generated by something like a PHP script running on the server. The entire page had required a reload once it was rendered on the server. From a user experience standpoint a complete page reload is not welcome.

Today, we live in a world of fast and efficient JavaScript engines such as the V8 engine found in Chrome. Websites and web apps today access third-party services, and cloud-based back-end APIs. Many of such interactions are done with JavaScript.

Imagine you click a button in a modern-day web app, that requires some processing and fetching data from a server. The request would be handled by JavaScript, which will then proceed to send it in the background to the server in an asynchronous fashion. Once the server finishes processing, it sends the result back in either JSON or XML format, which is then parsed and rendered directly in the user interface section that needs the data, rather than reloading the entire page.

All of this is done by a set of technologies that are collectively known as **AJAX (Asynchronous JavaScript And XML)**, though the XML bit is something that is rarely used these days. AJAX is not a rocket science, as you'll soon see. The reason why you would want to use AJAX on your sites is the seamless user experience that it provides. No longer do you need to reload entire pages: AJAX let's you update specific portions of the page in a dynamic manner.

History of AJAX

The history of AJAX starts at Microsoft in 1996 with the introduction of the `iframe` tag. This tag allowed content from other documents and pages to be embedded directly into portions on a single web page. This meant that those frames may reload the content without the need to reload the entire page.

Microsoft proceeded to implement the **XMLHTTP ActiveX** feature in Internet Explorer 5, which was the beginning of what would ultimately become AJAX. The XMLHTTP was adopted by Mozilla, Safari and Opera as the `XMLHttpRequest()` object, which is essentially the heart and core of AJAX. A landmark moment in the evolution of technologies that lead to the rise of AJAX was when Google introduced AJAX in GMail, back in 2004.

The term "AJAX" itself was introduced by Jesse James Garret in 2005. Finally, in 2006, the World Wide Web Consortium formally released the first draft of technologies that are referred to as AJAX. But what we are really talking about here is essentially the `XMLHttpRequest()` object. It is responsible for sending and receiving data from a server, or a service, in an asynchronous manner using HTTP or HTTPS protocols.

AJAX in action

Let's say, you have a weather widget on your site which asks the user to choose a city so that they can see the weather updates that are fetched from a cloud-based weather service provider.

The user begins by typing in the city and the country name, and then clicks the "Get Weather" button. At this stage, the `XMLHttpRequest()` object can be used to send the city name as a JSON object to the weather service provider using a GET request. Once the weather service provider returns data in JSON format, it is received by the page, and the UI element is updated to show the resulting data.

Implementing AJAX with JavaScript

To demonstrate this I am going to use vanilla JavaScript without bringing jQuery in the picture:

```
var xhr = new XMLHttpRequest();

xhr.open("GET", "/api/city/london,uk", true);
xhr.onload = function() {
    var jsonData = JSON.parse(xhr.response);
    weatherStatus.innerHTML = jsonData.weather_status;
    currentTemp.innerHTML = jsonData.temp;
};

xhr.send();
```

We set `xhr` variable to a new instance of the `XMLHttpRequest()` object. The `open` method accepts three arguments:

- The request method ("GET" in our case).
- URL on the server, which needs to be called.
- Whether this AJAX call should be asynchronous.

The `onload` event listener will be fired once the `XMLHttpRequest()` receives data from the server. Inside the function we are simply parsing the results from our server.

To invoke the AJAX request we employ `xhr.send()`.

Introduction to jQuery AJAX handling methods

jQuery provides us with powerful and easy shorthand methods, including a full blown `ajax` method for advanced and custom configurations. We are going to discuss these methods throughout this lesson.