Ajax

Introduction

We're gonna learn about **AJAX** today. It's a very important concept and you'll use it frequently in **both** frontend and backend development.

AJAX stands for ***Asynchronous JavaScript And XML***. It is about updating a web page without reloading the entire web page. This is very useful when only some parts of a web page need to be changed.

Before learning about asynchronous, let us see what the meaning of **synchronous** calls is? If a ***synchronous call happens then any further operation on the client gets blocked,*** i.e. any further execution of operations stops executing until a response is not received. Like in JavaScript or other languages, the statements execute in a sequence.

Asynchronous means that the operations on the client do not get blocked. This means that the client can work on other things and does not need to wait for the response. Examples of asynchronous calls in JavaScript are - setTimeout(), setInterval(), etc.

AJAX is not a programming or a scripting language. It is a ***group of inter-related technologies*** like JavaScript, XML, HTML, CSS, and DOM.

AJAX has these benefits:
- Update a web page without reloading the page
- Receive and send data asynchronously
- Send only important data to the server
- Makes the application faster

EXTRA:
You can check out the link below to understand more about ajax -
***https://www.codementor.io/sheena/ajax-tutorial-web-development-du107rzaq***

HTTP REQUESTS

HTTP means **HyperText Transfer Protocol.** It is the underlying protocol used by the World Wide Web that *defines how messages are formatted and transmitted.*

HTTP is a **stateless protocol.** This means that the server does not require to maintain information or status about every user for the duration of multiple requests. This also means that the server does not need to maintain a continuous connection with the client.

In a stateless protocol, the server opens a connection for every request and closes it after the response is sent. Steps in HTTP requests processing -
- Client requests a connection with the server
- The server opens a connection with the client
- The client makes a request to the server
- The server processes the request
- The server sends a response to the client
- The client closes the connection
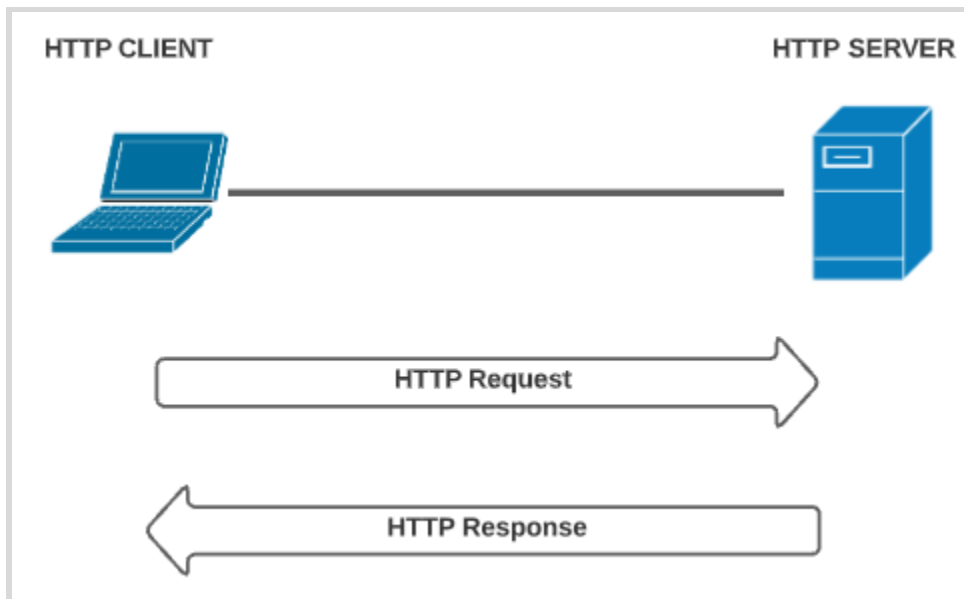
HTTP Requests

The **HTTP** *request method is used to indicate the action to be performed on the data transmitted to the server.* These are the types of request that you can make using HTTP:
- **GET** - requests for a data
- **HEAD** - requests for data but without the response body
- **POST** - submits data causing a change in state on the server
- **PUT** - replaces the specified data with request data
- **DELETE** - deletes the specified data

- **CONNECT** - establishes a tunnel to the server
- **OPTIONS** - describes the communication options for the target data
- **TRACE** - perform message loop-back test along the path to the target data
- **PATCH** - apply partial changes to data

_EXTRA_*: You can look at these requests from the link below -*
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

Response Codes

**The response codes also called status codes are a 3 digit integer to identify what has happened to the request sent to the server.** Responses are grouped in 5 classes which are identified by the first digit of the code -

➔ **1xx (Informational Resources)** - the request has been received and the process is continuing
➔ **2xx (Successful Responses) -** the request was successfully received, understood, and accepted
➔ **3xx (Redirection Messages)** - further action must be taken in order to complete the request
➔ **4xx (Client Error Responses)** - request contains incorrect syntax or cannot be fulfilled
➔ **5xx (Server Error Responses)** - server failed to fulfill an apparently valid request

_EXTRA_*: You can look at the above codes from the below link -*
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

API

API stands for _**Application Programming Interface.**_ In simple words, you can say that it is a _**medium that allows two applications to talk to each other.**_

With a web-based API, your browser/client sends an HTTP request. But instead of the response being delivered as a web page, it's received and returned in a specific format that applications can easily parse, like JSON.

So, an application sends data to a server. Then the server retrieves the data and interprets it. Some action is performed on the data. The server sends the information back to the application. The application interprets the data. The application presents the interpreted information in a readable way.
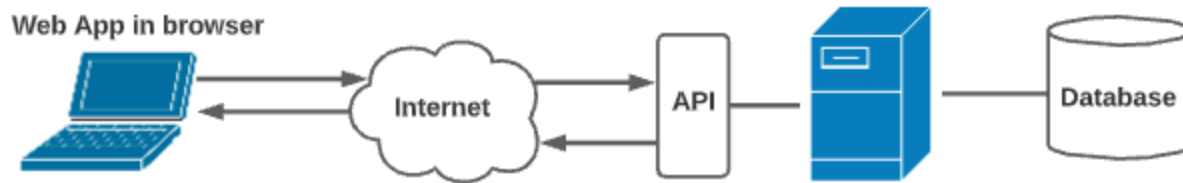
The communication between the application and the server is all done via API. The exchange of data is mostly done using JSON.

EXTRA:
Youcan read about API from the link below -
https://www.mulesoft.com/resources/api/what-is-an-api
https://medium.com/@perrysetgo/what-exactly-is-an-api-69f36968a41f

JSON

JavaScript Object Notation (JSON) is an acronym for JavaScript Object Notation. It is used in AJAX to **transfer data between a browser and a server**. It's **simple to understand**, and it's **faster than XML** at exchanging data. **Arrays, objects, strings, numbers,** and **values** are all supported.

Despite the fact that the 'X' in AJAX stands for XML, JSON is now **used more than XML** due to its many advantages, including being **lighter** and being part of JavaScript.

The syntax of JSON is *derived from the syntax of the JavaScript object notation*, but the **JSON format is text/string only**. So, it's a **list of names/value pairs,** but it's **not the same thing as an object.** The name is **enclosed in double quotations**, but this is not the case in JavaScript..

Example of JSON -

```
var myJSON = '{
  "name": "John Doe",
  "age": 21,
  "Employees_under": [
    { "firstName": "Aman", "lastName": "Davies" },
    { "firstName": "Saksham", "lastName": "Rana" },
    { "firstName": "Shubham", "lastName": "Chaurasia" }
  ]
```

```
}
```

JavaScript methods for JSON

The JSON format is syntactically identical to the code for creating JavaScript objects. Because of this similarity, a JavaScript program can easily convert JSON data into native JavaScript objects.

You can use the JavaScript built-in function **'JSON.parse()'** to *convert the string into a JavaScript object -*

```
var obj = JSON.parse(text);
```

If you have data stored in a JavaScript object, you can convert the object into JSON using 'JSON.stringify()' object -

```
var myJSON = JSON.stringify(myObj);
```


AJAX REQUEST

AJAX c*ommunicates asynchronously with the server using* the XMLHttpRequest object.

AJAX performs the following operations -
> ➢ In the background, sends data from the client.
> ➢ The data is received from the server.
> ➢ The web page is updated without having to be reloaded.

Create an **XMLHttpRequest** object, open the URL and **submit the request for an HTTP request.**
The code below shows an example of AJAX request -

```javascript
var xhrReq = new XMLHttpRequest();
xhrReq.onload = function() {
    var resJSON = JSON.parse(xhrReq.response);
    console.log(responseJSON);
};
xhrReq.open("get", "url", true);
xhrReq.send();
```

The **open()** method initializes a newly-created request.
It has the following syntax - XMLHttpRequest.open(method, url, async, user,

```
password)
```
The **parameters other than 'method' and 'url' are optional** but has to be in the sequence as shown in the syntax above.

The **'async'** option is a boolean parameter defining asynchronous request when true (by default is true) or synchronous request when it is false.

The **'response'** property returns the response body content.

The **'send()'** method sends the request to the server.
The syntax is - `XMLHttpRequest.send(body)`
The **'body'** is an optional parameter which is ignored when the request method is **'GET'** or **'HEAD'**.

The constructor function XMLHttpRequest isn't limited to only XML documents. It starts with **'XML'** because when it was created the main format that was originally used for Asynchronous Data Exchange was XML, but now JSON is mostly used.

EXTRA: You can look at the link below to know about all the methods and properties of XMLHttpRequest -
https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest

AJAX REQUEST USING JQUERY

Writing AJAX code can be a bit tricky because different browsers have different syntax for AJAX implementation. However, using jQuery you can use AJAX functionality with only few lines of code.
The jQuery '**ajax()**' method sends **asynchronous HTTP requests to the server.**
The syntax for ajax() is - `$.ajax( url [, settings ] )`

We will learn about these from the example below -
```
$.ajax({
    url: 'example.html',
    method: 'GET',
    data: { name: 'Rahul Kumar', age: 19 }
    success: function(data, status, xhr) {
        console.log(data);
    }
});
```

url - a string containing the URL to which the request is sent. The 'url' can be provided as seen in the syntax or there is also a url property in 'settings' as seen in the example.

method - the HTTP method to use for the request (e.g. "POST", "GET", "PUT").

data - data to be sent to the server. It can be JSON object, string or array.

success - a callback function to be executed when Ajax request succeeds.

The ***ajax() method returns a jqXHR*** object which allows you to ***assign multiple callbacks*** on a single request.
There is an ***alternative for success callback option -***
`jqXHR.done(function(data,status,xhr) {});`

EXTRA: You should check out the link below for other options as well -
http://api.jquery.com/jquery.ajax/
You can check out the link below for callback chains -
http://api.jquery.com/category/deferred-object/

Other Simpler Methods

All Ajax requests sent by jQuery are based on the **$.ajax()** function. It is often **not** necessary to call this function, given that there are **several higher-level alternatives available** and easier to use, such as **$.get()** and **$.load()**.

Let's see an example of this:

```
$.get("example.html", { name: 'Rahul Kumar', age: 19 },
function(data, status, xhr) {
    console.log(data);
});
```

This works similar to the ajax() method 'get' call. You can also write this get() method as -

```
$.get("example.html", { name: 'Rahul Kumar', age: 19 })
.done(function(data, status, xhr) {
    console.log(data);
});
```

Similar to this you can use other functions like post(), load(), delete(), etc.

HANDLING ERRORS

The ***AJAX also provides error handling***. So if a request gets failed somehow, you can run code for that as well, like displaying an error message or redirecting to an error page.

There are two ways to do error handling just like 'success' and 'done()'. This applies to all the AJAX method - ajax(), get(), post(), etc.

You can use the **'error'** property or '**fail()**' method. We can see an example below -

```
$.get("example.html", { name: 'Rahul Kumar', age: 19 })
    .done(function(data, status, xhr) {
        console.log(data);
    })
    .fail(function() {
        console.log("Request failed");
    })
```