Week 7: AJAX, Fetch & Wrap Up

INFSCI 2560
Web Standards & Technologies

Today's Agenda

JavaScript!

1. AJAX
2. Fetch API
3. Promises
4. Activity 7

Assignment 2 - Learn as JS Framework

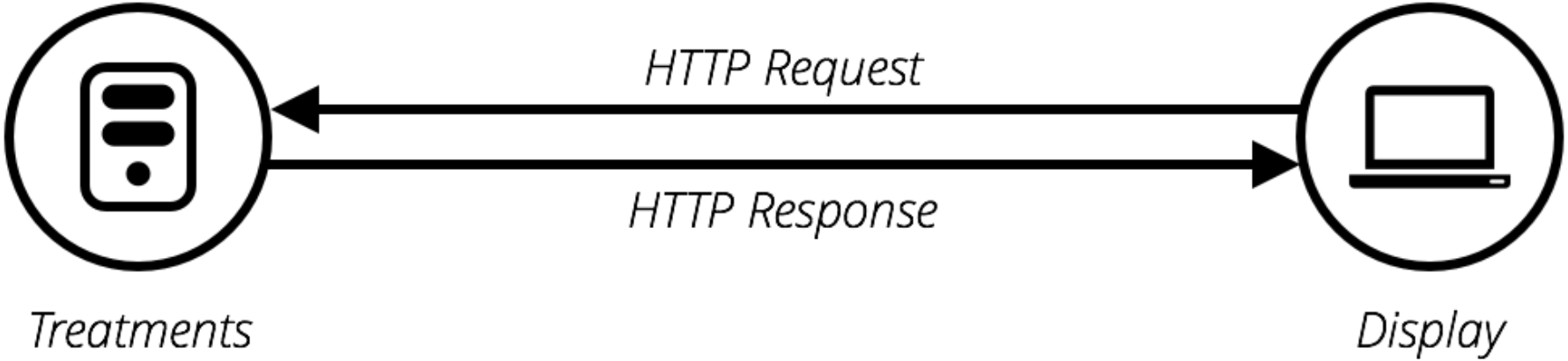
- There are too many JavaScript Frameworks
  - Constantly changing
- The “hot” framework keeps changing
  - JQuery then Angular then React then ???
- For this assignment you need to pick a front-end JavaScript framework to learn and build a small website/app taking advantage of that framework’s capabilities.
  - Learn to learn
- Two Major Components
  - Develop a website using that framework
  - Document what you learned
- DUE MARCH 9TH

AJAX

The Problem

- Normally, to update a web page with new information, regardless of how much information, requires a page refresh
- Sometimes you want to retrieve data from a server without reloading the entire page
  - New email
  - New Tweets
  - News alerts
  - Updated book/movie recommendations

Making incremental updates to web pages



- Request made to the server, the server would send HTML and other assets (images, CSS, JavaScript) back and the browser would render the page.
- How do we make incremental updates without reloading the page?

What is AJAX?

- The term stands for “Asynchronous JavaScript and XML”
- Enabled web pages to send and receive data asynchronously
  - In the background
- Not a standard, but a set of techniques
  - Built on top of standards
- Emerged as part of Internet Explorer in 1999
  - Only cutting edge thing IE did for the world
- AJAX is a misleading name. AJAX applications might use XML to transport data, but it is equally common to transport data as plain text or JSON text.
- GMail in 2004 was the first big, AJAX application

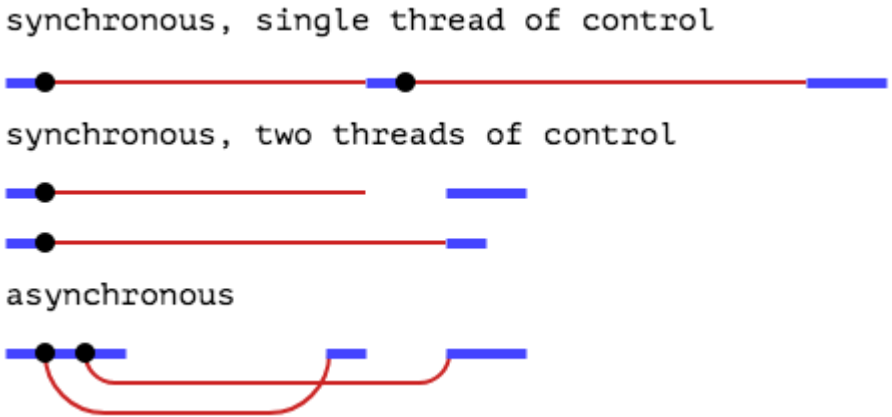
Why?

This is a really good thing because:

- Page updates are a lot quicker and you don't have to wait for the page to refresh, meaning that the site feels faster and more responsive.
  - This is main reason
- Less data is downloaded, meaning less wasted bandwidth. This may not be such a big issue on a desktop on a broadband connection, but it's a major issue on mobile devices and in developing countries that don't have ubiquitous fast Internet service.
  - Less of an issue because of local cache

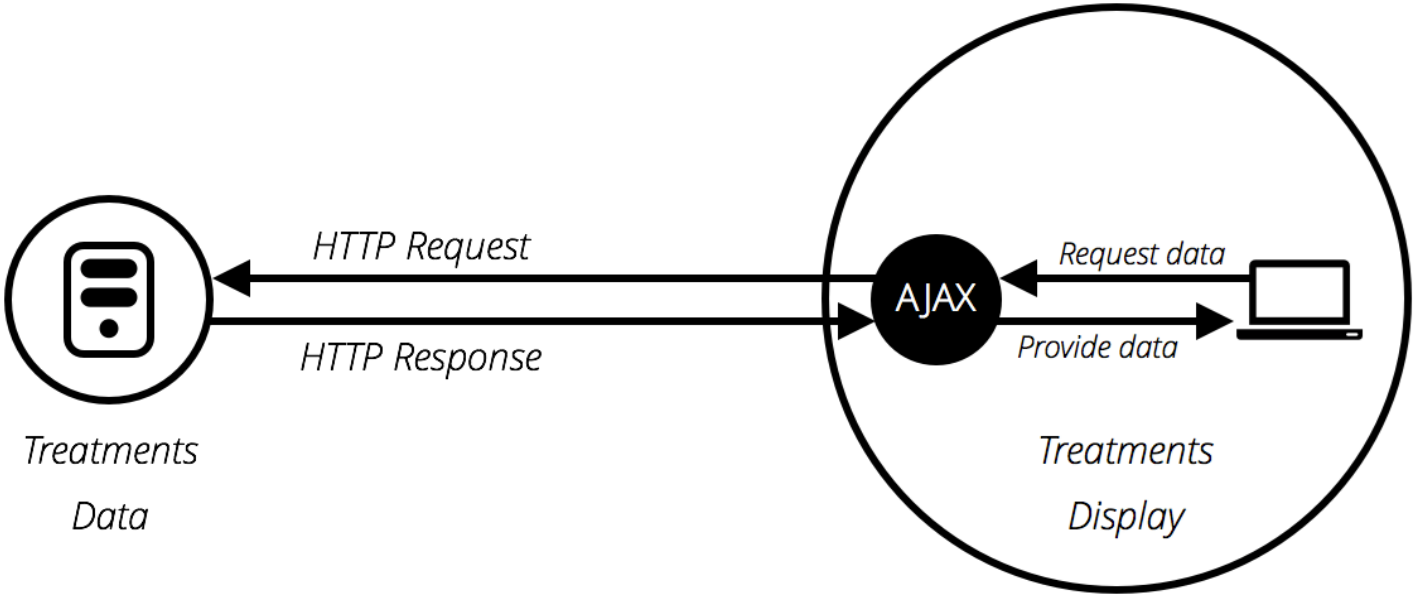
AJAX

- Allows us to make asynchronous requests
- Load data outside of the standard page request and loading process.
- Makes web pages faster and more responsive
- Cuts down on the amount of network traffic and loading required on the client and server.



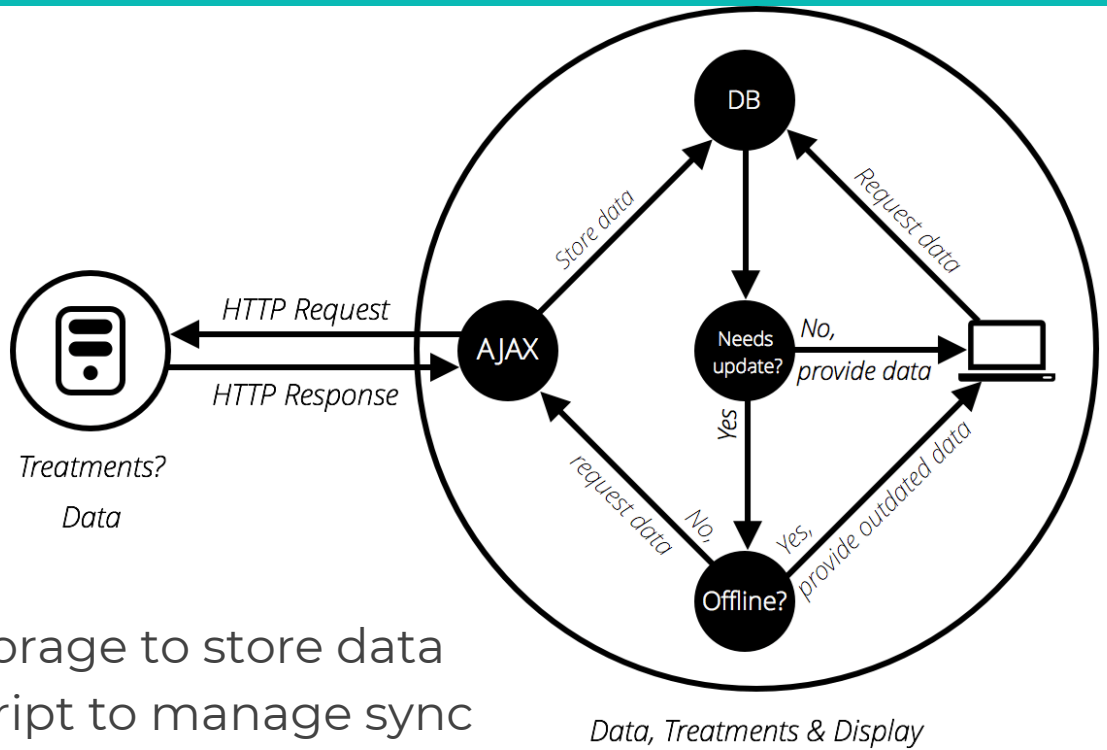
Visual difference between synchronous and asynchronous processing. Image from Eloquent JavaScript

AJAX Flow - Background Updates



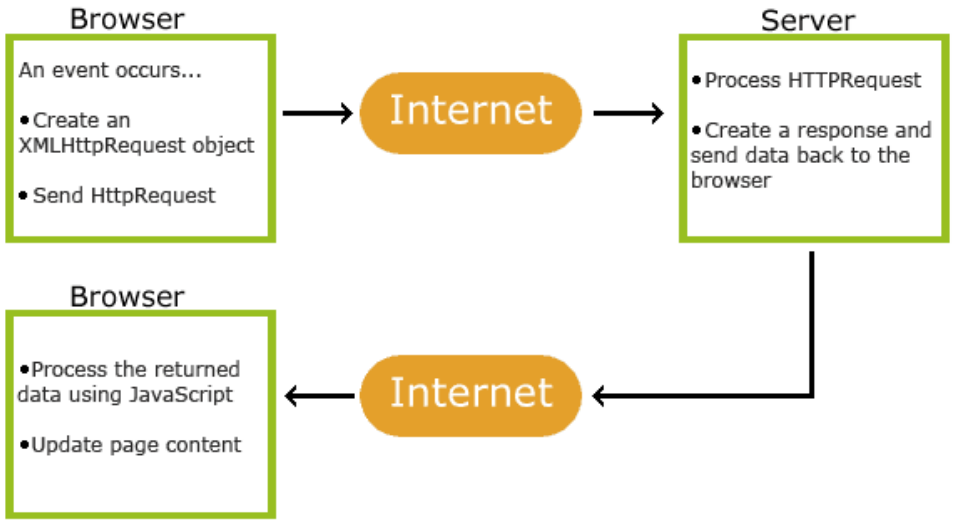
- Uses JavaScript to initiate HTTP requests and process HTTP responses

AJAX Flow - Local Sync



- Uses localStorage to store data
- Uses JavaScript to manage sync

How AJAX Works - The Old Way



1. An event occurs in a web page (the page is loaded, a button is clicked)
2. An XMLHttpRequest object is created by JavaScript
3. The XMLHttpRequest object sends a request to a web server
4. The server processes the request
5. The server sends a response back to the web page
6. The response is read by JavaScript
7. Proper action (like page update) is performed by JavaScript

Example from https://www.w3schools.com/xml/ajax_intro.asp

XMLHttpRequest Example on JavaScript.info

https://javascript.info/xmlhttprequest

The Fetch API

Fetch - A Modern Asynchronous API

- The XMLHttpRequest object and workflow is old and wonky
- When the introduction of HTML5 APIs there was an opportunity to update this workflow
- Uses modern asynchronous features like Promises and async/await
  - Less callback hell
- Well supported by all browsers
  - Except Internet Exploder

Fetch API

Allows us to fetch resources without reloading the page.

`fetch(url)` - GETs a resource

The then() part is the promise - event handler function with the onload event.

```
//a very basic Fetch example
fetch("http://example.com/api")
          .then(res => res.json())
          .then(json => console.log(json));
```

Explaining a Fetch API example

```
// make a simple GET request
fetch("http://example.com/orders")

     //Once the fetch completes, do this
          .then(rJSON => rJSON.json())

     //Once the JSON arrives, do this
          .then(res => res.map(order =>
order.id))

     //Now process the array of orderIds
          .then(orderIds =>
console.log(orderIds));
```

1. The `fetch()` function returns a promise with the `Response` of the operation.
2. The `json()` method takes the `Response` and returns a promise that will trigger once the JSON response has arrived.
3. The JSON object contains an array of orders. Each order has an `id` attribute. Use the map function to retrieve the id from each order.
4. Print each orderId to the console

```
["92998-3874", "90566-7771", "59590-4157", "53919-4257", "33263", "23505-1337", "58804-1099", "45169", "76495-3109", "31428-2261"]
```

Fetch Demos on JavaScript.info

https://javascript.info/fetch

Break (10 minutes)

Asynchronous Programming

Understanding Promises

Promises can be a bit confusing, but they are the underpinning of many modern JavaScript APIs (like Fetch)

It represents a placeholder for an object that will eventually have a value.

It provides a generic interface, then(), for executing a callback function, passed as an argument to then(), when the object has value.

This means you can do method chaining and write cleaner code.

This, this and this are good resources for learning more about promises.

Understanding Promises

You can make an expression wait or pause for a Promise to be fulfilled using the `await` expression.

You will see examples online that use this notation.

Can only be used inside an `async` function.

```
async function f1(){

        var x = await waitNseconds(10);

}
```

Asynchronous Programming on javascript.info

https://javascript.info/callbacks

https://javascript.info/async-await

A note about security

Fetch follows the same-origin policy

This means they can only make requests to URLs with the same protocol, port, and host

Unless you specify a Cross-Origin Resource Sharing (CORS) header to grant permission to access resources at other origins

So if you load a page from the following URL you will only be able to load some resources with Fetch.

A note about security

URL	Outcome	Reason
`http://store.company.com/dir2/other.html`	Same origin	Only the path differs
`http://store.company.com/dir/inner/another.html`	Same origin	Only the path differs
`https://store.company.com/page.html`	Failure	Different protocol
`http://store.company.com:81/dir/page.html`	Failure	Different port (`http://` is port 80 by default)
`http://news.company.com/dir/page.html`	Failure	Different host

CORS Fetching on JavaScript.info

https://javascript.info/fetch-crossorigin

Activity 7

Due Monday by 11:59

The purpose of this activity is to allow you to play around with promises and the Fetch API in JavaScript. Your task is to create a simple GET request using the Random User API and then display the results.

https://glitch.com/~infsci2560-2023-activity7

AJAX Demo Glitch Project

https://glitch.com/~infsci2560-2023-demo-ajax