# Ajax And Async

# Recap

Two weeks ago we covered

- Object Creation with Object Literals
- Object Creation with Function Literals
- Inheritance
- Using Prototypical Inheritance to share code

Any questions before moving on?

# Today’s Outline

- What is AJAX?
- Async vs Sync
- Drawbacks

# AJAX

Formal definition:

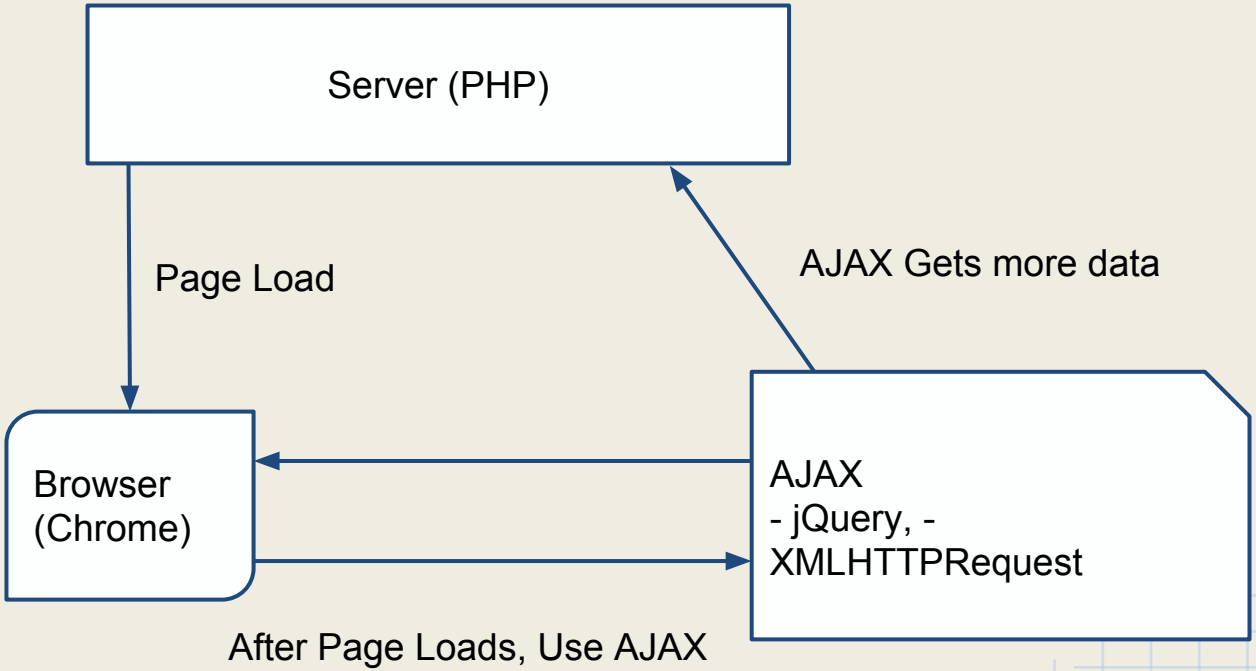
Asynchronous Javascript and XML

- Loading information from a server to the user's browser or sending info to a server without forcing a full page refresh.

# AJAX - Why

- Much Better user experience, page response times directly correlate to user retention.
- Code cleanliness; ajax allows developers to create logical code separations and reusable components.
- Create websites that behave more like portable, OS native apps
- Peer Pressure.  SPA. Everyone’s doing it.

# AJAX


Server (PHP)
Page Load
AJAX Gets more data
Browser
(Chrome)
AJAX
- jQuery, -
XMLHTTPRequest
After Page Loads, Use AJAX

# AJAX - Live Examples

- Google - search
- Twitter - everything, fetches and injects HTML
- AirBnB - tracking user events
- Gmail - All the things
- Medium - Sidebar (tags and top stories)
- Campus Labs - Beacon

# AJAX - When

- Lots of data to show on one page
- Reusable components across pages (event feed etc…)
- Tracking;  Google uses ajax to track user clicks and page events when you use Google Analytics.
- Multiple actions on one page and do want the user to refresh.
- Interact with third-party data, Facebook & Twitter

# AJAX - What

You can send or receive almost any type of information to a server using AJAX including:

- HTML
- JSON
- XML
- Files (new, only IE 10 and above)

# AJAX - How

The gross way

- XMLHTTPRequest

This is as low level as it gets.  We'll go over one example and then move on to a micro framework to get the job done.

Only recently has this been standardized in browsers (IE!!).  It's verbosity and clumsy (but getting better, used to be way worse) syntax is what leads to other wrapping frameworks to be built.

# AJAX - How

```
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "/people", true);

xmlhttp.onreadystatechange = function() {
	if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
		console.log(JSON.parse(xmlhttp.responseText));
	}
};

xmlhttp.send();
```

# AJAX - Demo

Demo

# AJAX - Errors

Occasionally, servers can throw errors due to high-traffic, programming bugs or network latency.  We need to handle these cases when making async requests.

With XMLHTTPRequest, we can check the status code of the request object to determine how the server responded.

200 generally means the server responded “OK”.

# AJAX - Errors

Demo Handling Errors

# AJAX - A Better Way

Always use a framework for AJAX (unless business or framework reasons dictate otherwise).

We can use jQuery to simplify the syntax.

Who is familiar with jQuery?

Ajax Demo With jQuery.

# AJAX - Weird Syntax

- Why do we need onreadystatechange for xmlhttprequest?
- What is with the second parameter to the jquery “get” ajax call.

```
$.get('/peopleHTML', function (data) {
    document.getElementById('baseballPlayers').innerHTML = data;
});
```

# Async

Calling out to a server can be a potentially time-consuming process.  Your browser (because of javascript) will process the network call and allow other code to continue executing.

When the call is complete, we use a callback function to respond to whatever change was made.

# Async

setTimeout is a built in javascript function that demonstrates the usage of callbacks.

```
setTimeout(function () {
    console.log('Running this after half a second');
}, 500);

setTimeout(function () {
    console.log('Running this after a quarter of a second');
}, 250);
```

# Async - End Result

- Because functions in javascript might run asynchronously, you cannot guarantee order in code.
- The only way to combat this is to know the API of the framework you're using, or be familiar with the code your team is writing.
- Make sure any dependant code is running in a callback function.

# Async - Demo

Long Running Process Demo

# AJAX - Drawbacks

- Writing complex, ajax-heavy javascript applications can lead to unmaintainable code.
- Too many nested callback functions can lead to what's commonly referred to as callback hell.
- Async code is hard to follow and predict unless care is taken during it's construction.

# Questions

# Workshop

Building google