Welcome to class!

AGENDA: MODEL DATA

- Feedback overview
- Friday assignment review
- Web typography
- JSON
- AJAX

We are talking about how to handle data in your app. This is the “Model” layer. We’ll write code to give us a handle on how to manipulate and work with the data.

CLASS FEEDBACK REVIEW: GOING WELL

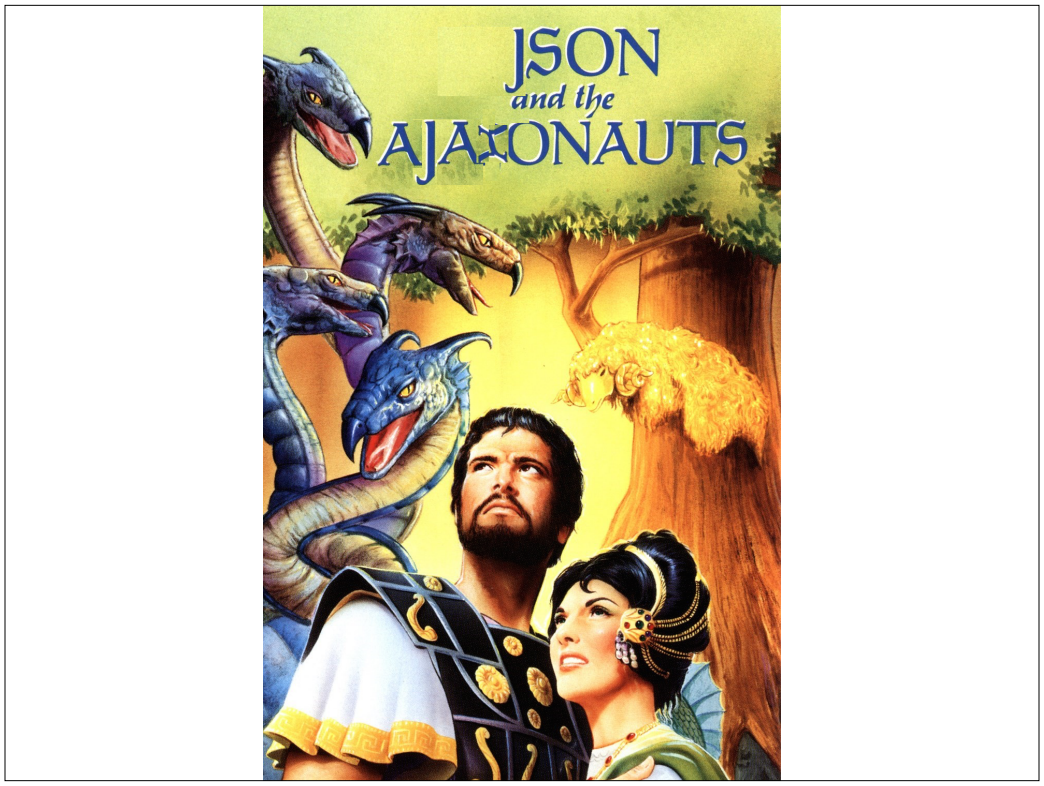
- “I found the code reviews most useful. The review allowed us to go through the code line by line and understand how it works.”
- “Code samples are helpful for guidance on how to approach a problem and where to start”
- “The assignments given each day, though intense, were able to effectively reinforce the concepts in class and readings”
- “TAs and instructor are really awesome and willing to help”
- “Pair programming. That was the best thing ever for myself and my partner.”
- “I've met so many smart and interesting, hardworking people in this class and am excited to keep going, and keep helping others and learning from them”

CLASS FEEDBACK REVIEW: NEEDS IMPROVEMENT

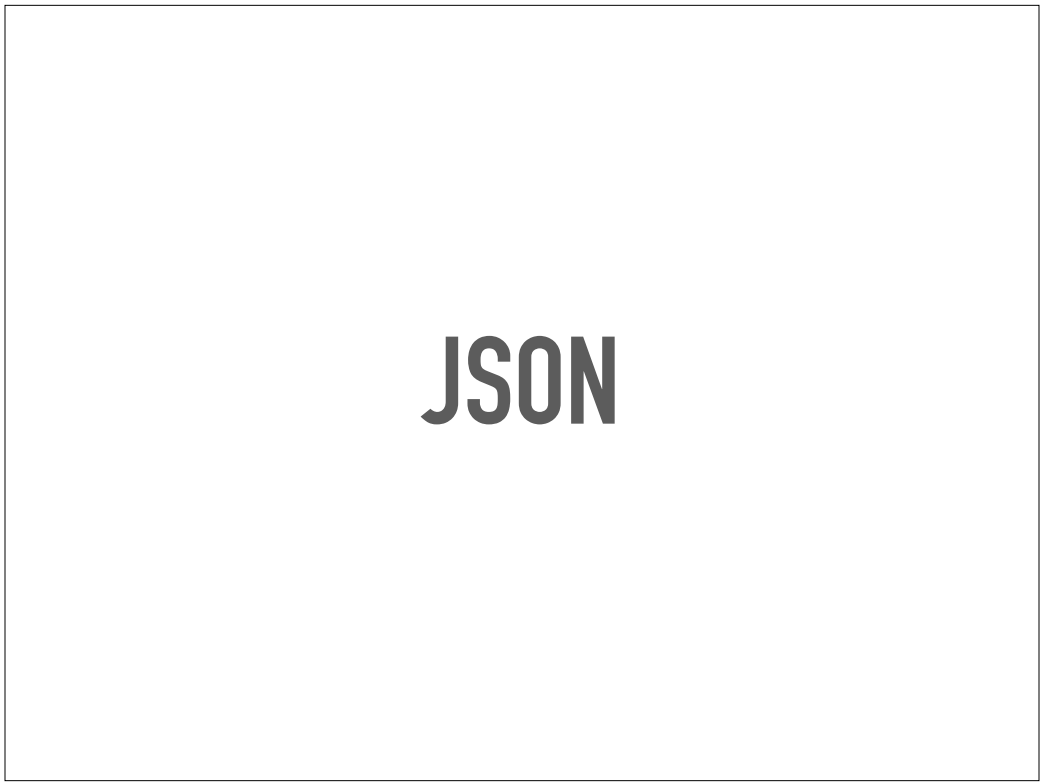
- “I would have liked to have more pre-work sent out sooner”
- “The first day was just really tough”
- “I cried and thought about quitting”
- “Learning things like mobile first web design, smacss and javascript libraries will not really help the people that want to go into Python or iOS 401s”
- “When we come up the 2nd floor is full, so it's hard to sit together”
- “A lack of adequate help/assistance/guidance”

CODE REVIEW

Let’s look over how things went with your Friday pair assignment.



Let’s dive in to today’s topics. Are you ready, brave Ajaxonaut?

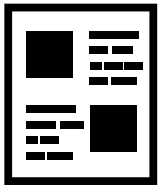


Let’s start with JSON

JSON

- Looks like this:

```
{
  "author": "Virginia Sawayn",
  "title": "Navigating Solid State Multi-byte Monitors"
}
```



Let’s start with an example. Doesn’t this kind of look like regular old JavaScript objects?

JSON: WHAT

- JavaScript Object Notation
- JS objects are lists of key-value pairs

Take a JS object, turn it into a string. Then, when we need it, be able to turn it back into an object. This has won out over XML as a standard for serialization and data interchange

JSON: WHY

- JSON is a standard way to **serialize** your objects
  - "Dehydration for data structures"
  - Reconstitute it later when needed
- Kicked XML's ass
  - More human readable
  - More "object-oriented"

Take a JS object, turn it into a string. Then, when we need it, be able to turn it back into an object. This has won out over XML as a standard for serialization and data interchange

JSON VS. XML NOTATION

- In most situations, JSON is more compact than XML
- JSON

```
1 {
2   "company": "Volkswagen",
3   "name": "Vento",
4   "price": 800000
5 }
```

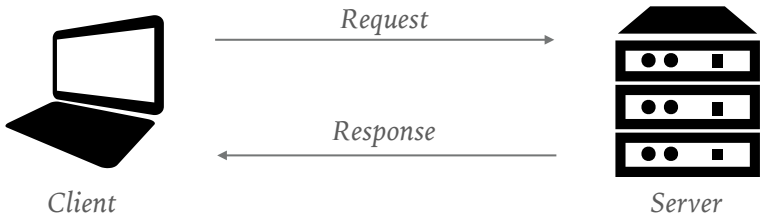
- XML

```
1 <car>
2    <company>Volkswagen</company>
3    <name>Vento</name>
4    <price>800000</price>
5 </car>
```

XML is all: “Don’t you wanna write that attribute name TWICE?”

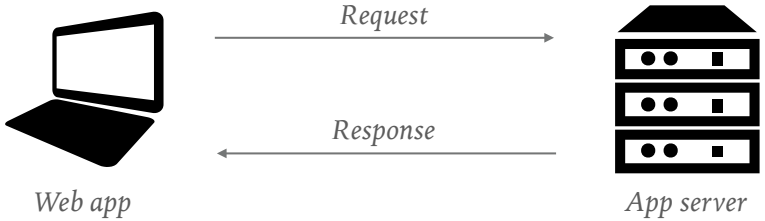
JSON: WHY

- We want to do all this because…?
  - Data lives over there
  - We want it over here
  - **Serialization** allows interchange



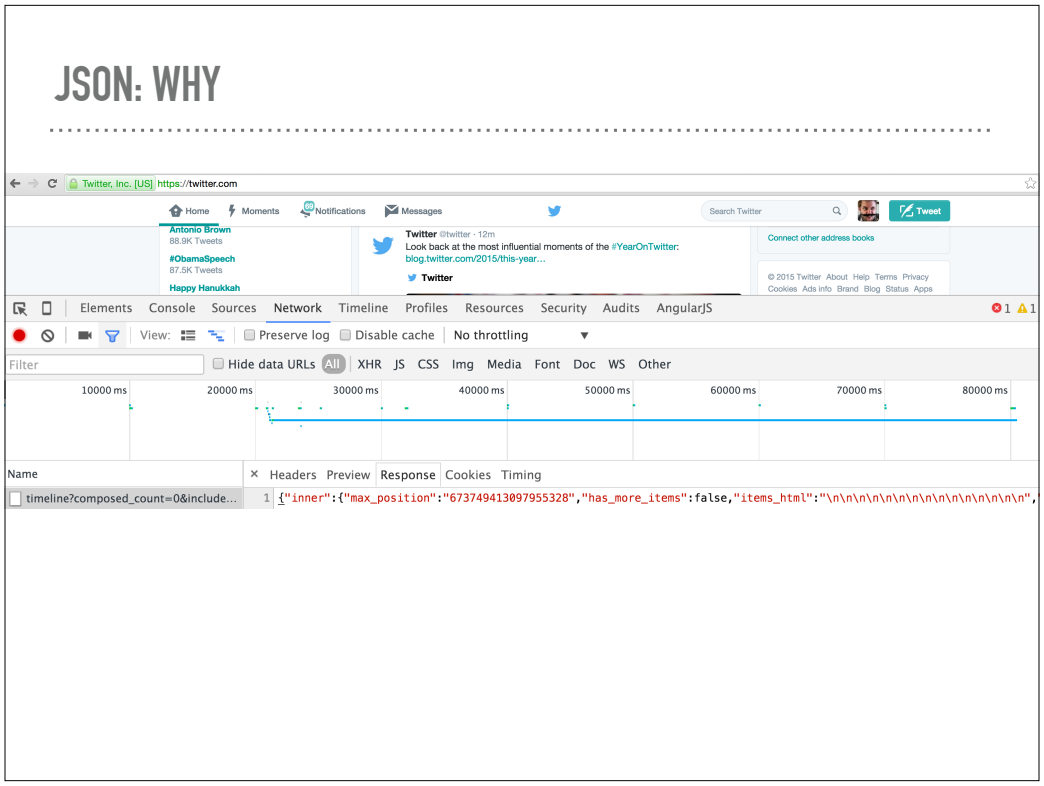
Can you think of some examples of when a client would want to request DATA from a server?

JSON: WHY



- When a web app needs an update,
- Web app requests new info from the server
- Gets a JSON response with data it can process
- Goes great with AJAX

For example, twitter.com will periodically ask the server for new info.



Open up twitter.com, reveal the Network tab of the Inspector, and you'll see some requests being made. Click the request in the "Name" column, and you'll see the JSON response that came from the server.

JSON: WHY

- JSON is a standard way to **serialize** your objects
  - "Dehydration for data structures"
  - Reconstitute it later when needed
- Data can be persisted via the localStorage API
  - Keep user preferences
    - What tab was open?
    - Which articles have I expanded?
  - Cache data
    - Prevent unnecessary requests to the server

The other place we can store serialized objects is in localStorage, right in the browser.
Think of localStorage as a gym locker for your JSON.

JSON: BASIC RULES

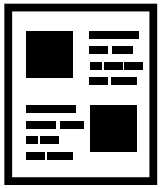
- Always use double quotes
- Keys are quoted strings
- Values can be:
  - String
  - Number
  - Object
  - Array
  - true/false
  - null

Take a JS object, turn it into a string. Then, when we need it, be able to turn it back into an object.

JSON

- Object Literal

```
1 {
2   author: 'Virginia Sawayn',
3   title: 'Navigating Solid State Multi-byte Monitors'
4 }
```



- JSON

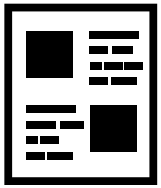
```
1 {
2   "author": "Virginia Sawayn",
3   "title": "Navigating Solid State Multi-byte Monitors"
4 }
```

Let's see that example again.

JSON

- Object Literal

```
{
  author: 'Virginia Sawayn',
  title: 'Navigating Solid State Multi-byte Monitors'
}
```



- JSON

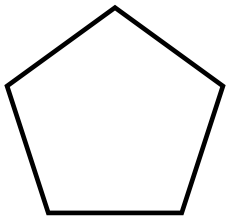
```
{
  "author": "Virginia Sawayn",
  "title": "Navigating Solid State Multi-byte Monitors"
}
```

Let's look at how JSON objects can be built up. Remember this? Strings everywhere!

JSON: HOW

- Object Literal

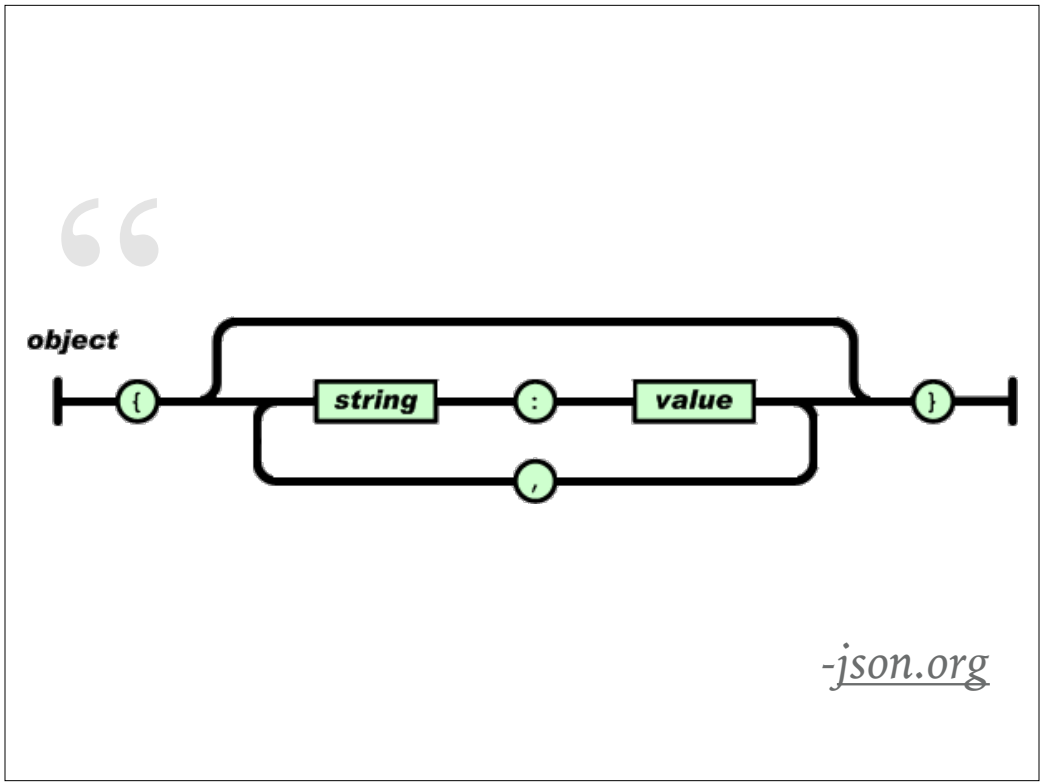
```
{
  name: 'Pentagon',
  sides: 5,
  sideLength: 12,
  regular: true
}
```



- JSON

```
{
    "name": "Pentagon",
    "sides": 5,
    "sideLength": 12,
    "regular": true
}
```

Numbers and boolean values can be represented as well.

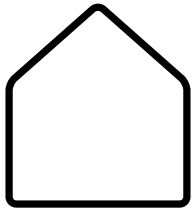


JSON is primarily a list of key-value pairs, just like JS object literals.

JSON

- Object Literal

```
{
  name: 'Pentagon',
  sides: 5,
  sidesLength: [10, 8, 8, 10, 10],
  regular: false
}
```



- JSON

```
{
    "name": "Pentagon",
    "sides": 5,
    "sidesLength": [
        10,
        8,
        8,
        10,
        10
    ],
    "regular": false
}
```

What if our pentagon doesn't have the same length for every side? A key can also have an array as it's value

JSON

- Object Literal

```
{
  name: 'American Pharoah',
  jockey: {
    name: ‘Victor Espinoza',
    yob: 1972
  },
  breeder: {
    name: 'Zayat Stables',
    location: {
      city: 'Hackensack',
      state: 'New Jersey'
    }
  }
}
```

- JSON

```
{
    "name": "American Pharoah",
    "jockey": {
        "name": "Victor Espinoza",
        "yob": 1972
    },
    "breeder": {
        "name": "Zayat Stables",
        "location": {
            "city": "Hackensack",
            "state": "New Jersey"
        }
    }
}
```

We can even embed objects [within objects!] within JSON

JSON

```
{
  "pilots": [
    {
      "name": "Amelia Earhart",
      "yob": 1897,
      "yod": 1936
    },
    {
      "name": "Anne Morrow Lindbergh",
      "yob": 1906,
      "yod": 2001
    },
    {
      "name": "Chuck Yeager",
      "yob": 1923,
      "yod": null
    }
  ]
}
```

Even arrays of objects!

JSON DEMO

Show stringify'ing a few objects in the console. Re-hydrate with JSON.parse().
Give me a problem domain! We'll model it as a JSON object, with embedded arrays and objects.

JSON: YOUR TURN

- You can mix and match data types as needed
- What can you come up with?

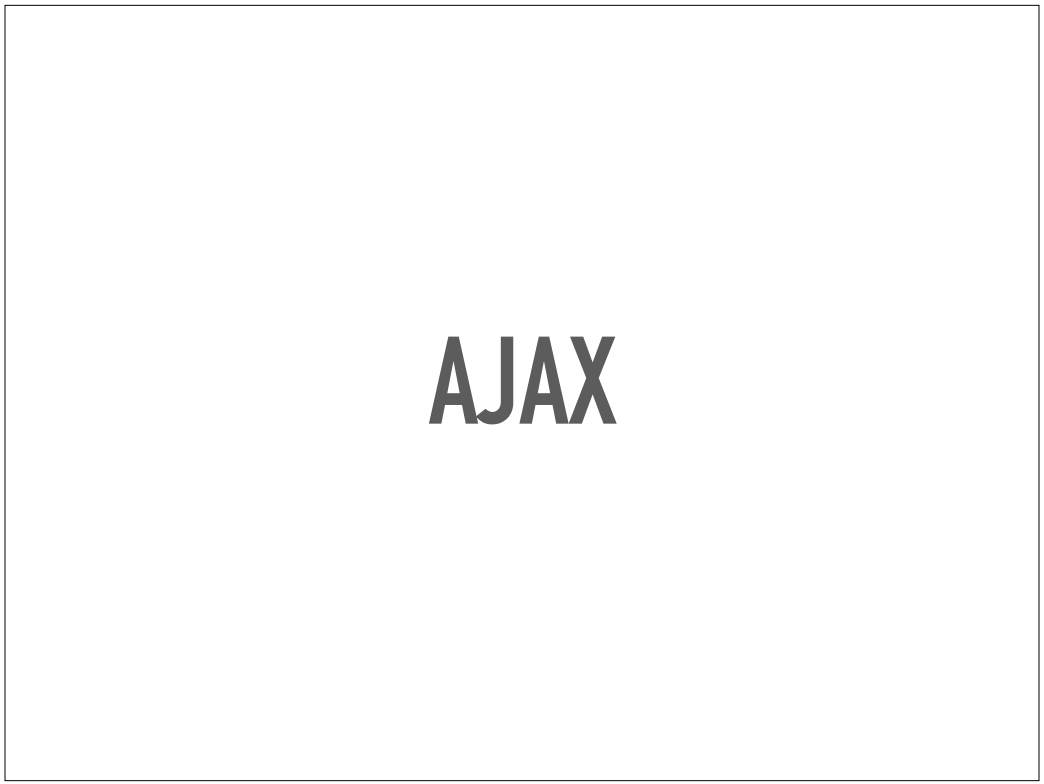
As a group of 3, pick a problem domain, and practice building a more complex JSON representation of some objects from that space.

JSON: SUMMARY

- Lightweight data interchange
- Machines can talk to machines
- Our code can talk to output of other code

JSON: RESOURCES

- http://json.org/
- https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/JSON
- http://www.drowningintechnicaldebt.com/RoyAshbrook/archive/2007/07/09/top-10-quot-why-xml-sucks-quot-articles.aspx



Ready for Web 2.0? Please meet AJAX.

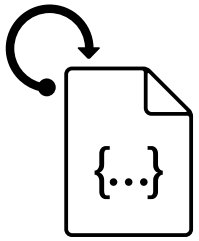
AJAX: WHAT

- "Asynchronous JavaScript And XML"
- But's it's mostly JSON these days, as seen at twitter.com
- …or HTML that's retrieved on-demand
- No one wants to figure out how to pronounce…
  - **AJAJ / AJAH**

No one bothered to update the acronym.

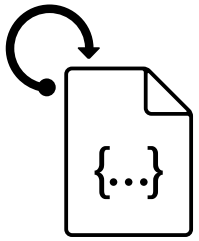
AJAX: WHEN

- 1998: MS Outlook Web Access uses async with ActiveX
- Dec 6, 2000: Firefox releases JS compatible version
- 2004: Gmail, Support in Safari 1.2
- 2005: Google Maps, Google Suggest, “AJAX”
- 2006: Support in Internet Explorer 7
- Widespread usage followed
- “Web 2.0” was born!

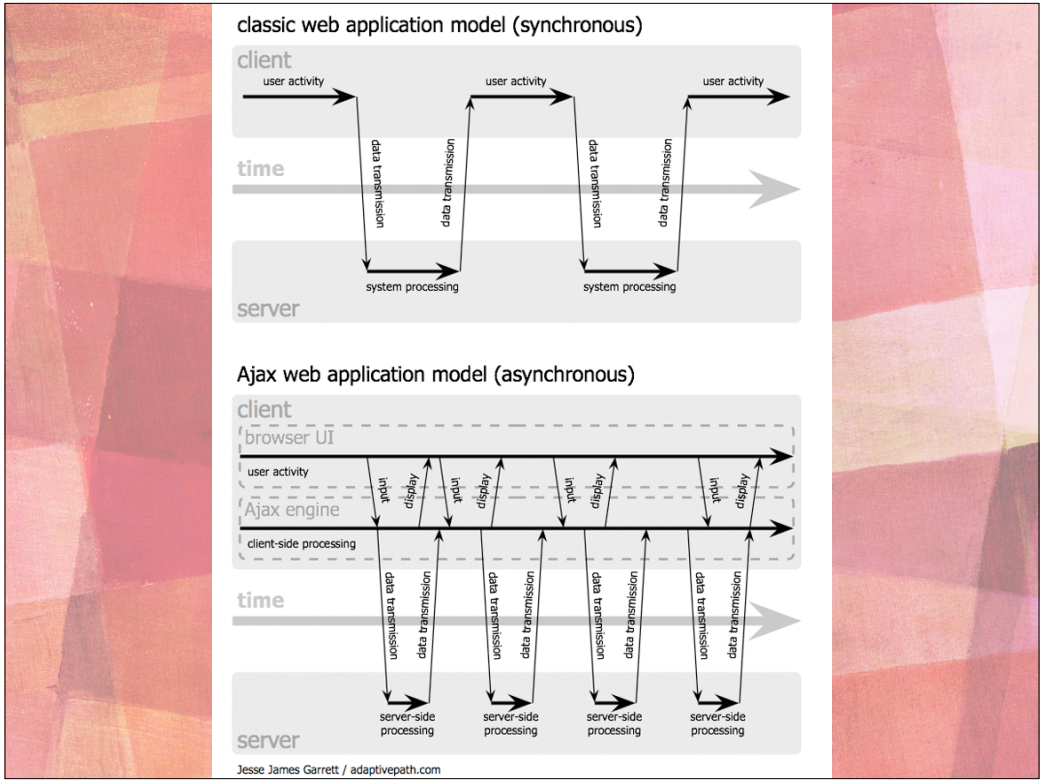


AJAX: WHAT

- Did you say **Asynchronous**?
- Async, like, with callbacks?
- AJAX is about asynchronous **requests**
  - Don't reload the whole page & DOM
  - Make a request programmatically
    - No longer limited to script load order
    - Could be during event handling
    - Could be on a timer (see: twitter.com)
    - Undetermined duration: Callback runs "when done"



Time to think like the JS interpreter! We don't know how long any given AJAX request will take, so we need to register a callback to handle the result of the request.



From the original blog post that coined the term “AJAX”… Smaller, simpler requests let us update the display a lot faster.
Graphic via: http://adaptivepath.org/ideas/ajax-new-approach-web-applications/

AJAX: POWER UP

- Enables an entirely new class of application
  - Watch these tiles load: https://www.google.com/maps
  - Every browser based multi-player game
  - Faster interactions mean more usable apps
  - Single-Page Apps (SPAs): http://fontflame.com/
- Fetch just what you need
- Update only what changes in the page
- Stay in touch with the server

AJAX: WITH JQUERY

1. Send a request
2. Register an async callback function to handle the response
3. Profit!

AJAX

Global Ajax Event Handlers
.ajaxComplete()
.ajaxError()
.ajaxSend()
.ajaxStart()
.ajaxStop()
.ajaxSuccess()

Helper Functions
jQuery.param()
.serialize()
.serializeArray()

Low-Level Interface
jQuery.ajax()
jQuery.ajaxSetup()

Shorthand Methods
jQuery.get()
jQuery.getJSON()
jQuery.getScript()
.load()
jQuery.post()

AJAX: SEND A REQUEST

1. Send a request
   - Specify the request “method”, URL, settings
   - Request method options:
     - GET
     - POST
     - HEAD
     - etc…
   - jQuery has helper methods to simplify:

     `$.ajax, $.load, $.get, $.getJSON, $.post`

AJAX: CALLBACKS

2. Register an async callback function to handle the response
   - Write code to handle each kind of response:
     - `success` option or chain `.done()`
     - `error` option, or chain `.fail()`
     - complete option, or chain `.always()`
   - Anything that depends on the response, must be in a callback!
   - Use named functions to avoid callback-hell
     - Seriously: http://callbackhell.com/

We register a callback to deal with the data we get from the server.
We can use the data in a success callback, handle errors in the error callback, or run some code no matter what happens in the `complete` callback. There are method chaining versions of those as well.

AJAX DEMO

Let’s look over the example code from the book.
Let’s also create an AJAX request for our blogArticles.json file.

AJAX: SUMMARY

- Critically useful technique for making modern web apps
- Client can control communication with the server

AJAX: FURTHER RESOURCES

- https://en.wikipedia.org/wiki/XMLHttpRequest
- http://adaptivepath.org/ideas/ajax-new-approach-web-applications/
- https://teamtreehouse.com/library/ajax-basics

RECAP

RECAP

- JSON & AJAX: Taste great together
- Organize your data management
  - Transit
  - Storage & caching

These technologies put you in control of your data.
We'll be looking further at how to manage data within your app.

REFERENCES AND SOURCES

- Vector icons via The Noun Project
- http://i.jeded.com/i/jason-and-the-argonauts-1963.29114.jpg
- https://commons.wikimedia.org/wiki/File:Amelia_Earhart_awaits_transatlantic_flight_1928.jpg
- https://en.wikipedia.org/wiki/Ajax_(programming)