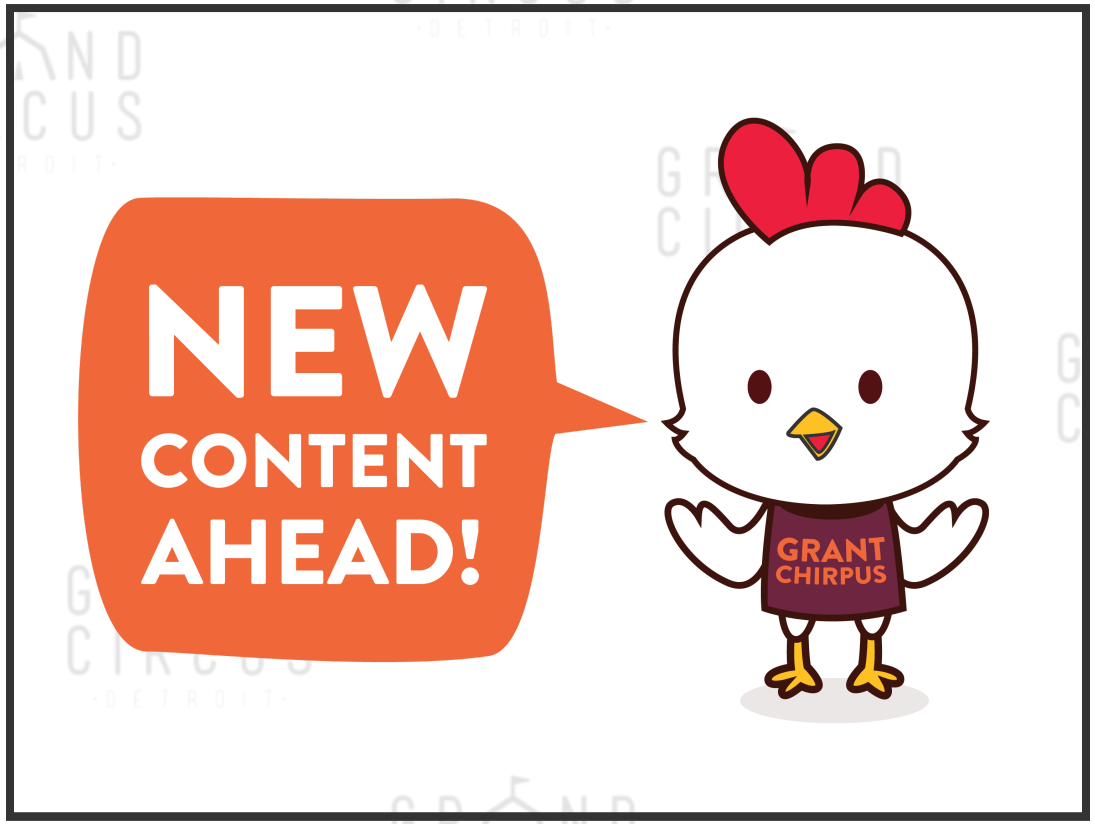

NEW
CONTENT
AHEAD!
GRANT
CHIRPUS

AJAX & JSON

GOALS FOR THIS UNIT

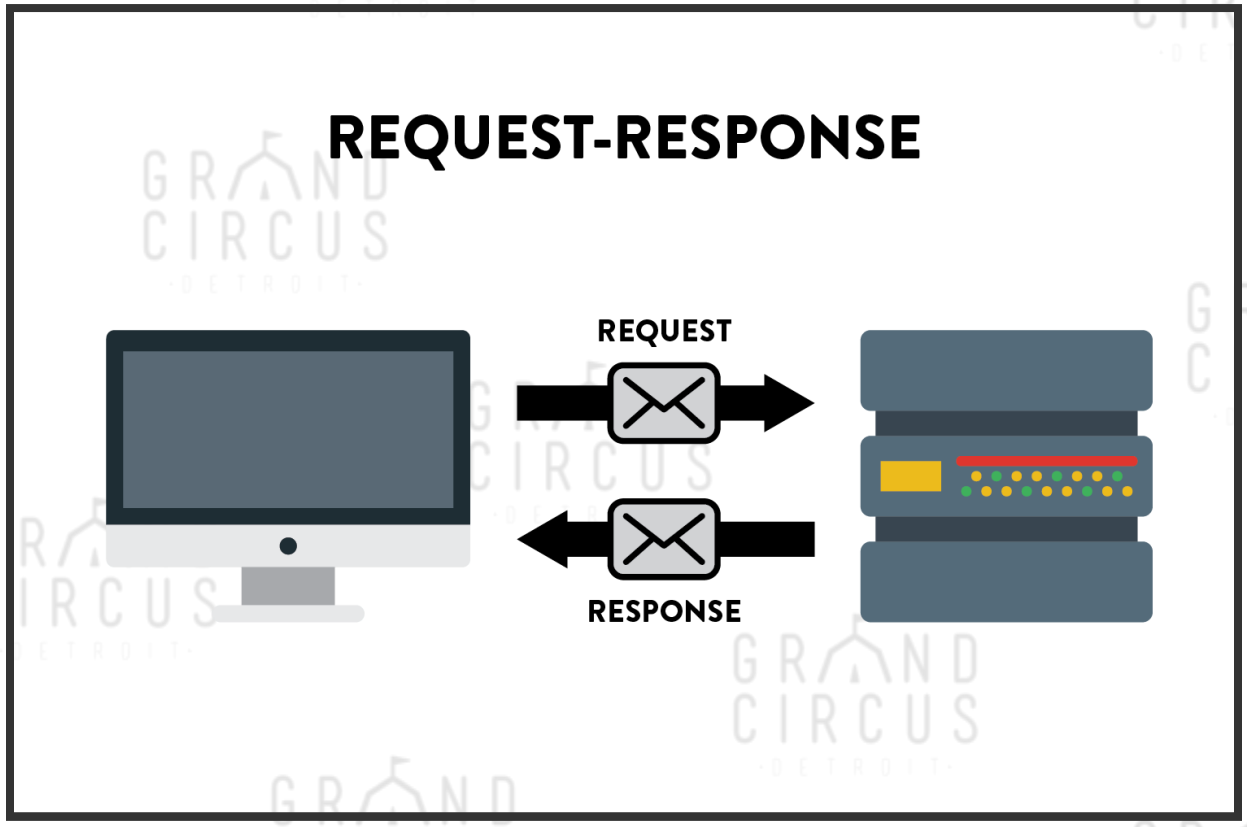
1. Review
2. Requests & Responses
3. Submitting Forms
4. Error Handling

REVIEW

HOW THE INTERNET WORKS

Time for a bit of crucial theory.

CLIENT / SERVER



The browser (client) sends a *request* to the server.
Then the server sends back a *response*.

THE REQUEST

The request kicks off the communication. It is a message sent from the client to the server. It includes several pieces of information:

1. The envelope: Where is the request going? What server?
2. The payload: What do we want the server to do?

THE RESPONSE

Unless the internet breaks, a response always comes back from the server. This message includes:

1. Status: Was the server able to do what the client wanted?
2. Data: Any information that was requested.

THE BREAKDOWN

Let's look at each part in detail.

REQUEST - ENVELOPE

Where is the request going?

It's determined by the URL of the request. This is what is in the browser bar for a webpage and also the *href* of a link and *src* of an image or script tag.

PARTS OF A URL

http://www.example.com/index.html

PROTOCOL

This first part tells us how we're going to talk to the server. It's sort of like deciding whether to start a conversation in English or Japanese or Sign Language.

HTTP is the most common protocol on the web and the only one we'll look at in this class.

PARTS OF A URL

http:// *www.example.com* /index.html

HOST

This tells us what server to send the message to. It can be a domain name (like this one here) which is easy for humans to read, or it can be an IP address, which is just a buch of numbers.

PARTS OF A URL

http:// *localhost* :8080/index.html

HOST

"localhost" is a special domain name that always points back to your current computer. Usually the client and server are on different computers, but in this case the they are both on the same computer.

PARTS OF A URL

http:// localhost: *8080* /index.html

PORT

The host told us which computer the server was on. The port tells us which program on that computer is the server. A single computer is able to run many servers at the same time. Each one *listens* for requests on a different port.

Port is an optional part of the URL. If omitted, there is a default port for every protocol. 80 is the default for HTTP.

REQUEST - ENVELOPE

www.example.com
Apt. Port 80
93.184.216.34

REQUEST - PAYLOAD

What do we want the server to do?

1. Method
2. URI / Path
3. Query (or Search) Parameters
4. Headers
5. Body / Content (for some requests)

REQUEST - METHOD

Tells the server what sort of thing you're trying to do. There are a lot of options, but these two are the most common.

- *GET* - Get a file from the server. This is most requests.
- *POST* - Often used for submitting forms.

REQUEST - URI / PATH

http:// www.example.com*/index.html*

Portion of URL after the host and port. It usually selects a file on the server.

REQUEST - QUERY (OR SEARCH) PARAMETERS

www.example.com/product-search.html*?search=fluffy&category=shoe*

Optional. Extra *key-value pairs* of information. Starts after "?" and each pair is separated by an "&".

REQUEST - HEADERS

Headers are key-value pairs that provide *metadata* about the request. For example:

- Accept - What kind of data do I want the server to send back
- Content-Type - What kind of data am I sending to the server
- Content-Length - How much data am I sending to the server (bytes)
- User-Agent - What browser am I using

REQUEST - BODY / CONTENT

POST requests can also send a chunk of data.

GET requests do not have a Body.

RESPONSE

RESPONSE

1. Status Code
2. Headers
3. Body / Content

RESPONSE - STATUS CODE

Five categories of 3-digit codes.

- *1xx* Informational.
- *2xx* Success.
- *3xx* Redirection.
- *4xx* Client Error.
- *5xx* Server Error.

RESPONSE - STATUS CODE

Here are some of the most common.

- *200* OK
- *302* Redirect to a different URL.
- *404* File not Found
- *403* You don't have permission. Maybe you need to log in first.
- *500* There is an error on the server.

See all the codes here.

RESPONSE - HEADERS

Metadata about the response. For example:

- Content-Type - What kind of data is being sent back (HTML, CSS, image, etc.)
- Content-Length - How much data (bytes)

RESPONSE - BODY / CONTENT

Most responses do send back a chunk of data. This would be the HTML file, CSS file, image, etc. that the client asked for.

CHROME NETWORK TAB

AJAX

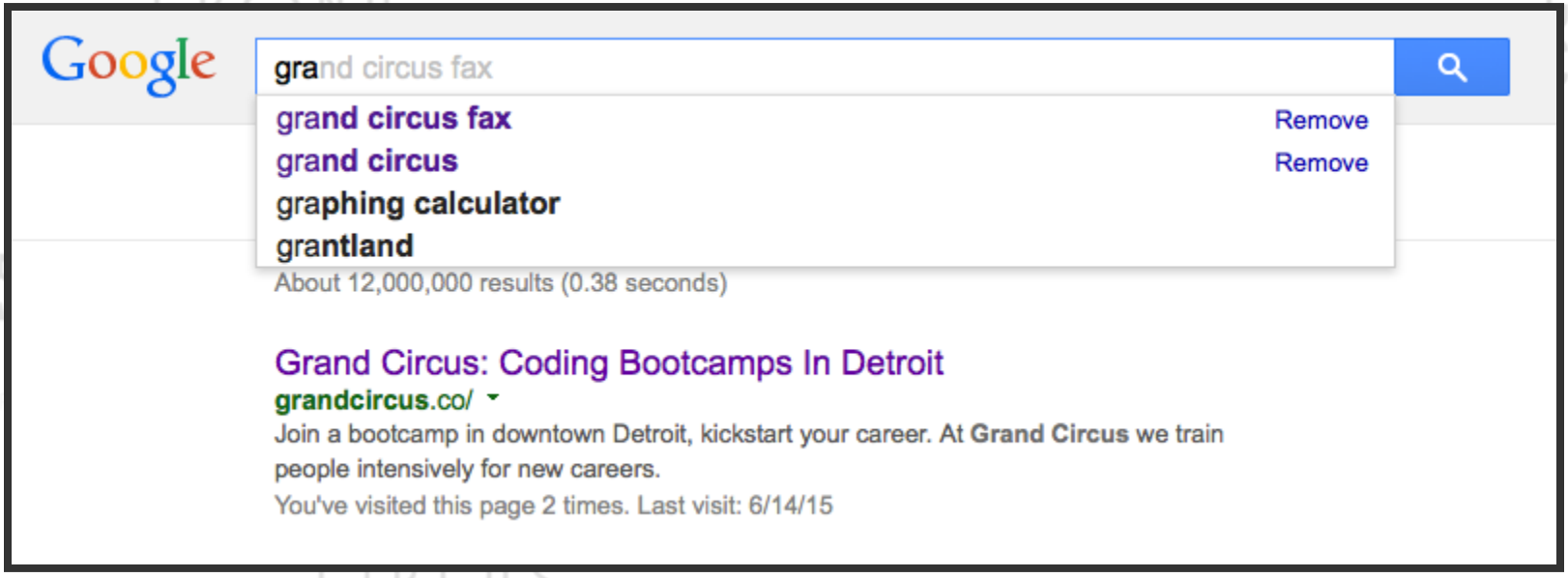
HISTORY LESSON

AJAX was originally an acronym for Asynchronous JavaScript And XML. These days, people still use it, but the term Ajax actually refers to a group of web technologies used for asynchronous programming.

We're going to use "AJAX" moving forward.

ASYNCHRONOUS PROCESSING

When we use AJAX, the browser can request data without needing to wait to load the rest of the page. This works for loading an entire page as well as just parts of a page.


Google
grand circus fax
grand circus fax
Remove
grand circus
Remove
graphing calculator
grantland
About 12,000,000 results (0.38 seconds)
Grand Circus: Coding Bootcamps In Detroit
grandcircus.co/
Join a bootcamp in downtown Detroit, kickstart your career. At Grand Circus we train people intensively for new careers.
You've visited this page 2 times. Last visit: 6/14/15

JQUERY & AJAX

METHODS

jQuery provides four methods to handle our AJAX requests.

- `$.get()`
- `$.post()`
- `$.getJSON()`
- `$.getScript()`

USING $.GET()

USING $.GET()

1. Define the request.
2. Define a function to handle the response.

DEMO

ASYNCHRONOUS

```
var message = "before";

$('button').on('click', function(){
    message = "success";
});

console.log(message); // > "before"
```

ASYNCHRONOUS

```
var message = "before";

$.get('https://www.reddit.com/r/aww/.json', function(responseBody){
    message = "success";
});

console.log(message); // > "before"
```

ASYNCHRONOUS

```
var message;

$.get('https://www.reddit.com/r/aww/.json', function(responseBody){
    message = responseBody;
});

console.log(message); // > undefined
console.log(responseBody); // > undefined
```

ASYNCHRONOUS

```
$.get('https://www.reddit.com/r/aww/.json', function(responseBody){
    // Put all your code to handle the response inside this function.
    console.log(responseBody);
});
```

SENDING & SUBMITTING FORMS

COLLECTING DATA

In addition to the four methods already mentioned, jQuery provides us with the `.serialize()` method, which takes all the information contained in a form, puts it into a string we can send to the server (encoding characters when necessary).

THE BREAKDOWN

1. Select the form.

```
$('#register')
```

THE BREAKDOWN

2. Using the `on()` method, we'll create a block of code to be run when the form is submitted. The first argument is the trigger and the second is the function that will respond to that trigger.

```
$('#register').on('submit', function(e){
});
```

THE BREAKDOWN

3. We can prevent the form from submitting immediately (which would be its default behavior).

```
$('#register').on('submit', function(e){
  e.preventDefault();
});
```

THE BREAKDOWN

4. To prepare the form data to be sent to the server, we use the `.serialize()` method.

```
$('#register').on('submit', function(e){
  e.preventDefault();
  var details = $('#register').serialize();
});
```

THE BREAKDOWN

5. The `$.post()` method sends the data.

```
$('#register').on('submit', function(e){
  e.preventDefault();
  var details = $('#register').serialize();
  $.post('register.php', details, function(data) {

  });
});
```

THE BREAKDOWN

6. The function passed to the `$.post()` method indicates where the result should be displayed.

```
$('#register').on('submit', function(e){
  e.preventDefault();
  var details = $('#register').serialize();
  $.post('register.php', details, function(data) {
    $('#register').html(data);
  });
});
```

SUCCESS & FAILURE

FAILURE HAPPENS

Every so often, you will make a request of the server and it will fail. This is inevitable, so be ready! Plan ahead for moments like these so your page isn't completely broken with every little thing that doesn't go as expected.

HANDLING IT EITHER WAY

The following methods can be chained after any of jQuery's shorthand AJAX methods.

SUCCESS!

Code passed to the `done()` method will only run if the request is completed successfully.

FAILURE :(

Code passed to the `fail()` method will only run if the request is *not* completed successfully.

EVERY TIME

Code passed to the `always()` method will run regardless of the status of the request.

DEMO

```
$.get('https://www.reddit.com/r/aww/.json').done(function(responseBody) {
  console.log("DONE", responseBody.data.children[0].data.title);
}).fail(function() {
  console.log("FAIL");
}).always(function() {
  console.log("ALWAYS");
});
```

See it live

HOMEWORK

From JavaScript & jQuery:

- Chapter 7: 359-366

LAB 11

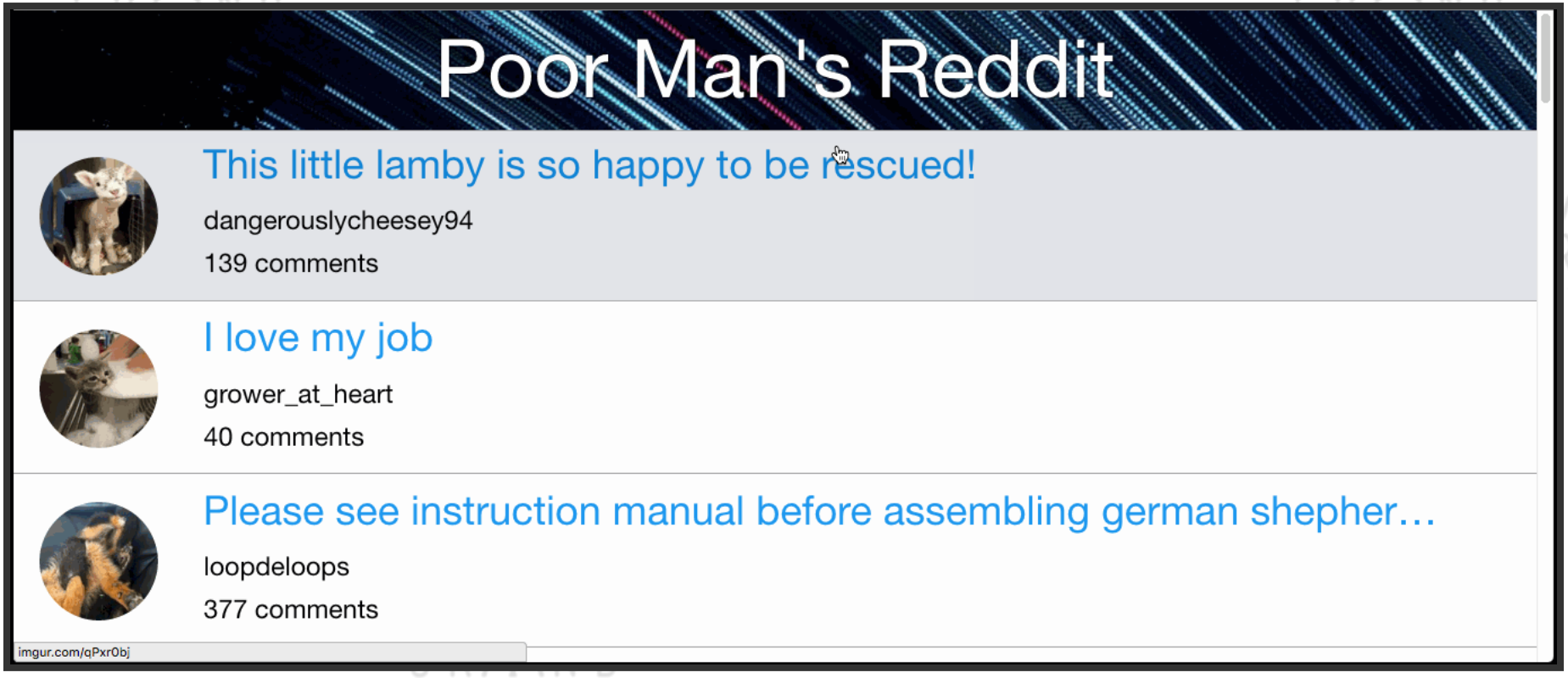
POOR MAN'S REDDIT

INSTRUCTIONS

Take the JSON payload available from /r/aww (check with the instructor before using anything else) and use the data to create a feed consisting of the first ten posts.

1. Break into small groups and delegate tasks.
2. Look at the raw JSON to see how objects are organized.
3. Create a webpage that pulls information from the payload.

PRE-GAME


Poor Man's Reddit
This little lamby is so happy to be rescued!
dangerouslycheesey94
139 comments
I love my job
grower_at_heart
40 comments
Please see instruction manual before assembling german shepher…
loopdeloops
377 comments
imgur.com/qPxr0bj