**Intro to APIs & AJAX**

**{Remember, Axios is a package that has tp be installed via NPM**

**Learning Objectives**

* Describe what an API is and why we might use one.
* Understand the format of API data.
* Explain the common role of JSON on the web.
* Use Axios to make GET requests for data.
* Use Axios' "promise" methods to handle AJAX responses.
* Render new HTML content using data loaded from an AJAX request.
* Perform POST, PUT and DELETE requests to an API to modify data.

**Framing (5 minutes / 0:05)**

**What is an API?**

Basically, an API is a service that provides raw data for public use.

API stands for "Application Program Interface" and technically applies to all of software design. The DOM, jQuery, and React are actually examples of APIs! Since the explosion of information technology, however, the term now commonly refers to web URLs that can be accessed for raw data.

Many APIs publish data for public use. As third-party software developers, we can access an organization's API and use their data within our own applications.

**Q: Why do we care?**

As we move into building single page applications, now is the perfect time to start understanding how to obtain data on the client side and then render it on the browser.

**API Exploration**

**Check out these stock quotes...**

* <http://dev.markitondemand.com/Api/Quote/json?symbol=AAPL>
* <http://dev.markitondemand.com/Api/Quote/json?symbol=GOOGL>

Does the JSON look unreadable in the browser? If you're using Chrome, install the [JSON View plugin](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en).

**Think, Pair, Share (5 minutes / 0:10)**

Form pairs and explore the API links in the below table. Record any observations that come to mind. In particular, think about what you see in the URL and the API response itself.

| **API** | **Sample URL** |
| --- | --- |
| [**This for That**](http://itsthisforthat.com/) | <http://itsthisforthat.com/api.php?json> |
| [**Giphy**](https://github.com/Giphy/GiphyAPI) | <http://api.giphy.com/v1/gifs/search?q=funny+cat&api_key=dc6zaTOxFJmzC> |
| [**OMDB API**](http://www.omdbapi.com/) | <http://www.omdbapi.com/?t=Game%20of%20Thrones&Season=1> |
| [**StarWars**](http://swapi.co/) | <http://swapi.co/api/people/3/?format=json> |
| [**Stocks**](http://dev.markitondemand.com/MODApis/) | <http://dev.markitondemand.com/Api/Quote/xml?symbol=AAPL> |

**Why Just Data?**

Sometimes that's all we need. All this information, from all these browsers and all these servers, has to travel through the network. That's almost certainly the slowest part of the request cycle. We want to minimize the bits. There are times when we just need the data. For those times, we want a concise format.

**What is Serialized Data? (10 minutes / 0:20)**

All data sent via HTTP are strings. Unfortunately, what we really want to pass between web applications is **structured data** (i.e., arrays and hashes). Thus, native data structures can be **serialized** into a string representation of the data. This string can be transmitted and then parsed back into data by another web agent.

There are **two** major serialized data formats...

**JSON**

**JSON** stands for "JavaScript Object Notation" and has become a universal standard for serializing native data structures for transmission. It is light-weight, easy to read and quick to parse.

Remember, JSON is a serialized format. While it may look like an object, it needs to be parsed so we can interact with it as a true Javascript object.

*Unparsed*

{"users":[{"name": "Bob", "id": 23},{"name": "Tim", "id": 72}]}

*Parsed*

{ "users": [ {"name": "Bob", "id": 23}, {"name": "Tim", "id": 72} ] }

**XML**

**XML** stands for "eXtensible Markup Language" and is the granddaddy of serialized data formats (itself based on HTML). XML is fat, ugly and cumbersome to parse. Thankfully, JSON has mostly eliminated the need to ever use XML. You'll probably always favor using a JSON API, if available.

<users> <user id="23"> <name><![CDATA[Bob]]></name> </user> <user id="72"> <name><![CDATA[Tim]]></name> </user> </users>

**Many APIs publish data in multiple formats, for example...**

* <http://dev.markitondemand.com/Api/Quote/json?symbol=AAPL>
* <http://dev.markitondemand.com/Api/Quote/xml?symbol=AAPL>

**AGAIN, ALWAYS CHOOSE JSON WHEN YOU CAN**

**Where Do We Find APIs? (5 mins)**

APIs are published everywhere. Chances are good that most major content sources you follow online publish their data in some type of serialized format. Heck, [even Marvel publishes an API](http://developer.marvel.com/documentation/getting_started). Look around for a "Developers" section on major websites.

**That sounds hard. Can't you just give me a freebie?**

Try the [Programmable Web API Directory](http://www.programmableweb.com/apis/directory), [Mashape](https://market.mashape.com/explore) or the [Public APIs Github Repo](https://github.com/abhishekbanthia/Public-APIs/).

**You do**

Spend 10 minutes exploring these APIs and post some of the interesting data sets to the classroom channel in Slack.

**What Is An API Key? (5 minutes / 0:25)**

While many APIs are free to use, many others require an API "key" that identifies the developer requesting data access. This is done to regulate usage and prevent abuse. Some APIs also rate-limit developers, meaning they have caps on the free data allowed during a given time period.

**Try hitting the**[**Giphy**](https://api.giphy.com/)**API...**

* No key: <http://api.giphy.com/v1/gifs/search?q=funny+cat>
* With key: <http://api.giphy.com/v1/gifs/search?q=funny+cat&api_key=dc6zaTOxFJmzC>

**It is very important that you not push your API keys to a public Github repo.**

This is especially true when working with [Amazon Web Services (AWS)](https://aws.amazon.com/). Here's an example of a [stolen key horror story](https://wptavern.com/ryan-hellyers-aws-nightmare-leaked-access-keys-result-in-a-6000-bill-overnight).

**AJAX (5 minutes / 0:30)**

**So we know what an API is. Now what?**

How can we use an API to dynamically manipulate the DOM with the given data? **AJAX**. As you'll come to learn, this allows us to build single page applications that do not require refreshes.

**AJAX**, which stands for "Asynchronous Javascript and XML," is the method through which we are able to make HTTP requests. The standard requests we will be making are GET POST PUT PATCH and DELETE.

| **Type of Request** | **What's It Do?** | **An Example URL** |
| --- | --- | --- |
| GET | Read | <http://swapi.co/api/planets/> |
| POST | Create | <http://swapi.co/api/planets/> |
| PUT | Update | [http://swapi.co/api/planets/2/](http://swapi.co/api/planets/2) |
| PATCH | Update | [http://swapi.co/api/planets/2/](http://swapi.co/api/planets/2) |
| DELETE | Delete | [http://swapi.co/api/planets/2/](http://swapi.co/api/planets/2) |

**Q: Why do you think there is a "2" at the end of the URLs in the last three rows? (Hint: you could replace it with any number)**

**Break (10 minutes / 0:45)**

**GET Data From an API Using AJAX**

**I Do: GET From OMDB (15 minutes / 1:00)**

**Do not follow along for this portion of the lesson.** You will have the opportunity to do it yourself later.

Let's build a very simple app that posts a movie title and poster after searching for it. We'll do this using the [OMDB API](http://www.omdbapi.com/).

The starter code is linked above. It contains a basic HTML/CSS/JS setup. If you open up the HTML in the browser, you will notice that searching for something returns no results.

Let's go ahead and add in the AJAX request. There are many ways to write an AJAX request, but we will be using a library called axios during this class. Axios is a Promise-based HTTP client that takes advantage of many new JavaScript features and uses JavaScript's built in Promise API. Let's see it in action now.

// Get value from search input field. const keyword = document.getElementById("#keyword").value; const url = `https://www.omdbapi.com/?t=${keyword}&apikey=d31f1a94`; axios.get(url).then(response => { console.log(response); });

Three important things to notice about axios here...

1. The method called is .get(). We can also use .post(), .put(), .delete(), and more.
2. The .get() method only takes one argument. The url of the api we want to hit.
3. We can tell axios what to do with the data by calling .then().

**Q: How did we know which URL to use?**

**Promises**

After making an API call, we need to let our program know what to do with the data when it comes back. We can do this by doing something with the its return value, which comes in the form of a **promise**. We can use **promise methods** to tell the axios request what to do if the request is successful or not. We will use two promise methods when working with Axios...

**.then()**

.then() is called whenever the previous method returns information. In the case of AJAX calls, this typically means that .then() will be called once the information has been requested, processed, and sent back to our program. We use .then() to tell our program what to do with this JSON data.

axios.get(url).then(response => { var data = response.data document.window.append(data); });

**Q: What are we passing into the `.then` promise method? Does this remind you of anything we've done previously in class?**

**.catch()**

A promise method for when the AJAX call fails. These failures can occur for a variety of reasons. Internet went down, url is incomplete, the server you're hitting running into errors... it's important to know why things go wrong when they do. That's why the catch method allows you to catch and interpret errors when they occur.

.catch(function (error){ console.log("fail"); console.log(error) });

**async / await**

Promises are great! They save us from ["callback hell"](http://blog.mclain.ca/assets/images/callbackhell.png) and provide some very nice error-handling. The downside to Promises is that they require a fair amount of boilerplate code. Each synchronous step requires a brand new .then() or .catch() block, and each of those blocks requires a new anonymous function or arrow function.

With React's Webpack set-up, we have brand new ES7+ features available to us. Let's take a look at how async / await, one of the best new Javascript language features, will give us this same functionality with even less boilerplate!

Using async / await, we can re-write the axios calls above like this:

const getSomeData = async () => { const data = await axios.get(url) document.window.append(data) }

If we break this down, there are two steps we'll need to follow:

1. Any synchronous code must be run inside of a function with the async prefix. This is as simple as just typing the word async before our function definition.
2. Any time we want to stop and wait for a block of code to complete, we prefix the command with await. In the block of code above, we are telling our code to await the return of our axios call, and only then will we append the data to the DOM.

Note: We can await as many blocks of code as we want, as long as they are all inside of the async function.

While **Promises** are still totally valid, let's see if we can start refactoring our React code to use this great, clean syntax!

**BREAK**

**Making API Calls in a React Application.**

Through the use of AJAX and axios, we allow our React app to communicate with servers to retrieve and insert information. Since we are using AJAX, we can update our database and get information back from the server without ever having to refresh the browser. Let's build an application that communicates both with the Giphy API and with an Express API deployed on Heroku.

**Strange Things**

Giphy is home to some weird stuff... we're going to create a React app that allows users to save and rank the strangest gifs that Giphy has to offer as well as explore random gifs being sent by their API

**APIS**

We're going to using 2 APIs for this app: [**GIPHY**](https://developers.giphy.com/) [**Strange-Things-API**](https://strange-thing-api.herokuapp.com/api)

**Set Up**

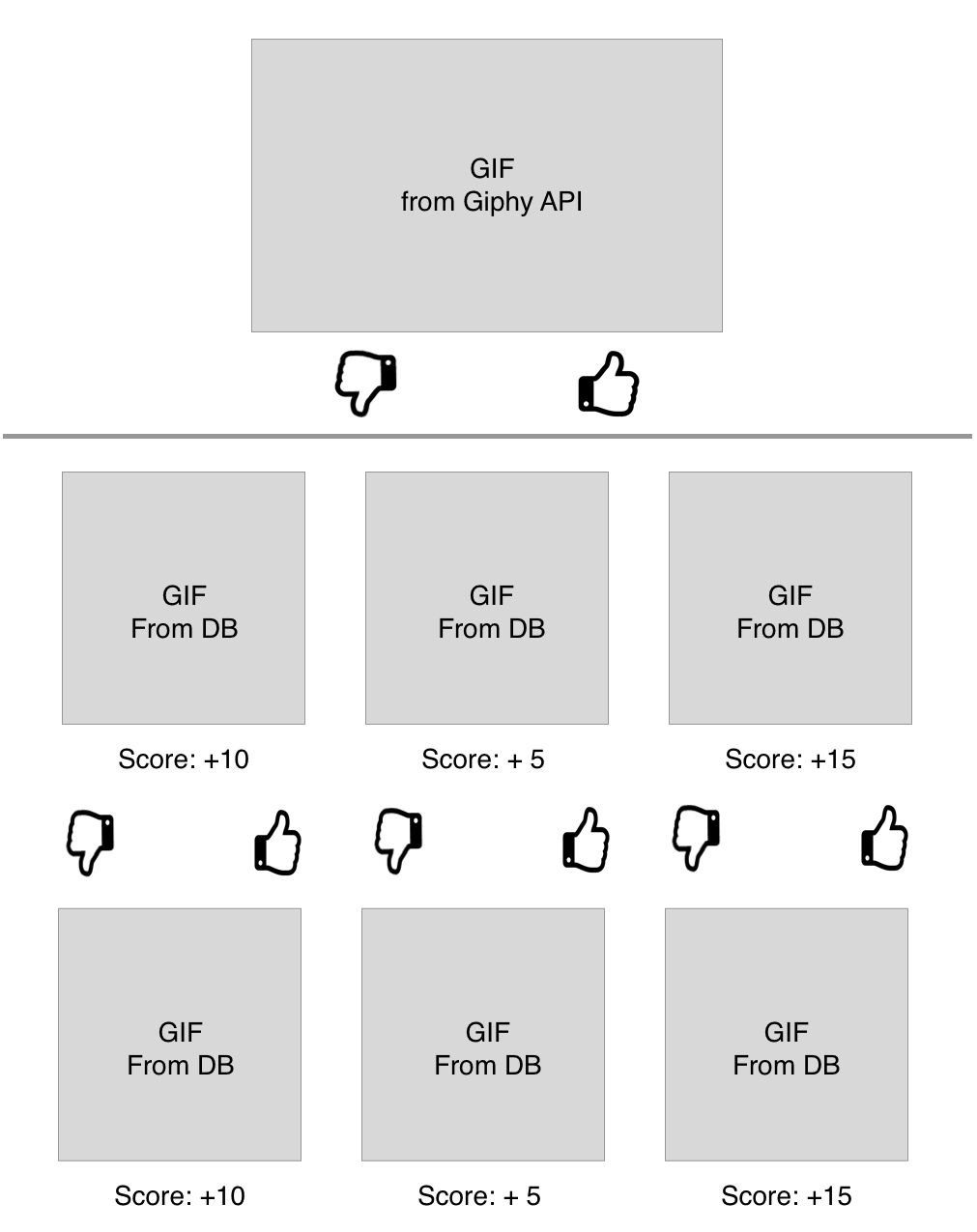
Let's start out by cloning some starter code. The starter code is generated from create-react-app but has some additional styling, fonts, and includes axios.

git clone https://git.generalassemb.ly/atl-wdi/React-Ajax cd react-ajax npm install npm start

If everything installed correctly, you should see a Hello World! page.

**Designing our Component Structure**

Before we start writing components, let's think about what our app is doing and how we should properly design it. Here is a basic wireframe of our app.

[](https://imgur.com/GxpiM6o.png)

**You Do: (5 min)**

Work with the students around you to and discuss how many different React components we need for this application. What would you name each of these components?

**Getting gifs from our Heroku API**

Let's start out by building the React component that will retrieve and render information about the gifs that are available on our API that is already deployed on Heroku.

// ./components/SavedGifts.js import React, { Component } from 'react'; class SavedGifs extends Component{ render(){ return( <h1>Hello From SavedGifs!</h1> ) } } export default SavedGifs; // ./App.js // Remember to import your new component in App.js import React, {Component} from "react"; import SavedGifs from './components/SavedGifs'; class App extends Component { render() { return ( <div className="App"> <img className="backgroundImg" src="http://i.imgur.com/DysM91b.png"/> <h1>Hello World</h1> <SavedGifs /> </div> ); } } export default App;

We should now see Hello From SavedGifs in our UI. Now we are ready to use axios to give us data from the server.

**Retrieving Data on Load**

We want axios to make an API call whenever React is ready to mount the SavedGifs component. Think back to our previous lesson on the React Component Lifecycle, what special method would be able to do this for us?

componentDidMount will allow us to initiate an API call and setState whenever the data gets back. Let's add that to our SavedGifs component. We'll walk through this process step-by-step.

import React, { Component } from 'react'; // Step 1: Import axios import axios from 'axios'; class SavedGifs extends Component{ // Step 2: Set your state to contain a property called 'savedGifs' state = { savedGifs: [] }; // Step 3: Call 'componentDidMount' for making your API call componentDidMount(){ // Step 4: Use axios.get to retrieve all saved gifs from our API axios.get("https://strange-thing-api.herokuapp.com/api") // Step 5: Save the response array to this.state.savedGifs .then((response) => { const savedGifs = response.data.strangeThings; this.setState({savedGifs: savedGifs}); }) .catch((error) => { console.error("Error: ", error); }); } render(){ // Step 6: Use '.map' to loop through this.state.savedGifs and save a variable with the jsx for all of your gifs. Finally, refer to this variable within the return statement. const savedGifs = this.state.savedGifs.map((gif, i) => ( <div className="gif" key={i}> <img src={gif.url} /> <div>Score: +{gif.strangeness}</div> </div> )); return( <div className="savedGifsContainer"> <h3>Previously Saved Gifs</h3> <div className="savedGifsFlexContainer"> {savedGifs} </div> </div> ) } } export default SavedGifs;

Let's hop back into our browser and we can see that we successfully retrieved and rendered all of the gifs and the scores associated with them.

**You Do: (30 min)**

Create another component called RandomGif.js and build a component that will retrieve and display a random gif generated from the Giphy api.

* Look at the wireframe above to determine where to put the component in the UI
* The URL you will be calling is http://api.giphy.com/v1/gifs/random
  + Use the [GIPHY Documentation](https://developers.giphy.com/docs/#random-endpoint) to figure out what request parameters you are required to send.
  + Use the [axios Docs](https://github.com/mzabriskie/axios) to figure out how to send request params when making a .get() call.
* Remember you are only going to show one gif (hint: you don't need to do a .map() with the info you get back)
* If the request fails, log "FAIL" in the JavaScript console.

**POST and PATCH**

The next thing we want to build for our app is a thumbs up and thumbs down button within RandomGif that will save a gif if we click thumbs up and will generate a new gif after we click.

First, let's create the events whenever a user if a user clicks thumbs up or down

// ./components/RandomGif.js ... componentDidMount() { this.getRandomGif(); } // Event to be attached to thumbs down. disapprove = () => { this.getRandomGif(); } // Event to be attached to thumbs up. approve = () => { // Prepare the response that you are posting to the API const payload = { url: this.state.gifUrl, strangeness: 1 }; // Use axios.post() and target the Heroku API and send the payload defined earlier axios.post(this.props.url + "/api", payload).then((res) => { // Get a new gif once the post is successful this.getRandomGif(); }); } // Move the axios.get to it's own function to DRY up the code. getRandomGif = () => { return axios.get("http://api.giphy.com/v1/gifs/random", { params: { api\_key: "SECRET\_API\_KEY", } }).then(res => { this.setState({ gifUrl: res.data.data.image\_original\_url }); }); } ...

Now that we have our actions defined, let's create elements in our jsx that can trigger the events.

... render(){ return( <div> <img className="randomGif" src={this.state.gifUrl}/> <br /> <div className="thumbs"> <i onClick={this.disapprove} className="fa fa-3x fa-thumbs-down"></i> <i onClick={this.approve} className="fa fa-3x fa-thumbs-up"></i> </div> </div> ) } ...

**HOLD ON** Let's go back and look at the wireframes. It looks like the thumbs up/down are also used in the saved gifs part of the app. This is a great opportunity to refactor the thumbs up/down into their own Component.

// ./components/Thumbs.js const Thumbs = (props) => { const { disapprove, approve } = props; return ( <div className="thumbs"> <i onClick={disapprove} className="fa fa-3x fa-thumbs-down"></i> <i onClick={approve} className="fa fa-3x fa-thumbs-up"></i> </div> ) } export default Thumbs; // ./components/RandomGif.js ... render() { return ( <div> <img className="randomGif" src={this.state.gifUrl}/> <br /> <Thumbs approve={this.approve} disapprove={this.disapprove} /> </div> ); } ...

**Q: What are some other ways to acquire data in the client?**

**You Do: PATCH (30 min)**

Now that you've seen me POST new gifs to the API, you will now write a PATCH that will increase or decrease the strangeness score by 1 every time the user clicks thumbs up/down.

* Use the Thumbs component we used before
* Create an approve and disapprove method within the StoredGifs class.
* Make an Axios request that will trigger a patch to https://strange-thing-api.herokuapp.com/api/:id

Since you will all be modifying the same API, **please only update gifs that you yourself have created!**

**Bonuses**

* If you haven't already, make it so that a successful GET, POST, or PATCH request re-renders the all gifs' score in the browser to reflect the API.

**Conclusion (5 mins)**

* What is an API?
* Why are APIs useful/important?
* What is AJAX?
* What information might we need to pass into an AJAX call?
* How do we go about interacting with the response of an AJAX call?

**Hungry For More?**

[Take a look at some API exercises used in other GA campuses](https://github.com/ga-wdi-exercises/fun_with_apis#need-an-app-idea)

**Resources**

* [Andy's blog](http://andrewsunglaekim.github.io/Server-side-api-calls-wrapped-in-ruby-classes/)
* [Postman](https://www.getpostman.com/)
* [Intro to APIs](https://zapier.com/learn/apis/chapter-1-introduction-to-apis/)
* [Practice with APIs](https://github.com/ga-dc/weather_teller)
* [Beautify your JSON in Chrome](https://chrome.google.com/webstore/detail/jsonview/chklaanhfefbnpoihckbnefhakgolnmc?hl=en)

import React, { Component } from 'react';

import axios from 'axios'

class RandomGif extends Component {

state = {

randomGif: []

}

componentDidMount() {

axios.get(

"https://api.giphy.com/v1/gifs/random?api\_key=mPEAVABXFZAdVeg9TQX19BvPYhBpA5yp&tag=&rating=G")

.then((response) => {

const randomGif = response.data.data

this.setState({ randomGif: randomGif })

})

.catch((error) => {

console.error("Error: ", error);

});

}

render() {

return (

<div>

<img src={this.state.randomGif.image\_url} />

<div>Score +</div>

</div>

)

}

}

export default RandomGif;

An example of a component that creates a random GIF from Giphy.

We can create an api and use axio to call it to yourelf.