**I. Overview**

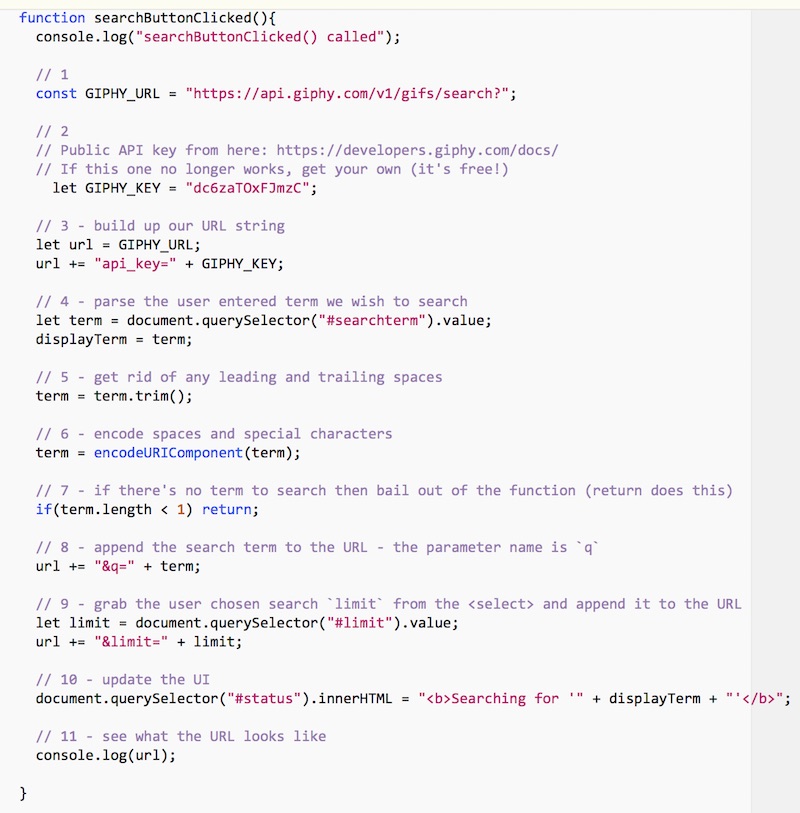
* *GIF Finder* will utilize the Giphy **web service** to search for GIFs based on keywords, and then display the results to the user.
* **What is a Web Service?** *A Web service is a method of communication between two electronic devices over World Wide Web. A Web service is a software function provided at a network address over the web or the cloud; it is a service that is "always on" ...* - <http://en.wikipedia.org/wiki/Web_service>
* The documentation for the Giphy Web Service is here: <https://developers.giphy.com/docs/>
* A summary of what this app will do:
  1. The user will type in a search term and hit return.
  2. The app will create a properly formatted URL (that includes the search term) that points at the web service.
  3. The app will send a request to download the file located at that URL.
  4. A few seconds later the response (a file) will be downloaded by the web browser.
  5. The downloaded file is in the JSON (JavaScript Object Notation) format - it's basically an object literal with just data, no methods.
  6. The app will parse the JSON data and display the results to the user.

**II. Web Service Terms**

1. **Web Service** - <http://en.wikipedia.org/wiki/Web_service> - A Web service is a method of communication between two electronic devices over World Wide Web. A Web service is a software function provided at a network address over the web or the cloud; it is a service that is "always on" ...
2. **API** - <http://en.wikipedia.org/wiki/Api> - An application programming interface (API) specifies how some software components should interact with each other.
3. **Query String** - <http://en.wikipedia.org/wiki/Query_string> - In the World Wide Web, a query string is the part of a uniform resource locator (URL) that contains data to be passed to web applications such as CGI programs.
4. **Arguments** (or parameters) - <http://en.wikipedia.org/wiki/Parameters_(computer_science)> - In computer programming, a parameter is a special kind of variable, used in a subroutine to refer to one of the pieces of data provided as input to the subroutine.
5. **API Key** - <http://en.wikipedia.org/wiki/API_key> - An application programming interface key (API key) is a code passed in by computer programs calling an API (application programming interface) to identify the calling program, its developer, or its user to the Web site. API keys are used to track and control how the API is being used, for example to prevent malicious use or abuse of the API (as defined perhaps by terms of service).

## IV. Capture the user intent and format a URL

We need to write code to build a URL to the web service. This URL will contain the user's search preferences (search term and number of results).

**Type this code in:**

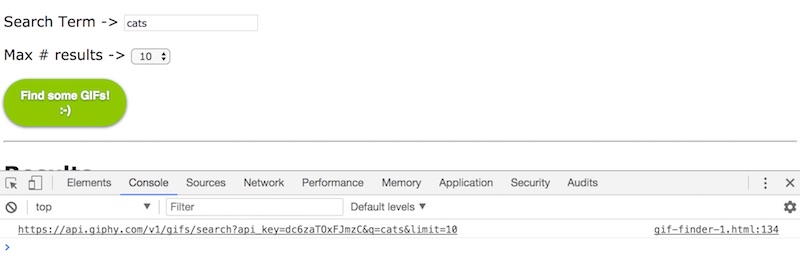
### A. Explanation (Read this! Don't just type in the code without understanding it!)

* #1 above - this URL is the Giphy Search endpoint. Here's an example of another endpoint, the Giphy "Trending" endpoint: https://api.giphy.com/v1/gifs/trending
* #2 above - this API key identifies you to the owner of the service. API keys are used to track and control how the API is being used - so if the user of this key is abusing the service it can be "turned off". This particular key is a public key and may no longer work when you do the exercise. If it does not, head here to get your own key (it's free!): <https://developers.giphy.com/docs/>
* #3 above - we specify a parameter - api\_key - and then give it a value
* #4 above - get the .value of the text input field
* #5 above - get rid of leading and trailing spaces. URLs do not work with spaces! Read about String.trim() here: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String/Trim>
* #6 above - encodeURIComponent() will escape characters like spaces (because we still might have spaces in the middle of the search term), ampersands, $ signs, + symbols and so on so that they are properly represented for a URL. For example, a space becomes %20. You can read the docs here: <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/encodeURIComponent>
* #7 above - bail out if we have nothing to search for
* #8 above - add the search term to the url - the web service requires this parameter name to be q
* #9 above - grab the value of the <select>, and then add the limit parameter to the url. Note that parameters are formated as name=value and separated by ampersands.
* #10 above - update the UI with the user's search term
* #11 above - finally! log out the URL

## V. Testing the URL

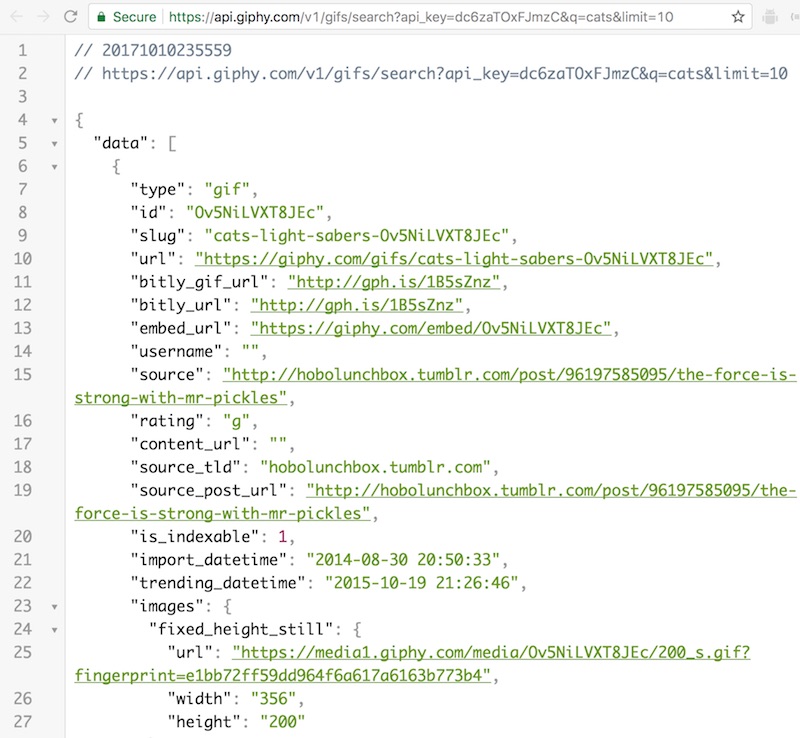
* Run the code and click the button. Now head to the console and you should see the complete URL to the web service.
* Think of this URL as being like calling a function and expecting to get a value back, except in this case the function resides on a remote server.
  + Another difference is that when you call a function in your code, you pass arguments like this doStuff(arg1,arg2,arg3)
  + With a web service, you pass arguments in the query string like this http://www.sample.com/doStuff?arg1=stuff&arg2=moreStuff&arg3=evenMoreStuff
* Now click on the completed URL in the console, it should open a new browser window that will show you the results of the Giphy API request. (If nothing happens, check that your code is correct, and/or copy/paste the URL to a browser location box)

### A. After clicking the search button, you should see the URL in the console



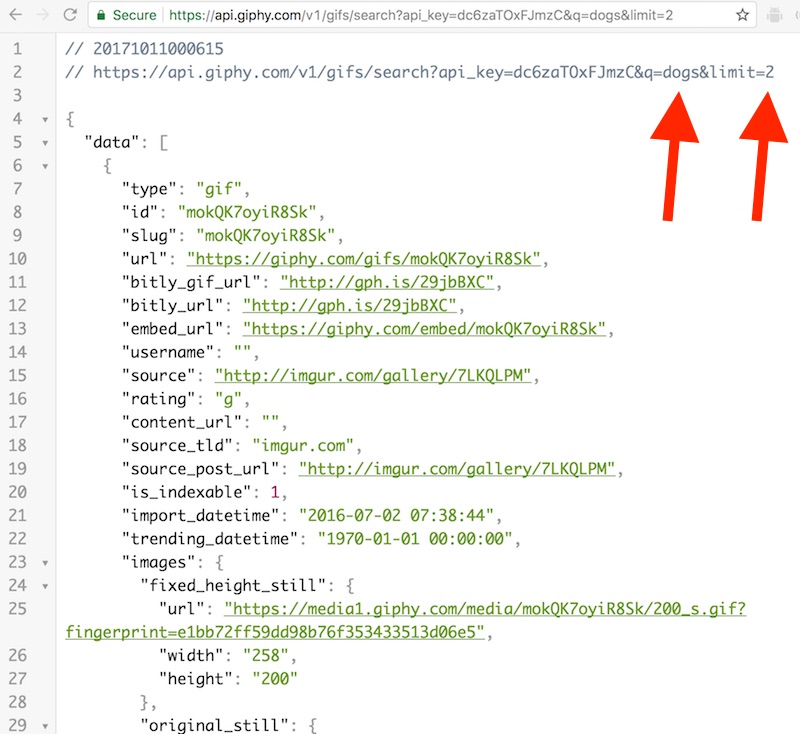
### B. Clicking on the link opens a web page that shows the JSON response to your search.

* You can see that you are getting back a JS Object literal, and there is a top-level property named data that contains an array of objects. Each of these objects represents a animated GIF on Giphy - note some of the properties that will come in handy: url, rating, width, height and so on.
* If your JSON isn't as nicely formatted as mine, it's because I am using the Chrome JSON Viewer extension which you can get here: <https://chrome.google.com/webstore/detail/json-viewer/gbmdgpbipfallnflgajpaliibnhdgobh>

[](https://github.com/tonethar/IGME-235-Shared/blob/master/tutorial/_images/gif-finder-5.jpg)

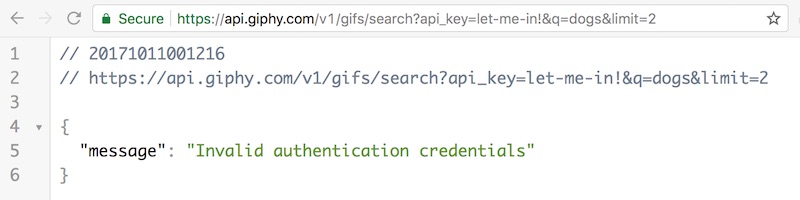
### C. Modify the URL in the location box

Below we changed the q parameter to dogs and the limit parameter to 2 - you can see we received a different JSON file from the server!

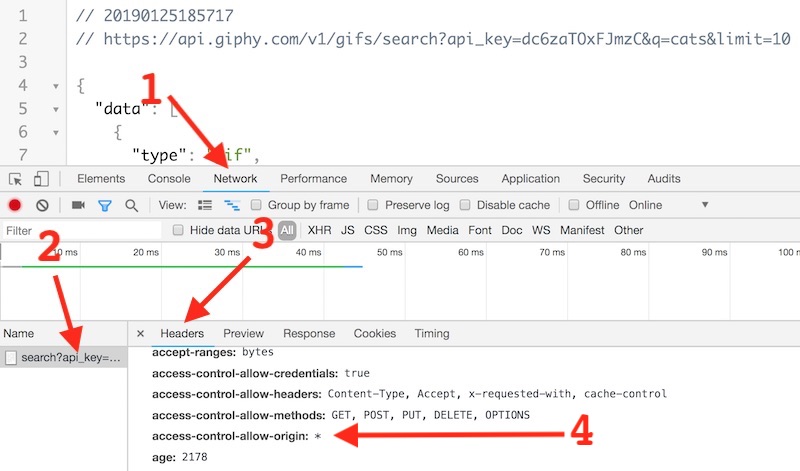
[](https://github.com/tonethar/IGME-235-Shared/blob/master/tutorial/_images/gif-finder-6.jpg)

### D. Modify the API key value

If you do that, oops, you will probably get an error message (unless it is a valid API key)

[](https://github.com/tonethar/IGME-235-Shared/blob/master/tutorial/_images/gif-finder-7.jpg)

### E. Check out the HTTP Response Headers that the web service sent

[](https://github.com/tonethar/IGME-235-Shared/blob/master/tutorial/_images/gif-finder-16.jpg)

* You can see these by opening the Chrome web inspector and:
  + #1 --> Select the **Network** tab (and then reload the page)
  + #2 --> Click on the file name on the left
  + #3 --> Click on the **Headers** tab
* #4 --> See the access-control-allow-origin: \* header?
  + This header means that a web browser from any domain (indicated by \*) is allowed to download the data
  + This is the header we like to see, because it means that our JavaScript app will definitely be able to get the data
  + When you are working on your web service project, ideally your chosen web service has this turned on
  + Please read about **Cross-Origin Resource Sharing (CORS)** --> <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> - which is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin (and access-control-allow-origin: is an element of CORS)

## VI. Utilizing XMLHTTPRequest

* To download the data, we are going to utilize the XMLHTTPRequest API (also known as **XHR**)
* You can read about this API here: <https://developer.mozilla.org/en-US/docs/Web/API/XMLHttpRequest>
* What MDN says --> "Use XMLHttpRequest (XHR) objects to interact with servers. You can retrieve data from a URL without having to do a full page refresh. This enables a Web page to update just part of a page without disrupting what the user is doing. XMLHttpRequest is used heavily in AJAX programming."

## VII. Downloading the data with XHR

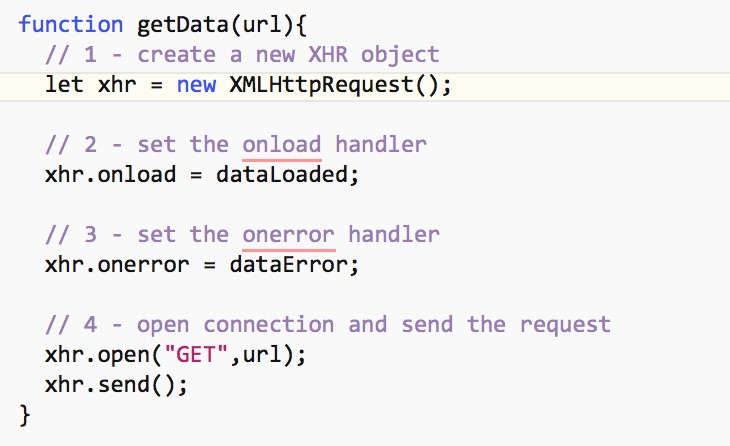
### A. Now we are going to tell XHR to download the data from the URL we assembled above

**Add this code to the bottom of searchButtonClicked():**

// 12 Request data!

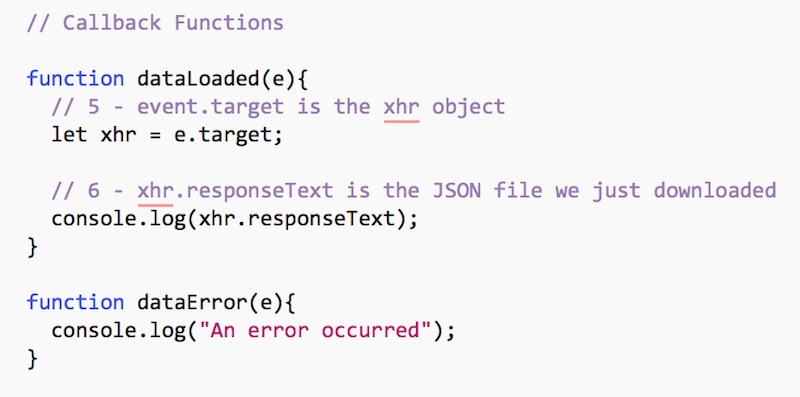
getData(url);

**Here is the implementation of getData():** - place this code outside of searchButtonClicked():\*\*



### B. Create the callback functions

**Below are the two callback functions - place this code outside of getData() and searchButtonClicked():**

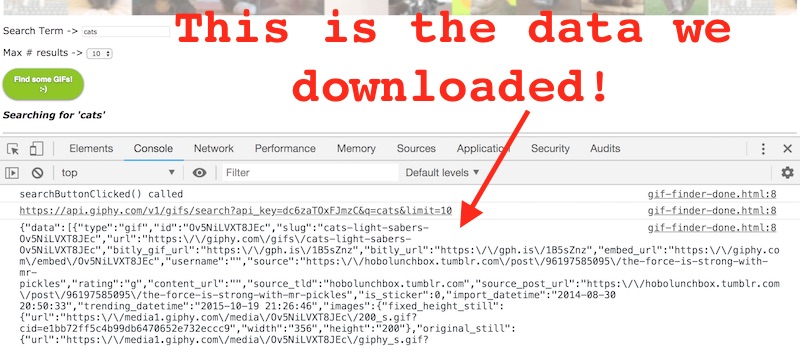


### C. Explanation

* #1 above - we create a new XHR object
* #2 - set the onload handler - this is called when the data is successfully loaded
* #3 - set the onerror handler - this is called when some errors occur
* #4 - open the connection and send the request. "GET" is the HTTP method we are using - it means that we are sending web service parameters in the query string (part of the URL) and not as a separate file (which is what the "PUT" method is)
* #5 - after the data has loaded, get a reference back to the XHR object
* #6 - log out the JSON file we have downloaded

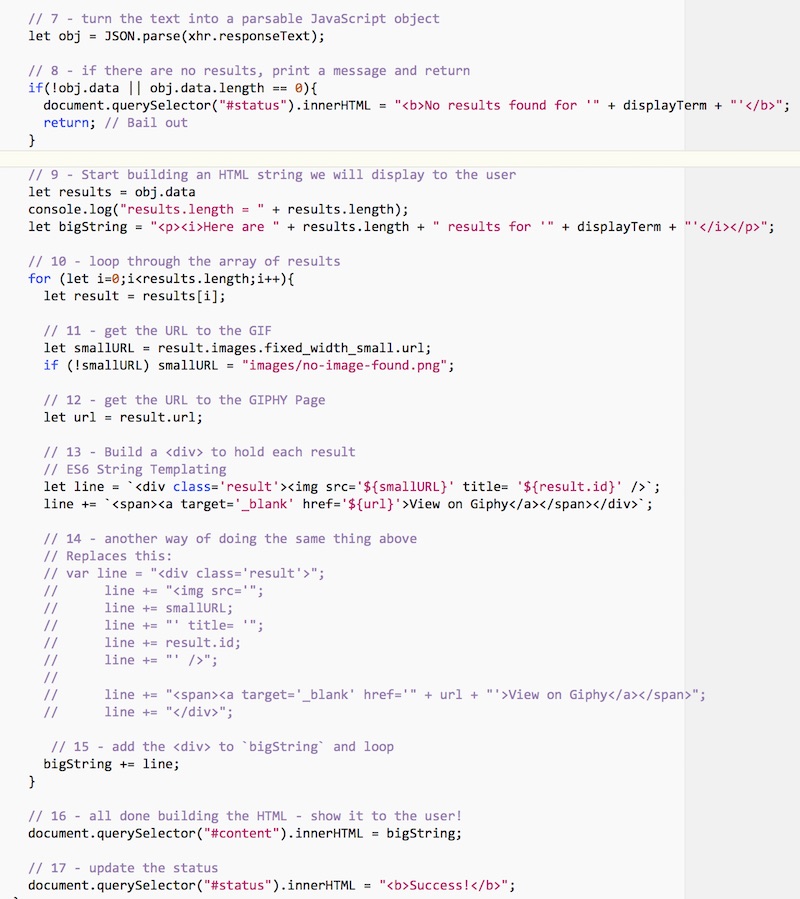
### D. Run the app and click the search button

* You should see the JSON we downloaded in the console:



## VIII. Formatting the results for the user

* Now we just need to take the results, loop through them, and create some HTML. Here's the code you need to add to dataLoaded():



* Go ahead and run the code and test the app by performing a search. It should pretty much look like the second screenshot at the top of this page.

### A. Explanation

* #7 - above we turn the text into a parsable JavaScript object
* #8 - if there are no results, bail out by returning
* #9 - start building the HTML string
* #10 - loop though the array of results
* #11 - get the URL to the GIF - we will use this in an <img> tag
* #12 - grab the main Giphy page URL, which we will later put in a link
* #13 - build a <div> for each result - here we are using ES6 template string. The ` character is a backtick, not a single quote. You can find this character on the upper-left of your keyboard.
* #14 - The old-fashioned way to build a string utilizing concatenation
* #15 - add the <div> to bigString and loop
* #16 - all done building the HTML - show it to the user by updating the #content <div>
* #17 - update the #status <div>

## IX. Wrap up

* Looks like you are done with most of the coding! Congratulations on your typing! ***Bring it with you to Lab (if you have in-person Lab, see below).***
* There are a lot of new concepts in this exercise - please review the [10 - Web Services](https://github.com/tonethar/IGME-235-Shared/blob/master/tutorial/web-apps-10.md) chapter, this page and all of the GIF Finder code to be sure you understand everything. If you don't understand something, please be prepared to ask questions about it in class (or on Slack)!

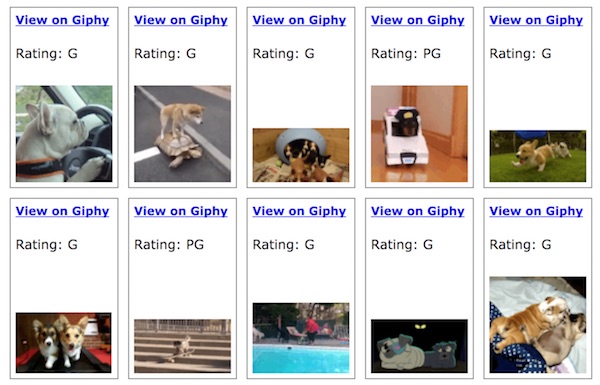
## X. In-Lab Portion

***We will discuss and implement the following tasks in the GIF Finder Follow-up Video. If you attend lab in-person, you may go ahead and try to do everything in advance if you wish... If you do, you can spend the next lab working on the more advanced/optional challenges, extra credit exercises, and/or researching what you'll do for Project 2.***

1. Write JS that utilizes the rating property of each result by displaying it for each GIF. You can use .toUpperCase() to capitalize each rating (see example screenshot below).
2. This HTML page is now a kluge of quite a bit of HTML, CSS, and JavaScript. We need to clean it up by moving the JS code and CSS into different files - "separation of concern" is the term for this.
   * Move all of your CSS to a **styles/main.css** file and link to it. Note: You will need to slightly change the url of the <header> background image rule to keep things working.
   * Move all of your JavaScript to a **js/main.js** file and link to it.
   * Your HTML file should now consist of only HTML - see the screenshot below.
   * Verify that everything still works.
3. Improve the CSS styling of this page (which shouldn't be too hard). See the screenshot below as an example of what you could do.
4. (**optional**) Did you notice **spinner.gif** in the start images? Go ahead and show it when the user starts a search, and hide it when a search has completed. Maybe put it in the #status <div>

* You will be expected to submit your finished version to the GIF Finder Assignment on myCourses by its due date.

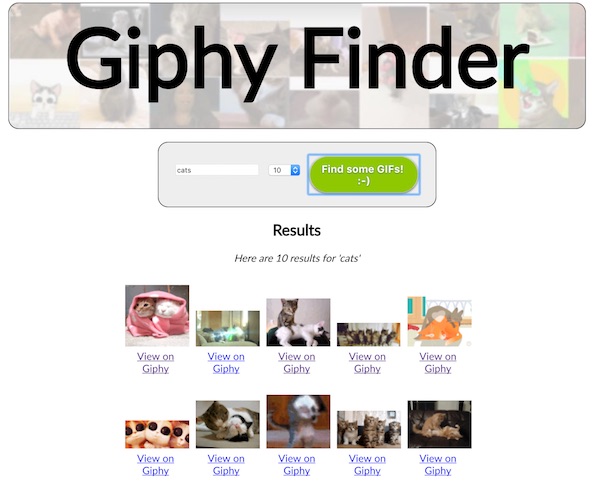
#### A. Results that display a rating for each GIF



#### B. Cleaned up HTML page, with CSS and JS kept in separate files



#### C. Better page design



## XI. Totally Optional - (also, things you may be expected to do depending on what "adventure" you pick for Project 2)

* You might have noticed that no matter how many results you request, you never get more than 25 back? That is because the public key we gave you is capped at 25 results. **Solution:** Get your own API key! (it takes 2 minutes and it's free)
* Add **next** and **previous** buttons - another really nice option is to allow the user to "page" through large numbers of results. Did you notice that we always get the same 100 "cat" GIFs back when we search?
  + This is because there are ***thousands*** of cat GIFs on GIPHY, and if we don't otherwise specify we will always get them returned from the web service starting at index 0, which means we always get the first 100 (index 0-99) back.
  + We can instead write code that requests a higher starting index.
  + In the GIPHY API this can be done by tracking and adding an offset value to the query string that is sent over to the API.