

# Scene It App

## *Part 3: Querying the OMDB API*

### Your task :

Once you've finished part 2, your Scene It app should allow the user to save movies to their "watch list". Now that the core of your front end application is finished, your task for part 3 is to hook up your front end to the OMDB API back end!

When you're done, you'll be able to delete *data.js* - instead of pulling in that hard coded data, we'll be pulling in search data from [omdbapi.com](http://omdbapi.com)!

### Let's do it!

#### Step 1 - Look at the code we're going to refactor

Take a look at the callback function inside of your search form submit listener. So far, this code just renders the movie data array in *data.js*, saves the resulting HTML string to **movieHTML**, and then uses `innerHTML` to insert that HTML into the DOM (specifically, inside the `<div>` with `class="movies-container"`). Instead of passing **movieData** into `renderMovies()`, we're going to pull a similar array from the OMDB API and pass that through `renderMovies()`.

#### Step 2- Get the search string from the `<input>` tag

We know that the submit listener's callback will trigger whenever the user submits a search string in the search bar. We might as well grab the string they typed and save it to a variable!

- 1) Create some empty space right after `e.preventDefault();`
- 2) In that space, use the javascript to select the search bar and save its content to a

variable called **searchString**

a) You could use a line like this:

b) `const searchString = document.getElementById('search-bar').value`

We need this **searchString** variable because we'll be passing it to the OMDB API in the *querystring*. The url we'll be targeting is:

**`http://www.omdbapi.com/?apikey=59354c85&s=SearchStringGoesHere`**

Our goal is to inject the contents of the **searchString** variable right after the `&s=` part of the url. For example if the user inputs "Fun", the url we'll target is:

**`http://www.omdbapi.com/?apikey=59354c85&s=Fun`**

But what if the user includes spaces in the url? Like with "Fun with"?

**`http://www.omdbapi.com/?apikey=59354c85&s=Fun With`**

Last time I checked, urls aren't allowed to have spaces in them! Thankfully, we can "sanitize" the user's input with a handy built-in function called **encodeURIComponent()**. If we use this function on **searchString**, we'll end up with a url-friendly search string like this:

**`http://www.omdbapi.com/?apikey=59354c85&s=Fun%20With`**

Notice that the space got replaced with `%20` in the querystring!

- 1) Create another variable called **urlEncodedSearchString**
- 2) Set it equal to **encodeURIComponent()**
- 3) In the same line, pass in **searchString** as the parameter
- 4) You should end up with a line like:
- 5) **`const urlEncodedSearchString = encodeURIComponent(searchString);`**

### Step 3- Use ajax to query the OMDB API

Now that we know what the user wants to search for, we can use ajax to poll the OMDB API. You can take a look at the documentation for the omdb api here:

<http://www.omdbapi.com/>

According to their website, you can request movies by imdbID or movie title, *OR* you can

search for movies using a keyword (which is exactly what we want to do!). Notice that the only thing we need to do is provide two *querystring* parameters, **apikey** (which is the super-secret password I pay \$1 per month for), and **s** (which is the keyword/s we're searching for). That's why, as I mentioned above, the url we want to hit is:

**`http://www.omdbapi.com/?apikey=59354c85&s=SearchStringGoesHere`**

You can use Postman to test out a search for the keyword "Dark" - you should see a response like: (not shown in the screenshot below are the keys "totalResults" and "Response")

```
1 {
2   "Search": [
3     {
4       "Title": "The Dark Knight",
5       "Year": "2008",
6       "imdbID": "tt0468569",
7       "Type": "movie",
8       "Poster": "https://ia.media-imdb.com/images/M/MV5BMTMxNTMwODM0NF5BMl5BanBnXkFtZTcwODAyMTk2Mw@@._V1_SX300.jpg"
9     },
10    {
11      "Title": "The Dark Knight Rises",
12      "Year": "2012",
13      "imdbID": "tt1345836",
14      "Type": "movie",
15      "Poster": "https://images-na.ssl-images-amazon.com/images/M/MV5BMTk4ODQzNDY3Ml5BMl5BanBnXkFtZTcwODA0NTM4Nw@@._V1_SX300.jpg"
16    },
17    {
18      "Title": "Thor: The Dark World",
19      "Year": "2013",
20      "imdbID": "tt1981115",
21      "Type": "movie",
22      "Poster": "https://ia.media-imdb.com/images/M/MV5BMTQyNzAwOTUxOF5BMl5BanBnXkFtZTcwMTE0OTc5OQ@@._V1_SX300.jpg"
23    },
24    {
25      "Title": "Transformers: Dark of the Moon",
```

Notice that the array under the key "Search" looks *exactly* like the hard coded data we've been using in *data.js*!

Alrighty, enough dilly-dallying, let's get to it!

- 1) Still inside the submit callback, use `fetch` to get the data from the OMDB Api. Pass the url into `fetch()`

`"http://www.omdbapi.com/?apikey=59354c85&s=" + encodeURIComponentSearchString`

- 2) Chain a `.then()` after `fetch()`;
- 3) Provide `.then()` with an anonymous function that has **response** as its parameter.
- 4) Use the `response.json()` method to parse the response from JSON into a something we can use

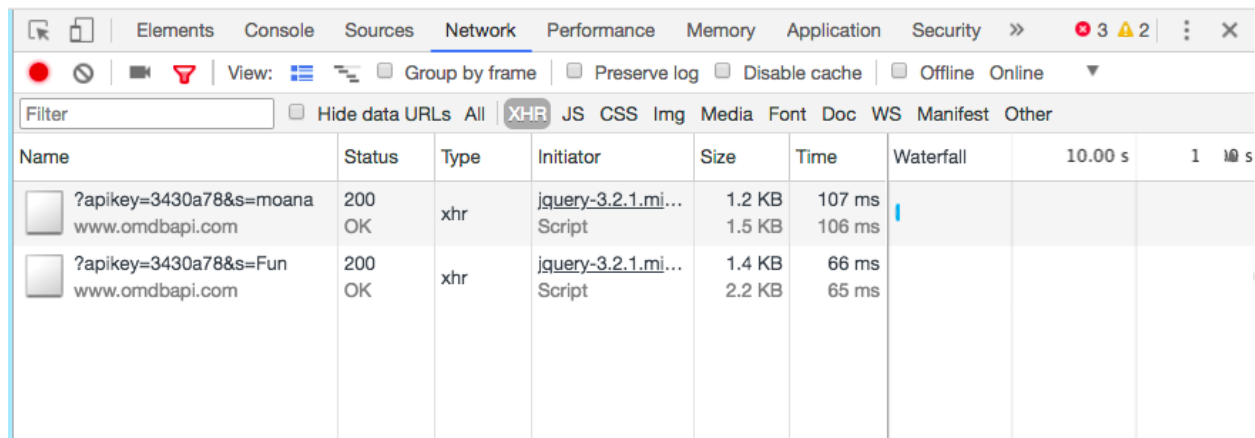
```
function(response) {
  return response.json();
}
```

5) Write another `.then` method and pass data to the function.

```
function(data) {  
    // use data here  
}
```

If we were using jQuery, we would use `$.ajax()` instead of `fetch()` and we would pass data to the first `.then()` callback function instead of response.

We're not done yet! We still need to pass the omdb results through our **renderMovies()** function. Now, when you submit a new search, you should see network requests showing up in the *Network* tab in Chrome Developer Tools.



The screenshot shows the Chrome Developer Tools Network tab. The 'Network' tab is selected, and the 'XHR' filter is applied. Two requests are visible, both to 'www.omdbapi.com' with status '200 OK' and type 'xhr'. The first request is for 'moana' and the second is for 'Fun'. Both requests are initiated by 'jquery-3.2.1.min.js' and have a size of 1.2 KB and 1.5 KB respectively. The time taken for each request is 107 ms and 66 ms respectively. The waterfall view shows the requests are sequential.

Name	Status	Type	Initiator	Size	Time	Waterfall	10.00 s	1 s
?apikey=3430a78&s=moana www.omdbapi.com	200 OK	xhr	jquery-3.2.1.min.js	1.2 KB 1.5 KB	107 ms 106 ms			
?apikey=3430a78&s=Fun www.omdbapi.com	200 OK	xhr	jquery-3.2.1.min.js	1.4 KB 2.2 KB	66 ms 65 ms			

## Step 4- Show OMDB's movie data using renderMovies()

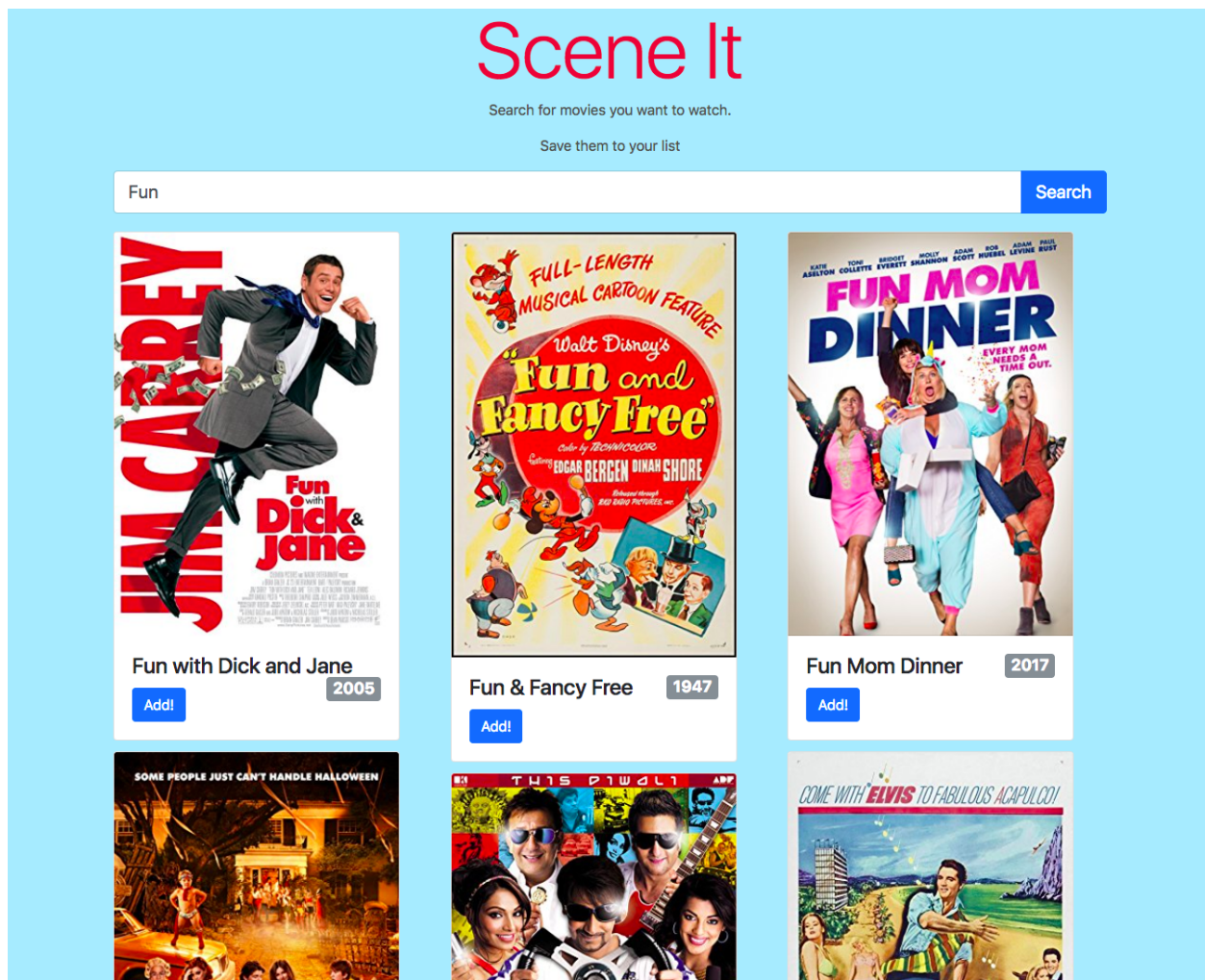
We're at the finish line! If you haven't deleted your old code already, you should see two lines in your code like:

```
const movieHTML = renderMovies(movieData);  
moviesContainer.innerHTML = movieHTML;
```

- 1) Go ahead and copy these two lines to your clipboard and then delete them.
- 2) Paste them into the **.then()** callback function
- 3) Replace **movieData** with **data.Search**

- a) Remember, when we tried the OMDB query in Postman, the movie data was contained under the “Search” key.
- b) The parameter **response** contains the data that came back from the OMDB API

That’s it! Now you should be able to search for whatever movies you want!



- 4) There’s 1 more step. What happens if you click the ‘add’ button now and try to save a new result to the watchlist? It doesn’t work! This is because our `.find()` method is still looking for new movies in the old, hardcoded data. Let’s fix that.
  - a) We can just update the original `movieData` variable to store the new search

results:

```
movieData = data.Search
```

b) Try a new search, and then try the add button.

## **That's it! What's next?**

Dang, that's a pretty spiffy application you've got there! It's definitely something to be proud of!

First things first, let's make sure your code on github is up to par. You should probably:

- 1) Delete any scratch code
- 2) Remove any unhelpful `console.log()`s or comments
- 3) Make sure the version you push up to github is in working order