Scene It App

Part 1: Rendering Movies and Search

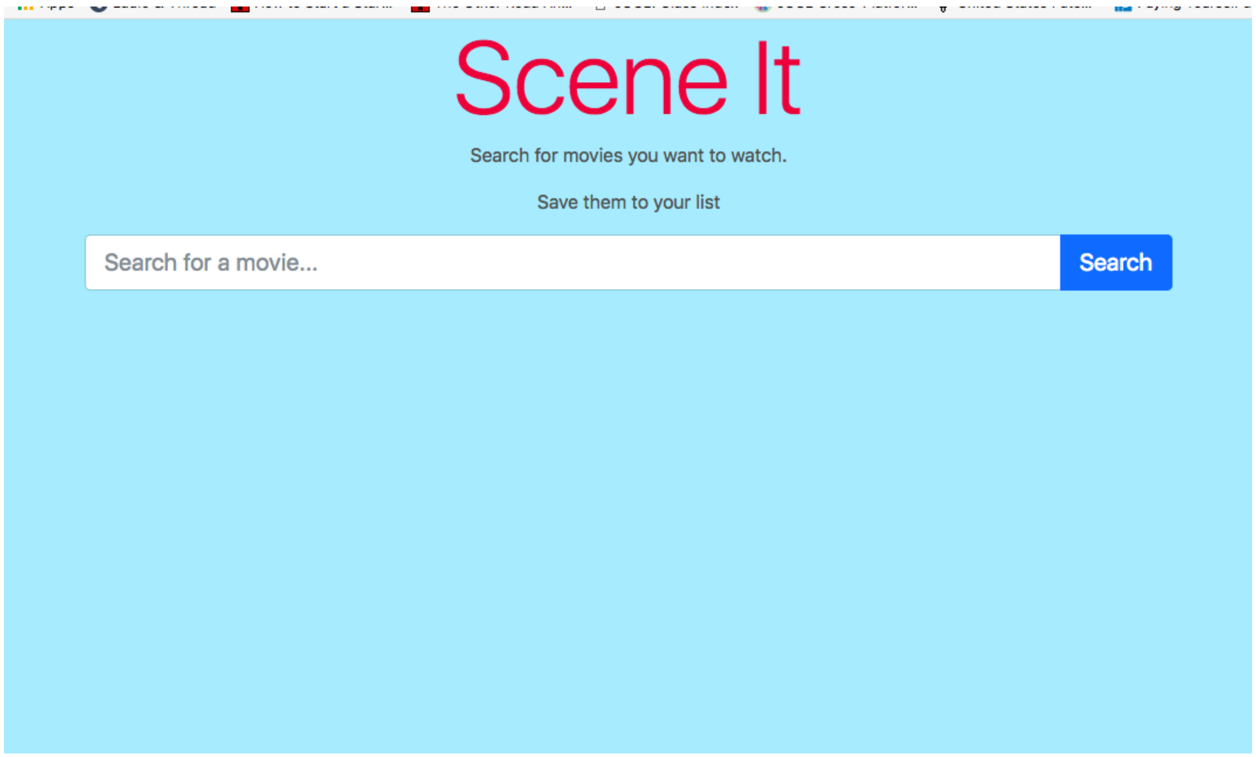
Objective

Now that we've got a good general understanding of how we can render data to the DOM, we're going to take a stab at building a practical application, worthy of a spot in our portfolios!

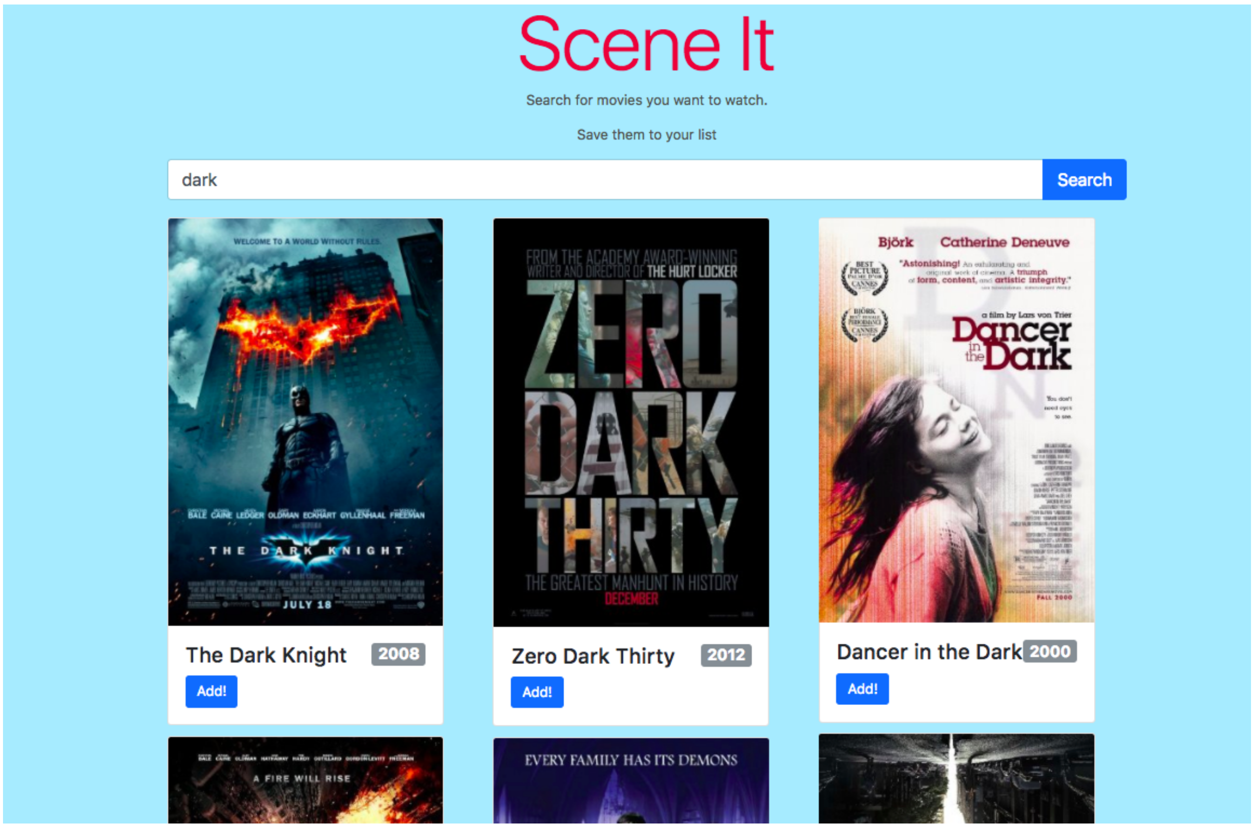
The app is called Scene It (lol, get it? Like a movie "scene"? #sorrynotsorry). It's a simple tool for searching for movies, flagging ones you want to watch, and later giving them a thumbs up or thumbs down after you watch them. The finished product will use AJAX to make requests to an external API called OMDB, but we'll handle that in part 2!

Your task :

Your task is to build a UI that "consumes" the data that comes from the OMDB API. When you fork the starter repository called scene-it-starter, you'll see something like this:



When you're done, you might have something that looks like this:



Since we're not actually using the OMDB API until part 2, for now there's a file called *data.js* that contains some movie data. The search bar will "work", but it will always show the list of movies in *data.js*, no matter what you enter into the search bar.

Let's do it!

Step 0 - Clone the starter repo

Make a fork of the following repo and then clone it into your projects folder

https://github.com/lachieh/scene-it-starter

Step 1 - Design the HTML

Before we get to writing some javascript, we might as well tackle the HTML and CSS required. In the screenshot above, you can see that movies show up as Bootstrap Cards (https://getbootstrap.com/docs/4.4/components/card/) . You can try to code something that looks *exactly* like the screenshot, but I think adding your own spin to it might be better! For example, instead of using Bootstrap Cards, you could have the search results show up as rows in a table. It's up to you! No matter how you design your app, the general process is the same:

1) In *index.html*, find the comment that says `<!-- MOVIES SHOW UP HERE! -->`
2) Add a `<div>` tag with the class "movies-container". This div will contain all the individual movie renderings.
3) Inside the "movies-container" div you just created, add another `<div>` tag with the class "movie". This div will represent a single movie.
4) Now we add the first movie placeholder. Using your HTML (i.e. not javascript), flesh out your movie div. Note that all of this is just placeholder information for now:
    a) Your movie div should contain an `<image />` tag somewhere for the movie poster.
    b) Include a tag for the movie title
    c) Include a tag for the movie release date
    d) Include an "add" button somewhere
5) Style your movie `<div>` and its children with CSS.
    a) If you end up using Bootstrap cards, you might not have to do any CSS!
6) Once you're satisfied with your movie `<div>`, copy and paste it a couple of times to see what it looks like with multiple results
7) Style your "movie" `<div>` and the "movies-container" `<div>` to make sure there's enough padding, margin, etc. between movies. It is a good idea to make use of the bootstrap column layouts for this (or you can use flex-box if you want!)

Step 2- Render movies

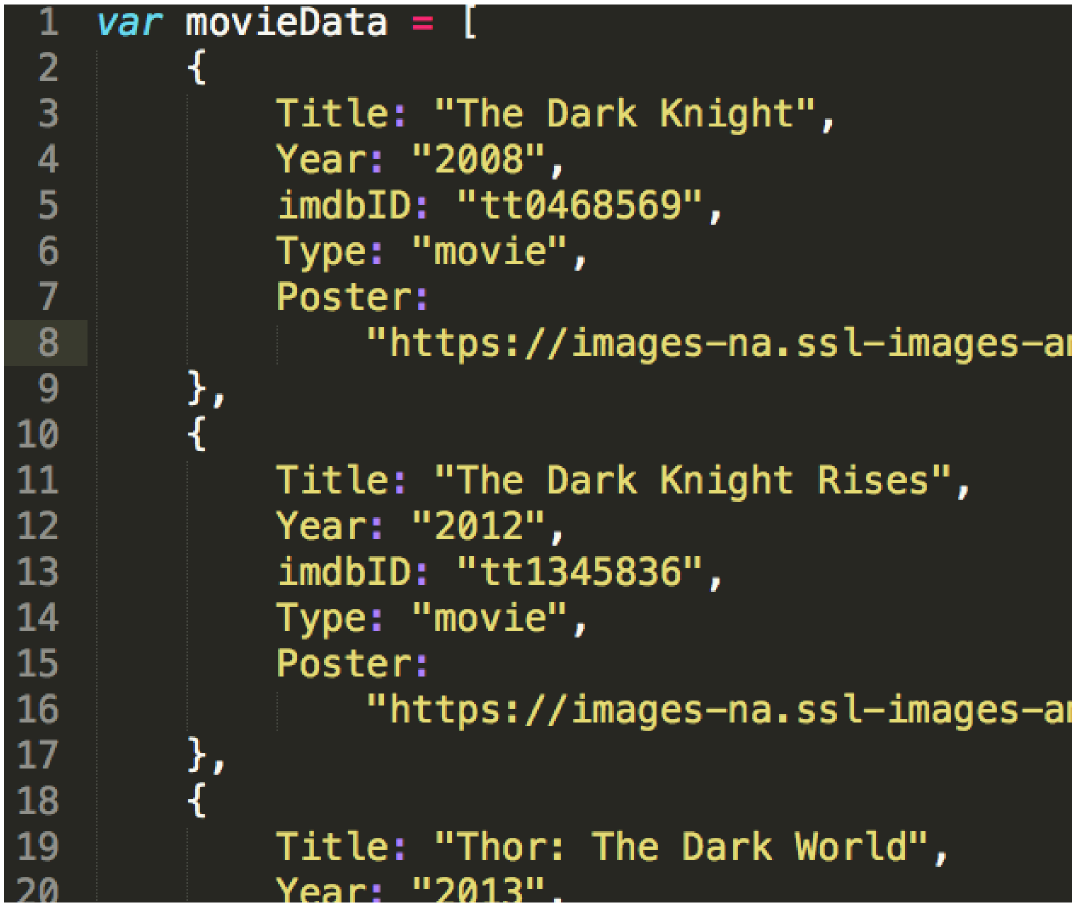
Now that you have the appropriate HTML and CSS, you can write a renderMovies() function that will generate the right HTML based on the data in *data.js*.

When you're done with this step, you'll have a function that will take in **an array of movie objects** and return **a string of HTML that looks like the HTML you wrote in step 1**

1) In *index.js* start off by writing a "document ready" block

```
document.addEventListener('DOMContentLoaded', function() {
    // code here will execute after the document is loaded
});
```

2) Outside of the “document ready” block, define a function called **`renderMovies`**
3) Make the function take one parameter called **`movieArray`**
4) Write a `.map()` loop on the **`movieArray`** parameter:
    a) If you need help with the Array.map() method, take a look at the mdn docs for an example
    b) The .map() method on an array takes an anonymous function as a parameter and returns a new array.
    c) This anonymous function should have one parameter, you can call it **`currentMovie`**
    d) Take the result of this .map() and save it to a variable called **`movieHtmlArray`**
5) Take a step back! Before you put anything inside of your new map function, let’s look at the data in *data.js:*



```
var movieData = [
    {
        Title: "The Dark Knight",
        Year: "2008",
        imdbID: "tt0468569",
        Type: "movie",
        Poster:
            "https://images-na.ssl-images-a
    },
    {
        Title: "The Dark Knight Rises",
        Year: "2012",
        imdbID: "tt1345836",
        Type: "movie",
        Poster:
            "https://images-na.ssl-images-a
    },
    {
        Title: "Thor: The Dark World",
        Year: "2013",
```

    a) Notice that this array contains many objects
    b) The movieData variable will be passed in as **`movieArray`** in the renderMovies function we're currently writing
    c) That means that inside of our map function, the **`currentMovie`** parameter will be *just one* of these objects from inside of the **`movieArray`**
    d) If we want to use a movie's title, for instance, you'd would access it using the dot notation of **`currentMovie.Title`**
6) Knowing this, use template literals (a string with backticks ``) on **`currentMovie`** to start building out your movie HTML. For example, this will return an h2 inside of a div with the current movie's title:
```
movieArray.map(function(currentMovie) {
    return `<div>
        <h2>${currentMovie.Title}</h2>
    </div>`
});
```
7) Before the end of the **`renderMovies`** function, make sure to return the html data after joining it into a single string with **`return movieHtmlArray.join('');`**
    a) This return statement should be *after* the map() loop, not inside of it!
8) Test that your render function works!
    a) Somewhere in your document ready block, set the innerHTML of `movies-container` to the result of **`renderMovies(movieData);`**
    b) Check that the movies show up. You should see a bunch of movies show up in your DOM.

Step 3- Make movies show up whenever you use the search bar

Now that we know we can get movies to show up on screen, we can control exactly *when* that happens. Remember, we're building a movie search tool! We only want movies to show up when you search for them.

1) Add a form submit listener. *The below is an example, you'll need to adjust a few things to get it attached to your search form*:
```
const myForm = document.getElementById('myForm');
myForm.addEventListener('submit', function(e){
    // event listener code goes here
})
```
2) Move your code from part 8 of Step 2 inside of the event listener callback function

3) You'll notice that if you try to do a search right now, the page refreshes! We can stop that with the `Event` parameter's **`preventDefault();`** method.
   a) Make sure the anonymous function in the submit listener has one parameter called **`e`**. This is the `Event` object that all event listeners will receive.
   b) The first line in the anonymous function should be **`e.preventDefault();`**
4) Finally move the code that shows the movies to the screen into the same event handler function.

And that's it! Now, when you refresh the page there won't be any movies. Only when you search for something will you see the movies. *Remember, it's not actually hooked up to the API just yet, so it will always display the same results.*

In part 2, we'll tackle those "add" buttons. When you click on the buttons, the movie id will be saved to LocalStorage and rendered to your "watchlist".