Rev. 08Aug2019

Exercise 52

WORKING WITH APIS

When a user types something like **http://facebook.com** into their browser, the following things happen:

An IP (Internet Protocol) address is a four-part series of numbers, separated by dots. For example, the IP address for facebook.com is 31.13.70.36

1. The browser asks a special server called a *domain name* server for the IP address associated with facebook.com.

2. The browser then sends an HTTP (HyperText Transfer Protocol) *request* to the associated IP address.

3. The Facebook server sends back an HTML *response*.

4. The browser translates the HTML into the browser *Document Object Model* (DOM) and renders the page.

You've been learning about *front-end* programming, working with HTML, CSS, and JavaScript. There is another realm of development, called *back-end* programming, that is responsible for serving up the HTML response (step 3 above).

As a front-end programmer, you don't need to know about how the server does its work. That was not always the case. In the early days of web development, a programmer needed to know how to work in both front-end and back-end environments. Today, they are two entirely separate fields.

Something, though, needs to connect the two. We don't try to load up all of Facebook or Amazon in the browser. We need a mechanism for communicating between front end and back end. That mechanism is the *Application Programming Interface* (API).

Building complete SPAs using the extremely popular technologies, React and Redux, is covered in my course at codingcareer.com.

What is an API? Think of it as a URL that can be called to do some specific things. What kind of things? Rather than try to explain what an API is and how to use it, let's get some actual experience. In this exercise, we'll use an API provided by the Online Movie Database.

DATA- ATTRIBUTES

Begin by looking at `index.html`. The only interesting thing to note is the use of a `data-title` attribute for the movie titles.

Usually when we want to get information from an HTML element, we would read either the `id` or the `class` attributes. As we've seen in previous exercises, that works fine for most things. Sometimes, though, we want information to be available that doesn't fit nicely into the id/class options. In this exercise, I want to attach the movie title to the `<div>` element. Could that be an `id`? Yes, but it's clumsy. It feels like we're using `id` for something other than its intended use.

HTML has a more elegant solution: we can create any custom attribute we wish, so long as it has a `data-` prefix. And we can use as many of these as we want. In addition to `data-title`, we could have `data-genre`, `data-yearReleased` — or anything else we wanted. Once we examine `index.js`, we'll see how these are used.

INDEX.JS

Let's start at the bottom of the page, where we set an event listener on each movie. The event listener is this:

```
let movieInfo = async event => {
  let data = await makeRequest(event.target.dataset.title)
  document.querySelector('#title').innerHTML = data.Title
  document.querySelector('#actors').innerHTML = data.Actors
  document.querySelector('#director').innerHTML = data.Director
  document.querySelector('#year').innerHTML = data.Year
  document.querySelector('#rating').innerHTML = data.imdbRating
  document.querySelector('#plot').innerHTML = data.Plot
  document.querySelector('img').src = data.Poster
}
```

In the second line of code, we see the snippet `event.target.dataset.title`. We just looked at the custom `data-` attributes. Here is how they become accessible to us. All custom `data-` attributes are grouped into the `dataset` variable. From there, individual `data-` components are available: `data-title` as `dataset.title`, `data-genre` as `dataset.genre`, and so forth.