Lesson Weekend

Intermediate JavaScript (/intermediate-javascript) / Asynchrony and APIs (/intermediate-javascript/asynchrony-and-apis) / Introduction to APIs

Text

In this lesson, we'll discuss the general concept of APIs, including what they are and how they work. This lesson is only meant to be a quick overview of APIs. We'll learn a lot more about how they work in future lessons.

Application Programming Interfaces

The term **API** stands for **Application Programming Interface**. In simple terms, APIs are a set of requirements that determine how one application may communicate, or **interface**, with another. APIs can be used with any programming language and are not specific to JavaScript.

At this point, we're very familiar with Web APIs (https://developer.mozilla.org/en-US/docs/Web/API), which are a collection of APIs, each with tools that developers can use in their applications to interact with the browser window, the DOM, events, and other browser structures. These tools are made up of object types, properties, and methods that are built-in to our browser.

Let's consider another example of an API. When you're reading an article online, you might click a *Share on Twitter* button to publish the article to your Twitter feed. Following the requirements of the Twitter API, the news application communicates (or *interfaces*) with Twitter, publishing the article to your feed.

While we won't be using the Twitter API, we'll be working with similar **third-party APIs**. A third-party API (https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Client-side_web_APIs/Third_party_APIs) is provided by a company to use their data or services. Think of MapQuest (https://developer.mapquest.com/), sites with free images like Unsplash (https://unsplash.com/developers), music sites like Spotify (https://developer.spotify.com/documentation/web-api/), or even the US government (https://data.gov/developers/apis/index.html) and NASA (https://api.nasa.gov/). There's *a lot* out there!

Requests and Responses

API calls involve making a **request** to a server and then waiting for a **response** from that server. In the Twitter API example, the request-response process goes like this:

1. A user clicks the *Share on Twitter* button.
2. Then the news application makes a request to Twitter's API with all the information needed to add a tweet of the article to the user's account.
3. Finally, Twitter's API will respond with a message. In this kind of situation, the message usually just states whether the request was successful or not. If it is, the news application will inform the user. If not, the application will tell the user that there was an error and they should try again.

This is unlike the process of using Web APIs, which are built into our browsers. With Web APIs, we simply need to know which object (or property) we're targeting, and make sure that we access it properly (like using the right syntax when we use it).

On the other hand, with third-party APIs we always need to make a request to a server for information and wait for a response.

Header and Body

An API response has several parts including the header and the body:

- The **header** contains information such as the date, content type, any authorization information, and so on.
- Meanwhile, the **body** contains any messages from the API, a status code, plus data we've requested.

In this section, we will mostly focus on working with the body of an API response.

Status Codes

All API responses come with a status code that lets us know the status of the API call. These codes are used in the browser as well and some will be familiar to you already. There are a *lot* of different status codes but there are only a few that we really need to be familiar with. Don't worry about trying to memorize these now. Just look them over so you have some general familiarity. You can always look up status codes as needed later.

- `200` : The API call was successful! This is also known as 200 OK.
- `400` : Bad request. We'd better check the API request and make sure it's correct.
- `401` : Unauthorized. We aren't authorized to access the resource, which might mean we haven't logged in correctly.
- `403` : Forbidden. We aren't allowed to access that content.
- `404` : Not found. The resource couldn't be found.
- `500` : Internal server error. Not our fault! Something is going on with the API. In general, if we get any `500` errors, the server is having problems.

For a full list of status codes, see Wikipedia's List of HTTP status codes (https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

API Calls are Asynchronous

The entire process of making an API call is asynchronous. Even though the user clicks a button *now*, they won't get a result back until *later* even if the call only takes a fraction of a second to process.

It should be clear why API calls need to be asynchronous. It takes time to make a request and receive a response — and we don't want the user's browser to freeze up when that happens. That also means that the code itself is asynchronous as well so we need to handle it with special code to ensure that we don't handle API requests until *after* they are complete.

GET and POST requests

There are several different kinds of API requests. The most common are **GET** and **POST** requests. A GET request gets information. For instance, if we wanted to display all tweets mentioning the *@Epicodus* Twitter account, we could send the Twitter API a request asking for this information. The Twitter API will respond with a list of tweets mentioning *@Epicodus*.

In our earlier Twitter API example when we posted the news article to our Twitter feed, the news application makes a **POST** request because information needs to be *added* to Twitter's server. The news application isn't asking to receive some information — it's asking for something to be changed.

We will be focused on making GET requests in this section. Only GET requests will be required for this section's independent project. When you study a backend language such as Ruby or C#, you will learn about making other kinds of requests, like POST requests.

Before we actually start writing JavaScript code to write API calls, we'll do the following:

- Get an API key for the OpenWeather Map API and explore its API documentation
- Download Postman, a tool for making API calls, and learn the basics of making an API call
- Practice parsing the JSON body of an API call

Only then will we be ready to start making API calls in our JavaScript code.

Previous (/intermediate-javascript/asynchrony-and-apis/asynchrony-in-javascript)
Next (/intermediate-javascript/asynchrony-and-apis/api-documentation-and-keys)

Lesson 3 of 33
Last updated more than 3 months ago.

disable dark mode

(http://www.epicodus.com)

© 2023 Epicodus (http://www.epicodus.com/), Inc.