**What exactly IS an API?**

Have you ever heard the word “API” batted about, and wondered exactly what an this is? Do you have a vague idea, but want to know more about what you might do with an API, and how you might build one?

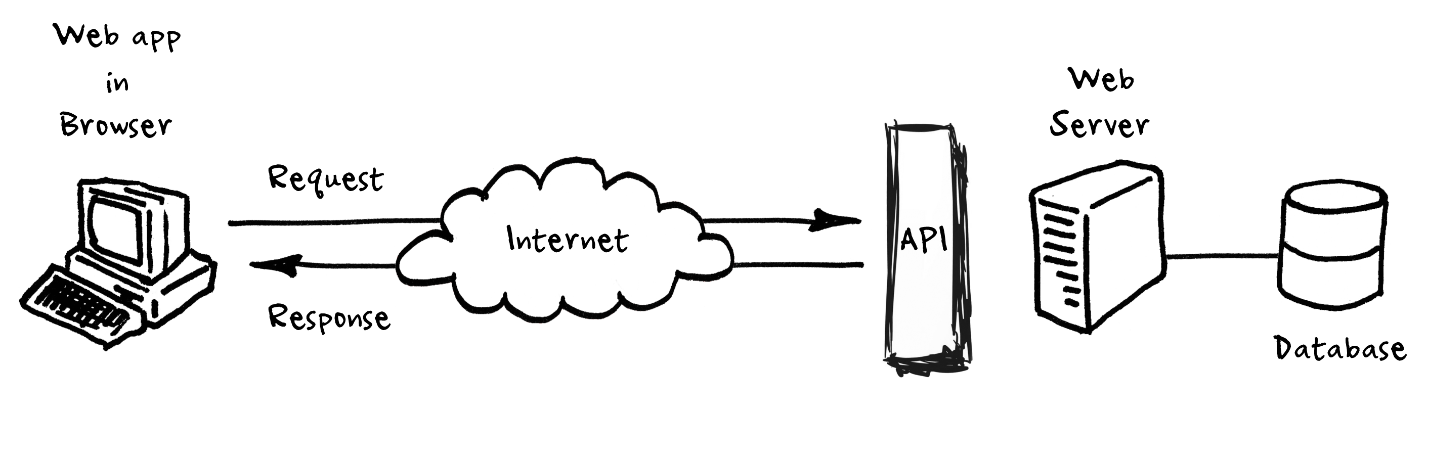
Don’t fret. I can help!

Let’s get down to it and pick apart what an API is, and how building an API is similar (or different!) than building other types of web apps and websites. Here’s a short primer on what an API is, what makes it different, and what you would need to build one yourself.

**Application Programming Interface (API)**

In basic terms, APIs just allow applications to communicate with one another.

When people speak of “an API”, they sometimes generalize and actually mean “a publicly available web-based API that returns data, likely in JSON or XML”. The API is not the database or even the server, it is the code that governs the *access point(s)* for the server.



An API is not a database. It is an access point to an app that can access a database.

In this post, we will focus on these specific kinds of APIs — web based APIs that return data in response to a request made by a client.

They allow us to go get data from outside sources.

1. We can send an API a request detailing the information we want.
2. APIs allow our sites to alter data on other applications, too. For instance, you’ve probably seen “Share on Facebook” or “Share on Twitter” buttons on miscellaneous websites. When/if you click one of these buttons, the site you’re visiting can communicate with your Facebook or Twitter account, and alter its data by adding new status or tweet.

**But why would we need an API?**

Imagine the following scenario: You (as in, your application, or your client, this could be a web browser) wants to access another app’s data or functionality. For example, perhaps you want to access all Twitter tweets that mention the *#epicodus* hashtag.

You could email Twitter and ask for a spreadsheet of all these tweets. But then you’d have to find a way to import that spreadsheet into your application; and, even if you stored them in a database, as we have been, the data would become outdated *very* quickly. It would be impossible to keep it up to date.

It would be better and simpler for Twitter to provide you a way to query their application to get that data, so that you can view or use it in your own application. It would stay up to date automatically that way.

An API brokers access to a different application to provide functionality or access to data, so data can be included in different applications.

**Who Creates Public, Web-Based APIs?**

Large tech companies, especially social media companies frequently make their aggregate data available to the public, but APIs are also maintained by government organizations, conferences, publishing houses, software startups, fan groups, eSports leagues and even individuals, in order to share anything from social media content to trivia questions, rankings, maps, song lyrics, recipes, parts lists and more.

In short, any person or organization that collects data might have an interest in making that data available for use by a different app. Maybe you have an API you’d like to build and make available?

**How an API Differs from a “Regular” Database-Backed Project**

An API is different from a database backed (or static) web application or site in that it does not generally need to contain a frontend — no HTML, CSS is necessary to be shown to the user via static pages or dynamically generated templates that fuse data with reusable layouts.

Requests to retrieve or write data are generally done without a frontend, by sending an HTTP request to a server.

If you are familiar with building server side apps with Java, Ruby, PHP, JS via node or other language, you will likely recognize the need for certain *routes:* URLs that trigger route handlers to issue certain actions when they receive a request with some data. APIs have routes the same way a different site might, they just generally do not serve templates on those routes.

**So What is JSON and why do we use it?**

JSON stands for **J**ava**S**cript **O**bject **N**otation and is basically a way of representing data that looks like JavaScript objects.

Let’s look at a very typical JavaScript Object for a Restaurant, which might look a bit like this:

“restaurant”: {  
 “name”: “Fish Witch”,  
 “address”: “214 NE Broadway”,  
 “zipcode”: “97232”,  
 “phone”: “503–000–0000”,  
 “website”: “<http://fishwitch.com>",  
 “email”: “[hellofishy@fishwitch.com](mailto:hellofishy@fishwitch.com)”  
}

Neat. This is fairly easy to read — our data is stored as key/value pairs. This means that we can see the **key** on the left, and the **value** on the right. The key stays the same for each Restaurant object, but the value would be different. A different Restaurant would have a different address, but its properties would be the same — it would always have a name, address, zip, phone and email.

JSON is *everywhere* in modern web applications. It’s readable, it’s lightweight, and it works super well with applications written in JavaScript, as it *is* JavaScript. But is also comparatively easy to get applications written in other languages to read it and generate it as well — including Java. This means that an API that returns JSON can be accessed by an application written in Java, Ruby, Python, JS, PHP and many more. This makes an API *scalable* and platform independent. Aha! Scalable! Platform Independent! Good words, powerful words, $$ words.

**I want to build an API!**

Awesome! Building an API is totally within reach of any developer who knows a little bit about server side web application programming and routing, such as with Java Spark, JavaScript Express, ASP.net for C#, Ruby on Rails and so on.



Example of a simple API built in Java Spark using the GSON library

In order to build a public API, you’ll need the following:

* A backend with routing of some sort as mentioned above
* A database where your application can store its data. This could be a database server you are running, such as MySQL or Postgres, or it could be a BaaS (backend as a service) DB such as Firebase.
* A server, likely a VPS (Virtual Private Server) that is accessible to the internet, where your application can run. (You might look into DigitalOcean, Amazon Web Services or Microsoft Azure for this)

**I want to tie an API into my App!**

So cool. APIs can really extend your App’s functionality in major ways. Adding a map, weather data or other information can be incredibly useful.

* Find a public API that offers the kind of information you want to access. Make sure it is well documented and maintained. Some sites that offer many APIS are [ProgrammableWeb](https://www.programmableweb.com/), [AnyApi](https://any-api.com/), and [public.apis.zone](https://public.apis.zone/)
* Read through the documentation to make sure the API is workable for you. Many APIs require you to register for an API key to access their data, or they have authentication flows your app needs to go through before you can access information.
* Understand the shape of the data available. Go ahead and check out the Postman tool if you haven’t already, it’s an awesome piece of software that makes many aspects of working with APIs much easier. It helps you send test requests to APIs so you can experiment with the data you receive.
* Call the API from your app and process the data that you receive, displaying it in your app! This is the most exciting part. Once you are successfully able to query the API with a tool like [Postman](https://www.getpostman.com/), you can write the code that integrates the data the API returns into your own work and display it to your users.