https://github.com/LearningNerd/intro-apis-workshop/

Intro to APIs (Application Programming Interfaces)

You've certainly heard the term ***API*** thrown around a lot -- not just among techies, but also in the news and sometimes even in casual conversation! So what's this all about? ***API*** stands for ***Application Programming Interface***. In this section, we'll look at examples of interfaces and then examples of *application programming* interfaces to get a better sense of the overall picture.

What is an interface?

Application Programming Interfaces (APIs) are a pretty broad category of things -- as are interfaces in general! The term "API" is used to mean lots of different things, which definitely adds to the confusion. So let's first zoom out and look at what ***interfaces*** are in general:

⭐***An interface allows two different entities (people, computers, software programs, etc) to interact or communicate.*** It's the *connection* and the *set of rules or conventions* that allow the two entities to communicate.

Some examples of interfaces:

- **User interfaces** allow humans to interact with anything from toasters to cars to computer programs to space ships! There are **physical** user interfaces like a computer keyboard, or general things like knobs and buttons and switches. There are also abstract **software** user interfaces, like a command line interface versus a graphical user interface that we use to move files around on our computers.
- **Hardware interfaces** are the mechanical/electrical components that allow different pieces of a machine or a computer to interact with each other -- for example, the wires that connect a computer's hard drive to its motherboard, or the HDMI cable you might use to connect a computer to a TV.
- **Application binary interfaces** allow code for computer applications to communicate with the operating system and the 1s and 0s, the machine code that the computer understands!
- And finally, an **application programming interface** or **API** is a higher-level software interface that allows different pieces of software to communicate or interact with each other. APIs can take many forms -- there are APIs for everything from web apps and web browsers to operating systems and everything in between!

Curious about interfaces in general? I borrowed most of the above info directly from Wikipedia's page on interfaces, which is a really interesting read: https://en.wikipedia.org/wiki/Interface_(computing)

https://github.com/LearningNerd/intro-apis-workshop/

What is an Application Programming Interface (API)?

Just as a graphical user interface (GUI) allows users to interact with software programs, ***an API allows one software program to interact with another.***

Examples:

- Microsoft maintains a Windows API for their operating system, to make it easier for developers to build applications that run on Windows!

- **Web browser APIs** are what allow us to use the JavaScript language to interact with a web browser! We've already been using the **Document Object Model API** to let our JavaScript interact with a web page's HTML or CSS, using built-in API functions like `document.getElementById()`.
- The Console API is another built-in web browser API, which lets us interact with the web browser's console -- like our favorite `console.log()` function!

- The Web Audio API lets us use JavaScript to create and play sounds (or music) right within the web browser.

And there are *many* other built-in web browser APIs! Take a look through this list from MDN: https://developer.mozilla.org/en-US/docs/WebAPI

- **Third party web APIs** are probably the type of API that you'll hear about the most. These are APIs created by *other companies* that allow your program to communicate with their program *over the internet*. You may have heard of Twitter's API, YouTube's API and plenty of others!

Note: The term ***"third party"*** means that it was created by some other company or organization. If you created *your own* API that lets your own programs talk to each other over the internet, we would still call that a web API, just not a *third party* API. (And it is common for companies to build APIs for themselves!)

https://github.com/LearningNerd/intro-apis-workshop/

The basics of HTTP requests and responses

⭐***Accessing an API over the internet is essentially the same thing as going to a website.*** There are only a couple differences:

- Instead of typing a URL into your web browser to access a website, you'd access an API from *within your code* (like inside your JavaScript file) or via the *command line* directly.
- Compared to visiting a website as a *user*, accessing an API as a *developer* does require you to read some documentation beforehand to learn each API's unique rules and conventions. (So it's not quite as easy as just typing in a URL.)

Aside from learning the rules for each specific API you're working with, you'll also need a bit of background knowledge about how information is sent over the internet:

HTTP and the request-response model

HTTP stands for ***HyperText Transfer Protocol***, and it's the set of rules that allow computers to communicate with each other over the internet -- it's the backbone of the World Wide Web!

Fun fact: HTTP was initially created by Tim Berners-Lee, inventor of the World Wide Web, in 1989 at CERN.

Bonus fun fact: CERN is the home of the Large Hadron Collider, the largest single machine in the world, which smashes atoms and even smaller particles together at crazy high speeds in the name of science!

Since HTTP is already so widely adopted (used by practically every computer in the world!), it's very convenient for programmers to build on top of it! That's why so many APIs are based on HTTP.

Any time your computer (the ***client***) visits a website, it makes a ***request*** via HTTP, which the web browser sends over the internet to a ***server***. Then the server (which is just another computer) will receive that request, run some code to decide what to send back, and finally it went send a ***response*** back to the client.

We call this the ***request-response model***, and you can think of it like communicating via snail mail or telegram:

Very simple, isn't it? It's an old-fashioned idea, but it works because our computers and internet connections are fast enough that browsing the web feels *nothing* like communicating via telegram. :)

https://github.com/LearningNerd/intro-apis-workshop/

The four parts of an HTTP Request

The following four things are sent by the client (your computer) every time you visit a web page (in other words, every time you make an HTTP request to some server on the internet):

1. The **URL** (**U**niform **R**esource **L**ocator), like https://example.com -- this is the unique address for the data or web service you're interacting with.
2. The **HTTP request method** (also called the *HTTP verb*), which specifies what type of action is being requested for the server to perform. (See examples in the table below.)
3. The **headers**, which contain important *meta-information* about the request -- for example, what type of data format the client will accept (examples: plain text, JSON, XML or others), whether the client is a mobile phone or a desktop computer, and lots more!
4. The **body of the request**, which contains any *actual data* that needs to be sent to the server. For example, if you create a new account on Twitter, the body of the request would include your username, email, and password.

The four most common HTTP request methods (HTTP verbs)

1. **GET** - Request some data (ex: to view any web page, or to search for something on Google)
2. **POST** - Submit data to the server, usually to create something new (ex: create a new project on Glitch or GitHub)
3. **PUT** - Replace existing data with new data (ex: update your relationship status on Facebook)
4. **DELTE** - Remove a piece of data entirely (ex: delete an embarrassing video that you had uploaded to YouTube)

The **GET** and **POST** request methods are the most common by far. Most of the time when you're browsing the web, your web browser is just sending a bunch of GET requests. When you submit a form (especially with any private information like a password, bank account info, etc), that's usually sent as a POST request.

The three parts of an HTTP response

As we saw above, any time you visit a website or make an API call, you're sending a request to the web server. Then each time the web server receives a request, it sends a ***response*** to the client. The response contains three pieces of information:

1. A **status code** (like 200 or 404), which indicates whether the HTTP request has been successfully completed -- or if not, what type of error occurred. (Wikipedia has a great list of all the common HTTP status codes.)

2. The **headers**, which contain important *meta-information* about the response -- just like headers in the request, except with different meta-information like what data format the server used to package the information sent in the response.

3. The **body of the response**, which contains any *actual data* sent from the server back to the client. For example, if you send a request to Google with a search term, Google's servers will send a list of search results included in the *body* of the response.

https://github.com/LearningNerd/intro-apis-workshop/

Example HTTP requests and responses

Example 1: You go to https://example.com by typing that URL into your web browser.

The request contains:

1. The URL: https://example.com
2. The HTTP request method: GET
3. The headers probably include things similar to: "Accept: text/html" and "Accept-Language: en-us" and "User-Agent: Mozilla/5.0", etc
4. The body of a GET request is usually empty

The response contains:

1. If it worked correctly, the server would send the status code `200`. (See list of status codes.)
2. The response headers would likely include "Content-type: text/html" and some other meta-information
3. The body of the response would contain all the HTML code for the homepage of example.com -- this is what your browser downloads every time you visit a website!

Example 2: You access GitHub's API via the command line to create a new repository on your GitHub account.

The request contains:

1. The URL: https://api.github.com/user/repos
2. The HTTP request method: POST
3. The headers would include "Authorization: token YOUR-TOKEN-HERE" to authenticate the user (proving you are who you say you are!)
4. The body of the request: The name of your new repository formatted as JSON data: `{"name": "my-test-repo"}`

The response contains:

1. Again, if it worked correctly, the server would send the status code `200`.
2. The response headers would likely include "Content-type: application/json" because GitHub's programmers decided to use JSON as their primary data format for their API.
3. The body of the response would contain JSON data about the repository that was just created -- its name, the URL to the repository's page, etc.

https://github.com/LearningNerd/intro-apis-workshop/

Authentication, API keys and rate limits for third-party web APIs

Accessing a third-party web API can be as simple as sending an HTTP request to a URL like in the examples above. (See section 5.5: intro to API challenges for a step-by-step walkthrough of accessing an API!)

But many API providers have extra rules for using their APIs, which we'll take a quick look at below:

Authentication

To ***authenticate*** a user means to verify that they are who they say they are. Authentication is an important step in accessing private information or acting on behalf of a user via an API.

For example: you can create a new GitHub repo through GitHub's API, but first you have to authenticate, either by providing your username and password, or by providing an ***access token*** that's like a temporary password issued only to you.

There's a *lot* more to learn about authentication; see the extra resources at the top of this page if you're curious about more of the details!

API rate limits

Many API providers will *limit* how often you can use their API. Why? Here are two main reasons:

- Remember, sending things over the web uses up bandwidth, which costs money! API providers will create rate limits to help keep their expenses under control.
- Even more importantly, rate limits also help to prevent abuse of the API -- for example, by overloading the API provider's servers with what's called a denial-of-service (DoS) attack.

For example, GitHub's API limits you to *60 requests per hour* if your requests are made anonymously. (If you associate your requests with a GitHub account, then you get 5000 requests per hour!)

See GitHub's API documentation for more on their rate limiting rules: https://developer.github.com/v3/#rate-limiting

Another example: Google Maps provides an API for calculating the distance between two places, which is free up to a point, and then they start charging you money after that. You can see all the details here: https://developers.google.com/maps/documentation/distance-matrix/usage-limits

https://github.com/LearningNerd/intro-apis-workshop/

API keys

An ***API key*** is a unique ID assigned to your application.

Many APIs don't require authentication -- for example, if all of their data is already publicly available information (like an API that provides a weather report). But many companies that provide APIs still want to track who's using their API, both to prevent abuse/illegal activities and to collect data on who uses their API.

To use the API for Twitter, Facebook, and many other web services, you to first have to make an account on their website, register your own web app with them with a description of what it does, and then finally they'll issue you an API key. That way, they'll know who you are whenever you use their API within your own programs.