2020-04-11

APPLICATION PROGRAMMING INTERFACES

A Introduction to Interacting with Systems on the Web

ABOUT ME

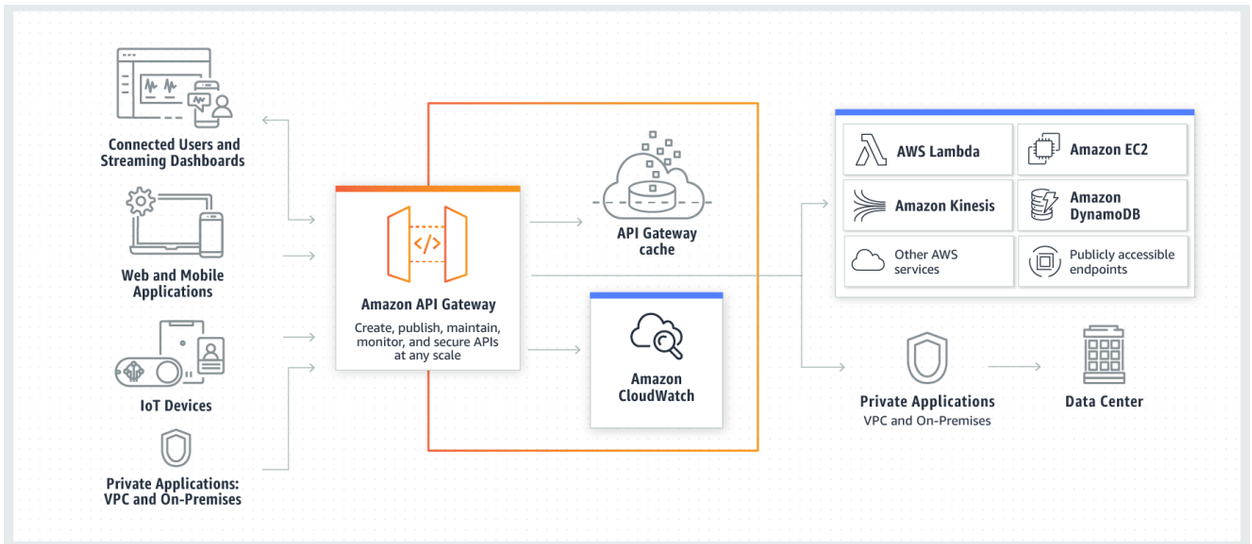
- Matthew Barlowe
- Application Developer, National Hockey League
- Operations Intelligence Analyst, McKesson

SO WHAT IS AN API?

WHAT IS AN API?

- **Definition:** ***a computing interface to a software component or a system, that defines how other components or systems can use it.***

this definition says that an API is just a way for a computer, or piece of software on that computer(an app or a script) to communicate, or interface, with another computer or piece of software. Today if you mention API without any other context people take this to be a system of communicating between things over HTTP or “the web”, but it has a larger context than that as well and can be used to refer to any means of interacting with a piece of software. ggplot and matplotlib are APIs for graphing just as pandas and dplyr are apis for working with data. Generally APIs are for software developers to use to build other software and aren’t generally available for use to the broader public



Example of AWS API Service

You can see in this diagram the basic structure of web APIs. On the left you have users or devices that either need data or compute from the things on the right. And the way the items on the left get what they need is to go through the API which is represented by the orange block. So think of an API as a technical middle man that brokers requests between businesses.

WHY USE AN API?

WHY USE AN API?

- **Ease of Use for 3rd Parties**
- **Scalability**
- **Accessibility**
- **Encapsulation**

1. Ease of Use
Used to be if you needed to get, change or update data in a database you needed to be able to login and run the commands needed on the system itself. This meant creating users and dealing with permissions and a whole bunch of overhead. But with an api you only need one user, the API itself, to do the things needed on the system and now anybody wanting to interact with your system just needs the web address of the API. There are serious security implications with this model though so be careful.
2. Scalability
Since APIs are generally low bandwidth users as their responses are generally plain text this makes it easier to scale hardware to demand as opposed to delivering other high bandwidth media.
3. Accessibility
Because of their text over HTTP nature, web APIs can be used by almost any programming language out there no matter what language the API itself is written in. Python, R, Ruby, Java, C++ and the list goes on and on. This allows developers to be flexible when building around the data and use the tools they feel best for the job instead of having a monolithic code base which could hamper other areas of the business.
4.Encapsulation
Encapsulation is an Computer Science term that refers to binding together methods of dealing with data together into one object and hiding details end users don't need to see. Using an API allows you to only expose the parts of your data or app over the web that you want your end users to see and walls off the rest. Similar to a car there are defined inputs the user to can use to start and drive it so and API would be analogous to the key and the steering wheel. With the very important caveat "if done properly". APIs can still be a security issue if done improperly

WHAT DOES AN API RETURN?

WHAT DOES AN API RETURN

- **XML**
- **JSON**

Technically an API could return anything, but generally it responses come in one of two categories: XML or Extensible Markup Language looks very similar to HTML but they are actually quite different. Both are Markup languages, but in HTML the tags always have the same meanings while in XML one can define the tags and their relationship to each other. If that doesn't make a lot of sense don't worry because XML doesn't make a lot of sense to a lot of people and is being phased out for JSON or JavaScript Object Notation for many reasons such as its easier for computers AND humans to read, some of the most popular languages such as JavaScript and Python natively support it, and its more lightweight than XML. A lot of systems still use XML though so if you really start getting into working with APIs be prepared.

```xml
<mediawiki xml:lang="en">
  <page>
    <title>Page title</title>
    <!-- page namespace code -->
    <ns>0</ns>
    <id>2</id>
    <!-- If page is a redirection, element "redirect" contains title of the page redirect to -->
    <redirect title="Redirect page title" />
    <restrictions>edit=sysop:move=sysop</restrictions>
    <revision>
      <timestamp>2001-01-15T13:15:00Z</timestamp>
      <contributor>
        <username>Foobar</username>
        <id>65536</id>
      </contributor>
      <comment>I have just one thing to say!</comment>
      <text>A bunch of [[text]] here.</text>
      <minor />
    </revision>
    <revision>
      <timestamp>2001-01-15T13:10:27Z</timestamp>
      <contributor><ip>10.0.0.2</ip></contributor>
      <comment>new!</comment>
      <text>An earlier [[revision]].</text>
    </revision>
    <revision>
      <!-- deleted revision example -->
      <id>4557485</id>
      <parentid>1243372</parentid>
      <timestamp>2010-06-24T02:40:22Z</timestamp>
      <contributor deleted="deleted" />
```

Example of XML from Wikipedia

Here’s an example of XML from one of the larger API’s still using it Wikipedia. As you can see you have the opening and closing tags similar to HTML if you are familiar with that but each one has been given a name by the developer in the template instead of having standard tag names like HTML. You would parse this similar to how you would scraping HTML by iterating over the nodes to get to what you need unfortunately (or fortunately depending on who you ask) we won’t be covering parsing XML in this talk

```
[
  {
    "ast": 3.7,
    "blk": 0.6,
    "dreb": 5.8,
    "fga": 12.6,
    "fgm": 5.4,
    "fta": 3.5,
    "ftm": 2.4,
    "gp": 58,
    "mins": 33.0,
    "oreb": 1.8,
    "pf": 2.1,
    "player_id": 203932,
    "player_name": "Aaron Gordon",
    "plus_minus": -1.4,
    "points": 14.4,
    "position": "Forward",
    "season": 2020,
    "stl": 0.9,
    "teams": "ORL",
    "tov": 1.6,
    "tpa": 3.9,
    "tpm": 1.2
  },
  {
    "ast": 3.3,
    "blk": 0.2,
    "dreb": 1.9,
    "fga": 8.6,
    "fgm": 3.5,
    "fta": 1.2,
    "ftm": 1.1,
    "gp": 58,
    "mins": 23.6,
    "oreb": 0.4,
    "pf": 1.8,
    "player_id": 1628988,
    "player_name": "Aaron Holiday",
    "plus_minus": 1.8,
    "points": 9.4,
    "position": "Guard",
    "season": 2020,
    "stl": 0.8,
    "teams": "IND",
    "tov": 1.3,
    "tpa": 3.5,
    "tpm": 1.4
  }
```


JSON Example

Here we have an example of a JSON from my own created API for my website which we will be working with shortly. If you have worked in python at all you will note it is extremely similar to a dictionary and in fact once you read it in you can access the key value pairs with the same syntax you would a python dictionary.

ACCESSING THE API

ACCESSING THE API

- **Base URL**
- **The Path**
- **Parameters**

An API url is very similar to your every day web site addresses. There is a base url, often called the root end point that would be very similar to something like www.google.com. They normally won't start with www but are often a subdomain of the original domain. Like my API is stats.theseventhman.net while my website is www.theseventhman.net. Github has api.github.com versus github.com and Twitter has api.twitter.com instead of the normal twitter.com. The next part is the path this could be a variety of names or version numbers that tells the API what resources you want to access that often corresponds with certain tables in the database. The last part is the parameters. These are variables you pass to the API to filter the data you want to be returned such as only want posts from one user or all posts after a certain date.

ANATOMY OF AN API URL

- **Root: https://stats.theseventhman.net/**
- **Path: stats/api/v1/players/**
- **Parameters: ?agg=no&player=&season=2020&team=&toc=**
- **Use '+' to pass more than one filter i.e. &season=2020+2019**

Here's an example of each component using an URL from my own API. You see the root as I mention before which is stats.theseventhman.net. The path is defined by me to help organize things. stats lets me know its about counting stats. v1 is the version of the api that is currently in production and players lets me know the stats that will be returned deals with players versus teams. Next we have the parameters. Parameters will always be passed with a question mark to start this lets whatever is parsing the url know these should be passed a different way. In this example we have five parameters agg, or aggregate, this tells the API to aggregate season. Player which you can pass a player id to filter the data by player, season which you can see is being passed the year 2020, team for filtering by team, and toc (short for time on court) to filter players by playing time. Each parameter will be separated by an ampersand usually although semicolons can also be used. You can pass the more than one filter using the '+' sign this allows the server to separate each of the words into an array. So in this example if i wanted stats from from both the 2020 season and the 2019 season i would connect them with the plus sign as shown in the slide.

THE CODE

- Using my own API
- Docs for my API on my GitHub page https://github.com/mcbarlowe/seventhmanapi
- Completely open source

TIME TO CODE

NOT ALL API'S ARE PUBLIC

- Use web browser developer tools
- look for JSON/XML type or xhr/fetch cause
- copy as cURL

A lot of API's are not as public or as open as mine is. You'll have to dig into the guts of a webpage to find them using the Dev tools of your browser. You should be able to access them with control alt i for windows or command alt i for macs for both chrome or firefox. If you use another browser you might have to look up the command to open them up on the page you need to get data from. From there you'll need to proceed to the network tab and and refresh the page and you'll see all the resources the web page pulls in to display the page you are looking at. Usually data that you want can be identified in three ways: It will have the largest load time, the data type will be JSON or XML or the cause of pulling it in will be xhr (XMLHttpRequest) or fetch. Some non fully public will need headers passed to them you can check and see what these headers are by right clicking on the link for the data and selecting copy as cURL.

NOT ALL API'S ARE PUBLIC

- curl 'https://stats.theseventhman.net/stats/api/v1/players/distinct/' -H 'User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10.15; rv:74.0) Gecko/20100101 Firefox/74.0' -H 'Accept: */*' -H 'Accept-Language: en-US,en;q=0.5' --compressed -H 'Referer: https://theseventhman.net/' -H 'Origin: https://theseventhman.net' -H 'Connection: keep-alive' -H 'Cache-Control: max-age=0' -H 'TE: Trailers'
- everything with "-H" is a header to pass

QUESTIONS?

MY OTHER PROJECTS

- **theseventhman.net**
- **nba_scraper**
- **nba_parser**

SOURCES

- Application Programming Interfaces, Wikipedia.
- Amazon API Diagram, AWS
- What is an API and Why Should I Use One?, Tyler Elliot Bettilyon, Medium
- Has JSON Overtaken XML, Bhuvanesh Mohankumar, C# Corner
- A Technical Introduction to XML, xml.com
- Understanding and Using REST APIs, Zell Liew, Smashing Magazine
- Query String, Wikipedia
- Accessing Web Data (JSON) in R using httr, Akshay Mahale, Data Science Plus
- What is an API? In English Please, Petr Gazarov, FreeCodeCamp
- What Exactly does OOP Encapsulation Encapsulate, Andrew Koenig-Bautista, GitConnected Medium