APIs Introduction for Beginners

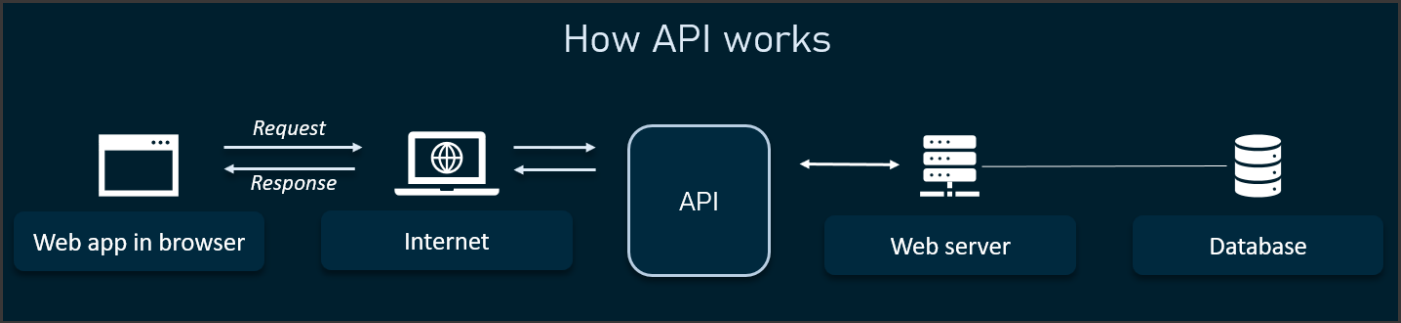
- Durgesh A. Mahajan
Amalner, Jalgaon, Maharashtra
2nd year IT Engineer @DYPCOE Pune

Index

https://www.linkedin.com/in/durgesh-mahajan-99bab0212/ durgeshmahajan1722@gmail.com @durgeshm01722
https://github.com/durgeshm01722

Overview

APIs (Application Programming Interfaces) are software programs that give developers access to computing resources and data. Companies from many different fields offer publicly available APIs so that developers can integrate specialized tools, services, or libraries with their own applications and codebase.



APIs - What and Why

As mentioned earlier, an API (Application Programming Interface) is a software program that gives developers access to computing resources and data. APIs adhere to specific rules and methods to clearly communicate requests and responses.

The ability to access data and computing resources greatly increases a developer's efficiency. It is much easier to use an API than to build every single program, method, or dataset from scratch. APIs are built on the principle of abstraction - you don't need to understand the inner workings or complexities of an API to use it in your own environment.

APIs are built with the developer in mind and often times do not offer a graphical user interface (GUI). However, there are exceptions to this standard.

Cloud APIs

Google offers APIs that can be applied to many different fields and sectors. APIs are often used in web development, machine learning, data science, and system administration workflows. However, these are only a handful of use cases. If you explore AnyAPI, for example, you will start to see just how many APIs are available.

API Architecture

APIs are a set of methods that allow programs to communicate with one another. To communicate effectively, programs need to adhere to a clear protocol that governs the transfer and interpretation of data.

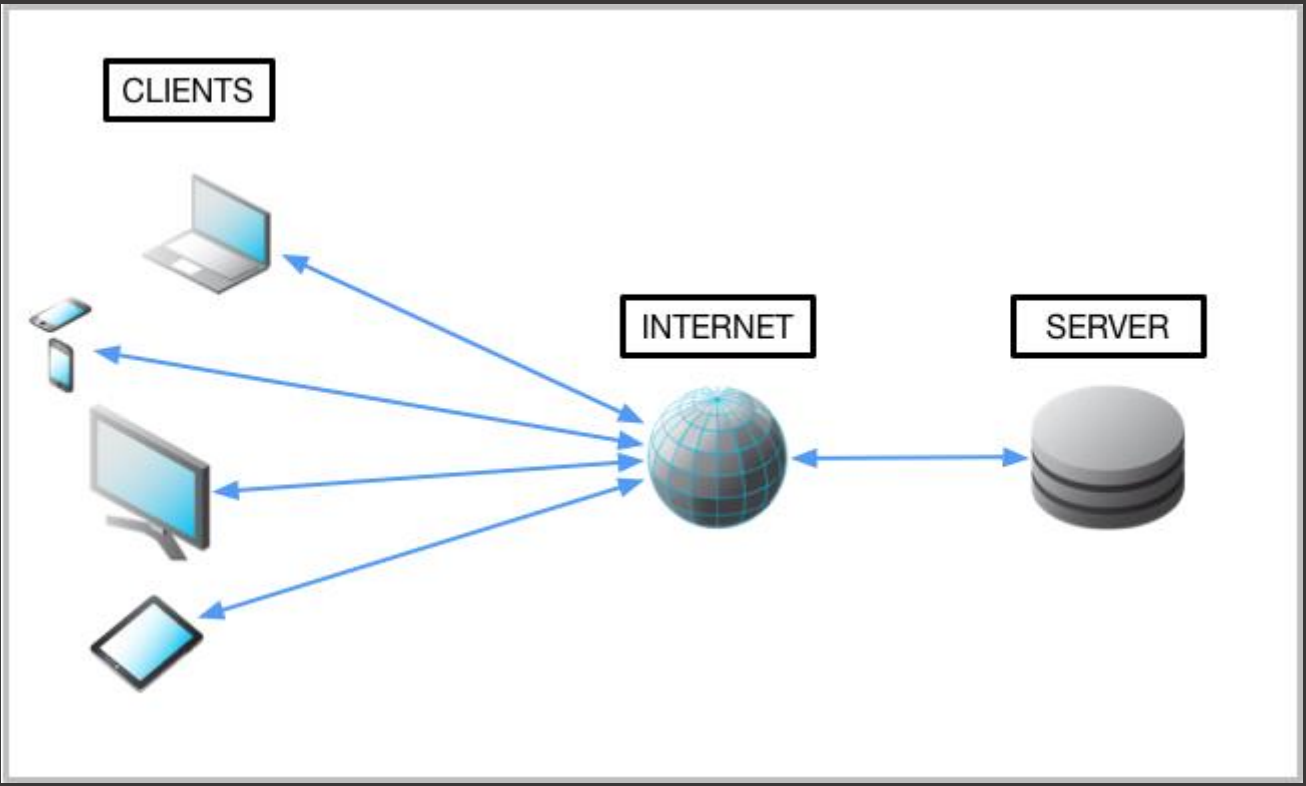
Client-server model

The internet is the standard communication channel that APIs use to transmit requests and responses between programs. The client-server model is the underlying architecture that web-based APIs use for exchanging information.

The client is a computing device (e.g. a smartphone, laptop, etc.) that makes a request for some computing resource or data. The client's request needs to be formatted in the agreed upon protocol.

The server has data and/or computing resources stored on it. Its job is to interpret and fulfill a client's request.

The following is a visual representation of the client-server model:



HTTP protocol and request methods

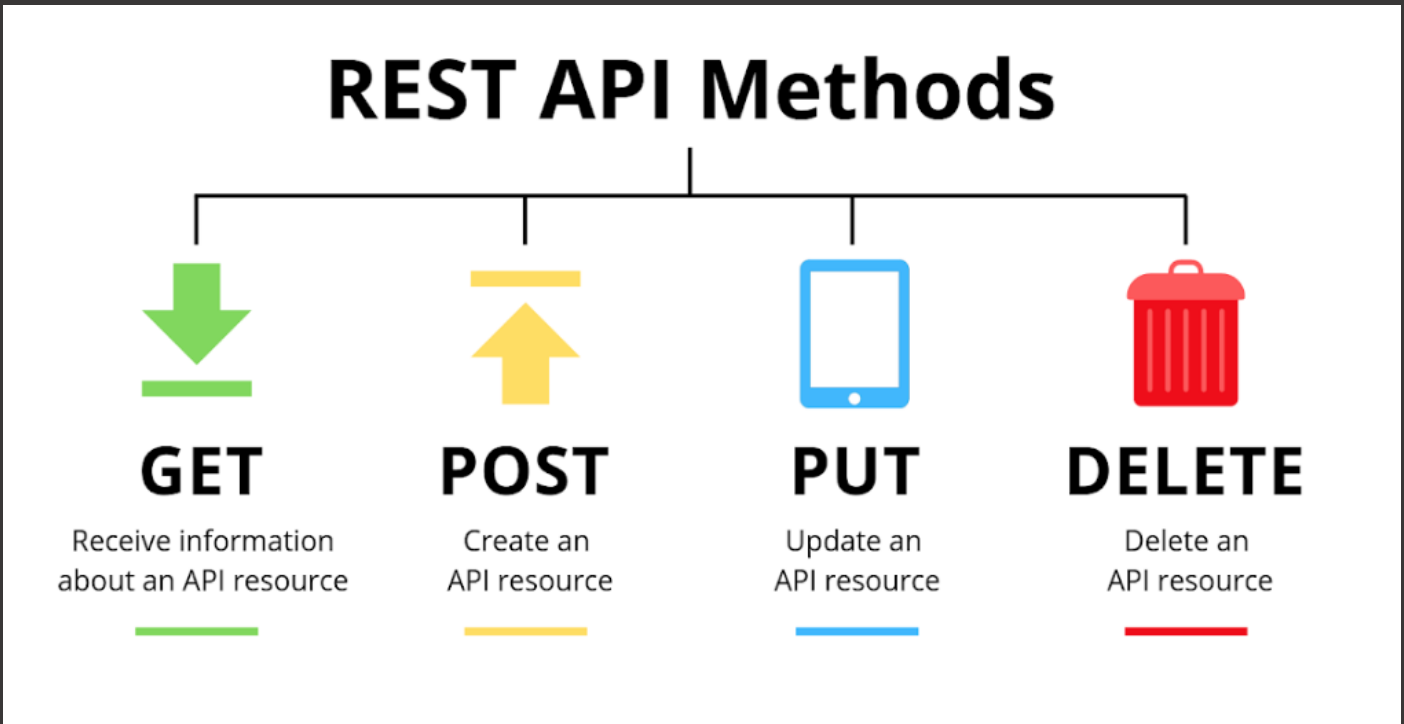
Since APIs use the web as a communication channel, many of them adhere to the HTTP protocol, which specifies rules and methods for data exchange between clients and servers over the internet. The HTTP protocol is not only used by APIs - it is the standard for web communication where data is sent and received over the internet.

APIs that utilize the HTTP protocol use HTTP request methods (also known as "HTTP verbs") for transmitting client requests to servers. The most commonly used HTTP request methods are GET, POST, PUT, and DELETE.

- The **GET** request method is used by a client to fetch data from a server. If the requested resource is found on the server, it will then be sent back to the client.
- The **PUT** method replaces existing data or creates data if it does not exist. If you use PUT many times, it will have no effect - there will only be one copy of the dataset on the server.
- The **POST** method is used primarily to create new resources. Using POST many times will add data in multiple places on the server. It is recommended to use PUT to update resources and POST to create new resources.
- The **DELETE** method will remove data or resources specified by the client on a server.

Method	Function/Description
GET	Retrieve information about the REST API resource
POST	Create a REST API resource
PUT	Update a REST API resource
DELETE	Delete a REST API resource or related component

Although there are hundreds of APIs out there, all with their own unique purposes and specializations, it's important to realize that at the end of the day they all use the same protocol and underlying methods for client-server communication.



Endpoints

APIs use HTTP methods to interact with data or computing services hosted on a server. These methods are useless if there isn't a way to access specific resources with consistency. APIs utilize communication channels called endpoints so that clients can access the resources they need without complication or irregularity.

Endpoints are access points to data or computing resources hosted on a server and they take the form of an HTTP URI. Endpoints are added to an API's base URL (e.g. http://example.com) to create a path to a specific resource or container of resources. **The following are some examples of endpoints:**

http://example.com/storelocations
http://example.com/accounts
http://example.com/employees

The following are also valid endpoints:

http://example.com/storelocations/sanfrancisco
http://example.com/storelocations/newdelhi
http://example.com/storelocations/london

You can add query strings to endpoints (e.g. http://example.com/endpoint/?id=1) to pass in variables that may be needed to complete an API's request. Endpoints are referred to as the "nouns" that verbs (HTTP methods) act on, and APIs use this framework to fulfill requests.

More specifically, a client sends a request composed of an HTTP method (verb) and an endpoint (noun) to receive specific data or to perform a particular action on the server. It's important to realize that the server is the one that fulfills a client's request by translating and performing a specific operation based on the method and endpoint provided.

Since the backend is where all of the heavy lifting takes place, it could be said that an API that utilizes HTTP methods and endpoints lives on the server, acting as an implementer for client requests. This model loosely defines RESTful APIs, which are examined in more detail in the next section.

RESTful APIs

APIs that utilize the HTTP protocol, request methods, and endpoints are referred to as RESTful APIs. REST (Representational State Transfer) is an architectural style that prescribes standards for web-based communication. The Google description of a RESTful system:

...resources are stored in a data store; a client sends a request that the server perform a particular action (such as creating, retrieving, updating, or deleting a resource), and the server performs the action and sends a response, often in the form of a representation of the specified resource.

This resource-oriented design is a key principle of REST. RESTful APIs are modelled as:

...collections of individually-addressable resources... The resources and methods are known as nouns and verbs of APIs. With the HTTP protocol, the resource names naturally map to URLs, and methods naturally map to HTTP methods...

These terms should sound familiar since you examined these building blocks in the previous sections. REST is the most widely used framework for APIs. In 2010, about 74% of public network APIs were HTTP REST APIs.

Besides query strings, RESTful APIs can also use the following fields in their requests:

- Headers: parameters that detail the HTTP request itself.
- Body: data that a client wants to send to a server.

The body is written in the JSON or XML data formatting language.

API Data Formats (JSON)

RESTful APIs use either XML or JSON (JavaScript Object Notation) as file formats for data held in the body of an HTTP request method.

JSON has surpassed XML in RESTful API use largely because JSON is lightweight, easier to read, and faster to parse. Next, a brief introduction to JSON syntax and structure will be covered. For a more comprehensive reference, be sure to check out the W3C's JSON syntax documentation.

JSON supports the following data types:

- Numbers: all types - no distinction between integers and floating point values.
- Strings: text enclosed in quotes.
- Booleans: True or False values.
- Arrays: a list of elements grouped by similar type.
- Null: an "empty" value.

JSON data is composed of key-value pairs. These are linked pieces of data that are composed of a unique identifier (a key) that references piece(s) of data (value). The key must be of type string and the value can be any of the data types listed above.

The following is an example of a simple key-value pair in JSON:

```
 "Key1" : "Value 1"
```

A JSON object uses curly braces { } to group data that's arranged in key-value pairs. The following is an example of an object that contains three key value pairs:

```
{
    "Name": "Julie",
    "Hometown": "Los Angeles, CA",
    "Age": 28
}
```

Commas separate the key-value pairs stored in an object.

JSON Validator

JSON files can contain any number of key-value pairs and/or objects. In professional development, it's not uncommon for some files to be hundreds, if not thousands, of lines long. As a developer, you know that one small error in formatting or syntax is enough to break your entire codebase.

JSON validators like jsonlint or, if you use Chrome as your primary browser, the JSONView extension quickly identify syntax and formatting issues in your JSON code and pinpoint ways to fix it.

Get some practice with JSON validation. Open the jsonlint validator in a new tab.

Paste the following codeblock into the validator:

```
{
    "Name": "Julie",
    "Hometown": "Los Angeles, CA",
    "Age": 28
}
```

Then click Validate JSON. You should receive a green message that says Valid JSON in the results section.

Now paste the following codeblock in the validator:

```
{
        "Name": "Julie"
            "Hometown": "Los Angeles, CA",
        "Age": 28
}
```

Click Validate JSON.

You will see that it has a missing comma and does not maintain proper indentation. The indentation gets corrected and the validator highlights where things went wrong.

The validator identified that there was a missing identifier (a comma) after the second line, which is what was anticipated.

Authentication and Authorization

The final piece to cover is the scheme of API authentication and authorization.

- Authentication refers to the process of determining a client's identity.
- Authorization refers to the process of determining what permissions an authenticated client has for a set of resources.

Authentication identifies who you are, and authorization determines what you can do.

There are three types of authentication/authorization services that Google APIs use. These are "API Keys", "Service accounts", and "OAuth". An API will use one of these authentication services depending on the resources it requests and from where the API is called from.

API Keys

API keys are secret tokens that usually come in the form of an encrypted string. API keys are quick to generate and use. APIs that use public data or methods and want to get developers up and running quickly will oftentimes use API keys to authenticate users.

OAuth

OAuth tokens are similar to API keys in their format, but they are more secure and can be linked to user accounts or identities. These tokens are used primarily when APIs give a developer the means to access user data.

While API keys give developers access to all of an API's functionality, OAuth client IDs are all based on scope; different privileges will be granted to different identities.

Service Accounts

A service account is a special type of Google account that belongs to your application or a virtual machine (VM) instead of to an individual end user. Your application assumes the identity of the service account to call Google APIs, so that the users aren't directly involved.

You can use a service account by providing its private key to your application, or by using the built-in service accounts available.

Thanks for Reading!

Do share your valuable feedback in the comments if you found this content helpful.

You can also email your feedback or suggestions on my email - durgeshmahajan1722@gmail.com