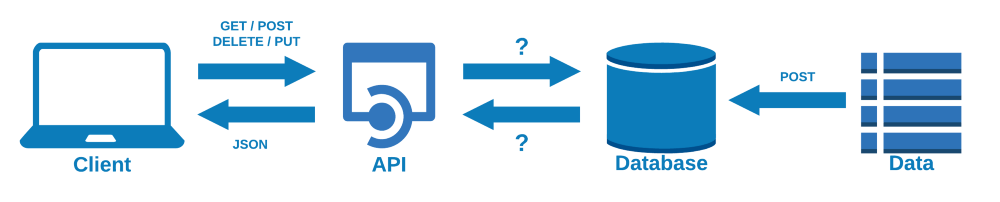
## Introduction to REST API

An Application Programming Interface (API) is a set of rules to facilitate this communication between different programs. [1] APIs let the product or service communicate with other products and services without having to know how they are implemented.[2]

**Re**presentational **S**tate **T**ransfer (REST) is an architectural Style for designing networked applications (i.e apps that use some form a network to communicate). It is the most popular style for building web APIs. REST determines specifications of the API through a set of rules that are followed when a REST API is created.

REST treats any data (e.g. image, video, text, etc) as a *resource*that the client can fetch/edit/delete. REST mandates that a client should be able to perform the appropriate operation by accessing a specific URL and sending a *request*. The server then sends an appropriate *response*.

REST is stateless, which implies that each request from the client must have all the necessary information for the server to understand it. For example, the client cannot assume that the server remembers what they had asked for earlier. With REST APIs, we usually send HTTP requests (more on HTTP [here](https://www.freecodecamp.org/news/http-and-everything-you-need-to-know-about-it/)) such as GET , PUT or POST[1]



*Picture show the main framework of REST API in which the client-side of the API allows for data to be added and removed using GET, POST, DELETE and PUT methods.[3]*

Simple Object Access Protocol, more known as SOAP APIs designed with SOAP use XML for their message format and receives requests through HTTP or SMTP. SOAP makes it easier for apps running in different environments or written in different languages to share information.[2]

APIs are RESTful as long as they comply with the 6 guiding constraints of a RESTful system:

* **Client-server architecture:** REST architecture is composed of clients, servers, and resources, and it handles requests through HTTP.
* **Statelessness:** No client content is stored on the server between requests. Information about the session state is, instead, held with the client.
* **Cacheability:** Caching can eliminate the need for some client-server interactions.
* **Layered system:** Client-server interactions can be mediated by additional layers. These layers could offer additional features like load balancing, shared caches, or security.
* **Code on demand (optional):** Servers can extend the functionality of a client by transferring executable code.
* **Uniform interface:** This constraint is core to the design of RESTful APIs and includes 4 facets:
* **Resource identification in requests**: Resources are identified in requests and are separate from the representations returned to the client.
* **Resource manipulation through representations**: Clients receive files that represent resources. These representations must have enough information to allow modification or deletion.
* **Self-descriptive messages**: Each message returned to a client contains enough information to describe how the client should process the information.
* **Hypermedia as the engine of application state**: After accessing a resource, the REST client should be able to discover through hyperlinks all other actions that are currently available.

## REFERENCES:

[1] Introduction to REST APIs; <https://towardsdatascience.com/introduction-to-rest-apis-90b5d9676004>; Dec 13, 2020;

[2] What is an API?; <https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces>

# [3] Launch your own REST API using Flask & Python in 7 minutes; <https://towardsdatascience.com/launch-your-own-rest-api-using-flask-python-in-7-minutes-c4373eb34239>; Jun 15, 2020

# [4] REST vs SOAP; https://www.redhat.com/en/topics/integration/whats-the-difference-between-soap-rest