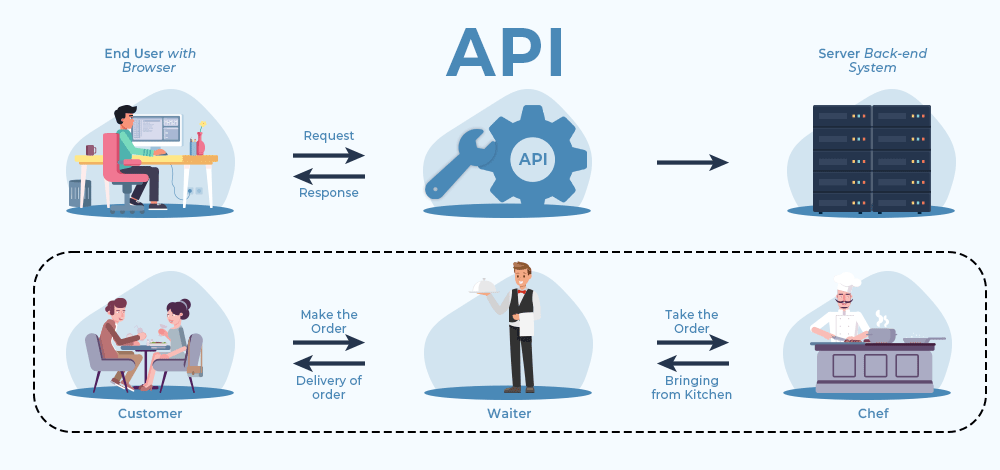
**API**

**What is an API?**

**API full form** is an **Application Programming Interface** that is a collection of communication protocols and subroutines used by various programs to communicate between them. A programmer can make use of various API tools to make their program easier and simpler. Also, an API facilitates programmers with an efficient way to develop their software programs. A simple API meaningis when an API helps two programs or applications to communicate with each other by providing them with the necessary tools and functions. It takes the request from the user and sends it to the service provider and then again sends the result generated from the service provider to the desired user.

A developer extensively uses APIs in his software to implement various features by using an API call with and without writing complex codes for the same. We can create an API for an **operating system,** **database system,** **hardware system, JavaScript file**, **webpages or websites** or similar object-oriented files.

APIs are the building blocks for today’s websites in which heavy data is transferred from the client to server and vice versa.



**How do APIs Work?**

The working of an API can be clearly explained with a few simple steps. Think of a client-server architecture where the client sends the request via a medium to the server and receives the response through the same medium. An API acts as a communication medium between two programs or systems for functioning. The client is the user/customer (who sends the request), the medium is the **application interface programming**, and the server is the backend (where the request is accepted, and a response is provided). Steps followed in the working of APIs –

* The client initiates the requests via the APIs URI (Uniform Resource Identifier)
* The API makes a call to the server after receiving the request
* Then the server sends the response back to the API with the information
* Finally, the API transfers the data to the client

APIs are considered safe in terms of attacks as they include authorization credentials and an API gateway to limit access so as to minimize security threats. To provide additional security layers to the data, HTTP headers, query string parameters, or cookies are used.

API’s architectures are:

* **REST (Representational State Transfer)**
* **SOAP (Simple Object Access Protocol)**

**What are REST APIs?**

REST stands for Representational State Transfer and follows the constraints of REST architecture allowing interaction with RESTful web services. It defines a set of functions (GET, PUT, POST, DELETE) that clients use to access server data. The functions used are:

* GET (retrieve a record)
* PUT (update a record)
* POST (create a record)
* DELETE (delete the record)

**Working:** A request is sent from client to server in the form of a web URL as HTTP GET or POST or PUT or DELETE request. After that, a response comes back from the server in the form of a resource which can be anything like HTML, XML, Image, or JSON. But now JSON is the most popular format being used in Web Services.

/Endpoint

Send request

client

server

Send response

HTML, XML, Image, JSON

**Functions:**

* **GET:** The HTTP GET method is used to **read** (or retrieve) a representation of a resource. In the safe path, GET returns a representation in XML or JSON and an HTTP response code of 200 (OK). In an error case, it most often returns a 404 (NOT FOUND) or 400 (BAD REQUEST).
* **POST:** The POST verb is most often utilized to **create** new resources, it’s used to create subordinate resources. That is, subordinate to some other (e.g. parent) resource. On successful creation, return HTTP status 201, returning a Location header with a link to the newly created resource with the 201 HTTP status.
* **PUT:** It is used for **updating** the capabilities. However, PUT can also be used to **create** a resource in the case where the resource ID is chosen by the client instead of by the server. In other words, if the PUT is to a URI that contains the value of a non-existent resource ID. On successful update, return 200 (or 204 if not returning any content in the body) from a PUT. If using PUT for create, return HTTP status 201 on successful creation. PUT is not safe operation but it’s idempotent.
* **PATCH:** It is used to **modify** capabilities. The PATCH request only needs to contain the changes to the resource, not the complete resource. This resembles PUT, but the body contains a set of instructions describing how a resource currently residing on the server should be modified to produce a new version. This means that the PATCH body should not just be a modified part of the resource, but in some kind of patch language like JSON Patch or XML Patch. PATCH is neither safe nor idempotent.
* **DELETE:** It is used to **delete** a resource identified by a URI. On successful deletion, return HTTP status 200 (OK) along with a response body.