

Setting up http server

🕒 Created	@April 25, 2022 7:59 PM
▼ Class	
▼ Type	
🔗 Materials	
☑ Reviewed	<input type="checkbox"/>

How the Web Works ?

Different things in web, like websites, torrent, messaging apps, emails etc everything works with a different architecture.

The main architecture that we have to focus on is Client - Server architecture.

What the hell is a server ?

So server has a very wide perspective, at some places we refer it as a hardware and at some places we refer it as a software. Server is a computer or a program which provides service to outside world.

When I say, server is a computer that means I can turn my laptop into a server, or may be I can run a process on my laptop that can act as a server.

When we talk about cloud, in computer science we refer cloud machine or cloud server as a real machine which is present in a data centre at some remote place on the earth. So now a days we have a lot of services like AWS, Google cloud, MS Azure, Digital Ocean, Heroku etc which provides these cloud solutions where we can setup our own server as well.

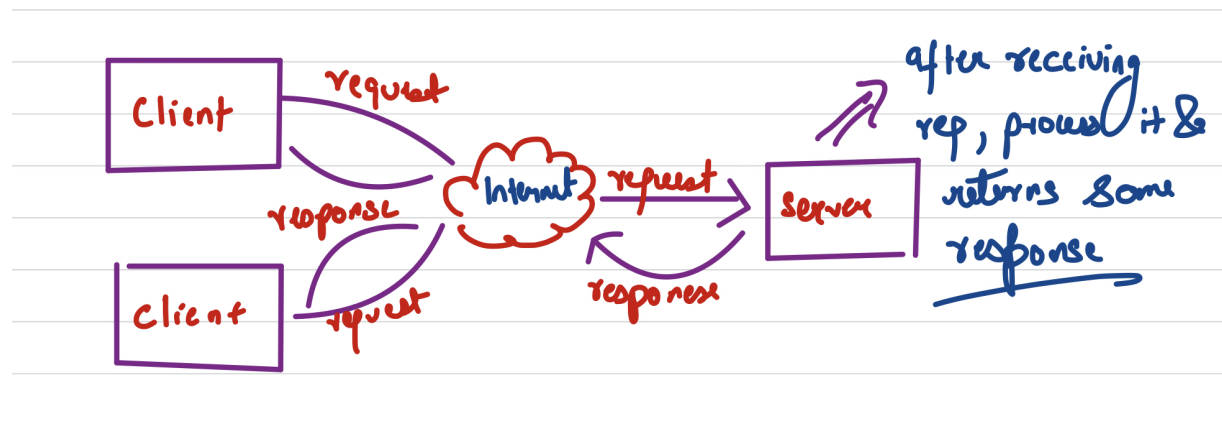
What is a client ?

Client is referred to a service that can obtain information from end server and actually display meaningful data to the end users. When we say client, then website, webapps, mobileapps, watchapps, etc everything is a client.

Example:

Zomato's android or ios apps are the clients and the servers through which they fetch data about restaurant and food are their backend servers.

Client Server Architectures



In the client server architecture, there can be multiple clients who based on their needs and user preferences will be raising request. This request will go over the network and reach the corresponding servers who are ready to respond to these queries. As soon as these servers receive the request they start processing it and then return the desired response.

But there is a bigger question ! When we say these client and servers are going to send request and receive response, how they actually communicate ?

How two or more machines are going to communicate over the network is defined by some set of rules. These rules might differ based on the application or the environment through which the machines will communicate. These rules are called as **protocols**. Examples of few protocols are **http**, **https**, **smtp**, **tcp**, **ftp** etc.

HTTP (Hyper text transfer protocol)

Before HTTPS, let's understand what is Hyper text !

Hyper text is just another form of document, which includes hyper links (activated working links using which we can access things) which connect the document with other hyper text documents or sometimes with images, videos etc.

The language that we use to build this hyper text document, is `HTML` (hyper text markup language).

Now what is HTTP ?

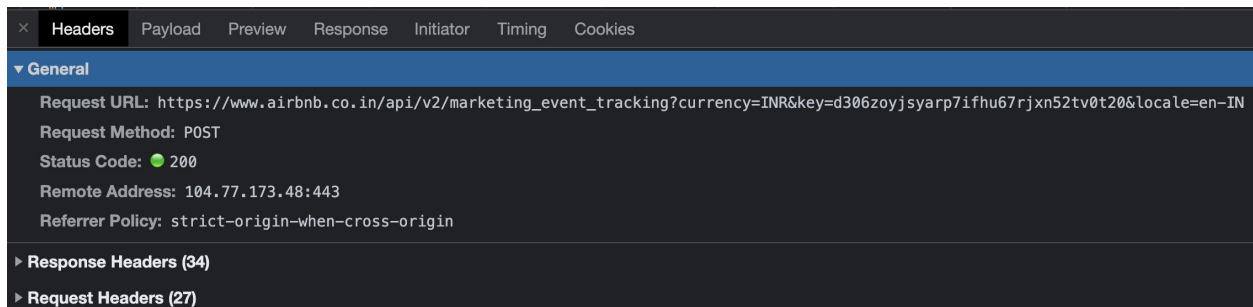
It is a set of rules that defines how two machines can transfer data in the form of hyper text.

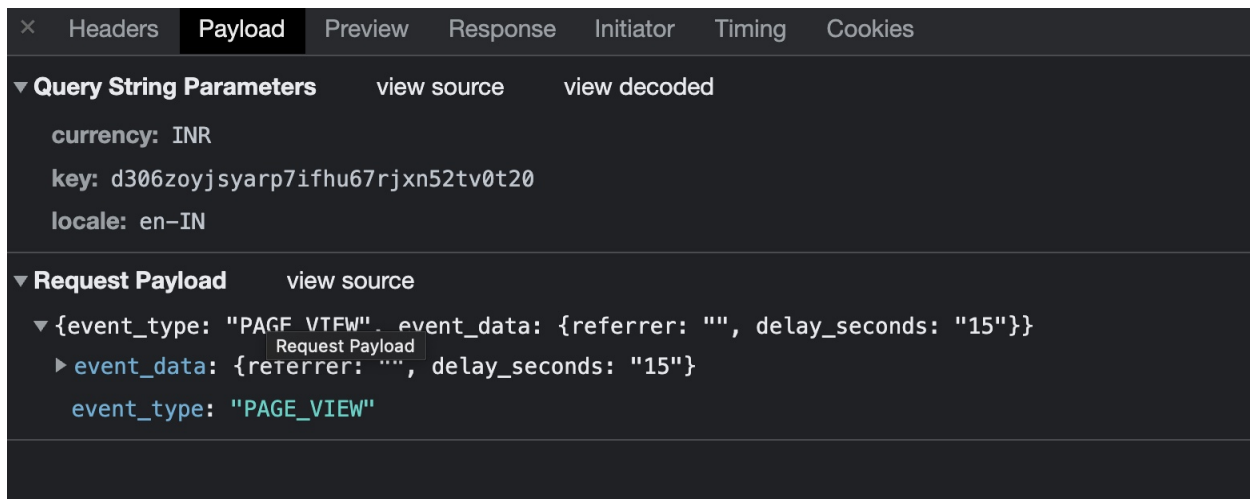
What are these rules ? We will learn in computer networks.

HTTP Request and Response

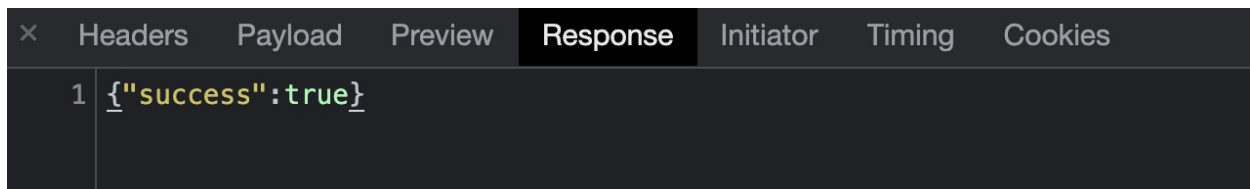
So the request that client sends to a server is a HTTP request and the response that server returns to a client is HTTP response. We can actually visualise these request and response objects on our browsers.

HTTP Request





HTTP Response



HTTP Status Codes


So what happened to the request, the status about it is given by HTTP status codes. These codes represent the current situation of the http request response cycle. Whether it was successfully completed, or may be the client was not authorised, or may be the server had some failures anything that can go wrong or right during the http communications, it's status is represented by these codes.

- The range **100-199** ⇒ Informational status codes that means they provide extra information about the request.
- The range **200-299** ⇒ Successful status codes that means they generally refer to the success of the request response cycle.
- The range **300-399** ⇒ Redirection messages that means the server redirected you from the URL you requested.
- The range **400-499** ⇒ Client side error codes, that means some error was made from the client that's why request failed
- The range **500-599** ⇒ Server side error codes, that means some error was made from the server side that's why request failed.

HTTP response status codes - HTTP | MDN

This interim response indicates that the client should continue the request or ignore the response if the request is already finished. 101 Switching Protocols This code is sent in response to an request header from the client

 <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

 mdn web docs

HTTPS - HTTP Secure

So **HTTPS** is secure version of http. In normal http whatever request response messages you're sending are plain as it is. But in HTTPS they're encrypted. So, no one can intercept the data when it is getting transferred over the internet.

Let's Setup a basic HTTP Server in NodeJS

We will setup a basic server in NodeJS. This server will be setup on our own laptops and for the time being, we can access these servers from our laptop only.

So first of all, we need to setup a node/npm project.

- In the directory where you want to setup the project execute the command `npm init`, this will prompt basic npm project setup instructions, we can fill the details or just skip them by pressing `enter` key again and again.
- This will give us a new file `package.json`. (Will describe this file later)
- So the node server we want to setup is just an ordinary javascript program, which will use the features of **NodeJS** (which is the runtime environment features).

To setup the server NodeJS provides a module called as `http`.

- So we will make a new file `server.js` and use this module for creating the server.

Note:

On your mobile, you've a lot of applications. When an http response comes from the server, how will it detect that in the whole internet which device this response is expected to be sent on ?

So, just like how we all have addresses to locate ourselves uniquely, devices connected on the internet also has got addresses, called as IP address.

As soon as the response detects which IP address it has to go, how will it detect on the IP Address which process is expecting this response ? This is Identified by `ports` . Port is an identifier that uniquely identifies a process running on a machine. Every device has got a lot of ports that belongs to different applications. Some ports are free also for new applications to come.

Example: Port 80 is reserved for managing HTTP processing.

Port 3000 is free etc.

Every port has a `port number` using which we refer it.

Code for basic server setup

// File: `index.js`

```
const http = require('http'); // We have imported the http module

const listener = function(req, res) {
  /*
    this function will be executed whenever we will receive any
    http request from the client.
    Details of the http request will be present in the `req` parameter.
    Details of the http response that we will send is going to be set inside `res`
  */
  console.log("A new HTTP Request received");
}

/*
  In the http object we will be using the createServer function to create the server
  This function takes a callback as argument, this callback will be executed again and again
  whenever we receive any http request
*/
const server = http.createServer(listener);

server.listen(3000); // On my machine my current nodejs server will listen on port 3000.
// Make sure we enter a number which is of a free port: 3000, 3001, 30002, 8080, 8000 etc
```

You can run `node index.js` to run the above file (assuming index.js is your filename). This will start your server.

To make a call to the server you can type `127.0.0.1:3000` on your browser and it will make a call to the server, you can read the logs for it.

```
node> npm install
sanketsingh@Sankets-MacBook-Pro basic_node_server % node index.js
A new HTTP Request received
A new HTTP Request received
```

`127.0.0.1` is an IP address that if you call from your machine, refers to your machine only, So this server we are running locally on our machine on port `3000` so we type `127.0.0.1:3000` to access it. Instead of `127.0.0.1` we can also write `localhost` to access our machine i.e.

`localhost:3000`

The previous code will not return any response to the client, to return it we can use `res.end()` function and pass our response. Below is the updated code for it. Restart the server and BOOM!!

```
const http = require('http'); // We have imported the http module

const listener = function(req, res) {
  /*
    this function will be executed whenever we will receive any
    http request from the client.
    Details of the http request will be present in the `req` parameter.
    Details of the http response that we will send is going to be set inside `res`
  */
  console.log("A new HTTP Request received");
  // To prepare a response for the client we can use res parameter of this function
  res.end('Hello world!! Welcome to the server'); // Ending the request with the following response
}

/*
  In the http object we will be using the createServer function to create the server
  This function takes a callback as argument, this callback will be executed again and again
  whenever we receive any http request
*/
const server = http.createServer(listener);

server.listen(3000); // On my machine my current nodejs server will listen on port 3000.
// Make sure we enter a number which is of a free port: 3000, 3001, 30002, 8080, 8000 etc
```

CURL

Curl is a service using which we can make http calls from the terminal. Then we won't need a browser

`curl localhost:3000` is the command that we can use.

