Application Programming Interface

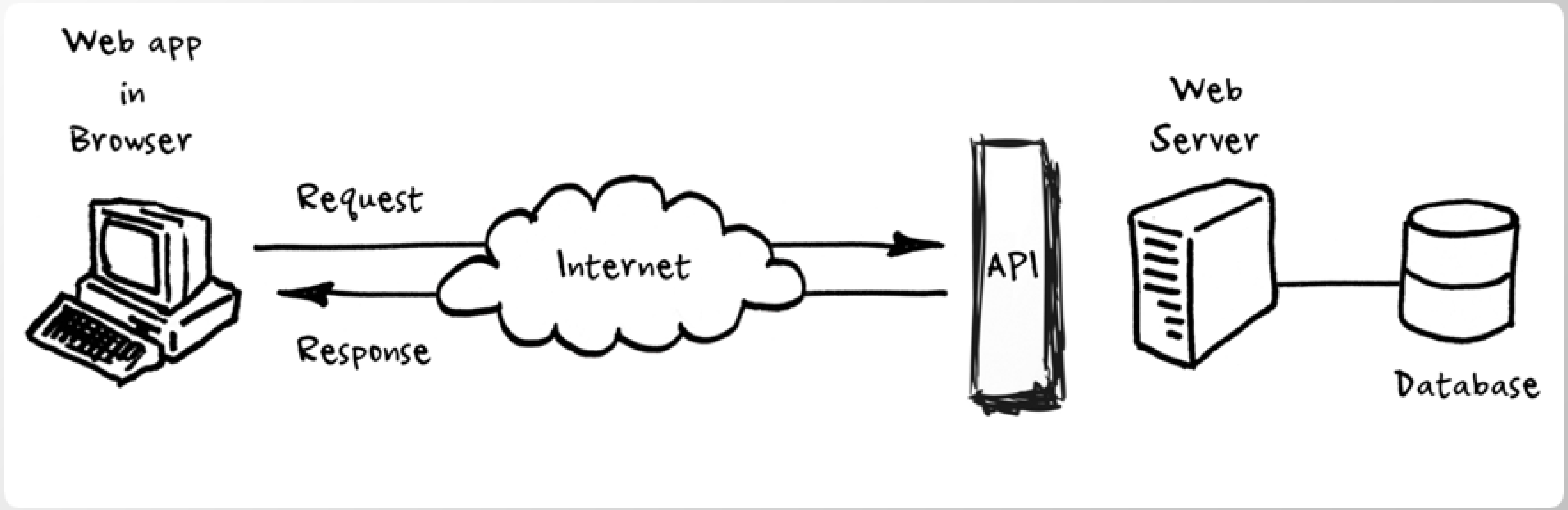
Created for

techigai

What is API ?

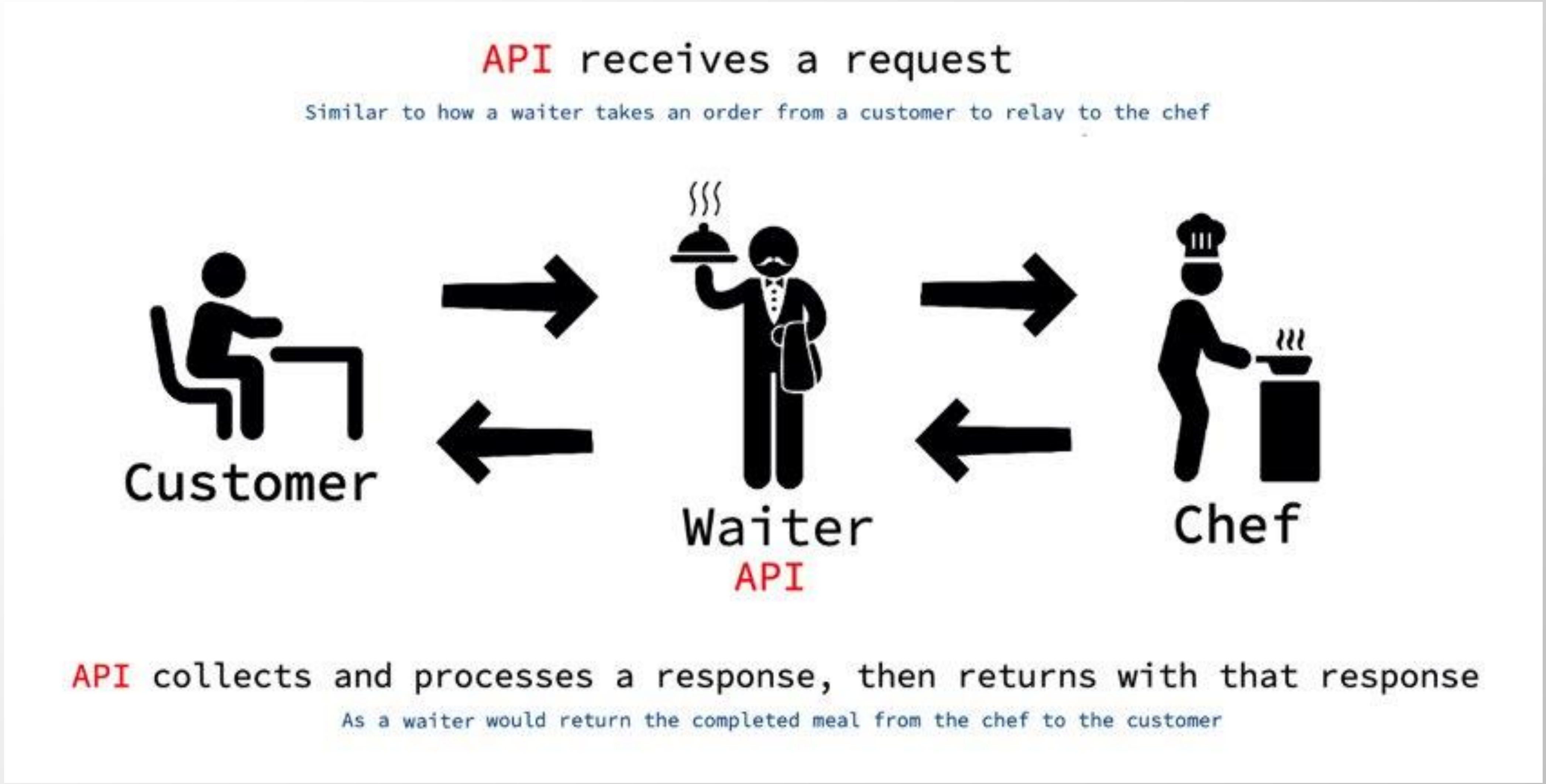
•API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other.

•The API is not the database or even the server, it is the code that governs the access point(s) for the server.



API Analogy

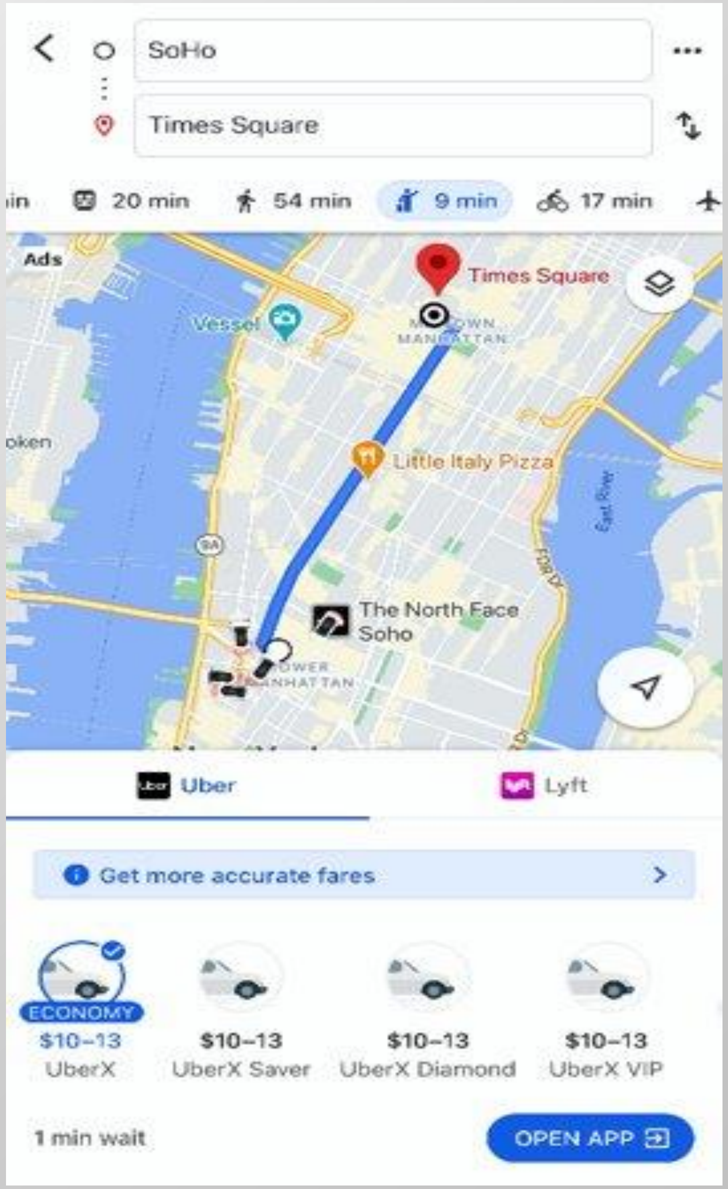
techigai


API receives a request
Similar to how a waiter takes an order from a customer to relay to the chef
Customer
Waiter
API
Chef
API collects and processes a response, then returns with that response
As a waiter would return the completed meal from the chef to the customer

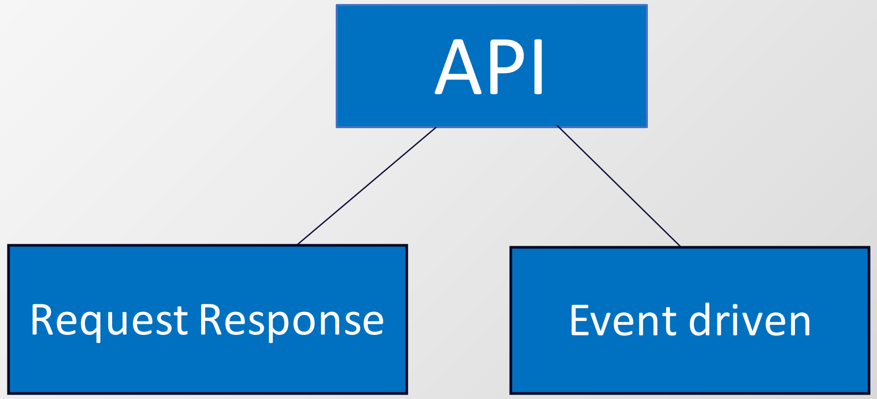
techigai

Real world example of API

- Google Maps is using Uber's API to request information by sending a HTTP request function with the start and end points of the ride.
- Uber's API then sends information back such as how many cars are available, where are they currently located, and the costs for the various ride options.
- This information is then processed by Google Maps and is available to the user directly linking them the Uber app if it is previously downloaded.



Types of API



Request Response

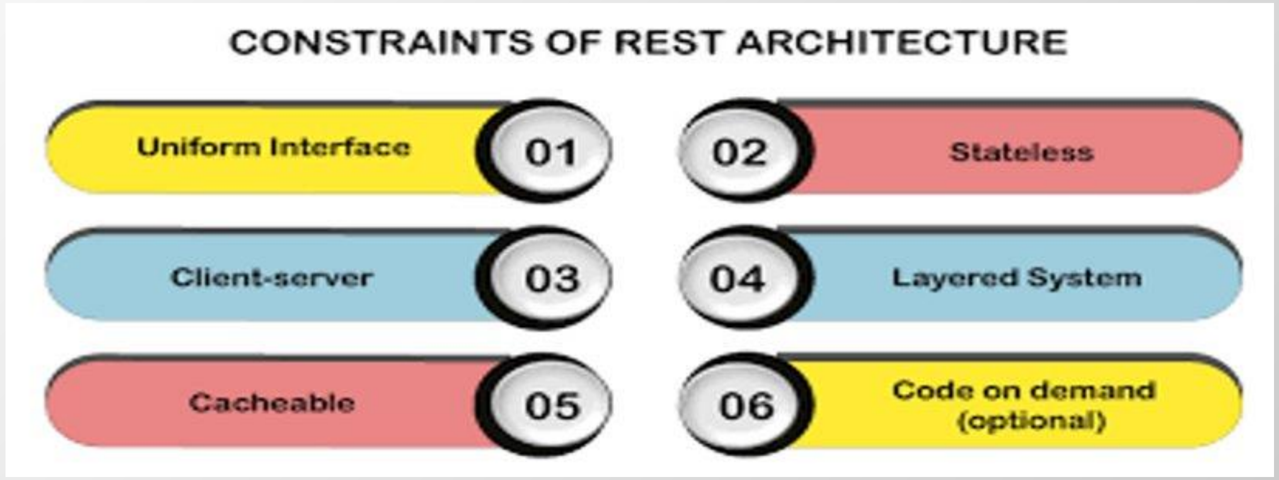
- RPC - Remote Procedure Calls
- SOAP - Simple object Access Protocol
- REST – Representational State transfer
- GraphQL

Event Driven

- Web Hooks
- Web Sockets
- Http Streaming

REST

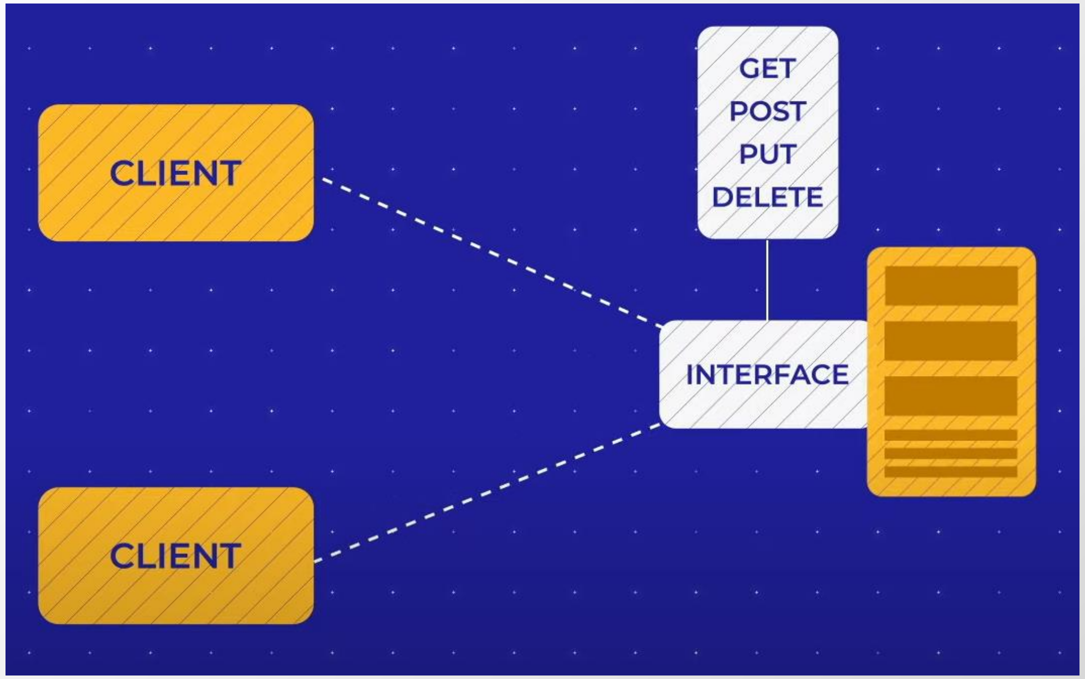
- REST stands for Representational State Transfer. REST is a software architectural style that defines the set of rules to be used for creating web services.
- REST works on top of the HTTP transport So, it takes advantage of HTTP's native capabilities, such as GET, PUT, POST and DELETE i.e, CRUD operations.
- There are six architectural constraints which makes any web service are listed below:



- The only optional constraint of REST architecture is code on demand. If a service violates any other constraint, it cannot strictly be referred to as RESTful.

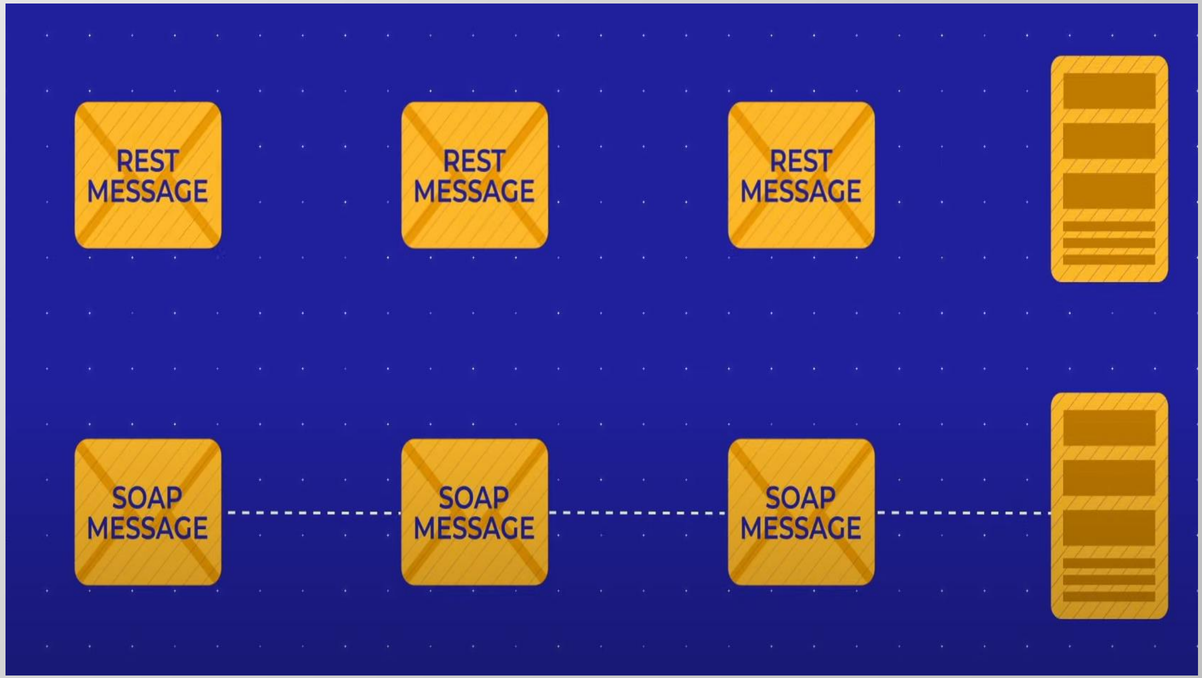
Uniform Interface

permitting a uniform way of interacting with a given server regardless of device or application type.



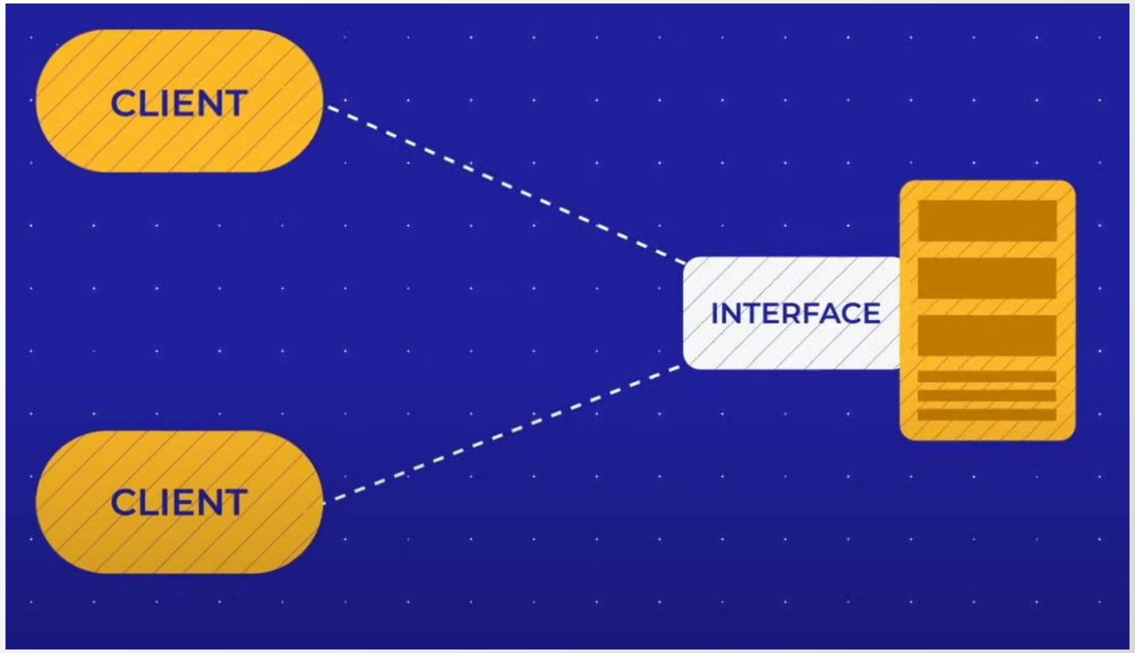
Stateless interactions

Server doesn't store any info related to previous session. It considers every request as new request.



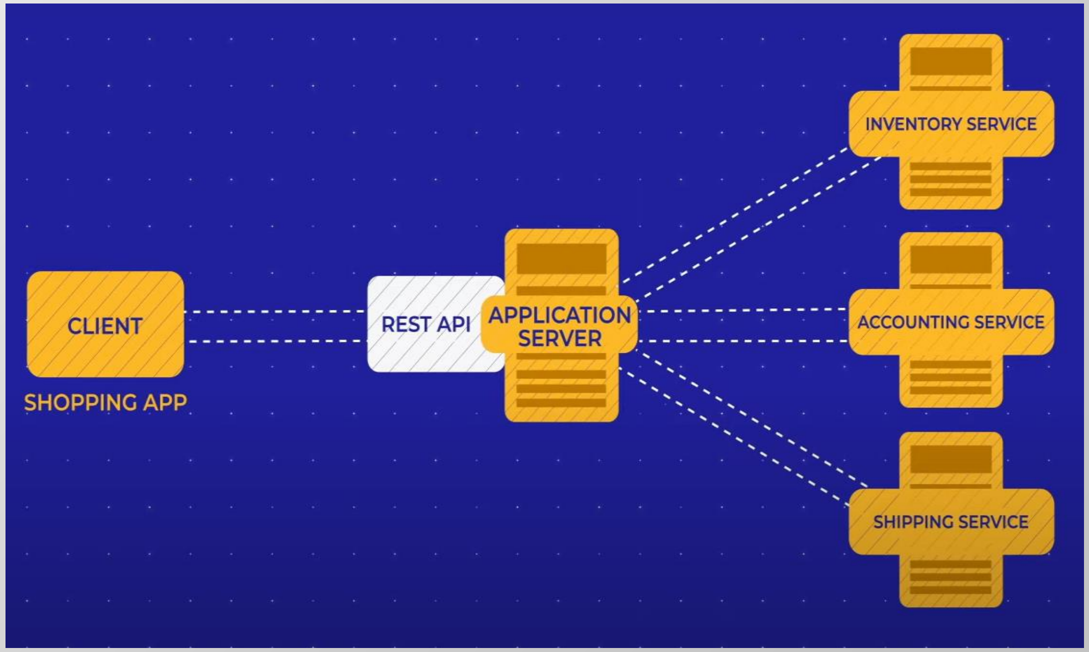
Client – Server Architecture

allowing for independent evolution of either side



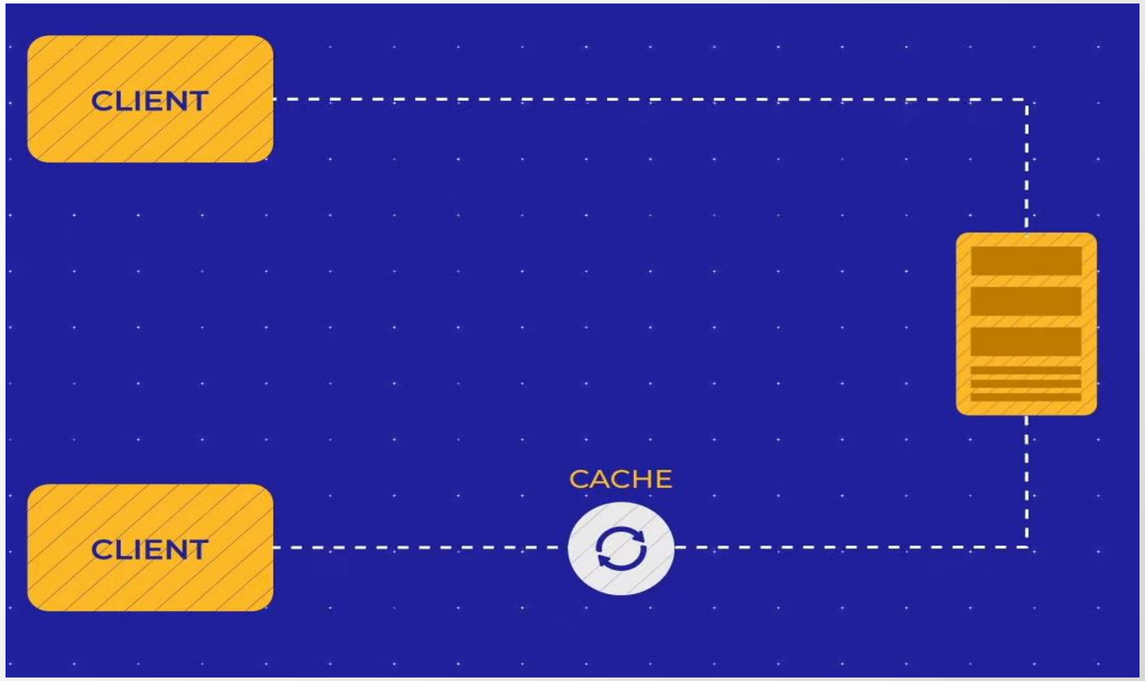
Layered System

Client only concern about the reponse of contact server and doesn't care about what behind the interface. It helps in scalability.



Caching

It uses HTTP caching so client can retain the content hence reduce the load on server



Code on Demand

It is an optional feature. According to this, servers can also provide executable code to the client. The examples of code on demand may include the compiled components such as Java applets and client-side scripts such as JavaScript.

Rest vs Restful

- The APIs which follow all rest architecture constraints are called as **Restful APIs**.

- The APIs which follow few of the rest architecture constraints are called as **Rest APIs**.

- If we want our API to have some features which voilates rest constraints like :
  - ➢ Session
  - ➢ Stateful
  - ➢ Returns images or videos

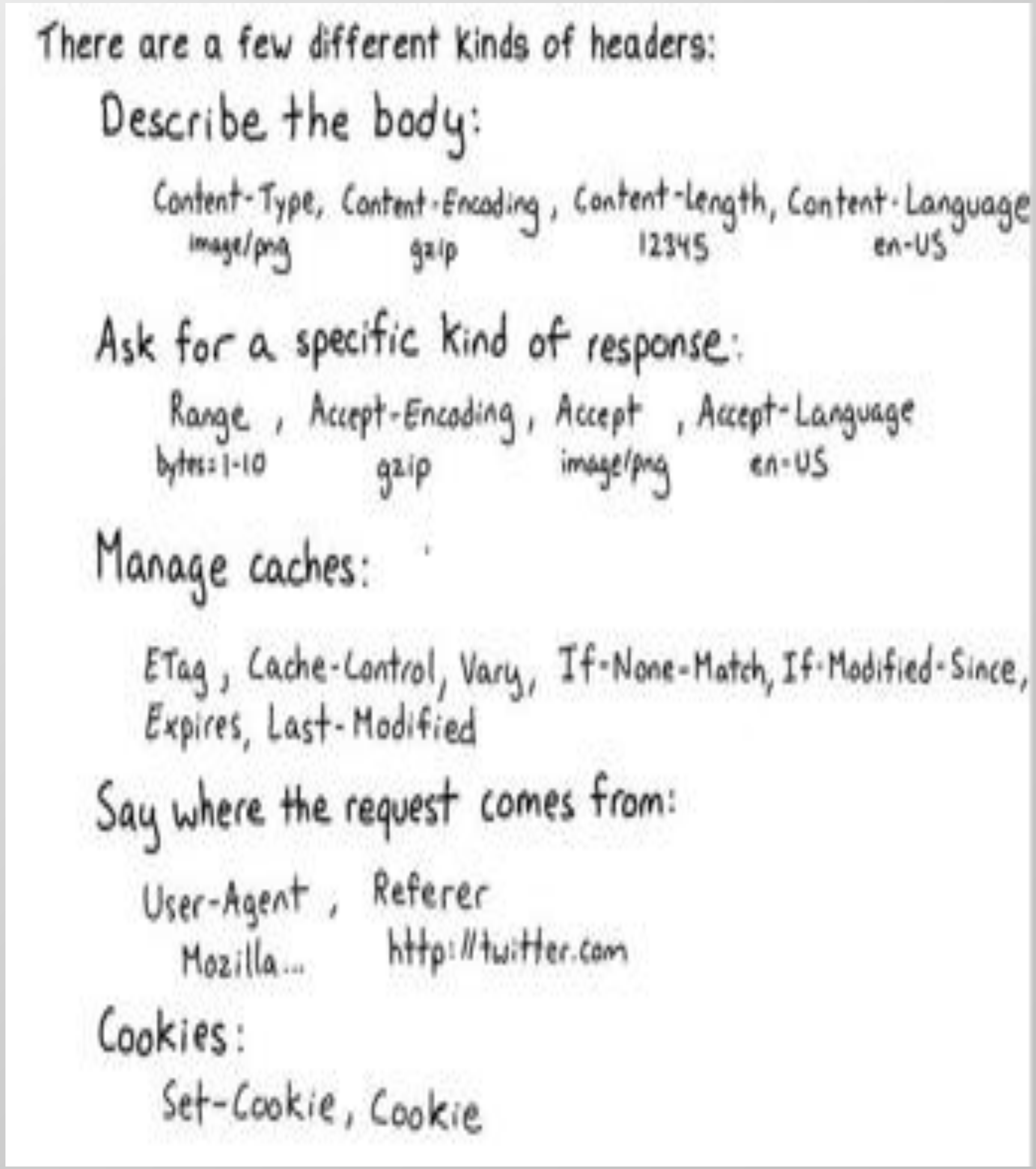
  Then we will go for rest.

HTTP and HTTP Headers

- **HTTP** is a server-client protocol. All communication is initiated by the client, in the form of an HTTP request. On receiving this request, the server sends an HTTP response back to the client.
- **HTTP headers** let the client and the server pass additional information with an HTTP request or response. An HTTP header consists of its case-insensitive name followed by a colon (**:**), then by its value

Headers can be grouped according to their contexts:

- Request headers contain more information about the resource to be fetched, or about the client requesting the resource.
- Response headers hold additional information about the response, like its location or about the server providing it.
- Representation headers contain information about the body of the resource, like its MIME type, or encoding/compression applied.



There are a few different kinds of headers:

Describe the body:

Content-Type, Content-Encoding, Content-Length, Content-Language
image/png gzip 12345 en-US

Ask for a specific kind of response:

Range, Accept-Encoding, Accept, Accept-Language
bytes:1-10 gzip image/png en-US

Manage caches:

ETag, Cache-Control, Vary, If-None-Match, If-Modified-Since, Expires, Last-Modified

Say where the request comes from:

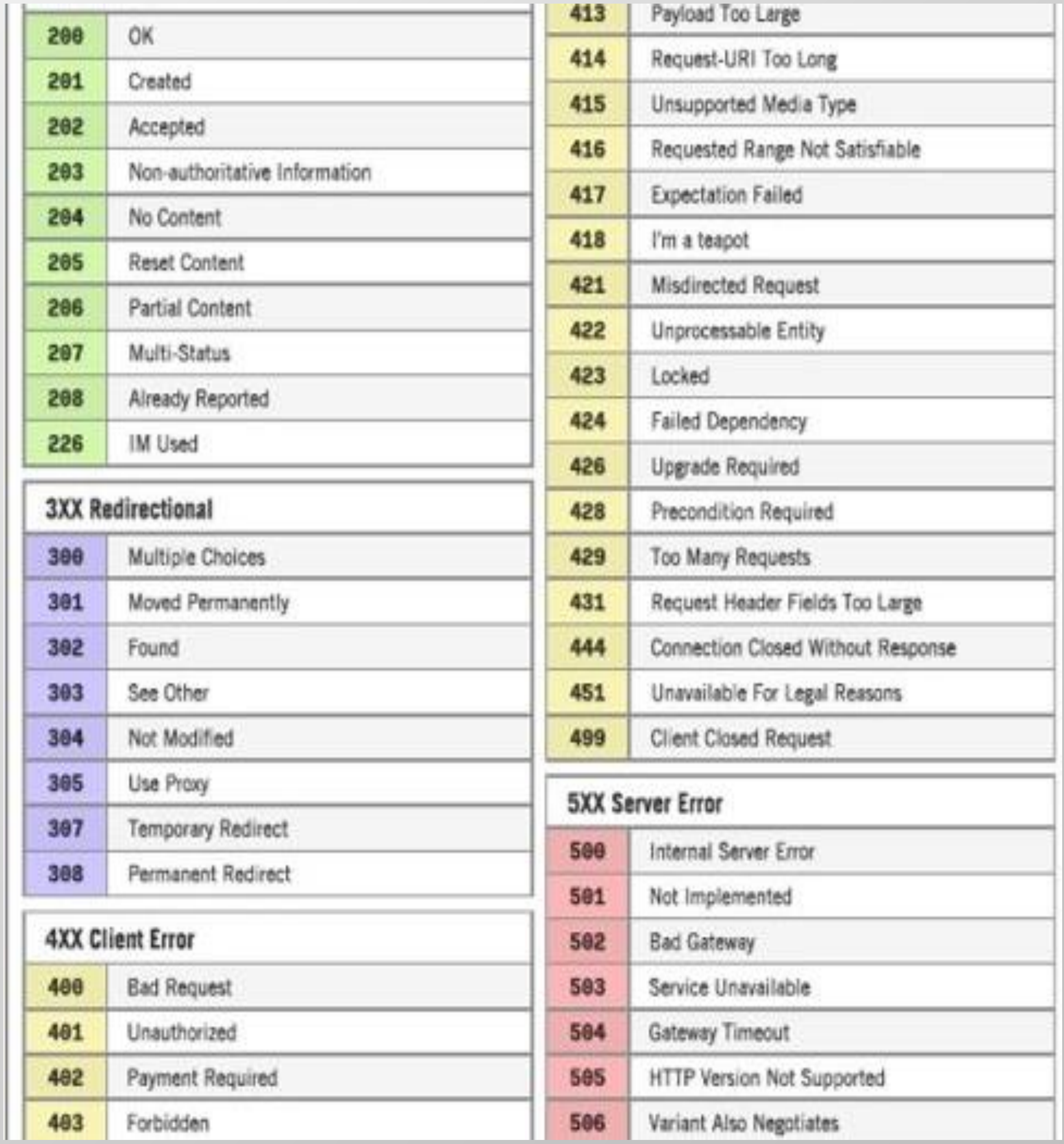
User-Agent, Referer
Mozilla... http://twitter.com

Cookies:

Set-Cookie, Cookie

HTTP Status Codes

- When a client requests something from the server, the server responses something to a client. To facilitate the readability, it was created the status code, in this way each code means something to the client. The status codes are separated into 5 categories:
- **1xx** [Informational]: The request was received, continuing process;
- **2xx** [Successful]: The request was successfully received, understood and accepted;
- **3xx** [Redirection]: Further action needs to be taken to complete the request;
- **4xx** [Client Error]: The request contains bad syntax or cannot be fulfilled;
- **5xx** [Server Error]: The server failed to fulfill a valid request.
- For instance, if the client receives a response **200**, that means that everything went well and the result was **OK**. However, if a response is **400**, that means the result is a **bad request**.



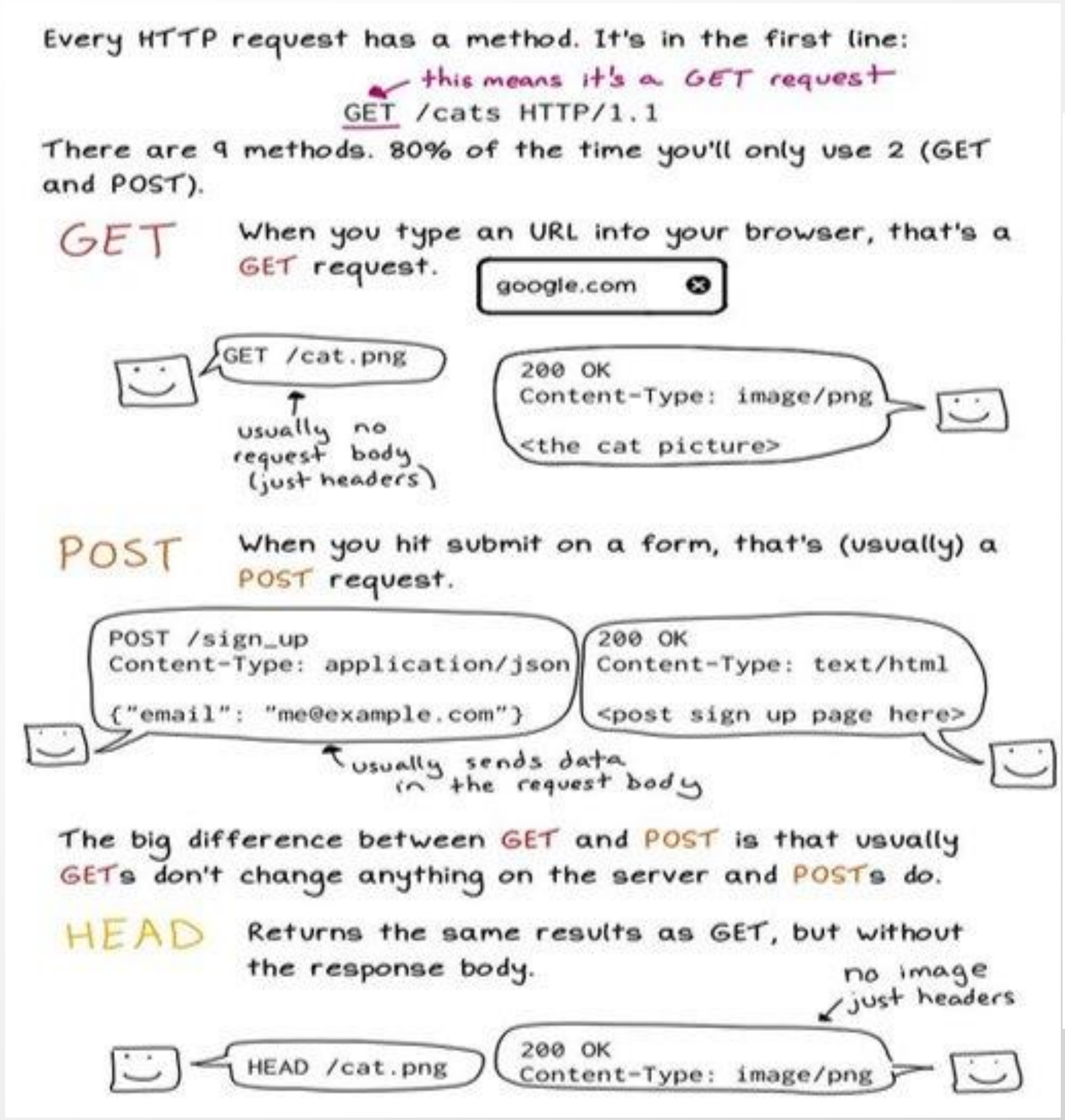
200	OK
201	Created
202	Accepted
203	Non-authoritative Information
204	No Content
205	Reset Content
206	Partial Content
207	Multi-Status
208	Already Reported
226	IM Used

3XX Redirectional	
300	Multiple Choices
301	Moved Permanently
302	Found
303	See Other
304	Not Modified
305	Use Proxy
307	Temporary Redirect
308	Permanent Redirect

4XX Client Error	
400	Bad Request
401	Unauthorized
402	Payment Required
403	Forbidden
413	Payload Too Large
414	Request-URI Too Long
415	Unsupported Media Type
416	Requested Range Not Satisfiable
417	Expectation Failed
418	I'm a teapot
421	Misdirected Request
422	Unprocessable Entity
423	Locked
424	Failed Dependency
426	Upgrade Required
428	Precondition Required
429	Too Many Requests
431	Request Header Fields Too Large
444	Connection Closed Without Response
451	Unavailable For Legal Reasons
499	Client Closed Request

5XX Server Error	
500	Internal Server Error
501	Not Implemented
502	Bad Gateway
503	Service Unavailable
504	Gateway Timeout
505	HTTP Version Not Supported
506	Variant Also Negotiates

HTTP Methods



Every HTTP request has a method. It's in the first line:

this means it's a GET request

GET /cats HTTP/1.1

There are 9 methods. 80% of the time you'll only use 2 (GET and POST).

GET When you type an URL into your browser, that's a GET request.

google.com

```
GET /cat.png
```

usually no request body (just headers)

```
200 OK
Content-Type: image/png

<the cat picture>
```

POST When you hit submit on a form, that's (usually) a POST request.

```
POST /sign_up
Content-Type: application/json

{"email": "me@example.com"}
```

```
200 OK
Content-Type: text/html

<post sign up page here>
```

usually sends data in the request body

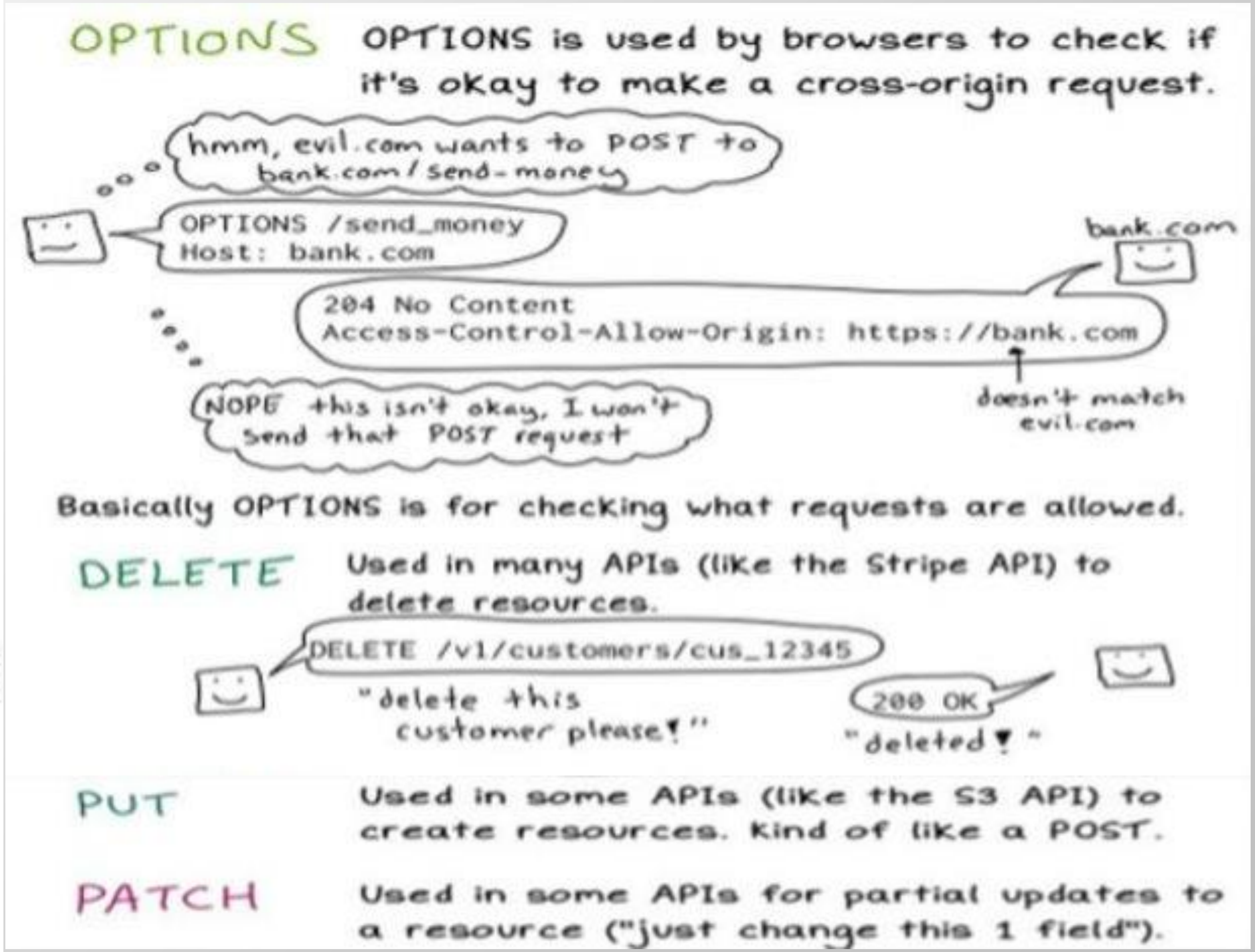
The big difference between GET and POST is that usually GETs don't change anything on the server and POSTs do.

HEAD Returns the same results as GET, but without the response body.

no image just headers

```
HEAD /cat.png
```

```
200 OK
Content-Type: image/png
```



OPTIONS OPTIONS is used by browsers to check if it's okay to make a cross-origin request.

hmm, evil.com wants to POST to bank.com/send-money

```
OPTIONS /send_money
Host: bank.com
```

bank.com

```
204 No Content
Access-Control-Allow-Origin: https://bank.com
```

doesn't match evil.com

NOPE this isn't okay, I won't send that POST request

Basically OPTIONS is for checking what requests are allowed.

DELETE Used in many APIs (like the Stripe API) to delete resources.

```
DELETE /v1/customers/cus_12345
```

"delete this customer please!"

200 OK

"deleted!"

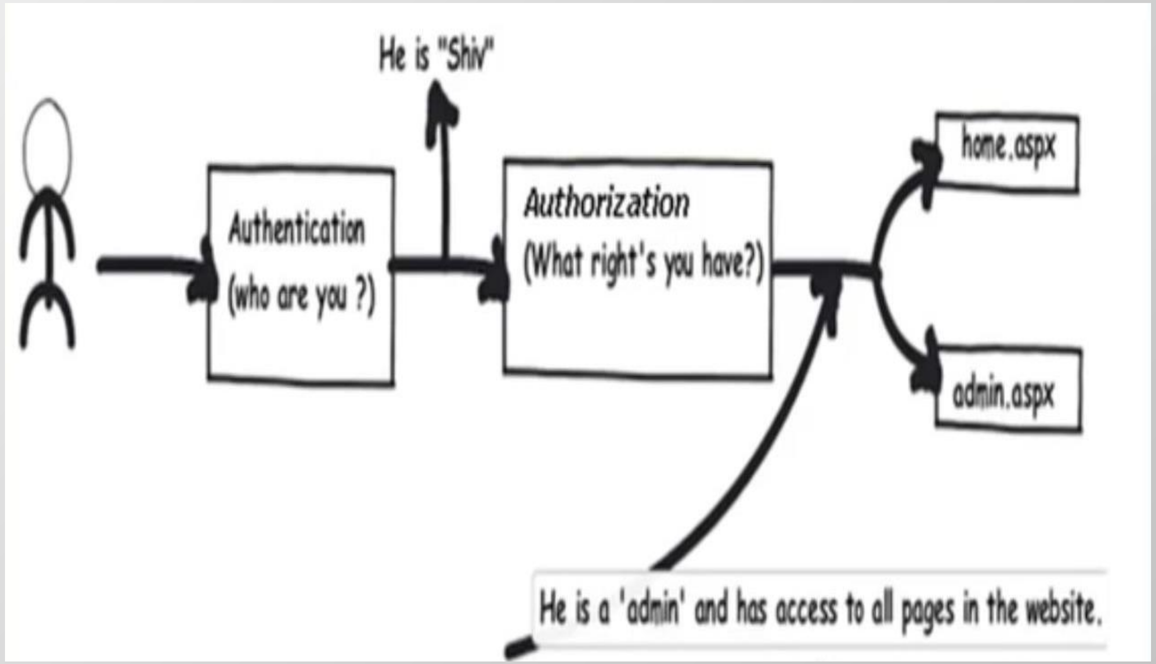
PUT Used in some APIs (like the S3 API) to create resources. Kind of like a POST.

PATCH Used in some APIs for partial updates to a resource ("just change this 1 field").

Authentication vs Authorization

Authentication is the process of verifying who someone is

Authentication
LOGIN
Username :
Password :
Who are you?
Vs
What permissions do you have?
Authorization


He is "Shiv"
Authentication (who are you ?)
Authorization (What right's you have?)
home.aspx
admin.aspx
He is a 'admin' and has access to all pages in the website.

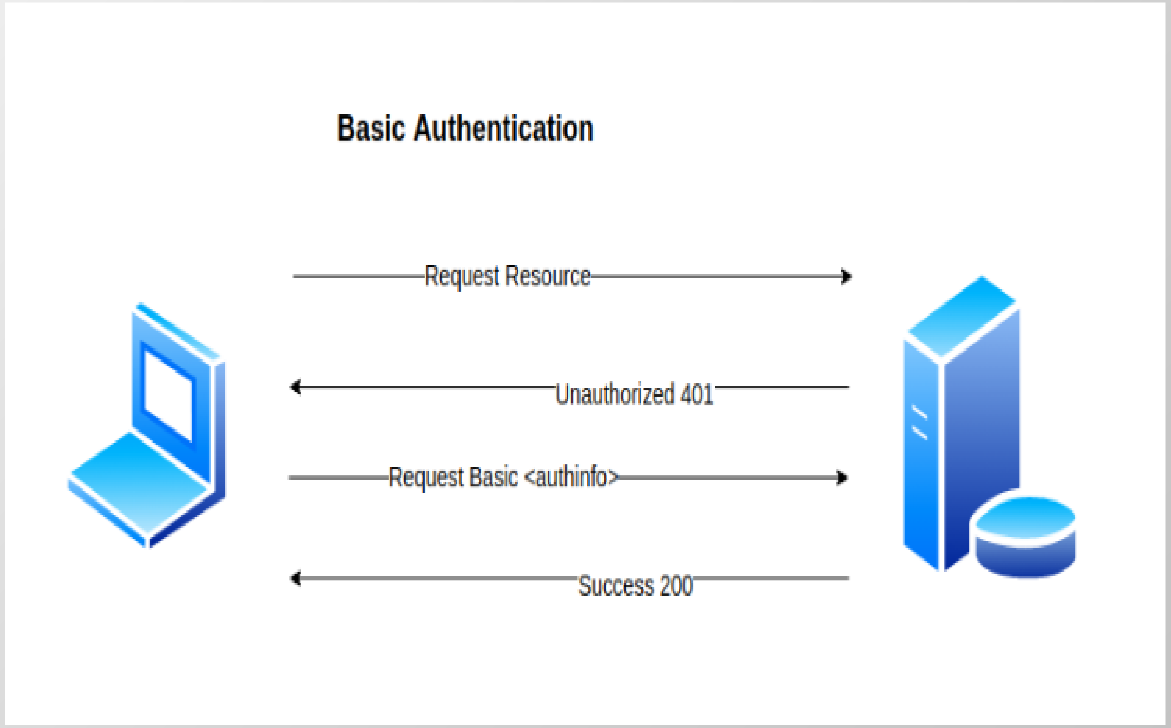
Authorization is the process of verifying what specific applications, files, and data a user has access to

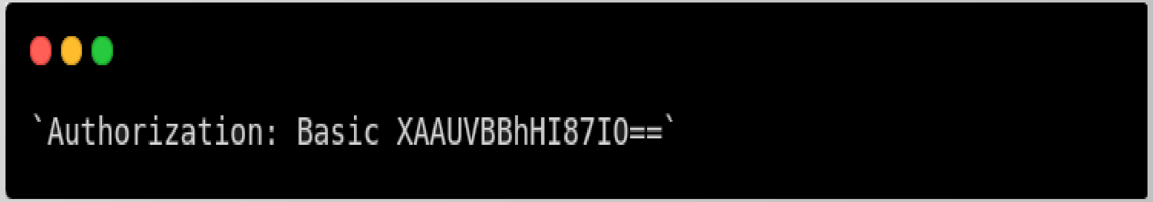
Different ways of Authentication

Basic Authentication

Basic Authentication is the simplest authentication mechanism to authenticate access to resources over HTTP. The credentials are send in the request headers.

- User submits the credentials
- The username and password are concatenated into a single string: `username:password`
- Encodes the string using `base64` algorithm
- Set it in the Authorization header with `Basic` keyword and send it along each HTTP Request.

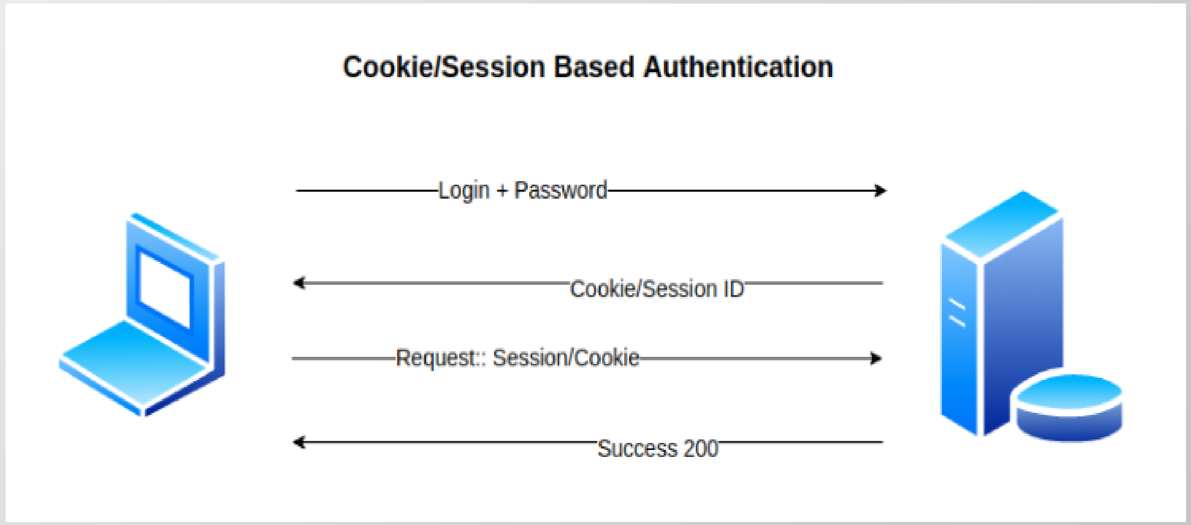




```
`Authorization: Basic XAAUVBBhHI87IO==`
```

Cookie Based Authentication

The cookie Based Authentication is also known as session based authentication. In this method the user is assigned some unique identifier and this identifier is stored on the server in memory. Client sends this session id in all the requests and server uses it to identify the user.



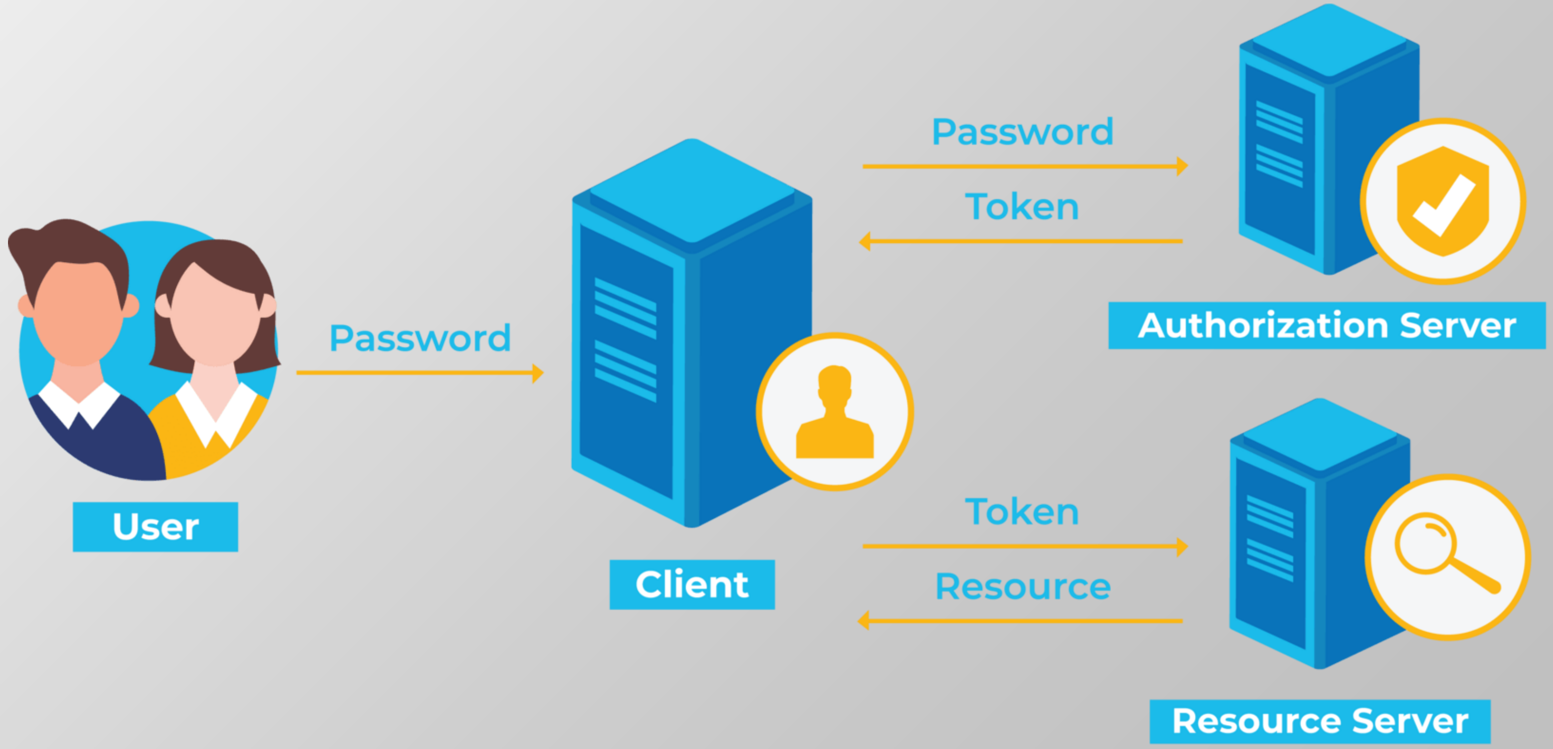
1.Client sends the login request
2.Server validates the credentials, creates a session and stores it in memory assigned to current user and returns back the generated session id
3.Client receives the session id and stores it in a cookie
4.Client sends next requests with the current session id in its storage
5.when the user logs out , the session is destroyed (cookie removed + session removed from the server) and same session id cannot be reused

Token Based Authentication

Token Based authentication (also called bearer authentication) is an HTTP authentication scheme that involves security tokens called bearer tokens. Instead of sending username and password over for authentication we use a server generated token.

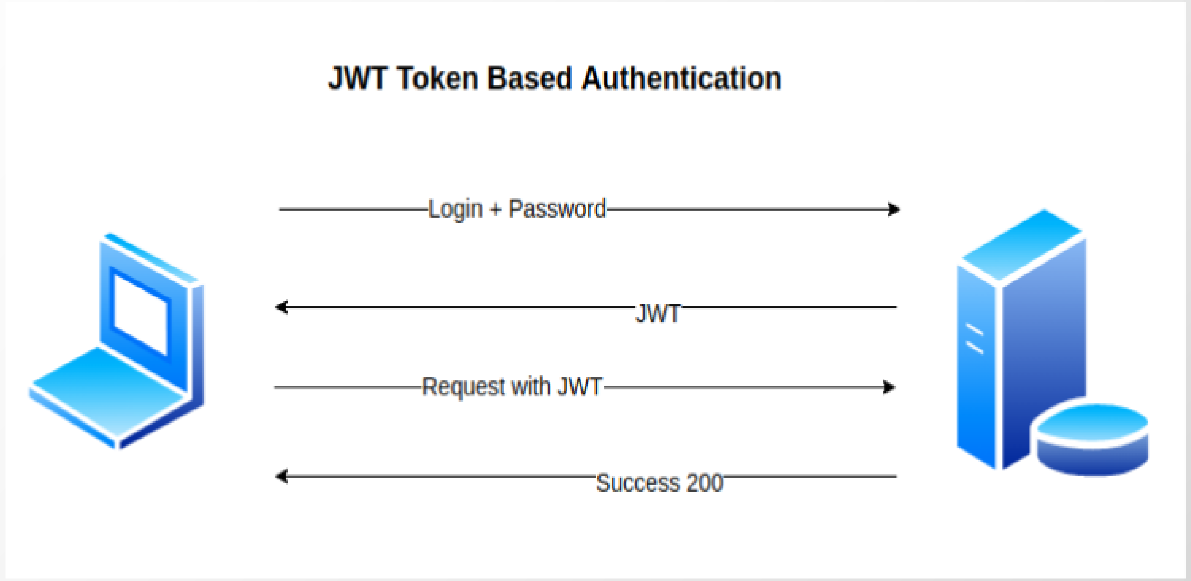
Types of Token Based authentication

- JSON Web Tokens (JWT)
- Open Authorization (OAuth)



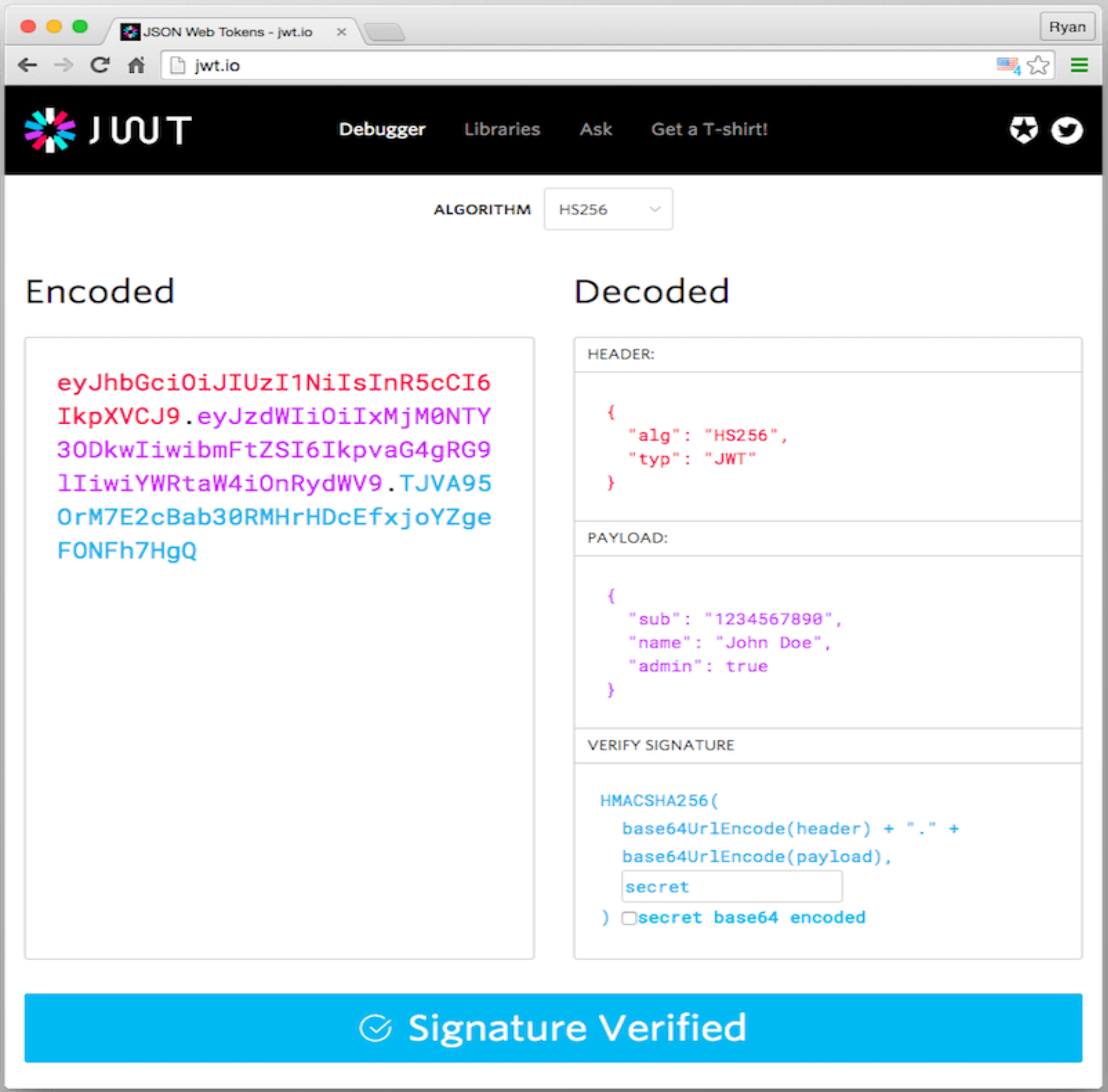
JSON Web Token (JWT)

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.



How does it work?

1. User submits a username and password
2. Server validates and returns a JWT token
3. Use the token to allow future requests



ALGORITHM HS256

Encoded

```
eyJhbGciOiJIUzI1NiIsInR5cCI6
IkpXVCJ9.eyJzdWIiOiIxMjM0NTY
3ODkwIiwibmFtZSI6IkpvaG4gRG9
lIiwiYWRtaW4iOnRydWV9.TJVA95
OrM7E2cBab30RMHrHDcEfxjoYZge
FONFh7HgQ
```

Decoded

HEADER:

```
{
  "alg": "HS256",
  "typ": "JWT"
}
```

PAYLOAD:

```
{
  "sub": "1234567890",
  "name": "John Doe",
  "admin": true
}
```

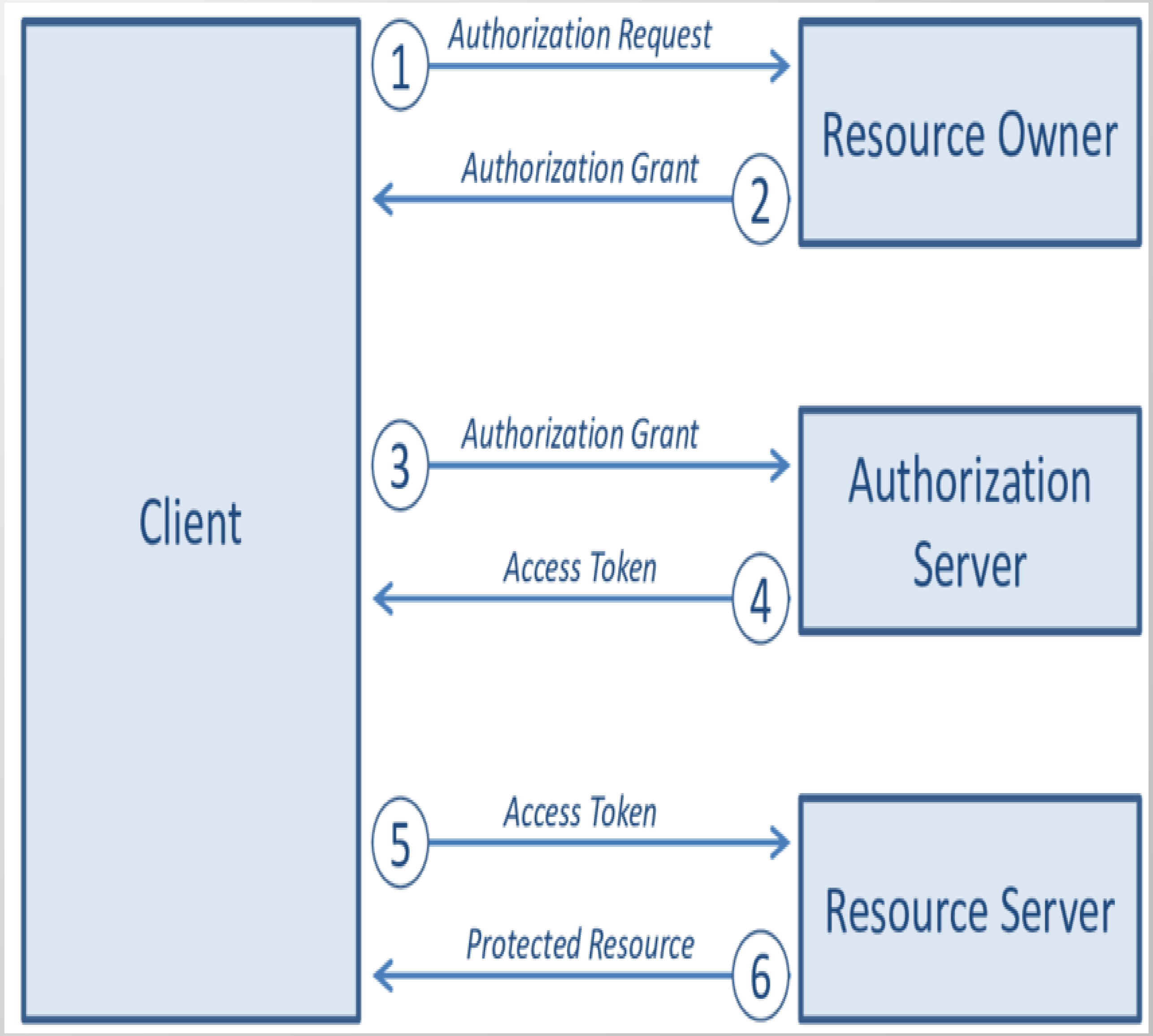
VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret
) ☐secret base64 encoded
```

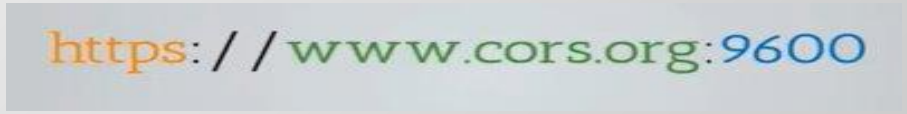
Open Authorization (OAuth)

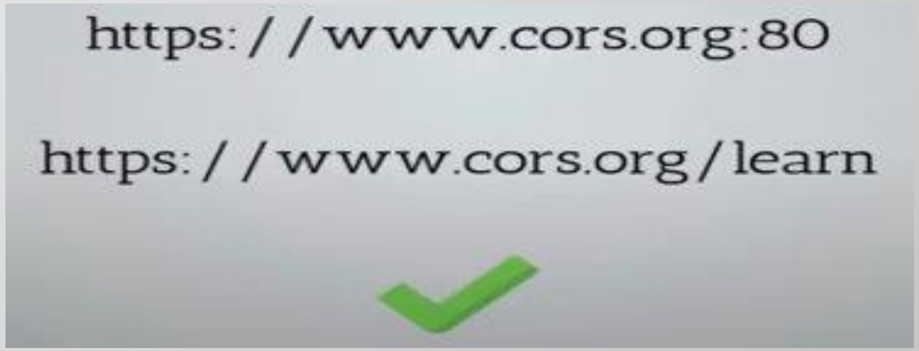
❖ Open Authorization is an advanced version of Token based authorization. Often we use Facebook/Google/Twitter to sign-in to an application. These are the examples of Open Authorization.

1. Spotify (the client) sends a message to user's Facebook account (the Resource Owner) requesting the rights to access his public profile, friend list, email and birthday.
2. User provides Spotify with a grant to collect said data.
3. Spotify sends the grant to a Facebook API.
4. Facebook API verifies grant and sends an access token for Spotify to access protected resources.
5. Spotify sends the access token to another Facebook API given by the authorization server.
6. Facebook API sends the requested data to Spotify.



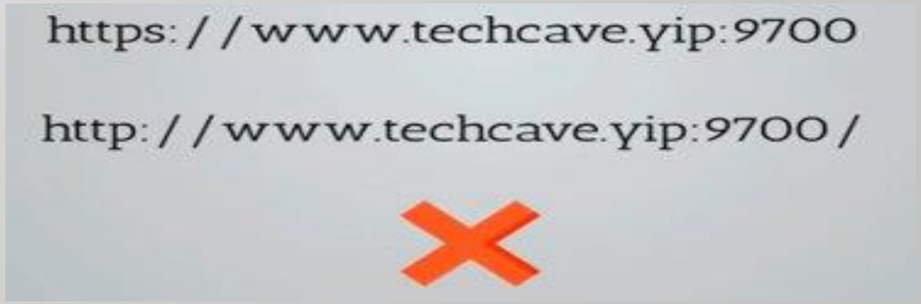
Origin = protocol + domain + port


https://www.cors.org:9600


https://www.cors.org:80

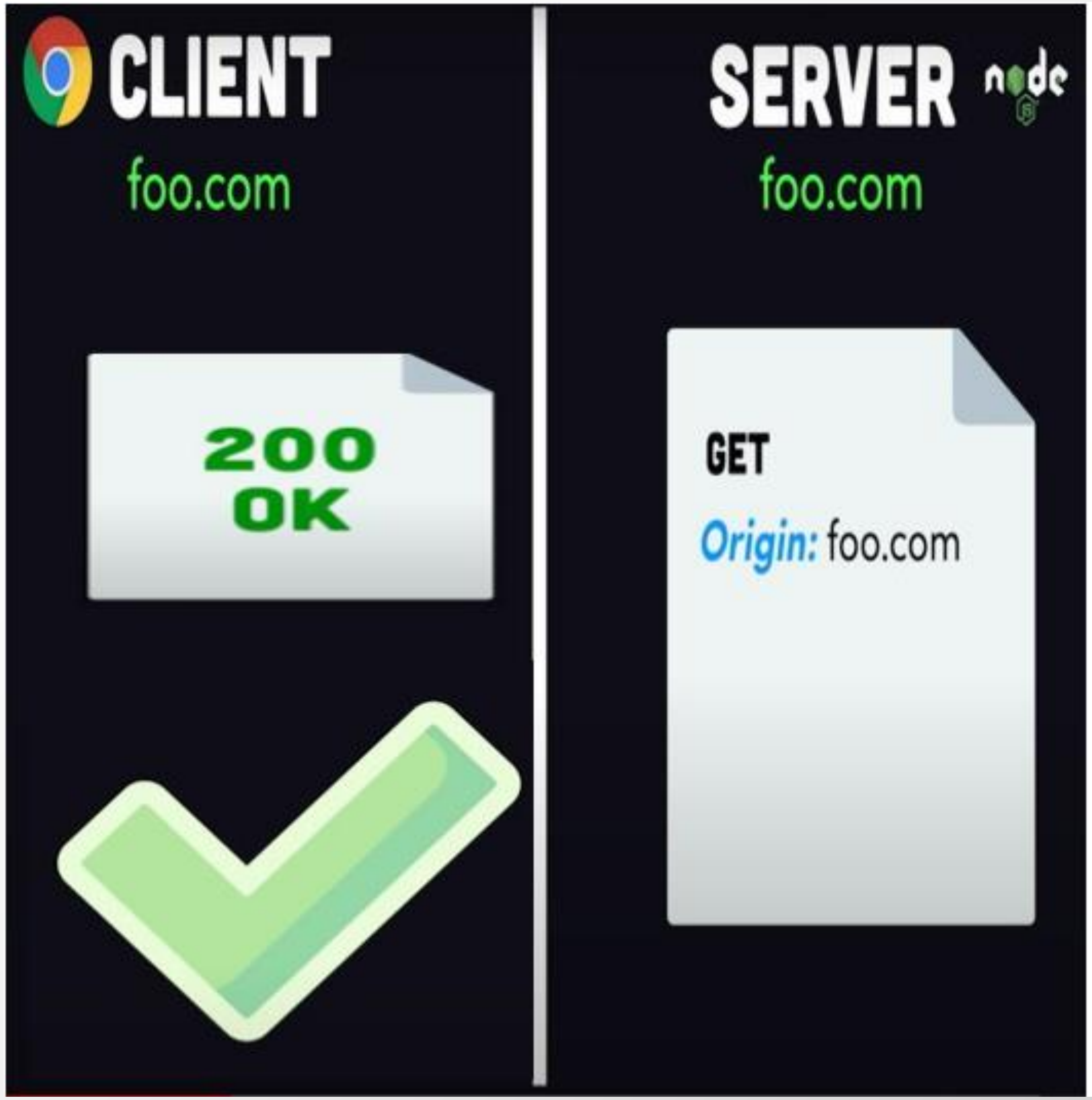
https://www.cors.org/learn

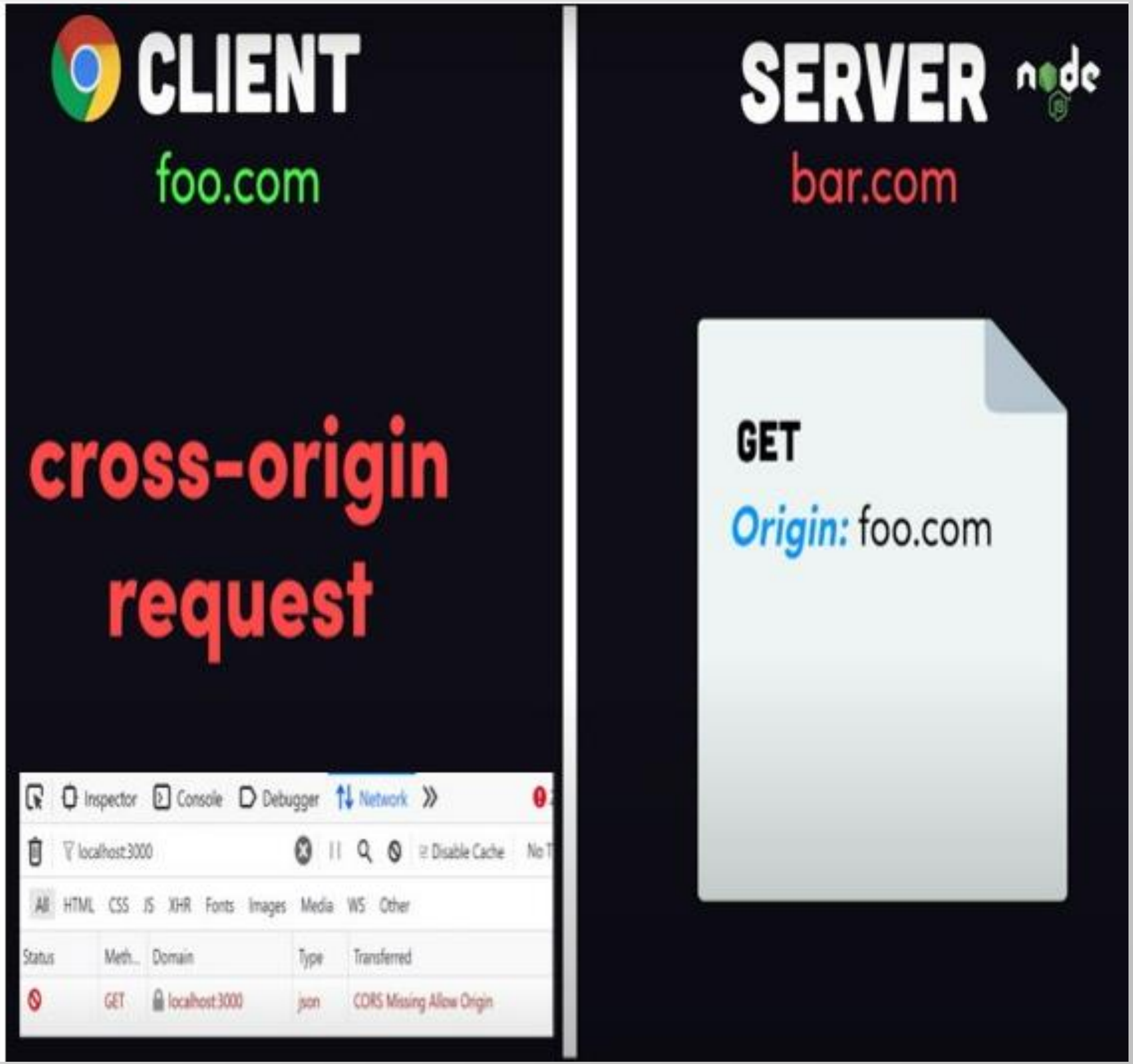
✓


https://www.techcave.yip:9700

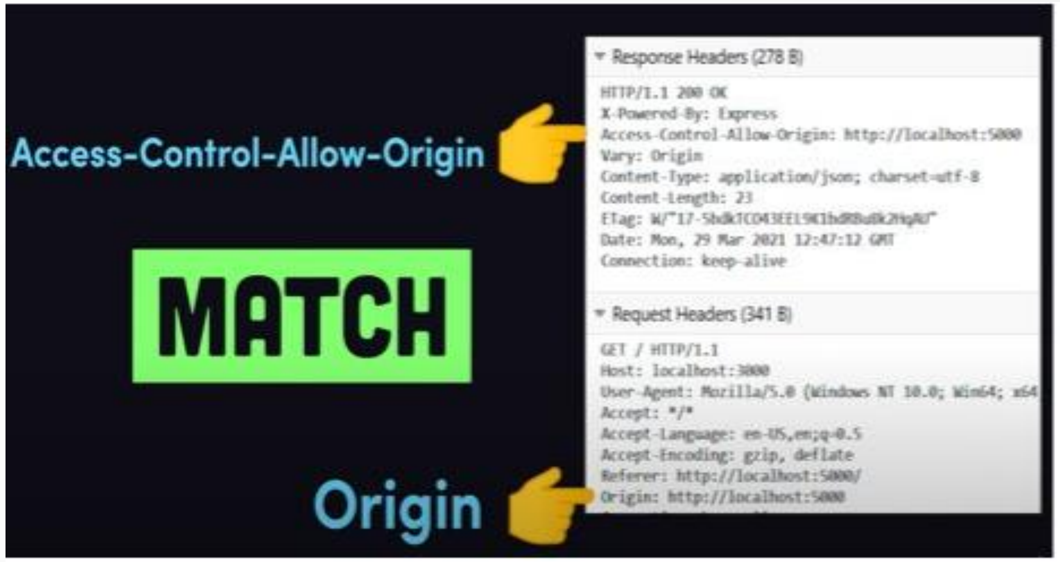
http://www.techcave.yip:9700/

✗





Status	Meth...	Domain	Type	Transferred
🚫	GET	localhost:3000	json	CORS Missing Allow Origin

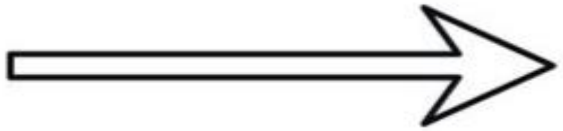


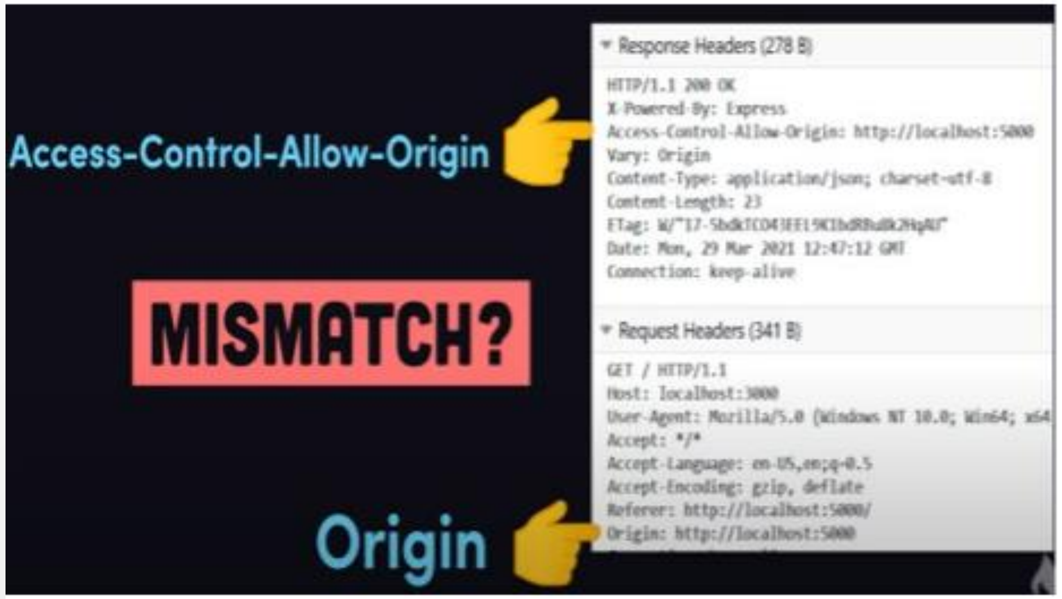
▾ Response Headers (278 B)

```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:5000
Vary: Origin
Content-Type: application/json; charset=utf-8
Content-Length: 23
ETag: W/"17-5bdkTCO43EEL9K1hdRBu0k2HqAU"
Date: Mon, 29 Mar 2021 12:47:12 GMT
Connection: keep-alive
```

▾ Request Headers (341 B)

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Origin: http://localhost:5000
```



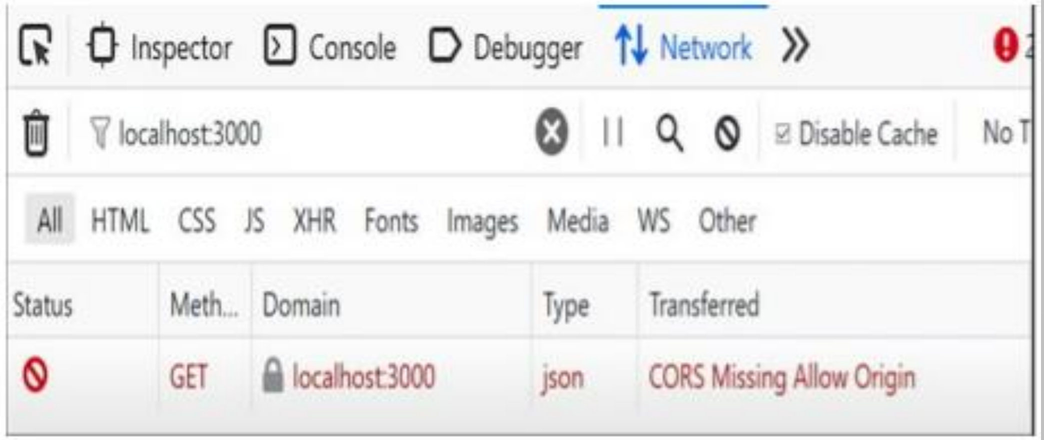


▾ Response Headers (278 B)

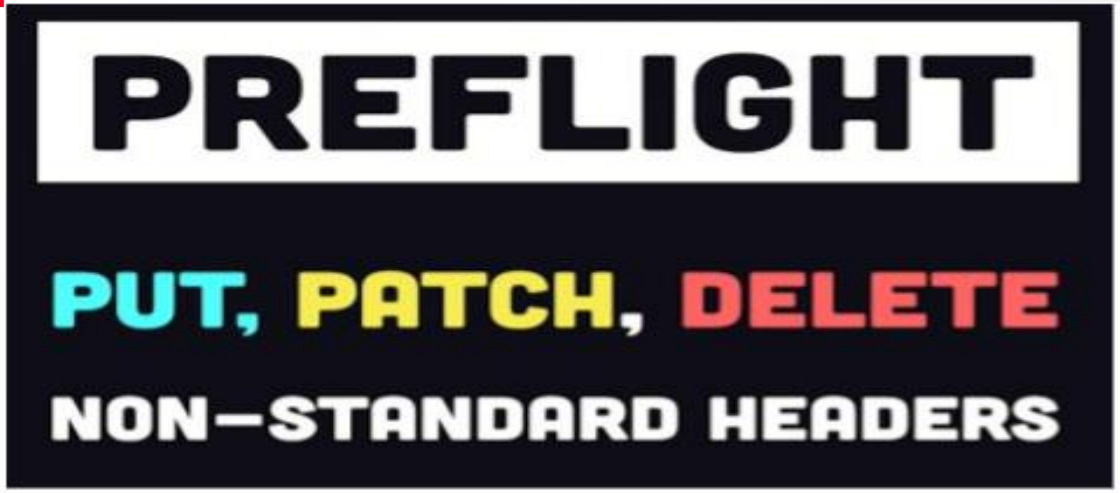
```
HTTP/1.1 200 OK
X-Powered-By: Express
Access-Control-Allow-Origin: http://localhost:5000
Vary: Origin
Content-Type: application/json; charset=utf-8
Content-Length: 23
ETag: W/"17-5bdkTCO43EEL9K1hdRBu0k2HqAU"
Date: Mon, 29 Mar 2021 12:47:12 GMT
Connection: keep-alive
```

▾ Request Headers (341 B)

```
GET / HTTP/1.1
Host: localhost:3000
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64
Accept: */*
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost:5000/
Origin: http://localhost:5000
```

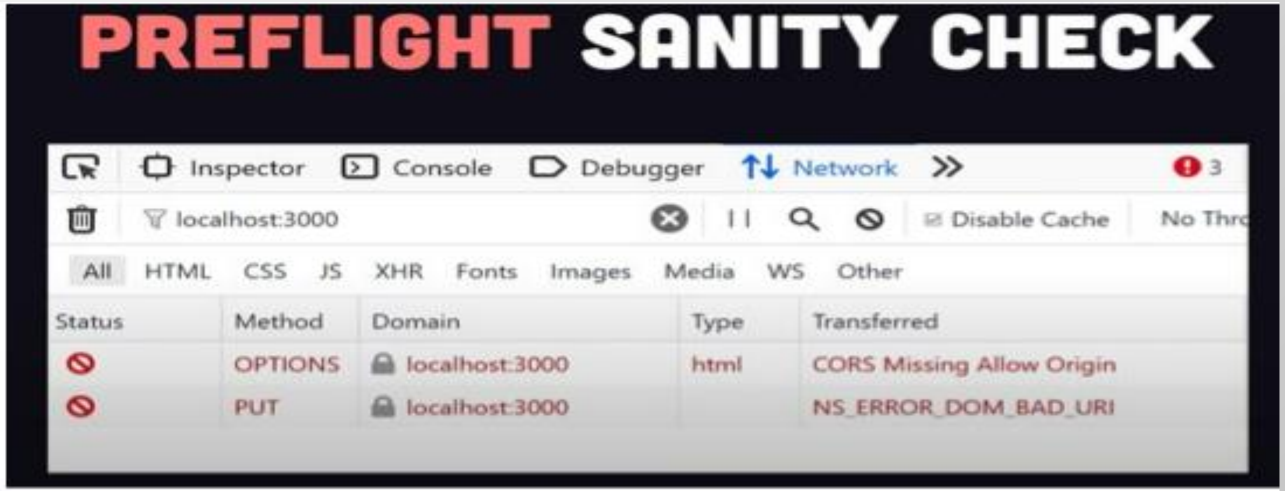


Status	Meth...	Domain	Type	Transferred
🚫	GET	localhost:3000	json	CORS Missing Allow Origin

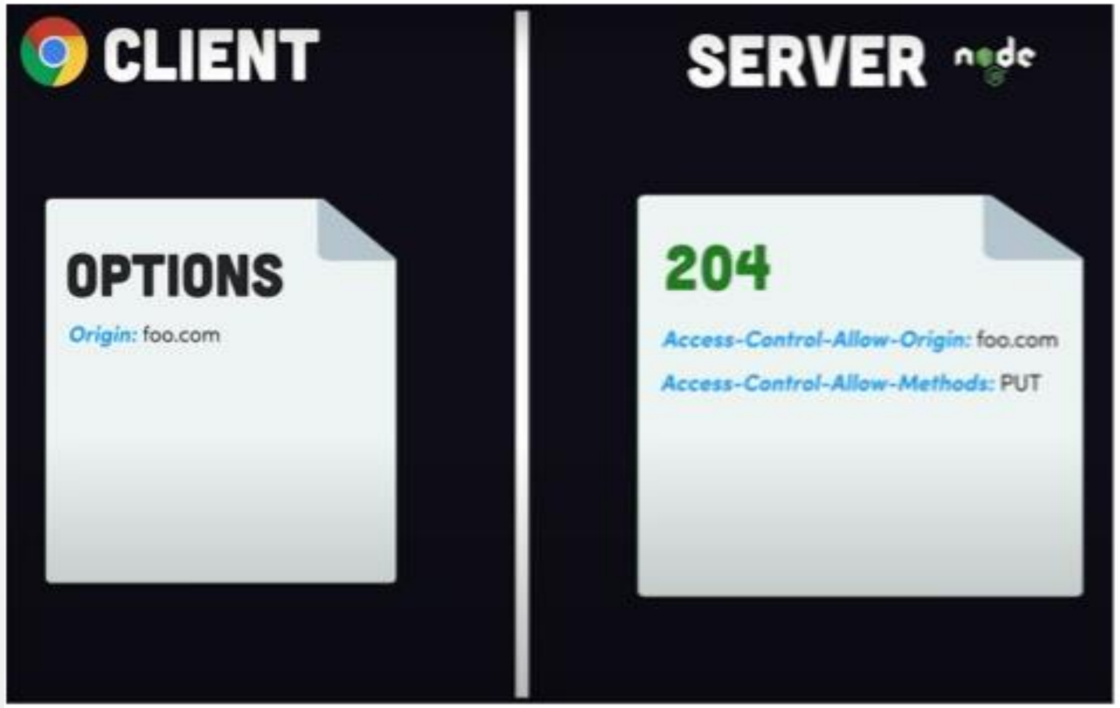

PREFLIGHT

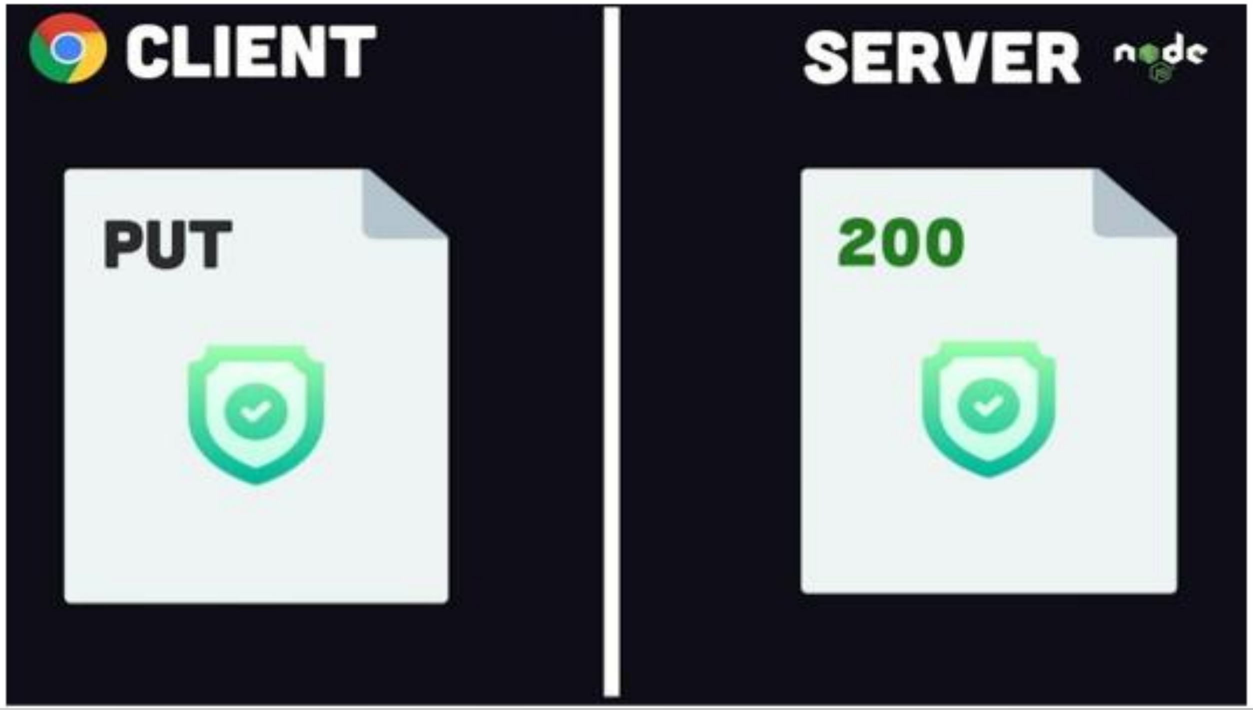
PUT, PATCH, DELETE

NON-STANDARD HEADERS


PREFLIGHT SANITY CHECK

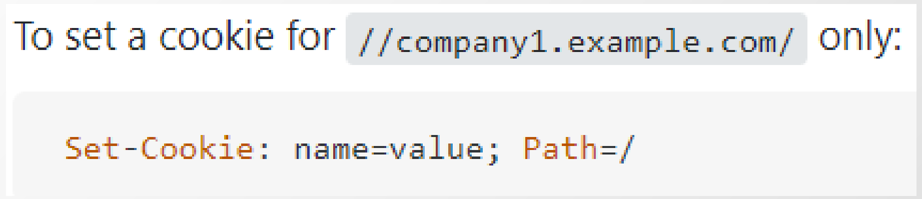
Status	Method	Domain	Type	Transferred
🚫	OPTIONS	🔒 localhost:3000	html	CORS Missing Allow Origin
🚫	PUT	🔒 localhost:3000		NS_ERROR_DOM_BAD_URI


CLIENT
SERVER
node
OPTIONS
Origin: foo.com
204
Access-Control-Allow-Origin: foo.com
Access-Control-Allow-Methods: PUT



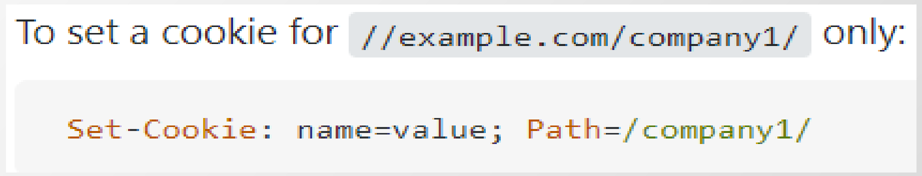
Cookies

- Cookies are small files of information that a web server generates and sends to a web browser.
- Web browsers store the cookies they receive for a predetermined period of time, They attach the relevant cookies to any future requests the user makes of the web server.
- A Cookie-based authentication uses the HTTP cookies to authenticate the client requests and maintain session information on the server over the stateless HTTP protocol. (Used for state management)



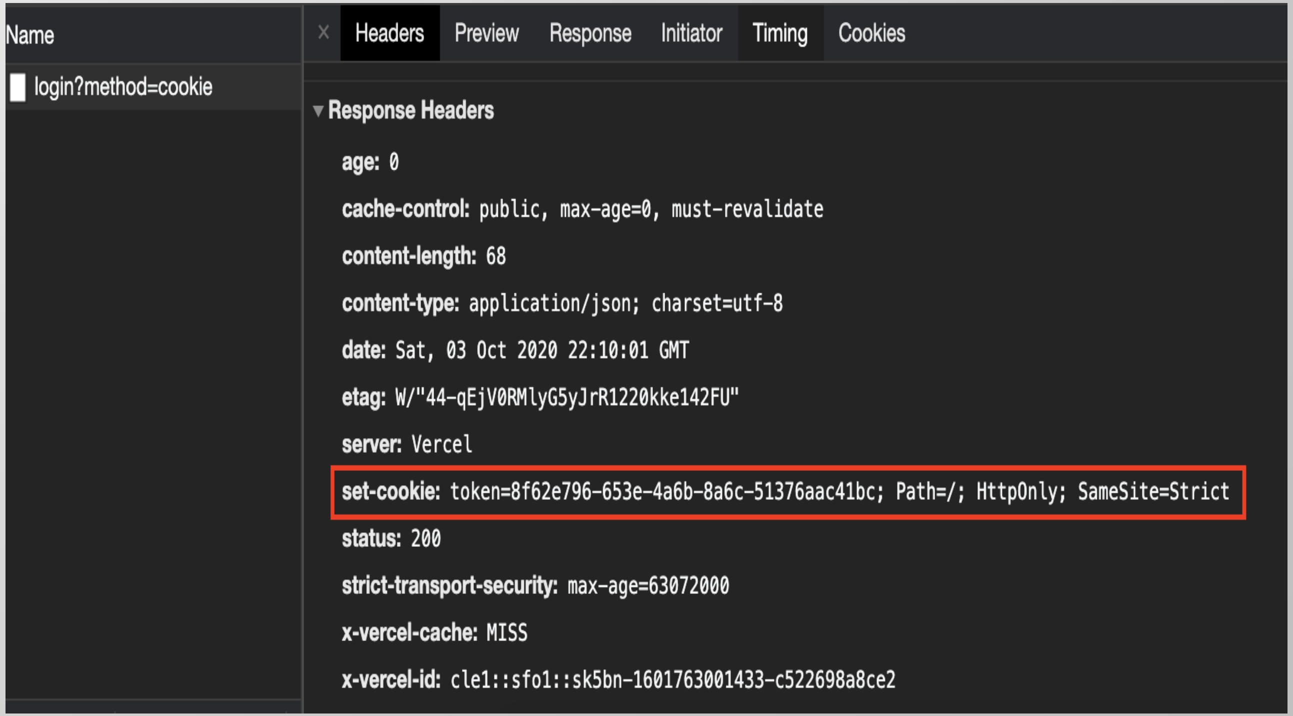
To set a cookie for `//company1.example.com/` only:

```
Set-Cookie: name=value; Path=/
```



To set a cookie for `//example.com/company1/` only:

```
Set-Cookie: name=value; Path=/company1/
```



▾ Response Headers

```
age: 0
cache-control: public, max-age=0, must-revalidate
content-length: 68
content-type: application/json; charset=utf-8
date: Sat, 03 Oct 2020 22:10:01 GMT
etag: W/"44-qEjV0RMlyG5yJrR1220kke142FU"
server: Vercel
set-cookie: token=8f62e796-653e-4a6b-8a6c-51376aac41bc; Path=/; HttpOnly; SameSite=Strict
status: 200
strict-transport-security: max-age=63072000
x-vercel-cache: MISS
x-vercel-id: cle1::sfo1::sk5bn-1601763001433-c522698a8ce2
```

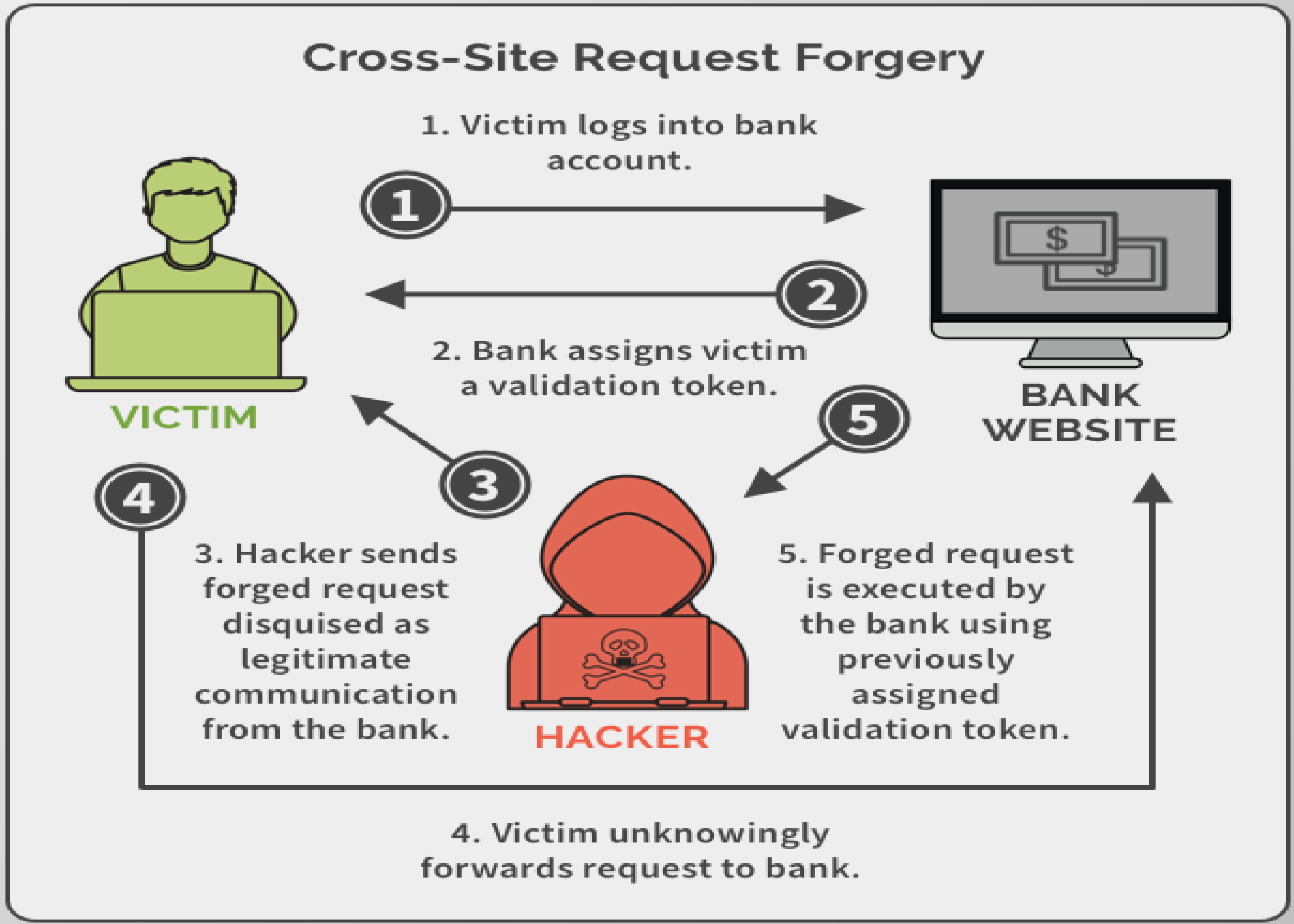
CSRF Attack

Cross-Site Request Forgery (CSRF) is an attack that forces authenticated users to submit a request to a Web application against which they are currently authenticated.

Using various social engineering methods attacker tricks victim to load malicious url.(example sending a malicious url through email)

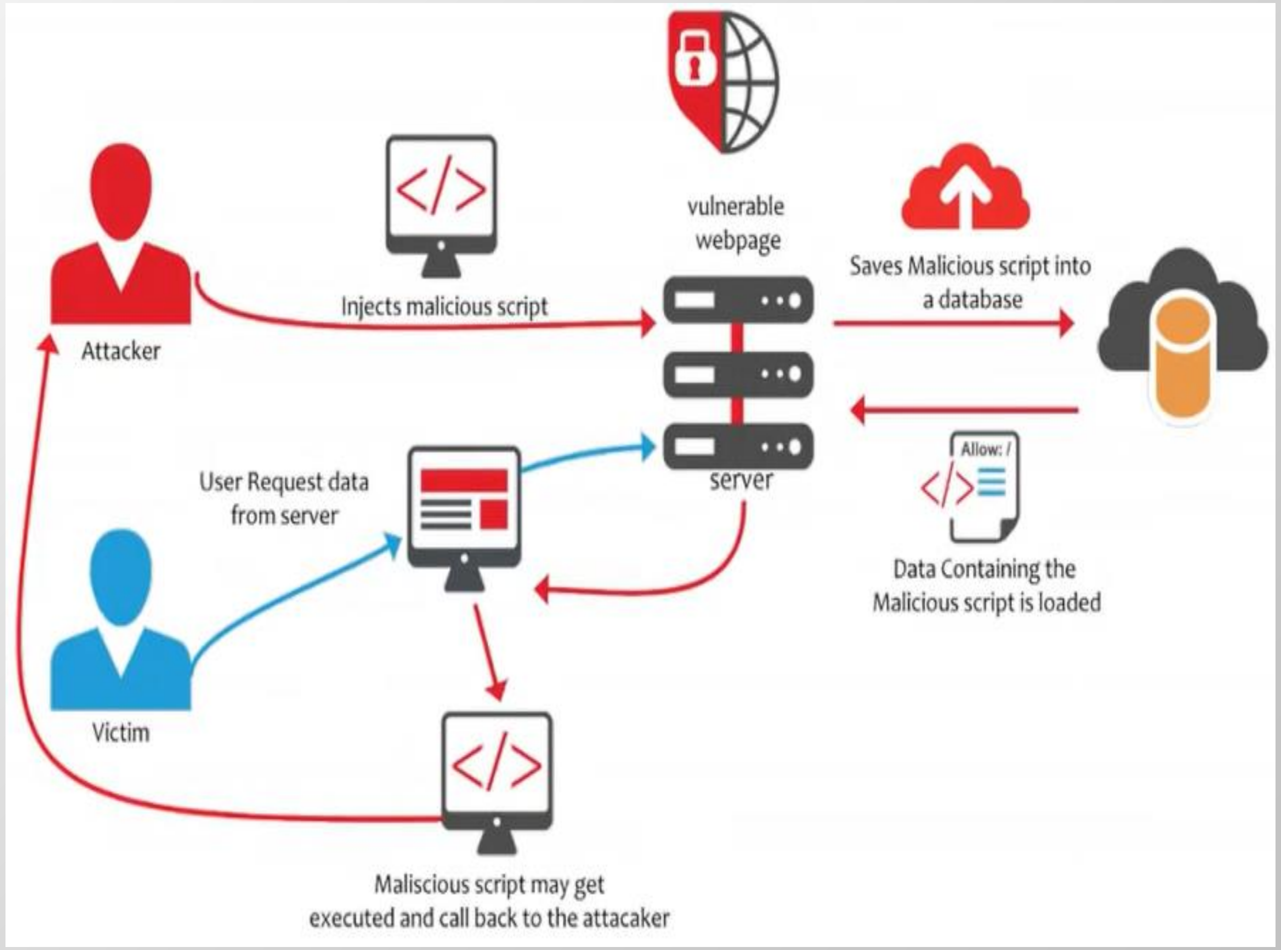
To carry out a successful CSRF attack, consider the following:

- The attack will only be successful if the user is in an active session with the vulnerable application.
- An attacker also needs to find the right values for the URL parameters. Otherwise, the target application might reject the malicious request.


Cross-Site Request Forgery
1. Victim logs into bank account.
1
2
2. Bank assigns victim a validation token.
VICTIM
BANK WEBSITE
$
3
4
5
3. Hacker sends forged request disquised as legitimate communication from the bank.
HACKER
5. Forged request is executed by the bank using previously assigned validation token.
4. Victim unknowingly forwards request to bank.

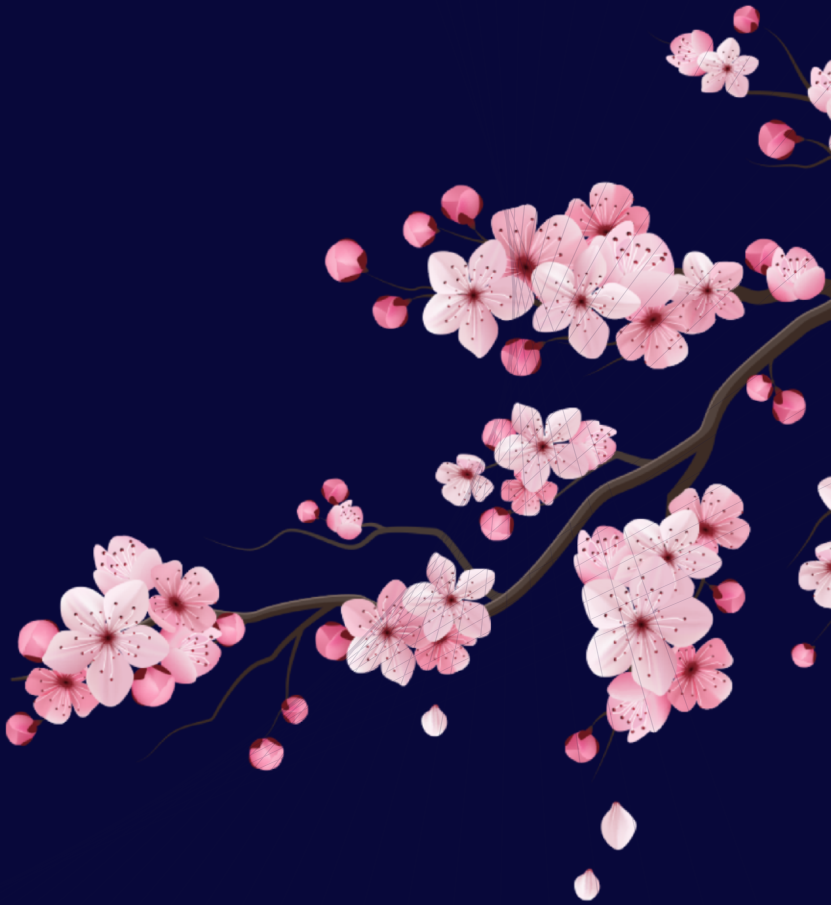
XSS Attack

- Cross site scripting (XSS) is an attack in which an attacker injects malicious executable scripts into the code of a trusted application or website.
- The main difference between XSS and CSRF is in XSS attack inject the malicious code in client application which may steal the victim's information But, in the case of CSRF the attacker forge the request and uses the victim's credentials without stealing them.


vulnerable webpage
Attacker
Injects malicious script
Saves Malicious script into a database
Allow: /
server
User Request data from server
Data Containing the Malicious script is loaded
Victim
Maliscious script may get executed and call back to the attacaker

techigai

Thank You



ADDRESS

Headquarters:

7924 Preston Rd, Suite 350,Plano
TX 75024 USA

Development Center:

Wing B, 9th Floor, Aurobindo Galaxy, Opposite IKEA,
Raidurg, Hyderabad, Telangana 500019

Email:

info@techigai.io