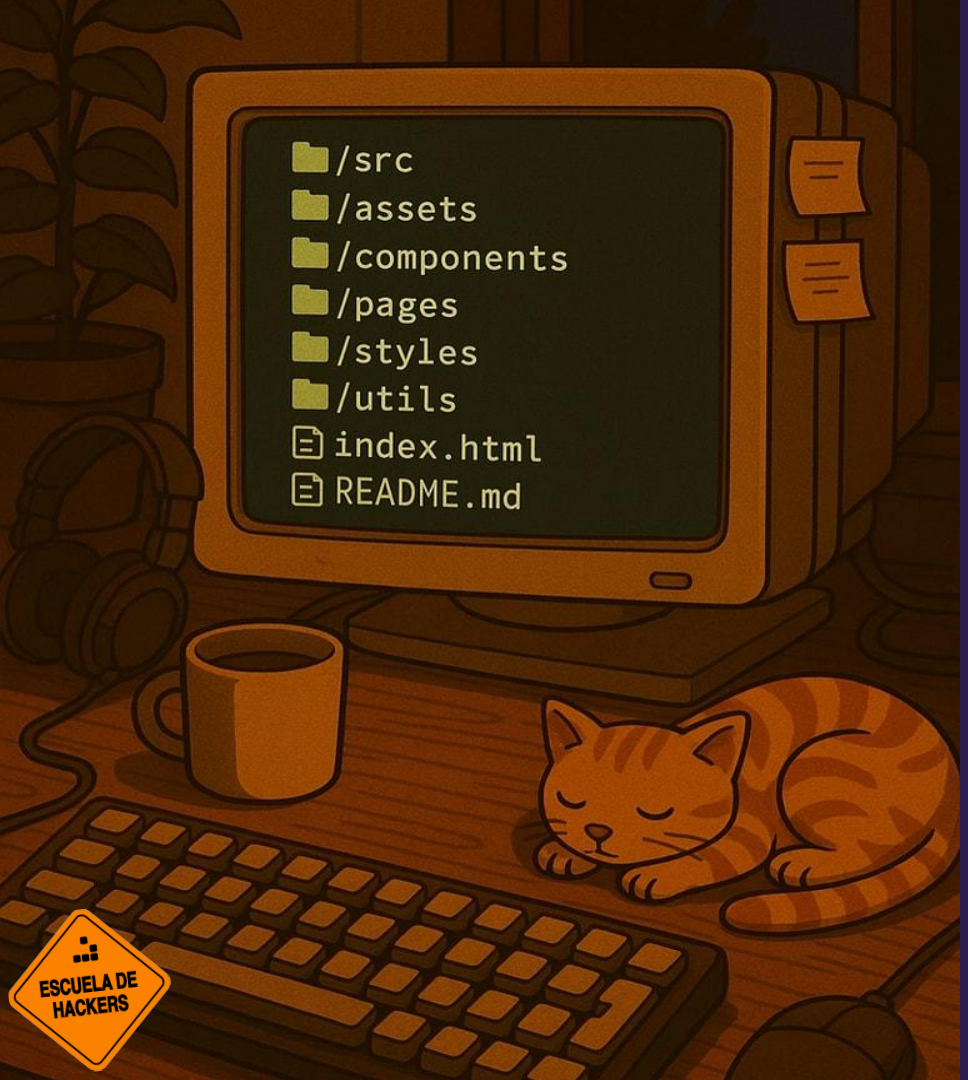
Arquitectura Cliente - Servidor

DEV.F
DESARROLLAMOS(PERSONAS);

Elaborado por: Jesua Luján
Jesua Hadai Luján

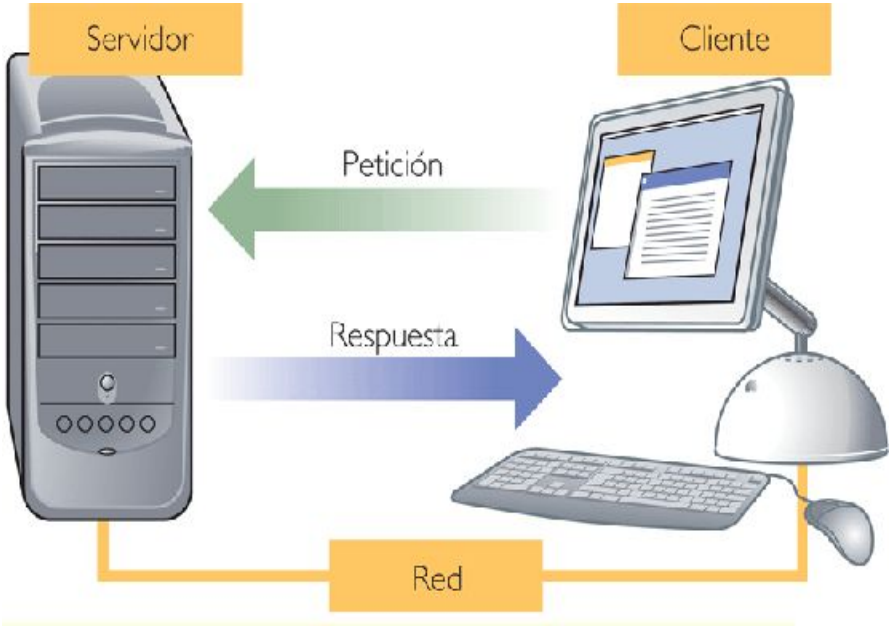
{DEVF}



Objetivos de la sesión 🔥

- Entender qué es la arquitectura Cliente - Servidor & protocolo HTTP🤔
- Entender la arquitectura REST API (status code, peticiones, req, res, etc.) 👩‍💻
- Ver los diferentes Status Code y la importancia de ellos. 😈
- Aprender a usar herramientas para hacer peticiones a una API y así mismo consumir datos de una API.

DEV.EX

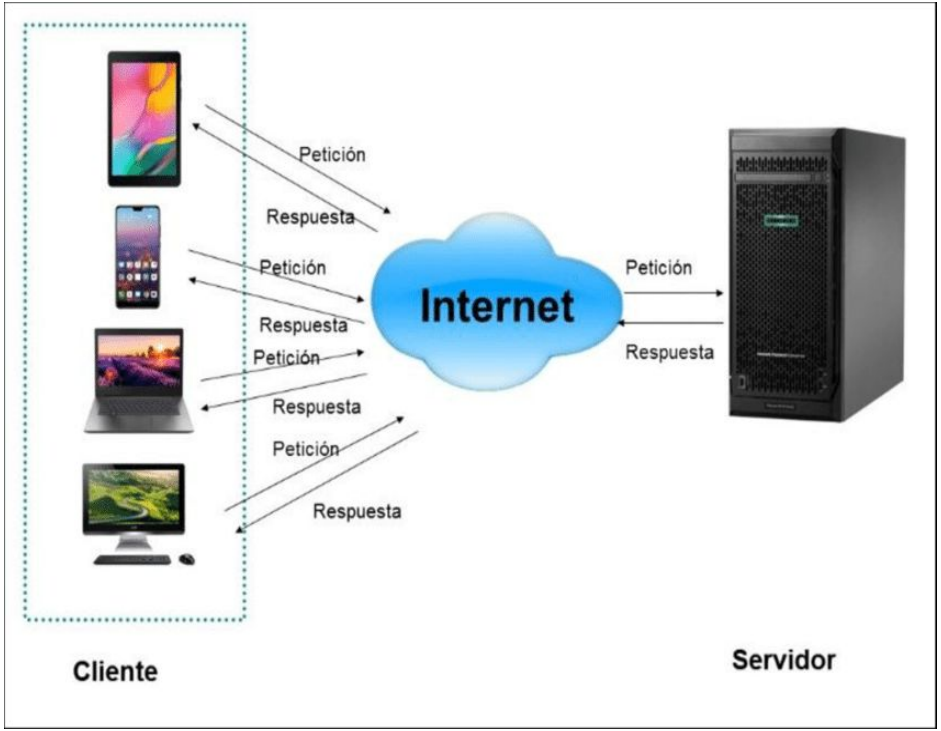


🧭 Guía para Principiantes: Cliente-Servidor, HTTP y Status Codes

👩‍💻 1. ¿Qué es la arquitectura cliente-servidor?

imagina un restaurante:

- **Cliente**: tú, con hambre, haciendo un pedido.
- **Servidor**: el chef y el equipo de cocina, recibiendo el pedido y devolviéndole comida (¡esperemos que caliente!).



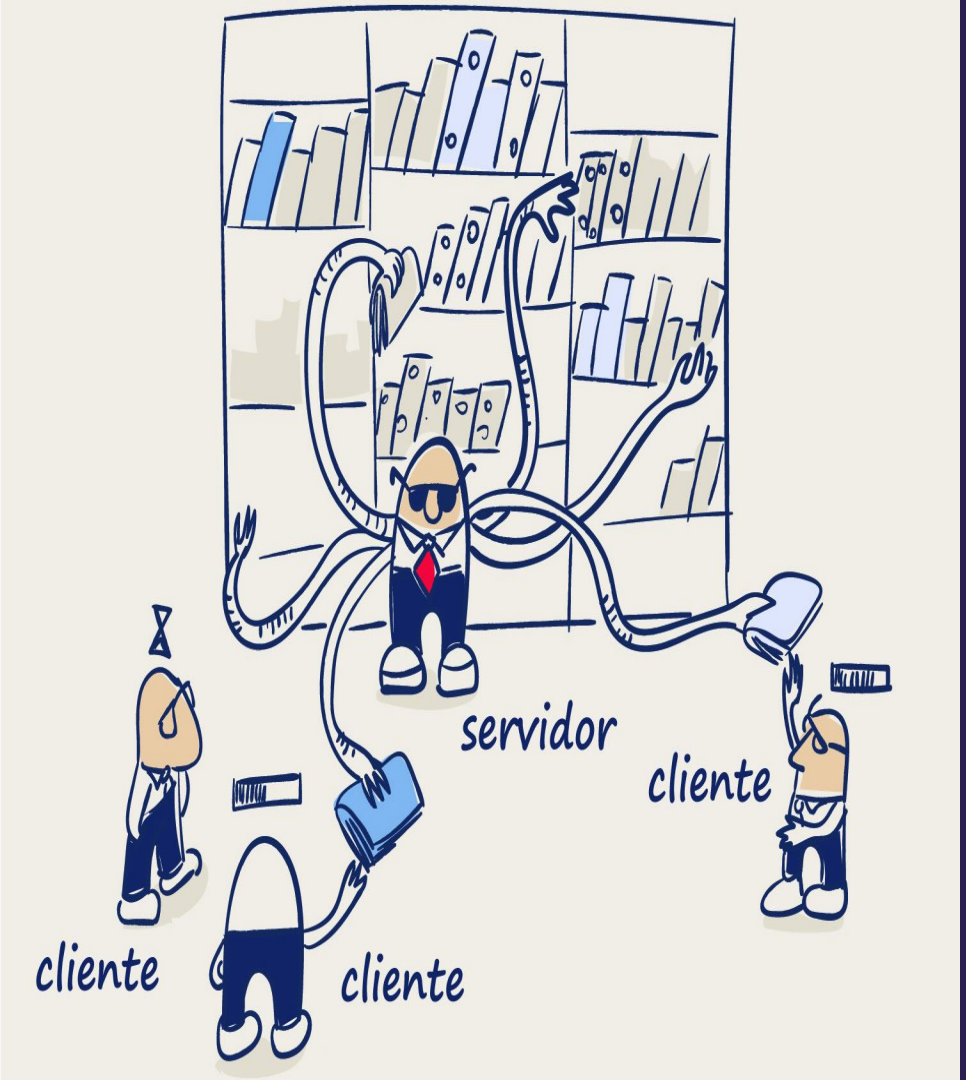
🧭 Guía para Principiantes: Cliente-Servidor, HTTP y Status Codes

En el mundo web:

- **Cliente**: normalmente, un navegador (como Chrome o Firefox) o una app.

- **Servidor**: una máquina (física o en la nube) que tiene los datos y la lógica del sistema.

DEV.F



🧭 Guía para Principiantes: Cliente-Servidor, HTTP y Status Codes

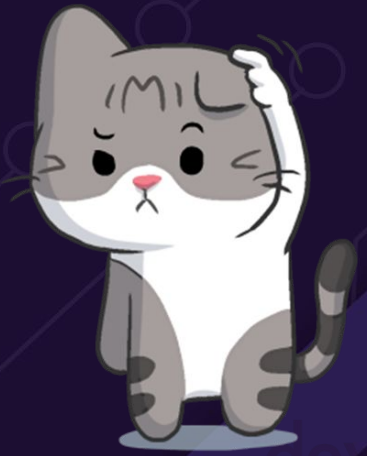
- 📦 El **cliente pide** cosas (páginas, datos, imágenes) y el **servidor responde** con lo que pidió (o con un error, si todo se fue al carajo).

- Esta arquitectura es la base de cómo funciona internet: una comunicación **bidireccional**, pero **iniciada por el cliente**.

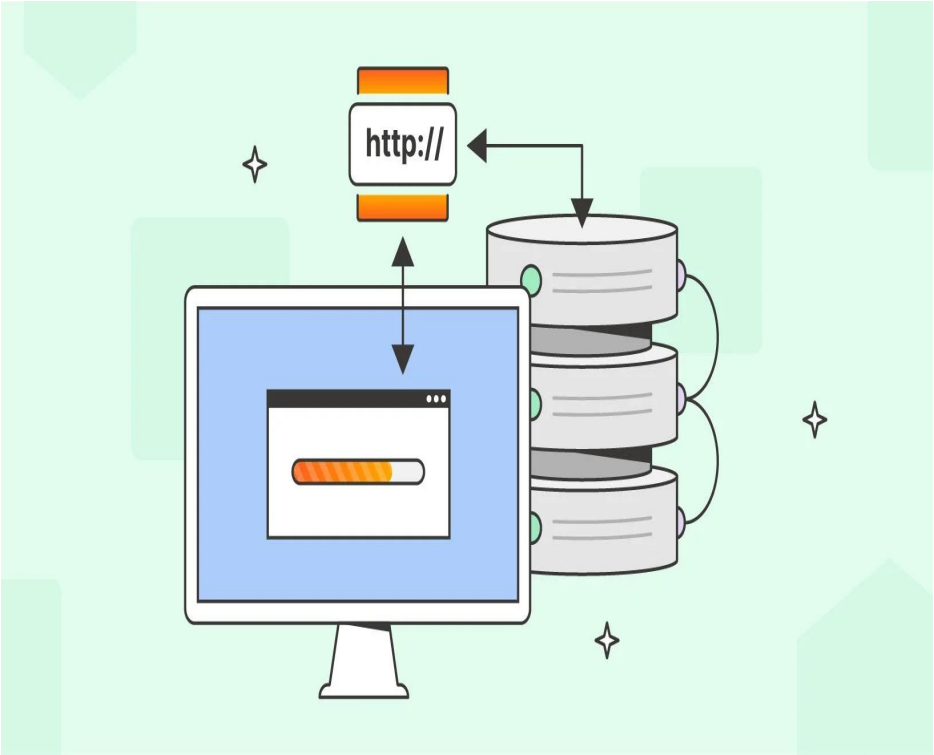
DEV.F

Protocolo HTTP

DEV.F::
DESARROLLAMOS(PERSONAS);



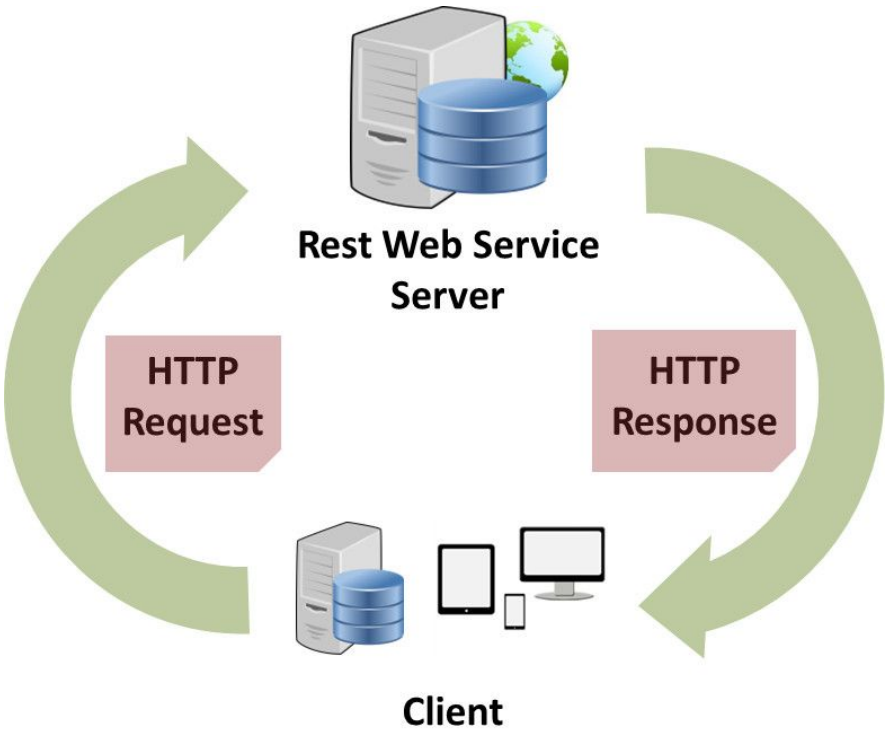
{DEVF}



🌐 2. ¿Qué es el protocolo HTTP?

HTTP (HyperText Transfer Protocol) es el idioma oficial de los navegadores y servidores web. **Es como el "mesero digital"** que lleva los pedidos y trae las respuestas.

DEV.F



🌐 2. ¿Qué es el protocolo HTTP?

HTTP es un protocolo, o un conjunto definido de reglas, para acceder a recursos en la web. Los recursos pueden ser cualquier cosa, desde archivos HTML hasta datos de una base de datos, fotos, texto, etc.

🌐 Protocolo HTTP

🔄 El ciclo típico HTTP:

1. El cliente (ej. tu navegador) hace una **solicitud HTTP (request)**.

2. El servidor procesa esa solicitud.

3. El servidor devuelve una **respuesta HTTP (response)**.

📨 Una solicitud HTTP incluye:

- **Método** (verbo): lo que quieres hacer. Ejemplos:
  - GET: dame algo.
  - POST: aquí te mando algo nuevo.
  - PUT: actualiza esto.
  - DELETE: bórralo.
- **URL**: la dirección de lo que pides.
- **Headers**: metadatos sobre la solicitud (como el idioma o si llevas una sesión iniciada).
- **Body**: solo en algunos métodos (como POST), aquí va el contenido que mandas.

DEV.F

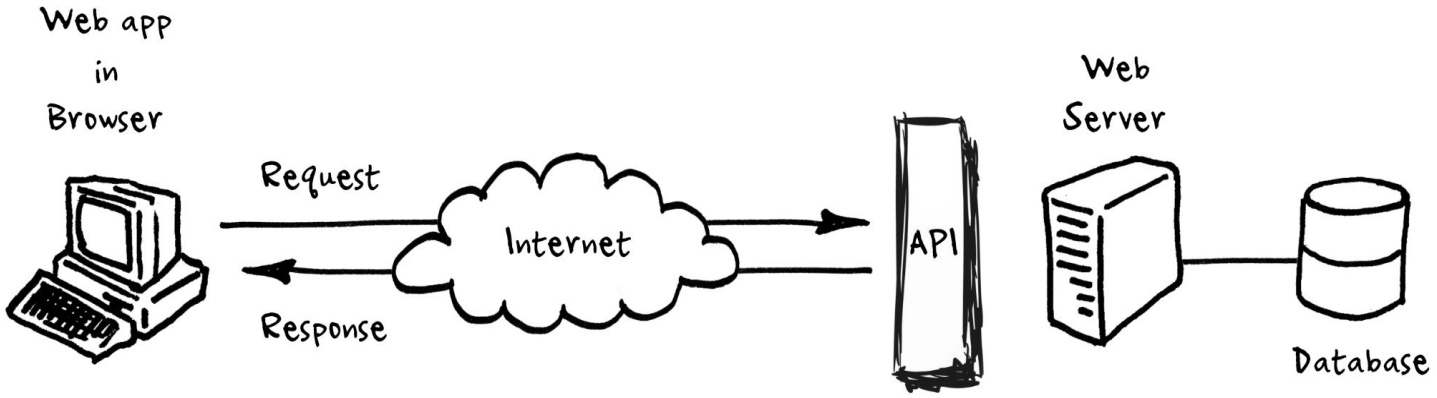
🌐 Protocolo HTTP

🧾 Una respuesta HTTP incluye:

- **Código de estado** (status code): indica si todo salió bien o si el servidor se tropezó con una piedra.
- **Headers**: info sobre la respuesta.
- **Body**: los datos pedidos o un mensaje de error (a veces, un chiste de mal gusto si el backend tiene humor).

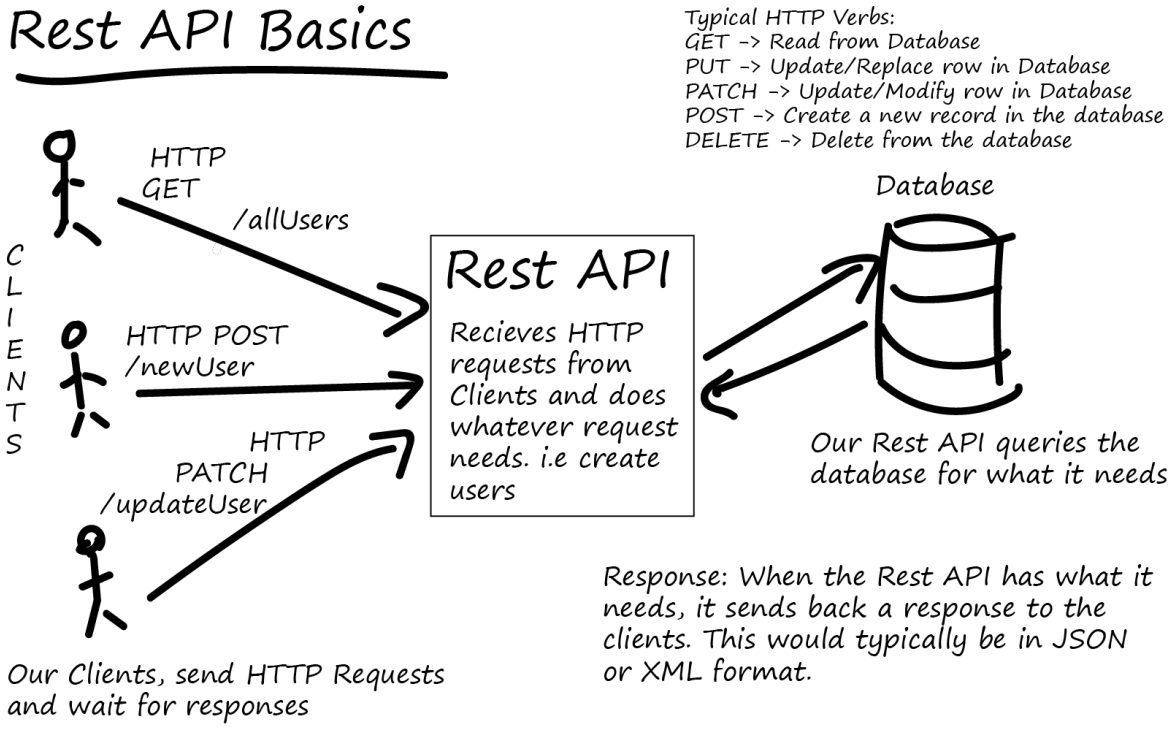
Métodos de petición HTTP

Estos recursos se ponen a nuestra disposición a través de una API y nosotros hacemos peticiones a estas APIs a través del protocolo HTTP. API significa interfaz de programación de aplicaciones. Es el mecanismo que permite a los desarrolladores solicitar recursos.



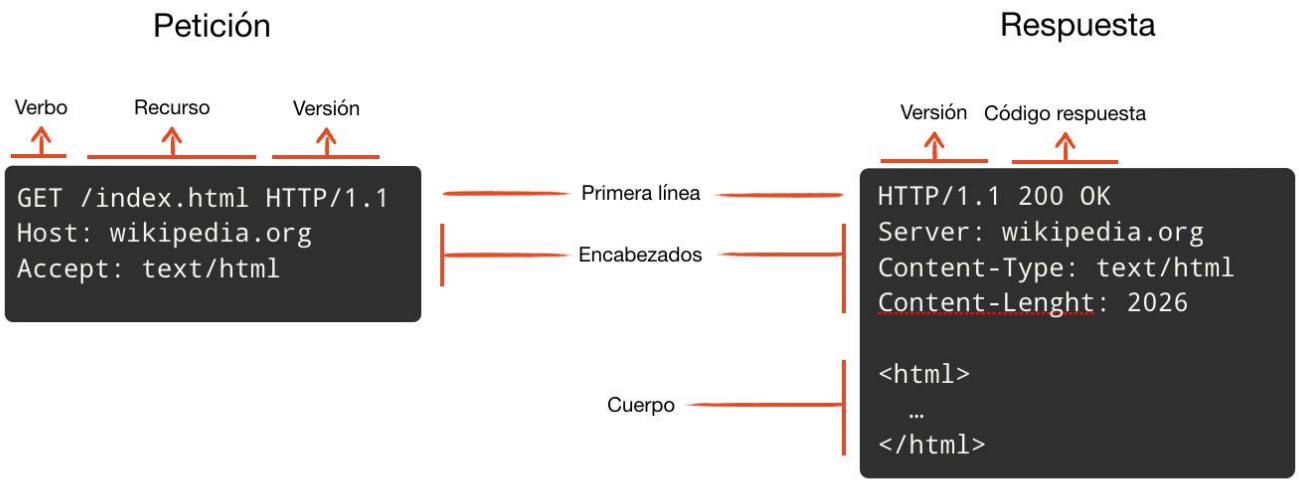
Mensajes HTTP

Los mensajes HTTP, son los medios por los cuales se intercambian datos entre servidores y clientes.



Petición - Respuesta

La línea de inicio y las cabeceras HTTP, del mensaje, son conocidas como la cabeza de las peticiones, mientras que su contenido en datos se conoce como el cuerpo del mensaje.




```
$ node index.js
Started at 3000
GET / 200 11.721 ms - 19
GET /favicon.ico 404 1.627 ms - 150
```

Hay paquetes como morgan (logger) que nos permiten
ver esto mismo en consola

Method Path Protocol version

```
GET / HTTP/1.1
Host: developer.mozilla.org
Accept-Language: fr
```

Headers

DEV.F

Protocol version Status code Status message

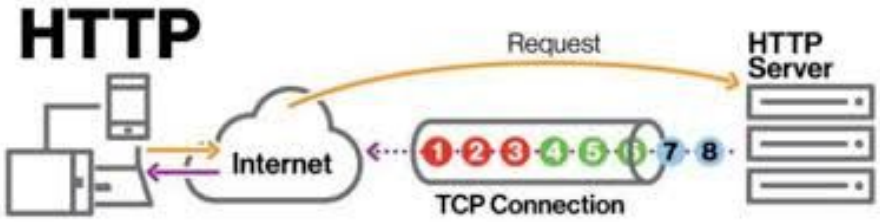
```
HTTP/1.1 200 OK

date: Tue, 18 Jun 2024 10:03:55 GMT
cache-control: public, max-age=3600
content-type: text/html
```

Headers

DEV.F

Respuesta enviada por el servidor

```
HTTP/1.1 200 OK
Date: Sat, 09 Oct 2010 14:28:02 GMT
Server: Apache
Last-Modified: Tue, 01 Dec 2009 20:18:22 GMT
ETag: "51142bc1-7449-479b075b2891b"
Accept-Ranges: bytes
Content-Length: 29769
Content-Type: text/html
```



Respuesta enviada por el servidor

Request

Pretty | Raw | Hex | \n | ≡

```
1  POST / HTTP/1.1
2  Host: 127.0.0.1:3003
3  Content-Length: 22
4  X-Http-Method-Override: DELETE
5
6
7
8
9
10
11
12
```

Method Override

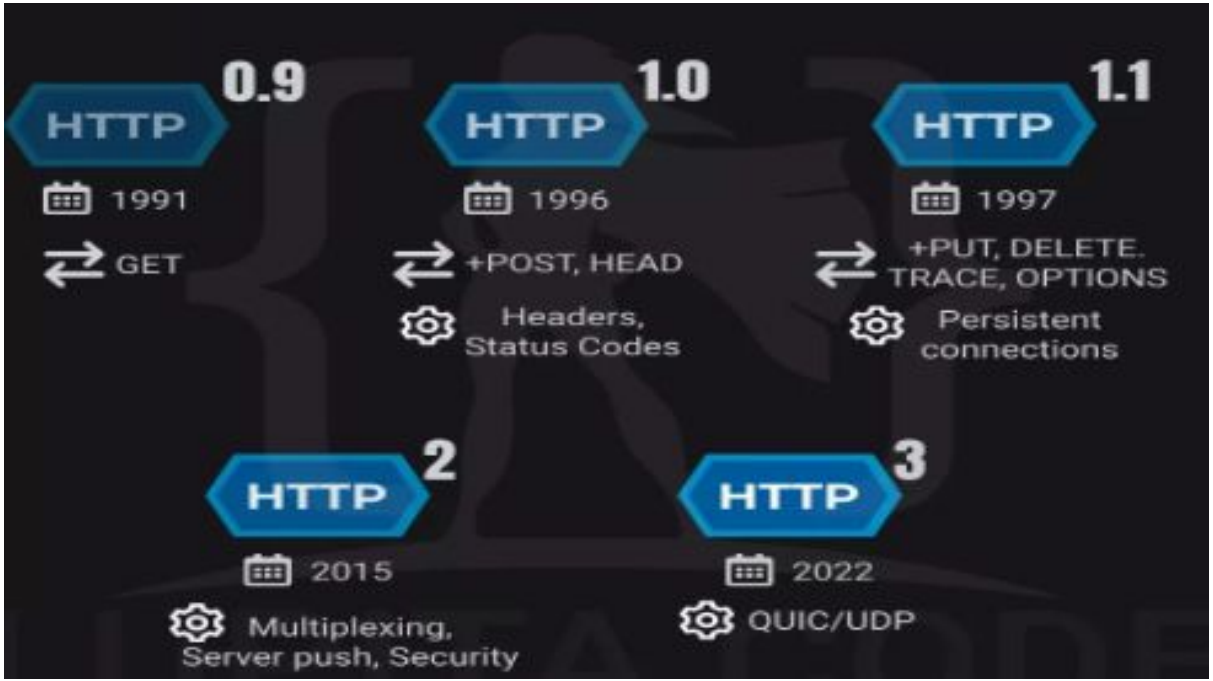
Response

Pretty | Raw | Hex | Render | \n | ≡

```
1  HTTP/1.1 200 OK
2  Server: nginx/1.18.0 (Ubuntu)
3  Date: Wed, 17 Nov 2021 21:05:21 GMT
4  Content-Type: text/html; charset=utf-8
5  Content-Length: 22
6  Connection: keep-alive
7  X-Powered-By: Express
8  Vary: X-Http-Method-Override
9  ETag: W/"16-HFdPLkDY9lV5KEoGV4Tqgy/AQpw"
10 Allow: GET, POST, HEAD
11
12 DELETE method received
```

DEV.F

HTTP VERSIONS

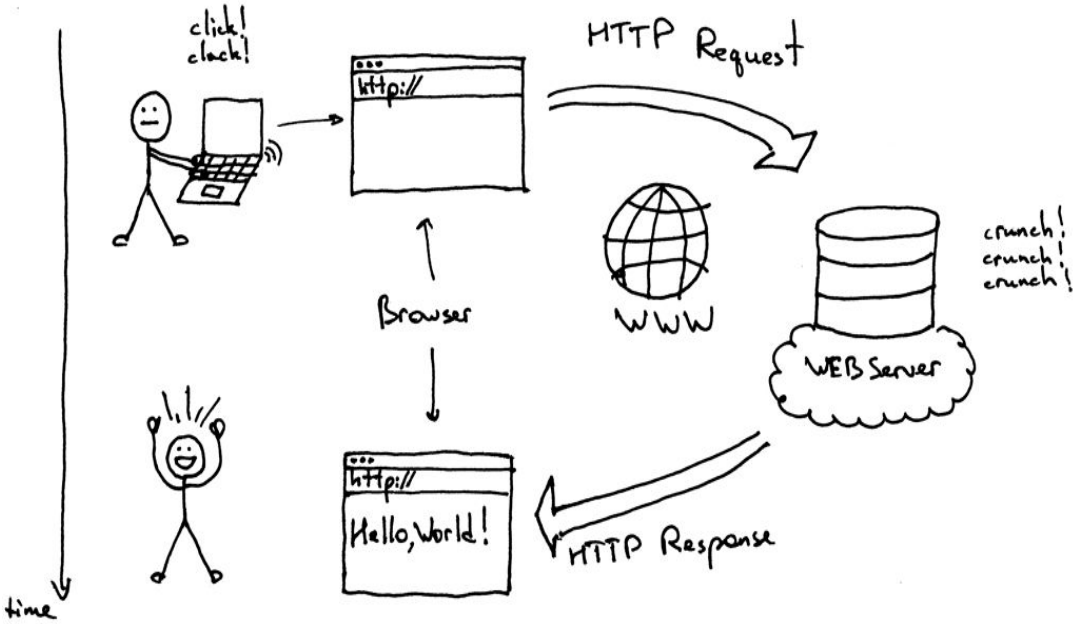


http es el protocolo en el que se basa la web

Repaso ARQ. CLIENTE - SERVIDOR

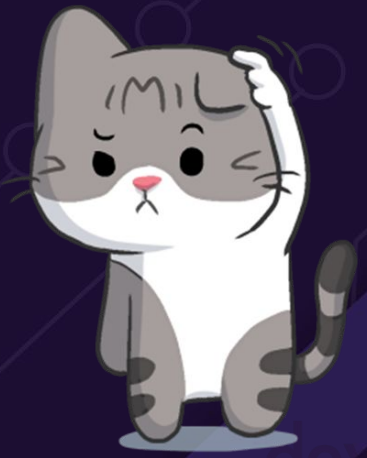
Arquitectura cliente-servidor

Para entender los métodos HTTP, es importante cubrir el concepto de arquitectura cliente-servidor. Esta arquitectura describe cómo funcionan todas las aplicaciones web y define las reglas de su comunicación.

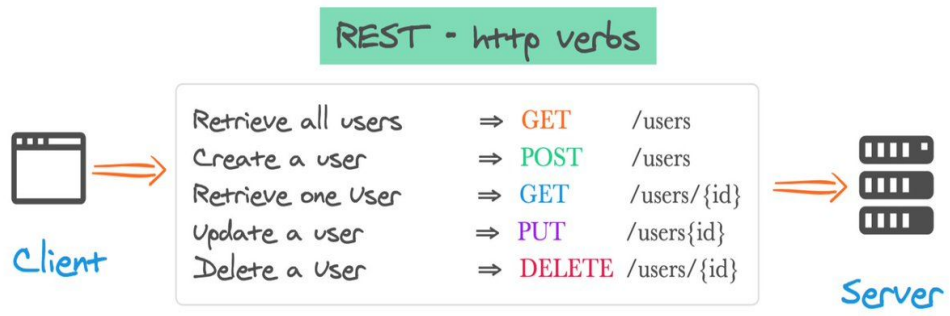


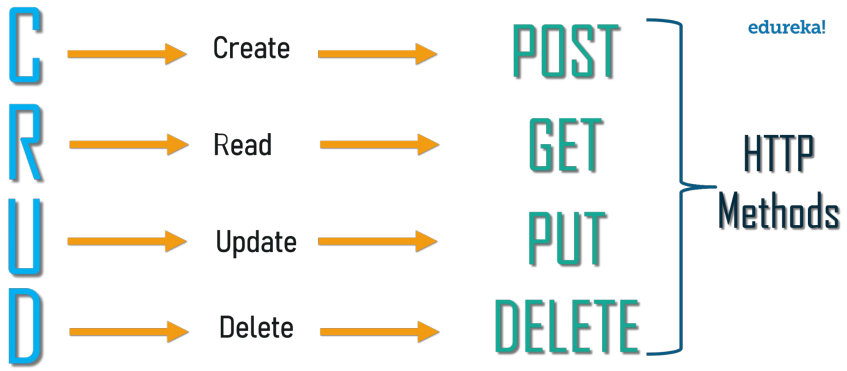
Verbos HTTP

DEV.F
DESARROLLAMOS(PERSONAS);



{DEVF}





VERBOS HTTP

La primera línea de un mensaje de petición empieza con un verbo (también se le conoce como método). Los verbos definen la acción que se quiere realizar sobre el recurso.

Los verbos más comunes son:

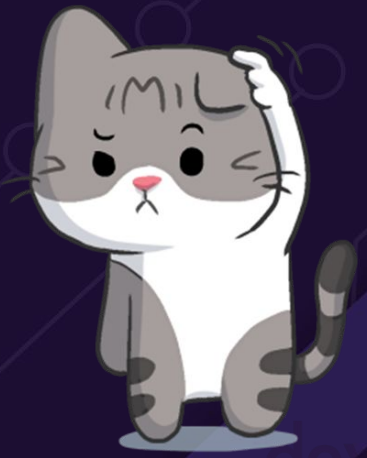
- **GET:** Solicitar un recurso o varios.
- **POST:** Publicar un recurso.
- **PUT:** Reemplazar un recurso.
- **DELETE:** Eliminar un recurso.
- **PATCH:** Actualizar un recurso

Nota: Cuando ingresas a una página desde un navegador, por debajo el navegador envía un mensaje GET, lo mismo cuando oprimes un vínculo a otra página.

DEV.F

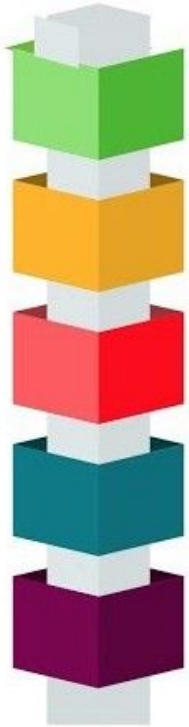
Códigos de Respuesta (Status Code)

DEV.F
DESARROLLAMOS(PERSONAS);



{DEVF}

HTTP Status Codes



1XX
INFORMATIONAL

2XX
SUCCESS

3XX
REDIRECTION

4XX
CLIENT ERROR

5XX
SERVER ERROR

¿Qué Código de estado debe devolver el servidor?

Código de Estado	Descripción
200	OK: La solicitud ha tenido éxito.
201	Created: La solicitud ha tenido éxito y se ha creado un recurso nuevo.
204	No Content: La solicitud ha tenido éxito pero no devuelve contenido.
400	Bad Request: La solicitud es incorrecta o malformada.
401	Unauthorized: Se requiere autenticación para acceder al recurso.
403	Forbidden: No tienes permiso para acceder al recurso.
404	Not Found: El recurso solicitado no se ha encontrado.
500	Internal Server Error: Error genérico del servidor.
502	Bad Gateway: El servidor recibió una respuesta inválida de otro servidor.
503	Service Unavailable: El servidor no está disponible temporalmente.

Response Codes

La primera línea de un mensaje de respuesta tiene un código de 3 dígitos que le indica al cliente cómo interpretar la respuesta, las "XX" hacen referencia a diferentes números entre 00 y 99

Los códigos de respuesta se dividen en cinco categorías dependiendo del dígito con el que inician:

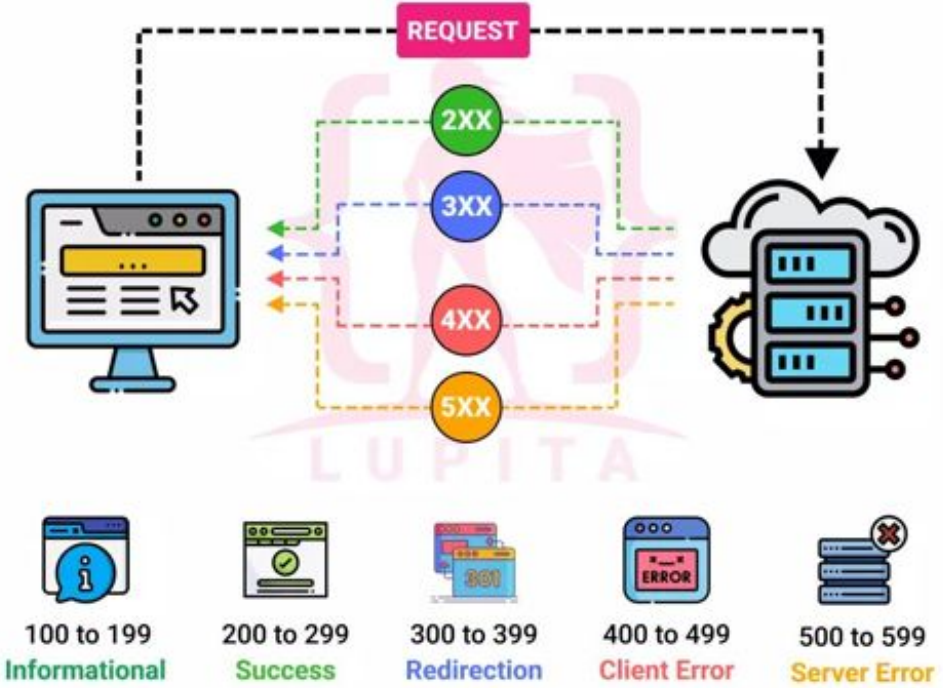
- 1XX: Información
- 2XX: Éxito
- 3XX: Redirección
- 4XX: Error en el cliente
- 5XX: Error en el servidor

¿Recuerdas el famoso error 404?

DEV.F

HTTP STATUS CODES

Son códigos de 3 dígitos enviados del servidor al cliente que indican si la petición del cliente fue exitosa o no.



200 - Ok

201 - Created

301 - Moved Permanently

302 - Found

400 - Bad request

403 - Forbidden

404 - Not Found

429 - Too Many Requests

500 - Internal server error

502 - Bad Gateway

503 - Service Unavailable

504 - Gateway Timeout

🚦 3. Status Codes HTTP más comunes

✅ 1xx – Informativos (rara vez los ves)

- 100 Continue: el servidor recibió parte del mensaje y espera más.

✅ 2xx – Todo bien, capitán

- 200 OK: éxito total.
- 201 Created: se creó algo nuevo (por ejemplo, un nuevo usuario).
- 204 No Content: éxito, pero no hay nada que mostrar.

⚠️ 3xx – Redirecciones

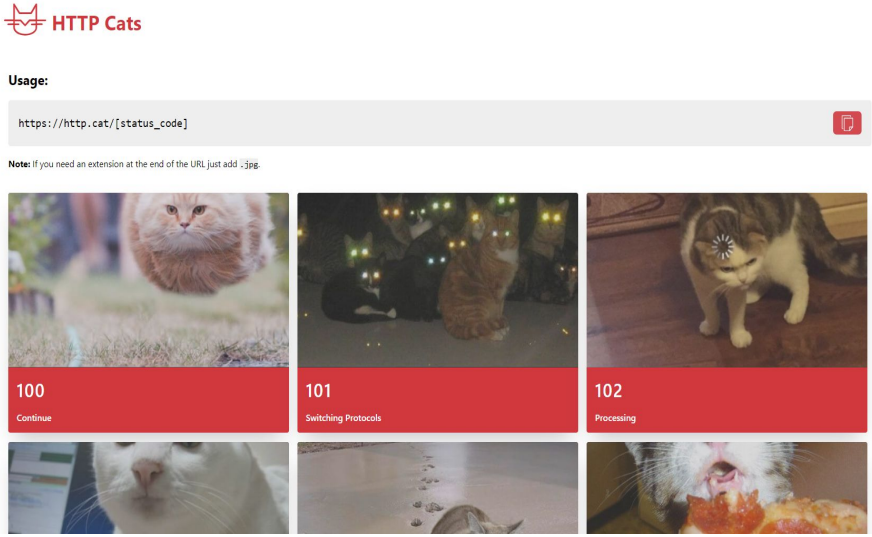
- 301 Moved Permanently: la URL cambió para siempre.
- 302 Found: la URL cambió temporalmente.
- 304 Not Modified: no necesitas recargar, nada cambió.

❌ 4xx – Errores del cliente

- 400 Bad Request: pediste algo mal (tal vez JSON mal formado).
- 401 Unauthorized: necesitas iniciar sesión.
- 403 Forbidden: ni lo intentes, no tienes permiso.
- 404 Not Found: lo que buscas no existe. (El ídolo del rock de los errores).
- 429 Too Many Requests: ¡ey! deja de spammear.

💀 5xx – Errores del servidor

- 500 Internal Server Error: algo explotó en el backend.
- 502 Bad Gateway: el servidor intermedio no pudo hablar con el otro.
- 503 Service Unavailable: el servidor está ocupado o dormido.



https://http.cat/

https://httpstatusdogs.com/

DEV.F

PRO TIP

Los códigos de respuesta son una convención no una regla.

Cada desarrollador puede asignar el código de respuesta que desee a cada petición, pero se recomienda seguir las convenciones (lo más posible)

{DEVF}

Rest API

DEV.F::
DESARROLLAMOS(PERSONAS);

{DEVF}

¿Qué es Rest?

REST es una interfaz para conectar varios sistemas basados en el protocolo HTTP y nos sirve para obtener, generar datos y operaciones, devolviendo esos datos en formatos muy específicos, como XML y JSON.

- Protocolo cliente/servidor "SIN ESTADO"
- Utiliza verbos http
  - GET (leer)
  - POST (crear)
  - PUT (editar)
  - DELETE (borrar)
- Devuelve
  - JSON
  - XML

HTTP STATUS

200 – OK
201 – CREATED
403 – FORBIDDEN
404 – NOT FOUND

Ejemplos:

GET /rest/api/pedido/{id}
DELETE /rest/api/pedido/{id}
POST /rest/api/pedido
PUT /rest/api/pedido/{id}/producto/{idProducto}

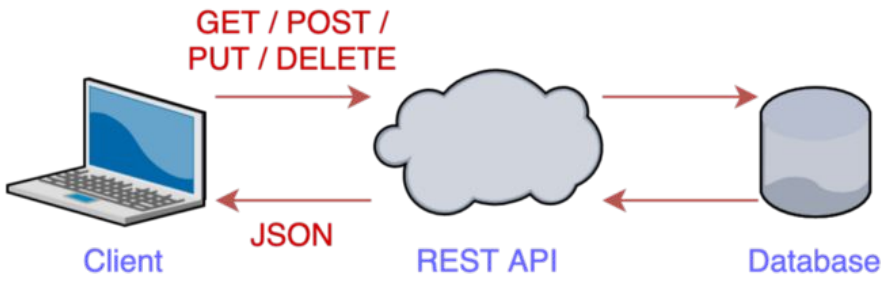
Devuelve
JSON/XML

```
{
    "nombre": "Pepito",
    "coches" : ["Ford", "Fiat", "Audi"]
}
```

REST + API = REST API

REST: es un conjunto de restricciones que se utilizan para que las solicitudes HTTP cumplan con las directrices definidas en la arquitectura: cliente servidor, sin estado, con cache, etc.

API: utiliza peticiones HTTP responsables de las operaciones básicas necesarias para la manipulación de datos (GET, POST, etc).



REST API es el conjunto de buenas prácticas utilizadas en las peticiones HTTP realizadas por una API en una aplicación web.

Es decir, cuando se habla de REST API, significa utilizar una API para acceder a aplicaciones back-end, de manera que esa comunicación se realice con los estándares definidos por el estilo de arquitectura Rest.

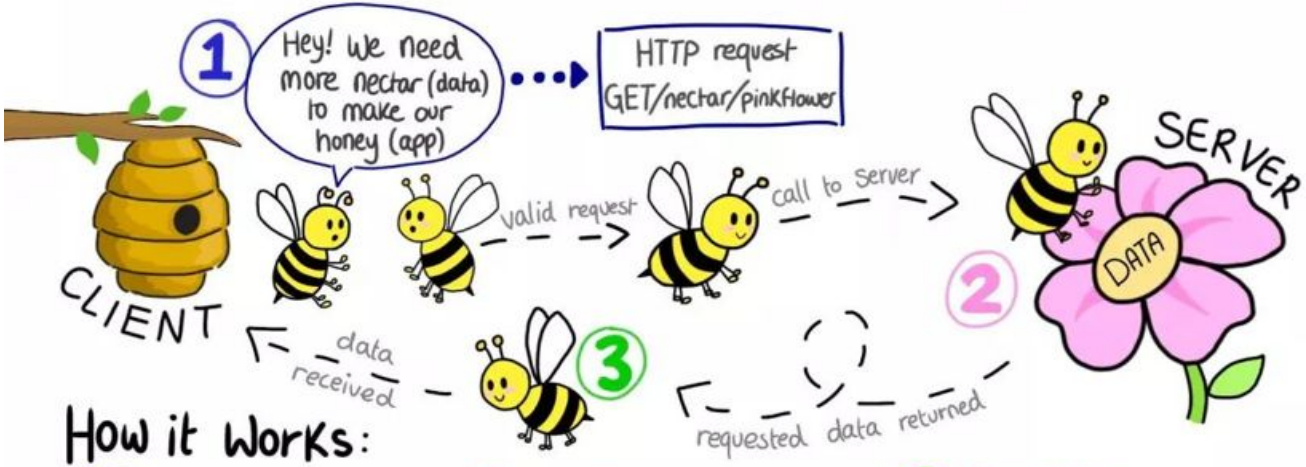
REST API

DEV.F

What is an API?

@Rapid_API

An application programming interface allows two programs to communicate. On the web, APIs sit between an application and a web server, and facilitate the transfer of data.

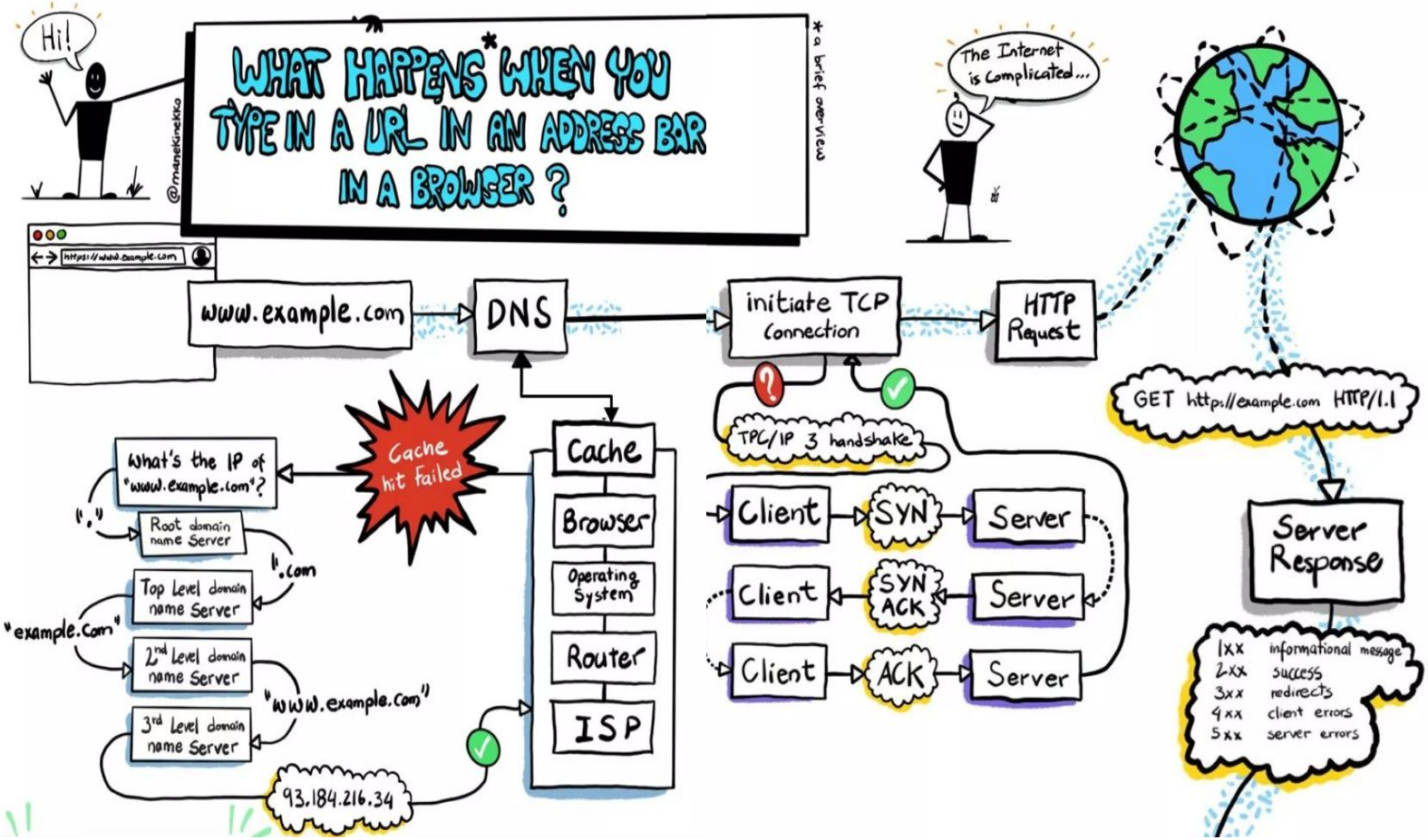


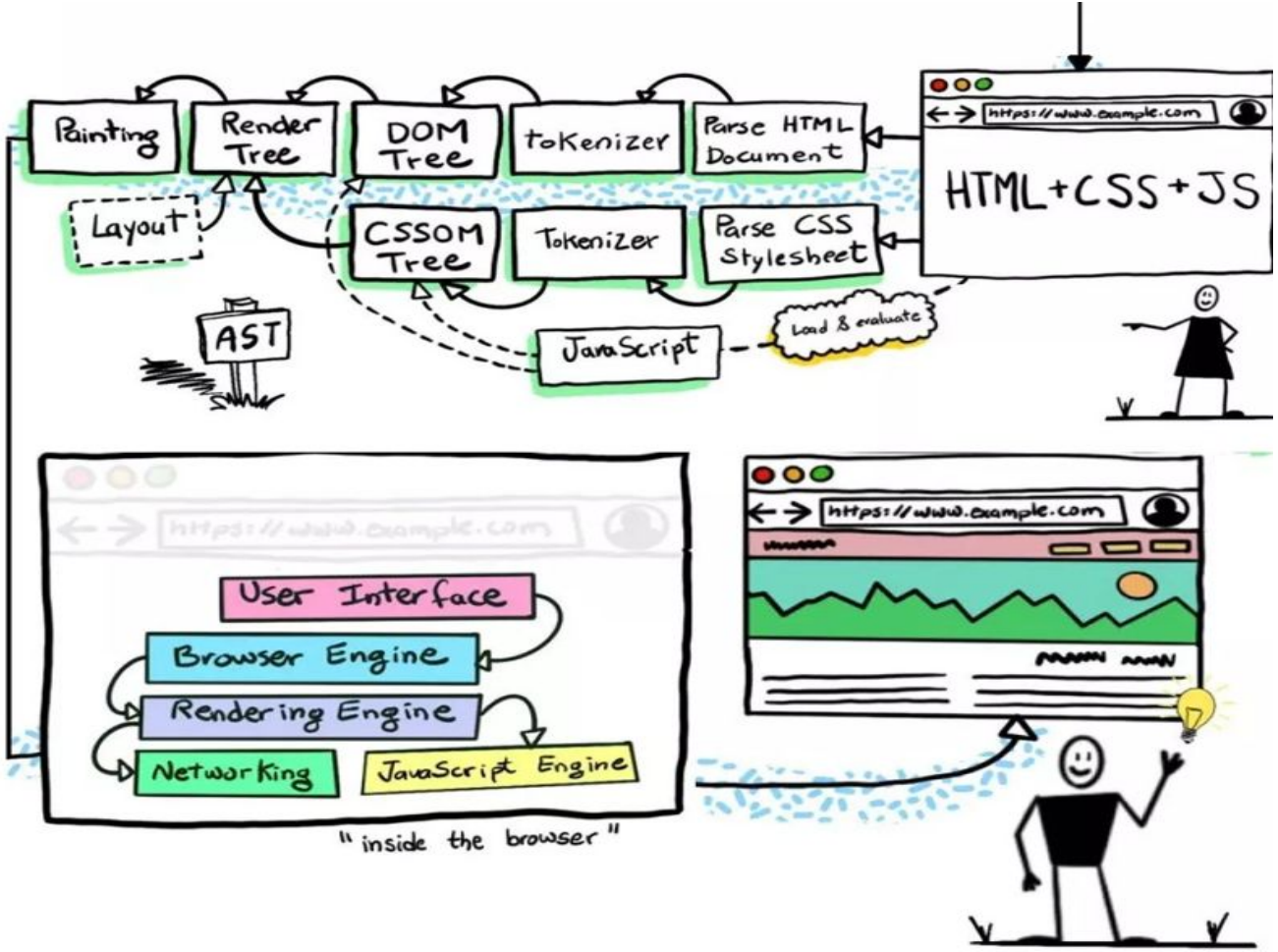
How it Works:

1 Request
API call is intiated by by the Client application via a HTTP request

2 Receive
Our worker bee acts as an API, going to a flower (server) to collect nectar (data)

3 Response
The API transfers the requested data back to the requesting application, usually in JSON format


Hi!
@manekinekko
WHAT HAPPENS* WHEN YOU TYPE IN A URL IN AN ADDRESS BAR IN A BROWSER ?
*a brief overview
The Internet is Complicated...
https://www.example.com
www.example.com
DNS
initiate TCP Connection
HTTP Request
GET http://example.com HTTP/1.1
Cache hit failed
Cache
Browser
Operating System
Router
ISP
What's the IP of "www.example.com"?
"."
Root domain name Server
".com"
Top Level domain name Server
"example.com"
2nd Level domain name Server
"www.example.com"
3rd Level domain name Server
93.184.216.34
TPC/IP 3 handshake
Client
SYN
Server
Client
SYN ACK
Server
Client
ACK
Server
Server Response
1xx informational message
2xx success
3xx redirects
4xx client errors
5xx server errors


Painting
Render Tree
DOM Tree
tokenizer
Parse HTML Document
https://www.example.com
HTML+CSS+JS
Layout
CSSOM Tree
Tokenizer
Parse CSS Stylesheet
AST
JavaScript
Load & evaluate
https://www.example.com
User Interface
Browser Engine
Rendering Engine
Networking
JavaScript Engine
"inside the browser"
https://www.example.com

¿Qué es {JSON}?

JSON (JavaScript Object Notation) es un formato ligero de intercambio de datos, que usa la sintaxis de objeto de JS.

JSON, not Jason!

myfile.json

```
{
  "name"          : "Lupita Code",
  "id"            : 123,
  "isProgrammer"  : true,
  "email"         : null,
  "languages"     : ["Python", "JS"],
  "address"       : {
    "street"      : "5 main st",
    "city"        : "Boston"
  }
}
```

JSON RULES:

El **JSON Object** comienza y termina con llaves {}

Pares: **key:value** separados por comas.

Los valores pueden ser: **string, number, boolean, null, object y array.**

Las claves deben ser **únicas** y deben estar entre **comillas dobles.**

USOS

En API REST o Graphql

Archivos de configuración

Interacciones AJAX o WebSocket

JSON🔥

JSON (JavaScript Object Notation) es un formato de texto ligero para intercambiar datos. Es fácil de leer y escribir para los humanos y sencillo de interpretar y generar para las máquinas. Se utiliza ampliamente en aplicaciones web para enviar y recibir datos entre el cliente y el servidor.

DEV.F

Json

```json
{
  "name": "Juan",
  "age": 30,
  "city": "Madrid",
  "hobbies": ["leer", "fútbol", "viajar"],
  "email": "juan@example.com",
  "isMarried": true,
  "children": [
    {
      "name": "Ana",
      "age": 5
    },
    {
      "name": "Luis",
      "age": 3
    }
  ],
  "address": {
    "street": "Calle Mayor",
    "number": 45,
    "postalCode": "28013"
  },
  "employment": {
    "company": "Tech Solutions",
    "position": "Software Developer",
    "yearsActive": 8
  }
}
```

Características

- **Estructura:** Usa pares clave-valor.
- **Sintaxis:** Similar a la de los objetos de JavaScript.
- **Datos:** Soporta tipos de datos como cadenas, números, objetos, arreglos, booleanos y null.

- Es como un idioma internacional para datos: los humanos pueden leerlo fácilmente y **las máquinas pueden procesarlo sin problemas.**

DEV.F

Nombrando Endpoints en API Rest

Endpoint	Descripción
`/users`	Lista todos los usuarios
`/users/123`	Lista un usuario en específico
`/users/123/orders`	Lista los pedidos de un usuario específico
`/users/123/orders/0001`	Lista una orden específica de un usuario específico

DEV.F

Endpoints y Verbos Http

Un mismo Endpoint, hara diferentes acciones dependiendo el verbo http usado para accederlo

Endpoint	Verbo HTTP	Acción
`/users`	GET	Lista todos los usuarios
`/users`	POST	Crea un nuevo usuario
`/users/:id`	GET	Obtiene la información de un usuario específico
`/users/:id`	PUT	Actualiza la información de un usuario específico
`/users/:id`	DELETE	Elimina un usuario específico
`/users/:id/orders`	GET	Lista los pedidos de un usuario específico
`/users/:id/orders`	POST	Crea un nuevo pedido para un usuario específico
`/users/:id/orders/:orderId`	GET	Obtiene la información de un pedido específico de un usuario
`/users/:id/orders/:orderId`	PUT	Actualiza la información de un pedido específico de un usuario
`/users/:id/orders/:orderId`	DELETE	Elimina un pedido específico de un usuario

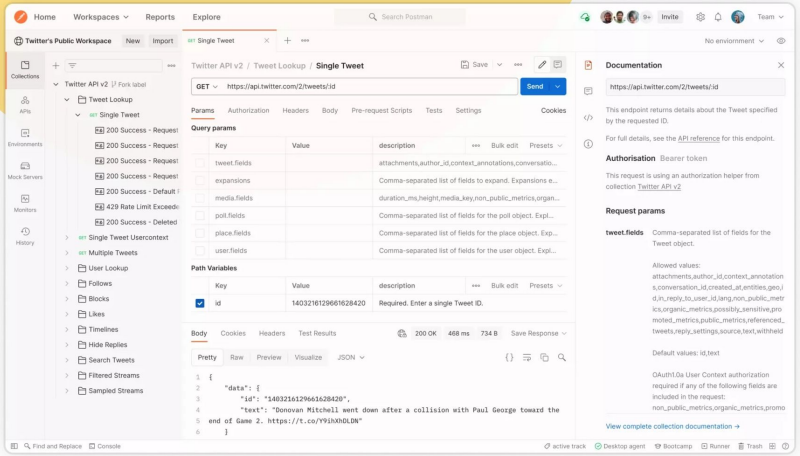
DEV.F

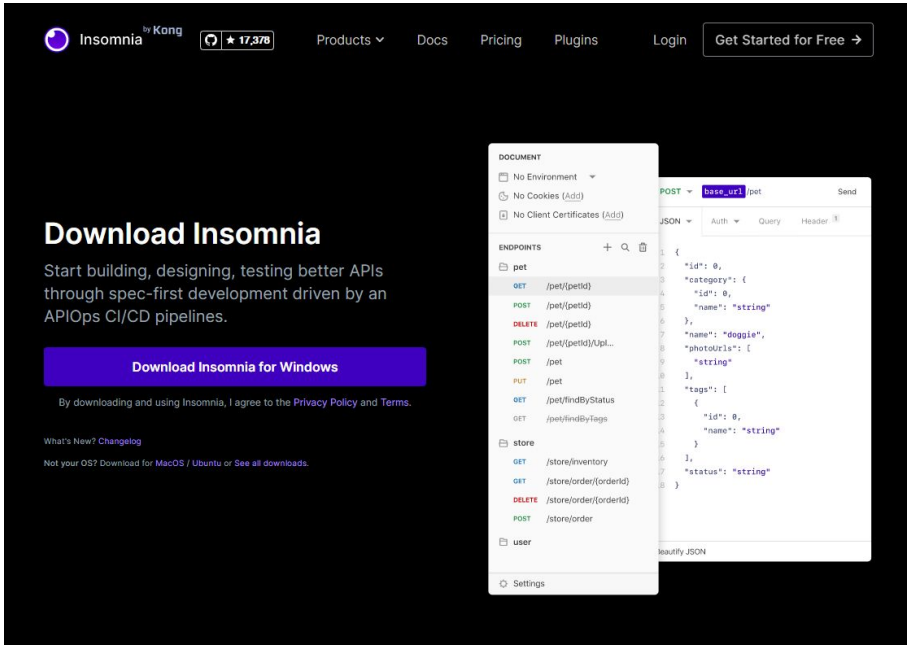
Herramientas para consumir una API

DEV.F::
DESARROLLAMOS(PERSONAS);

{DEVF}

Herramientas para probar APIs





POSTMAN

INSOMNIA

https://www.postman.com/

https://insomnia.rest/

Postman y pruebas de API

Postman es una herramienta para simular peticiones HTTP sin necesidad de escribir código. Permite ver cómo se comporta un endpoint: qué datos recibe, qué devuelve y cómo responde ante errores.

DEV.F:

Postman y pruebas de API

Es como usar un walkie-talkie para probar si otra persona (el servidor) te escucha. Dices algo (envías un mensaje) y esperas una respuesta. Si la respuesta no llega o llega mal, sabes que hay un problema en la comunicación.

Postman es una herramienta para simular peticiones HTTP sin necesidad de escribir código. Permite ver cómo se comporta un endpoint: qué datos recibe, qué devuelve y cómo responde ante errores.

APIs para probar

DEV.F::
DESARROLLAMOS(PERSONAS);

{DEVF}

APIs Públicas para Practicar

Nombre	Descripción	URL
JSONPlaceholder	Simula una API REST con usuarios, posts, comentarios, etc.	https://jsonplaceholder.typicode.com
Reqres	Simula usuarios y login. Ideal para probar POST, PUT, DELETE.	https://reqres.in
PokeAPI	Info detallada sobre Pokémon, habilidades y tipos.	https://pokeapi.co
The Dog API	Devuelve imágenes de perros aleatorios.	https://thedogapi.com
The Cat API	Igual que la anterior pero con gatitos.	https://thecatapi.com
OpenWeatherMap	Clima actual y pronósticos. Requiere API key gratuita.	https://openweathermap.org/api
Bored API	Sugerencias de cosas para hacer cuando estás aburrido.	https://www.boredapi.com/api/activity

DEV.F

APIs Públicas para Practicar

Nombre	Descripción	URL
Advice Slip API	Devuelve consejos aleatorios.	https://api.adviceslip.com
Universities API	Lista de universidades por país. Ideal para practicar filtros.	http://universities.hipolabs.com
Fake Store API	Datos falsos de productos tipo e-commerce (nombre, precio, imagen).	https://fakestoreapi.com
OpenLibrary API	Info sobre libros, autores y ediciones.	https://openlibrary.org/developers/api
IP Geolocation	Información geográfica basada en IP.	https://ipapi.co/json/
CoinGecko API	Precios actualizados de criptomonedas. No requiere autenticación.	https://www.coingecko.com/en/api
JokeAPI	Chistes aleatorios, programáticos u oscuros.	https://v2.jokeapi.dev

DEV.F

🧪 Proyecto del Día: *"Rick & Morty API"*

https://rickandmortyapi.com/

🔥

DEV.F

Proyecto del día

Crear un pequeño programa en JavaScript que simule una petición a la API - Rick and Morty combinando operaciones asíncronas (async/await, console.log, node-fetch opcional), y visualizar por consola el resultado.

📌 Objetivos del proyecto

- Aprender a consumir una API
- Realizar un código de Javascript asíncrono
- Reforzar los temas teóricos vistos en clase.

DEV.F::

Recursos recomendados para estudiar

Categoría	Recurso / Sitio
Documentación Oficial	MDN Web Docs – https://developer.mozilla.org/es/docs/Web/HTTP
Curso gratuito	freeCodeCamp – https://www.freecodecamp.org/
API Playground	Postman – https://learning.postman.com/
Práctica de códigos HTTP	httpstat.us – https://httpstat.us/
Videos explicativos	Traversy Media, HolaMundo (YouTube)
Libro recomendado	*HTTP: The Definitive Guide* – O'Reilly

https://www.freecodecamp.org/news/learn-rest-api-principles-by-building-an-express-app/?ref=dailydev