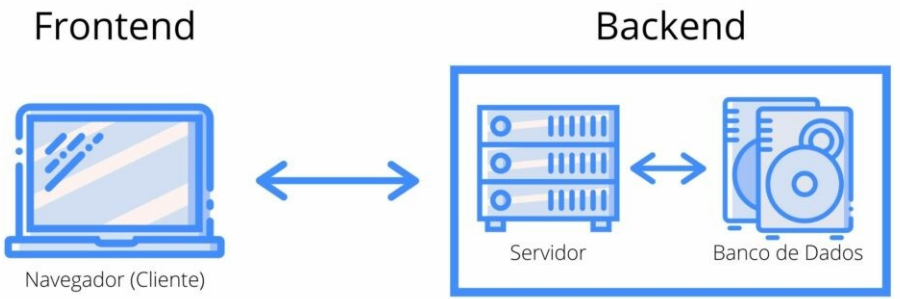
Peticiones HTTP

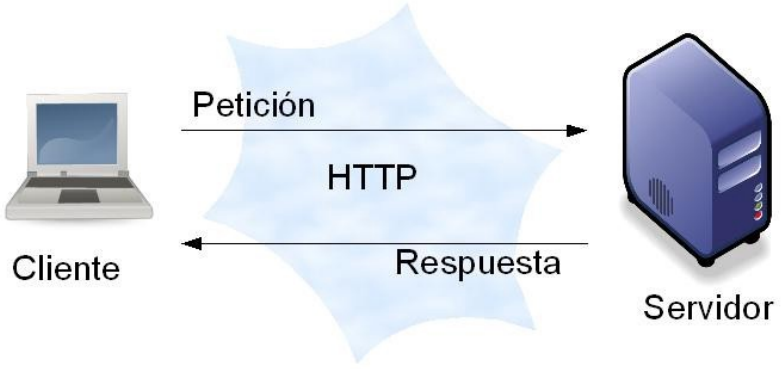
Introducción

- En el desarrollo de sistemas, se suele utilizar la arquitectura cliente – servidor
- El cliente está conformado por la aplicación a la que el cliente tiene acceso (app web, app móvil, app de escritorio)
- El servidor es un programa que corre en un servidor, el cual maneja peticiones del cliente, maneja la lógica del sistema, se conecta a una base de datos, etc



Arquitectura cliente - servidor

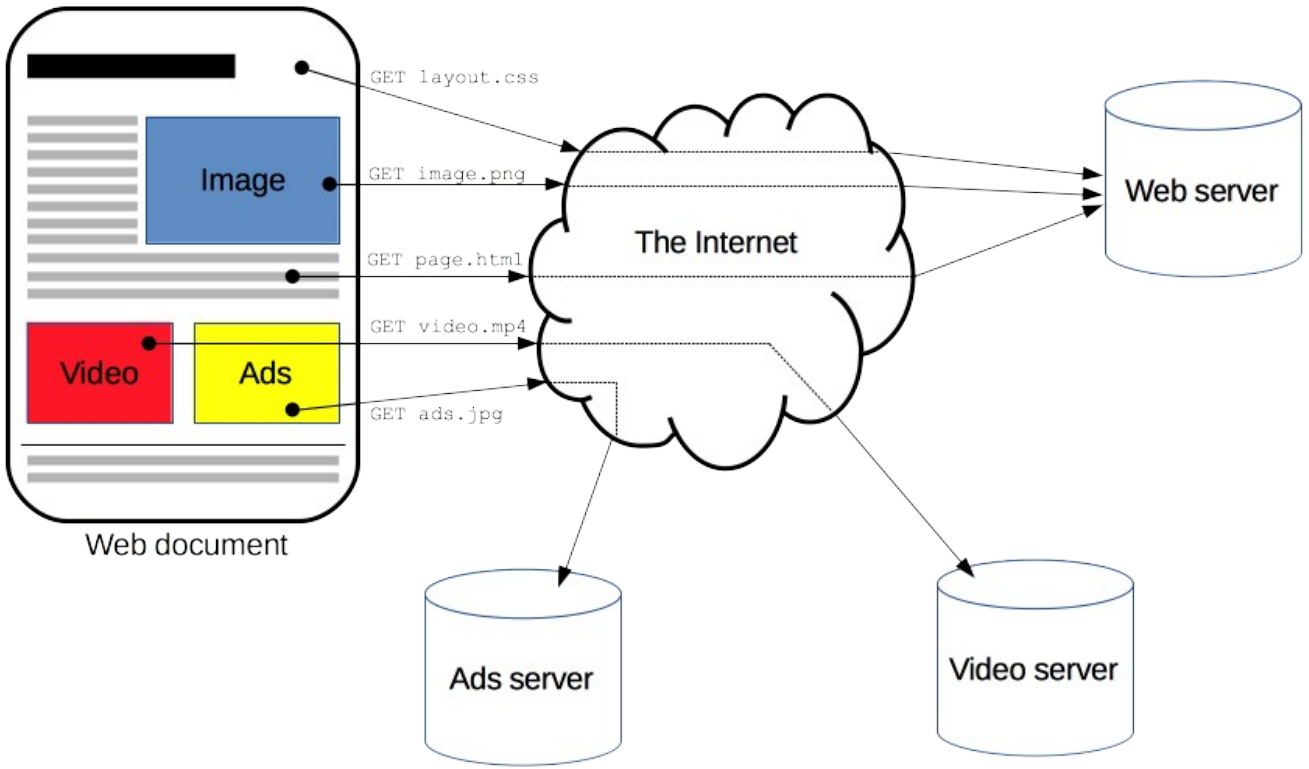
- Por ejemplo, al hacer un login en un sistema, el cliente le manda los datos al servidor (usuario, contraseña)
- El servidor se encarga de validar estos datos (consultando en una base de datos por ejemplo)
- El servidor le envía una respuesta al cliente
- Esta respuesta puede ser un estado, algún dato, o un JSON



Peticiones HTTP

- EL intercambio de datos entre cliente y servidor, se realiza a través de peticiones, las cuales siguen un protocolo llamado HTTP.
- HTTP (Hypertext Transfer Protocol) nos permite realizar una petición de datos y recursos
- Estos recursos pueden ser documentos HTML, css, js, o información en algún formato como JSON
- Es la base de cualquier intercambio de datos en la Web

HTTP


GET layout.css
Image
GET image.png
GET page.html
GET video.mp4
Video
Ads
GET ads.jpg
Web document
The Internet
Web server
Ads server
Video server

AJAX

- Tradicionalmente, se realizan peticones HTTP con ajax
- Ajax nos permite lo siguiente:
  - Leer datos de un servidor, después de que se haya cargado la página
  - Actualizar una pagina web sin recargar la página
  - Enviar datos a un servidor en segundo plano

- Antes de AJAX, a la hora de enviar información a un servidor, la pagina debía recargarse

XMLHttpRequest

- Antiguamente se utllizaba XMLHttpRequest() para las peticiónes

```
1.  var xmlHttp = new XMLHttpRequest();
2.  xmlHttp.responseType = 'json';
3.  var url = "https://api.geoapify.com/v1/isoline?
    lat=47.68388118858141&lon=8.6142781886542 32&type=time
    OUR_API_KEY";
4.  xmlHttp.onreadystatechange = () => {
5.      if (xmlHttp.readyState === 4) {
6.          if (xmlHttp.status === 200) {
7.              // check xmlHttp.responseText here;
8.              console.log(xmlHttp.response);
9.          } else {
10.             console.log(xmlHttp.response);
11.         }
12.     }
13. };
14. xmlHttp.open("GET", url, true); // true for asynchron
```

Fetch

- Hoy en día, se utiliza la interfaz fetch() una función mas sencilla para realizar peticiones.
- Fetch está disponible de forma nativa en JavaScript (no hay que instalar o importar nada)
- Fetch es asíncrono, y utiliza **promesas**
- Ejemplo básico:

Ejemplo básico de fetch
Obteniendo un JSON del servidor

```
// Realizamos la petición y guardamos la promesa
const request = fetch("http://localhost:5000/get_persons");

// Al ser una Promise, llamamos al metodo then()
request
.then(response => response.json())
// este primer then vuelve a retornar otra promesa del json
.then(jsonData => console.log(jsonData))
// aquí finalmente obtenemos el json que nos dió el servidor
```

Métodos de las peticiones

- Tenemos varios métodos distintos de las peticiones http
- Estos son algunos de ellos:
    - GET → Nos permite obtener un recurso o dato
    - POST → nos permite enviar datos
    - PATCH → Nos permite aplicar modificaciones (update)
    - DELETE → Nos permite eliminar un recurso o dato

- Los mas utilizados son GET y POST

Uso de fetch()

- El método fetch recibe dos parámetros
  - URL de la petición al servidor (en forma de string)
  - Un objeto (opcional) para configuraciones de la petición
- En el objeto de configuración podemos indicar el método de la petición, y otros parámetros de configuración, como los headers o el body

- En caso que hagamos una petición sin indicar el objeto con configuración, la misma será tomada como del tipo GET

Ejemplo básico de fetch()

```
const request2 = fetch("http://localhost:5000/insert_person", {
    method: "POST",     // método
    mode: "cors",    // política de cors
    // cabecera de la petición
    headers: {
        "Content-Type": "application/json"
    },
    // en el body, los datos a enviar
    body: JSON.stringify({ name: "Juan Rodríguez", age: 40 })
})

request2
.then(response => response.json())
.then(data => console.log(data))
.catch(err => console.error(err))
```

Hagamos mas ejemplos...

- Hora de picar código!

API REST pública para practicar

- Hay varias APIs publicas y gratuitas para practicar peticiónes
- Una de ellas es JSON-Placeholder:
- https://jsonplaceholder.typicode.com/

Pero... ¿Qué es una API REST?

- ¿Qué es una API?:
  - Las apis son interfaces que conecta aplicaciones para que compartan información
  - ¿Que es una interfaz?
    - Una interfaz es una capa que conecta dos sistemas

- ¿Que es una API REST?
  - Rest es una arquitectura para apis que sigue el protocolo http