Peticiones HTTP: GET

Curso 2025/26

Paco Segura

Protocolo HTTP

- **Protocolo** -> *"Conjunto de reglas que se establecen para la comunicación entre dos sistemas".*
- Secuencia de acciones-reacciones.
- "Pedimos algo de una manera determinada" -> "Se nos proporciona algo de una manera determinada".
- Petición -> formato convenido entre las 2 partes.

Protocolo HTTP

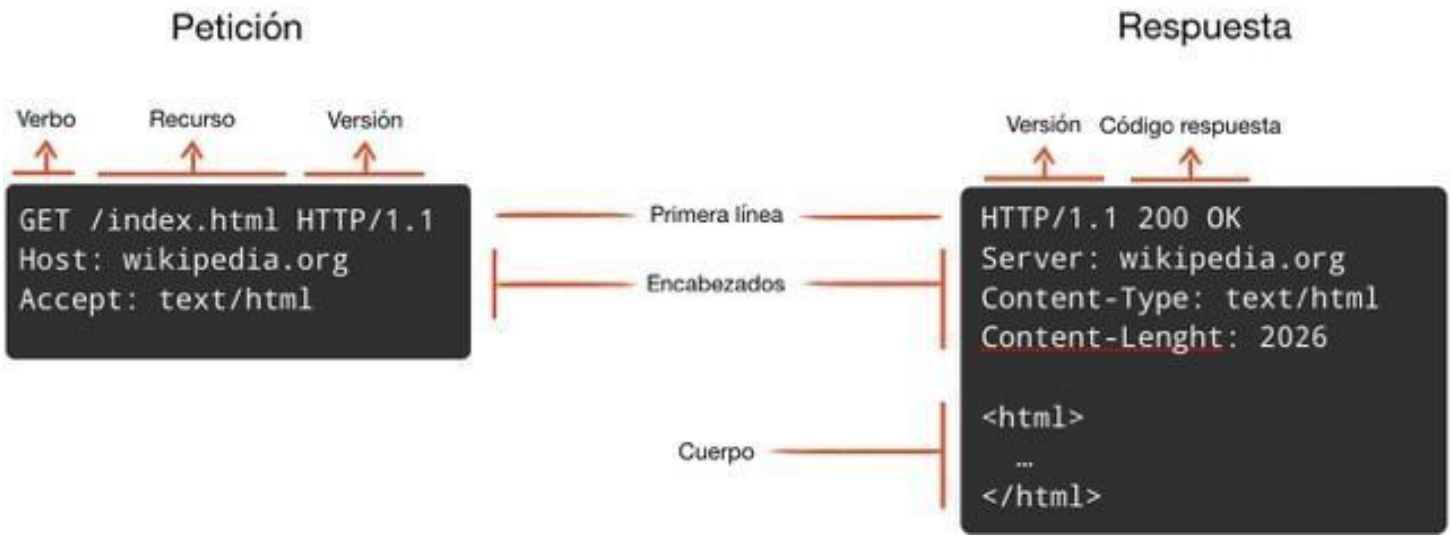
- HTTP ->“Hyper Text Transfer Protocol”
- Tres partes

    -El verbo (tipo de petición).

    -El recurso (URL o “*path*” al que queremos pedir datos o enviarlos).

    -La versión.

Protocolo HTTP



Tipos de peticiones HTTP

- GET
  - Para pedir recursos de un servidor (≈ "*select*")
- POST
  - Para enviar información de creación a un servidor (≈ "*insert*")
- PUT
  - Para enviar información de modificación a un servidor (≈ "*update*")
- DELETE
  - Para enviar solicitudes de borrado a un servidor (≈ "*delete*")

Peticiones HTTP con JavaScript

- Se pueden realizar peticiones HTTP en JavaScript de varias formas:
    - Utilizando AJAX.
    - Utilizando .fetch() –desde ES6-.
    - Utilizando librerías (Axios).

Introducción a JavaScript asíncrono

- JavaScript es "*Single Threaded*" -> cada ejecución de código JS se realiza en un único hilo de procesador (1 núcleo / core de la CPU).

- Problema -> una llamada a una función que puede tardar en procesarse, genera una espera sin que se puedan ejecutar instrucciones adicionales.

- **Solución: Programación asíncrona.** Permite 'pasar' a otras tareas mientras la operación se completa**. Las peticiones HTTP son asíncronas.**

AJAX

- AJAX -> *Asyncronous JavaScript and XML*. XML es el formato con el que se trabajaba en peticiones AJAX. Actualmente hay más formatos –JSON-.
- Se utiliza para hacer uso de las capacidades asíncronas de JavaScript.
- Pueden hacerse peticiones GET y POST con el objeto XMLHttpRequest.
- Debemos construir el objeto utilizando 'new' y crear una función que manejará la respuesta del objeto y abrirá y enviará la petición.

AJAX: sintaxis petición GET

```
//crea objeto XMLHttpRequest
const xhr = new XMLHttpRequest();


const url = 'http:direccion_api_a_llamar'


//Maneja la respuesta
xhr.responseType = 'text';
xhr.onreadystatechange = () => {
   if(xhr.readyState === XMLHttpRequest.DONE){
     //Código a implementar con la respuesta
  }
}


//Abre la peticion y envia objeto
xhr.open('GET',  url);
xhr.send();
```

JavaScript asíncrono: *Promises*

- Objeto Promise: representa el resultado de una operación asíncrona.
  - Función que garantiza que antes o después se dará una respuesta a una petición.

- Tipos de respuesta:
  - Éxito: “resuelto”.
  - Fallo: “rechazado”.

JavaScript asíncrono: *Promises*

```
const promise = new Promise((resolve, reject) => {
    resolve('resuelta');
    reject('rechazada');
});

promise.then(result => {
    console.log(result);
});
```

.fetch()

- Funcionalidad disponible desde ES6.

- fetch() devuelve una promesa –es decir, un objeto que representa si la petición ha sido ‘resuelta’ o ‘rechazada’-.

- Podemos encadenar métodos .then() para manejar promesas devueltas por fetch(). El método json() convierte la promesa devuelta en un objeto JSON.

Funciones async / await

- Funcionalidad disponible desde ES8.

- Async -> palabra reservada utilizada para crear funciones que devuelven promesas.

- Await -> palabra reservada utilizada para que el programa continúe ejecutándose mientras la promesa se resuelve. Solo puede utilizarse con funciones declaradas con async.

- Las funciones async / await devuelven una promesa que se resolverá con el valor devuelto por la función o será rechazado por una excepción lanzada desde el interior de la misma función async.

fetch con async / await: sintaxis petición GET

```
const getData = async () => {
    try {
        const response = await fetch('http:direccion_api_a_llamar');
        if (response.ok) {
            const data = await response.json();
            return data;
        }
    }
    catch (error) {
        console.log(error);
    }
}
```

Async y **await** son palabras reservadas, indican que es una función asíncrona que devuelve una promesa. No se puede usar **await** sin antes usar **async**.

try{...} catch(error){...}
Bloque para almacenar la respuesta a la petición y capturar si existe un error

fetch(dirección_petición_http) -> sintaxis petición get() utilizando fetch.

Axios

- Cliente HTTP de alto rendimiento.
- Librería de poco peso.
- Muy empleado.
- Instalación:
  - npm install axios

Axios: Sintaxis petición HTTP

```
const getDatos = async () => {
    try {
        const response = await axios.get('https://direccion_api');
        console.log(response.data);
    } catch (error) {
        console.error(error);
    }
};
```

Async y **await** son palabras reservadas, indican que es una función asíncrona que devuelve una promesa. No se puede usar **await** sin antes usar **async**.

try{...} catch(error){...}
Bloque para almacenar la respuesta a la petición y capturar si existe un error

axios.get(dirección_petición_http) -> sintaxis petición get() utilizando la librería (axios).

Ejemplos peticiones GET HTTP

- Vamos a ver un ejemplo práctico de cómo utilizar fetch y axios.

- Realizaremos una petición GET a la REST API pública de The Cat API:
    - The Cat API - Cats as a Service.
    - API pública que devuelve imágenes aleatorias de gatos.
- Las peticiones GET a APIs públicas devuelven respuestas en forma de arrays, objetos o arrays de objetos. Inicialmente las respuestas pueden resultar confusas. Solución: POSTMAN
- POSTMAN: software que permite realizar peticiones HTTP y visualizar de forma clara y ordenada las respuestas.

Ejemplo AJAX

```
const getData = () => {
    const xhr = new XMLHttpRequest();
    const url = 'https://api.thecatapi.com/v1/images/search?size=full';
    xhr.responseType = 'text';
    xhr.onreadystatechange = () => {
        if (xhr.readyState === XMLHttpRequest.DONE) {
            try {
                let data = JSON.parse(xhr.responseText);
                console.log(data);
            } catch (err) {
                console.log(err.message + " in " + xhr.responseText);
                return;
            }
        }
    }
    xhr.open('GET', url);
    xhr.send();
}
```

Ejemplo fetch

```
const getData = async () => {
    try {
        const response = await fetch('https://api.thecatapi.com/v1/images/search?size=full');
        if (response.ok) {
            const jsonResponse = await response.json();
            console.log(jsonResponse);
        }
    }
    catch (error) {
        console.log(error);
    }
}
```

Utilizamos siempre **.json()** para convertir a JSON el contenido de la respuesta. Para esta API viene en forma de Array con un Objeto.

Ejemplo AXIOS

```
const getData = async () => {
    try {
        let response = await axios.get('https://api.thecatapi.com/v1/images/search?size=full');
        console.log(response.data);
    } catch (error) {
        console.error(error);
    }
}
```

Con **axios** debemos utilizar siempre **.data** para acceder al contenido de la respuesta. Para esta API viene en forma de Array con un Objeto.

});