

INTRODUCCIÓN



JavaScript



CÓMO FUNCIONA LA WEB



PETICIÓN A TRAVÉS DEL PROTOCOLO HTTP

`http://domain.com`



RESPUESTA (HTML, CSS, JS...)



Con AJAX interceptamos esta respuesta para evitar que la página se recargue. De esta forma en lugar de solicitar todos los datos (html,css,etc.), hacemos la petición por HTTP pero sólo solicitamos ciertos datos y los guardamos en un objeto.



PETICIÓN A TRAVÉS DEL PROTOCOLO HTTP

`http://domain.com`



{data}



RESPUESTA (data)



AJAX es asíncrono. Normalmente cuando hacemos una petición y esperamos una respuesta, el navegador se queda cargando y hasta que no recibimos una respuesta no se carga la página. Con AJAX conseguimos que esto no ocurra. Conseguimos que se cargue la página y que los datos solicitados lleguen mas tarde. Es decir, la página va por un lado y la petición y los datos van por otro.

Normalmente las peticiones a los servidores se suelen hacer con **PHP** o con **Node.js**



Para no tener que instalarnos php ni una base de datos para explicar la parte de AJAX vamos a utilizar una API que se llama JSONplaceholder. Podemos hacer peticiones normales o peticiones AJAX. <https://jsonplaceholder.typicode.com/>

Ejemplo con AJAX

```
const button = document.getElementById('button')

button.addEventListener('click', () => {
  let xhr
  if (window.XMLHttpRequest) xhr = new XMLHttpRequest()
  else xhr = new ActiveXObject("Microsoft.XMLHTTP")

  xhr.open('GET', 'https://jsonplaceholder.typicode.com/users')

  xhr.addEventListener('load', (data) => {
    console.log(data.target)
  })

  xhr.send()
})
```

Si solo se utiliza JQuery para peticiones AJAX podemos evitar usar JQuery con estas 2 líneas. XMLHttpRequest está presente a partir de Internet Explorer 11 debido a esto hay que controlar XMLHttpRequest está presente o no

Abrimos una petición al servidor

Hay que añadir un evento para que nos avise cuando se han cargado los datos debido a que estamos trabajando de forma asíncrona.

Enviamos la petición al servidor

Ejemplo con AJAX mostrando la información en un JSON

```
const button = document.getElementById('button')

button.addEventListener('click', () => {
  let xhr
  if (window.XMLHttpRequest) xhr = new XMLHttpRequest()
  else xhr = new ActiveXObject("Microsoft.XMLHTTP")

  xhr.open('GET', 'https://jsonplaceholder.typicode.com/users')

  xhr.addEventListener('load', (data) => {
    console.log(JSON.parse(data.target.response))
  })

  xhr.send()
})
```

Convertimos el String a JSON

EJERCICIO AJAX y JSON

- Modifica el ejemplo anterior de forma que muestre en mi página web en una lista, los datos del JSON recibidos. Deberá aparecer lo siguiente

AJAX

Get Data

- 1 - Leanne Graham
- 2 - Ervin Howell
- 3 - Clementine Bauch
- 4 - Patricia Lebsack
- 5 - Chelsey Dietrich
- 6 - Mrs. Dennis Schulist
- 7 - Kurtis Weissnat
- 8 - Nicholas Runolfsson
- 9 - Glenna Reichert
- 10 - Clementina DuBuque

FETCH API – MISMO EJEMPLO ANTERIOR

```
const button = document.getElementById('button')

//res = response = respuesta
button.addEventListener('click', () => {
  fetch('https://jsonplaceholder.typicode.com/users')
    .then(res => res.ok ? Promise.resolve(res) : Promise.reject(res))
    .then(res => res.json())
    .then(res => console.log(res))
})
```


FETCH API

- Es el reemplazo moderno del XMLHttpRequest
- Proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas.
- También provee un método global `fetch()` que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red. Está basado en promesas, por lo cual tiene un `response` y un `reject` internos. `Response` tiene varios métodos
 - **`arrayBuffer()`**: Archivos binarios en bruto (mp3, pdf, jpg, etc). Se utiliza cuando se necesita manipular el contenido del archivo.
 - **`blob()`**: Archivos binarios en bruto (mp3, pdf, jpg, etc). Se utiliza cuando no se necesita manipular el contenido y se va a trabajar con el archivo directamente
 - **`clone()`**: crea un clon de un objeto de respuesta, idéntico en todos los sentidos, pero almacenado en una variable diferente.
 - **`formData()`**: Se utiliza para leer los objetos `formData`
 - **`json()`**: Convierte los archivos json en un objeto de JavaScript
 - **`text()`**: Se utiliza cuando queremos leer un archivo de texto. Siempre se codifica en UTF-8
- En Internet Explorer no funciona

EJERCICIO AJAX y JSON

- Modifica el ejemplo anterior de forma que muestre en mi página web en una lista, los datos del JSON recibidos utilizando **FETCH**. Deberá aparecer lo siguiente

AJAX

Get Data

- 1 - Leanne Graham
- 2 - Ervin Howell
- 3 - Clementine Bauch
- 4 - Patricia Lebsack
- 5 - Chelsey Dietrich
- 6 - Mrs. Dennis Schulist
- 7 - Kurtis Weissnat
- 8 - Nicholas Runolfsson
- 9 - Glenna Reichert
- 10 - Clementina DuBuque

FETCH API

- Es el reemplazo moderno del XMLHttpRequest
- Proporciona una interfaz JavaScript para acceder y manipular partes del canal HTTP, como peticiones y respuestas.
- También provee un método global `fetch()` que proporciona una forma fácil y lógica de obtener recursos de forma asíncrona por la red. Está basado en promesas, por lo cual tiene un `response` y un `reject` internos. `Response` tiene varios métodos
 - **`arrayBuffer()`**: Archivos binarios en bruto (mp3, pdf, jpg, etc). Se utiliza cuando se necesita manipular el contenido del archivo.
 - **`blob()`**: Archivos binarios en bruto (mp3, pdf, jpg, etc). Se utiliza cuando no se necesita manipular el contenido y se va a trabajar con el archivo directamente
 - **`clone()`**: crea un clon de un objeto de respuesta, idéntico en todos los sentidos, pero almacenado en una variable diferente.
 - **`formData()`**: Se utiliza para leer los objetos `formData`
 - **`json()`**: Convierte los archivos json en un objeto de JavaScript
 - **`text()`**: Se utiliza cuando queremos leer un archivo de texto. Siempre se codifica en UTF-8
- En Internet Explorer no funciona

FETCH API -POST

- Para hacer peticiones POST, fetch admite un segundo parámetro.

```
fetch(url, {  
  method: 'POST',  
  body: Los datos que enviamos. Si es un objeto hay que convertirlo con  
  JSON.stringify(datos),  
  headers: {          cabeceras de información sobre lo que estamos enviando  
    https://developer.mozilla.org/es/docs/Web/HTTP/Headers    }  
})
```


FETCH API –POST. EJEMPLO

```
const button = document.getElementById('button')

button.addEventListener('click', () => {
  const newPost = {
    title: 'A new post',
    body: ' Lorem ipsum dolor sit amet consectetur adipisicing
elit.',
    userId: 1
  }

  fetch('https://jsonplaceholder.typicode.com/posts', {
    method: 'POST',
    body: JSON.stringify(newPost),
    headers: {
      "Content-type": "application/json"
    }
  })

  .then(res => res.json())
  .then(data => console.log(data))
})
```

EJERCICIO AJAX, JSON Y BASE DE DATOS

1. Crea la base de datos que se proporciona en el archivo marvel.sql
2. Interpreta el código marvel.php y colócalo en el servidor y utilízalo cuando creas conveniente.
3. Crea una página html que muestre:
 - Un tabla cuyos títulos sean: Name, Gender, Fighting Skills
 - Un campo de tipo select donde deberán mostrarse los superheroes que hay cargados en la base de datos.
4. Crea un botón de tipo submit que se llame "Get Data"
5. Al pulsar el botón se deberá modificar el comportamiento para que en lugar de realizar el submit por defecto se llame una nueva función llamada "getData".
6. Crea una función que se llame getData y que reciba un id. Dentro de esta función , se deberá realizar una petición AJAX que recupere los datos de la base de datos en un JSON y que rellene el select creado de superheroes si este no está relleno. En caso de estar relleno, al seleccionar un superheroe del select y pulsar el botón "Get Data" deberá crear una fila en la tabla con las características del superheroe seleccionado.

EJERCICIO AJAX, LECTURA DE ARCHIVOS

1. Crea una página con 2 botones. Uno para mostrar una imagen y otro para mostrar un pdf
2. Utilizando fetch. Al pulsar el botón de mostrar imagen deberá mostrar la imagen. Para ello puedes usar la función `blob()` y `URL.createObjectURL`. Busca cómo funcionan
3. Utilizando fetch. Al pulsar el botón mostrar pdf deberá mostrar el pdf a través de un link.

API –WEB STORAGE

Nos permite guardar información en el dispositivo con el conjunto de clave – valor. Muy parecido a una cookie o a un objeto Javascript pero el tamaño es mucho mas grande.

Tiene dos mecanismos en el almacenamiento web que son los siguientes:

- **sessionStorage** mantiene un área de almacenamiento separada para cada origen que está disponible mientras dure la sesión de la página (mientras el navegador esté abierto, incluyendo recargas de página).
- **localStorage** hace lo mismo, pero persiste incluso cuando el navegador se cierre y se reabra.

Ambos funcionan con **clave:valor** y tienen dos métodos fundamentales:

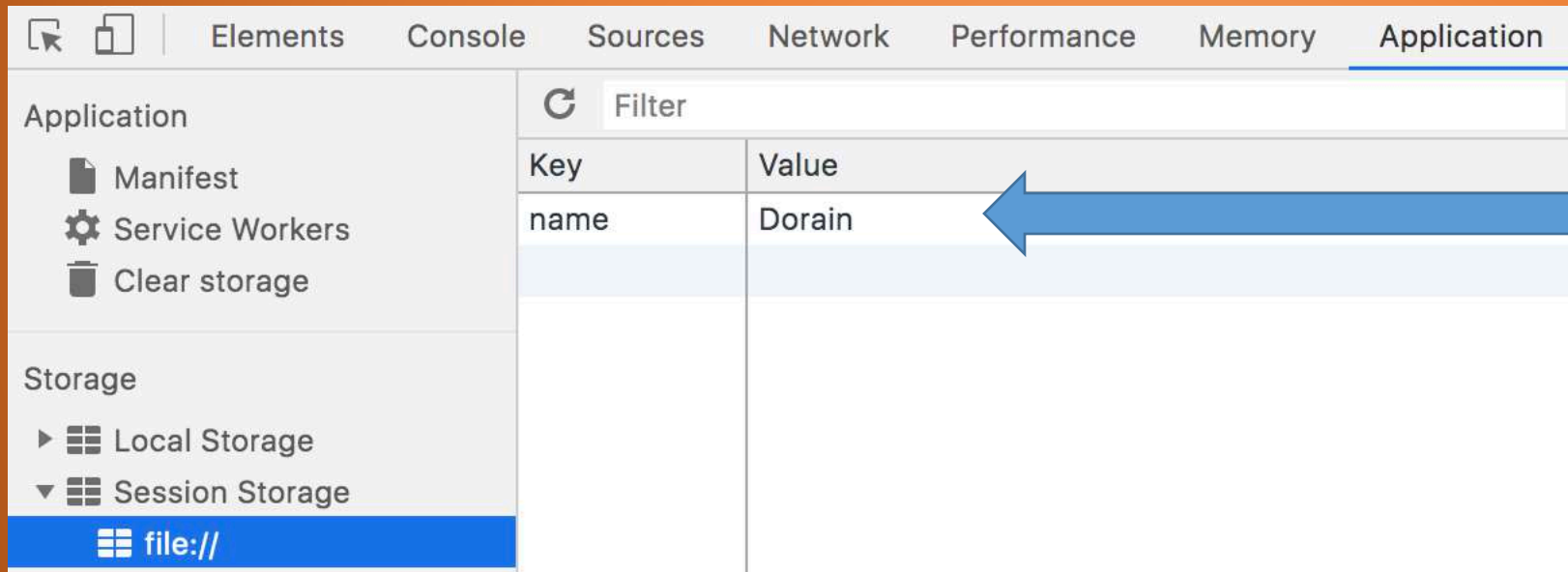
- **setItem()** para asignar una clave:valor
- **getItem()** que recibe como parámetro la clave de la que queremos obtener el valor

API –WEB STORAGE. EJEMPLO

```
const form = document.getElementById('form')
const keys = document.getElementById('keys')

form.addEventListener('submit', (e) => {
  e.preventDefault()

  sessionStorage.setItem('name', 'Dorian')
})
```



The screenshot shows the Chrome DevTools 'Application' tab. On the left sidebar, under 'Storage', 'Session Storage' is expanded, and 'file://' is selected. The main panel displays a table of stored items. A blue arrow points from the text on the right to the 'Dorain' value in the table.

Key	Value
name	Dorain

Si pulsamos F12 en nuestro navegador podemos ver los valores del Web Storage

API –WEB STORAGE. EJEMPLO CON OBJETO JAVASCRIPT

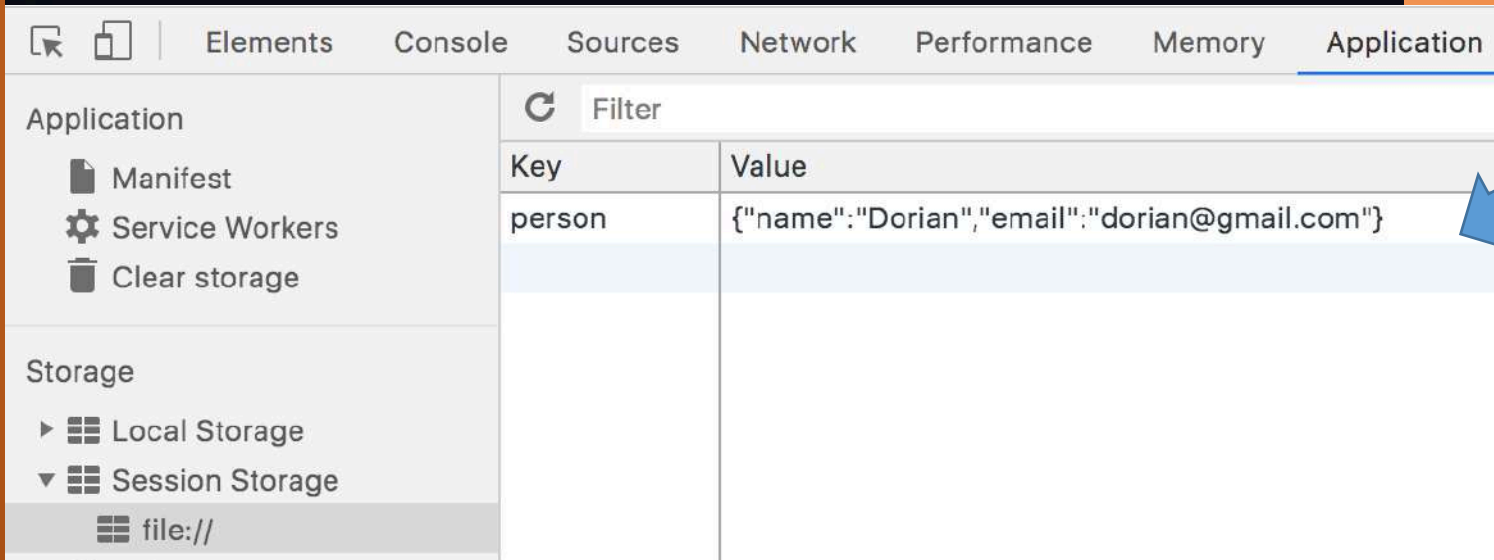
```
const form = document.getElementById('form')
const keys = document.getElementById('keys')

form.addEventListener('submit', (e) => {
  e.preventDefault()

  const person = {
    name: 'Dorian',
    email: 'dorian@gmail.com'
  }

  sessionStorage.setItem('person', JSON.stringify(person))
})
```

Como tiene que recibir una cadena, convertimos el objeto a cadena



The screenshot shows the Chrome DevTools Application tab. On the left, the 'Storage' section is expanded, showing 'Session Storage' for the file:// protocol. The main pane displays a table of stored items. A blue arrow points from the text 'Como tiene que recibir una cadena, convertimos el objeto a cadena' to the JSON string value in the table.

Key	Value
person	{"name":"Dorian","email":"dorian@gmail.com"}

API –WEB STORAGE. EJEMPLO CON DATOS DE FORMULARIO

Key

Value

Select key

```
sessionStorage.setItem(form.key.value, form.value.value)
```

Dado el siguiente formulario, suponiendo que lo tenemos cargado en la variable `form`. Podemos guardar el valor del input `key` y el valor del input `value` en el `sessionStorage` de esta manera

Para recuperar valores del `sessionStorage` podemos utilizar la siguiente instrucción:
`sessionStorage.getItem(nombre de la clave a recuperar)`

API –WEB STORAGE. EJEMPLO CON DATOS DE FORMULARIO

- Para recuperar valores del `sessionStorage` podemos utilizar la siguiente instrucción:
`sessionStorage.getItem(nombre de la clave a recuperar)`
- Para limpiar los datos utilizaríamos `sessionStorage.clear()`
- Para borrar un elemento utilizaríamos `sessionStorage.removeItem(clave de lo que queremos borrar)`

TODOS LOS MÉTODOS DE `sessionStorage` ME SIRVEN DE IGUAL MANERA PARA `local Storage`