JavaScript - Ajax 2

Realizar peticiones Ajax

Hemos visto lo que es el protocolo HTTP. Ahora que tenemos instalado un servidor que nos proporciona una API (json-server), vamos a realizar peticiones HTTP en nuestro código javascript usando Ajax.

Para hacer una petición debemos crear una instancia del objeto **XMLHttpRequest** que es el que controlará todo el proceso. Los pasos a seguir son:

1. Creamos la instancia del objeto: `const peticion=new XMLHttpRequest()`
2. Para establecer la comunicación con el servidor ejecutamos el método **.open()** al que se le pasa como parámetro el tipo de petición (GET, POST, …) y la URL del servidor: `peticion.open('GET', 'https://jsonplaceholder.typicode.com/todos')`
3. OPCIONAL: Si queremos añadir cabeceras a la petición HTTP llamaremos al método **.setRequestHeader()**. Por ejemplo, si enviamos datos con POST hay que añadir la cabecera *Content-type* que indica al servidor en qué formato van los datos: `peticion.setRequestHeader('Content-type', 'application/x-www-form-urlencoded)`
4. Enviamos la petición al servidor con el método **.send()**. A este método se le pasa como parámetro los datos a enviar al servidor en el cuerpo de la petición (si es un POST, PUT o PATCH le pasaremos una cadena de texto con los datos a enviar:

```
peticion.send('dato1='+encodeURIComponent(dato1)+'&dato2='+encodeURIComponent(dato2))
```

   Si es una petición GET o DELETE no le pasaremos datos:

```
peticion.send()
```

1. Ponemos un escuchador (*listener*) al objeto *peticion* para saber cuándo está disponible la respuesta del servidor.

Eventos de XMLHttpRequest

Tenemos diferentes eventos que el servidor envía para informarnos del estado de nuestra petición y que nosotros podemos capturar:

El evento **readystatechange** se produce cada vez que el servidor cambia el estado de la petición.

Cuando hay un cambio en el estado cambia el valor de la propiedad **readyState** de la petición. Sus valores posibles son:

0: petición no iniciada (se ha creado el objeto XMLHttpRequest)

1: establecida conexión con el servidor (se ha hecho el *open*)

2: petición recibida por el servidor (se ha hecho el *send*)

3: se está procesando la petición

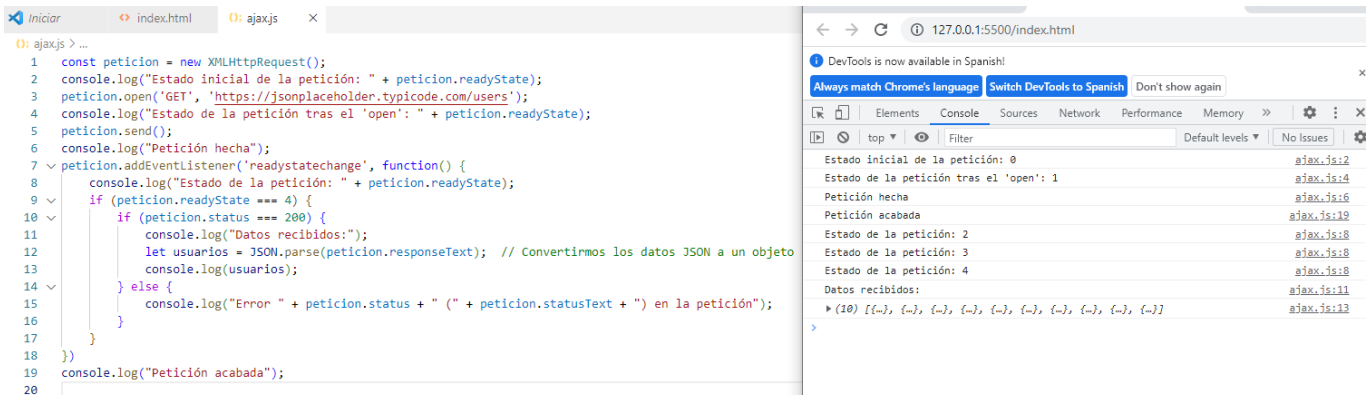
4: petición finalizada y respuesta lista (este es el evento que nos interesa porque ahora tenemos la respuesta disponible). A nosotros sólo nos interesa cuando su valor sea 4 que significa que ya están los datos. En ese momento la propiedad **status** contiene el estado de la petición HTTP (200: *Ok*, 404: *Not found*, 500: *Server error*, …) que ha devuelto el servidor.

Cuando *readyState* vale 4 y *status* vale 200: tenemos los datos en la propiedad **responseText** (o **responseXML** si el servidor los envía en formato XML).

Ejemplo:

```
const peticion = new XMLHttpRequest();
console.log("Estado inicial de la petición: " + peticion.readyState);
peticion.open('GET', 'https://jsonplaceholder.typicode.com/users');
console.log("Estado de la petición tras el 'open': " + peticion.readyState);
peticion.send();
console.log("Petición hecha");
peticion.addEventListener('readystatechange', function() {
    console.log("Estado de la petición: " + peticion.readyState);
    if (peticion.readyState === 4) {
        if (peticion.status === 200) {
            console.log("Datos recibidos:");
            let usuarios = JSON.parse(peticion.responseText);  // Pasamos los datos JSON a un objeto
            console.log(usuarios);
        } else {
            console.log("Error " + peticion.status + " (" + peticion.statusText + ") en la petición");
        }
    }
})
console.log("Petición acabada");
```

El resultado de ejecutar ese código es el siguiente:



Nótese que cuando cambia de estado la petición, *readyState* cambia de valor.

- *readyState* vale 0 al crear el objeto XMLHttpRequest
- *readyState* vale 1 cuando abrimos la conexión con el servidor
- Luego se envía al servidor y es éste el que va informando al cliente de cuándo cambia el estado

MUY IMPORTANTE: La última línea ('Petición acabada') se ejecuta antes que las de 'Estado de la petición'. Ocurre así porque es una **petición asíncrona** y la ejecución del programa continúa sin esperar a que responda el servidor.

Como normalmente no nos interesa saber cada cambio en el estado de la petición, sino que sólo queremos saber cuándo ha terminado de procesarse, tenemos otros **eventos** que nos pueden ser de utilidad:

- **load**: se produce cuando se recibe la respuesta del servidor. Equivale a *readyState===4*. En *status* tendremos el estado de la respuesta
- **error**: se produce si sucede algún error al procesar la petición (de red, de servidor, …)
- **timeout**: si ha transcurrido el tiempo indicado y no se ha recibido respuesta del servidor. Se puede cambiar el tiempo por defecto modificando la propiedad *timeout* antes de enviar la petición.
- **abort**: si se cancela la petición (se hace llamando al método **.abort()** de la petición).
- **loadend**: se produce siempre que termina la petición, independientemente de si se recibe respuesta o sucede algún error (incluyendo un *timeout* o un *abort*).

Ejemplo de código que sí usaremos:

```
const peticion=new XMLHttpRequest();
peticion.open('GET', 'https://jsonplaceholder.typicode.com/users');
peticion.send();
peticion.addEventListener('load', function() {
    if (peticion.status===200) {
        let usuarios=JSON.parse(peticion.responseText);
        // procesamos los datos que tenemos en usuarios
        console.log('correcto:');
        console.log(usuarios);
    } else {
        muestraError(peticion);
    }
})
peticion.addEventListener('error', muestraError);
peticion.addEventListener('abort', muestraError);
peticion.addEventListener('timeout', muestraError);

function muestraError(peticion) {
```

```
    if (peticion.status) {
        console.log("Error "+peticion.status+" ("+peticion.statusText+") en la petición");
    } else {
        console.log("Ocurrió un error o se abortó la conexión");
    }
}
```

Conviene recordad que tratamos con peticiones asíncronas por lo que tras la línea:

```
peticion.addEventListener('load', function() {
```

no se ejecuta la línea siguiente:

```
 if (peticion.status===200) {
```

sino la de:

```
peticion.addEventListener('error', muestraError);
```

Una petición asíncrona es como pedir una pizza: tras encargarla por teléfono, lo siguiente no es ir a la puerta a recogerla, sino que seguimos haciendo cosas por casa y cuando suena el timbre de casa entonces vamos a la puerta a por ella.

Prueba el ejemplo anterior con la petición correcta (tal como está) y cambiando petición.open() para que se produzca algún error.

Prueba a hacer peticiones válidas y erróneas con el fichero *peliculas.json* en json-server. Prueba con el servidor json-server apagado.

Ejemplo:

Realiza peticiones petición GET al archivo de datos “peliculas.json” utilizando el serivor **json-server** que estará a la escucha de peticiones en el puerto 4000 de tu equipo.

a) Una petición debe devolver las películas cuyo director sea Tarantino. Visualiza el resultado en consola.
b) Realizar otra petición GET errónea para visualizar el error.
c) Realiza una petición válida con el servidor apagado.

Ejemplos de envío de datos

Vamos a ver algunos ejemplos de envío de datos al servidor con POST. Supondremos que tenemos una página con un formulario para dar de alta nuevos productos:

El fichero **productos.json** de partida puede ser el siguiente:

```
{
  "productos": [
    {
      "id": 1,
      "name": "Teclado",
      "descrip": "Teclado mecánico Cherry ps/2"
    }
 ]
}
```

Index.html sería:

```
<form id="addProduct">
    <label for="name">Nombre: </label><input type="text" name="name" id="name" required><br>
    <label for="descrip">Descripción: </label><input type="text" name="descrip" id="descrip"
required><br>

    <button type="submit">Añadir</button>
</form>
```

Enviar datos al servidor en formato JSON

```
document.getElementById('addProduct').addEventListener('submit', (event) => {
  const newProduct={
        name: document.getElementById("name").value,
        descrip: document.getElementById("descrip").value,
    }
    const peticion=new XMLHttpRequest();
    peticion.open('POST', 'http://localhost:4000/productos');
    peticion.setRequestHeader('Content-type', 'application/json');  // Siempre tiene
                                          //que estar esta línea si se envían datos
    peticion.send(JSON.stringify(newProduct));  // Hay que convertir el objeto
                                          // a una cadena de texto JSON para enviarlo
    peticion.addEventListener('load', function() {
      // procesamos los datos
    })
  })
```

Para enviar el objeto hay que convertirlo a una cadena JSON con la función **JSON.stringify()** (es la opuesta a **JSON.parse()**). Y siempre que se envían datos al servidor hay que indeicar el formato que tienen en la cabecera de *Content-type*:

```
peticion.setRequestHeader('Content-type', 'application/json');
```

Para evitar problemas: Ejecutar la página en el navegador después de iniciar el servicio json-server en el servidor.

Enviar datos al servidor en formato URIEncoded

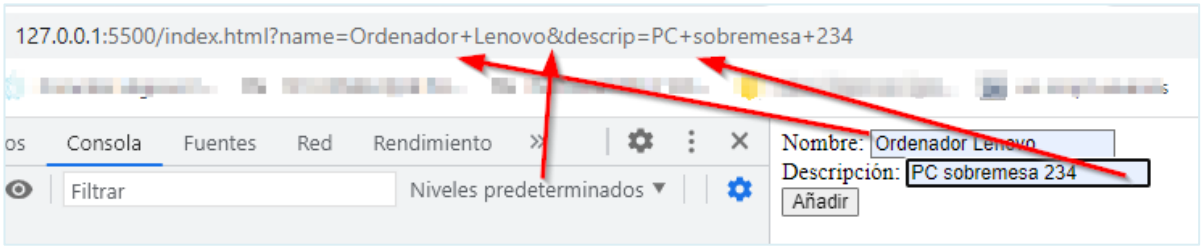
Con el mismo archivo productos.json y el mismo index.html, cambiamos el .js

```
document.getElementById('addProduct').addEventListener('submit', (event) => {
    // Aquí va el código para comprobar que los datos son correctos
    const name=document.getElementById("name").value;
    const descrip=document.getElementById("descrip").value;

    const peticion=new XMLHttpRequest();
    peticion.open('POST', 'http://localhost:4000/productos');
    peticion.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');// formato
    // la siguiente línea lleva & para separar los parámetros
    peticion.send('name='+ encodeURIComponent(name)+'&descrip='+ encodeURIComponent(descrip));
     peticion.addEventListener('load', function() {
      // procesamos los datos
```

```
    })
})
```

En este caso los datos se envían como hace el navegador por defecto en un formulario. Recordad siempre codificar lo que introduce el usuario para evitar problemas con caracteres no estándar y **ataques *SQL Injection***.



Enviar ficheros al servidor con FormData

FormData es una interfaz de XMLHttpRequest que permite construir fácilmente pares de `clave=valor` para enviar los datos de un formulario. Se envían en el mismo formato en que se enviarían directamente desde un formulario ("multipart/form-data") por lo que no hay que poner encabezado de 'Content-type'.

Vamos a añadir al formulario un campo donde el usuario pueda subir la foto del producto:

```
<form id="addProduct">
    <label for="name">Nombre: </label><input type="text" name="name" id="name" required><br>
    <label for="descrip">Descripción: </label><input type="text" name="descrip" id="descrip"
required><br>
    <label for="photo">Fotografía: </label><input type="file" name="photo" id="photo" required><br>

    <button type="submit">Añadir</button>
</form>
```

Podemos enviar al servidor todo el contenido del formulario:

```
  document.getElementById('addProduct').addEventListener('submit', (event) => {
    const peticion=new XMLHttpRequest();
    const datosForm = new FormData(document.getElementById('addProduct'));
    // Automáticamente ha añadido todos los inputs, incluyendo tipo 'file', blob, ...
    // Si quisiéramos añadir algún dato más haríamos:
    formData.append('otrodato', 12345);
    // Y lo enviamos
    peticion.open('POST', 'https://localhost/products');
    peticion.send(datosForm);
    peticion.addEventListener('load', function() {
       // procesamos los datos aquí
    })
  })
```

También podemos enviar sólo los campos que queramos:

```
  document.getElementById('addProduct').addEEventListener('submit', (event) => {
```

```
    const formData=new FormData();  // creamos un formData vacío
    formData.append('name', document.getElementById('name').value);
    formData.append('descrip', document.getElementById('descrip').value);
    formData.append('photo', document.getElementById('photo').files[0]);

    const peticion=new XMLHttpRequest();
    peticion.open('POST', 'https://localhost/products');
    peticion.send(formData);
    peticion.addEventListener('load', function() {
        // procesamos los datos aquí
    })
})
```

Más información de cómo usar formData en MDN web docs.