Prácticas JS

AJAX

Práctica 1: Almacén

No tiene sentido que cada vez que cargo mi página no tenga datos y cada vez que la cierro se pierdan todos. Deberíamos almacenarlos en una base de datos y crear una aplicación del lado servidor para acceder a dichos datos (un API-REST).

Preparación del entorno

Vamos a utilizar un API-REST ya hecho que utiliza ficheros JSON para almacenar los datos. Se trata del servicio json-server

Instalación (-g global a todos nuestros proyectos): npm install -g json-server
Nota: Es posible que tengas que lanzar el comando con permisos de superusuario (sudo) en equipos UNIX o administrador (abre consola admin) en Windows.

Ejecución (indicando el .json del que servirá los datos): json-server datos.json

URL acceso a la API del json-server: http://localhost:3000

Si queremos que una tabla de json-server tenga un **campo autonumérico** debe llamarse obligatoriamente **id**.

Más info sobre json-server: aquí

Para familiarizarnos con esta API (o con cualquier otra) vamos a instalar una extensión en nuestro navegador que nos permita enviar peticiones API-REST y ver qué se recibe del servidor. Hay muchas diferentes como:

- Chrome: REST client

También se puede utilizar la aplicación Postman.

Una vez hecho probaremos a hacer peticiones a nuestro servidor JSON para añadir, modificar o borrar datos del mismo y familiarizarnos así con la API.

Guardar productos

Ahora vamos a hacer que los productos del almacén se guarden de forma persistente. Al cargar la página cargaremos todos los productos del almacén y cada vez que añadamos un nuevo producto éste se guardará en el servidor. Si se modifica o elimina el producto, también se modificará o eliminará en el servidor.

Si no queremos hacer grandes cambios en nuestra aplicación podemos conservar el modelo que tenemos (un 'Store' en memoria con un array de productos), pero además:

- Al cargar la página haremos una petición al servidor pidiendo los productos a la API, los añadiremos al 'Store' que tenemos y los pintaremos.
- Al añadir un nuevo producto haremos una petición a la API para que lo añada y, si todo va bien, lo añadiremos a nuestro 'Store' y lo mostraremos.

- Al modificar un producto, haremos una petición a la API para que lo modifique y, si todo va bien, se modificará en el 'Store' y mostramos los cambios en él.
- Al eliminar un producto, haremos una petición a la API para que lo elimine y, si todo va bien, se eliminará del 'Store' y de la vista.

Para simplificar algo el código podemos eliminar las comprobaciones que hacen los métodos de Store (addProduct y changeProduct comprueban que los datos recibidos sean correctos, delProduct comprueba que no tenga unidades y changeProductUnits comprueba que el producto no quede con unidades negativas), porque todo eso ya se ha validado en el controlador al introducir los datos en el formulario.

Selecciona una de las siguientes técnicas para crear las peticiones AJAX:

Promesas

Hemos visto el problema que plantea que Ajax sea asíncrono: cuando el usuario quiere añadir un nuevo producto se hace la petición Ajax, pero hasta que el servidor no conteste, no puede añadirse a los datos locales ni pintarse en la página. Por tanto, tenemos que llamar al código que pinta el producto dentro de la función de la API, o bien, crear una función a la que llamar desde allí para que lo haga.

La solución es implementar la API con *promesas* y así nuestro código vuelve a quedar bien estructurado.

Más info

fetch

La API *fetch* proporciona una interfaz para realizar peticiones Ajax mediante el protocolo HTTP, que devuelve en forma de promesas.

Más info

async/await

Instrucciones introducidas en ES2016 que permiten escribir el código de peticiones asíncronas como si fueran síncronas, lo que facilita su comprensión. Utilizando *async/await* junto con *fetch* quedará mucho más claro.

Más info