Ejercicio Entregable 8

Gestión de tareas con AJAX y Servicios Web

El ejercicio de esta hoja es entregable y puede realizarse en el laboratorio. Los ejercicios entregables realizados a lo largo del curso computan el 10 % de la nota final, pero su entrega no es obligatoria para aprobar la asignatura. La entrega se realiza por parejas.

Fecha límite de entrega: 12 de enero de 2018.

La entrega se realizará a través del Campus Virtual.

Las aplicaciones web que hemos desarrollado en las dos entregas anteriores y en la Práctica 1 obedecen a un modelo particular de aplicaciones web, que es el llamado MPA (*Multiple Page Application*). En este modelo, cada vez que el usuario/a de nuestra web realiza una acción, el servidor devolvía al navegador una página HTML con el resultado de esta operación. Es decir, la interacción con esta aplicación web se basa en la navegación por una sucesión de páginas.

En este ejercicio desarrollaremos una aplicación web utilizando otro modelo distinto: el modelo SPA (*Single Page Application*). En este modelo, la interacción entre cliente y el servidor se realiza a través de una única página web, la cual el navegador irá transformando dinámicamente. Cada vez un usuario/a realiza una acción con la herramienta, el cliente realiza una petición al servidor. Sin embargo, la respuesta del servidor ya no consiste en una página HTML entera con el resultado de la acción, sino en una representación JSON de dicha respuesta. El cliente, al recibir esta respuesta, actualiza los elementos de la página HTML actual conforme al resultado de la operación.

La temática de este ejercicio es la misma que en los ejercicios anteriores: un gestor de tareas. Sin embargo, ahora implementaremos este gestor mediante una aplicación SPA. Para realizar este ejercicio no es necesario haber realizado los ejercicios anteriores, pues en el CV tendréis una plantilla sobre la que empezar. Para concentrarnos en este nuevo modelo sin perdernos en detalles innecesarios, vamos a realizar las siguientes simplificaciones con respecto a las entregas anteriores:

- Eliminamos la posibilidad de que una tarea tenga asociada una o varias etiquetas. Una tarea consiste, solamente, en un texto.
- Tampoco realizaremos distinción entre tareas completadas y no completadas, de modo que el atributo `done` de nuestras tareas desaparece. Al lado de cada tarea habrá un botón `[Eliminar]` que borrará directamente el elemento correspondiente de la lista de tareas.
- No es necesario realizar gestión de usuarios. Supondremos que existe una única lista de tareas compartida entre todos los usuarios/as que accedan a la aplicación.
- Las tareas no se guardarán en una base de datos MySQL, sino en un array de objetos, que será accesible mediante una variable global en el servidor. El valor inicial del array será el siguiente:

```
let tasks = [
```

```
    {
        id: 1,
        text: "Preparar práctica PDAP"
    },
    {
        id: 2,
        text: "Mirar fechas congreso"
    },
    {
        id: 3,
        text: "Ir al supermercado"
    },
    {
        id: 4,
        text: "Mudanza"
    }
];
```

Tenemos un array con cuatro objetos. Cada uno de ellos representa una tarea, que consiste en un texto y un identificador numérico `id`, que será el que utilicemos para realizar operaciones sobre la misma (por ejemplo, para eliminarla). Suponemos, también, que existe una variable global `idCounter` que contiene el identificador que se asignará a la próxima tarea que se inserte. Esta variable se incrementará con cada inserción. Con esto pretendemos simular el comportamiento de los campos `AUTO_INCREMENT` en MySQL.

En la Figura 1 se muestra la nueva interfaz simplificada. El código HTML correspondiente puede encontrarse en el fichero `public/tasks.html`. Este documento hace refererencia, mediante la etiqueta `<script>`, a un fichero llamado `index.js`, que contiene el código a ejecutar en el lado del cliente. Para realizar esta práctica debes modificar dos ficheros:

- `app.js`, con la funcionalidad del lado del servidor (*back-end*).
- `public/js/index.js`, con la funcionalidad del lado del cliente (*front-end*).

1 Preparación del proyecto

Descarga desde el Campus Virtual el proyecto Node correspondiente a este ejercicio. Contiene:

1. El fichero `package.json` con la descripción y dependencias del proyecto. Instala dichas dependencias mediante `npm install`.
2. Un fichero `config.js`, con el que puedes indicar el puerto TCP en el que el servidor escuchará.
3. Un fichero `jsconfig.json`, cuyo único cometido es facilitar autocompletado para *jQuery* dentro de *Visual Studio Code*.
4. Un fichero `app.js` que crea un servidor *Express.js* y lo arranca en el puerto indicado en `config.js`.

Figura 1: Interfaz simplificada de la aplicación.

5. Una carpeta `public` con los recursos estáticos de la aplicación: fichero HTML con la única página de la aplicación, hoja de estilo, imágenes, y ficheros Javascript a ejecutar en el cliente.

Arranca el servidor y comprueba que funciona correctamente, accediendo a la URL `http://localhost:3000/`, que redirige a `http://localhost:3000/tasks.html`.

2 Desarrollo en el lado del servidor: `app.js`

Comenzaremos desarrollando la funcionalidad del lado del servidor. Nuestro servidor proporciona tres servicios al navegador:

- **Servicio 1**. Devolver todas las entradas contenidas en la lista de tareas:

  Método: `GET`

  URL: `/tasks`

  Parámetros de entrada: Ninguno

  Códigos de respuesta: `200` (OK)

  Tipos de resultado: JSON

  Resultado: Array con las tareas de la lista. Cada tarea es un objeto con dos atributos: `id` y `text`.

- **Servicio 2**. Añadir una nueva entrada a la lista de tareas.

  Método: `POST`

  URL: `/tasks`

  Parámetros de entrada: Un objeto JSON con un único atributo (`text`), que contendrá el texto de la tarea a insertar.

Códigos de respuesta: `200` (OK)

Tipos de resultado: JSON

Resultado: Objeto tarea insertado, con dos atributos: `id` y `text`. El identificador de la tarea será generado automáticamente por el servidor. El texto será el mismo que el especificado en el parámetro de entrada.

- **Servicio 3**. Devolver todas las tareas contenidas en la lista de tareas global:

  Método: `DELETE`

  URL paramétrica: `/tasks/:id`

  Parámetros de entrada: El parámetro `:id` de la URL paramétrica contiene el identificador de la tarea a eliminar.

  Códigos de respuesta: `200` (OK) si el `:id` indicado es un número entero, o `400` (Bad request) en caso contrario.

  Tipos de resultado: JSON

  Resultado: Un objeto vacío `{}`.

De estos tres servicios, el segundo ya viene implementado en la plantilla. Implementa los dos restantes. Antes de continuar con la siguiente apartado, es recomendable comprobar el funcionamiento de estas tres operaciones mediante un cliente REST como, por ejemplo, *Advanced REST Client*.

3 Desarrollo en el lado del cliente: `public/js/index.js`

En este fichero encontarás una función `taskToDOMElement(task)` que, a partir de un objeto con dos atributos (`id` y `text`) representando la información de una tarea, construye en elemento `<li>` con el contenido de dicha tarea y devuelve una selección al mismo. Este elemento contiene, además, un atributo personalizado con el identificador de la tarea. Este atributo se ha asignado mediante el método `.data()` explicado en clase.

El fichero `index.js` también incluye un manejador de inicialización del DOM, que se limita a llamar a la función `loadTasks()` y a asignar sendos manejadores de eventos a los botones `[Eliminar]` y al botón `[Añadir]` de la página. Debes implementar las siguientes tres funciones:

- `loadTasks()`

  Solicita al servidor la lista de tareas (mediante una petición AJAX) e inserta cada una de ellas en el DOM para que se visualicen en la página.

- `onRemoveButtonClick(event)`

  Maneja los eventos de clic en un botón `[Eliminar]`. Debe enviar una petición AJAX para eliminar la tarea correspondiente del servidor y eliminar dicha tarea del DOM.

- `onAddButtonClick(event)`

  Maneja los eventos de clic en el botón `[Añadir]`. Debe obtener la cadena introducida en el cuadro de texto situado a la izquierda de dicho botón. Si esta cadena es no vacía, realizará una petición AJAX al servidor para insertar la tarea en la lista de tareas. Además, debe añadir el elemento del DOM correspondiente.

En estas dos últimas operaciones (eliminación e inserción), la actualización del DOM debe hacerse modificando únicamente los elementos involucrados, es decir, eliminando el `<li>` correspondiente en el primer caso, y añadiendo un nuevo `<li>` en el segundo caso. No se permite eliminar todos los `<li>` y llamar a `loadTasks()` para obtener la lista actualizada.