Práctica 4

Descarga el zip incluido que contiene los archivos necesarios para este ejercicio. El ejercicio será una continuación de la semana 3 pero añadiendo la programación orientada a objetos y la comunicación AJAX con el servidor.

Estos son los archivos que debes completar. No debes modificar el HTML. Fíjate sobre todo, en los ejemplos 9 y 10 que son similares a lo que se pide.

constants.js

Define aquí constantes globales para toda la aplicación, como la dirección al servidor web.

http.class.js

Clase Http con los métodos necesarios para facilitar las llamadas AJAX a los servicios web. Este archivo ya está implementado y no hace falta tocar nada (es lo mismo que hay en el archivo “Indicaciones”, apartado “Encapsular llamadas AJAX con clases”).

evento-service.class.js

Crea una clase llamada EventoService. Esta clase será la encargada de llamar a los servicios web (utilizando a su vez la clase Http) .

El constructor no recibirá ningún parámetro y creará un objeto de la clase Http.

Método getEventos() → Llamará a http://SERVER/eventos por ‘GET’. El servidor devolverá un objeto JSON con una propiedad llamada eventos que contendrá un array de eventos como valor. Devuelve el array de eventos recibido. Ejemplo de respuesta recibida:


```
{
  "eventos": [
    {
      "id": 2,
      "name": "Este es un evento",
      "date": "2018-09-24",
      "description": "Descripción del evento",
      "price": 24.95,
      "image":
      "http://localhost:3000/img/2aafb87911d7641787df5fe0f833777a49f64.jpg"
```

```
    },
    {
        "id": 3,
        "name": "Otro evento",
        "date": "2018-09-29",
        "description": "Descripción de otro evento",
        "price": 15,
        "image":
        "http://localhost:3000/img/8eb66385c92fa52d27ba838bbf9863212ce90.jpg"
    }
  ]
}
```


Método post(evento) → Llamará a http://SERVER/eventos mediante POST, y enviará como datos el evento recibido. El servidor responderá con un JSON que contendrá un campo llamado evento, que será el evento insertado con la id generada y la url de la imagen.


```
{
  "evento": {
        "id": 4,
        "name": "Fin de año",
        "date": "2017-12-31",
        "description": "Si bebes no conduzcas",
        "price": 45.95,
        "image":
        "http://localhost:3000/img/7f3614831ca8187435c087bd6040082375428.jpg"
  }
}
```

Método delete(idEvento) → Llamará al servicio http://SERVER/eventos/:id usando DELETE (:id será la id del evento a borrar). El servidor devolverá una respuesta con la propiedad id (number) si se ha borrado correctamente, o un error en caso contrario. Devuelve dicha id.

```
{
   "id": 4,
}
```

index.js

Tendrá casi la misma funcionalidad que en el ejercicio anterior pero esta vez utilizando servicios web.

Lo primero que haremos será crear un objeto de la clase EventoService y llamar al método Evento.getEventos() y asignar el array de eventos recibido a una variable global. A continuación muestra todos los eventos del array al usuario, generando el HTML de cada card como en el ejercicio anterior incluyendo además un botón para eliminar el evento (mira el código de index.html para un ejemplo de estructura).

Debemos asignar el evento 'click' al botón de borrar, que llamará al método delete() del objeto EventoService, y si el evento se borra correctamente (no hay error), debemos eliminarlo del array global de eventos y del HTML (puedes eliminar el elemento del HTML directamente, o volver a reconstruirlos todos a partir del array que ya no contiene el evento).

```
...
botonBorrar.addEventListener("click", e => {
   // Borrar evento
});
...
```

También controlaremos el envío del formulario, validando que no haya ningún campo vacío y enviando el evento al servidor (crea un objeto JSON que represente al evento a partir de dicho formulario y llama al método post() del objeto EventoService). Si todo ha ido bien, añade el evento al array global y muéstralo. Si hay algún error, sería suficiente con imprimirlo por consola.

Opcional

Usa async/await para gestionar las promesas en lugar de usar then.