Bloque 3

Programación con JavaScript

Ejercicio bloque 3

Programación con JavaScript
Cefire 2017/2018
Autor: Arturo Bernal Mayordomo

Índice

Ejercicio bloque 3

Descarga el zip incluido que contiene los archivos necesarios para este ejercicio. El ejercicio será una continuación del bloque 2 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.

constants.js

Define aquí constantes globales para toda la aplicación, como la dirección al servidor web, o la ruta a las imágenes del servidor (SERVER + '/img').

http.class.js

Crea una clase llamada HTTP con un método estático llamado **ajax**.

- Este método recibirá una cadena con el método HTTP ('GET', 'POST', 'PUT', 'DELETE'), la url del servicio web y los datos a enviar al servidor (Si es GET o DELETE será null).
    - Los datos se enviarán como string al servidor, por lo que si el método los recibe como objeto, debe transformarlos a string con **JSON.stringify**.
- Devuelve una promesa que se resolverá cuando el servidor responda correctamente (status === 200), devolviendo el texto de respuesta en formato JSON. Si hay algún error la promesa será rechazada.

event.class.js

Crea una clase llamada EventItem (la clase Event ya existe en JavaScript).

- El constructor recibirá un objeto JSON con la información del evento como único parámetro, o los valores por separado (**name, date, description, image, price**), como prefieras. Asigna las propiedades al objeto actual (this), convirtiendo la fecha (date) a objeto **Date**, y el precio a **number**.
- Método **static getEvents()** → Llamará a http://SERVER/exercise3/events usando 'GET'. El servidor devolverá un objeto JSON con una propiedad llamada **events** que contendrá un array de eventos como valor.


```
{
  "events": [
    {
      "id": 2,
      "name": "This is an event",
      "date": "2018-09-24 00:00:00",
      "description": "This is the description of the event\nWhich has many
lines\nAt least three....",
      "price": 24.95,
      "image": "206/2aafb87911d7641787df5fe0f833777a49f64.jpg"
```

```
    },
    {
      "id": 3,
      "name": "Another event",
      "date": "2018-09-29 00:00:00",
      "description": "This event has also a description\nWith two lines",
      "price": 15,
      "image": "b71/8eb66385c92fa52d27ba838bbf9863212ce90.jpg"
    }
  ],
  "ok": true
}
```

Convierte el array JSON a un array de objetos EventItem y devuélvelo.

```
static getEvents() { // Este método también devuelve una promesa
    return Http.ajax('GET', `${SERVER}/events`)
        .then(// Comprobar la respuesta y devolver el array de EventItem);
}
```

- Método **post()** → Llamará a http://SERVER/exercise3/events mediante POST, y enviará como datos el propio objeto (this). El servidor responderá con un JSON que contendrá un campo llamado **ok** (booleano) y otro llamado **event**, que será el evento insertado con la id generada y la url de la imagen.

```
{
  "event": {
    "id": 4,
    "name": "Year's end",
    "date": "2017-12-31T20:00:00",
    "description": "We'll get drunk until our liver explodes.\nPromised!",
    "price": 45.95,
    "image": "662/7f3614831ca8187435c087bd6040082375428.jpg"
  },
  "ok": true
}
```

Si ok es true, transforma el evento a objeto EventItem y devuélvelo. Si es false, lanza un mensaje de error (throw).

```
post() { // Devuelve una promesa
    return Http.ajax('POST', `${SERVER}/events`, this)
        .then((response) => {
            // Comprobar la respuesta y devolver un EventItem o lanzar un error
        });
}
```

- Método **delete** → Llamará al servicio http://SERVER/exercise3/events/:id usando DELETE (:id será la id del evento actual). El servidor devolverá una respuesta con la propiedad **ok** (booleano) indicando si se ha borrado correctamente. Devuelve true o lanza un error en función de eso.

```
{
  "ok": false,
  "error": "Event not found"
}
```

```
delete() { // Devuelve una promesa
    return Http.ajax('DELETE', `${SERVER}/events/${this.id}`, this)
```

```
    .then((response) => {
      // Procesar la respuesta
    });
  }
```

- **Método toHTML** → Devolverá el elemento con la estructura HTML del evento creada dentro. Esta estructura será la misma que en el ejercicio anterior e incluirá un botón para eliminar el evento (mira el código de index.html para un ejemplo de estructura).

```
toHTML() {
    let card = document.createElement("div");
    card.classList.add("card");

    // Añadir el resto de la estructura
    return card;
  }
```

new-event.js

Aquí se validará el formulario de la página **new-event.html** (mismo formulario que en el ejercicio 2), y lo enviará al servidor (creando un objeto EventItem a partir de los campos y llamando a POST).

Si el evento se añade, redirige a index.html. Usa location.assing("index.html"). Si hay algún error, muéstralo (aunque sea con un alert).

index.js

Lo primero que haremos será llamar a **EventItem.getEvents()** y asignar el array de eventos recibido a una variable global. Llama a una función que crearás en este archivo llamada showEvents(events), que borrará todo el contenido del elemento con la id **eventsContainer** e insertará los eventos recibidos. Esto lo hará igual que en el ejercicio 2 (2 cards por cada card-deck), con la diferencia que la estructura de la card ahora la obtiene llamando al método **toHTML** de cada evento.

Además, cuando obtengamos el HTML de cada evento, debemos asignar el evento ‘click’ al botón de borrar. Llamará al método **delete** del evento actual, y si el evento se borra correctamente, debemos eliminarlo del array global de eventos y volver a llamar a **showEvents**.

```
...
let eventCard = event.toHTML();
eventCard.querySelector('.card-title .btn').addEventListener("click", e => {
      // Borrar y redibujar los eventos
});
...
```

Optativo

Esta parte es opcional y dirigida a quien disponga de más tiempo para hacerla.

Al hacer clic sobre el botón con la id **orderDate**, ordenará el array global de eventos por fecha y volverá a llamar a **showEvents**. Un clic en el botón con la id **orderPrice** hará lo mismo pero ordenando por precio.

En el input con la id **searchEvent**, cuando el usuario escriba algo (evento keyup), filtra el array global dejando solo los elementos que contienen la cadena escrita en el cuadro de búsqueda (sin diferenciar mayúsculas de minúsculas). Guarda estos eventos en un array auxiliar y pásaselo a showEvents.