UNIT 1

JAVASCRIPT

Part 3 - Exercise

Client-side Web Development
2nd course – DAW
IES San Vicente 2024/2025
Author: Arturo Bernal Mayordomo

Índice

Exercise JavaScript part 3

Download the included zip which contains the necessary files. It’s a more complete version of the week 2 exercise . You can’t modify any HTML file (only JS). Use **import/export** to use other functions, classes, etc from other files

When working with promises, it’s ok to use **then** or **async/await.**

constants.js

Here you’ll define global constants for the app like the server’s url.

http.js

This class contains the necessary HTTP methods for calling web services using GET, POST, PUT and DELETE. All methods return a promise with the response’s data (or null if there’s no data), or will throw an error if the server responds with an error.

events-service.js

Create a class called **EventsService**. This class will call to web services related to events (using the Http class).

- Instantiate a Http class object inside a private attribute we'll use to make ajax calls

- **getEvents()** → Will call http://SERVER/events using ‘GET’. The server will return an object containing a property called events with the array of events:

```
{
  "events": [
    {
      "id": 2,
      "name": "This is an event",
      "date": "2018-09-24",
      "description": "This is the description of the event\nWhich has many
lines\nAt least three....",
      "price": 24.95,
      "image": "http://localhost:3000/img/events/1632085936837.jpg"
    },
    {
      "id": 3,
      "name": "Another event",
      "date": "2018-09-29",
      "description": "This event has also a description\nWith two lines",
      "price": 15,
      "image": "http://localhost:3000/img/events/1632085933627.jpg"
    }
  ]
}
```

Return the array of events, not the full response (Remember that it returns a Promise).

- **post(event)** → Will call http://SERVER/events using 'POST', and send the event object. Example of an event object:

```
{
  "title": "This is a a new event",
  "description": "Description for the new event",
  "date": "2021-12-03",
  "price": 25.35,
  "image": "Image in base 64"
}
```

  The server will return a JSON object with the inserted event (which will contain the correct id and image's url):

```
{
  "event": {
    "id": 5,
    "title": "This is a a new event",
    "description": "Description for the new event",
    "image": "http://localhost:3000/img/events/1632141872433.jpg",
    "price": 25.35,
    "date": "2021-12-03"
  }
}
```

  Return the event object (not the full response).

- **delete(id)** → Will call http://SERVER/events/:id using 'DELETE' (**:id** is the id number of the event). The server will return an empty response if everything was ok, or an error if something went wrong.

new-event.js

This script will validate the form in **new-event.html** (same as exercise 2) and send it to the server (in JSON format). If the form is correctly validated, call the **post()** method of the **EventService** class.

If the event was inserted (no error), redirect to **index.html** (use location.assign). If there was an error, show an alert message and don't go anywhere.

index.js

The first thing this page will do is instantiate an **EventService** object and call **getEvents**(). Put the received events in a global array. It will send that array to a function called **showEvents(events)**, which will remove everything in the **#eventsContainer** element and insert the new events HTML there.

For every event card created, control the click event on the delete button inside it. It will call the **delete** method on EventsService and if the server deleted the event, delete it from the global array and call showEvents again (or remove the card from the DOM). Before deleting an event, ask the user using a confirm dialog.

Optional (2 points extra)

A click on **#orderDate** button, will order the global array of events by date and call **showEvents**. A click on **#orderPrice** will do the same but ordering by price. Before ordering, delete the value of the **#searchEvent** input element.

On the **#searchEvent** element. Every time the user writes something (**input** event), filter the global array and leave only the events which their title (you can also include description) contains the string written (case insensitive). Call **showEvents** with the resulting array.

Web services

Web services are published here → https://api.fullstackpro.es/svtickets-lite

If you prefer to run the services in your local computer, the code is in this repository: https://github.com/arturober/svtickets-services-lite. Instructions:

- Install Git if not installed.
- Install NodeJS if not installed (LTS version)
- Open VSCode with no opened folder. Select Clone Git repository. The url is: https://github.com/arturober/svtickets-services-lite.git
- Install MySQL (MariaDB) on your local computer (using XAMPP for example).
- Import the **SQL/svtickets_lite.sql** file using phpMyAdmin.
- Open the services folder with VSCode.
    - Execute **npm install.**
    - Edit **src/mikro-orm.config.ts** file and set the correct user and password for the database (user: 'root' and password: '' in XAMPP).
- Execute **npm start**. Services will run on http://localhost:3000

There's a Postman collection also available to test the web services before implementing using them in your application:

https://www.postman.com/downloads/