UNIT 1

JAVASCRIPT

Part 4 - Exercise

Client-side Web Development
2nd course – DAW
IES San Vicente 2025/2026
Arturo Bernal / Rosa Medina

Índice

Exercise JavaScript part 4

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

constants.js

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

http.class.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.

provinces.service.js

Create a class called **ProvincesService**. This class will have the necessary methods to get the list of provinces and towns from the server.

- Instantiate a Http class object inside a private attribute we'll use to make ajax calls
- **getProvinces** → Call http://SERVER/provinces using GET. The server will return an object containing a property (provinces) with the array:

```
{
  "provinces": [
    {
      "id": 1,
      "name": "Araba/Álava"
    },
    {
      "id": 2,
      "name": "Albacete"
    },
    {
      "id": 3,
      "name": "Alacant/Alicante"
    },
    ...
  ]
```

```
}
```


- **getTowns** → Call http://SERVER/provinces/:idProvince/towns using GET. It will return an object with an array inside of towns property:


```
{
  "towns": [
    {
      "id": 1030015,
      "name": "l'Atzúbia",
      "longitude": -0.15173035,
      "latitude": 38.84743029,
      "province": 3
    },
    {
      "id": 1030020,
      "name": "Agost",
      "longitude": -0.63891979,
      "latitude": 38.43949316,
      "province": 3
    },
    ...
  ]
}
```


properties.service.js

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

- **getProperties** → Call *http://SERVER/properties* using 'GET'. The server will return an object containing a property called properties with the array:


```
{
  "properties": [
    {
      "id": 8,
      "address": "Calle patata 15",
      "title": "New Property",
      "description": "Description\nOther line",
      "sqmeters": 140,
      "numRooms": 5,
      "numBaths": 3,
      "price": 450000,
      "mainPhoto": "http://localhost:3000/img/properties/1759181135151.jpg",
      "createdAt": "2025-09-29T21:25:35.151Z",
      "status": "selling",
      "town": {
        "id": 1010164,
        "name": "Bernedo",
```

```
            "longitude": -2.49825439,
            "latitude": 42.62676989,
            "province": {
               "id": 1,
               "name": "Araba/Álava"
             }
         }
      },
      ...
   ]
}
```


- **insertProperty** → Call *http://SERVER/properties* using 'POST', and send the property object. Example of a correct property to send:


```
{
   "title": "New Property",
   "description": "Description\nOther line",
   "price": 450000,
   "address": "Calle patata 15",
   "sqmeters": 140,
   "numRooms": 5,
   "numBaths": 3,
   "townId": 1010164,
   "mainPhoto": "Image in base64"
}
```


  The server will return a JSON object with the inserted property (which will contain the correct id and image's url). Return the property object (not the full response).

- **deleteProperty** → Will call *http://SERVER/properties/:id* using 'DELETE' (**:id** is the id of the property). The server will return an empty response (204) if everything was ok, or an error if something went wrong.

new-property.js

This script will validate the form in this page (same as exercise 2) and send it to the server (as a JSON object). Call the corresponding method of the **PropertiesService** class.

First of all, you'll have to load the provinces and add them to the **select#provinces** element. When a new province is selected, load the corresponding towns inside the **select#towns** element. The value of each option will be the id.

Important: Preserve the first option (value="") when adding data.

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

You don't need to reset the form anymore, because the page will be destroyed when you redirect to index.html.

There are a few changes from previous exercise (field names and new description field). See General considerations section.

index.js

Call the corresponding method from **PropertiesService** to load the events (if you use **async/await**, do this in an independent function!). Create a card for every property and add it to the DOM. The object is a bit different from what we create in the form (See General considerations section)

For every event card created, control the click event on the delete button inside it. It will call the delete method on **PropertiesService** and if the server deleted the property, remove the card from the DOM. Before deleting a property, ask the user using a confirm dialog.

General considerations

- When working with promises, it’s ok to use **then** or **async/await**.
- The property field for the photo (JSON) is called **mainPhoto**.
- There's a new field (included in the form) called **description** (the server needs it). It's just a required field.
    - You also need to show the description in the card.
- You must include a field called **townId** when inserting a new property. You don't need to send the province.
- Look carefully at the properties object structure you get from the server (to access the name of the province and town).
- When using liveServer, you must add the extension (.js) to the imported files. Other options we'll use in the future won't need it.

Web services

To run the services in your local computer, the code is in this repository: https://github.com/arturober/inmosanvi-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/inmosanvi-services-lite.git*
- Open the services folder with VSCode.
  - Execute **npm install** and **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/