Week 3 – Exercise

Download the included zip which contains the necessary files. It's a more complete version of the week 2 exercise (you can reuse code from that exercise). You can't modify any HTML file (only JS).

You can execute web services and the database in your computer (http://localhost:3000) or you can use the deployed web services in my server (http://arturober.com:5007)

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.

product.class.js

Create a class called **Product**.

- The constructor will receive a JSON object with the product's information. Assign its values (**id, title, description, category, price**) to the current object values. If there's no id, assign **null**.
- **static getAll()** → Will call http://SERVER/products using 'GET'. It will return an object containing a property called **products** with an array of products.
  - Although the products contain more properties (for the future projects), we will only use those which are specified below.

```
{
    "products": [
        {
            "id": 50,
            "title": "Useful suitcase",
            "description": "You'll never know when you're going to need it\nIt
has sentimental value",
            "price": 23,
            "mainPhoto":
"http://arturober.com:5007/img/products/1601282534779.jpg",
        },
        {
            "id": 51,
            "title": "Toilet minigolf",
            "description": "Your phone is out of battery?\nDon't worry!",
            "price": 99.99,
            "mainPhoto":
"http://arturober.com:5007/img/products/1601282689849.jpg",
        }
    ]
}
```

Map the JSON objects to a new array containing **Product** objects and return that.

- **post()** → Will call http://SERVER/products using 'POST', and send the current object (this). The server will return a JSON object with the inserted product (which will contain the correct id and the image url). An example of an object sent to the server.
  - JSON example to send to the server:

```
{
    "title": "hello product",
    "description": "This product is cheap",
    "image": "image in base64",
    "price": 20,
    "category": 1
}
```

- **delete()** → Will call http://SERVER/products/:id using 'DELETE' (:id will be the id number of the current product). The server will return 204 response with no content if everything is correct (or an error).

- **toHTML()** → Will return the card HTML object with all the product information.

Add a button at the end of the **div.card-body** element that will delete the product:

```
<div class="text-right">
  <button class="btn btn-danger">Delete</button>
</div>
```

new-product.js

First, you'll need to load all the categories from the server and append them to the **select#category** element. Send a GET request to http://SERVER/categories. The server will return the categories like this:

```
{
    "categories": [
        {
            "id": 1,
            "name": "Electronics"
        },
        {
            "id": 2,
            "name": "Motor and vehicles"
        },
        {
            "id": 3,
            "name": "Sports and hobbies"
        },
        ...
    ]
}
```

Also validate the form in **new-product .html** (same as exercise 2) and send it to the server (in JSON format). If the form is correctly validated (all fields are required), create a new Product object with all the information (you'll need to create a JSON object with the necessary data and send it to the constructor) and call **post()**.

If the product is inserted successfully, redirect to **index.html** (use location.assign). If there is an error, show (for example) an alert message with that error.

index.js

The first thing this page will do is call **Product.getAll()** and put the received products in a global array. It will send that array to a function called **showProducts(products)**, which will remove everything in the **#productsContainer** element and insert the new products there (call **toHTML()** on each product object to get the card HMTL and append it to the container).

On the **#search** element. Everytime the user writes something (keyup event), create a new products array filtering the original (at least by title, description is optional), and call **showProducts** again with the new array.

- **showProducts(products)** → Before adding the products, delete all elements inside the **#productsContainer** element.

AssBook Home Add product

Search

Useful suitcase

You'll never know when you're going to need it
It has sentimental value

Delete

Other 23.00€

Toilet minigolf

Your phone is out of battery?
Don't worry!

Delete

Sports and hobbies 99.99€

Running bike

Is your back ok?
Don't worry. With this bike it will hurt you soon!

Delete

Sports and hobbies 199.00€