**Exercise: HTTP and AJAX**

Problems for exercises and homework for the ["JS Front-End" course @ SoftUni](https://softuni.bg/trainings/4240/js-front-end-october-2023)

**Working with Remote Data**

For the solution of some of the following tasks, you will need to use an up-to-date version of the **local REST service**, provided in the lesson’s resources archive. You can [read the documentation here](https://github.com/softuni-practice-server/softuni-practice-server).

## Requirements

For each task you need to install all dependencies using the “**npm install**” command

Then you can start the front-end application with the “**npm start**” command

You also must also start the **server.js** file in the server folder using the “**node server.js**” command in another console **(BOTH THE CLIENT AND THE SERVER MUST RUN AT THE SAME TIME)**

At any point, you can open up another console and run “**npm test**” inside the **tests subfolder** for the problem to test the **current state** of your application, it’s preferable for **all of your test to pass locally** before you submit to the judge platform, like this:

Text

Description automatically generated

## Blog

Write a program for reading blog content. It needs to make **requests** to the **server** and display **all blog posts** and their **comments**.  
Request URL’s:

Posts - **http://localhost:3030/jsonstore/blog/posts**

Comments - **http://localhost:3030/jsonstore/blog/comments**

**Examples**

The button with ID "**btnLoadPosts**" should make a GET request to "**/posts**". The **response** from the **server** will be an **Object of objects.**

Each object will be in the following format:

{

body: {postBody},

id: {postId},

title: {postTitle}

}

Create an <option> for each post using its object key as value and current object title property as text inside the node with ID "**posts**".





When the button with ID "**btnViewPost**" is clicked, a GET request should be made to:

* “**/comments**“ **-** to obtain all comments. The request will **return** a **Object** of **objects**.



Each object will be in the following format:

{

id: {commentId},

postId: {postId},

text: {commentText}

}

You must find this comments that are for the current post (check the postId property)

Display the post title inside **h1** with ID "**post-title**" and the post content inside **p** with ID "**post-body**". Display **each comment** as a **<li>** inside **ul** with ID "**post-comments**". Do not forget to clear its content beforehand.





## Messenger

Write a JS program that records and displays messages. The user can post a message, supplying a name and content and retrieve all currently recorded messages.

**The url** for the requests - **http://localhost:3030/jsonstore/messenger**

When [**Send**] **button** is clicked you should create a **new** **object** and send a **post** **request** to the given url. Use the following message structure:

{

author: authorName,

content: msgText,

}

If you click over [**Refresh**] **button** you should **get all** messages with **GET** **request** and display them into the textarea. Use the following message format:  
"**{author}: {message}**"

### Examples

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

## Phonebook

Write a JS program that can load, create and delete entries from a Phonebook. You will be given an HTML template to which you must bind the needed functionality.

When the [Load] button is clicked, a GET request should be made to the server to get all phonebook entries. Each received entry should be in a li inside the ul with id="phonebook" in the following format with text "<person>: <phone> " and a [Delete] button attached. Pressing the [Delete] button should send a DELETE request to the server and delete the entry. The received response will be an object in the following format:  
{<key>:{person:<person>, phone:<phone>}, <key2>:{person:<person2>, phone:<phone2>,…} where <key> is an unique key given by the server and <person> and <phone> are the actual values.

When the [Create] button is clicked, a new POST request should be made to the server with the information from the Person and Phone textboxes, the Person and Phone textboxes should be cleared and the Phonebook should be automatically reloaded (like if the [Load] button was pressed).

The data sent on a POST request should be a valid JSON object, containing properties person and phone. Example format:   
**{**

**"person": "<person>",**

**"phone": "<phone>"**

**}**

The url’s to which your program should make requests are:

* GET and POST requests should go to http://localhost:3030/jsonstore/phonebook
* DELETE requests should go to http://localhost:3030/jsonstore/phonebook/:key> , where :key is the unique key of the entry (you can find out the key from the key property in the GET request)

### Screenshots

Graphical user interface, text, application, chat or text message

Description automatically generatedGraphical user interface, text, application, chat or text message

Description automatically generated

## Students

Your task is to implement functionality for creating and listing students from a database. Create a new collection called "**students**",

Each student has:

* firstName - string, non-empty
* lastName - string, non-empty
* facultyNumber - string of numbers, non-empty
* grade - number, non-empty

You need to write functionality for creating students. When creating a new student, make sure you name the properties accordingly.

You will also need to extract students. You will be given an HTML template with a table in it. Create an AJAX request that extracts all the students.

URL for this task: **http://localhost:3030/jsonstore/collections/students**

**Screenshots**

Graphical user interface, application

Description automatically generated

## Locked Profile

In this problem, you must **create a JS program** which **shows** and **hides** the additional information about users, which you can find by making a GET request to the server at address:

http://localhost:3030/jsonstore/advanced/profiles

The response will be an object with the information for all users. Create a profile card for every user and display it on the web page. Every item should have the following structure:

Graphical user interface, text, application

Description automatically generated

Graphical user interface, application

Description automatically generated

When one of the [**Show more**] **buttons** is clicked, the **hiden information** inside the div should be shown, only if **the profile is not locked**! If the current profile is **locked,** nothing should happen.



If the **hidden information is displayed** and we **lock** **the profile again**, the [**Hide it**] button should **not be working**! Otherwise, when the profile is **unlocked** and we click on the **[Hide it]** button, the new fields must hide again.

## Accordion

An **html** file is given and your task is to show **more**/**less** information for the selected article. At the start you should do a **GET** request to the server at adress: **http://localhost:3030/jsonstore/advanced/articles/list** where the response will be an object with the titles of the articles.

By clicking the **[More] button** for the selected **article**, it should **reveal** the content of a **hidden** div and **changes** the text of the button to **[Less]**. Obtain the content by making a **GET** request to the server at adress: **http://localhost:3030/jsonstore/advanced/articles/details/:id** where the response will be an object with property **id**, **title**, **content.** When the same button is clicked **again** (now reading **Less**), **hide** the div and **change** the text of the button to **More**. Link action should be **toggleable** (you should be able to click the button infinite amount of times).

**Example** A picture containing table

Description automatically generated

Graphical user interface, application

Description automatically generated

Every item should have the **following structure**:

Graphical user interface, text, application

Description automatically generated

You are allowed to add new attributes, but do not change the existing ones.

## Fisher Game

Use the provided skeleton and the server.

Graphical user interface, website

Description automatically generated

### Login User

The **Login** page contains a form for existing user authentication. By given **email** and **password,** the app should login an existing user.

* After a **successful login** the **home page should be displayed**.
* In case of **error**, an appropriate error **message** should be displayed and the user should be able to fill in the login form again.
* Keep the user data in the browser's **session or locale storage**.
* POST request: [**http://localhost:3030/users/login**](http://localhost:3030/users/login)
* Payload to test in postman:  
  {  
   “email”: [george@abv.bg](mailto:george@abv.bg),

“password”: “123456”,

}

Graphical user interface, application

Description automatically generated

If the user is not logged in, all the buttons should be disabled except the "LOAD" button.

### Register User

By given **email** and **password,** the app should register a new user in the system.

* In case of **error** (eg. invalid username/password), an appropriate error **message** should be displayed, and the user should be able to **try** to register again.
* Keep the user data in the browser's **session or local storage**.
* After a **successful registration** the **home page should be displayed**.
* POST request: **http://localhost:3030/users/register**

Graphical user interface

Description automatically generated

### Logout

The logout action is available to **logged-in users**. Send the following **request** to perform logout:

* Get: **http://localhost:3030/users/logout**

Required **headers** are described in the documentation. Upon success, the **REST service** will return an **empty response**. Clear any session information you’ve stored in browser storage.

If the logout was successful, **redirect** the user to the **Home** page and change the button in navigation.

### Load catches

By clicking it you have to load all the catches from the server and render them like on the picture:

* Pressing the **[Load]** button should **list all** catches. (For all users)
* Pressing the **[Update]** button should send a **PUT** request, updating the catch in **http://localhost:3030/data/catches/:id**. (**Only for the creator of the catch**)
* Pressing the **[Delete]** button should delete the catch from **http://localhost:3030/data/catches/:id**. (**Only for the creator of the catch**)
* Pressing the [Add] button should submit a new catch with the values of the inputs in the fieldset with id="addFrom". (**Only for logged in users**)
* Button [Add] should be disabled in there are no logged in user.
* Buttons [Update] and [Delete] should be disabled if the currently logged-in user is not the author of the catch.

Graphical user interface

Description automatically generated

Each catch should have:

* angler - string representing the name of the person who caught the fish
* weight - floating-point number representing the weight of the fish in kilograms
* species - string representing the name of the fish species
* location - string representing the location where the fish was caught
* bait - string representing the bait used to catch the fish
* captureTime - integer number representing the time needed to catch the fish in minutes

Use the following requests to access your data:

* **List All Catches**
  + Endpoint - **http://localhost:3030/data/catches**
  + Method: GET
* **Create a New Catch**
  + Endpoint: **http://localhost:3030/data/catches**
  + Headers: X-Authorization: “….” (accessToken after login)
  + Method: POST
  + Request body (JSON): {"angler":"…", "weight":…, "species":"…", "location":"…", "bait":"…", "captureTime":…}
* **Update a Catch**
  + Endpoint:[**http://localhost:3030/data/catches/:catchId**](http://localhost:3030/data/catches/:catchId)
  + Headers: X-Authorization: “….” (accessToken after login)
  + Method: PUT
  + Request body (JSON): {"angler":"…", "weight":…, "species":"…", "location":"…", "bait":"…", "captureTime":…}
* **Delete a Catch**
  + Endpoint: **http://localhost:3030/data/catches /:catchId**
  + Headers: X-Authorization: “….” (accessToken after login)
  + Method: DELETE

## Furniture

Your task is to write the functionality of app, which shows list of furniture. By logged in user there is a possibility to buy furniture and list the bought products of the logged user. Also logged user can create new products (offers).



### Home page (not logged)

When the page is loaded the app should list all the furnitures in a table:



The checkbox should be disabled. You can send GET request on the URL: **http://localhost:3030/data/furniture**

### Auth page

When "Login" is clicked, the app should redirect to "Login page". There are two possibilities:

- to register a new user, send a POST request to the URL: **http://localhost:3030/users/register**

- to login, send a POST request to the URL: **http://localhost:3030/users/login**



### Home page (logged in)

When the **"Create" button is clicked**, add a **new row to the table** for each piece of furniture with name**,** price, factor **and** img. Send POST request to: **http://localhost:3030/data/furniture**



When the **"**Buy**"** button is clicked, get all **checkboxes that are marked** and save the information for these orders on the server. Make POST request to: **http://localhost:3030/data/orders**

When the "**All orders**" button is clicked, get all bought furniture of the current user, and show their names and the total price, as shown on the picture:

****

This could happen with GET request on this URL: [**http://localhost:3030/data/orders?where=\_ownerId%3D{userId}**](http://localhost:3030/data/orders?where=_ownerId%3D%7buserId%7d)

## Submitting Your Solution

Place in a **ZIP** file the content of the given resources including your solution. Exclude the node\_modules & tests folders. Upload the archive to Judge.



Картина, която съдържа текст

Описанието е генерирано автоматично

Картина, която съдържа текст

Описанието е генерирано автоматично