Exercises: Client Side Rendering

Problems for exercises and homework for the "JavaScript Applications" course @ SoftUni Global.

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.

Each exercise must have package.json file with the following parameters:

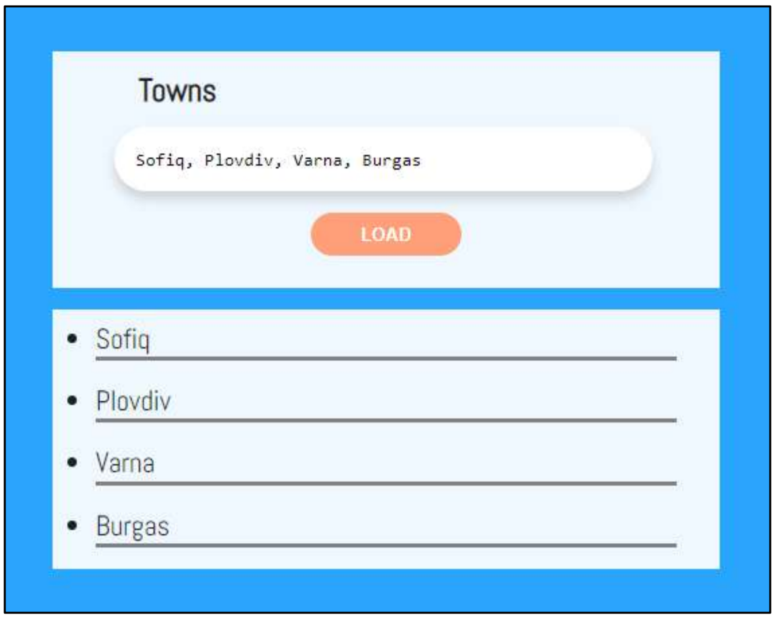
```
"test": "mocha tests",
"start": "http-server -a localhost -p 3000 -P http://localhost:3000? -c-1"
```

Look package.json in previous lecture for example.

1. List Towns

You are given an **input field** with a **button**. In the input field you should enter **elements separated** by comma and whitespace ("**,** "). Your task is to create a simple **template** that defines a **list** of towns. Each **town** comes from the **input** field. The list should be **rendered** inside the element with Id "**root**".

Screenshots



This is how the HTML should look like with the rendered template:

```
<div id="root">
    <ul>
        <li>Sofiq</li>
        <li>Plovdiv</li>
        <li>Varna</li>
        <li>Burgas</li>
    </ul>
</div>
```

2. HTTP Status Cats

We all love cats. They are also a fun way to learn all the HTTP status codes.

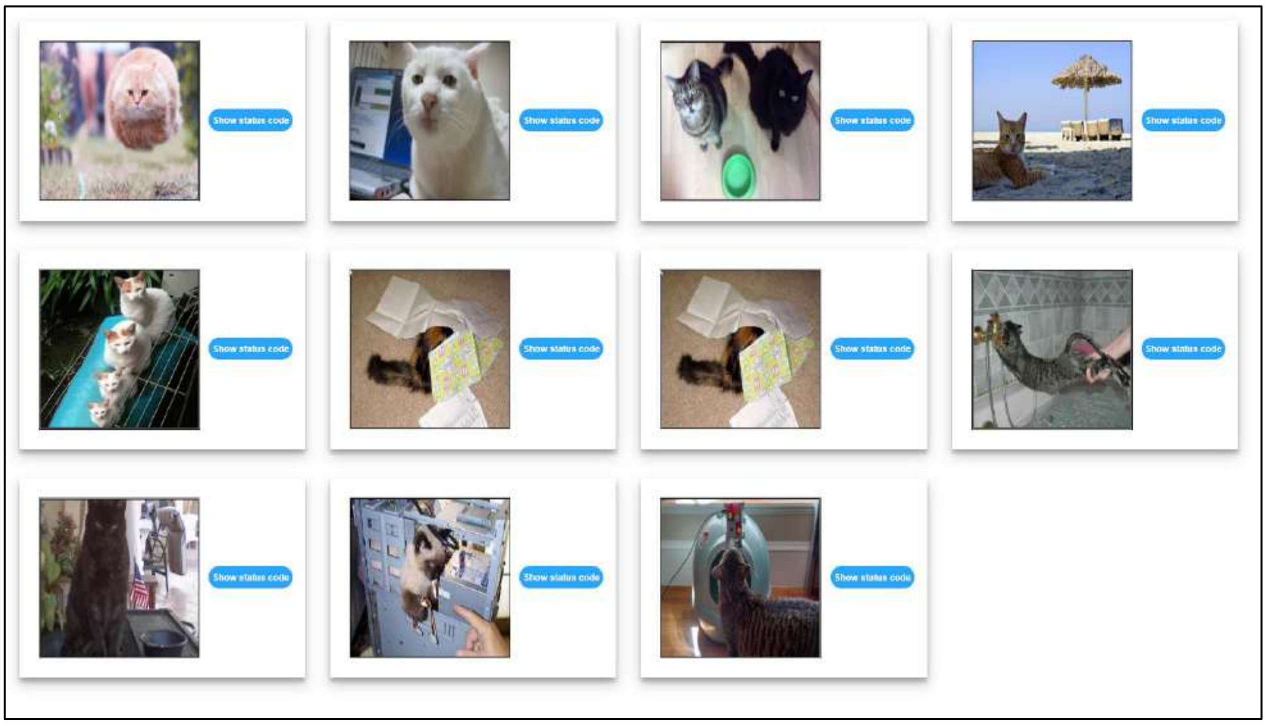
Your task is to create a **template** to represent an **HTTP cat card**. After you have **created** the template, **render** all the cats into the section with **id** **`"allCats"`**. Note that there should be a **nested** **`<ul>`** inside the section.

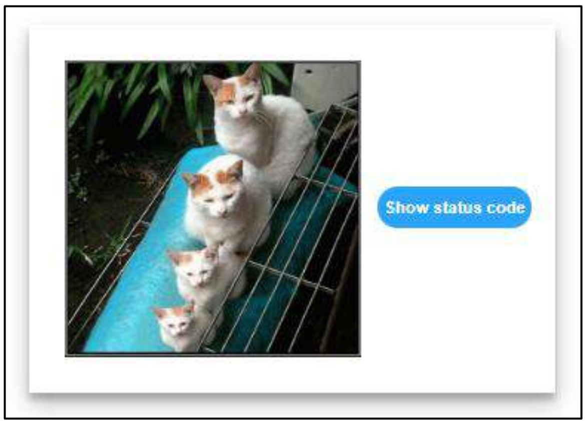
An **HTTP cat** has an **`id`**, **`statusCode`**, **`statusMessage`** and **`imageLocation`**. The cats are **seeded** using the **function** from the JS **file** named **`"catSeeder.js"`** – import this file as a module.

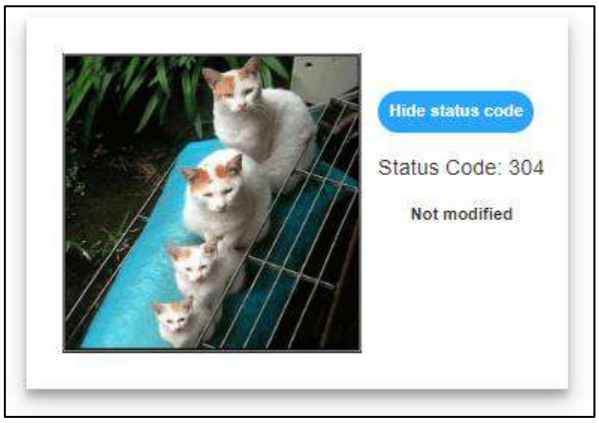
Each card block has a **button** that **reveals** its status code. You should **toggle** the button and change its text from "**`Show status code`**" to "**`Hide status code`**".

See the file **`example.html`** for an example of how the rendered HTML should look like.

Screenshots







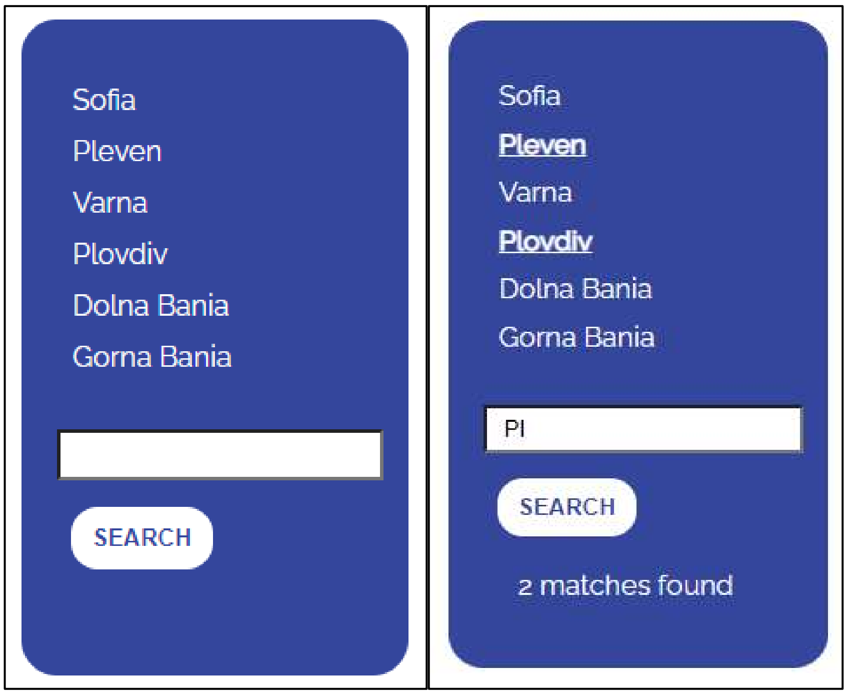
3. Search in List

An HTML page holds a **list** of towns, a **search** box and a **`[Search]`** button. Create a template for a list, containing all towns, that can be easily updated when the user performs a search. The list should be rendered inside the **`<div>`** element with **id "towns"**. Load the values from the file **towns.js**, which you can import as a module.

Implement the **`search`** function to **apply class "active"** to the items from the list which include the text from the **search** box. Also print the number of items the current search **matches** in the format **"<matches> matches found"**. The search should be **case-sensitive**.

See the file **`example.html`** for an example of how the rendered HTML should look like.

Screenshots



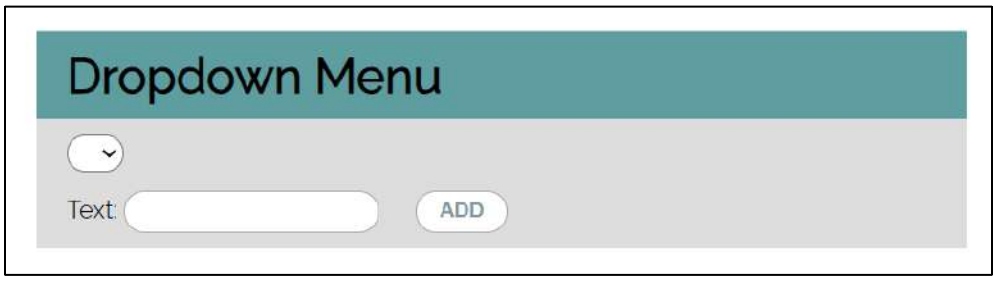
4. Fill Dropdown

Create functionality that **loads list items** from a remote service and displays them inside a **drop-down** menu. The user should also be able to **add new items** to the service by entering them in the **input** field on the page and submitting the form. Create a **template** for the **drop-down list** and the **items** inside it that can be **easily updated** with new entries.

When the program starts, the data should be automatically retrieved from the server via GET request from URL **`http://localhost:3030/jsonstore/advanced/dropdown`** and rendered as **`<option>`** items inside the **`<select>`** with **id "menu"**. Upon form submission, send a **POST** request to the same URL and if it is successful, update the list of options with the newly created item.

Each item has a property **`text`** entered by the user and **`_id`**, which is generated by the server. When creating the HTML elements, use the **`_id`** as option **value** and **`text`** as option **textContent**.

Example

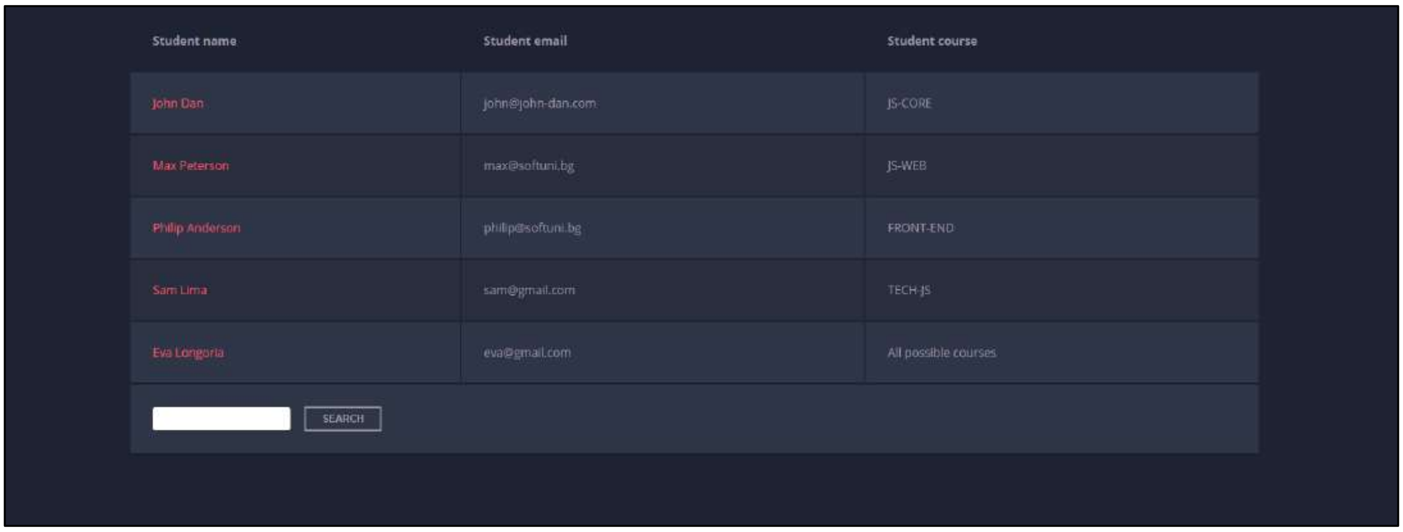


This is how the rendered HTML should look like:

```
▼ <select id="menu">
    <option value="985d9eab-ad2e-4622-a5c8-116261fb1fd2">Rome</option>
    <option value="3987279d-0ad4-4afb-8ca9-5b256ae3b298">Amsterdam</option>
    <option value="8f414b4f-ab39-4d36-bedb-2ad69da9c830">Munich</option>
  </select>
```

5. Table – Search Engine

Write a function that **searches** in a **table** by given input. Create a **template** for a **table row**, which can be **easily updated** with class values when the user performs a search. Load the data from the following URL with a GET request: **`http://localhost:3030/jsonstore/advanced/table`**



Student name	Student email	Student course
John Dan	john@john-dan.com	JS-CORE
Max Peterson	max@softuni.bg	JS-WEB
Philip Anderson	philip@softuni.bg	FRONT-END
Sam Lima	sam@gmail.com	TECH-JS
Eva Longoria	eva@gmail.com	All possible courses

When the "**Search**" **button** is **clicked**, go through all cells in the table body and check if the given input is **included** anywhere. The search should be **case-insensitive**.

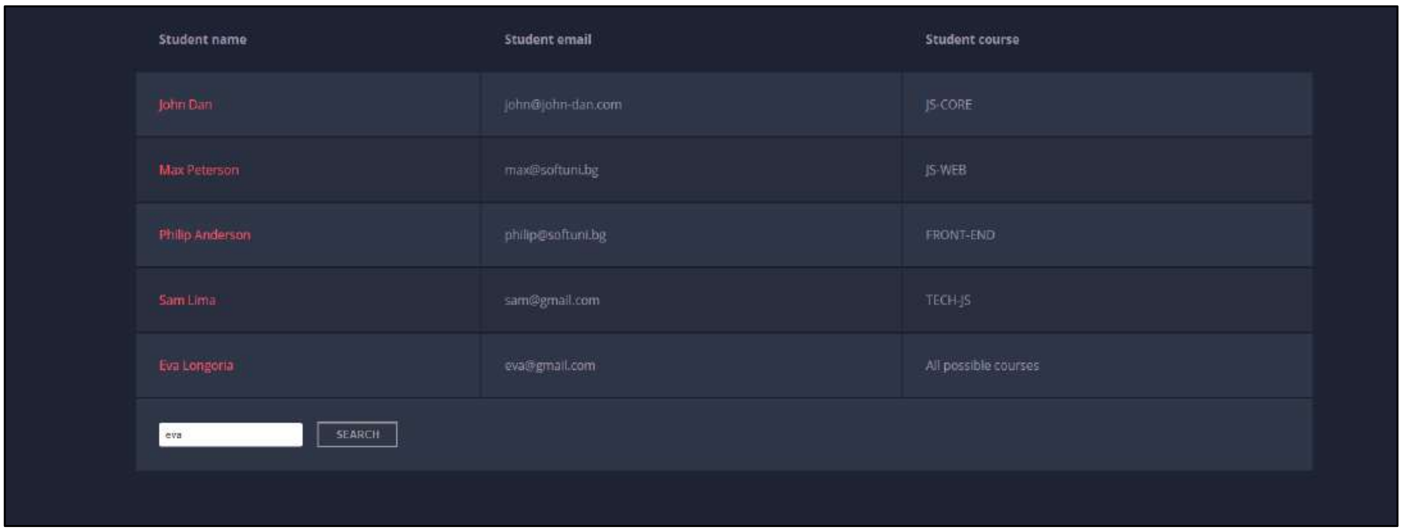
If any of the rows contains the submitted string, add a **`select`** class to that row. Note that more than one row may contain the given string. If there is no match **nothing** should be highlighted.

Note: After every search, **clear the input field** and **remove all already selected classes** (if any) from the previous search, in order for the **new search** to contain only the **new result**.

See the file **`example.html`** for an example of how the rendered HTML should look like.

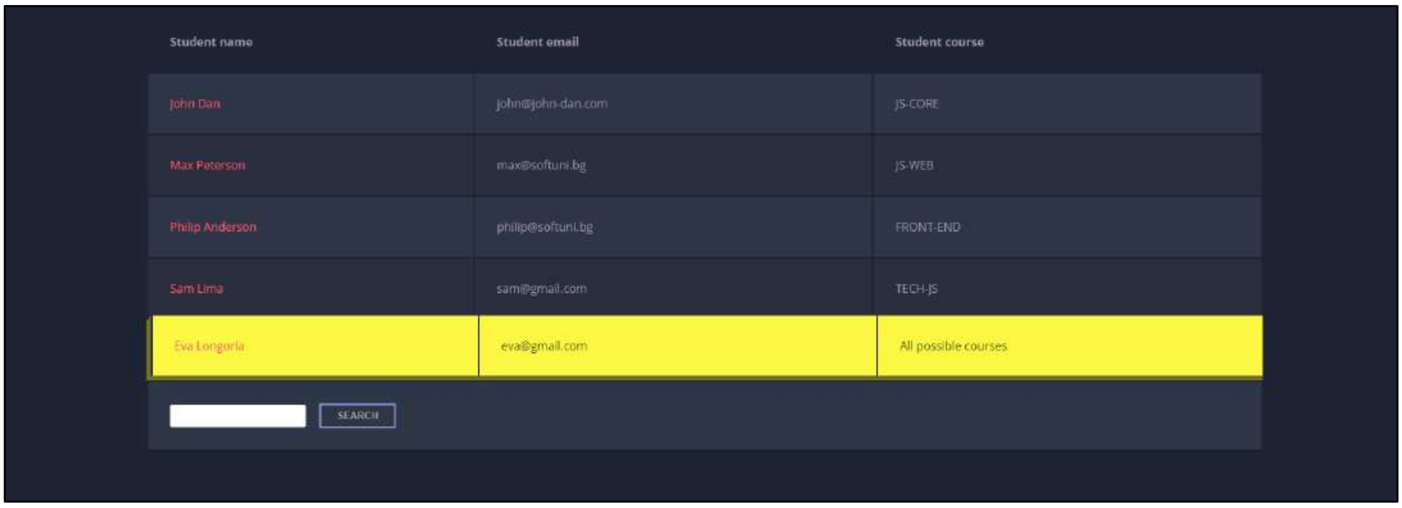
Example

For instance, if we try to find **eva:**



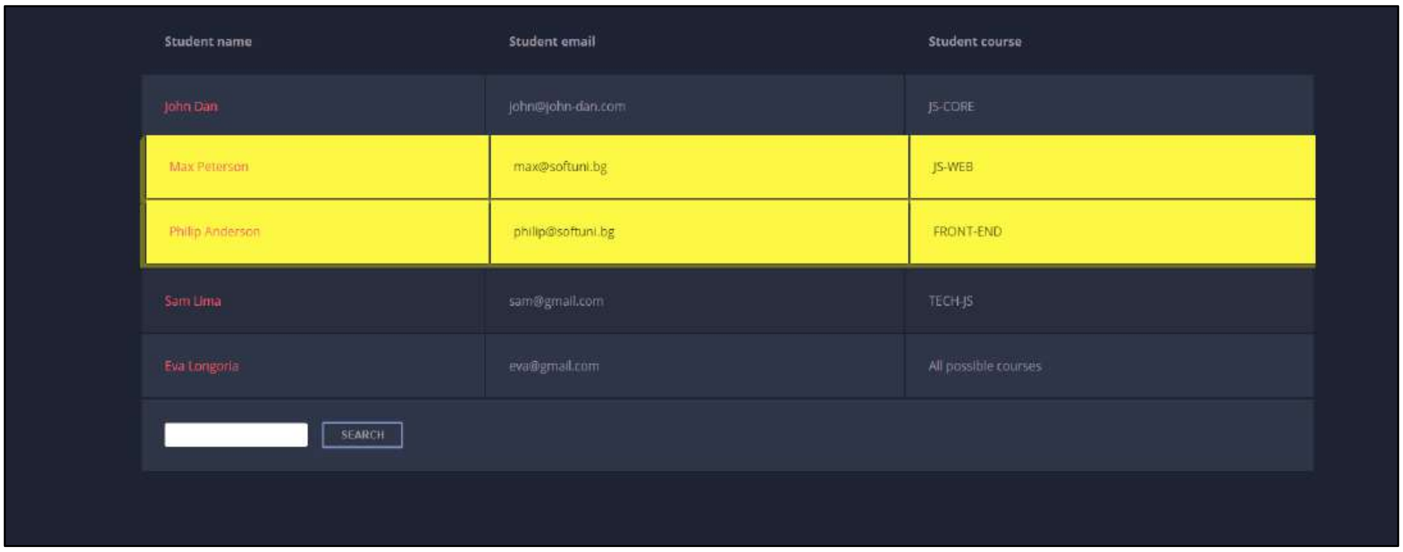
Student name	Student email	Student course
John Dan	john@john-dan.com	JS-CORE
Max Peterson	max@softuni.bg	JS-WEB
Philip Anderson	philip@softuni.bg	FRONT-END
Sam Lima	sam@gmail.com	TECH-JS
Eva Longoria	eva@gmail.com	All possible courses

The result should be:



Student name	Student email	Student course
John Dan	john@john-dan.com	JS-CORE
Max Peterson	max@softuni.bg	JS-WEB
Philip Anderson	philip@softuni.bg	FRONT-END
Sam Lima	sam@gmail.com	TECH-JS
Eva Longoria	eva@gmail.com	All possible courses

If we try to find all students who have email addresses in **softuni** domain, the expected result should be:



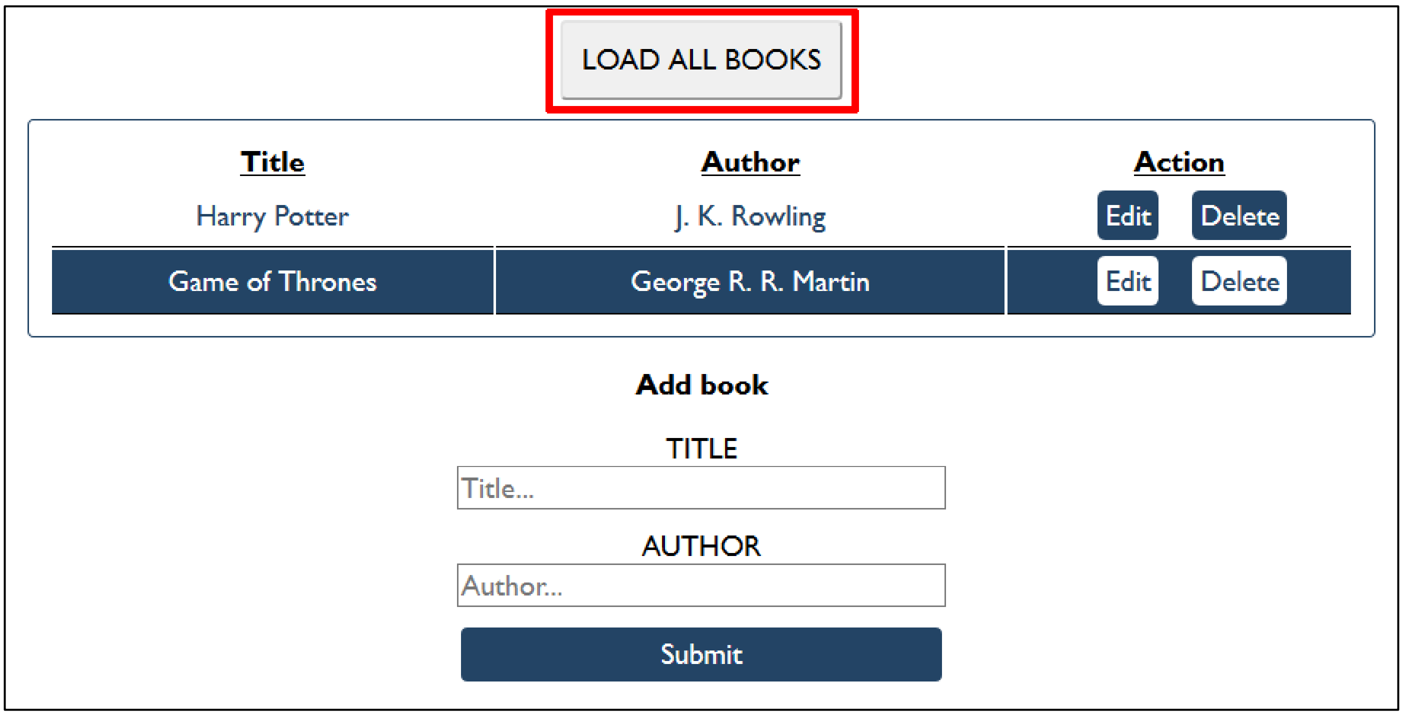
Student name	Student email	Student course
John Dan	john@john-dan.com	JS-CORE
Max Peterson	max@softuni.bg	JS-WEB
Philip Anderson	philip@softuni.bg	FRONT-END
Sam Lima	sam@gmail.com	TECH-JS
Eva Longoria	eva@gmail.com	All possible courses

6. Book Library

Create **templates** for **all items** on the page, as you see fit. See the file **`example.html`** for an example of how the rendered HTML may look like. You are free to add attributes that would help you implement the required functionality.

Get All Books

First task is to "**GET**" all books when the button "Load All Books" is clicked. To consume the data from the API, send a request to the **following URL**: **http://localhost:3030/jsonstore/collections/books**



Title	Author	Action
Harry Potter	J. K. Rowling	Edit Delete
Game of Thrones	George R. R. Martin	Edit Delete

Add book

TITLE

Title...

AUTHOR

Author...

Submit

Create Book

Initially, the form with **id "add-form"** should be displayed. Write functionality to create a new book, when the submit button is clicked. Before sending the request be sure the fields are not empty (make validation of the input). To **create** a book, you must send a "**POST**" request and the JSON body should be in the **following** format:

```
{
  "author": "New Author",
  "title": "New Title"
}
```

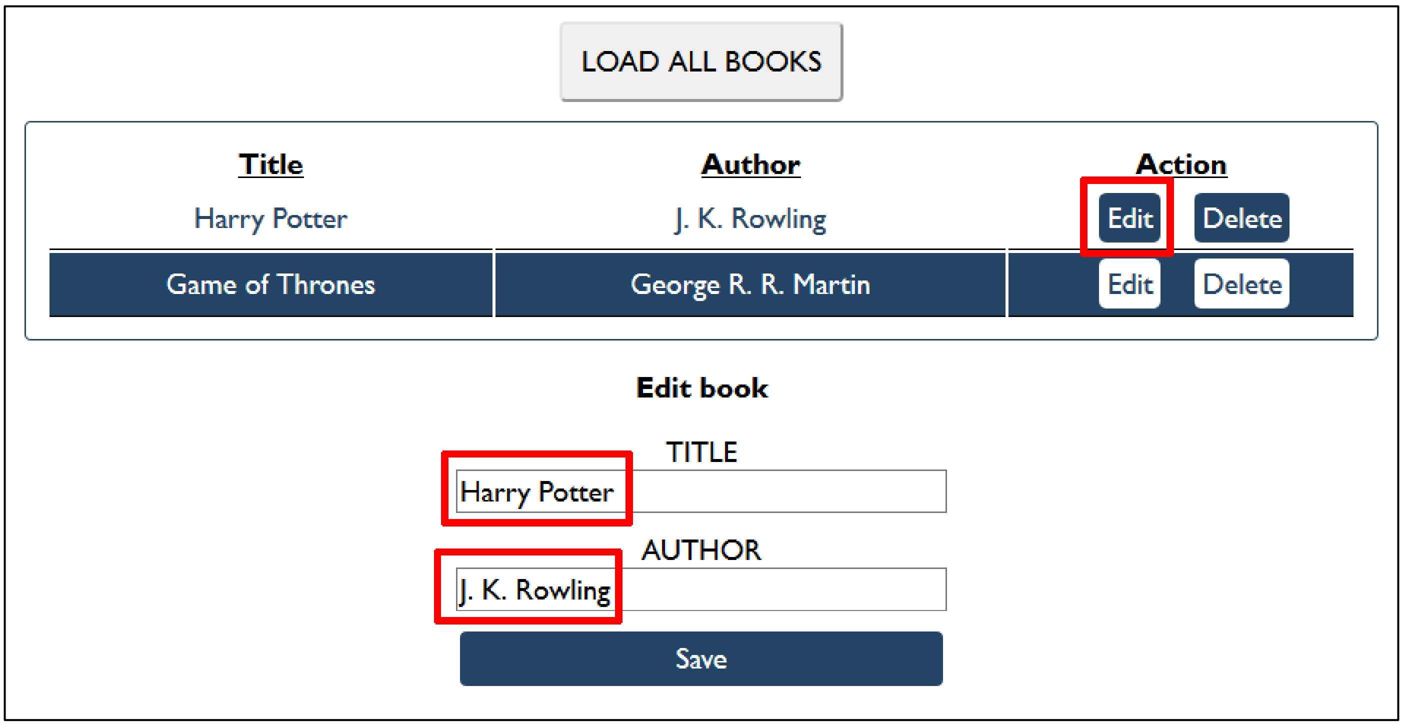
Get Book

Send a "GET" request to the following url:

http://localhost:3030/jsonstore/collections/books/:id

Update Book

By clicking the edit button of a book, display the form with **id "edit-form"** and populate its fields with the information from the selected book:



Title	Author	Action
Harry Potter	J. K. Rowling	Edit Delete
Game of Thrones	George R. R. Martin	Edit Delete

Edit book

TITLE: Harry Potter

AUTHOR: J. K. Rowling

The HTTP command "**PUT**" **modifies** an existing HTTP **resource**. The URL is:

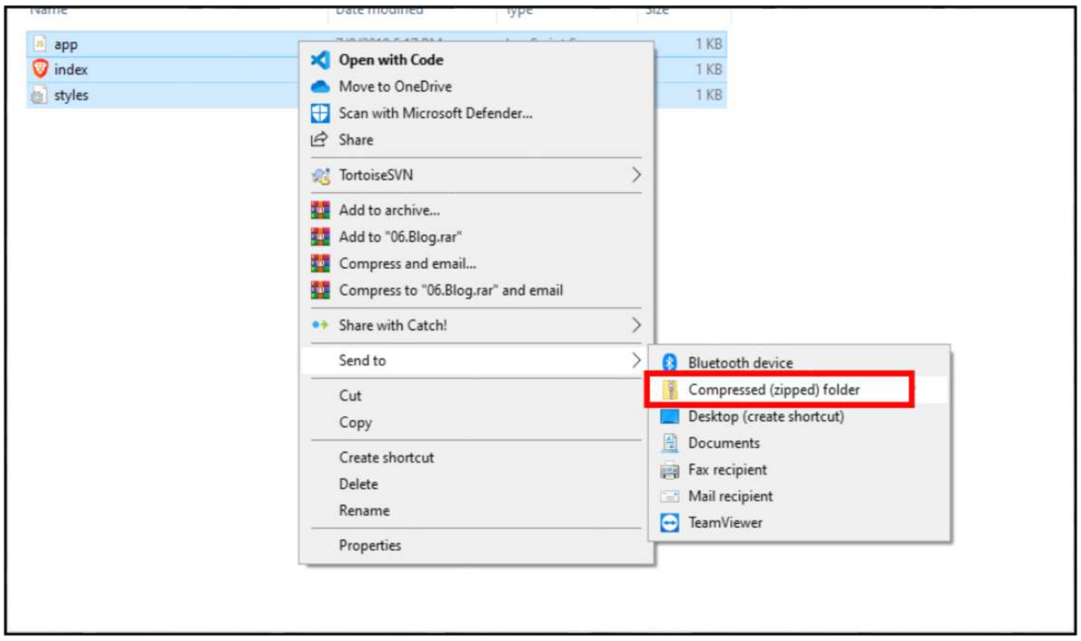
http://localhost:3030/jsonstore/collections/books/:id

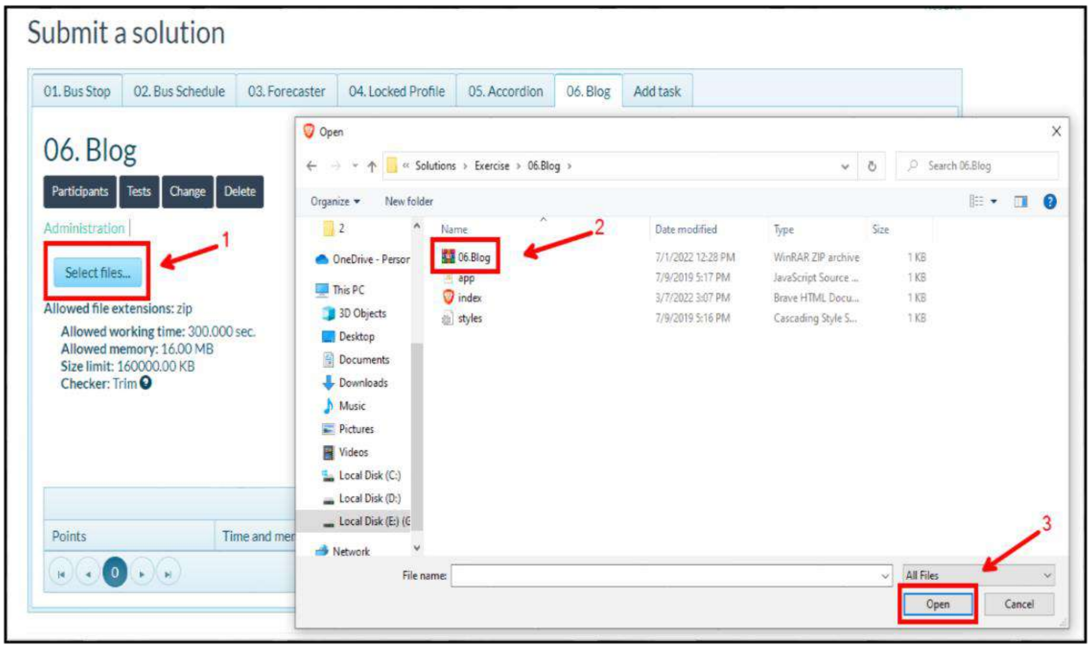
The JSON body should be in the **following** format:

```
{
  "author": "Changed Author",
  "title": "Changed Title"
}
```

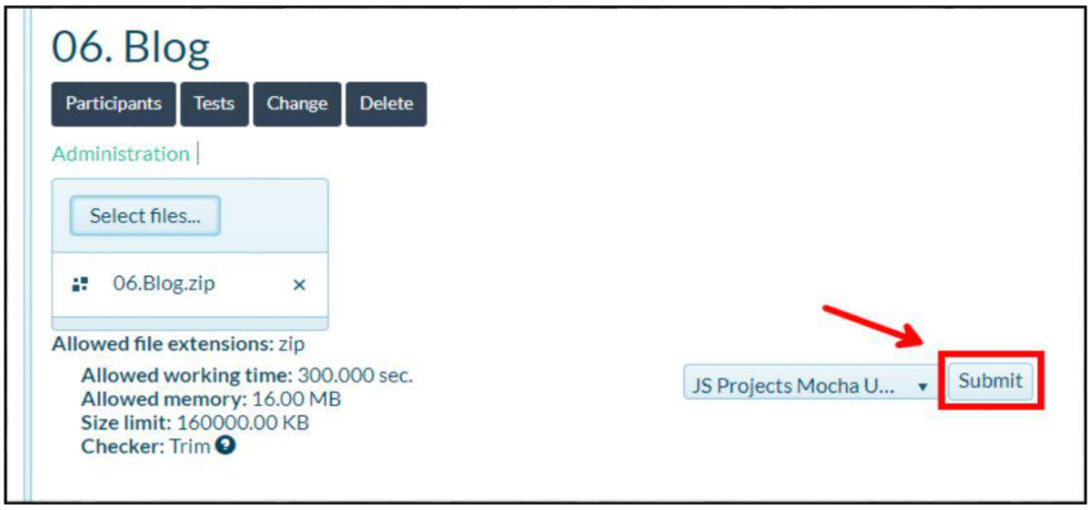
Submitting Your Solution

Place in a **ZIP** file the content of the given resources including your solution. Exclude the **node_modules** folder if there is one. Upload the archive to Judge.





Submit a solution
01. Bus Stop
02. Bus Schedule
03. Forecaster
04. Locked Profile
05. Accordion
06. Blog
Add task
06. Blog
Participants
Tests
Change
Delete
Administration
Select files...
1
Allowed file extensions: zip
Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim
Points
Time and mem
Open
Solutions › Exercise › 06.Blog
Search 06.Blog
Organize
New folder
Name
Date modified
Type
Size
06.Blog
7/1/2022 12:28 PM
WinRAR ZIP archive
1 KB
app
7/9/2019 5:17 PM
JavaScript Source ...
1 KB
index
3/7/2022 3:07 PM
Brave HTML Docu...
1 KB
styles
7/9/2019 5:16 PM
Cascading Style S...
1 KB
2
OneDrive - Person
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
Local Disk (D:)
Local Disk (E:)
Network
File name:
All Files
3
Open
Cancel



06. Blog
Participants
Tests
Change
Delete
Administration
Select files...
06.Blog.zip
Allowed file extensions: zip
Allowed working time: 300.000 sec.
Allowed memory: 16.00 MB
Size limit: 160000.00 KB
Checker: Trim
JS Projects Mocha U...
Submit