# Exercises: Client Side Rendering

Problems for exercises and homework for the ["JavaScript Applications" course @ SoftUni](https://softuni.bg/trainings/3589/js-applications-february-2022).

**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).

## 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:



## 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**,** statusMessageandimageLocation. 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 "Showstatuscode" to "Hidestatuscode".

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

### Screenshots





## 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-insensitive**.

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

### Screenshots



## 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:  


## 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**



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:**



The result should be:



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

