Exercise: HTTP and REST

Problems for exercises and homework for the ["JS Front-End" course @ SoftUni](https://softuni.org)

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.

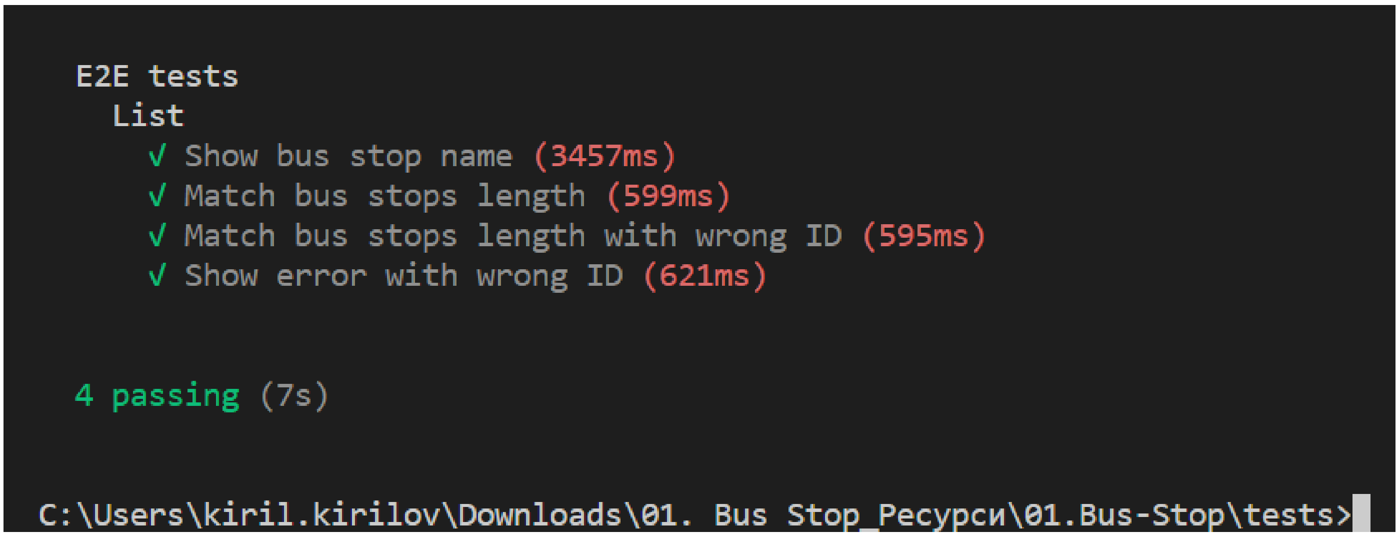
Requirements

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:



```
  E2E tests
    List
      √ Show bus stop name (3457ms)
      √ Match bus stops length (599ms)
      √ Match bus stops length with wrong ID (595ms)
      √ Show error with wrong ID (621ms)


  4 passing (7s)


C:\Users\kiril.kirilov\Downloads\01. Bus Stop_Ресурси\01.Bus-Stop\tests>
```

1. Bus Stop

Write a JS program that displays arrival times for all buses by a given bus stop ID when a button is clicked. Use the skeleton from the provided resources.

When the button with ID **'submit'** is clicked, the name of the bus stop appears and the list bellow gets filled with all the buses that are expected and their time of arrival. Take the **value** of the input field with id **'stopId'**. Submit a **GET** request to **http://localhost:3030/jsonstore/bus/businfo/:busId** (replace the highlighted part with the correct value) and parse the response. You will receive a JSON object in the format:

```
stopId: {
  name: stopName,
  buses: { busId: time, … }
}
```

Place the name property as text inside the div with ID **'stopName'** and each bus as a list item with text:

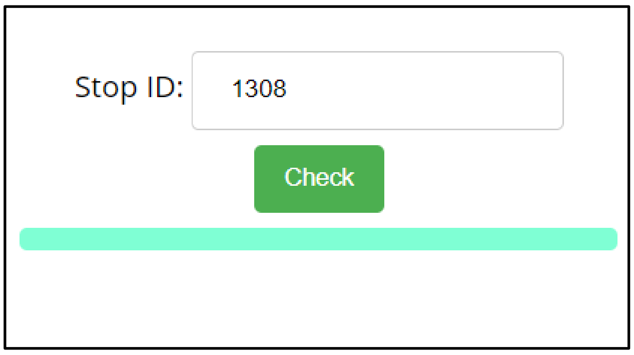
`"Bus {busId} arrives in {time} minutes"`

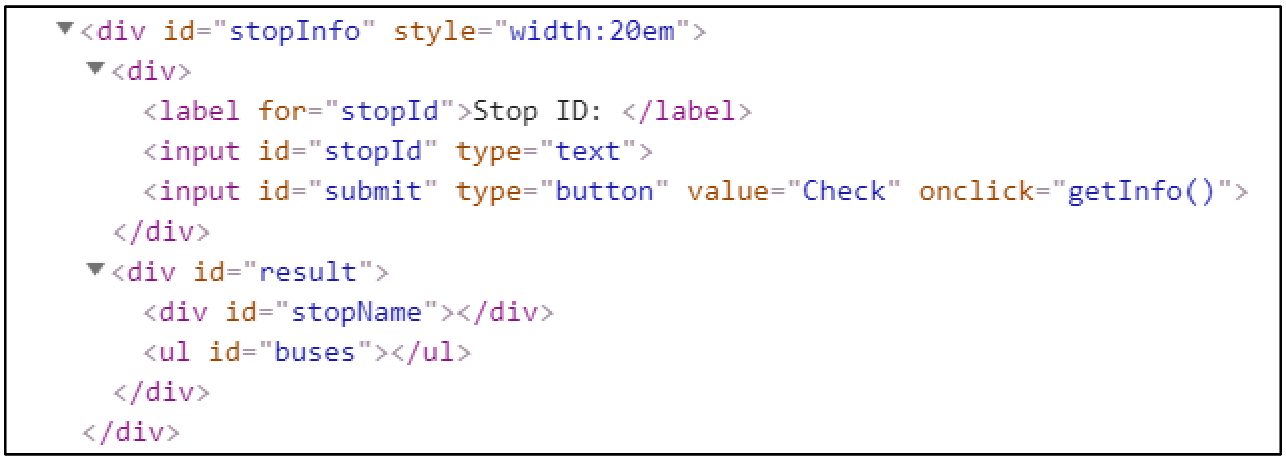
Replace all highlighted parts with the relevant value from the response. If the request is not successful, or the information is not in the expected format, display **"Error"** as **`stopName`** and nothing in the list. The list should be cleared before every request is sent.

Note: The service will respond with valid data to IDs 1287, 1308, 1327 and 2334.

See examples on the next page.

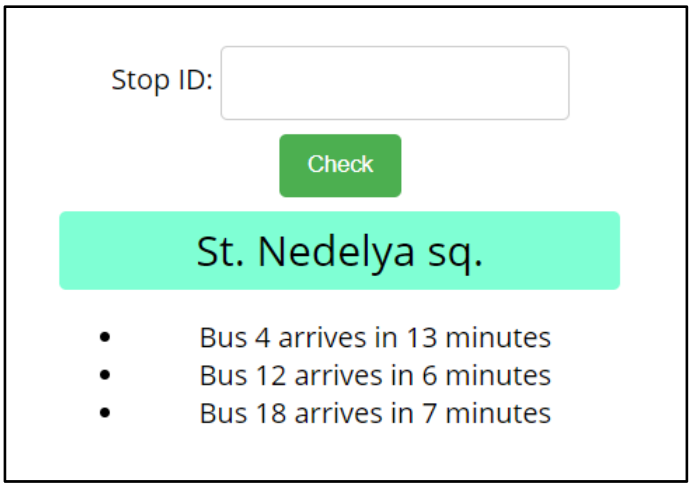
Examples

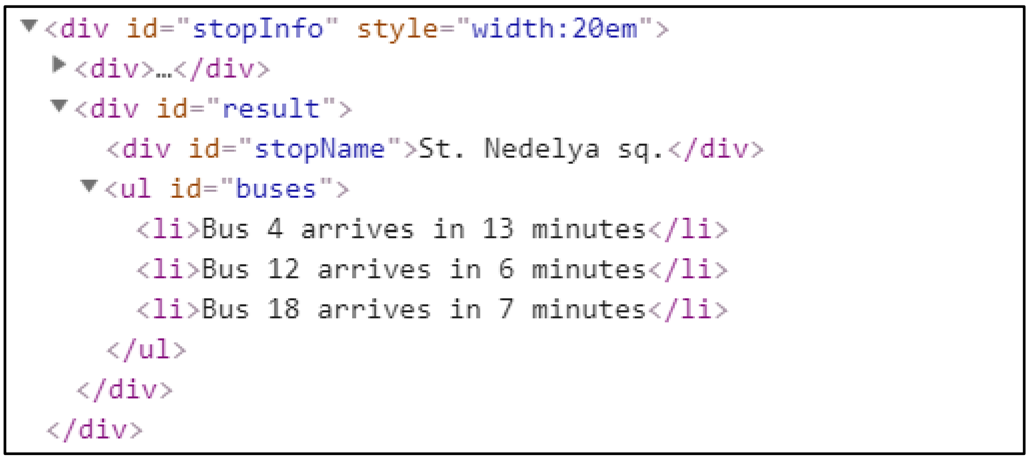




```
▼<div id="stopInfo" style="width:20em">
  ▼<div>
      <label for="stopId">Stop ID: </label>
      <input id="stopId" type="text">
      <input id="submit" type="button" value="Check" onclick="getInfo()">
    </div>
  ▼<div id="result">
      <div id="stopName"></div>
      <ul id="buses"></ul>
    </div>
  </div>
```

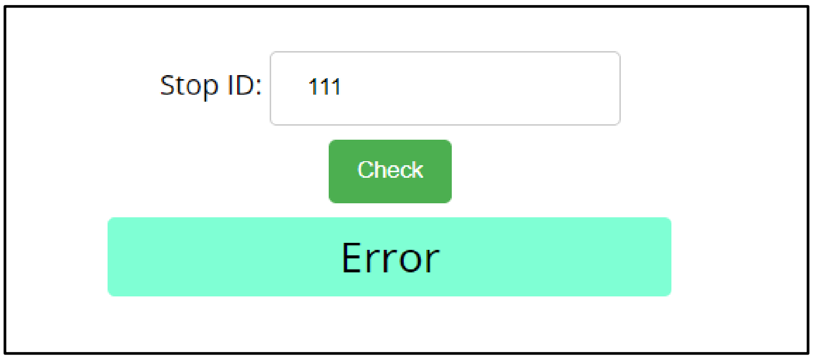
When the button is clicked, the results are displayed in the corresponding elements:

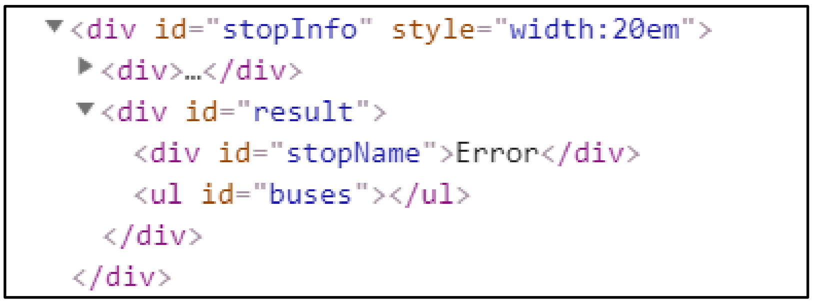

Stop ID:
Check
St. Nedelya sq.
Bus 4 arrives in 13 minutes
Bus 12 arrives in 6 minutes
Bus 18 arrives in 7 minutes



```
▼<div id="stopInfo" style="width:20em">
  ▶<div>…</div>
  ▼<div id="result">
      <div id="stopName">St. Nedelya sq.</div>
    ▼<ul id="buses">
        <li>Bus 4 arrives in 13 minutes</li>
        <li>Bus 12 arrives in 6 minutes</li>
        <li>Bus 18 arrives in 7 minutes</li>
      </ul>
    </div>
  </div>
```

If an error occurs, the stop name changes to Error:





```
▼<div id="stopInfo" style="width:20em">
  ▶<div>…</div>
  ▼<div id="result">
      <div id="stopName">Error</div>
      <ul id="buses"></ul>
    </div>
  </div>
```

2. Bus Schedule

Write a JS program that tracks the progress of a bus on it's route and announces it inside an info box. The program should display which is the upcoming stop and once the bus arrives, to request from the server the name of the next one. Use the skeleton from the provided resources.

The bus has two states – **moving** and **stopped**. When it is **stopped**, only the button "**Depart**" is **enabled**, while the info box shows the name of the **current** stop. When it is **moving**, only the button "**Arrive**" is **enabled**, while the info box shows the name of the **upcoming** stop. Initially, the info box shows "**Not Connected**" and the "**Arrive**" button is **disabled**. The ID of the first stop is "**depot**".

When the "**Depart**" button is clicked, make a **GET** request to the server with the ID of the current stop to address `http://localhost:3030/jsonstore/bus/schedule/:id` (replace the highlighted part with the relevant value). As a response, you will receive a JSON object in the following format:

```
stopId {
  name: stopName,
  next: nextStopId
}
```

Update the info box with the information from the response, disable the "Depart" button and enable the "Arrive" button. The info box text should look like this (replace the highlighted part with the relevant value):

```
Next stop {stopName}
```

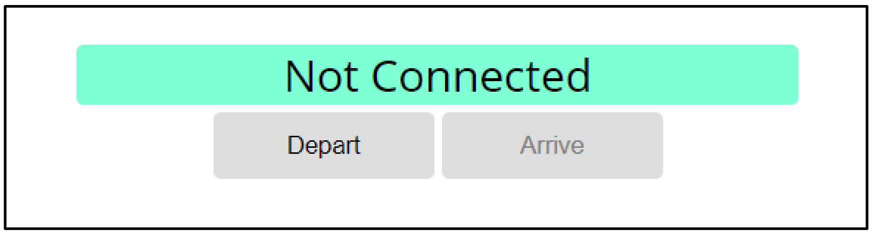
When the "**Arrive**" button is clicked, update the text, disable the "Arrive" button and enable the "Depart" button. The info box text should look like this (replace the highlighted part with the relevant value):

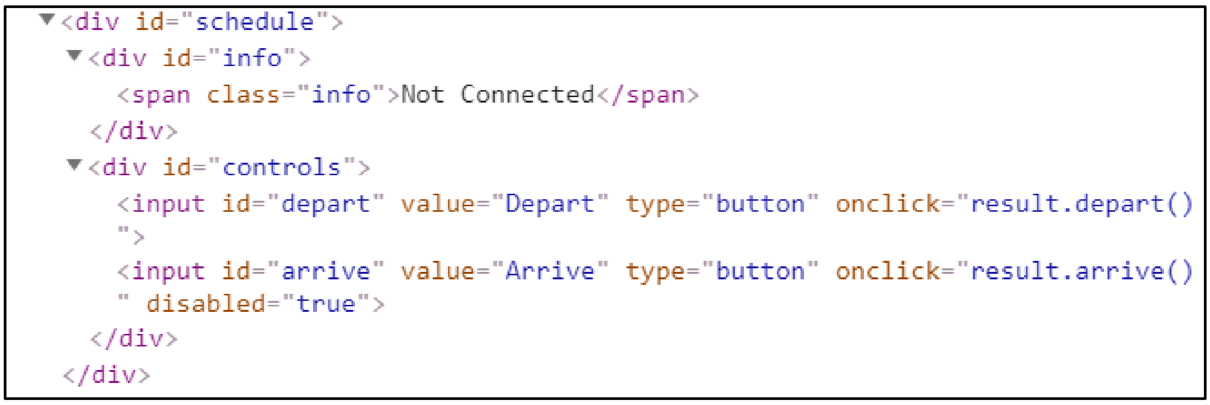
```
Arriving at {stopName}
```

Clicking the buttons successfully will cycle through the entire schedule. If invalid data is received, show "**Error**" inside the info box and **disable** both buttons.

Examples

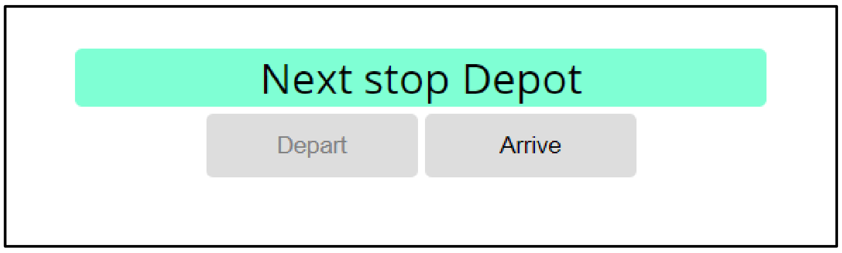
Initially, the info box shows "Not Connected" and the arrive button is disabled.

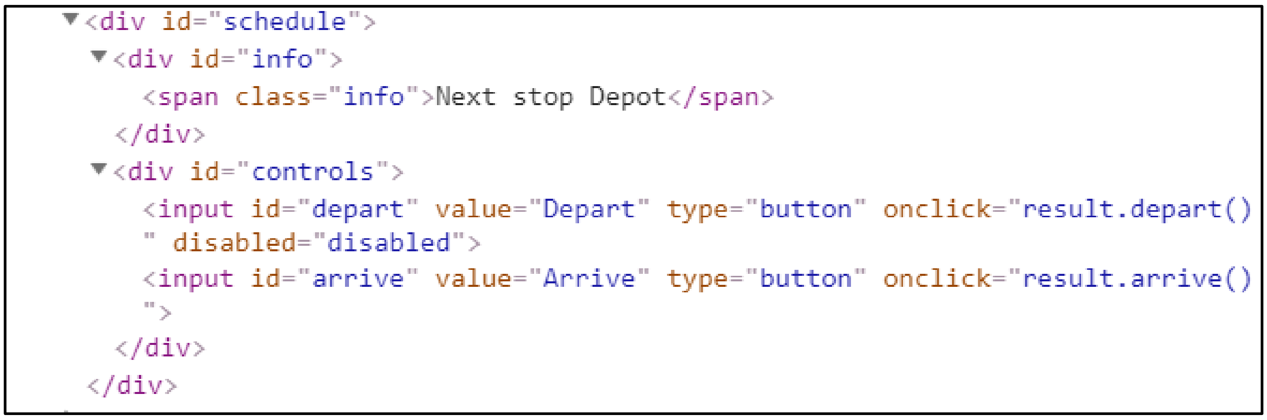




```
▼<div id="schedule">
  ▼<div id="info">
      <span class="info">Not Connected</span>
    </div>
  ▼<div id="controls">
      <input id="depart" value="Depart" type="button" onclick="result.depart()
      ">
      <input id="arrive" value="Arrive" type="button" onclick="result.arrive()
      " disabled="true">
    </div>
  </div>
```

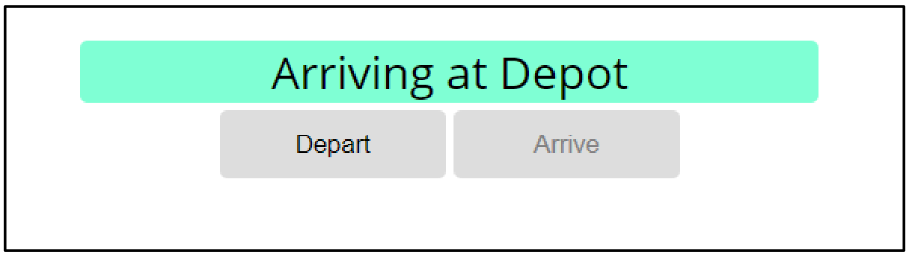
When Depart is clicked, a request is made with the first ID. The info box is updated with the new information and the buttons are changed:

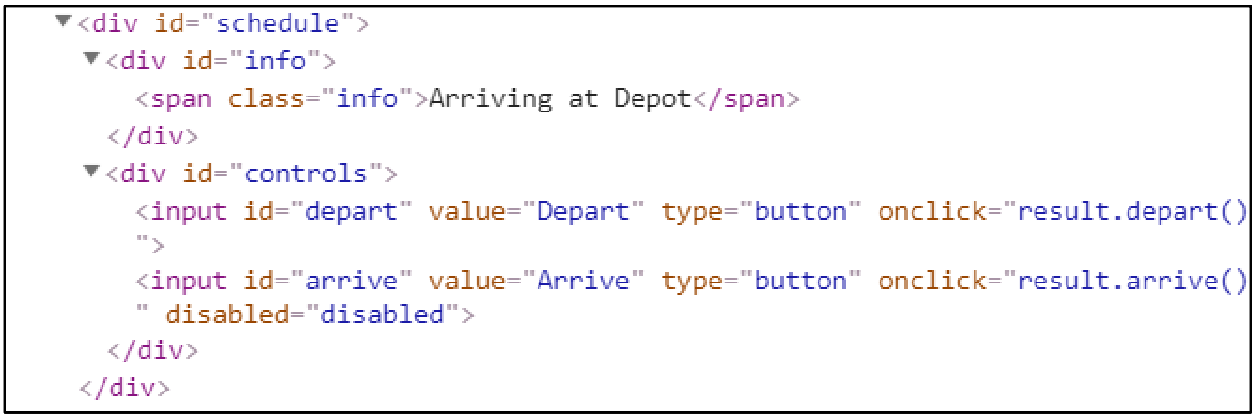




```
▼<div id="schedule">
  ▼<div id="info">
      <span class="info">Next stop Depot</span>
    </div>
  ▼<div id="controls">
      <input id="depart" value="Depart" type="button" onclick="result.depart()
      " disabled="disabled">
      <input id="arrive" value="Arrive" type="button" onclick="result.arrive()
      ">
    </div>
  </div>
```

Clicking Arrive, changes the info box and swaps the buttons. This allows Depart to be clicked again, which makes a new request and updates the information:





```
▼<div id="schedule">
  ▼<div id="info">
     <span class="info">Arriving at Depot</span>
   </div>
  ▼<div id="controls">
     <input id="depart" value="Depart" type="button" onclick="result.depart()
     ">
     <input id="arrive" value="Arrive" type="button" onclick="result.arrive()
     " disabled="disabled">
   </div>
 </div>
```

3. 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).

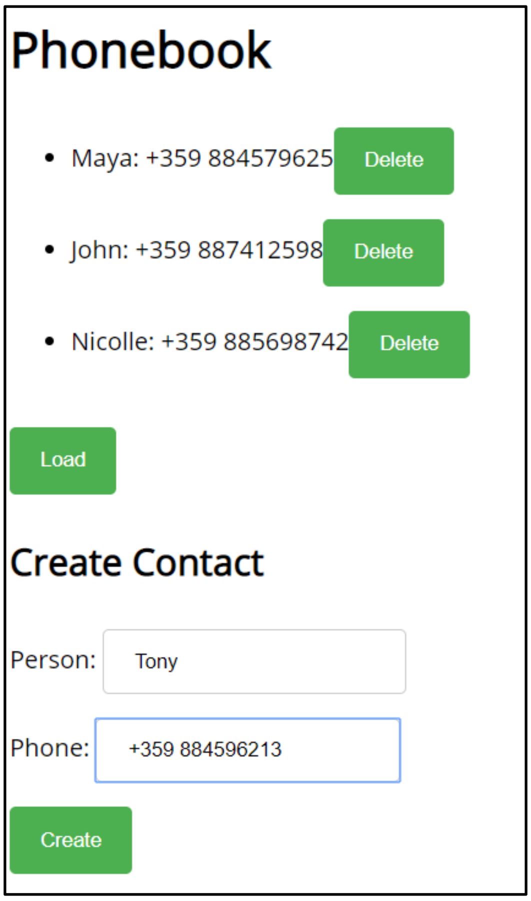
The data sent on a POST **request should be a valid JSON object, containing properties** person **and** phone. **Example format:**

```
{
  "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



Phonebook

- Maya: +359 884579625 Delete
- John: +359 887412598 Delete
- Nicolle: +359 885698742 Delete

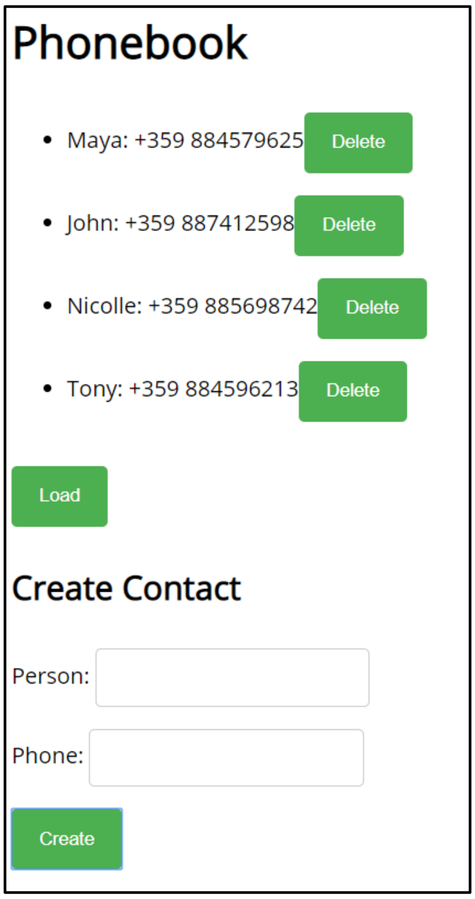
Load

Create Contact

Person: Tony

Phone: +359 884596213

Create



Phonebook

- Maya: +359 884579625 Delete
- John: +359 887412598 Delete
- Nicolle: +359 885698742 Delete
- Tony: +359 884596213 Delete

Load

Create Contact

Person:

Phone:

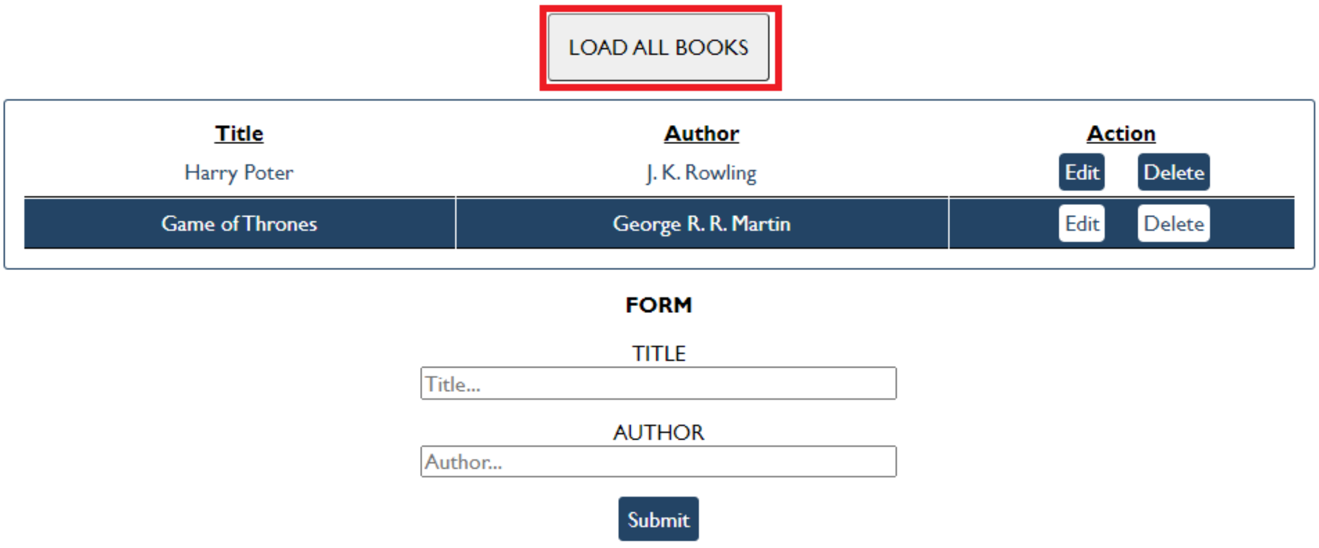
Create

4. Book Library

First task is to "**GET**" all books. To consume the request with **POSTMAN** your **url** should be the **following**: **http://localhost:3030/jsonstore/collections/books**

Using the provided skeleton, write the missing functionalities.

Load all books by clicking the button "LOAD ALL BOOKS"



LOAD ALL BOOKS

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

FORM

TITLE

Title...

AUTHOR

Author...

Submit

Get Book

This functionality is not needed in this task, but you can try it with postman by sending request to "GET" the Book with id:" d953e5fb-a585-4d6b-92d3-ee90697398a0". Send a GET request to this URL:

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

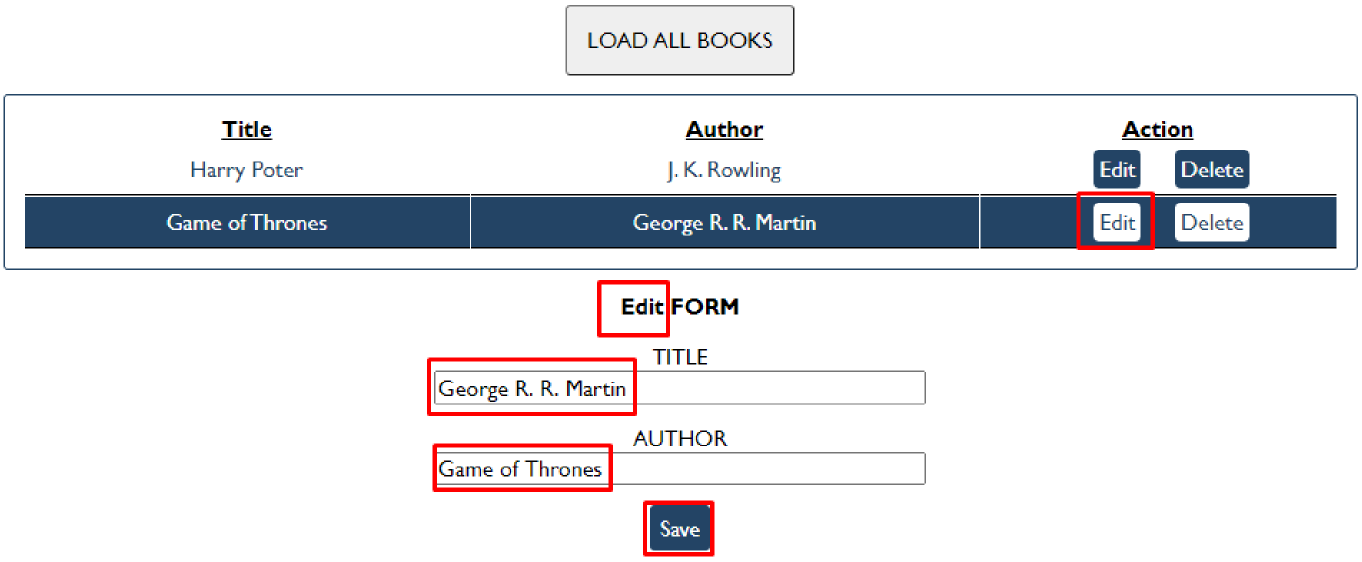
Create Book

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 have to send a "**POST**" request and the JSON body should be in the **following** format:

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

Update Book

By clicking the edit button of a book, change the form like this:



LOAD ALL BOOKS

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

Edit FORM

TITLE

George R. R. Martin

AUTHOR

Game of Thrones

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"
}
```

Delete Book

By clicking the delete button you have to delete the book, without any confirmation. To delete a book use "**DELETE**" command and send **REQUEST**: **http://localhost:3030/jsonstore/collections/books/:id**