Exercise: Asynchronous Programming

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.

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:

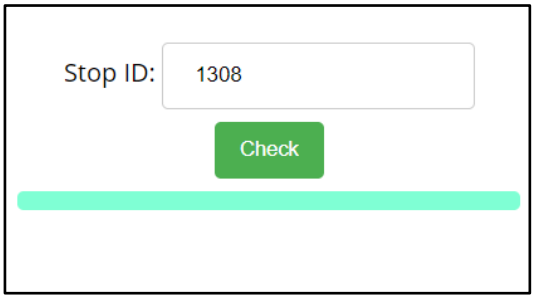
```
"Bus {busId} arrives in {time} minutes"
```

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.

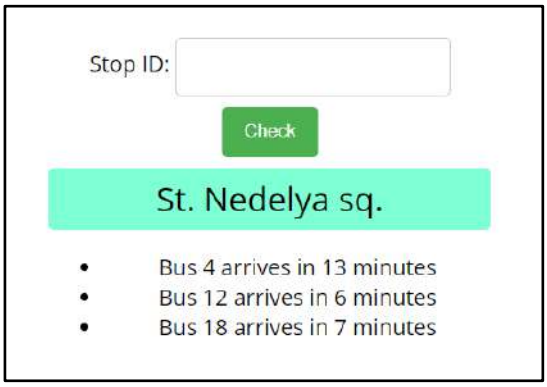
See examples on the next page.

Examples



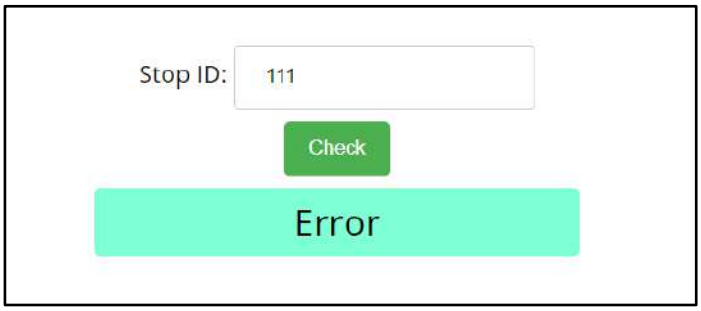
```
<div id="stopInfo" style="width:20em">
  <div>
    <label for="stopId">Stop ID: </label>
    <input id="stopId" type="text">
    <input id="submit" type="button" value="Check" onclick="getInfo()">
  </div>
  <div id="result">
    <div id="stopName"></div>
    <ul id="buses"></ul>
  </div>
</div>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

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




```
<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}`

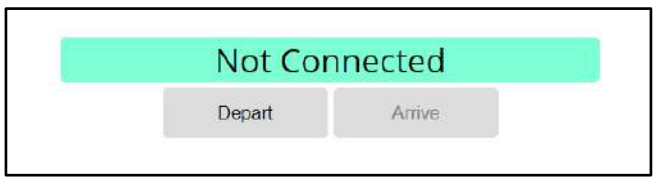
When the "**Arrive**" button is clicked, update the text, disable the "Arrive" button and enable the "Depart" button. The info box text should look like this (replace the highlighted part with the relevant value):

`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. Forecaster

Write a program that **requests** a weather report **from a server** and **displays** it to the user.

Use the skeleton from the provided resources.

When the user writes the name of a location and clicks "**`Get Weather`**", make a **`GET`** request to the server at address **`http://localhost:3030/jsonstore/forecaster/locations`**. The response will be an array of objects, with the following structure:

```
{
  name: locationName,
  code: locationCode
}
```

Find the object, corresponding to the name that the user submitted in the input field with ID "**`location`**" and use its **code** value to make **two more GET requests**:

- For current conditions, make a request to:

`http://localhost:3030/jsonstore/forecaster/today/:code`

The response from the server will be an object with the following structure:

```
{
  name: locationName,
  forecast: { low: temp,
              high: temp,
              condition: condition }
}
```

- For a 3-day forecast, make a request to:

http://localhost:3030/jsonstore/forecaster/upcoming/:code

The response from the server will be an object with the following structure:

```
{
  name: locationName,
  forecast: [{ low: temp,
               high: temp,
               condition: condition }, … ]
}
```

Use the information from these two objects to compose a forecast in HTML and insert it inside the page. Note that the ** ** with ID "**forecast**" must be set to **visible**. See the examples for details.

If an **error** occurs (the server doesn't respond or the location name cannot be found) or the data is not in the correct format, display "**Error**" in the **forecast section**.

Use the following codes for weather symbols:

- Sunny **☀** // ☀
- Partly sunny **⛅** // ⛅
- Overcast **☁** // ☁
- Rain **☔** // ☂
- Degrees **°** // °

Examples

When the app starts, the **forecast div** is **hidden**. When the user **enters a name** and **clicks** on the button **Get Weather**, the requests being.


```
▶<div id="request">…</div>
▼<div id="forecast" style="display: block;">
  ▼<div id="current">
     <div class="label">Current conditions</div>
    ▼<div class="forecasts">
       <span class="condition symbol">☀</span>
      ▼<span class="condition">
         <span class="forecast-data">New York, USA</span>
         <span class="forecast-data">8°/19°</span>
         <span class="forecast-data">Sunny</span>
       </span>
     </div>
   </div>
  ▼<div id="upcoming">
     <div class="label">Three-day forecast</div>
    ▼<div class="forecast-info">
      ▼<span class="upcoming">
         <span class="symbol">⛅</span>
         <span class="forecast-data">6°/17°</span>
         <span class="forecast-data">Partly sunny</span>
       </span>
      ▶<span class="upcoming">…</span>
      ▶<span class="upcoming">…</span>
     </div>
   </div>
 </div>
</div>
```

4. 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`

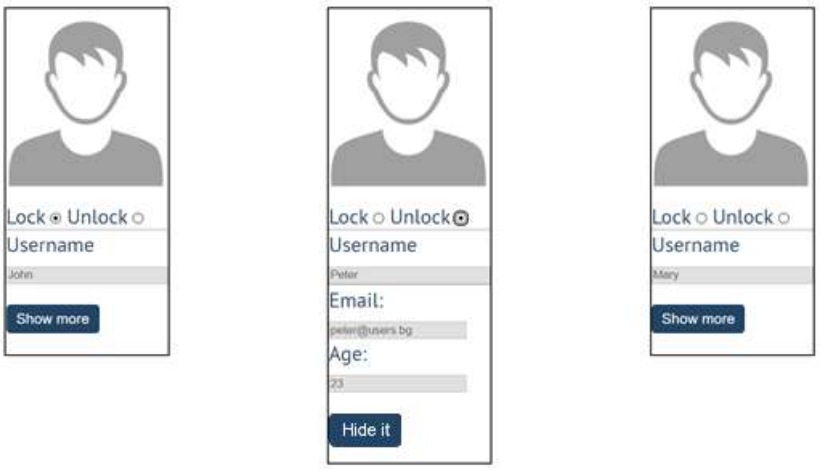
The response will be an object with the information for all users. Create a profile card for every user and display it on the web page. Every item should have the following structure:

```html
<main id="main">

    <div class="profile">
        <img src="./iconProfile2.png" class="userIcon" />
        <label>Lock</label>
        <input type="radio" name="user1Locked" value="lock" checked>
        <label>Unlock</label>
        <input type="radio" name="user1Locked" value="unlock"><br>
        <hr>
        <label>Username</label>
        <input type="text" name="user1Username" value="John" disabled readonly />
        <div id="user1HiddenFields">
            <hr>
            <label>Email:</label>
            <input type="email" name="user1Email" value="john@users.bg" disabled readonly />
            <label>Age:</label>
            <input type="email" name="user1Age" value="31" disabled readonly />
        </div>
        <button>Show more</button>
    </div>

</main>
```

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.

5. 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

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

```
<section id="main">
    <div class="accordion">
        <div class="head">
            <span>Scalable Vector Graphics</span>
            <button class="button" id="ee9823ab-c3e8-4a14-b998-8c22ec246bd3">More</button>
        </div>
        <div class="extra">
            <p>Scalable Vector Graphics (SVG) is an Extensible Markup Language (XML)-based vector image format for
                two-dimensional graphics with support for interactivity and animation. The SVG specification is an
                open standard developed by the World Wide Web Consortium (W3C) since 1999.</p>
        </div>
    </div>
</section>
```

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

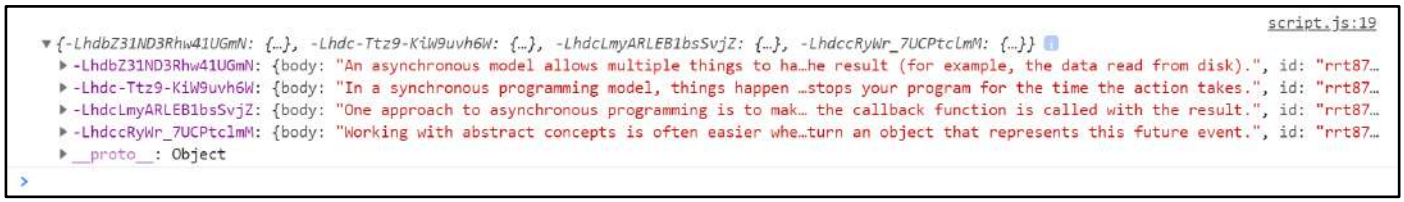
6. Blog

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`**

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**".


```
<select id="posts">
  <option value="-LhdbZ31ND3Rhw41UGmN">ASYNCHRONOUS PROGRAMMING</option>
  <option value="-Lhdc-Ttz9-KiW9uvh6W">SYNCHRONOUS PROGRAMMING</option>
  <option value="-LhdcLmyARLEB1bsSvjZ">CALLBACKS</option>
  <option value="-LhdccRyWr_7UCPtclmM">PROMISES</option>
</select>
```

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

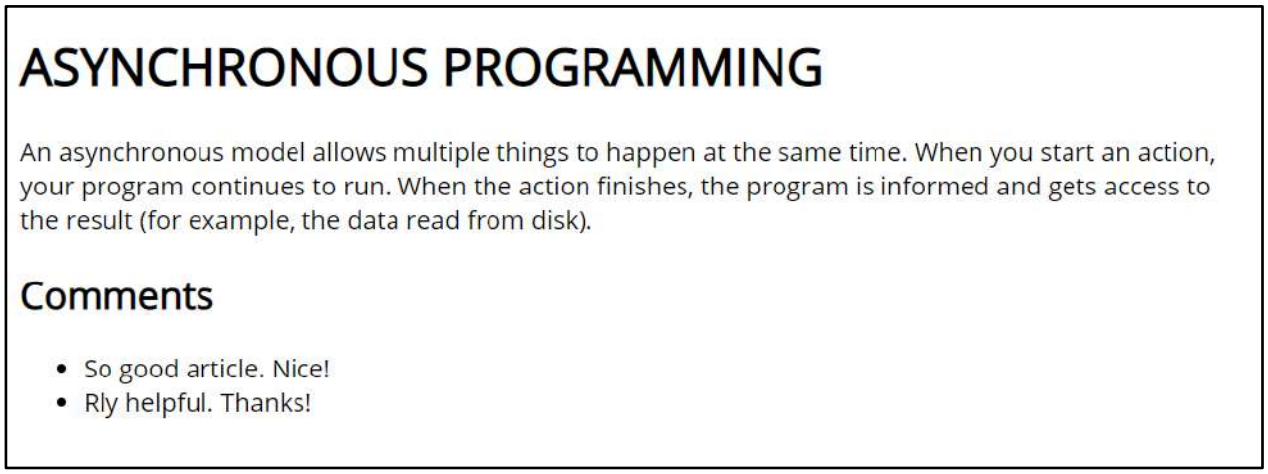
- "**/comments/:id**" to obtain the selected post (from the dropdown menu with ID "**posts**") - The following **request** will return **a single object** as described above.
- "**/comments** - to obtain all comments. The request will **return** a **Object** of **objects**.

Each object will be in the following format:

```
{
  id: {commentId},
  postId: {postId},
  text: {commentText}
}
```

You have to 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.




```
<h1 id="post-title">ASYNCHRONOUS PROGRAMMING</h1>
<p id="post-body">
  "An asynchronous model allows multiple things to happen at the same time. When you start an action, your program
  continues to run. When the action finishes, the program is informed and gets access to the result (for example, the
  data read from disk)."
</p>
<h2>Comments</h2>
<ul id="post-comments">
  <li id="rrt8713kjx1jda5r">So good article. Nice!</li>
  <li id="rrt878p0jx1jdgze">Rly helpful. Thanks!</li>
</ul>
```

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.