

Exercises: AJAX and jQuery AJAX

Problems for exercises and homework for the ["JavaScript Applications" course @ SoftUni](#).

Submit your solutions in the SoftUni judge system at

<https://judge.softuni.bg/Contests/1569/Exercise-AJAX-and-jQuery-AJAX>.

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 following HTML template to test your code:

buses.html
<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Bus Stop</title> <style> @import url(https://fonts.googleapis.com/css?family=Open+Sans); input[type=text] { padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box; } input[type=button] { background-color: #4CAF50; color: white; padding: 10px 16px; border: none; border-radius: 4px; cursor: pointer; } input[type=button]:hover { background-color: #45a049; } body { margin: auto; width: 25%; text-align: center; padding: 20px; } </style> </head> <body> <input type="text" value="Bus Stop ID" /> <input type="button" value="Get Buses" /> </body> </html></pre>

```

        font-family: 'Open Sans', serif;
    }
    #stopName {
        font-size: 1.5em;
        margin: 8px 0;
        font-weight: 400;
        padding: 0.25em;
        border-radius: 4px;
        background-color: aquamarine;
    }
</style>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
</head>
<body>
<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>
</div>
<script>
    function getInfo() {
        // TODO ...
    }
</script>
</body>
</html>

```

When the button with ID **'submit'** is clicked, the name of the bus stop appears and the list below 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 **<https://judgetests.firebaseio.com/businfo/{stopId}.json>** (replace the highlighted part with the correct value) and parse the response:

```

stopId: {
  name: stopName,
  buses: { busId: time, ... }
}

```

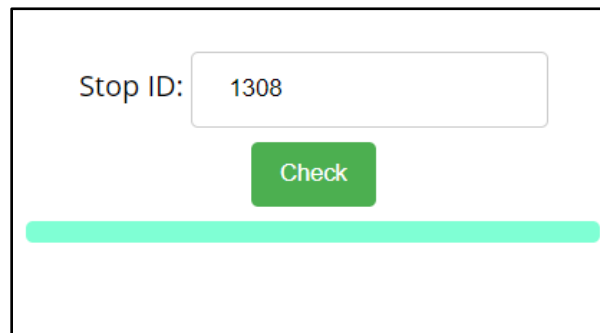
Place the name property as text inside the div with ID '**stopName**' and each bus as a list item with text:

"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.

Submit only the **getInfo()** function.

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:

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:

Stop ID:

Error

```

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

```

Hints

The webhost will respond with valid data to IDs 1287, 1308, 1327 and 2334.

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 following HTML to test your solution:

schedule.html
<pre><!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Bus Schedule</title> <style> @import url(https://fonts.googleapis.com/css?family=Open+Sans); input[type=text] { padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box; } input[type=button] { padding: 10px 16px; border: none; border-radius: 4px; cursor: pointer; } body { margin: auto; width: 25%; text-align: center; padding: 20px; font-family: 'Open Sans', serif; } #schedule { text-align: center; width: 400px; }</pre>

```

    }
    input {
        width: 120px;
    }
    #info {
        background-color: aquamarine;
        border: 1px none black;
        border-radius: 4px;
        margin: 0.25em;
    }
    .info {
        font-size: 1.5em;
        padding: 0.25em;
    }
}
</style>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
</head>
<body>
<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>
<script>
    function solve() {
        // TODO ...
        return {
            depart,
            arrive
        };
    }
    let result = solve();
</script>
</body>
</html>

```

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

`https://judgetests.firebaseio.com/schedule/{currentId}.json` (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):

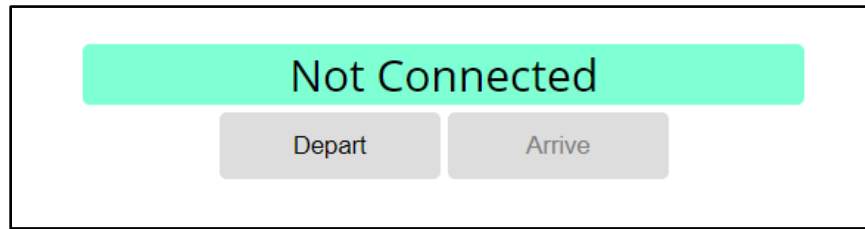
"Arriving at {stopName}"

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.

Submit only the **solve()** function that returns an object, containing the two click event handlers for **depart()** and **arrive()**, as shown in the sample HTML.

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:

Arriving at Depot

Depart

Arrive

```

▼ <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. 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. Use the following HTML to test your code:

messenger.html

```

<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Messenger</title>
  <style>
    @import url(https://fonts.googleapis.com/css?family=Open+Sans);
    body {
      margin: auto;
      width: 25%;
      text-align: center;
      padding: 20px;
      font-family: 'Open Sans', serif;
    }
    input[type=text] {
      padding: 12px 20px;
      margin: 8px 0;
      display: inline-block;

```

```

        border: 1px solid #ccc;
        border-radius: 4px;
        box-sizing: border-box;
    }
    input[type=button] {
        background: #4CAF50;
        color: white;
        padding: 10px 16px;
        border: none;
        border-radius: 4px;
        cursor: pointer;
        display: block;
    }
    input[type=button]:hover {
        background: #45a049;
    }
    label {
        display: inline-block;
        width: 5em;
    }
    #author, #content {
        width: 30em;
    }
    #controls {
        width: 38em;
    }
    #submit, #refresh {
        display: inline-block;
    }
    #submit {
        margin: 32px;
    }
</style>
<script src="https://code.jquery.com/jquery-3.1.1.min.js"></script>
</head>
<body>
<div id="main">
    <textarea id="messages" cols="80" rows="12" disabled="true"></textarea>
    <div id="controls">
        <label for="author">Name: </label><input id="author" type="text"><br>
        <label for="content">Message: </label><input id="content" type="text">
        <input id="submit" type="button" value="Send">
        <input id="refresh" type="button" value="Refresh">
    </div>

```

```
</div>
<script src="solution.js"></script>
<script>
    attachEvents();
</script>
</body>
</html>
```

Submit the `attachEvents()` function that attaches event listeners to the buttons and contains all program logic.

You will need to create the database yourself - use **Firebase** and set the access rules to be public, so that anyone can post a message and read what's been posted. Since Firebase objects are by default sorted **alphabetically**, you'll need to keep a **timestamp** property so they can be ordered by **most recently** posted instead. Use the following message structure:

```
{
  author: authorName,
  content: msgText,
  timestamp: time
}
```

The key associated with each message object is **NOT** important - when making a **POST** request with the message object as parameter, Firebase will **automatically** assign a random key. To get started, you can create a "messenger" entry in your Firebase and import the following **JSON** object:

messenger.json

```
{
  "-KWi2_-QHxL1yov93j5i" : {
    "author" : "George",
    "content" : "Hey, guys!",
    "timestamp" : 1479315195400
  },
  "-KWi2aENk0utP8BLnhi6" : {
    "author" : "Marry",
    "content" : "Whats up?",
    "timestamp" : 1479315200447
  },
  "-KWi3eFIUZbh8Z30jZEB" : {
    "author" : "George",
    "content" : "Not much, how about you?",
    "timestamp" : 1479315479039
  }
}
```

```
},  
"-KWiX5ixY39AqdD2hJzV" : {  
  "author" : "LJ",  
  "content" : "LEEEER000Y JEEEEENKIIINS",  
  "timestamp" : 1479323197569  
}  
}
```

Examples

George: Hey, guys!
Harry: Whats up?
George: Not much, how about you?
LJ: LEEER000Y JEEEEENKIIINS

Name:

Message:

Send

Refresh

George: Hey, guys!
Harry: Whats up?
George: Not much, how about you?
LJ: LEEER000Y JEEEEENKIIINS
Ritter: Not again

Name:

Message:

Send

Refresh

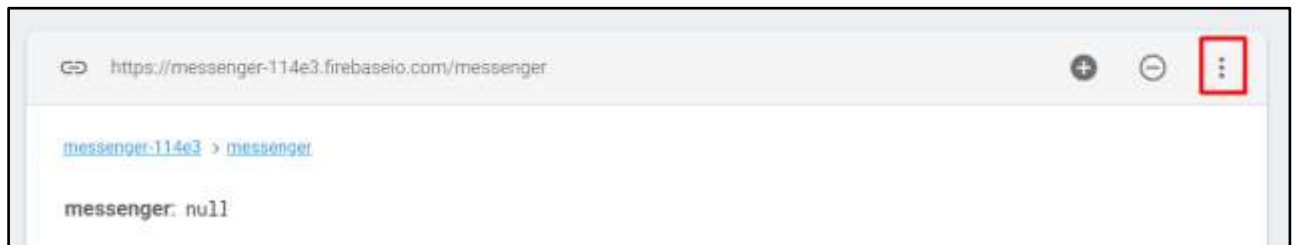
Hints

To get a useable timestamp, you can use **Date.now()** - this will return the number of milliseconds since 1st of January 1970. The exact value is irrelevant, what's important is, it will be greater for messages that are posted later. We can then sort them by this value.

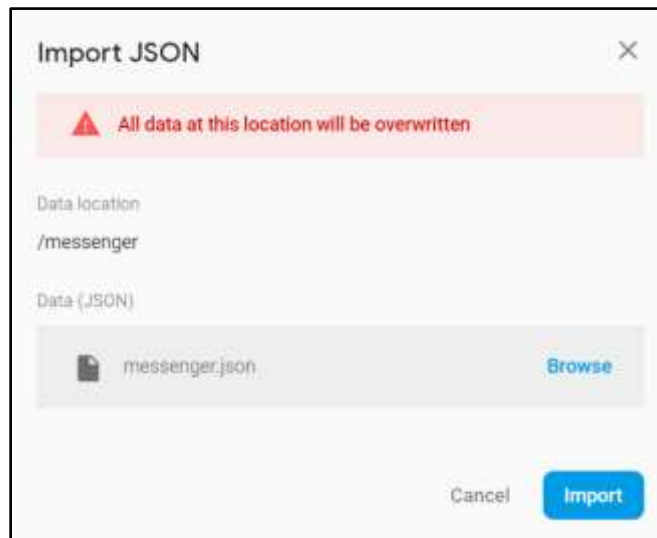
To create a new entry in Firebase, type its name in the address box and click Go:



You can then import content with the button in the upper right corner:



Put the sample data inside a file with extension **.json** and select it from the popup.



4. 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 should bind the needed functionality.

HTML Template

You are given the following HTML:

phonebook.html
<pre> <!DOCTYPE html> <html lang="en"> <head> <meta charset="UTF-8"> <title>Phonebook</title> <script src="https://code.jquery.com/jquery-3.1.1.min.js"></script> <link rel="stylesheet" type="text/css" href="style.css"> </head> <body> @import url(https://fonts.googleapis.com/css?family=Open+Sans); body { font-family: 'Open Sans', serif; } input[type=text] { padding: 12px 20px; margin: 8px 0; display: inline-block; border: 1px solid #ccc; border-radius: 4px; box-sizing: border-box; } </pre>

```

button {
  background-color: #4CAF50;
  color: white;
  padding: 14px 20px;
  margin: 8px 0;
  border: none;
  border-radius: 4px;
  cursor: pointer;
}
button:hover {
  background-color: #45a049;
}
</style>
</head>
<body>
  <h1>Phonebook</h1>
  <ul id="phonebook"></ul>
  <button id="btnLoad">Load</button>

  <h2>Create Contact</h2>
  Person: <input type="text" id="person"/>
  <br>
  Phone: <input type="text" id="phone"/>
  <br>
  <button id="btnCreate">Create</button>
  <script src="phonebook.js"></script>
  <script>
    attachEvents();
  </script>
</body>
</html>

```

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 a 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, they 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>"  
}
```

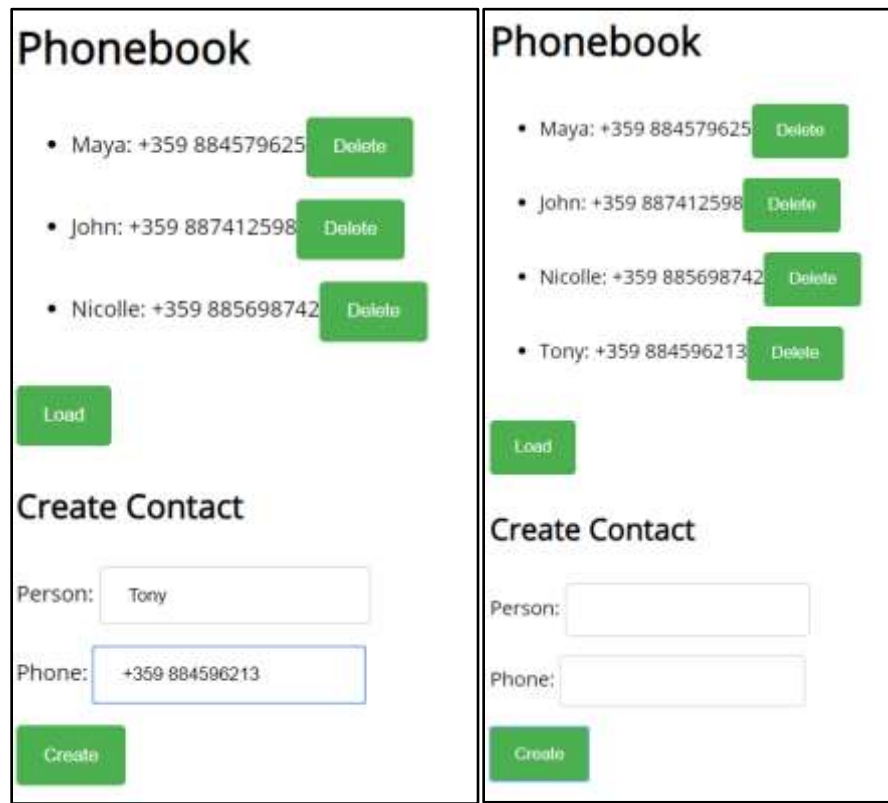
The **URL** to which your program should make requests is:

'<https://phonebook-nakov.firebaseio.com/phonebook>'

GET and **POST** requests should go to **<https://phonebook-nakov.firebaseio.com/phonebook.json>**, while **DELETE** requests should go to **<https://phonebook-nakov.firebaseio.com/phonebook/<key>.json>**, where **<key>** is the unique key of the entry (you can find out the **key** from the key property in the **GET** request)

You may create your own app in Firebase, the submitted code **will work** with any database from the same domain.

Screenshots:



Submit in the Judge only the `attachEvents()` function.

phonebook.js
<pre>function attachEvents() { //TODO }</pre>