JavaScript Day3 Agenda

When this day is over you should

- Have a basic understanding of Promises, the Promise API, and how promises are used by the fetch API
- Know how to use modern browsers fetch-API to:
  - Fetch data from a REST endpoint
  - Know how to manipulate the DOM using data fetched from external API's
  - Add (POST) data via a REST endpoint (primarily day-4)
  - Edit (PUT) data via a REST-endpoint (primarily day-4)
  - Delete data via a REST-endpoint (primarily day-4)

What you should read/watch before this lesson

📺	10 min.	JavaScript Promises In 10 Minutes *There is NO WAY you can learn everything about promises in 10 minutes given that this is our third day only, with JavaScript. I suggest you spend the 10 minutes by watching the first **five** minutes **twice**. You are NOT expected to write your own promise based code this semester, but you NEED to to know about promises to use the fetch API (next video)*
📺	6 min.	Learn Fetch API In 6 Minutes → Most important video for today If you want to code along (you do) you should use this API instead of the one he suggest, since the one he uses is no longer available: https://jsonplaceholder.typicode.com/users Here's a one minute video that explains how to set up your IDE, to code along with the "Learn Fetch …" video above,
📗	10-15 min.	How to use the fetch-api. Use this as a reference for how to use fetch, for the rest of the semester

Exercises

1) Dynamic UI manipulation using data obtained via **fetch**

Enter this URL in a browser and observe the result: https://jsonplaceholder.typicode.com/users/2

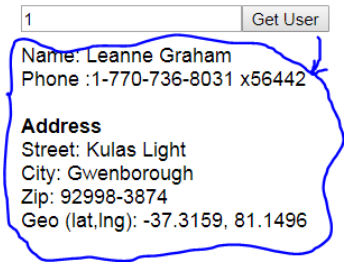
What is it, you get back?

Change the number at the end of the URL to any number <= 10 and observe the result.

See hints below before you start.

1a)

Implement a page, as sketched in this figure, that should fetch the requested user, and render his data.

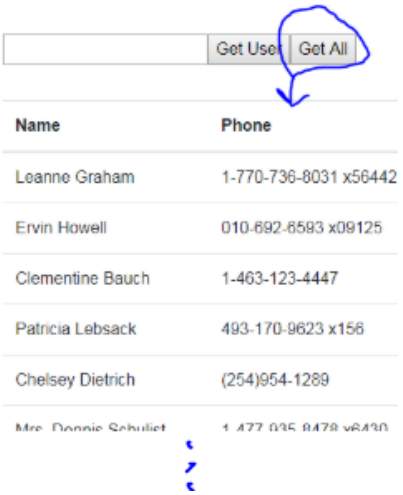


1b)

Enter this URL in a browser and observe the result: https://jsonplaceholder.typicode.com/users

Use this URL, and add a new button to the page as sketched in this figure.

When pressed, it should fetch all persons and render name + phone in a table



Hints:
Communication with a REST/JSON-based backend, is what we will do, almost on a daily basis for the rest of the semester (while coding). So for this getting-started exercise, just accept the hints below as something you do. The only conceptual difference between this exercise and the previous DOM-exercises is that data is fetched from a remote endpoint.

- For the first part you need to create the URL like this:
  let url = "https://jsonplaceholder.typicode.com/users/" **+ ID-FROM-INPUT**
- For the second part, there are no arguments, so you just use this url each time "Get All" is pressed:
  https://jsonplaceholder.typicode.com/users

HINTS: This is how you fetch the data (using the **url** variable declared above)

```
fetch(url)
  .then(res => res.json()) //for this exercise, just do this
  .then(data => {
   // Inside this callback, AND ONLY HERE the response data is available
   console.log("data",data);
  /* data now contains the response, converted to JavaScript
     Observe the output from the log-output above
     Now, just build your DOM changes using the data inside this block*/
})
```

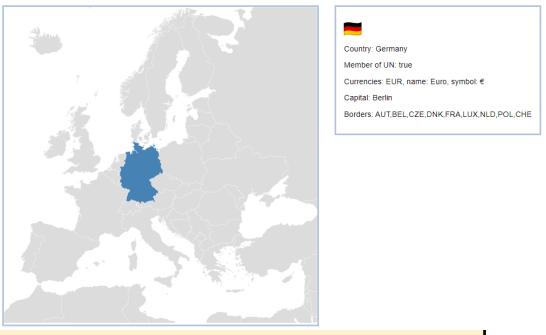
YOUR REPLY MUST INCLUDE (in a div on the page for the solution)
A "SHORT" description to the following

- What is a promise,
- What does the fetch method return
- Why do we need to have two `.then` callbacks with our fetch code?
- What can and should you do in the first?
- What can and should you do in the last?

2) DOM, Promises and fetch with SVG

This exercise is described in a separate document which you will find here

This is (somehow) how it should end (open developer tools → Network, and observe what happens when you click on a country):
https://countries.plaul.dk/



YOUR REPLY MUST INCLUDE (in a div on the page for the solution)
Explain in words how the concepts event-bubbling, DOM-manipulation and fetch was used in **your solution**

3) Using the Car's 'R' Us API

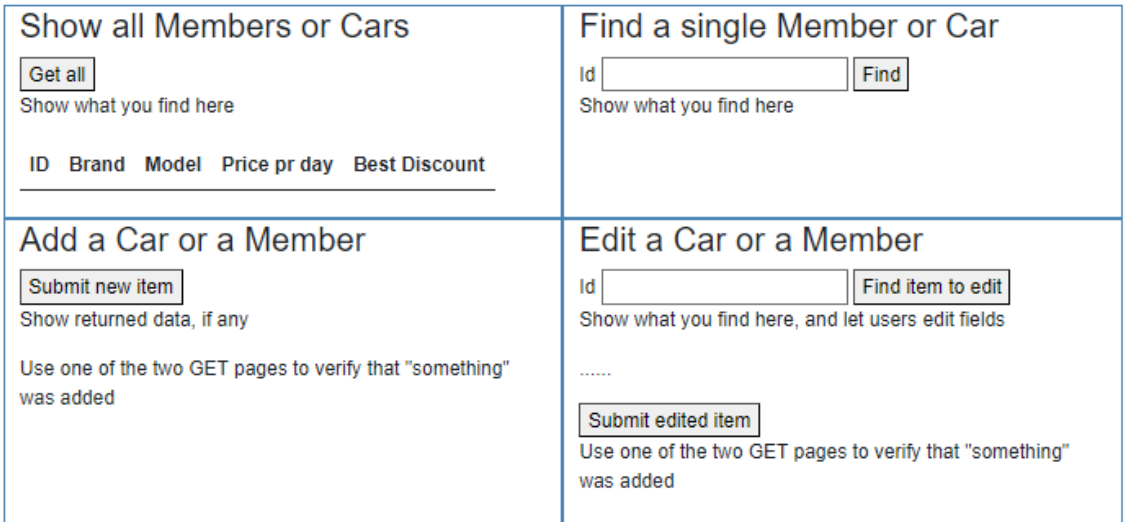
If you have problems with your own version of the Car's 'R' Us project, you can use mine which you find here (remember to add database-credentials to intellij)
https://github.com/kea-fall2023/cars-fall2023.git

SUPER IMPORTANT before you start.
*Change all your REST Controllers to include this annotation **@CrossOrigin** just below **@RestController**.*
If you don't, you will get strange errors from your JavaScript code. Details will follow in the next lecture.

This will be the first of several examples where you will use the Car's 'R' Us backend as your API.
The UI supplied for this exercise is DEFINITELY not how it should be done for real, don't worry about that.
Subsequent exercises will solve this problem. This is solely meant to introduce fetch with GET, POST, and PUT.

Create a new HTML file for this exercise. Feel free to add the required JavaScript embedded into this file, or **better**, in one or more separate JavaScript files.
Replace all content in the file with that from this link

This will provide you with HTML and CSS to render this very simple UI meant to be used with your Cars backend.



1. Start your (local) car's 'R' Us server, or choose either your CRUD API for Member or Car for this exercise.
2. Use Postman partly to verify that your API is functional, but also to see what you must include with the request to the API (URL, Headers and Body) for the required endpoints.
3. Implement the required JavaScript for "*Show all Members or Cars*", so a fetch GET-request is made whenever the button is pressed, and present the result either in a Table Ul or whatever.
4. Implement the required JavaScript for "*Find a single Member or Car*", so a fetch GET-request is made whenever the find-button is pressed, and present the result below the button.
   If you have time: Add the required changes to handle scenarios where a Member or Car was not found
5. Implement the required JavaScript for "*Add Car or a Member*", so a fetch POST-request is made, including data for the new item, read from the input fields, whenever the button is pressed. Present the result anyway you like.
6. Implement the required JavaScript for "Edit a Car or a Member'', so an initial fetch GET-request is made to get the item to be edited.
   When the user has edited the values, make a fetch PUT-request, including data for the edited item, whenever the submit button is pressed

IMPORTANT FOR THIS EXERCISE: This is a **fetch** exercise only (GET, POST, PUT) so DON'T focus on presentation in this exercise.

YOUR REPLY MUST INCLUDE (in a div on the page for the solution)

- Explain how you, for the POST-question, created the object that must match your DTO class on the server.
- What must be added to a fetch-POST request (similar to what you did with postman) Explain the purpose of each part

4) Beers

Extra - only if you are thirsty and HAVE TIME 😂

In this exercise you should create a simple WEB-page using this endpoint (test in a browser)
https://api.punkapi.com/v2/beers

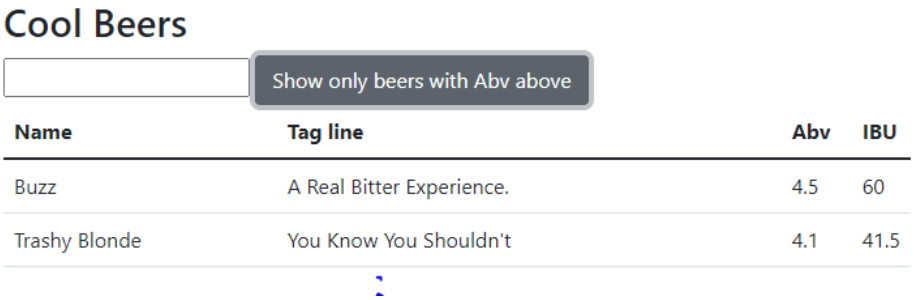
This API will provide you with a list of IPA beers with a huge amount of information, but for this exercise you only need the fields (for each beer) `name, tagline, abv, ibu`

Create a simple web-page with the following features

a) When the page initially loads, fehttps://github.com/kea-fall2023/cars-fall2023.gittch the list of beers, and do the following

- Render a table with name, tagline, abv and ibu as sketched below
- Store the list of beers for future reference (filtering)

b) Next implement a feature to lets uses see only beers with an abv (alcohol by volume) above what is entered in the input field



Hints: Add this HTML snippet to the body of your HTML file to get a look as sketched above. If you include the bootstrap css file in your file, it should render as above.

```
<h2>Cool Beers</h2>
  <input type="number" id="filter-abv">
  <Button id="abv-btn" class="btn btn-secondary">Show only beers with Abv above</button>
  <table class="table">
    <thead>
      <tr>
        <th>Name</th>
        <th>Tag line</th>
        <th>Abv</th>
        <th>IBU</th>
      </tr>
    </thead>
    <tbody id="tbl1"></tbody>
  </table>
```