AJAX with fetch and DOM manipulation

For each of these exercises, we suggest you use the simple SPA-startcode project, which you can clone from [here](https://github.com/Dat3SemStartCode/code_simple_SPA.git). All exercises, will use fetch. This is today the “preferred” way of making AJAX requests. Some older browsers however, does not come with fetch.

1. ES6 classes and Single Page Applications - No fetch()

Getting started

* This exercise assumes you have installed nodejs, and a lightweight JavaScript editor like vs-code
* Clone [this](https://github.com/Dat3SemStartCode/code_simple_SPA.git) project and navigate into the project folder:
* In this folder type **npm install** to fetch all dependencies (as had it been a Maven project)
* Type **npm run build** (yes before deployment this project has to be built)
* Take a quick look inside the generated **build** folder, and abstract away this folder for the rest of the exercise
* Now open the project in you favourite IDE (with vs code just (in the terminal) type code .)
* Back in the terminal type **npm start**.
* Now arrange your windows so you can see both your editor window (with the code) and the browser with the simple menu.
* Keep your windows arranged like this for the rest of the exercise

Finding individual jokes

In the public folder index.html file, add an input field,  a button with the text get joke, and a p-tag to hold the joke you will find. Investigate the start code and implement functionality (in index.js) to find a joke, given it’s id.

Adding new Jokes

Still only in the public folders index.html and in index.js, add the necessary changes to add new jokes to the internal joke-facade.

1. Small application to display a quote of the hour
2. Remove all html (and only this) inside the div with the container class  (in index.htm in the public folder)
3. Add a button to index.html + an empty div-tag, both with id’s so they are easy to “find”
4. In index.js remove all code meant for the initial joke-sample, add an event listener to the button’s click event and pass a callback that will update the div tag in the index.html with a new quote.
5. Initially, fetch the quote from a remote API:<https://studypoints.info/jokes/api/jokes/period/hour>
6. Use fetch() to get the quote.
7. Use developer-tools in your browser and it’s network options to monitor the AJAX-request. Explain why, what you did above, is even possible, when we know the Same Origin Policy governs when/where AJAX-request can go
8. Change the functionality to get a new quote every hour. (Hint: use setInterval() )

-------------------------------------------------------------------------------------------------------------------------------------------

3. JS Event handling, HTML5 and inline SVG ****

1. Download the file: [fourHearts.svg](https://github.com/Cphdat3sem2017f/StartcodeExercises/blob/master/JS/fourHearts.svg), and copy the content into the clipboard.
2. Either create a new SPA project or (suggested)  just paste the content into the body of your existing project.
3. Add the necessary event handlers, so when pressing each of the “hearts”, it will write the  message North, South, East or West respectively.

4. Ajax with a full REST CRUD Endpoint and Error-handling

*Creating Single Page Applications often requires the development of both a front-end (the SPA) and a backend (often REST\_JSON based). For this reason, it would be convenient if we had a way to quickly set up a* ***mock-backend*** *to use while developing our frontends.*

*Several such mock-backends exist, with* [*json-server*](https://github.com/typicode/json-server) *as one of the most popular. It allows us to set up a backend in less than a minute which can use data in a JSON-file as it’s “database”. Unfortunately, it does not provide errors as JSON, so for this exercise, we will use a* [*modified version*](https://github.com/Cphdat3sem2018f/code_jsonserver_with_errors)*, twisted for our needs this week.*

Setup the test backend required for this exercise (we will do together in the class)

*Note: This part assumes you have installed NodeJS*

Clone this modified json-server into a folder, somewhere on your system: <https://github.com/Cphdat3sem2018f/code_jsonserver_with_errors.git>

Copy the file [**users.json**](https://github.com/Cphdat3sem2018f/week6_javascript-2/blob/master/code/users.json) into the folder (the root) created by this project. This will actually act as the persistence media for your database, and will change when you start to add/edit/delete via the REST-API provided by the server:

Now type the following commands in the terminal:

npm install

npm start *(see the readme file in the project for alternative ways to start.  You will eventually need them all)*

Now you have a test server running with a full CRUD REST API, available via [http://localhost:3333/api/users](http://localhost/users)

Leave this terminal open for the rest of the exercise, and verify that the server is running

Enter this URL in your browser: [http://localhost:3333](http://localhost:3000) and read the instructions

Test the servers GET-methods in a browser, using these URLs

GET: http://localhost:3333/api/users/

GET: <http://localhost:3333/api/users/110>

GET: <http://localhost:3333/api/users/1111111111>  (to see an error-response)

Test the POST method

If not already done, install [Postman](https://www.getpostman.com/pricing)

Use Postman to make a **POST** request up against: **POST: http://localhost:3333/api/users/**

Set the following headers and body before posting:

Content-Type : application/json

Accept: application/json

Set this as **body  (raw)** (the new User we are going to create) (observe NO id)

{

"age": 23,

"name": "Peter Pan",

"gender": "male",

"email": "peters@pan.com"

}

Verify, via your browser that the new user have been added with an id.

Try, each of the following scenarios.

Same user as above, but age = 2. Observe the response (and status code)  
 "status": 400,

"msg": "Age must be >= 3 and <= 99"

Same user as above, but name=”ib” Observe the response (and status code)  
{

"status": 400,

"msg": "Age must be >= 3 and <= 99, Name must include at least 2 characters"

}

Same user as above, but gender =”mand” Observe the response (and status code)  
{

"status": 400,

"msg": "Age must be >= 3 and <= 99, Name must include at least 2 characters, Gender must contain 'male' or 'female' "

}

* Same user as above, but an illegal email.Observe the response (and status code)

 {

"status": 400,

"msg": "Age must be >= 3 and <= 99, Name must include at least 2 characters, Gender must contain 'male' or 'female' , Not a valid email"

}

Test the PUT method

Use Postman to make a **PUT** request up against: **PUT: http://localhost:3333/api/users/ID**

Add the ID for the user you are going to change to the end of the URL (red above)

Set the following headers and body before posting:

Content-Type : application/json

Accept: application/json

Set the body as the user returned from your first POST request, but change the name to Donald Duck

Verify, via your browser that the user have been changed

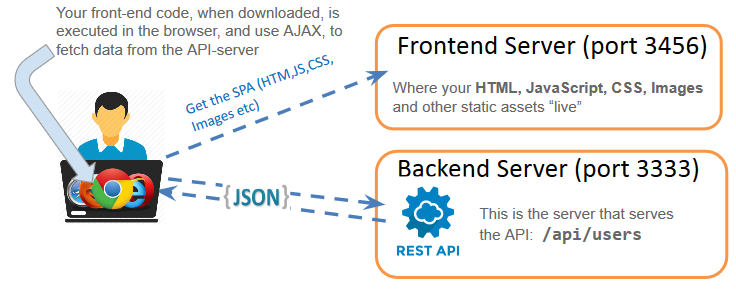
Test the DELETE method

Use Postman to make a **DELETE** request up against: **DELETE: http://localhost:3333/api/users/ID**

Add the ID for the user you are going to delete to the end of the URL (red above)

Verify, via your browser that the user have been deleted

Create a Single Page Application that uses our cool API :-)

*Since this (probably) is your first true Single Page Application, and since both the front- and backend-server involves a lot of new “magic”, like clone xxx, node, npm install, npm start etc. it can be hard to “see” the overall architecture of what you are building. This, however, is important, so use the figure below to get an idea about the “architecture” of what you will be building. Right now your backend should be running in a separate terminal (the API: /api/users) and you are about to start on the front-end (the SPA).*

Create a new project, using our simple SPA-start code project, for this exercise. Clone from here:

<https://github.com/Dat3SemStartCode/code_simple_SPA.git>

Use JavaScript, fetch and DOM-manipulation to create a SPA with the following functionality:

1. Show all users (in a table)
2. Show a single user, given an ID
3. Add a new User
4. Edit an existing user
5. Delete an existing user

You should only use the *index.html* file and the *index.js* file in the src-folder.

      If you want “more” pages, use DOM-manipulation, for example inspired by this [demo](https://www.w3schools.com/howto/howto_js_topnav.asp).

Hints: Read this [document](https://docs.google.com/document/d/1hF9P65v_AJKCjol_gFkm3oZ1eVTuOKc15V6pcb3iFa8/edit?usp=sharing) for hints about how to solve fetch-problems for this part

Error Handling

If not already done, provide the fetch examples above with sufficient error handling so that:

A request for a non-existing user, will render the error message *provided by the server*

Breaking one or more of the rules for new users, while submitting a new user, will render the error message *provided by the server*

Hints: Read this [document](https://docs.google.com/document/d/1hF9P65v_AJKCjol_gFkm3oZ1eVTuOKc15V6pcb3iFa8/edit?usp=sharing) (same as above) for hints about how to solve fetch-problems for this part

Document the API

This is probably the most important part of this exercise. You should postpone this part till Friday, and do it in a group, as described [here](https://docs.google.com/document/d/1upfPqiwEwJl5bONDW0oFtHkumh63e0_NC9p9Jo0WPUQ/edit?usp=sharing)

CORS (meant for Thursday)

Before you restart the server, monitor the request and responses sent to the server and explain how it’s possible to “break”

Go back to the terminal-window that executes your json-server backend. Stop the server (CTRL-C) and start it again, this time using this command:

**npm run nocors**

Test your SPA again, and explain the result.