Assignment 3

For ACS-3909-050

Due: November 21, 2023

Information:

- Your submission must **clearly indicate your name** in each file (use a comment in .js files and in .html files)
- Your **submitted filenames** must clearly indicate to which assignment and exercise they belong. For example, if you solve Assignment 1, exercise 1.1 name your file: "A1_E1-1_app.js" or something similar.
- Keep an eye on Nexus for clarifications/corrections of this assignment!
- **No late submissions**

The exercises will ask you to make yourself familiar with functions, packages, libraries, etc. that we have not explicitly introduced in the lectures. This is by design. I am linking for each exercise resources that can help you in finding the information you might need to fulfill the exercise.

In some instances, I will disallow or restrict the modules you are allowed to use. Besides theses instances there are many ways to do things, so see the resources I give you as hints – if you find another (even better) way of doing things: props to you!

Exercise 3.1 Templating with Pug

Possibly useful resources:

- Pug-Documentation: https://pugjs.org/api/getting-started.html
- Pug-Includes: https://pugjs.org/language/includes.html and others..., specifically: Interpolation, Inheritance, Iteration, Mixins

Warning:

- In the slides was a mistake: It says to register pug as view engine you would need `app.set('view-engine', 'pug')` but you need `app.set('view engine', 'pug')` no minus-sign between "view" and "engine"!

Part 1: Download the example for reactive variables from Nexus (found in week 8). This example, which we discussed in the lecture, uses Nunjucks templates. Change the server-file to use Pug templates instead and recreate the templates with Pug.

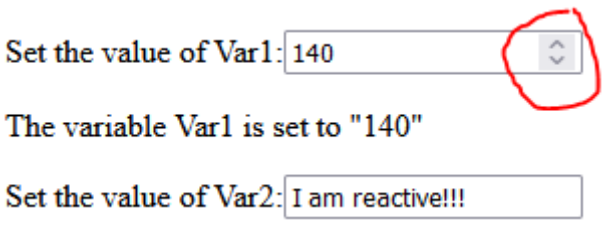
Part 2: Change your database entries, so that each document contains three fields like this:

```
_id: ObjectId('635180d4578e294a7eaae4f5')
varName: "Var1"
value: "150"
type: "number"
```

```
_id: ObjectId('635180e9578e294a7eaaf025')
varName: "Var2"
value: "I am reactive!!!"
type: "text"
```

```
_id: ObjectId('6351f7285e280871592ae60e')
varName: "Var3"
value: "I am not new anymore"
type: "text"
```

You can do this directly in MongoDB Compass or in the MongoDB web interface. Change your template (Pug or Nunjucks) so that integer-valued variables have a number-input as form-element like so:



Hints:

- Some editors do not like arrow-functions within Pug-files. This can result in breaking the live-review of HTMLs generated from Pug-files.
- When you are unsure how to access/combine variables or even Mixins with HTML-content (within paragraphs or scripts for example) *interpolation* is almost always the answer.
- Do not forget to update the information contained in credentials.js to your MongoDB server!
- The second part of this exercise is much less effort than you might expect.

Exercise 3.2 Towards Event-Sourcing

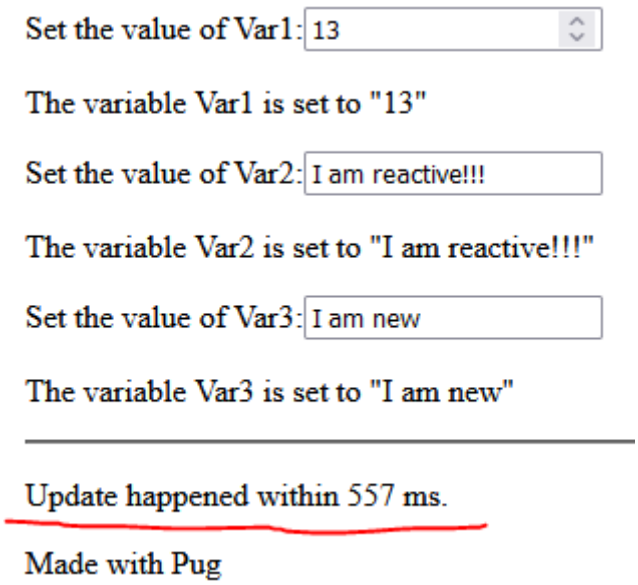
Resources:

- Date.now(): https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/now

In this exercise you are asked to improve the performance of changes to the variables in our reactive example. For this we investigate two methods.

Part 1: Before we can make any improvements, we must first measure the performance of our application as it is now. Change your templates such that the time that elapses between changing one of the input fields and updating the displays is tracked. To be precise:

- You should note the time whenever one of the input fields is updated.
- You should note the time after all -tags had been updated.
- Print the difference of these two times to the browser in a new line like so:



Unless you have changed the code, you will see that the first time to update a variable takes rather long (300+ ms) and subsequent updates are faster (e.g., around 70 ms for my setup). **Write an explanation for this phenomenon and submit it together with any code you submit for this exercise!**

Part 2: Now change your application such that:

- Whenever an update is sent from the browser to the express-application the web-socket answers immediately with the updated variables and **then also** updated the MongoDB.

- Finally, after the MongoDB has been successfully updated this update is still messaged through the WebSocket to the browser (This way every browser currently connected is being informed about changes, even if we manipulate the DB through other means, for example, via using MongoDB's web interface).

Measure now two times: 1) The time until the browser's -tags are updated the first time (immediate response by the express-application) and 2) The time until the browser's -tags are updated the second time (after the MongoDB has been updated). Like so:

Update committed to MongoDB within 400 ms.

Update happened within 5 ms.

Made with Pug

Above: Times for first update

Below: Times for second update

Update committed to MongoDB within 66 ms.

Update happened within 2 ms.

Made with Pug

Part 3: Now we change the application again. Instead of waiting for an update from the server-application we want to update the -tags immediately on the browser-side (without waiting for anything coming through the WebSocket, we **still** inform the express-app about the update, though). *Then* we will also update the tags when the response from the WebSocket comes. Again, measure both elapsed times. Here is an example, of what it could look like.

Update committed to MongoDB within 67 ms.

Update happened within 0 ms.

Made with Pug