Lab 3

This is the final laboratory in the web programming course, *objectives*:

1. Get experience using a REST api for fetching data.
2. Get experience with chaining `Prototypes`
3. Get experience using persistent storage in the web browser.

Background

The assignments bellow assumes you have a working solution for lab 2, i.e. a working react app with three components: `App`, `ComposeSalad`, and `ViewOrder`.

Assignments

1. We are going to remove the inventory from our compiled code and instead fetch the data from a REST server. The server is already implemented, but you need to download it and run it on your own computer. It is based on the `restify` package, so you need to download that as well:

```
> cd my_working_lab_2
> curl -o server.js http://fileadmin.cs.lth.se/cs/Education/EDAF90/labs/lab3/server.js
> npm install restify
> node server.js
```

The code above downloads the backend into your frontend project. Normally you want separate projects, but it is only one file, which you will not change, so let's keep things simple. The server should be running and waiting for requests. Test it using `curl` in the terminal, or write the urls in a browser:

```
> curl -is http://localhost:8080/foundations/
> curl -is http://localhost:8080/proteins/
> curl -is http://localhost:8080/extras/
> curl -is http://localhost:8080/dressings/
> curl -is http://localhost:8080/dressings/Dillmayo
```

2. Your next assignment is to fetch the inventory from the salad bar REST server. You could fetch the data when needed, but in this case i suggests you fetch all data when the app launches. You can recreate the same format for the inventory as in the `inventory.ES6.js` file. This leads to minimal change in the rest of your code. Open `App.js` and remove the import line:

```
import inventory from './inventory.ES6';
```

You will get some compile errors since the name `inventory` is no longer defined. We are going to store the data for the entire lifetime of the app. The best place to store it is in the state of `App`, in the constructor:

```
this.state = {orders: [], inventory: {}};
```

Update the rest of your code so it compiles (replace `inventory` with `this.state.inventory`.

3. Normally a web browser will only allow a web app to send http requests to the server it was downloaded from, its origin. The reason is to protect the user from cross site scripting attacks, which will be covered in the last lecture. The origin is both the IP-address and the port. The salad bar REST server is running on a different port than the react development server, so the servers have different origins and, by default, the browser prevents your app from communicating with the salad bar REST server. Luckily there is a way to relax this constraint. A server can allow communication with scripts from other origins using the Access-Control-Allow-Origin header. If you look at the headers returned by the salad bar REST server, see the curl commands above, you see that the server allows access from *, meaning any server. The browser still do not trust this communication, and hides most http headers. Do not bother looking for the header information in your app. In the lab you may assume that the body contains json data, however do check the status code to make sure your request was successful.

   When should you fetch the inventory data? Check out the react lifecycles, see `https://reactjs.org/docs/state-and-lifecycle.html`. Remember, must of course create an initial state in the constructor. Hint, if you block in the constructor or render, your app will freeze. Check your code, does it block? It it does, can you rewrite your cod so it do not? Here is a blogpost about this topic: `https://www.robinwieruch.de/react-fetching-data/`

   Use the `fetch(url, [options])` function to send a request. It might be easiest to build the url using string concatenation, but you can also check out the `URL(url, [base])` class. Check out the documentation for `fetch()`. It returns a `Promise` that resolves to a `Response` object. To get the body you can use `Response.json()`, which returns yet another `Promise`. When you have the ingredient name, the next step is to fetch its properties, with one more call, like `fetch('http://localhosr:8080/extras/Tomato')`. All of this is sequential by nature, so use `await` in `async` functions, or chains of `then`. Fetching several ingredients are independent and should be carried out in parallel. This is a perfect use case for `Promise.all()`. It takes an array of `Promise` objects, which will run in parallel. It returns a `Promise` that resolves when all inner promises are resolved. Use this to wait until all ingredients have been fetched befor updating `this.store`.

4. Show how you can send data to the server using a `POST` request, for example the list of salads in an order. The REST api can be tested using:

```
curl -isX POST -H "Content-Type: application/json"
     --data '["salad1", "salad2"]' http://localhost:8080/orders/
```

5. I have one more assignment for you. Store the order in local store, and load it when the app starts. This is done using the `window.localStorage` There are two functions, `setItem(key, value)` and `getItem(key)`. All values are text, so use `JSON.stringify()`, and `JSON.parse()`.Note, `parse` gives you a JavaScript object with the right structure, but it is not a `Salad`.The `price()` method et.c. are missing. This can be solved in two ways:

   - Use `Object.setPrototypeOf()` so set up the right prototype chain for the objects returned from `JSON.parse()`
   - Make a static function, `Salad.price(obj)`. This can not be achieved using ES6 classes, instead: `salad.price = function(obj)...`

Editor: Per Andersson

Contributors in alphabetical order: Per Andersson

Home: `https://cs.lth.se/edaf90`

Repo: `https://github.com/lunduniversity/webprog`

This compendium is on-going work.
Contributions are welcome!
Contact: `per.andersson@cs.lth.se`