**React and React Router**

General part

* *Describe the term Single Page Application and why it is relevant for modern web-applications*
* *Describe fundamental topics in React, like the Virtual Dom, JSX, Rendering, Components, Props, State, Life Cycle etc.*
* *Explain about Components and Routing in React*
* *Explain the concept CORS, and why it is (often) relevant for Single Page Applications*

CA or Semester Project

If you get stuck with the following exercise in a way that you do not have a lot of code to show, you can demonstrate (max 5 minutes) how you have used Routing/navigation, in one your CA's or in the semester project. Feel free to demonstrate a React Web Application or a React Native App.

Practical part

*Note: Skim all the questions before you start. If you get stuck somewhere in part 1-4 you can jump to part 5b. This could be just enough to make you pass.*

|  |  |
| --- | --- |
| **1) Getting started:**  Create a new React Project named *router\_exam* with create-react-app   * In the root of the project type npm install react-router-dom --save * Replace all content in App.js with this code: [**https://github.com/Cphdat3sem2017f/StartcodeExercises/blob/master/testexam/App.js**](https://github.com/Cphdat3sem2017f/StartcodeExercises/blob/master/testexam/App.js) * Replace all content in index.css with the content found here (to style similar to figures below): <http://sem3slides.mydemos.dk/style.css> * Run *npm start* and verify that you have an (almost) working router example. | |
| **2) Provide the necessary code to navigate between the three routes.**   1. For the Home link, and any non-existing route, it should display a simple message like: "Welcome …" 2. For the About, and Repositories Links, use the provided components (About and Repositories) | |
| **3) Spend 2 minutes (NOT more) observing the JSON returned by this REST-call:** <https://api.github.com/orgs/Cphdat3sem2017f/repos>  *Observe that you get an array containing objects representing Git-repositories. Initially you should* ***only*** *focus on the fact that each repository-object has a* ***name*** *property* ***(****ex****:*** *StartcodeExercises)*  In the Repositories Component, use fetch() to fetch data from the GitHub-API (given in the link above). Render an unordered list with all the name properties as shown in the image to the right. |  |
| **4)Add a Link, as shown in the image, to navigate to the Repository component** (not shown in the image).  This component should show the details of the selected repository in a new route.  Initially it's enough to just show the name, which is what you should pass into this new control as a URL-parameter.  Github provides an endpoint to fetch a *single* repository, given its name as shown here with a name parameter: <https://api.github.com/repos/Cphdat3sem2017f/:name>  *(test in a browser with one of the repository names returned by the link given in step 3)*  In the Repository component, fetch data for a specific repository based on the name like in the URL given above. Render just a few of the details like full\_name and size. |  |

**5)** For this part you decide whether you want to do **5a** (continues with React) or **5b** (REST + JPA). You are NOT expected to do both (unless you really feel you have the time).

**5a) Adding a facade to the React code**

The example above was not optimal when it comes to network performance. In step four we made a network request to fetch details for a single repository, but this information was already provided with the first request (step 3).

**5a\_1)** Refactor the code that fetches data into a façade class, and store the result from the call that provides all repositories (link given in step 3).

**5a\_2)** Provide the façade with a method getRepositoryFromName(name), which should search the stored repository, and return the repository with the provided name.

**5a\_3)** Refactor the code for step 3+4 to use this façade

**5a\_4)** Deploy the project, preferably using your own domain name, alternatively on surge.sh.

**5a\_5)** Using a stored version of the repository-data saves bandwidth, while user is navigating the app.

* What are the downside to this strategy?
* Can you come up with alternatives (only in words, not code) to minimize the downside

**5b) Create a simple Jax-RS endpoint to “mirror” the response we got from github**

In this part you must implement a simple Entity Class which should mirror some of the information we got from the first git-endpoint (so we can change the client code to use this new endpoint).

**5b\_1)** In Netbeans, create a new Maven Web Project

**5b\_2)** Use the wizard to create a new Entity class GitRepo with the following properties:

* id : Integer (auto generated)
* name: String (unique)
* full\_name: String

**5b\_3)** Create the matching table (via persistence.xml) , and populate it with a few dummy entries

**5c\_4)** Create a REST endpoint that returns all GitRepo-entries as a JSON-array.

**5c\_5)** Change the URL in part 3 above, to use this new endpoint (remember to solve the CORS-problem)

**5c\_6)** Deploy the project, or if you don't have time, just explain the steps you will have to complete, in order to do this