

React, React Router, and JAX-RS

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 Routing in React
- Explain the fundamental properties of REST
- Explain the concept CORS, and why it often is relevant for Single Page Applications

What if I get “stuck”

Note: Skim all the questions before you start. If you get stuck somewhere in part 1-4 you can jump to the alternative question (in this exercise this is just part-5. For a real question it would be a smaller part of a “real” question It will be something “quite different”). This could be enough to make you pass.

As a last alternative, if you get totally stuck with the practical part, you can chose to demonstrate how you have used the main topic for this exercise, in the semester project and/or in a CA.

If you select this option, you must first demonstrate how far you came with the practical part and the actual problem that made you select this option.

Practical part

1) Getting started

Create a new React Project 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: [App.js](#)
- Replace all content in `index.css` with this (just, to style your code similar to the figures below): [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

- For the Home link, and any non-existing routes, display a simple message like: "Welcome ..."
- 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/Cphdat3sem2018f/repos>

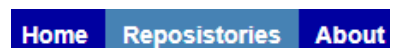
Observe that you get an array containing objects representing Git-repositories. Initially, you should focus **only** on the fact that each repository-object has a `name` property (ex: `StartcodeExercises`)

In the Repositories Component, use `fetch()` to fetch data from the GitHub-API (given in the link above). Use `map` to render an unordered list with all the name properties as sketched in this figure.



4) Add a Link to all items in the list, as sketched in this image, to navigate to the Repository component (not shown on the figure).

The `Repository` component must show the details for a selected repository.



4a)

Initially, it's enough to just show the name in the `Repository` component. This is what you should pass into the control via a *URL-parameter*.

4b)

Now use the name passed into the control, to fetch the details for that name. Github provides an endpoint to fetch a *single* repository, given its name as shown here:

<https://api.github.com/repos/Cphdat3sem2018f/: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 some of the details like *full_name*, *size* and *a few more*.

4c) Complete part 5 first, and then come back to this if you have the time

Observe that data returned for a single repository contains an *owner*-property, which is an object with details related to the owner. Come up with an addition to 4b) where users can press a button/link and see “all” the details for the owner. Feel free to implement this in any way you like (a new route, a modal or an alert-box or whatever you feel would make sense)

REST and JPA

5) Create a simple JAX-RS endpoint to “mirror” the response we got from GitHub

In this part, you must implement a simple Entity Class (+ a matching database table) with properties that should mirror (some of) the properties we got from the original git-endpoint. After that, you must implement a REST endpoint to return JSON-formatted information, fetched from this table

5.1) In Netbeans create a new Maven Web Project

Note: Step 5.2 + 5.3 suggest that you create the entity class first, and then generate the matching table from the class. Feel free to do it the other way around if you prefer (table first).

5.2) Use the wizard to create a new Entity class `GitRepo` with the following properties:

- `id` : `Integer` (auto generated)
- `name`: `String` (unique)
- `full_name`: `String`

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

5.4) Create a REST endpoint that returns all our `GitRepo`-entries as a JSON-array (remember to import the necessary dependencies in your POM-file).

5.5) Change the URL in part 3, to use this new endpoint (remember to solve the CORS-problem)

5.6) Deploy the project (Client and Server), or if you don't have time, explain the steps you will need to complete, in order to do this