Web Development and API Design

Lesson 05: SPA Routing

Prof. Andrea Arcuri

Goals

- Understand how routing between pages is done in a SPA
- Understand difference between *client-side* and *server-side* HTML rendering, and why you want to support **both** in a SPA

Single-Page-Application (SPA)

- As the name suggests, there is only one single HTML page
- All changes are made via JS and DOM manipulation
- Browsing from one page to another one just means changing current DOM via JS
- We give the user an *illusion* of changing pages
  - possibly, with no new HTTP request to the server, which would happen when following **<a>** links
- We even change the URL in the browser address bar, without triggering an HTTP request

React-Router

- Library to support SPA in React
  - Note: it is not made by Facebook… React is just a HTML rendering library, with no base support for routing
- Technically, one single “*index.html*”, with all components for all web pages there
- Which components will be displayed depends on the URL address bar

Cont.

```
<div>
    <div>/* Page for /foo */</div>
    <div>/* Page for /bar */</div>
    <div>/* Page for /     */</div>
</div>
```

- Example, single "*index.html*" for 3 web pages
- If URL has "*/foo*", then display the first page, and use JS to *hide* the others (ie, do not generate HTML for them)
  - eg, **localhost:8080/foo**
- *Navigation*: change the URL address bar content, and re-render the "*index.html*" page

Navigation Example

- With browser you open **localhost:8080**
- Browser will download *index.html* and all static assets referenced from it (eg *bundle.js* and CSS style files)
- Default path is the root “**/**”
  - e.g., typing **localhost:8080** is equivalent to **http://localhost:8080/**
- Clicking on a SPA link for “**/foo**” will change the address bar into **http://localhost:8080/foo** , *without* making a HTTP request to the server
- React-Router detects the change in the address bar, and re-render the page

Issues

- What if you bookmark **localhost:8080/foo** ?
- Or what if you refresh the page in the browser?
- In such cases, browser will make a *TCP* connection to **localhost:8080**, with a *HTTP GET* command to retrieve the resource **/foo**
- But the resource **/foo** does NOT exist, so you get a **404**…
  - recall, it is dynamically generated with JS
- Two possible solutions for this problem

Solution 1: Return *index.html*

- Instruct the server to never return a 404
- Each time a HTTP GET asks for something that does not exist, rather return the content of *index.html*
- *React-Router* will render the page accordingly based on the URL (as downloading all static assets, including *bundle.js*)
- If the URL points to something that is not recognized by the application, use JS to create a custom error message page
- Note: this solution is very easy to implement, but it is not the best… however we will use it for the rest of the course

Solution 2: Server-Side Rendering

- When server gets a HTTP request for a dynamic page, do create it on the server
  - this means running *React* on the server, and return the generated HTML as body payload of the HTTP response
- Server needs to be able to run JS code
  - simple in NodeJS, but tricky (not impossible) if your server is in a different language, eg Java, C# or PHP
- *Issues*: far from trivial… as there are **a lot** of edge cases to consider
  - eg frontend rendering depending on AJAX/WebSockets, and initialization of data-stores like *Redux*

Benefits of Server-Side Rendering

- *Can navigate the web application even without JS*
  - eg, *React-Router* will create SPA links using **<a>** tags with overridden JS handling for *onclick* events, but such special handling will not be executed without JS, and so defaulting on default **<a>** behavior of making a HTTP GET request
- Why should you care?
- **Search Engine Optimization (SEO)**
  - you want your web applications to appear in Google Search, Bing, etc.
  - not all crawlers execute JS, and, even if they do, they might be limited
  - you do not want crawlers to just see empty *index.html* files for your whole app