

Web Development and API Design

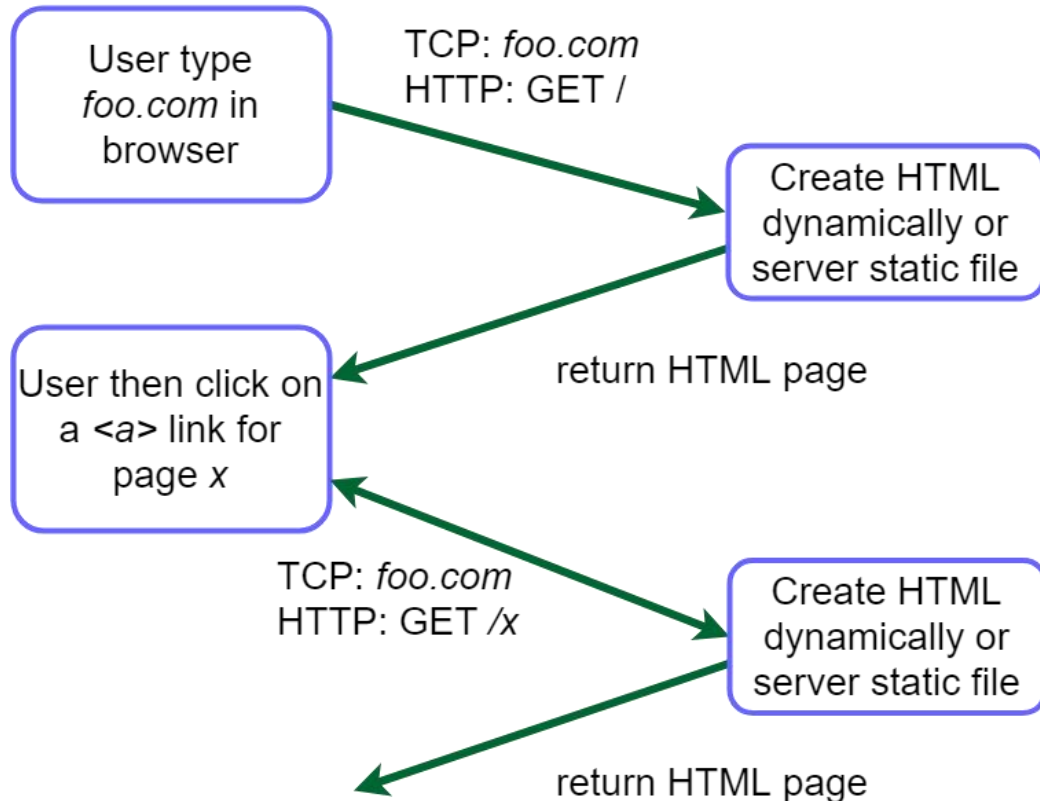
Lesson 03: SPA Components

Prof. Andrea Arcuri

Goals

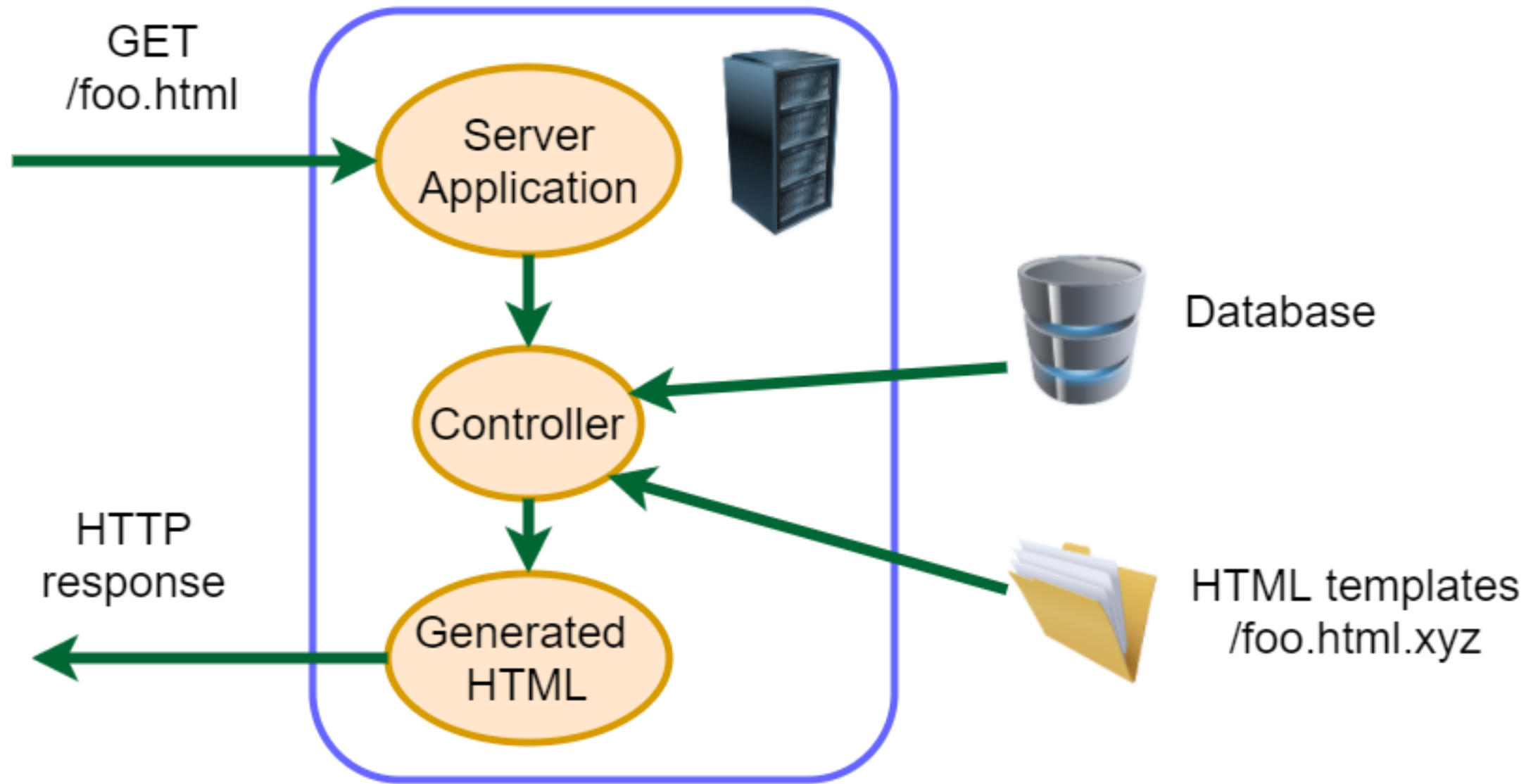
- Learn the main concepts behind Single-Page-Applications (SPA)
- Understand why direct DOM manipulation is not-recommended, and a library/framework should be rather used
- Understanding the need for *Components* in SPAs
- Introduction to *React*

Traditional Web Applications



- Navigation with HTML tags like **<a>** and **<form>**
- Each request is a HTTP message, eg GET or POST
- Get a full HTML page (could be dynamically generated server side)

Server-Side-Rendering



Single-Page-Applications (SPA)

- There is only one single HTML file, with no content
- All the HTML content is **dynamically generated on the browser** with JavaScript
 - ie, by manipulating the DOM
- **Navigation between pages is simulated** by modifying the GUI on the fly (including changing the URL in the address bar)

Fetching New Data

- Even if HTML is generated on browser with JS, we still need to communicate with server
 - to save/load data
- We will **NOT** get any new HTML file
- Just data in JSON format
 - *JavaScript Object Notation*
- JS will update DOM based on JSON data
- *Web Servers* will provide the JSON data
 - in rest of the course, we will see REST and GraphQL APIs

SPA Complexity

- Now we can have a LOT of JS code in the frontend
- *Manually* updating the DOM at each *state change*, and at each *browser event* is not scalable
 - Can be done, but it quickly becomes a mess
- We need design patterns and tool support to handle such complexity

Libraries/Frameworks

- Frontend technologies vary very quickly
- As this time of writing, there are 3 main ones, all *open-source*
- **React**: made by Facebook
 - the one we use in this course
 - most popular, widely used in many Norwegian companies
- **Angular**: made by Google
 - whole framework, heavy-weight
 - still widely used, but losing popularity
- **Vue**: one main developer/author
 - very popular in Asia
 - bus factor...

React Components

- Define components (e.g., like objects) with a **state**, and a way to **render** HTML based on such state
- Web page represented with a root component, with children components, in a *tree* structure
 - each component has its own state, and only knows how to render itself
- We will NOT call the rendering directly
- We just change the state of the component, and React will automatically re-render what needed

Rendering Optimizations

- There can be many events in a browser (user clicks, mouse movements, etc)
- React can *automatically* optimize when HTML needs to be re-rendered
 - eg, squashing together several updates that happen within few milliseconds
- Virtual-DOM
 - Even if a component's state is changed, it might be that only small parts of its HTML is now different, if any at all
 - React does not naively re-render the whole HTML, but just what is actually needed to be modified
 - It keeps a Virtual DOM in memory, and only updates the actual GUI in browser in what it differs from the VDOM

JSX

- A React Component will generate HTML code via its *render()* function
- Handling HTML as JS strings is too error-prone
 - e.g., lack of static validation of HTML grammar
- JSX: a file format for React in which you can mix JS and HTML together
- Browsers have NO clue of JSX... you need to use Babel to transform JSX into JS
- Note: we will use “.jsx” suffix to represent JSX files... but it is possible to use “.js” as well, although it is arguably a bad practice