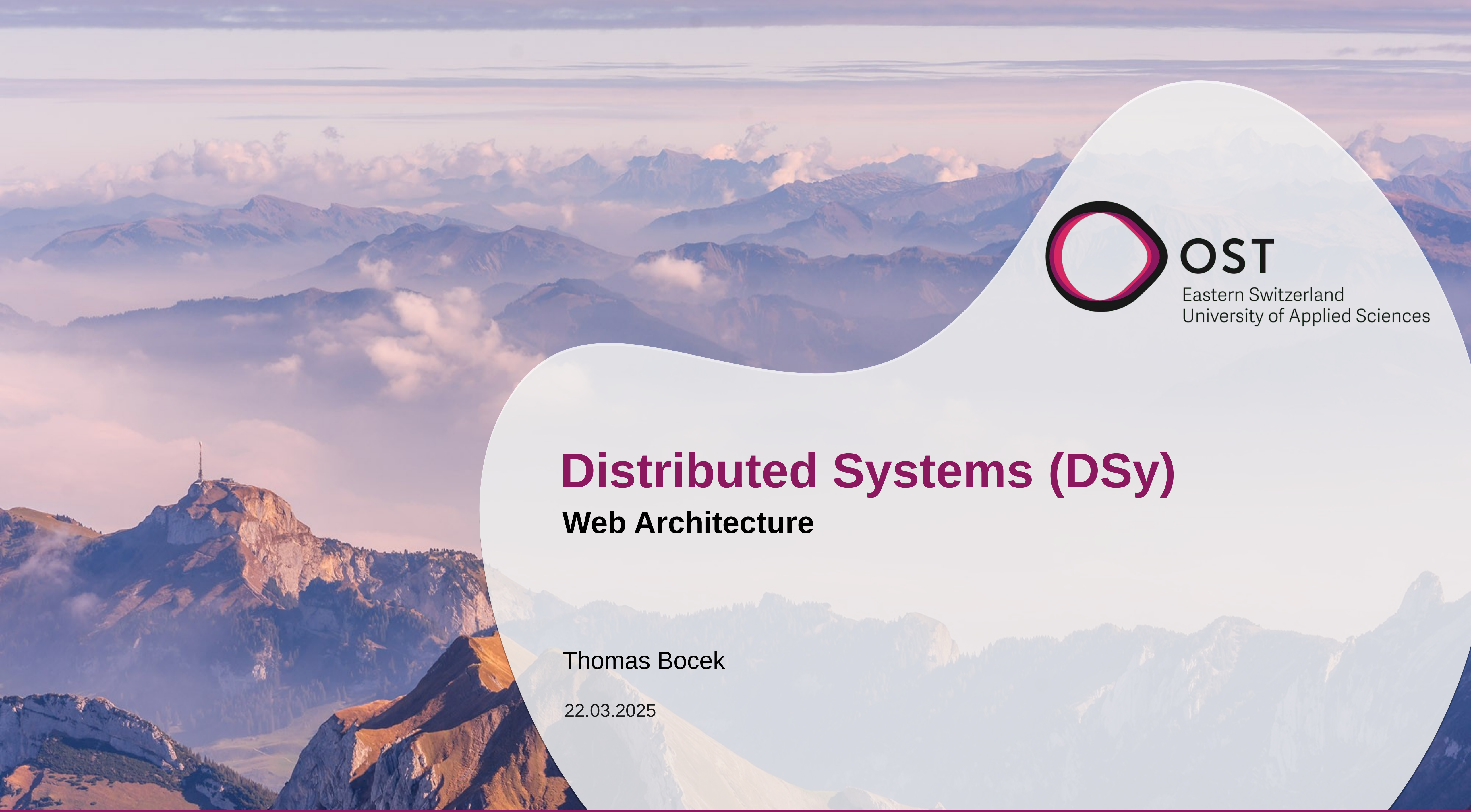


OST
Eastern Switzerland
University of Applied Sciences

Distributed Systems (DSy)

Web Architecture

Thomas Bocek

22.03.2025

Learning Goals

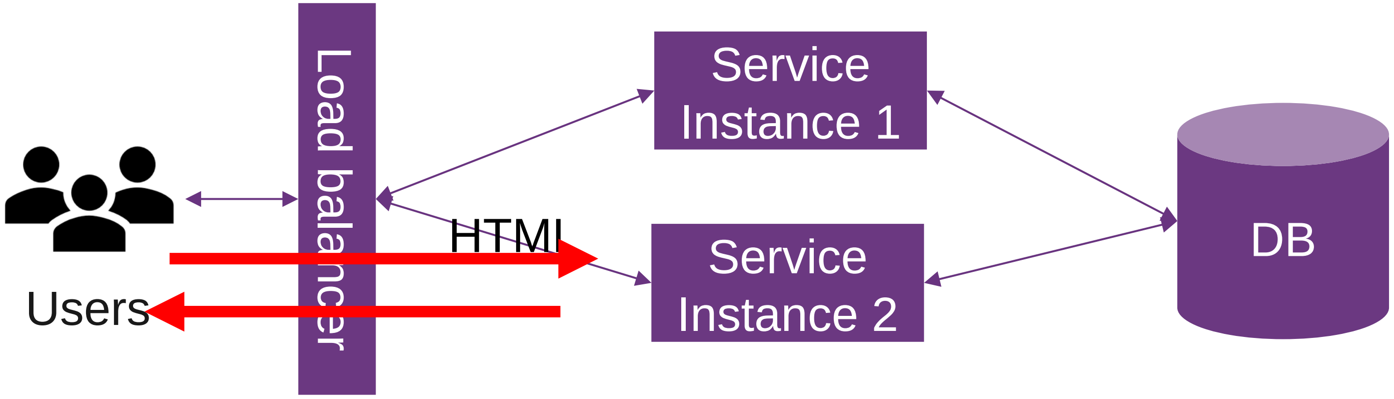
- Lecture 6
  - What are the options to build my challenge task?
  - What is currently “state-of-the-art”?
  - CORS

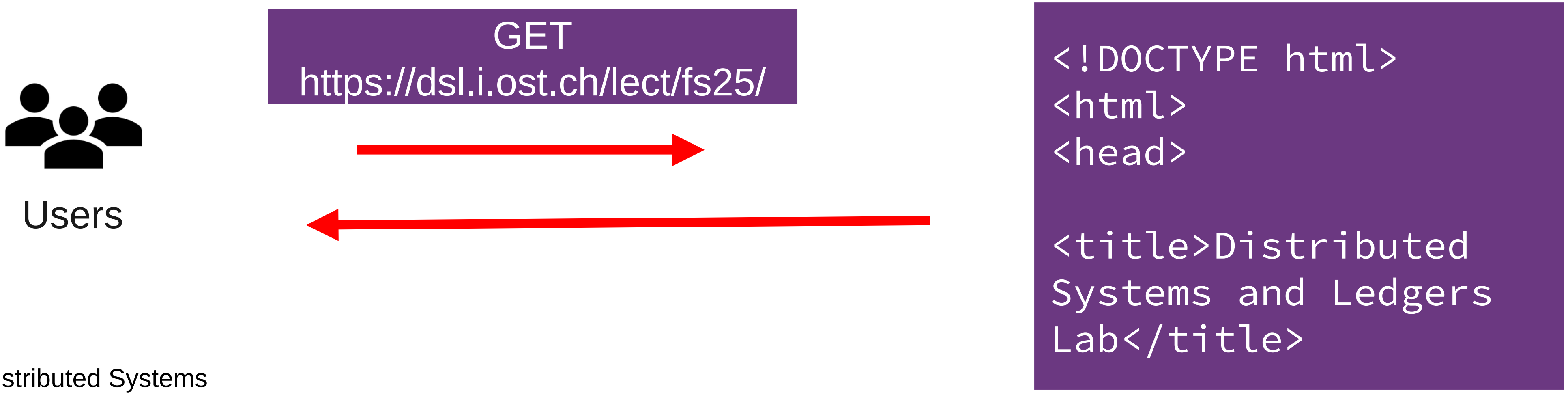
Server-Side Rendering

- “Classic” approach - “SSR”
- Server generates HTML/JS/CSS dynamically, sends the assets in real-time to the browser
    - User request: browser sends a request to the web server (server-side routing)
    - Server processing: server processes request by running server-side code (e.g., C#, Java, …),
        - Fetch required data from a database or other sources
        - Server-side code can use template engines to render the HTML - reusability
    - Response: Generate the appropriate HTML, CSS, and JavaScript for the requested page.
    - Browser rendering: browser receives response and renders page
- Big advantage: SEO, but needs the server rendering for every request (caching!)
- Static site generation: pre-render HTML/CSS/JS since its the same for every user. Done only once, resp, if the content changes.
    - https://dsl.i.ost.ch → markdown to HTML
    - Can also include DB access

Server side rendering (SSR) Simple Example

- Request entire page





GET https://dsl.i.ost.ch/lect/fs25/

```
<!DOCTYPE html>
<html>
<head>

<title>Distributed
Systems and Ledgers
Lab</title>
```

Single Page Application SPA / CSR

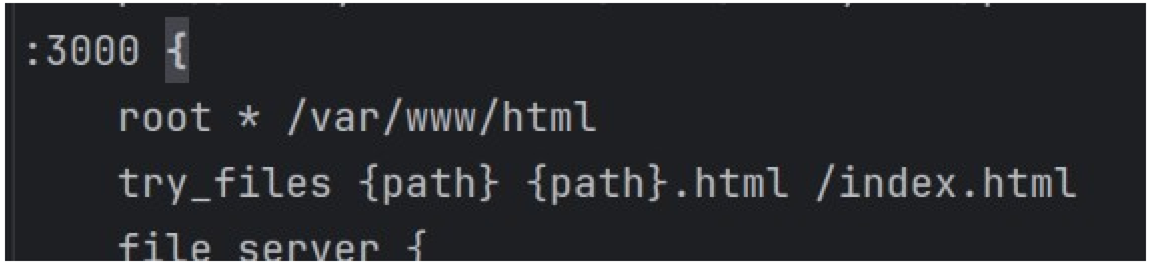
- Interactions occur within a single web page
- Client page dynamically updates as the user interacts with it, providing a smooth, app-like experience
- Relies on JavaScript to update UI
  - Initial request: browser sends a request to receive initial HTML/JS/CSS
  - Initial response: server returns a single HTML file with CSS/JavaScript. JavaScript files contain the application's logic
  - Browser rendering: shows HTML file, typically a spinner, then executes JavaScript

- User interactions: JavaScript manages the UI updates. Application does not require full page reloads. When you click a link in an SPA, instead of making a traditional HTTP request:
  - JavaScript intercepts the click event
  - Prevent default browser navigation
  - Update the URL using the History API
  - Render new content without requesting new HTML document
- API communication: When the SPA needs to fetch or send data, communicates through APIs

Single Page Application SPA / CSR

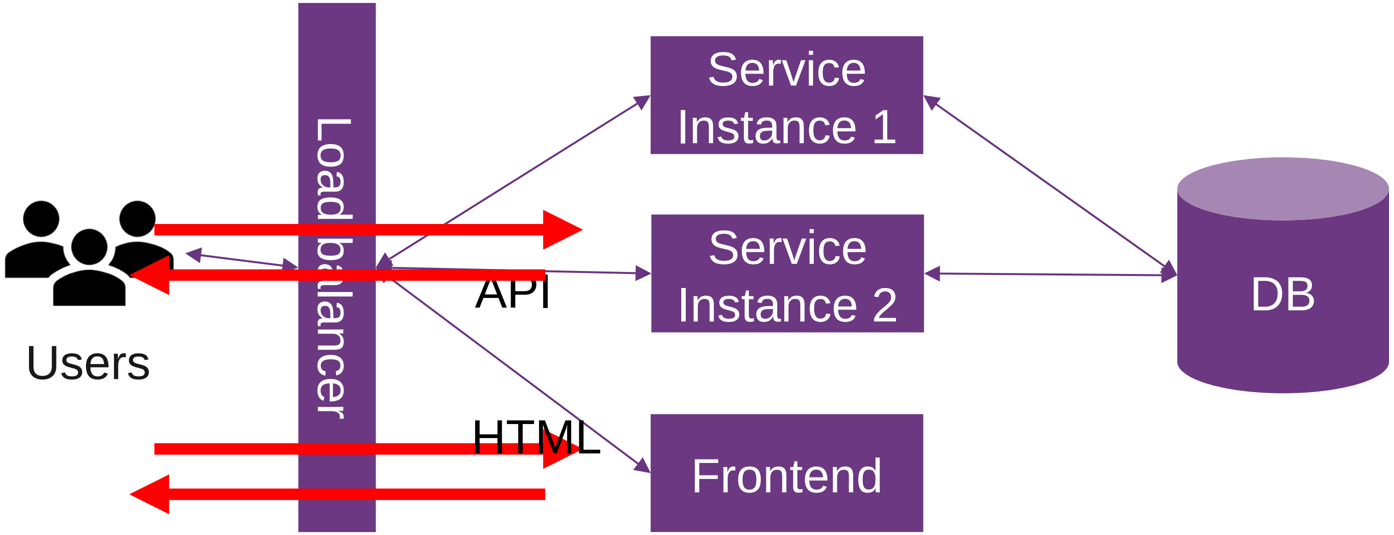
- Use a framework: React, Angular, Vue
- Feels more app like
- The backend serves API requests only
- SEO only works if JavaScript is executed at the SE.
  - Crawler gets JavaScript code, needs to execute, then it knows the content
    - Many corner cases
- Good separation: UI in HTML/CSS/JS, backend in /api

- Client-side routing: SPAs for navigation
  - Server side routing? – default to index.html, as client side routing “inside” index.html



```
:3000 {
    root * /var/www/html
    try_files {path} {path}.html /index.html
    file_server {
```

Simple Example



Users

Load balancer

API

HTML

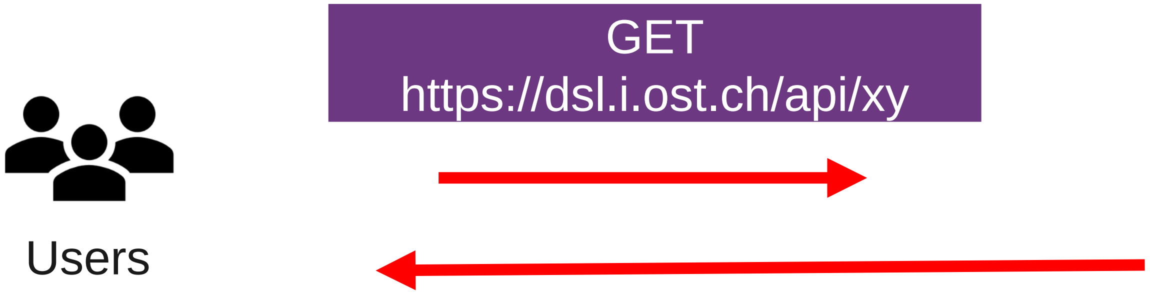
Service Instance 1

Service Instance 2

Frontend

DB

- Initial load: entire page
- Further requests: only updates partially



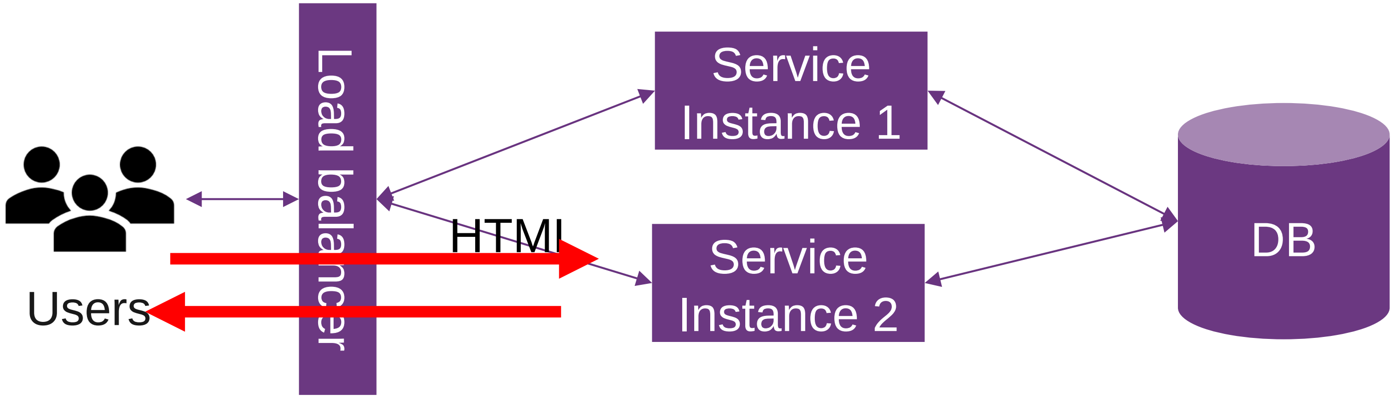
GET https://dsl.i.ost.ch/api/xy

Users

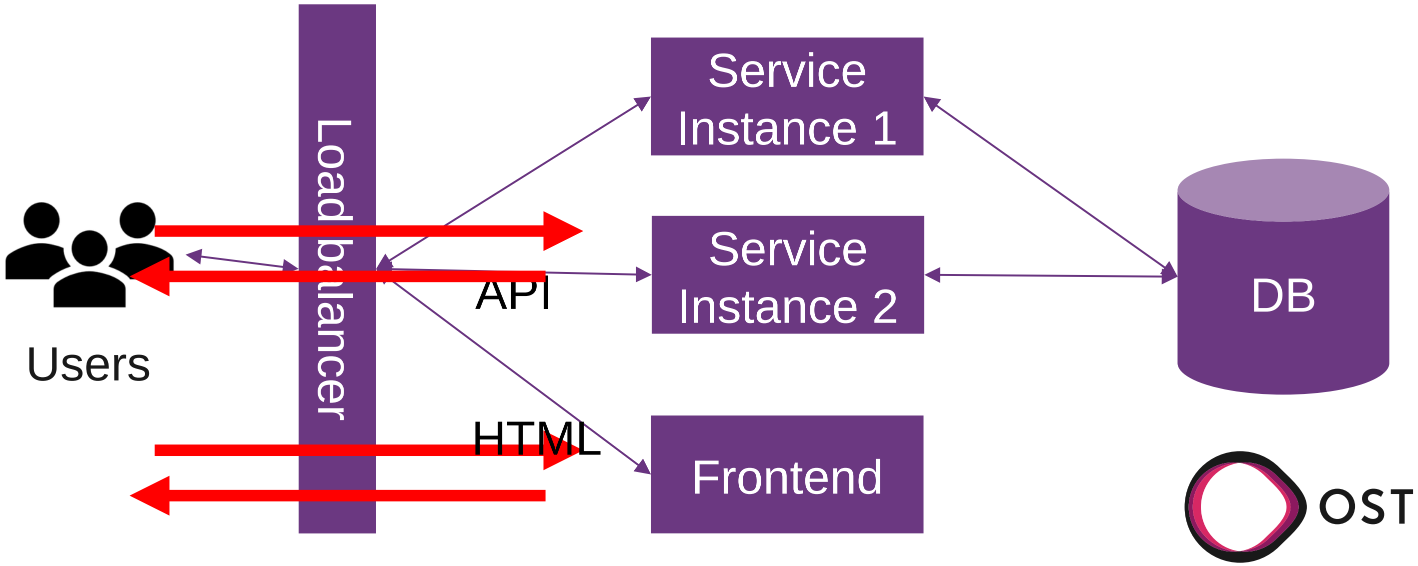
```
{"menu": {
  "id": "file",
  "value": "File",
...
```

Architecture Comparison

- Server side rendering (SSR)



- Single page application (SPA), client side rending (CSR)



CORS

- CORS = Cross-Origin Resource Sharing
  - For security reasons, browsers restrict cross-origin HTTP requests initiated from scripts (among others)
  - Mechanism to instruct browsers that runs a resource from origin A to run resources from origin B
- Solution
  - Use reverse proxy with builtin webserver, e.g., nginx, or user reverse proxy with external webserver.

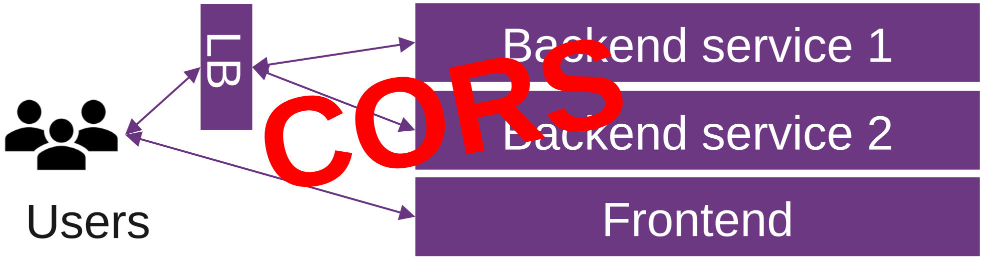
→ The client only sees the same origin for the API and the frontend assets

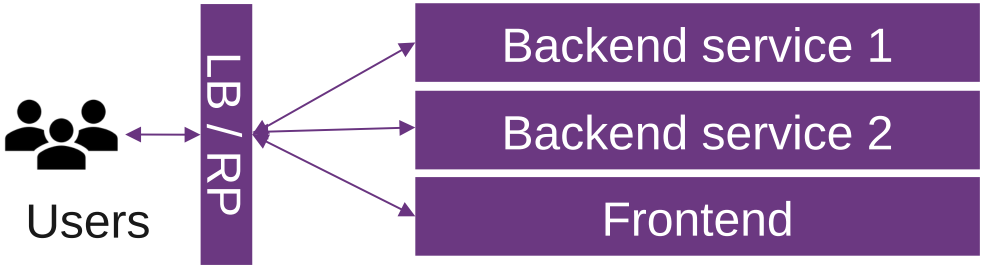
- Access-Control-Allow-Origin: https://foo.example

→ For dev: Access-Control-Allow-Origin: *

- w.Header().Set("Access-Control-Allow-Origin", "*")

- Reverse proxy

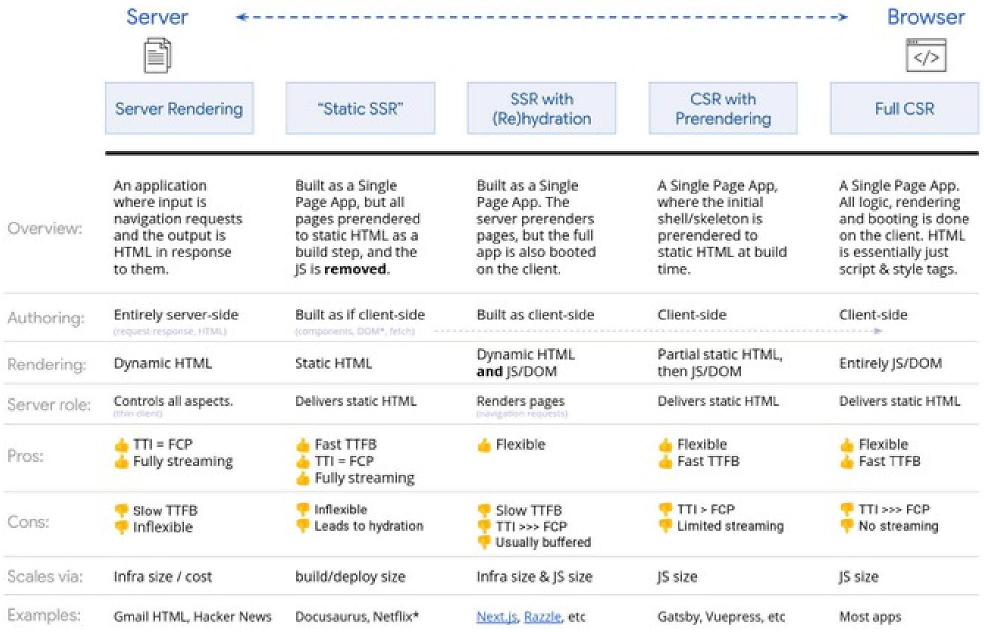




Web Architectures

- SPA: CORS - Cross-Origin Resource Sharing
  - HTTP-header based mechanism to indicate other origins (domain, scheme, or port) from which a browser can load assets.
- “State-of-the-art”: hydration
  - Initial HTML not with a “spinner”, but already the first content in HTML, like SSR (e.g., next.js server renders it for you - JavaScript)
  - Further access, with API, like SPA
  - Combine SSR/SPA
  - PrevelteKit: pre-SSR/SPA
    - Every user sees the same page, SSR can be pre-hydrated

- Hydration
  - Best of both worlds, but adds complexity, needs JavaScript in the backend
  - Overview: source


Server
Browser

	Server Rendering	"Static SSR"	SSR with (Re)hydration	CSR with Prerendering	Full CSR
Overview:	An application where input is navigation requests and the output is HTML in response to them.	Built as a Single Page App, but all pages prerendered to static HTML as a build step, and the JS is **removed**.	Built as a Single Page App. The server prerenders pages, but the full app is also booted on the client.	A Single Page App, where the initial shell/skeleton is prerendered to static HTML at build time.	A Single Page App. All logic, rendering and booting is done on the client. HTML is essentially just script & style tags.
Authoring:	Entirely server-side (request-response, HTML)	Built as if client-side (components, DOM*, fetch)	Built as client-side	Client-side	Client-side
Rendering:	Dynamic HTML	Static HTML	Dynamic HTML **and** JS/DOM	Partial static HTML, then JS/DOM	Entirely JS/DOM
Server role:	Controls all aspects. (thin client)	Delivers static HTML	Renders pages (navigation requests)	Delivers static HTML	Delivers static HTML
Pros:	👍 TTI = FCP 👍 Fully streaming	👍 Fast TTFB 👍 TTI = FCP 👍 Fully streaming	👍 Flexible	👍 Flexible 👍 Fast TTFB	👍 Flexible 👍 Fast TTFB
Cons:	👎 Slow TTFB 👎 Inflexible	👎 Inflexible 👎 Leads to hydration	👎 Slow TTFB 👎 TTI >>> FCP 👎 Usually buffered	👎 TTI > FCP 👎 Limited streaming	👎 TTI >>> FCP 👎 No streaming
Scales via:	Infra size / cost	build/deploy size	Infra size & JS size	JS size	JS size
Examples:	Gmail HTML, Hacker News	Docusaurus, Netflix*	Next.js, Razzle, etc	Gatsby, Vuepress, etc	Most apps

https://web.dev/articles/rendering-on-the-web