

Päivä 4 - Backend

2021-11-15

AaltoPRO - Websovelluskehitys

Sisältö

- Aamupäivä
 - Pieni kertaus & syntaktista sokeria
 - Palvelinsovellukset, polut, metodit, reititys, middleware & MVC
 - (kahvitauko)
 - Harjoitus - Opintopisterekisteri
- Lounas
- Iltapäivä
 - Erilaisia tapoja tuottaa sivu
 - SSG
 - SSR
 - CSR
 - (kahvitauko)
 - Harjoitus - Listoja

Kertausta & Sokeria

Kertaus: URL - Uniform Resource Locator

- Esim.

<https://wsd.cs.aalto.fi:443/1-introduction-and-tooling/4-http-protocol/#http-status-codes?foo=bar>

- Protokolla: https
- Alidomainit: wsd ja cs
- Domain: aalto
- TLD* fi
- Portti: 443
- Polku: /1-introduction-and-tooling/4-http-protocol/
- Ankkuri: #http-status-codes
- Query: ?foo=bar

- [What is a URL? - Learn web development | MDN](#)

* TLD, Top-Level Domain

Kertaus: HTTP - Request Methods

- GET (ei muuta palvelimen tilaa) ja POST (lähettää dataa palvelimelle, tila yleensä muuttuu)
- [GET - HTTP | MDN](#) ja [POST - HTTP | MDN](#)
- [HTTP request methods - HTTP | MDN](#)

JS & Syntakkinen sokeri

[Destructuring assignment - JavaScript | MDN](#)

```
const user = {  
  id: 42,  
  isVerified: true  
};
```

```
const {id, isVerified} = user;
```

```
console.log(id); // 42  
console.log(isVerified); // true
```

Oak & Konteksti

```
router.get("/foo",  async (ctx) => { ctx.response.body = JSON.stringify(ctx) });
```

<https://oakserver.github.io/oak/#context>

```
{
  "app":{
    "proxy":false,
    "state":{}
  },
  "cookies":{},
  "respond":true,
  "request":{},
  "response":{},
  "state":{},
  "matched":[
    {
      "methods":["HEAD","GET"],
      "middleware":[null],
      "paramNames":[],
      "path":"/foo",
      "regex":{},
      "options":{}
    }
  ],
  "router":{},
  "captures":[],
  "params":{}
}
```

Oak ja context destructuring

```
const courses = async ({response}) => {  
  const allCourses = await getCourses();  
  response.body = await renderFile("../views/layouts/courses.eta", {  
    courses: allCourses  
  });  
}
```

```
const courses = async (ctx) => {  
  const allCourses = await getCourses();  
  ctx.response.body = await renderFile("../views/layouts/courses.eta", {  
    courses: allCourses  
  });  
}
```


Asynkroninen ohjelmointi ja lupaukset

- Promise [Promise - JavaScript | MDN](#) [Using Promises - JavaScript | MDN](#)
- Kolme mahdollista tilaa
 - Pending
 - Fulfilled
 - Rejected
- `async & await`
 - `async myFunction() ...` palauttaa lupauksen (promise)
 - `await` odottaa kunnes lupauksen tila ei enää ole pending
 - (toimii ainoastaan `async`-funktioiden sisällä)
 - [Making asynchronous programming easier with async and await | MDN](#)
 - [Async/await](#) (javascript.info)



Client-side JavaScript

How to program and how to use JavaScript to add interaction to web pages?



Server-side runtime environment

How to create server-side programs?
Our runtime is mainly Deno

Web pages

What are web pages made of?
How to add content to a page and how to change its appearance?



Libraries

What different types of libraries exist and what can you create with them?

Requests and responses

What is a server and how pages get transferred between the server and the client (HTTP)?

Routing & Sessions

How to get the correct page or resource from a server and how to store data in the browser?



Testing, deployment, logging, analytics, and so much more...

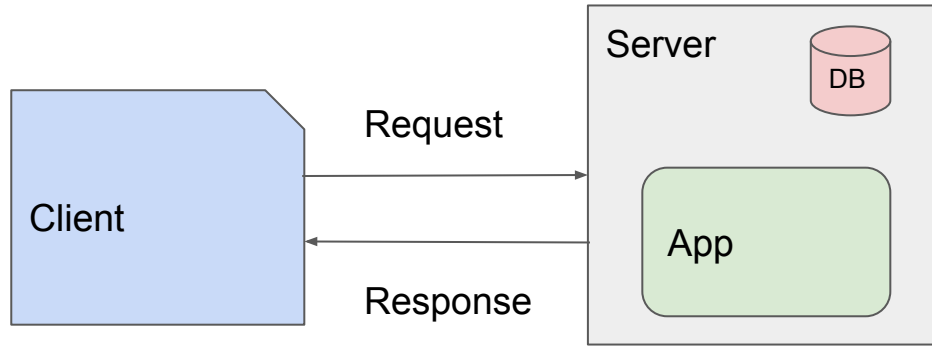
Data

How to retrieve, modify, and store data on the server?

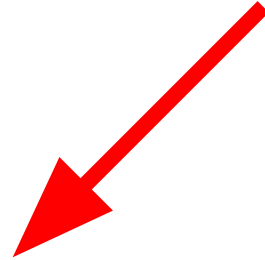
Olet täällä

Front vs. Backend

Pyyntö -> **Käsittely** -> Vastaus



*Tänään vain yksi
"laatikko",
todellisuudessa voi
koostua useista eri
palvelimista ja
palveluista*



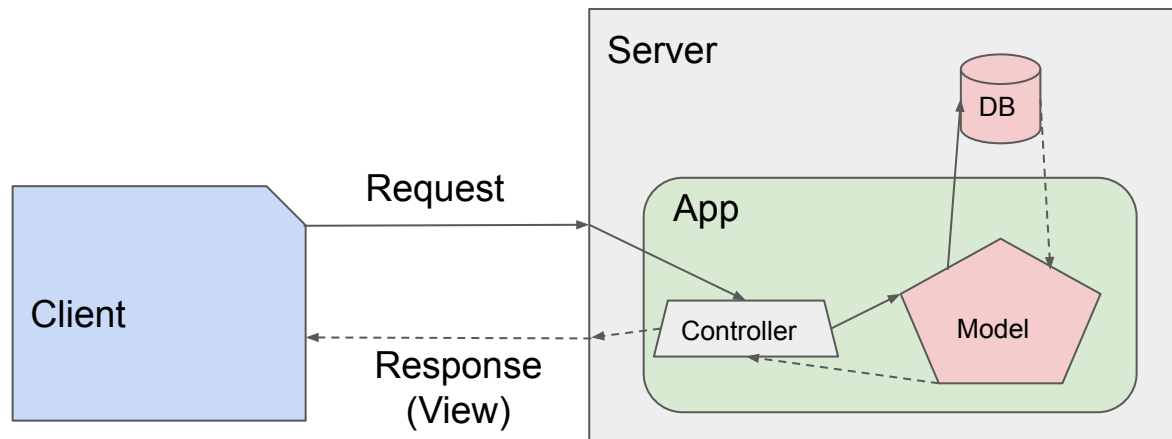
Lähestymisiä palvelinohjelmointiin

Monia tapoja tehdä palvelinohjelmointia

- LAMP (Linux, Apache, MySQL, PHP/Perl/Python) ([Wikipedia](#))
- MEAN (MongoDB, Express.js, AngularJS (or Angular), and Node.js) ([Wikipedia](#))
- Jamstack Javascript, API and Markup ([Wikipedia](#))

MVC - Model-View-Controller lyhyesti

- Malli (Model): ohjelman data & logiikka
- Näkymä (View): Datat esitys
- Kontrolleri/Käsittelijä (Controller): Ottaa käyttäjän syötteen, manipuloi mallia ja palauttaa tietyn näkymän dataan
- Ohjelmistokehykset (frameworks) "pakottavat" koodin rakennetta



selain



Lomake (form -- lomakkeella name ja notes)
→ POST /recs

editor.eta
editor.js

palvelin

`router.post("/recs", recordingController.addRecording);`

routes

`recordingController.addRecording({ request, response })`

controllers

`recordingService.addRecording(name, notes)`

services

`executeQuery`

database

tieto-
kanta



Huom. sokeri

Esim. project-01-piano: POST /recs (lomakkeella: name & notes)

routes.js:

```
router.post("/add", recordingController.addRecording);
```

recordingController.js:

```
const addRecording = async ({ request, response }) => {
  const body = request.body();
  const params = await body.value;
  await recordingService.addRecording(params.get('name'), params.get('notes').trim());
  response.body = await renderFile("../views/recordings.eta", {
    recordings: await recordingService.findRecordings(),
  });
};
```

recordingService.js:

```
const addRecording = async (name, notes) => {
  const res = await executeQuery("INSERT INTO recordings (name, recording) VALUES ($1, $2)", name, notes);
}
const findRecordings = async () => {
  const res = await executeQuery("SELECT * FROM recordings");
  return res.rows;
};
```

recordings.eta:

```
<% layout("../layouts/layout.eta") %>
<ul>
  <% it.recordings.forEach((rec) => { %>
    <li><a href="/recs/<%= rec.id %>"><%= rec.name %> (<%= rec.recording %>)</a></li>
  <% }); %>
</ul>
```

Reititys (Routing)

- Yhdistää käyttäjän pyynnön oikeaan resurssiin

Request Headers

authority: wsd.cs.aalto.fi

method: GET

path: /1-introduction-and-tooling/4-http-protocol/

Request Headers

authority: wsd.cs.aalto.fi

method: GET

path: /favicon.svg

Reititys project-01-piano

project-01-web-piano/piano/routes/routes.js:

```
import { Router } from "../deps.js";
import * as recordingController from "../controllers/recordingController.js";

const router = new Router();

router.get("/", recordingController.viewRecorder);
router.get("/recs", recordingController.listRecordings);
router.get("/recs/:id", recordingController.editRecording);
router.post("/edit", recordingController.editNotes);

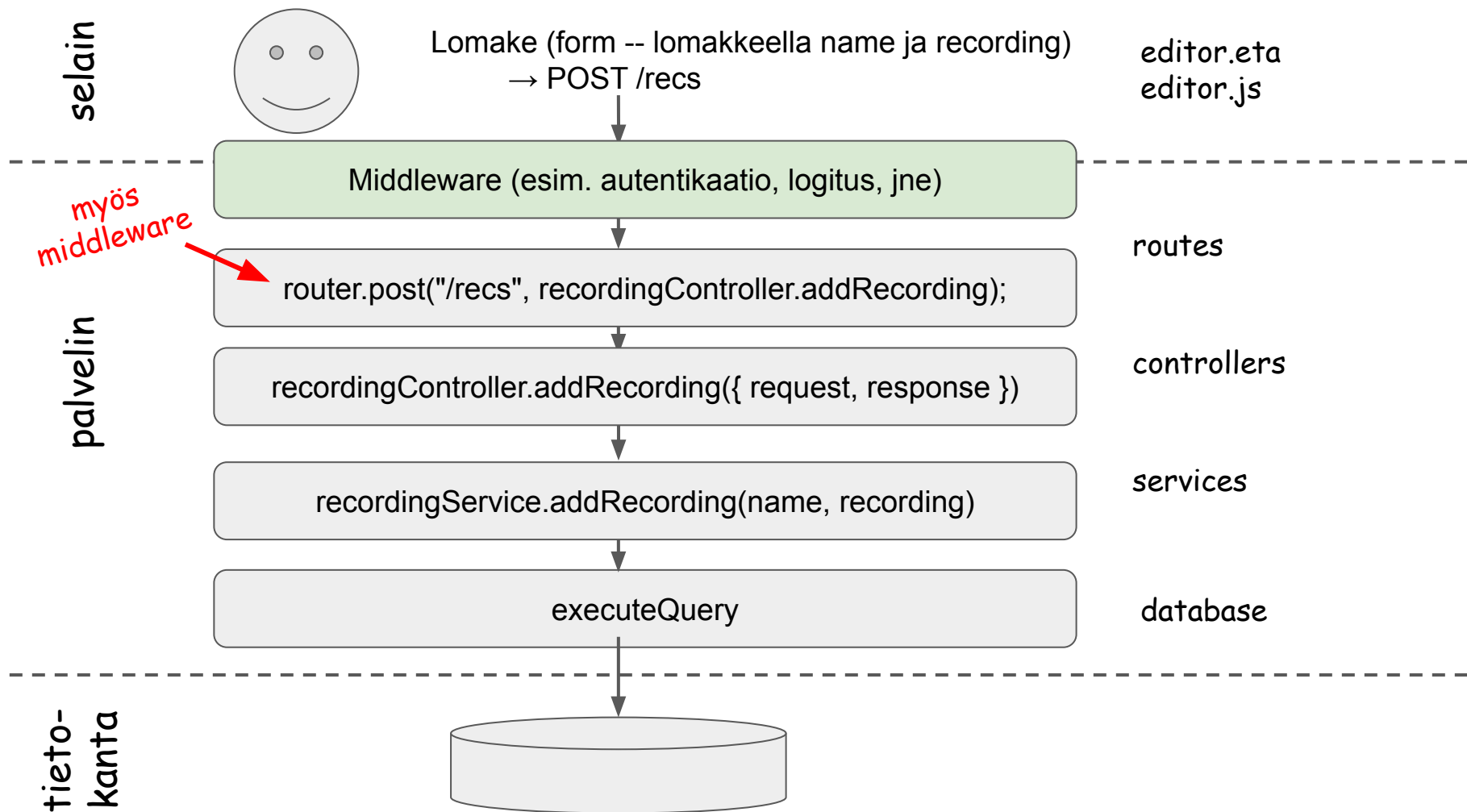
export default router.routes();
```

project-01-web-piano/piano/app.js

```
...
import routes from "../routes/routes.js";
...
app.use(middlewares.errorMiddleware);
app.use(routes);
app.use(middlewares.serveStaticMiddleware);
...
```

Middleware

- Yleisesti: mahdollistaa kommunikoinnin eri ohjelmien (tai ohjelman osien) välillä
- Web-kehityksessä erityisesti yhdistää pyynnön ja palvelimen. Esim.:
 - Logittaa virheitä
 - Tarkistaa onko käyttäjä kirjautunut sisään
 - Tarjoaa staattista sisältöä
 - .. käytännössä muokkaa jokaista pyyntöä tai käsittelee niitä jollain tavalla
- Deno & Oak <https://github.com/oakserver/oak>



Middleware esimerkki World explorer

```
import { createPlayer } from "../services/playerService.js";

const errorMiddleware = async (context, next) => {
  try { await next(); } catch (e) { console.log(e); }
};

const serveStaticMiddleware = async (context) => {
  await context.send({ root: `${Deno.cwd()}/static`, });
};

const sessionMiddleware = async (context, next) => {
  try {
    let playerId = await context.cookies.get("playerId");
    if (!playerId) {
      playerId = await createPlayer();
      await context.cookies.set("playerId", playerId);
    }
    context.playerId = playerId;
    await next();
  } catch (e) { console.log(e); }
};

export { errorMiddleware, serveStaticMiddleware, sessionMiddleware };
```

MVC kahdessa muussa sovelluskehyksessä (Django & Spring)

- Django on suosittu sovelluskehys Pythonille web-palveluiden luomiseen
 - <https://www.djangoproject.com/>
 - Esimerkkisovellus: <https://github.com/divio/django-polls>
 - Tutoriaali: <https://docs.djangoproject.com/en/3.2/intro/tutorial01/>
- Spring on suosituin(?) web-kehitykseen Javalla
 - <https://spring.io/>
 - Tutoriaali/esimerkkisovellus: <https://spring.io/guides/gs/serving-web-content/>
- Käytännössä jokaiselle ohjelmointikielelle löytyy useita vaihtoehtoja

Hands-on - Kurssin ja opiskelijan näkymät

- <https://github.com/aaltopro-weblearners/project-04a-courses>
- Implementoi näkymät opiskelijoille
 - /opiskelijat/ -> Näyttää opiskelijanumeron, nimen ja linkkaa opiskelijan kurssisuorituksiin
 - /opiskelijat/:id - Näyttää:
 - opiskelijan suoritettut kurssit
 - kurssien arvosanan keskiarvon
 - suoritettujen opintopisteiden määrän
- Implementoi näkymä kurssille
 - /kurssit/:id - Näyttää:
 - kurssin suorittaneet opiskelijat
 - keskiarvon kurssisuorituksista
 - annettujen opintopisteiden kokonaismäärän

Vinkkejä

- Kannattaa aloittaa `/opiskelijat/` tai `/kurssit/` näkymästä.
Tietokantakyselyt on toteutettu valmiiksi ja tiedot tarvitsee vain esittää HTML-muodossa
 - Tiedot tulevat templatelle (`courses.eta` ja `students.eta`), jolloin ne tarvitsee vain esittää taulukkomuodossa, ks. Esim [HTML table basics - Learn web development | MDN](#)
- `/opiskelijat/:id` ja `/kurssit/:id` näkymät kannattaa aloittaa toteuttamalla `servicet`, jolla saa tarvittavat tiedot
 - Ks. `/services/creditService.js` ja `getCredits()`, josta löytyy esimerkki kuinka tietoja haetaan kolmesta taulusta
 - Kannattaa ensin saada tiedot listattuna näkyvään, jonka jälkeen toteuttaa keskiarvojen/opintopisteiden laskenta

Huomioita courses-sovelluksesta

- Reititys
- Keskiarvojen laskeminen

Lounas

Staattiset HTML-sivut

1. Selain pyytää sivun "GET /index.html"
2. Palvelin palauttaa sivun

"200 content-type: text/html; charset=UTF-8 [index.html]"

3. Selain näyttää sivun
 - Yksinkertainen tapa näyttää sivuja selaimessa
 - Kaikki mahdolliset sivut täytyy luoda etukäteen
 - Usein käytetään staattisia sivu-generaattoreja (Static Site Generator, SSG)
 - Esim. [Gatsby](#) ([Static Site Generator](#))

Palvelimella muodostetut sivut



1. Selain pyytää sivun "GET /index.html"
2. Palvelin käsittelee pyynnön (esim. hakee tietokannasta tarvittavat tiedot & muokkaa niitä)
3. Palvelin sisällyttää haetut tiedot HTML-pohjaan (template) ja palauttaa sen
4. Selain näyttää sivun
 - Huom. sivujen pyytäminen saattaa tuottaa erilaisia sisältöjä eli sivu voi olla *dynaaminen*
 - Sivut "rendataan" palvelimella valmiiksi (Server-side rendering, SSR)
 - Esim. aamupäivän harjoitus

Selaimessa muodostetut sivut

1. Selain pyytää sivun "GET /index.html"
2. Palvelin palauttaa sivun, jossa on lisäresursseja
 - a. `<script src="/js/myApp.js"></script>`
 - b. ... joka puolestaan voi pyytää lisäresursseja, esim. `content.json`
3. Selain näyttää aluksi esim. tyhjän sivun ja lataa tarvittavat JS-kirjastot ja sisällöt
 - a. "GET /js/myApp.js" & "GET /content.json"
4. Selain käsittelee myApp.js kirjastolla tiedot, jotka noudetaan content.jsonista ja lisää elementtejä/muokkaa dokumenttia näiden mukaan
 - Sivut muodostetaan selaimessa (Client-side rendering, CSR)

Lisälukemista

- <https://nikolovlazar.com/how-to-render-your-website>
- Kuvat:
 - [SSG](#)
 - [SSR](#)
 - [CSR](#)
- <https://developers.google.com/web/updates/2019/02/rendering-on-the-web>

	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 <small>(request-response, HTML)</small>	Built as if client-side <small>(components, DOM*, fetch)</small>	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. <small>(thin client)</small>	Delivers static HTML	Renders pages <small>(navigation requests)</small>	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

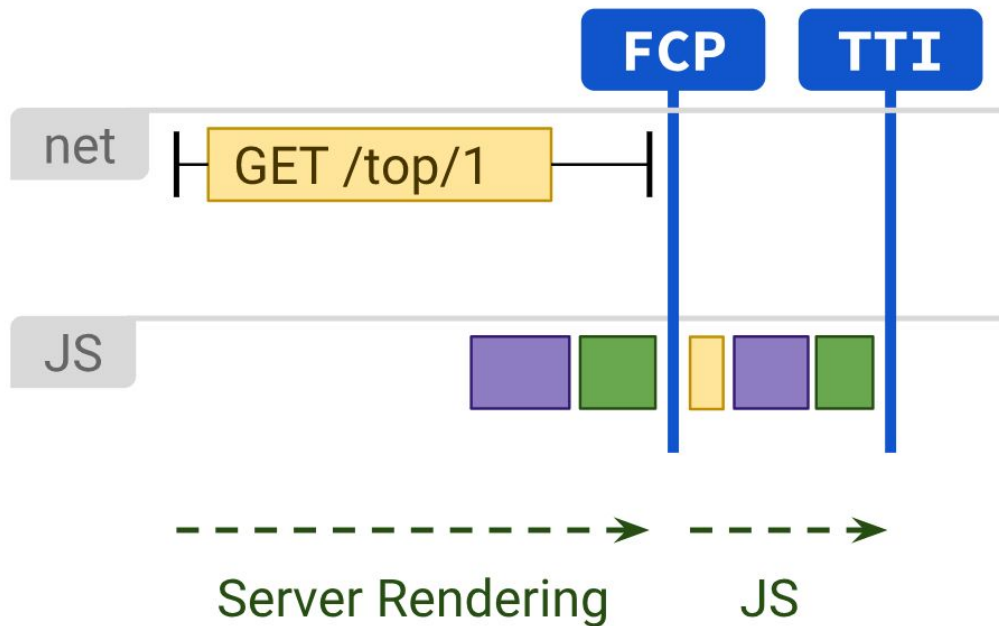
SSR

TTFB: Time to First Byte - seen as the time between clicking a link and the first bit of content coming in.

FP: First Paint - the first time any pixel gets becomes visible to the user.

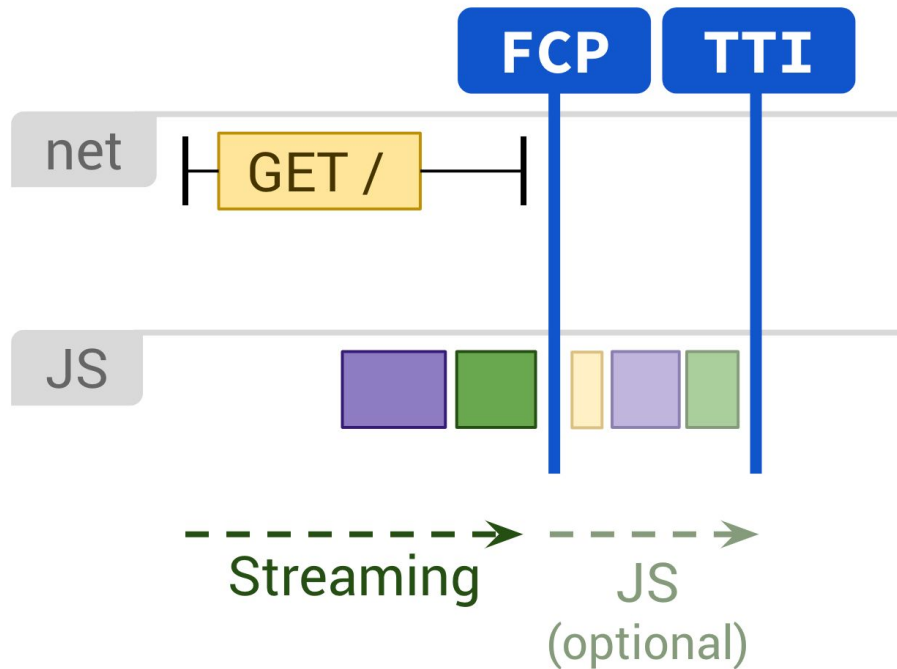
FCP: First Contentful Paint - the time when requested content (article body, etc) becomes visible.

TTI: Time To Interactive - the time at which a page becomes interactive (events wired up, etc).

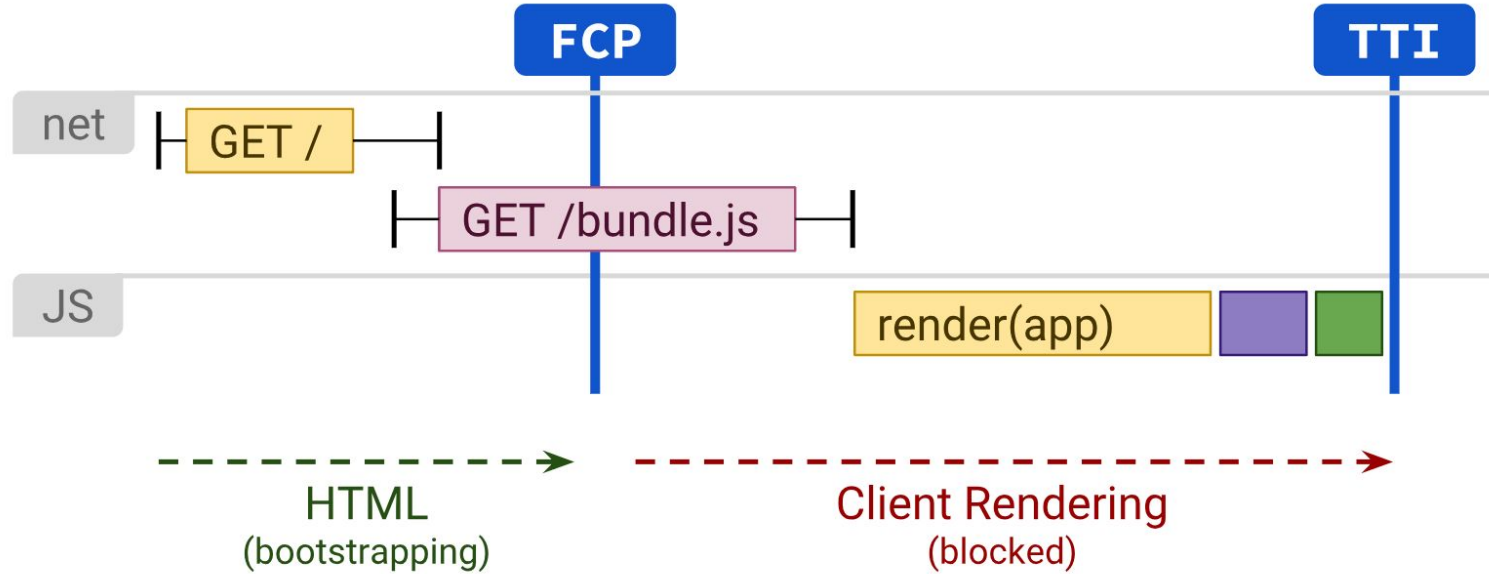




SSG

- Content Delivery Network, CDN



CSR



	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 <small>(request-response, HTML)</small>	Built as if client-side <small>(components, DOM*, fetch)</small>	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. <small>(thin client)</small>	Delivers static HTML	Renders pages <small>(navigation requests)</small>	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

Vähän API-asiaa

- Usein data, jota CSR tarvitsee tulee rajapinnasta (API)
- Monesti muotoa `/api/[resurssi]/[id]`
- Lisää rajapinnoista ja niiden rakenteesta jatkossa

- Esim. `/api/list-data/1`

```
[
  {
    "id":1,
    "checked":false,
    "list_id":1,
    "item_name":"Coffee"
  },
  {
    "id":3,
    "checked":false,
    "list_id":1,
    "item_name":"Honey"
  },
  {
    "id":2,
    "checked":false,
    "list_id":1,
    "item_name":"Tea"
  }
]
```

Hands-on - Listat

- <https://github.com/aaltopro-weblearners/project-04b-list-keeper>
- Implementoi puuttuvat controllerit ja servicet. Nimet voi päättää itse (oranssilla ehdotuksia)
- Templatet (/layout/*.eta) annettu valmiina, katso oletetut tiedot templatesta
- /static/js/list.js olettaa saavansa edellisen kalvon mukaiset tiedot
/api/lista-data/:id rajapinnasta

Palauttaa HTML-sivun:

```
router.get("/users/:id", showUserLists );  
router.get("/users/:userId/lists/:listId", showList );
```

```
router.post("/item/new/", addItem);  
router.post("/list/new/", addList);
```

Palauttaa JSONin:

```
router.get("/api/list-data/:id", getListData);  
router.post("/api/update-item/:listId/:itemId", checkListItem);
```

Yhteenvetoa

- MVC
- Erilaisia tapoja tuottaa sivuja
 - Static site generation (SSG)
 - Server-side rendering (SSR)
 - Client-side rendering (CSR)
 - ... ja erilaiset yhdistelmät