

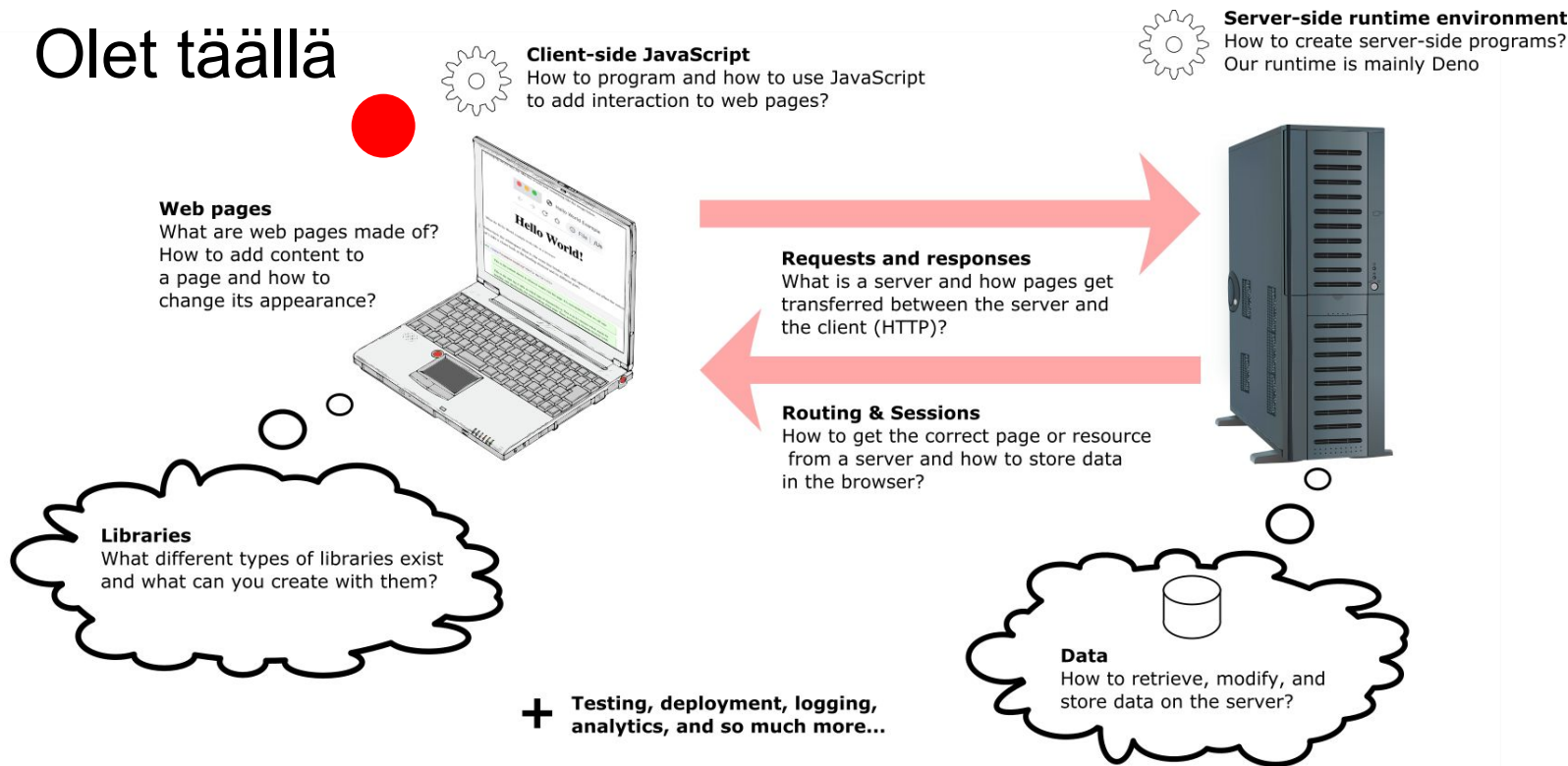
Päivä 11 - Selaimessa toimivat sovellukset

2022-02-04

AaltoPRO - Websovelluskehitys

Web-sovellukset korkealla tasolla

Olet täällä



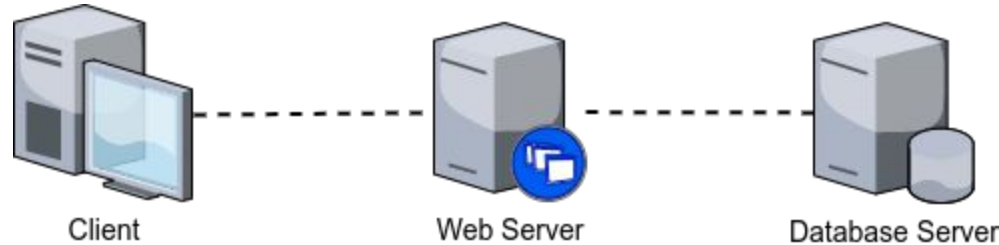
Päivä 11

- Pientä kertausta
- Komponentit
- Tilan hallinta
- Tunnistautuminen
- Testausta

Pientä kertautta

Klassiset websovellukset:

Palvelin luo HTML-sivun (1) pyynnön, (2) tietokannassa olevan tiedon, ja (3) pyyntöön yhdistettävän näkymätemplatien pohjalta. Tämä näytetään selaimessa käyttäjälle.

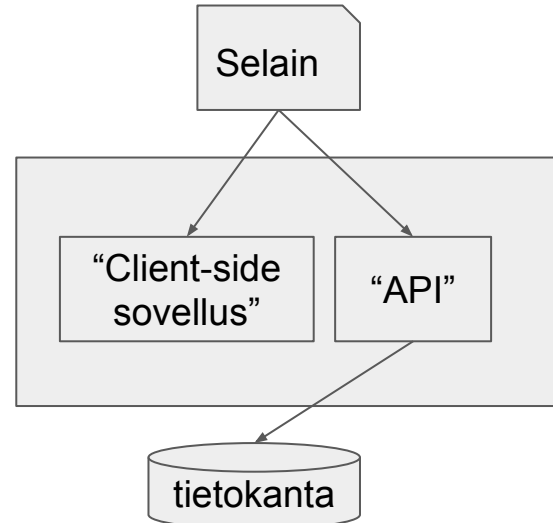


Selainpohjaiset websovellukset:

Selain hakee ensin HTML-dokumentin sekä siihen liittyvät JavaScript-tiedostot, joilla määritellään sovelluksen (alustava) toiminnallisuus. Palvelin tarjoaa rajapinnat, joista selain hakee (esim. JSON-muotoista) dataa. Selain tekee pyyntöjä rajapintoihin ja muokkaa näkymää rajapinnoista saatavan datan sekä käyttäjän toimien perusteella.

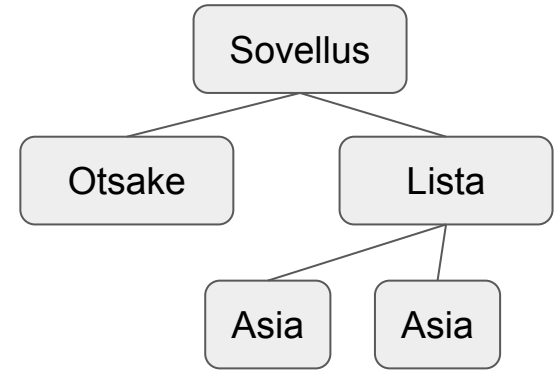
Selainpohjaiset sovellukset käytännössä

- Sovelluksessa kaksi palvelinta / palvelua, toisen vastuulla selainpuolen toiminnallisuus, toinen tarjoaa ohjelmointirajapinnan
- Näiden lisäksi käytössä tietokanta
- Palveluilla yhä oma looginen rakenteensa



Yleisiä ominaisuuksia

- Komponentit
 - Toiminnallisuus koostetaan komponenteista. Esim. otsakekomponentti, listakomponentti, nappikomponentti ...
- Tilan hallinta
 - Sovelluksella on tila (=tietoa, joka määrää näytettävän osan). Tilaa hallitaan selaimessa.
- Komponenttien piirtäminen tilan perusteella
- Kommunikaatio palvelimen kanssa



Komponentit

- Sovellukset koostetaan komponenteista. Jokaisella komponentilla on tehtävä, jonka se ratkaisee – tehtävä voidaan ratkaista myös yhteistyössä muiden komponenttien kanssa.
- Komponentissa: (1) JavaScript-koodia, (2) HTML:ää, (3) Tyylimäärittelyjä
 - Mutta! Kaikkea ei ole pakko olla.
- Komponentit sijaitsevat selaintoiminnallisuutta määrittelevän projektin src-kansion sisällä olevassa components-kansiossa.

Jatketaan TODO-sovelluksen parissa

- Pohjaksi joko oma aiempi sovellus tai sovellus osoitteesta:

<https://github.com/aaltopro-weblearners/project-10-todos-8-hours-later>

Komponentit – luonti ja käyttöönotto

- Vain HTMLää sisältävä Header.svelte, polku src/components/Header.svelte

```
<h1>Hei maailma!</h1>
```

- Tuodaan osaksi App.svelte-komponenttia

```
<script>  
  import Header from "@components/Header.svelte";  
</script>  
  
<Header />
```

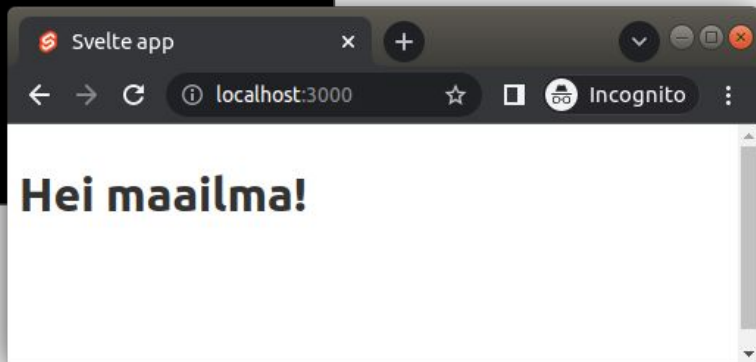
Komponentit – luonti ja käyttöönotto

- Vain HTMLää sisältävä Header.svelte, polku src/components/Header.svelte

```
<h1>Hei maailma!</h1>
```

- Tuodaan osaksi App.svelte-komponenttia

```
<script>  
  import Header from "@components/Header.svelte";  
</script>  
  
<Header />
```



Tietoa komponentin ulkopuolelta (props)

- Komponentille voidaan antaa tietoa komponentin ulkopuolelta
 - Komponentissa määritellään ulkopuolisen muuttujan nimi, esim. `export let viesti`
 - Komponenttia luodessa annetaan muuttujalle arvo, esim. `<Header viesti="Terve!" />`

- `src/components/Header.svelte`:

```
<script>  
  export let viesti;  
</script>  
  
<h1>{viesti}</h1>
```

- `src/App.svelte`:

```
<script>  
  import Header from "@components/Header.svelte";  
</script>  
  
<Header viesti="Terve!" />
```

Tietoa komponentin ulkopuolelta (props)

- Komponentille voidaan antaa tietoa komponentin ulkopuolelta
 - Komponentissa määritellään ulkopuolisen muuttujan nimi, esim. `export let viesti`
 - Komponenttia luodessa annetaan muuttujalle arvo, esim. `<Header viesti="Terve!" />`

- `src/components/Header.svelte:`

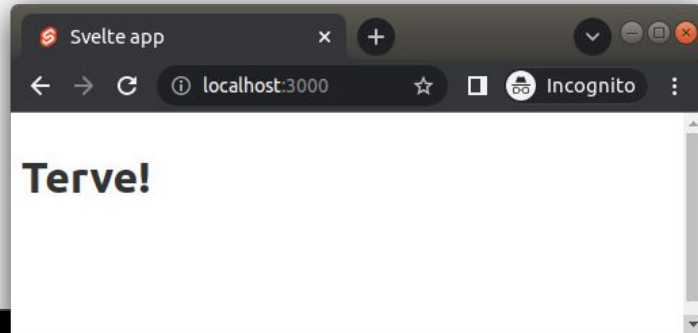
```
<script>
  export let viesti;
</script>

<h1>{viesti}</h1>
```

- `src/App.svelte:`

```
<script>
  import Header from "@components/Header.svelte";
</script>

<Header viesti="Terve!" />
```



Tietoa komponentin ulkopuolelta (props)

- Annettava tieto voi olla myös muuttuja (käytössä edellisen esimerkin Header.svelte)
- src/App.svelte:

```
<script>
import Header from "@components/Header.svelte";
let viesti = "Tänään ";
if (Math.random() < 0.5) {
  viesti += "sataa lunta.";
} else {
  viesti += "paistaa aurinko.";
}
</script>

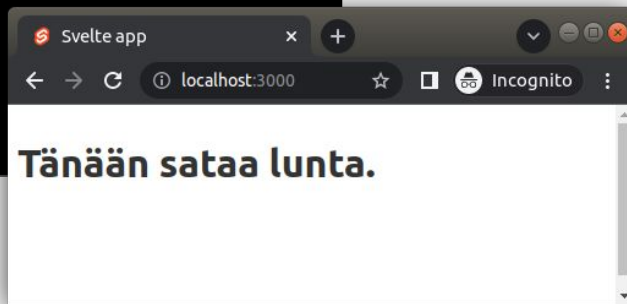
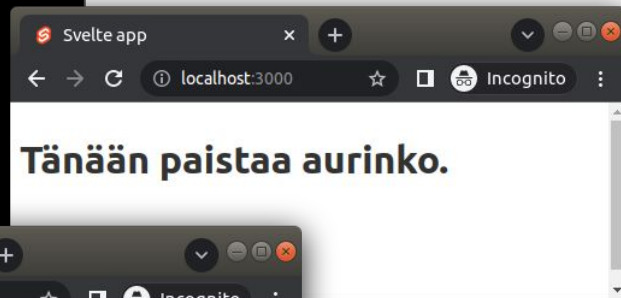
<Header viesti={viesti} />
```

Tietoa komponentin ulkopuolelta (props)

- Annettava tieto voi olla myös muuttuja (käytössä edellisen esimerkin Header.svelte)
- src/App.svelte:

```
<script>
import Header from "@components/Header.svelte";
let viesti = "Tänään ";
if (Math.random() < 0.5) {
  viesti += "sataa lunta.";
} else {
  viesti += "paistaa aurinko.";
}
</script>

<Header viesti={viesti} />
```



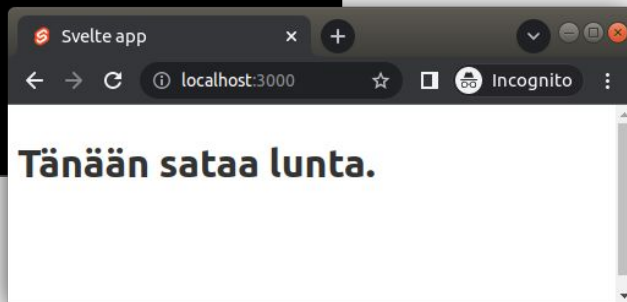
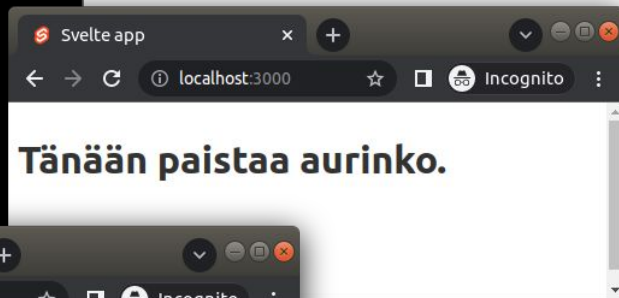
Tietoa komponentin ulkopuolelta (props)

- Annettava tieto voi olla myös muuttuja (käytössä edellisen esimerkin Header.svelte)
- src/App.svelte:

```
<script>
import Header from "@components/Header.svelte";
let viesti = "Tänään ";
if (Math.random() < 0.5) {
  viesti += "sataa lunta.";
} else {
  viesti += "paistaa aurinko.";
}
</script>

<Header viesti={viesti} />
```

*Muuttujan nimi
ei oleellinen!*



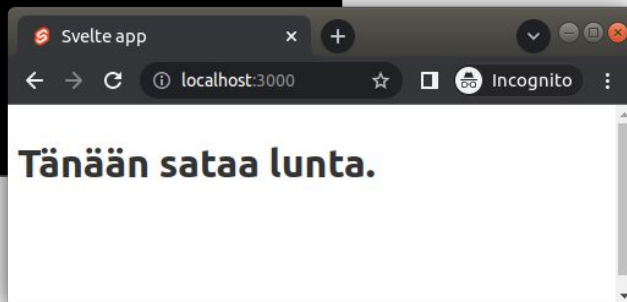
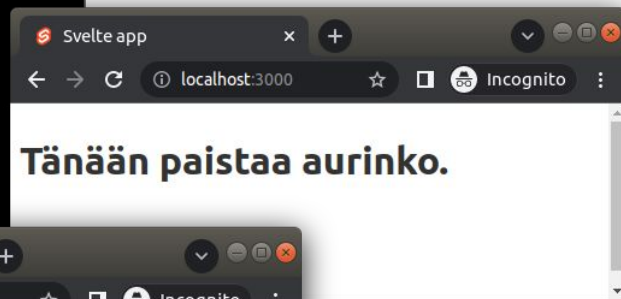
Tietoa komponentin ulkopuolelta (props)

- Annettava tieto voi olla myös muuttuja (käytössä edellisen esimerkin Header.svelte)
- src/App.svelte:

```
<script>
import Header from "@components/Header.svelte";
let msg = "Tänään ";
if (Math.random() < 0.5) {
  msg += "sataa lunta.";
} else {
  msg += "paistaa aurinko.";
}
</script>

<Header viesti={msg} />
```

*Muuttujan nimi
ei oleellinen!*



Tehtävä: Header-komponentin muokkaus

- Muokkaa sovelluksessa olevaa Header-komponenttia siten, että komponentti näyttää satunnaisesti valitun mietelauseen (kahdesta vaihtoehdosta).
- Kun tämä toimii, muokkaa komponenttia siten, että mietelause haetaan palvelimelta.
 - Luo palvelimelle polku mietelauseen hakemista varten, palauta palvelimelta JSON-muotoinen dokumentti, jossa on mietelause.
 - Lisää komponenttiin fetch-kutsu, joka tekee pyynnön palvelimelle määriteltyyn polkuun.
 - Hyödynnä fetch-kutsun vastauksena saatua dokumenttia ja näytä käyttäjälle mietelause dokumentista.

Komponentin tila

- Komponentin tilalla tarkoitetaan komponentin muuttujien arvoja sekä toisaalta niiden vaikutusta näytettyyn tietoon. Alla esimerkki komponentista, jolla on tila.
- src/components/Counter.svelte:

```
<script>
  let count = 0;
  const up = () => {
    count++;
  }
</script>

<h1>{count}</h1>

<button on:click={up}>Up</button>
```

Komponentin tila

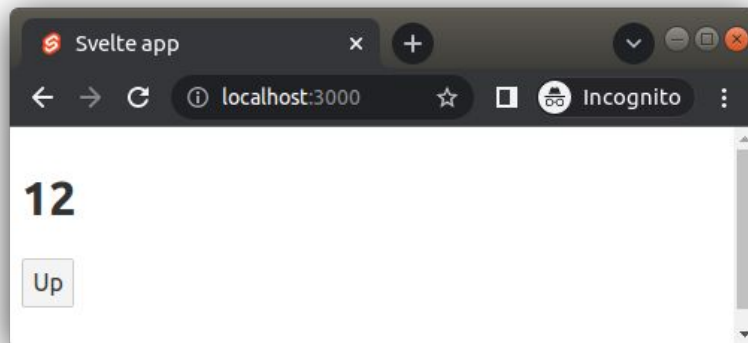
- Komponentin tilalla tarkoitetaan komponentin muuttujien arvoja sekä toisaalta niiden vaikutusta näytettyyn tietoon. Alla esimerkki komponentista, jolla on tila.
- `src/components/Counter.svelte`:

```
<script>
  let count = 0;
  const up = () => {
    count++;
  }
</script>

<h1>{count}</h1>

<button on:click={up}>Up</button>
```

Sovellukseen tuotuna
(import ...)



Tehtävä: Prokrastinaation määrä

- Lisää sovellukseen komponentti ProcrastinationCounter, joka pitää kirjaa tehtävien tekemisen välttelystä.
- Komponentin tulee sisältää teksti “Välttelyn määrä”, välttelyn määrää kuvaava luku ja nappi. Nappia painamalla luvun tulee kasvaa yhdellä.

Tehtävä: Prokrastinaation määrä

- Lisää sovellukseen komponentti ProcrastinationCounter, joka pitää kirjaa tehtävien tekemisen välttelystä.
- Komponentin tulee sisältää teksti “Välttelyn määrä”, välttelyn määrää kuvaava luku ja nappi. Nappia painamalla luvun tulee kasvaa yhdellä.

Tämä ei ota kantaa siihen, onko
kyseessä positiivinen vai
negatiivinen käyttäytymismalli..

Keskitetty komponentin tila

Keskitetty komponentin tila

- Mikäli komponentin tila on määritelty komponentin sisällä, tila vaikuttaa vain kyseiseen komponenttiin ja sen sisältämiin komponentteihin (tilan voi antaa komponentille kuten otsakkeen viestiä määriteltäessä).
- Svelte tarjoaa mahdollisuuden myös keskitettyyn tilan hallintaan. Tällöin käytetään varastoja "store". Luodaan kansioon `src` kansio `stores`, ja luodaan sen sisälle tiedosto `countStore.js`
- `src/stores/countStore.js`-tiedoston sisällöksi:

```
import { writable } from "svelte/store";  
  
const count = writable(0);  
  
export default count;
```

Varastotoiminnallisuus

Uuden varaston
luominen

Varaston jakaminen
muiden käyttöön

Keskitetty komponentin tila

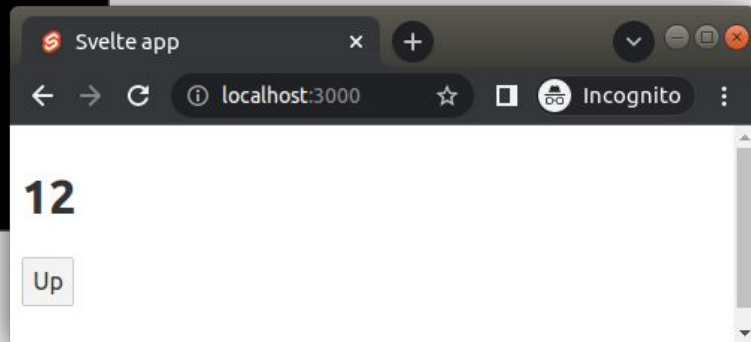
- Varastoon luodun muuttujan voi ladata käyttöön import-komennolla. Etumerkki \$ viittaa varastomuuttujaan; esim. Counter.svelte:
- src/components/Counter.svelte:

```
<script>
  import count from "@stores/countStore.js";
  const up = () => {
    $count++;
  }
</script>

<h1>{$count}</h1>

<button on:click={up}>Up</button>
```

Sovellukseen tuotuna
(import ...)



Keskitetty komponentin tila

- Varastoon luodun muuttujan voi ladata käyttöön import-komennolla. Etumerkki \$ viittaa varastomuuttujaan; esim. Counter.svelte:
- src/components/Counter.svelte:

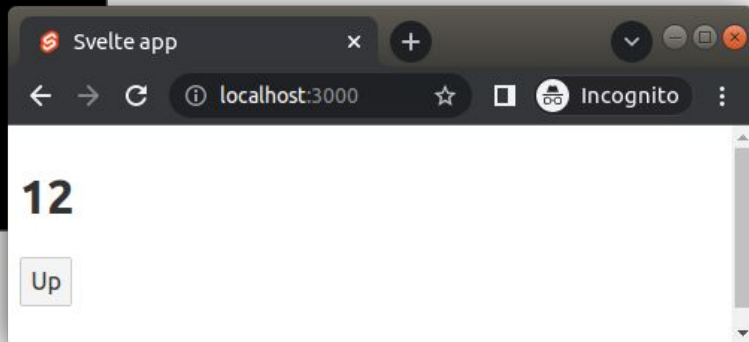
```
<script>
  import count from "@stores/countStore.js";
  const up = () => {
    $count++;
  }
</script>

<h1>{$count}</h1>

<button on:click={up}>Up</button>
```

Sovellukseen tuotuna
(import ...)

Huom \$



Keskitetty komponentin tila

- Keskitetyn tilan hyödyt tulevat näkyville siinä, että keskitettyyn tilaan tehdyt muutokset näkyvät kaikkialla missä tilaa käytetään.
- Luodaan komponentti DoubleCounter.svelte, joka näyttää laskurin tuplana.
- src/components/DoubleCounter.svelte:

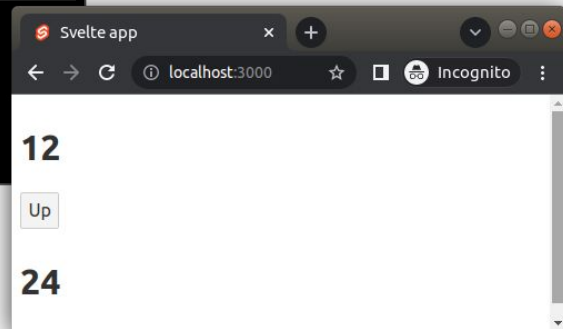
```
<script>  
  import count from "@stores/countStore.js";  
</script>  
  
<h1>{$count * 2}</h1>
```

Keskitetty komponentin tila

- Keskitetyn tilan hyödyt tulevat näkyville siinä, että keskitettyyn tilaan tehdyt muutokset näkyvät kaikkialla missä tilaa käytetään.
- Luodaan komponentti DoubleCounter.svelte, joka näyttää laskurin tuplana.
- src/components/DoubleCounter.svelte:

```
<script>  
  import count from "@stores/countStore.js";  
</script>  
  
<h1>{$count * 2}</h1>
```

Sovellukseen tuotuna
(import ...)



Keskitetty komponentin tila

- Edellisessä esimerkissä keskitetty tila sisälsi luvun. Keskitettyyn tilaan voi asettaa muitakin arvoja.
- Luodaan varasto todoStore.js, joka hallinnoi listalla olevia tehtäviä.
- src/stores/todoStore.svelte:

```
import { writable } from "svelte/store";  
  
const todos = writable([]);  
  
export default todos;
```

Keskitetty komponentin tila

- Luodaan seuraavaksi komponentti MoreTodos.svelte, joka listaa tehtävät.
- src/components/MoreTodos.svelte:

```
<script>  
  import todos from "@stores/todoStore.js";  
</script>  
  
<h1>Todo List</h1>  
  
<ul>  
  {#each $todos as todo}  
    <li>{todo.description}</li>  
  {/each}  
</ul>
```

Keskitetty komponentin tila

- Luodaan seuraavaksi komponentti `TodoForm.svelte`, joka mahdollistaa tehtävän lisäämisen. Huomaa, että komponentti on erillinen `MoreTodos`-komponentista.
- `src/components/TodoForm.svelte`:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos.push({description: todoText});
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

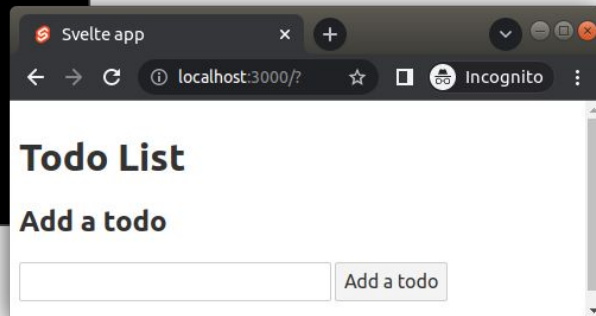
Keskitetty komponentin tila

- Luodaan seuraavaksi komponentti `TodoForm.svelte`, joka mahdollistaa tehtävän lisäämisen. Huomaa, että komponentti on erillinen `MoreTodos`-komponentista.
- `src/components/TodoForm.svelte`:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos.push({description: todoText});
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```



Keskitetty komponentin tila

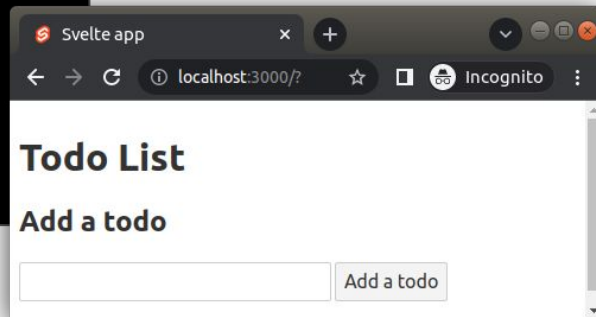
- Luodaan seuraavaksi komponentti `TodoForm.svelte`, joka mahdollistaa tehtävän lisäämisen. Huomaa, että komponentti on erillinen `MoreTodos`-komponentista.
- `src/components/TodoForm.svelte`:

```
<script>
import todos from "@stores/todoStore.js";

const addTodo = () => {
  const todoText = document.getElementById("todo").value;
  $todos.push({description: todoText});
  document.getElementById("todo").value = "";
}
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

Napin painaminen ei lisää
tehtävää listalle?



Keskitetty komponentin tila

- Luodaan seuraavaksi komponentti `TodoForm.svelte`, joka mahdollistaa tehtävän lisäämisen. Huomaa, että komponentti on erillinen `MoreTodos`-komponentista.
- `src/components/TodoForm.svelte`:

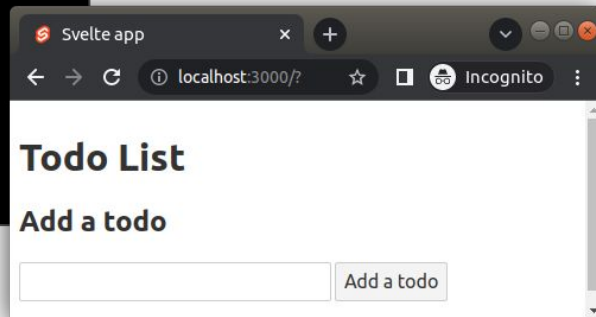
```
<script>
import todos from "@stores/todoStore.js";

const addTodo = () => {
  const todoText = document.getElementById("todo").value;
  $todos.push({description: todoText});
  document.getElementById("todo").value = "";
}
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

*Napin painaminen ei lisää
tehtävää listalle?*

*...kyllä lisää, mutta Svelte ei
tiedä että sovellusta pitäisi
päivittää*



Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos.push({description: todoText});
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos.push({description: todoText});
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos.push({description: todoText});
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

*Muutetaan asetusta
siten, että muuttujaan
asetetaan uusi arvo.*

Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos = [...$todos, {description: todoText}];
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

*Muutetaan asetusta
siten, että muuttujaan
asetetaan uusi arvo.*

Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

```
<script>
  import todos from "@stores/todoStore.js";

  const addTodo = () => {
    const todoText = document.getElementById("todo").value;
    $todos = [...$todos, {description: todoText}];
    document.getElementById("todo").value = "";
  }
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

Muutetaan asetusta siten, että muuttujaan asetetaan uusi arvo.

"Luo listasta kopio, ja aseta viimeiseksi arvoksi uusi arvo."

Keskitetty komponentin tila

- Keskitettyä tilaa käyttävä komponentti päivitetään, kun keskitetyn tilan arvo muuttuu. Listojen ym. tarjoamien metodien kutsuja ei kuitenkaan kuunnella – tarkastelussa vain konkreettiset arvon uudelleen asetukset.
- src/components/ToDoForm.svelte:

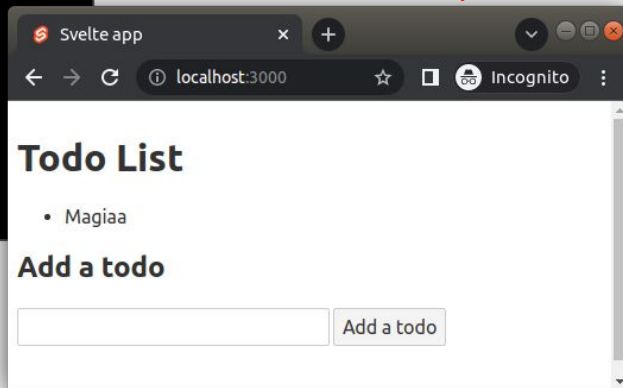
```
<script>
import todos from "@stores/todoStore.js";

const addTodo = () => {
  const todoText = document.getElementById("todo").value;
  $todos = [...$todos, {description: todoText}];
  document.getElementById("todo").value = "";
}
</script>

<h2>Add a todo</h2>
<input type="text" id="todo" />
<button on:click={addTodo}>Add a todo</button>
```

Muutetaan asetusta siten, että muuttujaan asetetaan uusi arvo.

"Luo listasta kopio, ja aseta viimeiseksi arvoksi uusi arvo."



Tehtävä: varasto tehtäviä varten

- Muokkaa sovellusta edellistä esimerkkiä noudattaen siten, että
 - sovelluksessa on varasto tehtävää varten
 - sovelluksessa listataan varastossa olevat tehtävät
 - sovelluksessa on erillinen komponentti tehtävien lisäämistä varten
- Tämän jälkeen, lisää komponentti, joka kertoo tehtävien määrän (`{ $todos.length }`)

Todo-komponentti

- Mielekästä tehdä myös komponentti Todo, jota käytetään tehtävän näyttämiseen.
- src/components/MoreTodos.svelte:

```
<script>
  import todos from "@stores/todoStore.js";
  import Todo from "@components/Todo.svelte";
</script>

<h1>Todo List</h1>

<ul>
  {#each $todos as todo}
    <li><Todo item={todo} /></li>
  {/each}
</ul>
```

Todo-komponentti

- Mielekästä tehdä myös komponentti Todo, jota käytetään tehtävän näyttämiseen.
- src/components/MoreTodos.svelte:

```
<script>
  import todos from "@stores/todoStore.js";
  import Todo from "@components/Todo.svelte";
</script>

<h1>Todo List</h1>

<ul>
  {#each $todos as todo}
    <li><Todo item={todo} /></li>
  {/each}
</ul>
```

Todo-komponentissa
käytössä muuttuja item

Todo-komponentti

- Todo-komponentissa ainakin sisällön näyttäminen.
- src/components/Todo.svelte:

```
<script>  
  export let item;  
</script>  
  
<span>{item.description}</span>
```

Tehtävä: todo-komponentti

- Muokkaa sovellusta siten, että
 - sovelluksessa on komponentti tehtävän näyttämistä varten
 - tehtävätekstiä klikkaamalla tehtävä muuttuu yliviivatuksi (ei vielä kommunikaatiota palvelimen kanssa)
 - Vinkkejä; uusi muuttuja, on:click, , {#if...

Kommunikaatio palvelimen kanssa

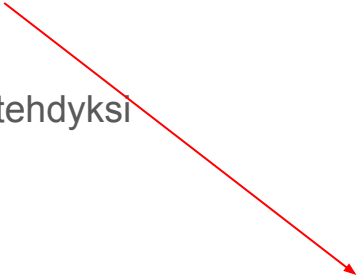
- Tapahtunut fetch-kutsulla, edellisellä kerralla kaikki toiminnallisuus samassa tiedostossa (vrt. Todos.svelte)
- Mielekästä tehdä oma tiedosto, johon kommunikaatiotoiminnallisuus keskitetään – näin myöhempi muokkaus myös helpompaa
- Tehdään kansioon src kansio http, ja luodaan kansioon tiedosto todoHandler.js

Kommunikaatio palvelimen kanssa

- Tiedostoon todoHandler.js määritellään palvelimen HTTP-apin kanssa tapahtuva kommunikaatio.
 - Hae tehtävät
 - Lisää tehtävä
 - Merkitse tehtävä tehdyksi
 - Poista tehtävä

Kommunikaatio palvelimen kanssa

- Tiedostoon todoHandler.js määritellään palvelimen HTTP-apin kanssa tapahtuva kommunikaatio.
 - Hae tehtävät
 - Lisää tehtävä
 - Merkitse tehtävä tehdyksi
 - Poista tehtävä



```
let serverName = "localhost";  
if (window.location.host.includes("todos")) {  
  serverName = "api";  
}  
  
const findTodos = async () => {  
  const todos = await  
    fetch(`http://${serverName}:4000/todos`);  
  return todos.json();  
};  
  
export { findTodos };
```


Sovelluksen alussa – onMount

- Svelte tarjoaa funktion onMount, jota voidaan käyttää sovelluksen avaamisen yhteydessä tapahtuviin toiminnallisuuksiin. Yksi hyvä käyttötapaus on tehtävien lataaminen.

```
<script>
import { onMount } from 'svelte';
import { findTodos } from
"@/http/todoHandler.js";
import todos from "@/stores/todoStore.js";
// jne..
onMount(async () => {
  $todos = await findTodos();
});
</script>
```

Tehtävä: tehtävien lataaminen

- Muokkaa sovellusta siten, että
 - tehtävät ladataan tietokannasta sovelluksen käynnistyessä
 - TodoForm-komponentti lisää tehtävän tietokantaan ja lataa tehtävät tietokannasta

Tunnistautuminen

Tunnistautuminen

- Kaksi osaa: autentikaatio ja autorisaatio
 - Autentikaatio → käyttäjän tunnistaminen esim. kirjautumisen yhteydessä
 - Autorisaatio → käyttäjän oikeuksien varmistaminen (saako tehdä toivotun operaation)
- Pääroolissa palvelin, miksi?

Tunnistautuminen

- Kaksi osaa: autentikaatio ja autorisaatio
 - Autentikaatio → käyttäjän tunnistaminen esim. kirjautumisen yhteydessä
 - Autorisaatio → käyttäjän oikeuksien varmistaminen (saako tehdä toivotun operaation)
- Pääroolissa palvelin, miksi?
*Selaintoiminnallisuus selaimessa,
joka on käyttäjän koneella →
helppo kiertää*

Tunnistautuminen

- Toiminnallisuus käyttäjän tunnistamiseen
 - Palvelimelle lähetetään käyttäjätunnus ja salasana → näitä etsitään tietokannasta → mikäli löytyy, palautetaan tästä kertova viesti sekä “tunnistekoodi”
 - Jatkossa pyyntöihin lisätään selaimessa tunnistekoodi – mikäli palvelin tunnistaa koodin, on käyttäjä tunnistautunut, muulloin ei.
 - Palvelinpäässä middleware, joka hakee koodin pyynnöistä ja selvittää sen perusteella käyttäjän tunnuksen
- Paljon valmiita kirjastoja, esim. jwt
 - <https://deno.land/x/djwt@v2.4>

Naiivi tunnistautuminen – demo

- Selain tallentaa käyttäjän tunnuksen, mikäli sellaista ei ole, luodaan uusi
- Selain lähettää tunnuksen pyynnön yhteydessä palvelimelle
 - Tunnus lähetetään pyynnön otsakkeena
- Muokkauksia tietokantatoiminnallisuuteen
 - Tietokantataulua muokataan siten, että tehtävässä käyttäjän tunnus
- Palvelintoiminnallisuuden muokkaus
 - Luodaan middleware, joka ottaa otsakkeesta tunnuksen, ja lisää sen pyynnön käyttöön (esim. kontekstiin)
 - Tehtäviä haettaessa, lisättäessä, ym, kyselyissä käytetään tunnusta

Lisää tehtäviä.

- Tehtävän merkitseminen tehdyksi
- Tehtävän poistaminen
- Testien kirjoittaminen

(mahdollisesti naiivilla tunnistautumisella)

Yhteenvetoa päivästä

+ *seuraava päivä lyhyesti*