**WikiSearchille Reactilla frontend**

Tätä harjoitusta ennen on täytynyt tehdä backend-valmiiksi, jonka ohjeet löydät aikaisemmasta harjoituksesta.

**LYHYT KERTAUS BACKENDIN API-TIEDOSTOON**

**backend/api.js**

const express = require("express");

var cors = require("cors");

const { wikisearch } = require("./wikisearch");

const app = express();

app.use(cors());

app.listen(4000, () => {

console.log("Server running on port 4000");

});

app.get("/hello", (req, res, next) => {

res.end("Hello world!");

});

app.get("/wiki", (req, res, next) => {

wikisearch(req.query.haku, (error, data) => {

if (error) {

res.end(error);

} else {

res.json(data);

}

});

});

app.listen (4000) = Kuunnellaan tässä porttia 4000

app.get("/wiki", (req, res, next)

käytössä on http-serveri **Express**, joka tukee http-kutsuja, jossa polku (path) **/wiki**

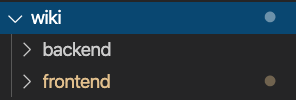
Perusperiaate on, että sieltä tulee pyyntö **request** (req) ja sinne lähettää takaisin **response** (res) eli vastaus.

Eli lyhyesti, kuinka backend-toimii. Tässä kirjoitetaanselaimen osoiteriville get-tyyppinen (**app.get**) kutsu, jotta se toimii selaimessa : localhost:4000/wiki?haku=koli

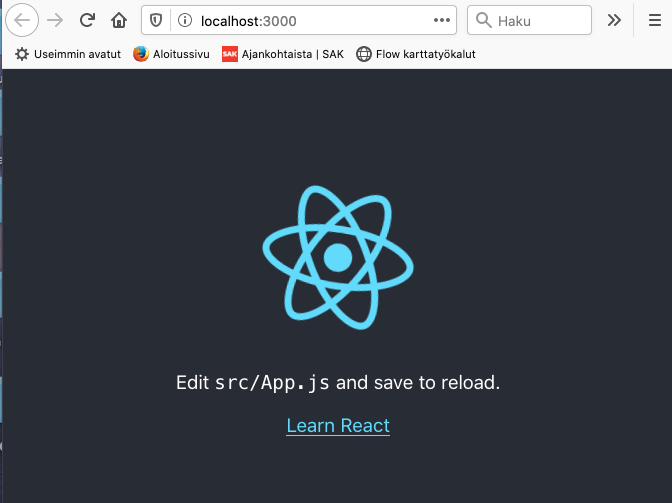
**LUODAAN FRONTEND-KANSIO**

1. Luo samaan edellisessä harjoituksessa luomaasi wiki-kansioon myös frontend-kansio komennolla

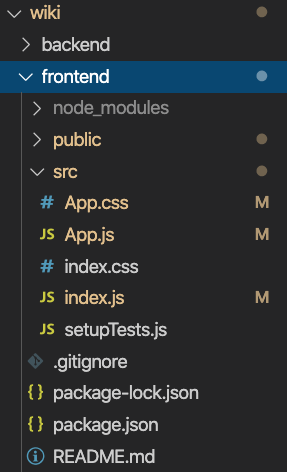
create-react-app frontend



1. Käynnistä **npm start**. ja tarkista selaimesta, että Localhost:3000-palvelin käynnistyy.



1. Siirry Terminal-ikkunassa frontend-kansioon **cd frontend**-komennolla.
2. Terminal-ikkunassa anna **npm install**-komento
3. Muokkaa hakemistorakenteesta ”turhat tiedostot” pois ja muokkaa **app.js** sekä **index**.**js** -tiedostojen sisällöt niin, että saat Reactin käynnistymään



1. Tällä kertaa jätämme tyylitiedoston App.css:n käyttöön, mutta tyhjennetään sisältö alla olevan näköiseksi:

**frontend/app.css**

.App {

text-align: center;

}

1. Muokataan App.js-tiedoston sisältö mahdillisimman minimiin

import React from "react";

import "./App.css";

function App() {

return <div className="App">Hello World!</div>;

}

export default App;

1. Käynnistä välilläTerminal-ikkunassa **npm start**, tulee teksti Hello World! selaimeen, niin silloin OK.
2. Seuraavana lähdetään tekemään yhteytä backendiin eli rakennetaan kommunikointi backendin kanssa. Tähän yhteyden luomiseen käytetään [**axios-kirjastoa**](https://www.npmjs.com/package/axios) (vastaava **Request**-kirjasto ei enää ole käytössä Reactissa).
3. Mene frontend-kansioon ja aja Terminal-ikkunassa

npm install axios

1. Muokataan fronend-kansiosta **app.js**-tiedostoa

**frontend/app.js**

import React, { useState } from "react";

import axios from "axios";

import "./App.css";

function App() {

axios

.get("http://localhost:4000/hello")

.then(function (response) {

console.log(response);

})

.catch(function (error) {

console.log(error);

});

return <div className="App">Hello World!</div>;

}

export default App;

axios. get illä http-pyynto porttiin 4000

.then-lausessa vastaus ja tiedot, jos kaikki meni ok.

.catch lausessa virheet

Jos yrität tässä vaiheessa testata tätä appia, selain blokkaa meidät tässä vaiheessa, koska yritetään hakea tietoa eri portista eli eri applikaatiosta tietoa. Backend toimii portissa 4000 ja tämä Frontend portissa 3000 eli niiden origin ei ole sama.

Tätä varten tarvitaan ”välipalikka”, jossa kerrotaan mitä hyväksytään ***Cors***:n avulla **tai** tehdään proxytys eli reititetään liikenne portin 3000-kautta. (CORS = Cross-Origin Resource Sharing)

1. Mennään backend-kansioon ja asenna sinne cors-paketti (**Lisätietoja**: [cors](https://www.npmjs.com/package/cors) ja [**Express cors**](https://expressjs.com/en/resources/middleware/cors.html))

npm install cors

Cors on ns. middleware eli kaikki pyynnöt menee sen läpi.

Lisää ensimmäisenä importtaus: var cors = require(”cors”);

Otetaan use:lla (app.use(cors())middleware käyttöön ja sallitaan kaikki origineista tulevat pyynnöt eli hyväksytään tässä minkä tahansa lähteen pyynnölle, kun Corssille ei anneta mitään ehtoja. Voit lukea myös [**Mozillan sivuilta**](https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS) lisätietoa.

**backend/api.js**

const express = require("express");

var cors = require("cors");

const { wikisearch } = require("./wikisearch");

const app = express();

app.use(cors());

app.listen(4000, () => {

console.log("Server running on port 4000");

});

app.get("/hello", (req, res, next) => {

res.end("Hello world!");

});

app.get("/wiki", (req, res, next) => {

wikisearch(req.query.haku, (error, data) => {

if (error) {

res.end(error);

} else {

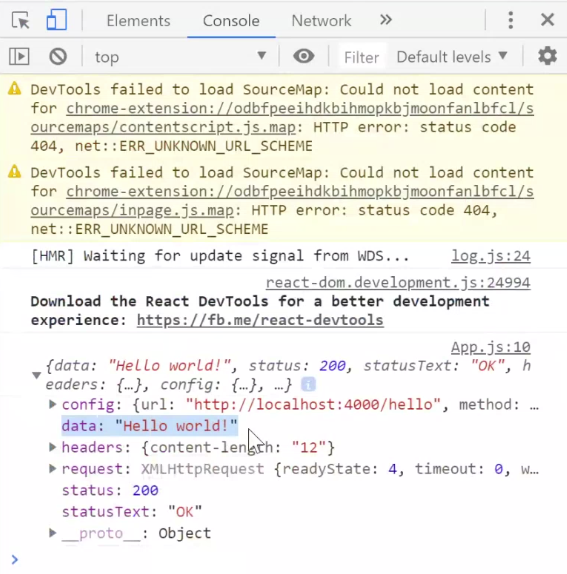
res.json(data);

}

});

});

1. Kokeile tässä vaiheessa käynnistää palvelin ja katso toimiiko.
2. Katso myös F12:sta konsolista, miten data siellä näkyy



1. Jotta pystymme tämän datan esittää jossain, tämä on pakko tallentaa, jossa käytetään useStatea.

importtaukseen ensin lisää useState.

message, jolla funktio asetetaan, joka saadaan useStatesta.

Jotta saat palautettua viestin selaimelle, lisää rivi {message}

Käytetään useEffectiä, koska muuten saattaisi haku mennä luuppiin. Eli jatkossa useEffect tapahtuu aina kun useEccectin toisena parametrinä olevassa listassa olevat arvot muuttuu. Jos tyhjä lista [], tehdään se vain kerran. (Nämä ns. Hookeja, Lisätietoa: [reactjs.org/docs/hooks-overview.html](http://reactjs.org/docs/hooks-overview.html))

setMessagessa on data 🡪 respose.data

**frontend/app.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [message, setMessage] = useState("Ei vielä mitään");

axios

.get("http://localhost:4000/hello")

.then(function (response) {

setMessage(response.data);

})

.catch(function (error) {

console.log(error);

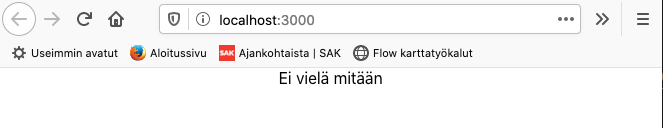
});

return <div className="App">{message}</div>;

}

export default App;

Testaa välillä selaimessa



1. Tehdäänpä seuraava välivaihe eli lisätään App.js:ään lisää koodia. Wikiä kutsutaan 🡪 http://localhost:4000/wiki?haku=joensuu"). Ja haetaan wikistä otsikko (title) ja sisältö (extract).

**frontend/App.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [data, setData] = useState();

useEffect(() => {

axios

.get("http://localhost:4000/wiki?haku=joensuu")

.then(function (response) {

setData(response.data);

})

.catch(function (error) {

console.log(error);

});

}, []);

console.log({ data });

return (

<div className="App">

{data?.[0]?.title && <h1>{data[0].title}</h1>}

{data?.[0]?.extract && <p>{data[0].extract}</p>}

</div>

);

}

export default App;

// Rivi20 tapa tehdä asia, eli jos eka ehto datalle ei toteudu ja saa arvon False,

// niin ei tehdä mitään eteenpäin:

// Voisi tehdä näinkin: <h1>{data?.[0]?.title || "Ei otsikko"}</h1>

Testaa välillä selaimessa



1. tehdään selaimella esitettävä hakukenttä, jotta saadaan tehtyä kyselyjä seuraavalla koodilla (alempaa löytyy koko ratkaisu):

<input

type="text"

placeholder="Haku"

onChange={(event) => setHaku(event.target.value)}

/>

Hakukenttä määritellään: <input type =”text” placeholder=”Haku” />, joka tulostaa vain pelkän kentän. Jotta tämä saisi jonkin arvon, Lisätään myös onChange {(event)} => event.target.value}.

Eli tämä saa eventin eli onChange ottaa funktio parametrikseen ja se funktio saa **eventin**, jolla on **targetti** joka on inputti ja sillä targetilla on value eli se tieto mitä tarvitaan.. ja mihin se halutaan asettaa on se tuo

haku (setHaku).

lisätään myös hakua varten uusi **useState**

const [haku, setHaku] = useState();

Tässä vaiheessa **fronend/App.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [data, setData] = useState();

const [haku, setHaku] = useState();

useEffect(() => {

axios

.get("http://localhost:4000/wiki?haku=joensuu")

.then(function (response) {

setData(response.data);

})

.catch(function (error) {

console.log(error);

});

}, []);

console.log({ haku });

return (

<div className="App">

<input

type="text"

placeholder="Haku"

onChange={(event) => setHaku(event.target.value)}

/>

{data?.[0]?.title && <h1>{data[0].title}</h1>}

{data?.[0]?.extract && <p>{data[0].extract}</p>}

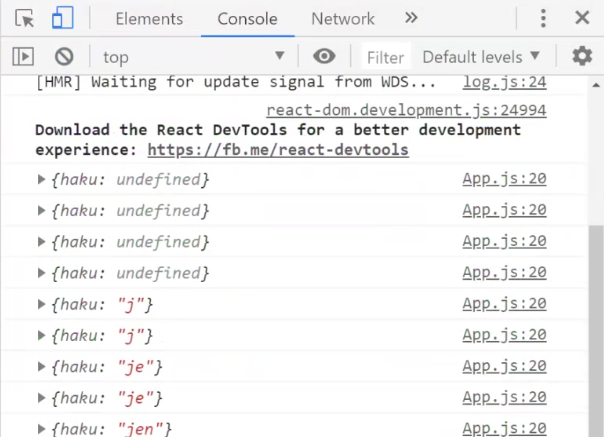
</div>

);

}

export default App;

Kun kirjoitat nyt selaimessa luomaamme hakukenttään, voit katsoa samalla konsolissa (F12), mitä tapahtuu, joka kirjoitetun kirjaimen jälkeen



1. Tehdään hausta järkevämpi. Huomioi pienet ”hipsut” (backspacen vierestä), jotta ne on oikeanlaiset, jotta voimme käyttää hakua **dollari-aaltosulku**-merkkiä.

.get(`http://localhost:4000/wiki?haku=${haku}`)

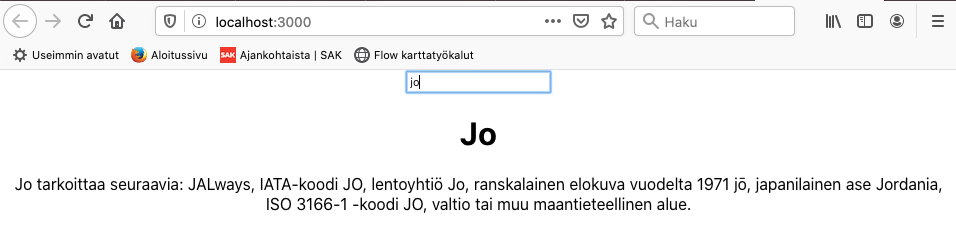
lisää vielä riville.

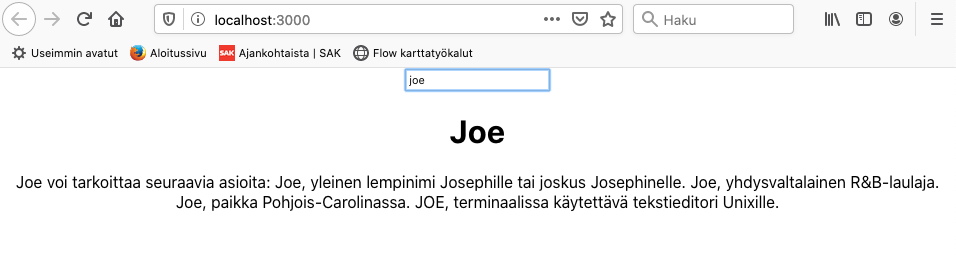
}, []);

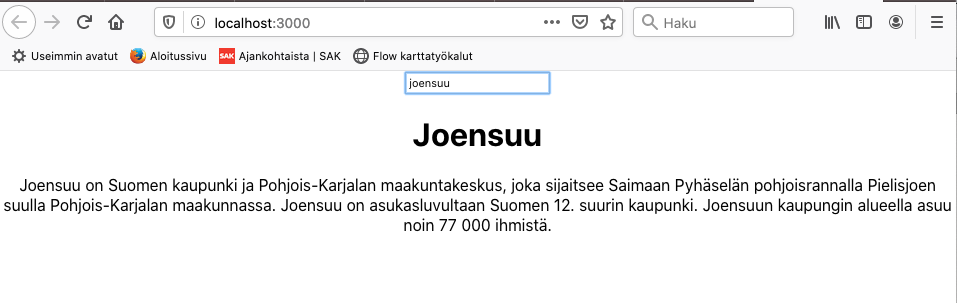
aaltosulkeiden sisälle haku, jotta, tämä rupeaa toimimaan.

}, [haku]);

Jos menet selaimeen ja kirjoitat tässä vaiheessa hakukenttään tehdäksesi halutun haun, joka kerta kun lähdemme kirjoittamaan hakukenttään, haku päivitetään (katso alapuolen kuvat). Tämä on tapa, jota käytetään, kun halutaan tehdä ehdottelemalla esim. hakusanoja.







1. Tehdään toimintoja helpottaakseen painike **Hae**

<button onClick={}>Hae</button>

Jotta haku toimisi oikein vasta, kun painetaan Hae-painikettta, täytyy määritellä useStatelle hakuTemp

const [hakuTemp, setHakuTemp] = useState();

Tämä asetaan vasta, kun Hae-painiketta painettu, hakusana on ennen sitä hakuTempistä ja haku tehdään vasta, kun klikataan Hae-painiketta, jolloin setHaussa on tieto hakua varten.

**frontend/App.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [data, setData] = useState();

const [hakuTemp, setHakuTemp] = useState();

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [data, setData] = useState();

const [hakuTemp, setHakuTemp] = useState();

const [haku, setHaku] = useState();

useEffect(() => {

axios

.get(`http://localhost:4000/wiki?haku=${haku}`)

.then(function (response) {

setData(response.data);

})

.catch(function (error) {

console.log(error);

});

}, [haku]);

return (

<div className="App">

<input

type="text"

placeholder="Haku"

onChange={(event) => setHakuTemp(event.target.value)}

/>

<button onClick={(event) => setHaku(hakuTemp)}>Hae</button>

{data?.[0]?.title && <h1>{data[0].title}</h1>}

{data?.[0]?.extract && <p>{data[0].extract}</p>}

</div>

);

}

export default App;

Eli meillä on nyt Funktionaalinen komponentti **App**, jolla on useEffectin sisällä pyyntö, joka tehdään axiosissa. Käsitellään se data (setData(response.data)), tallenetaan useStatella rakennettuun tilatietoon

Haetaan inputilla se hakutietoa ja tallennetaan se väliaikaisesti setHakuTempiin ja sitten Hae-napin painalluksella viedään se temppi tieto varsinaiseksi hakuparametriksi. Kun hakutieto muuttuu, useEffect laukaistaan uudestaan ja haetaan data. Kun se saa onnistuneesti datan, tuodaan title ja extractilla tieto

Jos haluat hakea wikipediasta thumnailkuvan, lisää seuraava koodi:

{data?.[0]?.thumbnail?.source && (

<img src={data?.[0]?.thumbnail?.source} />

)}

Backendin wikisearchiin muokkaa (lähde [Wikipedian API](https://www.mediawiki.org/wiki/API:Properties)):

"&format=json&formatversion=2&prop=extracts|pageimages&exintro&explaintext&exsentences=4";

Valmis **frontend/App.js**

import React, { useState, useEffect } from "react";

import axios from "axios";

import "./App.css";

function App() {

const [data, setData] = useState();

const [hakuTemp, setHakuTemp] = useState();

const [haku, setHaku] = useState();

useEffect(() => {

axios

.get(`http://localhost:4000/wiki?haku=${haku}`)

.then(function (response) {

setData(response.data);

})

.catch(function (error) {

console.log(error);

});

}, [haku]);

return (

<div className="App">

<input

type="text"

placeholder="Haku"

onChange={(event) => setHakuTemp(event.target.value)}

/>

<button onClick={(event) => setHaku(hakuTemp)}>Hae</button>

{data?.[0]?.title && <h1>{data[0].title}</h1>}

{data?.[0]?.thumbnail?.source && (

<img src={data?.[0]?.thumbnail?.source} />

)}

{data?.[0]?.extract && <p>{data[0].extract}</p>}

</div>

);

}

export default App;

