Essi Norri

Hiilijalanjälkilaskuri web-sovelluksena

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikan tutkinto-ohjelma

Insinöörityö

15.11.2022

Tiivistelmä

Tekijä:	Essi Norri
Otsikko:	Hiilijalanjälkilaskuri web-sovelluksena
Sivumäärä:	45 sivua
Aika:	15.11.2022
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikan tutkinto-ohjelma
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Lehtori Juha Pekka Kämäri

Tässä insinöörityössä tarkoituksena on toteuttaa hiilijalanjälkilaskuri modernina verkkosovelluksena. Ilmastokriisi on käynnissä, ja se vaatii toimenpiteitä. Avointa dataa käyttäviä teknologioita olisi mahdollista ottaa käyttöön osoittamaan ihmisille, kuinka paljon hiilidioksidia ja muita kasvihuonekaasuja heidän jokapäiväiset askareensa aiheuttavat. Lisäksi avoimen datan käytöllä voisi rohkaista ihmisiä tekemään tarpeellisia muutoksia elintavoissaan.

Kuitenkin tällaisen sovelluksen toteuttamista varten on ymmärrettävä valittuja teknologioita. Tässä insinöörityössä tehdään katsanto modernin verkkosovelluksen teoriaan ja käytännön toteutukseen. On tärkeää käydä läpi Node.js:n pääperiaatteet, ja se, kuinka sitä voidaan käyttää toteuttamaan palvelin, joka hakee luotettavasti dataa avoimesta rajapinnasta ja välittää datan selainpuolelle selkeässä ja helppokäyttöisessä käyttöliittymässä näytettäväksi. Erityisesti katse suunnataan Node.js:n asynkroniseen yksisäikeiseen tapahtumasilmukkaan ja modulaariseen luonteeseen kuten myös http-kutsujen toteuttamiseen Expressin ja Axiosin avulla. React on valittu käyttöön selainpuolelle. Työssä tutustutaan Reactin komponentteihin ja niiden tilanhallintaan Reactissa ja MobX:ssä.

Avainsanat: web-sovellus, React, Node.js, hiilijalanjälki

Abstract

Author:	Essi Norri
Title:	Carbon Footprint Calculator as Web Application
Number of Pages:	45 pages
Date:	15 November 2022
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Juha Pekka Kämäri, Senior Lecturer

In the present study, the goal was to implement a carbon footprint calculator as a modern web application. The climate crisis is ongoing and it requires urgent action. Technologies using open data could be harnessed to show people how much carbon dioxide and other greenhouse gases their daily actions end up emitting and to encourage them to make the necessary changes.

However, implementing such an application requires understanding of the chosen technologies. The paper looks at the theory and practical implementation of a modern web application covering the principles of how Node.js can be used to implement a server for the backend to fetch reliable data from an open API and pass that data to be shown in a clear and easy-to-use user interface. In particular, the asynchronic one-threaded event loop and the modular nature of Node.js as well as implementing HTTP calls with Express and Axios are examined.

React was chosen to go together with the JavaScript based Node.js. The components and state management of React and Mobx were therefore also introduced here.

Keywords: web application, React, Node.js, carbon footprint

Sisällys

Lyhenteet

HTTP: *Hypertext Transfer Protocol*. Sovellustason tiedonsiirtokäytäntö, joka perustuu pyyntöihin ja vastauksiin verkkoselaimen ja palvelimen välillä.

TCP: *Transmission Control Protocol*. Tiedonsiirtoprotokolla, jolla voidaan luoda yhteyksiä sovellusten ja tietokoneiden välille dataa sisältävien pakettien avulla.

IP: *Internet Protocol*. Alemman kerroksen protokolla, jolla toimitetaan verkkopaketteja IP-osoitteiden perusteella.

HTML: *HyperText Markup Language*. Merkintäkieli, jossa käytetään elementtejä kuten , <head>, <img> jne. verkkosivun tai -sovelluksen rakenteen luomiseksi.

CSS: *Cascading Style Sheets*. Tyylisivu, jolla määritetään HTML-elementtien tyylejä.

npm: *Node package manager*. Komentorivityökalu Node.js-moduulien asentamiseen.

REST: *Representational State Transfer*. Ohjelmistoarkkitehtuuri, jolla määritellään standardin mukaisia turvallisia rajapintoja.

XML: *Extensible Markup Language*. Merkintäkielien standardi.

DOM: *Document Object Model*. Verkkosivun rakenteen malli muistissa, joka sallii verkkosivun rakenteen ohjelmallisen muokkaamisen skripti- tai ohjelmointikielellä.

JSX: *JavaScript Syntax Extension*. JavaScriptiä ja XML-merkintäkieltä yhdistävä merkintätapa, joka luo React-elementtejä virtuaaliseen DOM:iin.

CORS: *Cross-Origin Resource Sharing*. HTTP-tunnisteisiin perustuva mekanismi, jolla palvelin voi osoittaa, mistä lähteistä selain voi ladata resursseja.

1 Johdanto

Ilmastonmuutos on suuri uhka ihmisten elinympäristölle ja sitä kautta ihmisten terveydelle ja elämälle maailmanlaajuisesti. Maailman terveysjärjestö WHO arvioi, että ilmastonmuutos aiheuttaa jo nykyään 150 000 kuolemaa vuosittain (1), ja 2030–2050 ilmastonmuutoksen takia tulee kuolemaan noin 250 000 ihmistä vuosittain (2). Yhdysvaltain liittovaltion sää- ja valtamerentutkimusorganisaatio NOAA listaa, kuinka ilmastonmuutoksen aiheuttamat sään ääri-ilmiöt, rankkasateet, tulvat, kuivuus, helleaallot ja hurrikaanit aiheuttavat vahinkoa myös ruuantuotannolle, infrastruktuurille ja ihmisten omaisuudelle sekä muille organismeille kuin ihmisille (3). Esimerkiksi Stanfordin yliopiston tutkimuksen mukaan ilmastonmuutoksen aiheuttamasta voimistuneesta sademäärästä johtuneet tulvat toivat yhteensä 75 miljardin dollarin kustannukset Yhdysvalloissa vuosina 1988–2017 (4).

Ilmastonmuutos on teollistumisen – eli pohjimmiltaan ihmisten – aiheuttama maailmanlaajuinen kriisi. Vaikka ilmaston lämpeneminen on jo aiheuttanut peruuttamattomia muutoksia eikä sitä voida enää täysin torjua, yhteiskuntien, yritysten ja yksittäisten ihmisten toiminnalla voidaan hillitä maapallon lämpenemistä ja sen pahimpia vaikutuksia. NOAA:n mukaan nykytahtiin kasvavilla hiilidioksidi- ja muilla kasvihuonekaasupäästöillä ilmaston keskilämpötila nousee 2,8–5,7 °C 1900-luvun alkupuoliskon keskilämpötilasta tämän vuosisadan loppuun mennessä. Jos vuosittaisia päästöjä kasvatetaan maltillisemmin ja vuosisadan puolesta välistä lähtien onnistutaan jopa vähentämään päästöjä, on mahdollista, että ilmaston keskilämpötila nousee vain 1,3–3,3 °C. (5.)

Ilmaston lämpenemisen hillitsemiseksi tarvitaan eri tahojen toimintaa mahdollisimman laajasti. Valtioiden on otettava hiilidioksidipäästöt huomioon lainsäädännössä, budjetoinnissa ja veroissa. Greenpeacen mukaan erityisesti valtiot ja yritykset ovat olennaisia ilmastonmuutoksen torjunnassa, ja niiden on siirryttävä pois fossiilisten polttoaineiden käytöstä sekä investoitava uusiutuvan energian

teknologioihin (6). Teknologiasta onkin usein povattu pelastajaa ilmastonmuutoksen ongelmiin. Esimerkiksi Sky Newsin artikkelissa mainitaan useita teknologioita: hiilidioksidin talteenotto, ilmaston korjaus, energiatehokkaammat laitteet ja datakeskusten parempi käyttö (7). Myös avoin data on olennaisessa asemassa ilmastonmuutoksen torjunnassa, sillä “saamalla pääsyn hiilidioksidipäästöjä ja energian kulutusta koskevaan tietoon kansalaiset […] voivat tehdä tietoon perustuvia päätöksiä vähentääkseen päästöjään” (8). Toisaalta pelkät teknologian ratkaisut eivät riitä ilmastonmuutoksen estämiseen. Esimerkiksi Financial Timesissa Cambridgen yliopiston tutkija Julian Allwood kirjoittaa, että vihreitä teknologioita ei saada tarpeeksi laajalti käyttöön, jotta niistä olisi tarpeeksi hyötyä tarpeeksi ajoissa. Etenkin sementin käyttöä, merenkulkua, lentämistä ja joitakin osia ihmisten ruokavalioissa on rajoitettava, jotta ilmastokriisin vaikutuksia saataisiin lievennettyä. (9.)

Yhteiskunnilla ja yrityksillä on tässä valtava työ. Kuitenkin yksittäisiä ihmisiäkin pidetään vastuullisena omasta toiminnastaan ja sen vaikutuksista muihin ihmisiin. Jos ihmisen toiminta vahingoittaa toista ihmistä, hän saattaa joutua maksamaan korvauksia tai jopa joutua vankilaan. Samalla tavalla voisi olettaa, että ihminen on kuluttamisestaan ja elämäntavoistaan omalta pieneltä osaltaan vastuussa, kun ne kumuloituvat miljoonien ihmisten toistaessa niitä ja aiheuttavat sitä kautta valtavia haittoja muille ihmisille ilmaston lämmetessä. Tietysti ihmisen perustarpeiden – ravinto, asuminen, sähkölaitteet – tyydyttämisestä syntyy hiilidioksidipäästöjä, mutta etenkin länsimaissa niitä syntyy myös asioista, joita ei välttämättä tarvita. Avointa dataa käyttävän hiilidioksidilaskurin avulla ihminen voi karsia omasta elämästään turhia hiilidioksidipäästöjen lähteitä ja tehdä vastuullisia valintoja, jotka ottavat myös muut ihmiset huomioon. Ehkäpä nykyisessä vuoden 2022 energiakriisissä laskurin avulla voisi myös säästää rahaa.

Suomessa on ainakin kaksi julkisen tahon yksilöille tarjoamaa ilmaista verkossa toimivaa laskuria omien elämäntapojen arvioimiseen ilmastokriisin näkökulmasta: Suomen ympäristökeskuksen Ilmastodieetti (ilmastodieetti.ymparisto.fi/ilmastodieetti) ja Suomen itsenäisyyden juhlarahaston Sitran elämäntapatesti

(elamantapatesti.sitra.fi). Molemmissa sovelluksissa voi laskea ilmastovaikutuksensa usealta elämän alalta ja saada tietoja siitä, miten eri valinnat vaikuttavat ilmastoon. Sovelluksissa on eroja, ja niitä vertailemalla huomaa, miten sovellusten hyviä ratkaisuja ja ominaisuuksia yhdistelemällä ja mahdollisesti muuntelemalla voisi toteuttaa vielä uuden erilaisen laskurin.

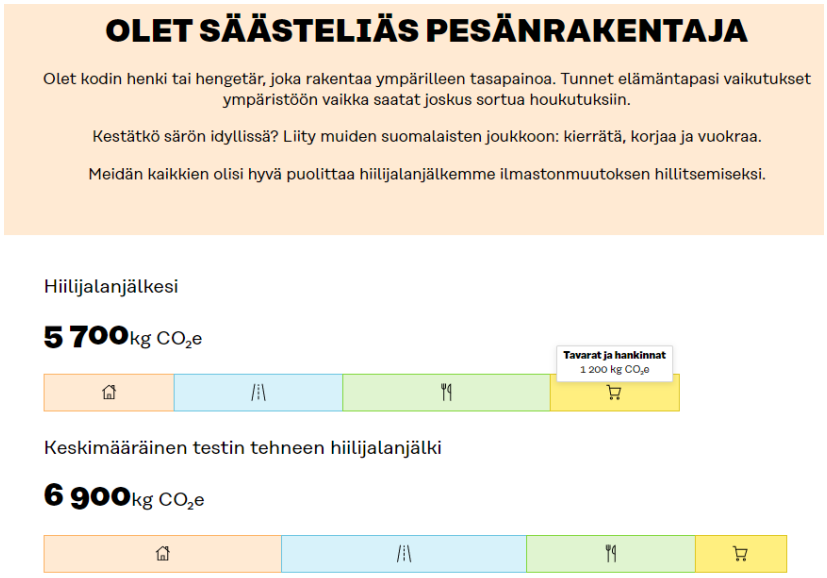
Ilmastodieetin testi on perusteellinen (kuva 1, s. 3), mutta sen pitkähköt lomakkeet voivat saada käyttäjän jättämään laskurin täyttämisen kesken. Valikoiden takaa löytyy aina lisää kysymyksiä vastattavaksi ja koko laskurin huolellisessa täyttämisessä kuluu aikaa. Internetin kyselylomakkeiden käyttöä tutkineet Elisabeth Deutskens et al. huomasivat, että pitkiin kyselylomakkeisiin vastattiin merkittävästi harvemmin kuin lyhyisiin lomakkeisiin (10). Jotta käyttäjä saisi laskurista mahdollisimman paljon irti, sen olisi hyvä olla niin tiivis, että sen saisi tehtyä helposti ja nopeasti loppuun asti.

Kuva 1. Ilmastodieetin aloitussivu.

Pelillisyys voisi olla toinen tapa saada käyttäjä jatkamaan laskurin täyttöä. Eri pedagogian tutkimuksissa on havaittu, että pelillisyyttä ja pelaamisen tunnelmaa voidaan käyttää motivoimaan ihmisiä sitoutumaan haluttuun toimintaan (11). Edes pieni pelillinen elementti voisi innostaa käyttäjää jatkamaan laskurin täyttämistä tulosten saamiseen asti.

Sitran elämäntapatesti on sisällöltään tiiviimpi ja pelillisempi kuin Ilmastodieetti. Kysymykset vilahtavat näkyville ja sanavalinnoissa on käytetty luovuutta (kuva

2, sivu 4). Sivusto on myös värikäs sekä osuvasti kuvitettu. Sen yleisilme on hauska ja kutsuva vakavasta aiheesta huolimatta.



Kuva 2. Osa Sitran elämäntapatestin tulossivusta.

Ilmastodieetissä ja Sitran elämäntapatestissä tulokset näkyvät kuitenkin pylväsdiagrammeina, joista ei saa tarkasti selville, kuinka paljon vertailtavat tulokset eroavat toisistaan. Vaikka laskuri antaa vain arvion hiilijalanjäljen suuruudesta, olisi kuitenkin mielenkiintoista vertailla laskurin tulosten numeroarvoja keskenään, jotta käyttäjä saa tarkemman käsityksen siitä, mitkä asiat hänen elämässään tuottavat paljon tai vähän päästöjä. Tuottavatko ruokavalinnat vai asumisolot eniten hiilidioksidipäästöjä? Mikä ruokavaliossa on erityisen huonoa ympäristön kannalta? Näiden tarkempien tietojen ja oman elämän tärkeysjärjestyksen perusteella käyttäjä voisi päättää helpommin, mitä tekoja ja valintoja omassa elämässä kannattaisi tehdä päästöjä vähentääkseen.

Tässä insinöörityössä on tarkoitus kehittää hiilijalanjälkilaskuri verkkosovelluksena, jossa toteutan laskurin nämä kolme ominaisuutta – lomakkeiden pituus, pieni pelillinen elementti ja tulosten esittäminen – eri tavalla kuin esimerkiksi Ilmastodieetissä tai Sitran laskurissa. Jotta sovelluksen helppokäyttöisyys ja no-

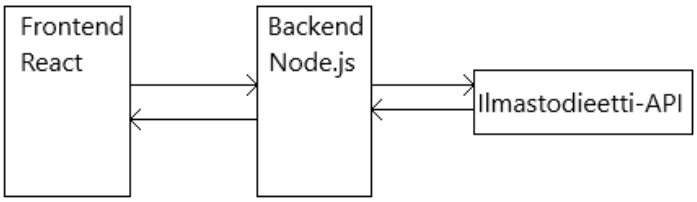
peus voidaan taata ja sovellus voidaan ensinnäkään luoda, on käytettävä soveltuvia teknologioita, kuten esimerkiksi Node.js:iä ja Reactia. Jotta niitä voi käyttää, on ymmärrettävä niiden toimintaperiaatteet. Lisäksi laskurin on laskettava hiilijalanjälki oikein, joten käytän Ilmastodieetin avointa rajapintaa datan saamiseen ja tulosten laskemiseen. Insinöörityön tarkoitus on selvittää, miten luoda helppokäyttöinen ja luotettava hiilijalanjälkilaskuri web-sovelluksena, jota voi käyttää elämäntapavalintojen tekemisen apuna.

2 Hiilijalanjälkilaskuri-sovelluksen rakenne

Jotta sovellus olisi mahdollisimman helppokäyttöinen, sen tulisi olla helposti saatavilla ja sitä pitäisi voida alkaa käyttää ilman mitään erityisiä toimenpiteitä. Sen takia on tarkoituksenmukaista toteuttaa laskuri web-sovelluksena. Web-sovellusta voi käyttää internetissä verkkoselaimen kautta tietokoneella tai mobiililaitteelta. Käyttäjän ei tarvitse asentaa mitään omalle laitteelleen vaan voi vain näppäillä sovelluksen osoitteen verkkoselaimeen.

Nykyään lähes kaikki verkkosivustot on toteutettu web-sovelluksina, jolloin ne toimivat dynaamisesti hakien ja lähettäen tietoa käyttäjän toiminnan mukaan. Sovellus koostuu selainpuolesta (frontend) ja palvelinpuolesta (backend). Frontend-puolelta lähtee HTTP (Hypertext Transfer Protocol) -pyyntöjä käyttäjän syötteiden mukaan ja backend-puolelta tulee niihin vastauksia. MDN Web Docsissa selostetaan, että HTTP on sovellustason protokolla eli yhteyskäytäntö, jonka avulla voidaan siirtää tietoa, kun asiakasohjelma, esimerkiksi selain, avaa yhteyden saadakseen pyyntöönsä palvelimelta vastauksen (12). Wikipedian mukaan avattava yhteys on TCP (Transmission Control Protocol) -yhteys, jolla voidaan lähettää tavujonoja tietokoneilta toisille tietokoneille vikatilanteet huomioon ottaen. HTTP-protokolla on rakennettu TCP-protokollan päälle, ja TCP-protokolla rakentuu IP-protokollan päälle. Verkkokerroksen IP-protokolla on ”alemman tason protokolla, joka vastaa päätelaitteiden osoitteistamisesta ja verkkopakettien reitittämisestä verkossa”. TCP- ja IP-protokollat muodostavat yhdessä TCP/IP-protokollaperheen. (13; 14; 15.) Tiedonsiirron arkkitehtuuri rakentuu siis eri kerroksille.

Tässä insinöörityössä ei mennä sovelluskerrosta syvemmälle, vaan pysytään HTTP-protokollan tasolla. Kehitettävässä sovelluksessa käyttäjä syöttää tietoja selaimessa käyttöliittymän lomakkeeseen, joista lähtee backend-palvelimelle HTTP-pyyntö saada syötteitä vastaavat hiilidioksidimäärät. Tässä tapauksessa palvelin välittää pyynnön avoimeen Ilmastodieetti-rajapintaan, josta tulee vastaus palvelimen kautta, ja vastauksen hiilidioksidipäästöjä koskevat tiedot esitetään käyttäjälle käyttöliittymässä (kuva 3, s. 6).



Kuva 3. Hiilijalanjälkilaskurin rakenne.

Suunnitellessa sovellusta on rakenteen lisäksi päätettävä käytettävät ohjelmointikielet ja sovelluskehykset. Verkkosovellusten pohjana on tietysti HTML (HyperText Markup Language) -merkintäkieli, jossa elementit muodostavat sivun rakenteen. CSS (Cascading Style Sheets) -tyylikieltä käytetään HTML:n elementtien muodostaman käyttöliittymän tyylin muokkaamiseen. Dynaamisissa verkkosovelluksissa on usein JavaScript käytettynä kielenä. JavaScript on tulkattava eli sovelluksen ajon aikana konekielelle muunnettava kieli, jota käytetään etenkin web-sovellusten selainpuolella. Se on varma valinta, mutta siitä puuttuu vahva tyypitys eli käytettäville muuttujille ei ole määriteltyjä tyyppejä. Esimerkiksi muuttuja voi olla merkkijono tai numero riippuen siitä, minkä arvon se kulloinkin saa. Vahvassa tyypityksessä muuttujan tyyppi määritellään koodissa. TypeScript on kotisivunsa mukaan ratkaisu tähän: otsikossakin jo lukee, että ”TypeScript on JavaScriptiä syntaksilla tyypeille” (16). Koska TypeScriptissä on tämä lisäominaisuus, se on hyvä valinta frontendin ohjelmointikieleksi.

JavaScriptin ja TypeScriptin kanssa voidaan käyttää eri frontendin kirjastoja tai sovelluskehyksiä. Geeks for Geeks -sivusto selventää termien eroa: Kirjasto tar-

joaa lisäfunktioita tai moduuleja, joita sovelluksen ohjelmakoodi voi kutsua tiettyjä tarkoituksia ja toimintoja varten, kun taas sovelluskehys sisältää funktio- tai oliopohjia, jotka ohjelmoija täydentää kirjoittamalla niihin koodia. Sovelluskehys myös kutsuu ohjelmakoodia, eli se toimii päinvastaisella tavalla kuin kirjasto. (17.) Sovelluskehyksiä ovat mm. Angular.js, Vue.js ja Ember.js. React puolestaan on laajuudestaan huolimatta kirjasto. Reactissa on useita hyviä puolia, joita esimerkiksi FreeCodeCamp luettelee: React on joustava, sillä sen pääominaisuutena on luoda komponentteja, joita voi käyttää eri tarkoituksiin. React on lisäksi helppo oppia, sekä sillä on laaja käyttäjäkunta ja hyvä suorituskyky, koska siinä minimoidaan DOM-puun päivitys. (18.) Muun muassa näistä syistä React valikoitui käyttöön hiilijalanjälkilaskurissa.

Backendin toteuttamiseen on myös useita mahdollisuuksia. Toteutuksessa voidaan käyttää esimerkiksi Javaa, Node.js:iä, PHP:tä tai Rubya. Koska Node.js pohjaa myös JavaScriptiin, JavaScriptin käyttö koko sovelluksessa yksinkertaistaisi sitä, ja lisäksi Chromen V8-JavaScript-moottori myös mahdollistaa Node.js:n erittäin hyvän suorituskyvyn (19). Jotta hiilijalanjälkilaskuria on nopea käyttää, on myös suorituskyky tärkeä valintaperuste.

Web-sovellus muodostuu siis eri osista, jotka kommunikoivat toistensa kanssa. Selain-puolelta käyttäjä lähettää pyynnön palvelinpuolelle, joka lähettää selainpuolelle vastauksen pyyntöön. Node.js toimii palvelinpuolella ja React selainpuolella. Seuraavissa luvussa käsitellään hiilidioksidilaskurin toteutuksessa käytettävien teknologioiden, Node.js:n ja Reactin, teoriaa.

3 Node.js

Web-sovellus muodostuu siis eri osista, jotka kommunikoivat toistensa kanssa. Selain-puolelta käyttäjä lähettää pyynnön palvelinpuolelle, joka lähettää selainpuolelle vastauksen pyyntöön. Node.js toimii palvelinpuolella ja React selainpuolella.

Node.js on JavaScript-pohjainen ajonympäristö. Ajoympäristö on sovellusten suorittamiseen tarvittava ympäristö, joka sisältää “laitteisto- ja ohjelmistoinfrastruktuurin, joka tukee halutun koodikannan ajamista reaaliajassa” (20). Node.js:ssä koodikanta on tulkattavaa JavaScript-kieltä. Node-ajoympäristö sisältää Chromen V8-moottorin, joka muuntaa koodin laitteistolle luettavaan muotoon, ja Noden rajapinnan ja JavaScript-moduuleja, jotka ovat uudelleenkäytettäviä koodilohkoja (21). Node-ajoympäristö mahdollistaa sen, että JavaScript-koodia voidaan käyttää palvelinpuolella ilman selainta vastaanottamaan selainpuolelta tulevia pyyntöjä ja vastaamaan niihin.

3.1 Node.js:n yksisäikeinen asynkroninen tapahtumasilmukka

Jotta Node.js:iä voi käyttää, täytyy ymmärtää, miten se toimii. Basarat Ali Syed ja Martin Bean kirjoittavat kirjassaan Beginning Node.js, että Node.js:n toiminta perustuu syötteiden ja tulosteiden käsittelemiseen yksisäikeisessä tapahtumasilmukassa asynkronisesti. Tapahtumasilmukka tarkoittaa sitä, että kun sovelluksessa käynnistyy tapahtuma (sovellus saa jonkin syötteen tai sille tulee asiakasohjelman pyyntö), tapahtumaa vastaava funktio (eli toiminnan sisältävä koodilohko) suoritetaan ja usein saadaan syötettä vastaava tuloste. Seuraavaan tapahtumaan siirrytään vasta, kun edellinen funktio on suoritettu. (22.)

Usein perinteisissä verkkopalvelimissa on useita säikeitä, joista kukin suorittaa ohjelmasta eri osia tai käsittelee asiakasohjelman eri pyyntöjä yhtäaikaisesti. Voisi ajatella, että on tehokkaampaa ja nopeampaa suorittaa useita funktiota säikeiden avulla samaan aikaan, mutta se tuhlaa laitteiston muistia ja suorittimen resursseja. (22.)

Kuitenkin JavaScriptin avulla Node.js on myös erittäin tehokas vain yhdellä säikeellä JavaScriptin ominaisuuksien vuoksi. JavaScriptissä on mahdollista välittää funktioita toisille funktioille parametreina. Funktioita, joita voidaan käsitellä kuin muuttujia eli asettaa parametreiksi tai palauttaa toisen funktion paluuarvona, sanotaan ensimmäisen luokan funktioiksi. Funktioita, jotka ottavat funktioita parametreina, sanotaan korkeamman asteen funktioiksi. Näitä molempia on

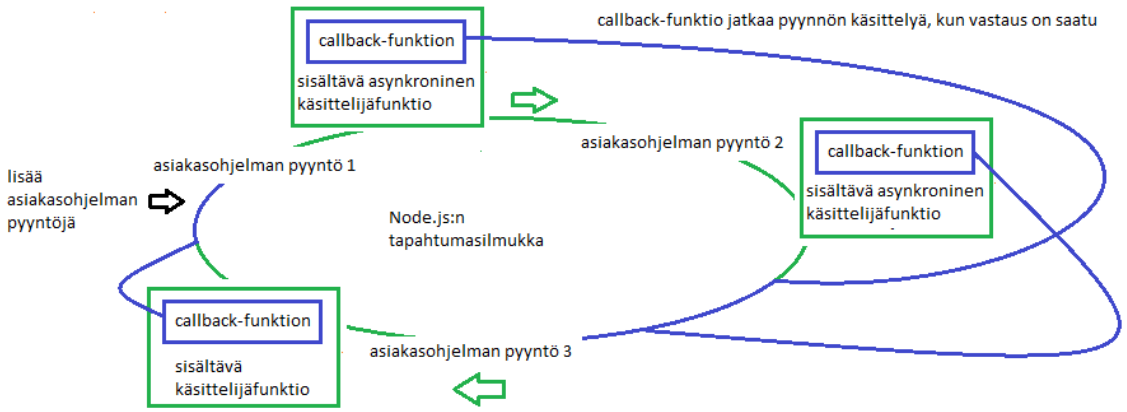
Node.js:n käyttämässä JavaScriptissä. Nämä funktiotyypit tekevät mahdolliseksi sen, että kun korkeamman asteen funktiossa kutsutaan parametrina saatua ensimmäisen luokan funktiota, ensimmäisen luokan funktio pystyy käyttämään myös korkeamman asteen funktion muuttujien arvoja myös sen jälkeen, kun korkeamman asteen funktio on tullut loppuunsa ja palauttanut paluuarvon. (22.) Esimerkkikoodi 1 sisältää korkeamman asteen funktion setTimeOut. Se saa parametrikseen ensimmäisen luokan funktion, joka tulostaa tekstin.

```
setTimeout(function () {
    console.log('2000 milliseconds have passed since this demo
started');
}, 2000);
```

Esimerkkikoodi 1. Esimerkissä esiintyy korkeamman asteen funktio ja ensimmäisen luokan funktio. Esimerkki on kirjasta Beginning Node.js. (22.)

MDN Web Docsin sanaston mukaan callback-funktio on ensimmäisen luokan funktio, joka välitetään toisella funktiolle parametrina ja jota kutsutaan toisen funktion sisältä tekemään loppuun haluttu toiminto. Callback-funktioita käytetään usein jatkamaan koodin suoritusta asynkronisen funktion sisällä. Asynkronisuus tarkoittaa sitä, että ohjelman suorituksen ei tarvitse jäädä odottamaan pyynnön vastausta, vaan voi siirtyä toiseen toimintoon ja palata ensimmäisen pyynnön loppuunsaattamiseen, kun vastaus on saatu. (23; 24.) Kun callback-funktio suoritetaan, se sisältää samat tiedot asiakasohjelman pyynnöstä kuin korkeamman asteen funktio, joka on suorittanut asiakasohjelman pyyntöä. Se pystyy saattamaan oikeaan asiakasohjelman pyyntöön vastaamisen loppuun käyttämällä korkeamman asteen funktiosta saamiaan pyynnön tietoja. (22.)

Tällä tavalla Node.js:n yhden säikeen tapahtumasilmukassa voi suorittaa tehokkaasti yhden tapahtuman laukaisseen asynkronisen funktion, siirtyä toiseen tapahtumaan ja palata ensimmäisen tapahtuman vastaukseen vasta, kun se on valmis (kuva 4, s.10). Asynkronisuuden avulla sovelluksen suorittaminen ei jumiudu odottamaan vastausta yhdelle pyynnölle, ja sovellus toimii käyttäjillä sujuvasti ilman muiden pyynnöistä johtuvia odotteluja. Asiakasohjelman moniin pyyntöihin vastaaminen onnistuu tehokkaasti.



Kuva 4. Tapahtumasilmukka käsittelee yhden tapahtuman kerrallaan ja siirtyy seuraavaan.

Kun funktioita haluaa suorittaa asynkronisesti, ohjelmoijan on huolehdittava siitä ohjelmakoodissa. Njong Emy kertoo Asynchronous JavaScript -artikkelissaan, että callback-funktioita voi käyttää asynkronisesti, mutta jos niitä tarvitaan monia sisäkkäin, ohjelmakoodia alkaa olla vaikeaa lukea ja voi syntyä ns. ”callback-helvetti”. Ratkaisuna ”callback-helvettiin” ovat lupaukset (promises), jotka rakentuvat callback-tekniikan päälle, ja niillä voidaan toteuttaa asynkronisuutta selkeämmin. (25.)

MDN docseista löytyy perusteellinen kuvaus nykyaikana callback-funktioita useammin käytetyistä lupauksista: Lupaus on olio, joka edustaa asynkronisen operaation onnistumista tai epäonnistumista ja operaation tuloksena saatavaa arvoa (26). Asynkroninen funktio tai metodi palauttaa lopullisen arvon sijaan lupauksen arvon saamisesta myöhemmin. Tämän ansiosta asynkroniset funktiot ja metodit voivat palauttaa jotakin samalla tavalla kuin synkroniset funktiot ja metodit, jotka suoritetaan peräkkäin yksi toisensa jälkeen muuttamatta aikajärjestyksessä mitään. Lupauksen lopullinen tila voi olla joko ”toteutunut”, jos operaatio onnistuu, tai ”hylätty”, jos operaatio epäonnistuu. Kumpaankin lupauksen lopputilaan liitetään niitä käsittelevät callback-funktiot, kuten esimerkkikoodissa 2. Jos lupauksen lopullinen tila on ”toteutunut”, sitä vastaava callback-funktio suoritetaan asynkronisesti ja asynkroninen funktio saa lopulta paluuarvon. Jos lopullinen tila on ”hylätty”, kutsutaan asynkronisesti callback-funktiota, joka käsittelee virhetilanteen. (27.)

```
const p1 = new Promise((resolve, reject) => {
  resolve("Success!");
  // or
  // reject(new Error("Error!"));
});
p1.then(
  (value) => {
    console.log(value); // Success!
  },
  (reason) => {
    console.error(reason); // Error!
  },
);
```

Esimerkkikoodi 2. Lupausolion luominen ja sille välitettävät callback-funktiot, jotka käsittelevät operaation onnistumisen tai virhetilanteen. (28.)

Then()-metodin lohkoon voidaan liittää lupauksen loppuun, ja siihen voidaan sisällyttää callback-funktiot eri tilanteisiin. Then()-metodi palauttaa myös lupauksen, joten then()-metodeja voi liittää ketjuksi toistensa perään, jolloin asynkronisia funktioita voi laittaa myös sisäkkäin ja peräkkäin. (28.) Ohjelmakoodista tulee then()-metodin avulla selkeämpää ja helpommin luettavaa kuin ”callback-helvetissä”.

Esimerkkikoodista 3 näkee, että lupauksissa voidaan käyttää myös async- ja await-taikasanoja then()-metodin sijaan (26). Async kertoo funktion olevan asynkroninen ja await aiheuttaa asynkronisen funktion suorituksen keskeytyksen, kunnes await –operaattorin avulla saatu lupauksen lopullinen tila on joko ”toteutunut” tai ”hylätty”, ja sen mukaisesti funktion suoritusta jatketaan (29). Tällä tavalla päästään lähelle synkronisten funktioiden esitystapaa. Myös ”hylättyjen” lupausten käsittely voidaan siirtää virheet hoitavaan catch-lohkoon. (30.)

```
async function foo() {
  try {
    const result = await doSomething();
    const newResult = await doSomethingElse(result);
    const finalResult = await doThirdThing(newResult);
    console.log(`Got the final result: ${finalResult}`);
  } catch (error) {
    failureCallback(error);
  }
}
```

Esimerkkikoodi 3. Async- ja await-tunnussanoja käyttävä asynkroninen funktio, joka palauttaa ketjutettuja lupauksia. (26.)

Tapahtumasilmukassa lupaukset siis varmistavat nykyään asynkronisuuden, ja sovellus toimii tehokkaasti.

3.2 Npm ja modulaarisuus

Ohjelmoidessa kaikkea koodia ei voi laittaa yhteen tiedostoon, kun koodi monimutkaistuu, kirjoittaa Sarvesh Kadam. Node.js:n modulaarisuus tulee tässä avuksi ja ohjelmakoodista voi tehdä erillisiä moduuleja, yksinkertaisimmallaan erillisiä JavaScript-tiedostoja, joita voi käyttää sovelluksen osana. Omia funktioita erillisistä JavaScript-tiedostoista voi käyttää muualla Node.js-sovelluksessa, kun tällaisen tiedoston loppuun muistaa lisätä ”module.exports = funktionNimi” -koodirivin. Tiedostoon, jossa tätä funktiota halutaan käyttää, pitää lisätä ”require('./tuotavanTiedostonNimi)”-rimpsu. (31.)

Ohjelmoija voi tehdä omia moduuleita, mutta lisäksi Node.js:ssä on omat ydinmoduulinsa, joita voi käyttää omassa sovelluksessa. Myös nämä moduulit on tuotava ”require”-lauseen avulla osaksi ohjelmakoodia. (31.)

Lisäksi npmjs.com-verkkosivulla on saatavilla kymmeniä tuhansia muiden Node-ohjelmoijien moduuleita veloituksetta, kertoo JP Evans verkkokurssissaan. Niitä voi asentaa npm:n eli Node.js:n komentoriviltä toimivan työkalun avulla. Komennolla ”npm init” luodaan package.json-tiedosto, josta kaikki ”npm install” -komennolla asennetut moduulit löytyvät. Lisäksi npm lisää moduulit node_modules -kansioon ja asentaa samalla haluttujen moduulien tarvitsemat muut moduulit. Myös näiden moduulien käyttöön omassa sovelluksessa tarvitaan ”require”-sanaa. (32.) Mayank Agarwal kirjoittaa, että npm:n komennolla ”npm start” voi myös ajaa sovelluksen (33).

3.3 Axios ja http-pyynnöt

Axioksen dokumentaation mukaan Axios on lupauksiin perustuva moduuli http-pyyntöjen tekemistä ja vastausten käsittelyä varten. Se voidaan lisätä npm in-

stall -komennolla osaksi Node.js-palvelinta. Se on siitä mielenkiintoinen JavaScript-kirjasto, että sitä voi käyttää palvelinpuolella Node.js:ssä sekä selaimessa. Palvelinpuolella tätä moduulia käytetään yhdessä Node.js:n natiivin http-moduulin kanssa, ja selainpuolella XMLHttpRequest-objektin kanssa. (34.)

Http-pyynnöt on jo mainittu tässä insinöörityössä, mutta XMLHttpRequest-objektia ei vielä ole. On kuitenkin vielä tärkeää tarkentaa http-pyyntöjen määritelmää ja käyttöä ennen XMLHttpRequest-pyyntöjen avaamista. Mozillan MDN Web Docs -sivustolla kerrotaan kattavasti http-pyynnöistä: Http-pyynnöt ovat siis tapa siirtää tietoa sovellusten välillä pyyntöjen ja vastausten kautta. Http-pyynnöt ovat http-viestejä, kuten myös http-vastaukset, jotka vastaavat useimmiten lupausten avulla http-pyyntöihin. Http-vastaukset sisältävät mm. statuskoodin, joka kertoo, onnistuiko pyyntö, ja mahdollisesti rungon (body), joka sisältää haetun datan. (35.)

Http-pyynnöt koostuvat metodista, polusta ja http-protokollan versionumerosta sekä usein myös valinnaisista tunnisteista (headers) ja rungosta. Http-metodi määrittää, haluaako asiakasohjelma saada tietoa palvelimelta tai toiselta sovellukselta, lisätä uutta tai päivittää vanhaa tietoa vai poistaa vanhentunutta tietoa. GET-metodilla pyyntö voi hakea tietoa, POST-metodilla lisätä uutta tietoa, PUT-metodilla päivittää vanhaa tietoa ja DELETE-metodilla voi poistaa vanhaa tietoa. Näiden lisäksi on myös muita http-metodeja. Polku on pelkistetty URL-reitti resurssiin, josta tietoa haetaan tai jossa sitä halutaan muuttaa. Esimerkiksi ”developer.mozilla.org” tai pelkkä TCP-portti voi olla http-pyynnössä oleva polku, jolla haluttu resurssi löytyy. (35.) Sivumainintana voi vielä todeta, että URL tarkoittaa “ainutlaatuista merkkijonoa, joka identifioi verkkoteknologioiden käyttämän loogisen tai fyysisen resurssin” (36).

Http-pyyntöjen tekemistä varten tarvitaan rajapinta kuten fetch-rajapinta tai Axios. Jacoby McCann kirjoittaa, että Axiosin avulla ohjelmakoodin syntaksi on helppolukuista, ja se selkeyttää http-pyyntöjen tekemistä esimerkiksi REST-ra-

japinnoille (37). REST eli Representational State Transfer on ohjelmistoarkkitehtuuri, joka määrää, miten sovelluksen tai palvelimen rajapinnan tulisi toimia, jotta se on turvallinen, luotettava ja tehokas. (38)

Esimerkkikoodissa 4 Axiosilla tehdään http-pyyntö käyttäen GET-metodia, jonka parametrina on polku haluttuun resurssiin. Then()-metodi huolehtii siitä, että lupauksen toteuduttua ja vastauksen saavuttua tiedon runko eli data käsitellään. Myös virheen käsittely on helppo lisätä yhdellä rivillä catch-lohkoon.

```
const getPosts = () => {
  .get(https://zenquotes.io/api/quotes)
  .then(data => console.log(data.data))
  .catch(error => console.log(error))
}
```

Esimerkkikoodi 4. Axiosilla toteutettu http-pyyntö. (37)

Selainpuolella Axios käyttää XMLHttpRequest-rajapintaa http-pyyntöjen tekemiseen, JavaScriptInfo-sivustolla kerrotaan. XMLHttpRequest on selaimeen sisällytetty objekti, jolla voi tehdä http-pyyntöjä. Vaikka termissä onkin XML, XMLHttpRequestillä voidaan käsitellä dataa myös eri muodoissa. (39.) XML on merkintäkielistandardi (40). Axios hyödyntää XMLHttpRequestiä, jotta Axios voidaan ottaa käyttöön myös selainpuolella (39). Axios myös piilottaa XMLHttpRequestin yksityiskohdat ns. ”pellin alle” eikä käyttäjän tarvitse pohtia niitä erikseen.

Syntaksiltaan Axios näyttää samalta selain- ja palvelinpuolella, mutta miten Axiosia voidaan käyttää molemmissa? Medium-sivuston Why Axios Can Be Used Both in Browser and Node.js -artikkelissa todetaan, että kun Axiosin avulla lähetetään http-pyyntö, Axios käyttää kahta eri adapteria pyynnön lähettämiseen palvelimelle. Axios pystyy päättelemään ympäristöstä, kumpaa adapteria sen tulee käyttää, XMLHttpRequest-objektia vai Node.js:n http-moduulia. (41.)

3.4 Express

Yksi npm:n avulla saatavista moduuleista on Express. Express on suosittu JavaScript-web-sovelluskehys, joka toimii Node.js-ajoympäristössä (42). JP Evansin mukaan Express-kehyksen avulla on yksinkertaisempaa hoitaa http-yhteydet, muokata dataa sekä huolehtia kutsujen reitityksestä (32). Myös John Smilga selostaa, että koko palvelimen ja APIn eli sovellusrajapinnan voi tehdä erittäin nopeasti Expressin avulla (43). MDN Web Docsien mukaan Express tekee helpoksi asettaa esimerkiksi palvelimen käyttämän portin verkkoyhteyttä varten. Sen ansiosta on myös helppoa luoda eri tapoja käsitellä eri http-metodikutsuja (esimerkiksi mainitut GET tai POST) ja erikseen käsitellä eri URL-polkuja eli reittejä vastaavat pyynnöt. (44.)

Esimerkkikoodissa 5 näkyy, kuinka vain kahdella koodirivillä voidaan luoda haluttua URL-polkua vastaavaan GET-pyyntöön oma erillinen käsittelytapansa. Tässä esimerkissä käsittelytapana on lähettää callback-funktiolla vastauksena pyyntöön pyynnön polusta saadut parametrit userId ja bookId. Expressin dokumentaation mukaan palvelin kuuntelee pyyntöjä, jotka vastaisivat pyynnön URL-polkua ja http-metodia. Kun sellainen pyyntö tulee, sitä vastaavaa callback-funktiota kutsutaan. Tässä esimerkkikoodissa callback-funktio sisältää vastausmetodi res.send():in, joka voi lähettää erityyppisiä vastauksia asiakasohjelman pyyntöön ja samalla se lopettaa pyyntö-vastaus-kierroksen. Expressissä on myös express.Router-luokka eri URL-reittejä vastaavien pyyntöjen käsittelyä varten. (45.)

```
app.get('/users/:userId/books/:bookId', (req, res) => {
  res.send(req.params)
})
```

Esimerkkikoodi 5. Expressissä voi yksinkertaisesti luoda halutuille http-pyynnöille ja URL-reiteille erilliset käsittelytavat. (45.)

Lisäksi Expressissä on muita hyödyllisiä ominaisuuksia. Express-kehys huolehtii myös väliohjelmistoista (englanniksi middleware). Wikipedian mukaan väliohjelmisto on sovelluksien osien tai eri sovellusten välinen ohjelma, palvelu tai ra-

japinta, joka toteuttaa eri toiminnallisuuksia (46). Geeks for Geeks -verkkosivulla kerrotaan, että Express toimii pyynnön ja sen saaman vastauksen välissä. Expressin väliohjelmistot toimivat väliohjelmistofunktioina, jotka suoritetaan sen jälkeen, kun palvelin saa pyynnön, mutta ennen vastauksen lähettämistä. Väliohjelmistolla on pääsy pyyntöön ja vastaukseen, ja se voi käsitellä vastausta ennen sen lähettämistä asiakasohjelmalle. (47.) Väliohjelmistoilla voidaan esimerkiksi parsia pyyntöjä, koota JavaScript-moduuleja tai tehdä mitä tahansa haluttua pyynnön ja vastauksen välissä (48).

4 React

React on selainpuolella käyttöliittymien rakentamiseen käytettävä JavaScript-kirjasto. Andrea Chiarelle kirjoittaa kirjassaan Beginning React, kuinka käyttöliittymän rakentaminen perustuu komponenttien yhdistämiseen (49). Myös käyttöliittymän ja muiden komponenttien tilanhallinta on olennainen osa Reactia. Tässä luvussa keskitytään Reactiin ja näihin peruskäsitteisiin.

4.1 Komponentit

Andrea Chiarelle kertoo, että Reactissa komponentit ovat eri käyttöliittymän eri osia, esimerkiksi lomakkeita, taulukoita, nappeja ja tekstejä. Useimmiten niitä voi käyttää uudelleen eri puolilla käyttöliittymää. Esimerkiksi samaa nappi-komponenttia voi käyttää koko käyttöliittymässä eri tekstillä. Reactin komponenteilla on omat käyttötarkoituksensa ja niillä voidaan esittää muuttuvaa dataa. Koko käyttöliittymä sekä koko React-sovellus ovat pienemmistä komponenteista koostuvia komponentteja. Komponentteja voidaan käyttää rinnakkain ja sisäkkäin kokoelmina komponentteja. (49.)

App-komponentti on Reactin pääkomponentti, joka määritellään App.js-tiedostossa React-sovelluksessa. Tavallisesti App-komponenttia käytetään sitomalla se id-tunnisteen ”root” sisältävään HTML-elementtiin index.html-tiedostossa. Tarkemmin sanottuna, React-sovelluksen lähtöpisteen sisältävässä index.js-tiedostossa App-komponenttia käytetään ReactDOM-olion render()-metodia, joka

sitoo App-komponentin ja HTML-komponentin kuvassa 5. ReactDOM-olio on React-kirjaston react-dom-moduulin olio, ja se yhdistää React-komponentit HTML:n DOM:iin eli dokumenttiobjekti-malliin render()-metodilla. Näin JavaScript-koodi saadaan yhdistettyä html-sivuun. Toinen tärkeä moduuli on react-moduuli, joka toimii komponenttien luomisessa ja tilanhallinnassa. (49.)

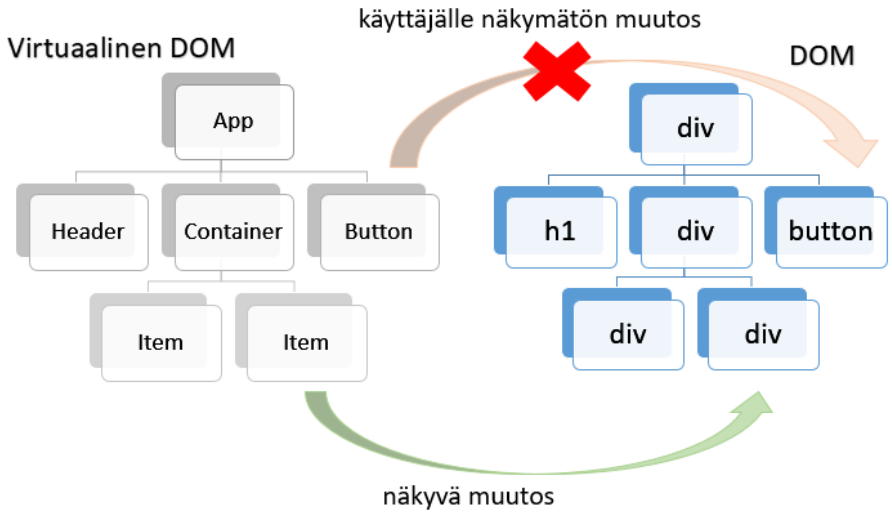
Kuva 5. ReactDOM:in render()-metodi yhdistää Reactin App-komponentin JavaScript-koodeineen osaksi HTML-rakennetta.

Mikä sitten on HTML:n DOM? DOM eli Document Object Model ”yhdistää verkkosivut skripteihin tai ohjelmointikieliin esittämällä dokumentin rakenteen – kuten HTML:n esittämän verkkosivun – muistissa”. Myös muita merkintäkieliä kuin HTML:ää voidaan käyttää muodostamaan dokumentin rakenne. DOM kuvaa dokumentin rakenteen loogisena puurakenteena, jossa puun oksat päättyvät objekteja sisältävään solmuun (node). DOM-verkkorajapinnan metodeilla pääsee muokkaamaan puun rakennetta tai sisältöä. (50.) Niitä voi käyttää osana JavaScript-koodia, ja näin koodilla voi muokata verkkosivun rakennetta ja sisältöä. Esimerkiksi ”document.getElementById("table")” -kutsu palauttaa koodin käsiteltäväksi olion, joka vastaa puun solmua, jonka id-ominaisuus on ”table”.

Tämä olio on DOM-hierarkiassa solmun Elementti-alarajapinnan (Element) tyyppinen olio. (51.) Elementti on perusluokka, josta kaikki dokumentin rakenteen elementtejä edustavat olio periytyvät (52). Esimerkiksi HTMLElement edustaa mitä tahansa dokumentin html-elementtiä DOM-puussa (53).

Andrea Chiarellen mukaan aiemmin mainittu render()-metodi määrittää komponentin ulkoasun. Jokaisessa React-komponentissa on render()-metodi, joka voi sisältää JavaScriptiä ja joka palauttaa merkintäkielisen kuvauksen komponentin ulkoasusta. Tämä merkintäkielinen kuvaus näyttää hyvin samalta kuin HTML, mutta se on kuitenkin JSX:ää (JavaScript Syntax Extension). JSX yhdistää JavasScriptin ja XML-merkintäkielen. Render()-metodin palauttama JSX-merkintäkielinen kuvaus sisältää yhden merkintäkielisen perusosan, mutta sen sisällä voi olla useampia JSX-perusosia. (49.)

JSX-kuvaus määrittelee objektin, jota kutsutaan elementiksi. Nämä react-elementit ovat samankaltaisia kuin yllämainitut DOM-elementit. React-komponentit tuottavat render()-metodilla JSX:n määrittelemiä React-elementtejä, jotka React-kirjaston moottori yhdistää muistissa oleviin DOM-elementteihin. React-elementit muodostavat virtuaalisen DOM:in, joka on kevytversio selaimen DOM:ista. React päivittää selaimen DOM:in vain silloin, kun se on tarpeellista, mikä tekee Reactin käyttöliittymän renderöinnistä tehokasta (kuva 6). (49.)



Kuva 6. Selaimen muistin DOM päivittyy virtuaalisen DOM:in mukaiseksi vain silloin, kun se on tarpeellista, esimerkiksi kun siinä tapahtuu käyttäjälle näkyvä muutos.

On lisäksi huomattava, että React-komponentin voi muodostaa luokkana tai funktiona. Jos React-komponentti on ohjelmoitu luokkana, se periytyy React.Component-luokasta ja sisältää render()-metodin. Jos React-komponentti on ohjelmoitu funktioksi, se joka tapauksessa palauttaa JSX-muotoisen ulkoasun kuvauksen aivan kuten render()-metodi.

Reactin dokumentaatio korostaa, että komponentit saavat sisälleen syötteenä tietoa. Se tapahtuu ominaisuuksien eli propsien avulla. Ne ovat tietoa, jotka halutaan esittää komponentin avulla tai käyttää komponentin muodostamiseen. Props-tiedot välitetään komponenttiin, mutta komponentti ei voi muuttaa niitä. Esimerkkikoodissa 6 Welcome-komponentilla on ominaisuus nimi. Kun Welcome-komponenttia halutaan käyttää, on määriteltävä arvo nimelle. Tässä tapauksessa nimelle annetaan arvo ”Sara” ja se välittyy komponenttiin. Käyttäjälle näkyy teksti ”Hello, Sara”. Komponentilla voi olla useita ominaisuuksia, esimerkiksi nimen lisäksi Welcome-komponentin ominaisuutena voisi olla esimerkiksi käyttäjätili tai käyttäjätiliin liitetty kuva. (54.)

```
function Welcome(props) {
  return <h1>Hello, {props.name}</h1>;
}
const root = ReactDOM.createRoot(document.getElementById('root'));
const element = <Welcome name="Sara" />;
root.render(element);
```

Esimerkkikoodi 6. Welcome-komponentti saa nimeä koskevalle propsille arvon ”Sara”. Koodissa näkyy myös funktiokomponentin palauttama JavaScriptiä ja XML-merkintäkieltä yhdistävän JSX-kuvaus. (54.)

4.2 Tilanhallinta

Propsien lisäksi React-komponteilla on toinen datankäsittelytapa eli tila. Tila on sisäänrakennettu React-komponentteihin. Se on kapseloitu datarakenne, johon asetetut tiedot säilyvät, vaikka komponentti renderöidään uudelleen. Kun käyt-

täjä käyttää sovelluksen käyttöliittymää, hän saattaa muuttaa jonkin komponentin tilaa. Esimerkiksi vaihtamalla taulukon datan aikaväliä, taulukon tila saattaa muuttua ja käyttäjä näkee eri tiedot. Tila vaikuttaa siihen, mitä käyttöliittymässä näkyy. Reactilla on omia keinoja hallita komponenttien tilaa. (55.)

Kirjassaan React in Action Mark Thomas kirjoittaa, että tilan voi määritellä myös kaikkena tietona, jonka sovellus sisältää jonakin hetkenä. Kun sovelluksen jotain muuttujaa muutetaan, sovelluksen tila muuttuu. Reactissa tila on tietoa, jota muutetaan komponentin sisällä. (56.) Hari Narayn lisää Just React!: Learn React the React Way -kirjassa, että tila on JavaScript-olio, jossa on tietoa komponentin tämänhetkisestä tilanteesta. Tavallisesti funktiolohkossa määritelty muuttuja katoaa funktiosta poistuttaessa, mutta Reactin funktio-komponenteissa tilamuuttujien senhetkiset arvot säilyvät. (57.)

Tärkein ero propseissa ja tilassa on se, että tila on komponentin sisällä ja komponentti voi muuttaa tilaansa (56). Propsit puolestaan välitetään komponentin ulkopuoliselta vanhemmalta, ja niitä ei voi muuttaa lapsikomponentin sisällä (58). Datarakenteet voidaan jakaa kahteen luokkaan: muuttumattomiin (immutable) ja muuttuviin (mutable). Muuttumattomassa datarakenteessa vanhaa arvoa ei voi korvata eli ylikirjoittaa uudella arvolla, mutta datasta voi olla tallennettuna useita versioita. Muuttuvassa datarakenteessa vanha data voidaan korvata uudella, mutta vanhasta datasta ei tallenneta versioita. Tila on muuttuva datarakenne, ja propsit ovat vain luettavissa eikä niitä pitäisi yrittää muuttaa. (56.)

Luokkakomponentit perivät tila-ominaisuuden React.Component-luokalta. Luokkakomponentin instanssilla on tila, ja siihen pääsee käsiksi instanssimuuttujan ”this.state” kautta. ”This.state” on olio, joka voi sisältää useita muuttujia. Jos tila-olioon liittyy esimerkiksi nimi tai päivämäärä, ne saadaan tietoon tulostamalla ”this.state.name” tai ”this.state.date”. Instanssimuuttujaa ”this.state” ei saa kuitenkaan muuttaa suoraan, vaan tilaa muutetaan aina setState()-metodin avulla. Metodia setState()-kutsutaan aina render()-metodin ulkopuolelta, esimerkiksi tapahtumankäsittelijäfunktiosta tai Reactin komponenttien lifecycle-metodeista, jotka voidaan suorittaa eri vaiheissa luokkakomponentin sykliä – esimerkiksi

heti renderöinnin jälkeen tai päivityksen yhteydessä. (56.) Kun setState()-metodia kutsutaan, tilan muutos tulee Reactin tietoon ja render-metodia kutsutaan, jotta DOM-puuta saadaan päivitettyä ja komponentti saadaan piirretyksi käyttäjälle uuden tilan mukaisesti (59).

Metodi setState() saa parametrikseen joko objektin tai funktion. Objekti voi sisältää esimerkiksi uuden nimen tai päivämäärän, joka halutaan näyttää käyttäjälle osana komponenttia. Jos tilassa on esimerkiksi nimi- ja päivämääräkentät, setState()-metodilla on mahdollista muuttaa vain toista tai molempia. Jos halutaan käyttää vanhaa tilaa pohjana uudelle tilalle, setState()-metodin parametrina voi käyttää funktiota, jolle voi antaa argumenttina vanhan tilan arvon. (59.)

Funktiona määritellyillä komponenteilla on poikkeava tapa käyttää ja muuttaa tilaa. Niissä käytetään Reactin koukkuja (hooks). Reactin koukut ovat JavaScript-funktioita, joilla voi eristää esimerkiksi tilalogiikan ja sivuvaikutukset itse komponentista (60). Koukkuihin liittyy kuitenkin muutama ylimääräinen sääntö. Koukkuja voi käyttää vain Reactin funktiokomponenteissa ja sielläkin vain ylimmällä tasolla. Koukkuja ei voi käyttää ehtolausekkeiden kanssa, funktiokomponentin sisällä olevissa funktioissa tai silmukoissa. (61.)

Funktiokomponentissa tilaan liittyvä useState-koukku korvaa luokkakomponentissa käytetyn this.state-instanssimuuttujan ja setState()-metodin, joita funktiokomponentissa ei voi käyttää. Koukku määrittää this.state-muuttujaa vastaavan tilamuuttujan, jolle voi antaa haluamansa nimen. Koukulle annetaan parametriksi alkutila ja se palauttaa tilan senhetkisen arvon ja sitä muuttavan funktion. (58.) Reactin dokumentaatiossa esimerkkikoodissa 7 määritetään count-niminen tilamuuttuja. Sen alkutila on 0, mutta kun tilaa muutetaan onClick-tapahtuman seurauksena useState-koukulla määritellyllä setCount-funktiolla, tila muuttuu. Tila saadaan esitettyä viittaamalla siihen count-tilamuuttujan nimellä. (62.)

```
function Example() {
const [count, setCount] = useState(0);
    return (
        <div>
```

```
            <p>You clicked {count} times</p>
            <button onClick={() => setCount(count + 1)}>
                Click me
            </button>
        </div>
    );
}
```

Esimerkkikoodi 7. Example-funktiokomponentissa count-tila saadaan näytettyä lisäämällä muuttuja kaarisulkeissa JSX-elementtiin. (62.)

Reactissa voi tehdä myös komponentteja, joissa ei käytetä propseja tai tilaa tai kumpaakaan. Lisäksi joissain tilanteissa tilan jakaminen komponenttien välillä voisi olla hyödyksi. Reactissa kuitenkin komponentti voi jakaa tilansa propseina vain lapsikomponenteilleen (58). On kuitenkin kirjastoja, joissa tilaa hallitaan eri tavoilla. Näistä esimerkkinä on seuraavaksi käsiteltävä Mobx.

4.3 Mobx

Mobx on tilanhallintaan keskittyvä kirjasto, joka käyttää funktionaalista reaktiivista ohjelmointia. Sitä käytetään usein Reactin kanssa, mutta sitä ei ole sidottu mihinkään arkkitehtuuriin. (63.) Se on helppo ottaa käyttöön npm install -komennolla (65). Mobx:n dokumentaation mukaan sen pääperiaatteena on pitää tila yhdenmukaisena hakemalla data automaattisesti suoraan sovellustilasta eli sovelluksen taulukoista, diagrammeista ja muuttujista (64). Sovellustilan datan voi säilöä mihin tahansa datarakenteisiin. Jos tätä dataa haluaa muuttaa, se on merkittävä havaittavaksi (observable), jotta Mobx pystyy havaitsemaan sen ja muuttamaan sitä. (65.)

Mobx:ssä havaittavaa tilaa muutetaan aina toiminnoilla (action). Kaikkien metodien, jotka muuttavat tilaa, on oltava toimintoja. Mobx:ssä voi tehdä tilan muutoksiin automaattisesti reagoivia johdoksia tai derivaatioita (derivations). Ne voivat olla tilan arvoista automaattisesti johdettuja tai laskettuja lisätietoja. Heti kun tilaa muutetaan, myös derivoitu arvo muuttuu. Ne ovat samankaltaisia kuin Mobx:n reaktiot (reactions), mutta reaktiot eivät kuitenkaan tuota johdettua arvoa. (65.) Reaktiot suorittavat automaattisesti jonkin tehtävän eli esimerkiksi

päivittävät DOM-puun tai tekevät pyynnön palvelimelle (64). Kun toiminta muuttaa tilaa, myös tilaan liittyvät derivoidut arvot muuttuvat ja reaktiot suorittavat tehtäviä automaattisesti. Näin data liikkuu toiminnoista tilojen ja derivoitujen arvojen kautta reaktioihin yhteen suuntaan. (65.)

Toinen suosittu tilanhallinta kirjasto on Redux. Front-end-kehittäjä Rajitha Abeyrathna vertailee Mobx:iä ja Reduxia. Molemmat kirjastot ovat sopivia siinä tilanteessa, kun sovelluksesta alkaa tulla iso ja monimutkainen. Redux perustuu funktionaaliselle ohjelmoinnille ja Flux-arkkitehtuurille. (66.) Flux on Facebookin käyttämä arkkitehtuuri, jossa data myöskin kulkee yhteen suuntaan toiminnoista (action) lähettäjän (dispatcher) kautta varastoon (store) ja eteenpäin näkymiin (view) (67).

Molemmat Redux ja Mobx käyttävät varastoja. Varasto on JavaScript-olio, jonka avulla komponentit voivat jakaa tilansa (68). Redux käyttää yhtä keskitettyä varastoa koko sovellustilan tallentamiseen. Mobx:ssä voi käyttää monia pieniä varastoja. Reduxin tila varastossa on muuttumatonta, ja tarvitaan reducer-funktioita muuttamaan tila. (67.). Reducer-funktio ottaa parametrikseen vanhan tilan ja toiminnon (action) ja luo vanhasta tilasta uuden kopion, johon on tehty halutut muutokset (69). Mobx:ssa tila on muuttuvaa. Redux käyttää tavallisia JavaScript-olioita tilan säilyttämiseen, kun taas Mobx tekee tiloista havaittavia ja sitä kautta niiden muutokset huomataan automaattisesti, mikä helpottaa tilanhallintaa. (67.)

Mobx:iä on käytetty myös hiilijalanjälkilaskurin toteutuksessa. Seuraavassa osiossa keskitytään hiilijalanjälkilaskurin toteutukseen.

5 Sovelluksen toteutus

Hiilijalanjälkilaskurin toteutuksen kuvaus on jaettu kahteen osaan. Ensiksi käsitellään sitä, miten palvelinpuoli on toteutettu Ilmastodieetin rajapinnan ja Node.js:n avulla. Sen jälkeen käsittelyssä on selainpuolen toteutus Reactilla.

Käyttöliittymän ja siihen haluttujen ominaisuuksien – helppokäyttöisyys, taulukot tulosten esittelyssä ja pelillinen elementti – toteutus käydään läpi.

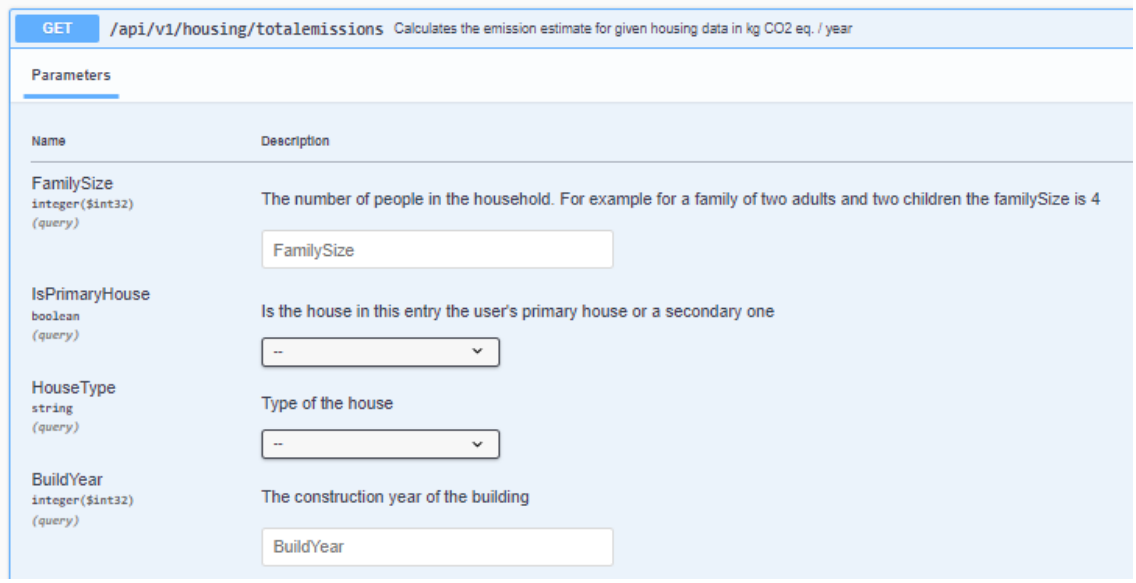
5.1 Palvelinpuoli

Sovelluksen palvelinpuoli huolehtii selainpuolelta tuleviin pyyntöihin vastaamisesta. Pyyntöjen mukaan palvelin hakee dataa Ilmastodieetin rajapinnasta ja välittää saamansa vastauksen asynkronisesti takaisin selainpuolelle näytettäväksi käyttöliittymässä.

5.1.1 Ilmastodieetin API

Ilmastodieetin API on avoin rajapinta, jonka avulla voi laskea, kuinka paljon hiilidioksidipäästöjä ihmisen elämänvalinnat aiheuttavat. Se on pohjana Ilmastodieetti-laskurille, mutta sitä voi käyttää kuka tahansa osana sovellustaan. Sitä voi kutsua GET- ja POST-metodeilla käyttämällä haluttua polkua. Esimerkiksi ”https://rajapinnat.ymparisto.fi/api/ilmastodieetti/api/v1/housing/totalemissions?”-polku on alku, johon on vielä lisättävä tarvittavat parametrit, jotta käyttäjä saa tietää asumistaan vastaavat hiilidioksidipäästöt. Tätä hyödyntävä kutsu on mukana myös insinöörityössä toteutetussa hiilijalanjälkilaskurissa.

Ilmastodieetti tarjoaa myös Swagger-käyttöliittymän rajapinnalleen. Kuvassa 7 näkyy esimerkkinä asumiseen liittyvä GET-kutsu. Swaggerissa kutsulle voi helposti liittää parametrit lomakkeen kenttiin, joten rajapintaa voi helposti manuaalisesti testata omia käyttötarkoituksiaan varten.



Kuva 7. Ilmastodieetti-rajapinnan asumiseen liittyvä GET-kutsu Swagger-käyttöliittymässä. (70)

5.1.2 Palvelin

Hiilijalanjälkilaskurin palvelimessa olennaista oli saada käyttäjän syötteet käyttöliittymästä selainpuolelta ja hakea niiden perusteella syötteitä vastaavat hiilidioksidipäästöt Ilmastodieetin avoimesta rajapinnasta. Tämän toteutukseen käytetyistä teknologioista on jo kerrottu aiemmin, mutta tärkeimpinä olivat siis Node.js, sen kehys Express ja moduuli Axios. Näitä käyttämällä on siis pyrkimys luoda selainpuolen asiakasohjelman pyyntöihin vastaava backend.

React-projektin voi luoda ”npx create react-app carbonfootprintcalc” -komennolla. Lisäämällä sen sisälle server-kansion, joka sisältää index.js-tiedoston, saa palvelimen jo alkuun. Index.js-tiedostossa on ensimmäisenä käytettävät Node.js-moduulit. Ne asennetaan ”npm install” -komennolla komentorivillä ja index.js-tiedostossa käytetään ”require”-tunnussanaa moduulien tuomiseksi osaksi palvelinta (esimerkkikoodi 8).

```
const express = require('express');
const app = express();
const cors = require('cors');
app.use(cors());
const dataRoute = require('../server/routes/data.routes')
app.use('/data', dataRoute)
```

Esimerkkikoodi 8. Palvelimen index.js:ssä käytettävät Node.js-moduulit.

Moduuleista tehdään oliot asettamalla ne muuttujien arvoiksi. Käyttöön otetaan Express-kehys ja siitä tehdään app-olio. Metodilla use sidotaan moduulien lisätoiminnallisuuksia osaksi app-oliota.

Myös CORS-väliohjelmisto (cross origin resource sharing) ladataan use-metodin avulla. ”CORS on http-tunnisteisiin perustuva mekanismi, jonka avulla palvelin voi osoittaa kaikki alkuperät (verkkotunnukset, portit, skeemat), joista selaimen pitäisi sallia resurssien lataamisen” (71). Tavallisesti selaimelle on sallittu vain saman alkuperän resurssien käyttäminen, ja muista alkuperistä saatavien resurssien käytön salliminen on tehtävä sovelluksen palvelinpuolella (72). Esimerkkikoodissa 8 esitetyssä CORS-väliohjelmiston käyttöönotossa palvelin sallii selaimelle resurssien lataamisen kaikista alkuperistä. Tällä tavalla sallitaan sovellukselle resurssien eli hiilidioksidipäästödatan hakeminen Ilmastodieetin rajapinnasta. Jos dataa tarvitsisi hakea myös muista alkuperistä uusien toiminnallisuuksien kehittämisen yhteydessä, se olisi mahdollista jo tällä koodilla. Lisäksi on huomattava, että cors()-väliohjelmistofunktiota kutsutaan aina, kun palvelin saa minkä tahansa http-kutsun, koska sen kutsumista varten ei ole annettu polkua use-metodin parametrina.

Palvelimen oma http-kutsujen reititystä varten tehty moduuli, tiedosto data.route.js, ladataan, ja sekin toimii väliohjelmistona. Kuitenkin use-metodilla määritetään, että tämän moduulin toiminnallisuutta käytetään vain silloin, kun tulee ”/data”-polkua vastaava http-pyyntö. Tiedostoon data.route.js täytyy laittaa ”module.exports = router;” -koodirivi, jotta tiedoston käyttö muualla onnistuu.

Lisäksi palvelimen index.js-tiedostossa on vielä asetettava palvelimelle portti kuuntelemaan yhteydenottoja. Esimerkkikoodissa 9 portiksi asetetaan 4000 listen()-metodin avulla.

```
const port = 4000;
app.listen(port, () => {
    console.log(`Server listening at ${port}`);
});
```

Esimerkkikoodi 9. Palvelin kuuntelee portilla 4000, tuleeko yhteydenottoja.

Esimerkkikoodista 10 näkee, että tiedostossa data.route.js luodaan reititin http-kutsuille. Router-luokan instanssin luominen tekee oikeastaan reitityksestä ”minisovelluksen”, sillä se on itsessään kokonainen väliohjelmisto, jolla voidaan tehdä modulaarisia reittejä (45).

```
let router = express.Router();
```

Esimerkkikoodi 10. Tiedostossa data.route.js http-kutsujen reiteille määritellään omat käsittelijät, jotka sisältävät uudet http-kutsut toiselle rajapinnalle.

Expressin use()-metodin sijaan reittien kanssa voi käyttää route()-metodia, johon voisi myös niputtaa useita http-metodinkäsittelyjä (73). Esimerkkikoodissa 11 siihen ei ole tarvetta, ja metodi toimii hyvin näinkin hakemaan tiedot siitä, kuinka paljon hiilidioksidipäästöjä vastaa käyttäjän ilmoittamia kuukautisia ostoksia. Route()-metodi saa parametrikseen http-pyynnön pelkistetyn polun. Selainpuolella pyynnön polussa on mukana koneen oma verkko-osoite ja ”/data/”-välihakemisto, sillä sovellusta ei ole julkaistu missään. Polussa pyynnön parametrit seuraavat sitä edeltävää määrittelevää sanaa: esimerkiksi clothingPrice-sanaa seuraa käyttäjän selaimessa syöttämä vaatteisiin käyttämä rahamäärä. GET-metodi osoittaa, että pyynnöllä halutaan saada dataa muualta, ja pyyntöä varten annetaan sille käsittelijäfunktio parametrina. Käsittelijäfunktiolla on pääsy pyyntöolioon ja siihen saatavaan vastaukseen.

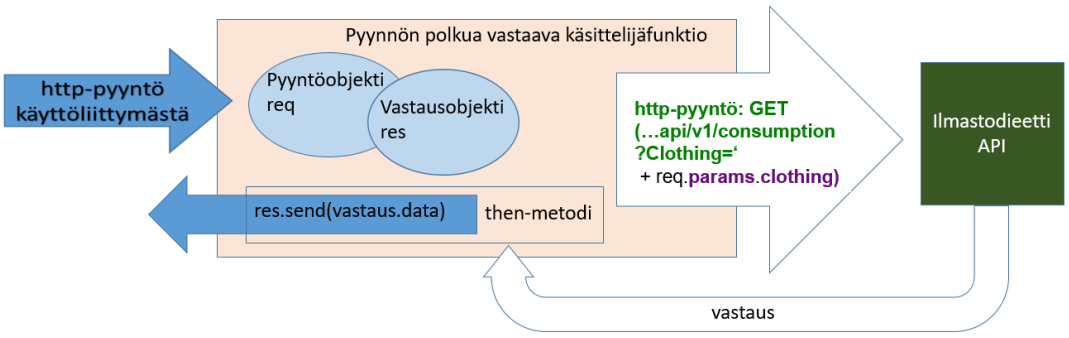
```
router.route(
    '/purchases/clothingPrice/:clothing/communicationsPrice/:inter
net/electronicsPrice/:electronics/paperPrice/:pa
per/shoesPrice/:shoes')
    .get((req, res) => {
    let purchasesFromIlmastodieetti = axios.get(
        'https://rajapinnat.ymparisto.fi/api/ilma
stodeetti/api/v1/cosumption?Clothing=' + req.params.clothing

        + '&Communications=' + req.params.internet
        + '&Electronics=' + req.params.electronics
        + '&Other=0'
        + '&Paper=' + req.params.paper

        + '&Recreation=0'
        + '&Shoes=' + req.params.shoes)
        .then((response) => {
            console.log("purchases " + response.data)
            res.send(JSON.stringify(response.data))
        }).catch((error) => {
            console.log(error)
        })
});
```

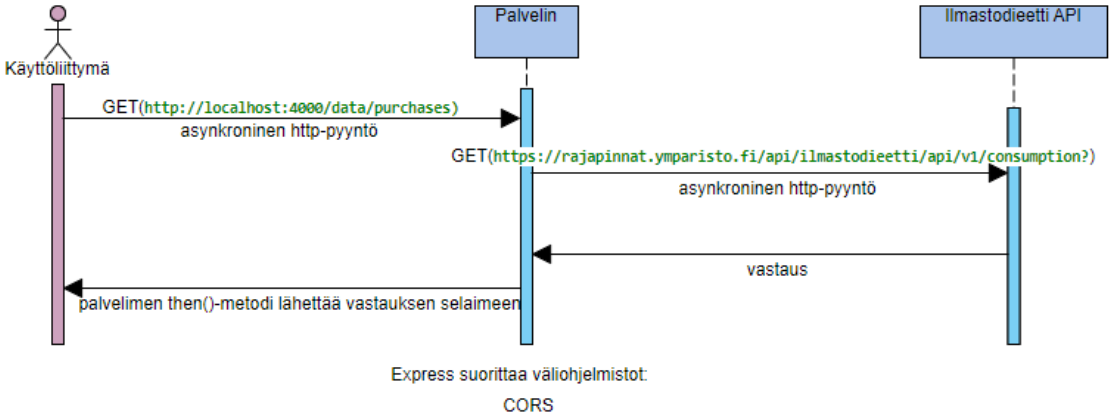
Esimerkkikoodi 11. Tiedostossa data.route.js http-kutsujen reiteille määritellään omat käsittelijät, jotka sisältävät uudet http-kutsut toiselle rajapinnalle.

Käsittelijäfunktio on asynkroninen Axiosilla toteutettu http-kutsu Ilmastodieettirajapinnalle. Siinä halutaan GET-metodilla hakea parametrina olevaa polkua vastaavat tiedot käyttäjän ostoksiin liittyvistä hiilidioksidipäästöistä. Käsittelijäfunktio pääsee käsiksi käyttöliittymästä tulleen pyyntöolion tietoihin, jotka löytyvät osana polkua. Se onnistuu req.params-objektin avulla, ja esimerkiksi req.params.clothingPrice kertoo edelleen, kuinka paljon rahaa käyttäjä kertoo käyttöliittymässä kuluttaneensa vaatteisiin kuukaudessa. Se voidaan lisätä Ilmastodieettiin tehtävään http-pyyntöön muiden polusta saatavien tietojen kanssa. Kuva 8 esittää, kuinka käsittelijäfunktio saa parametreina käyttöönsä pyyntöolion req ja vastausolion res. Osaa Ilmastodieetin API:n haluamista tiedoista ei kysytä käyttöliittymässä, joten niiden kohdilla arvoiksi kovakoodataan nolla.



Kuva 8. Selainpuolelta tulevan http-pyynnön käsittelevä käyttää Expressin pyyntö- ja vastausobjekteja lisätiedon hakemiseen Ilmastodieetti-rajapinnasta ja http-pyyntöön vastaamiseen.

Käsittelijäfunktio on luonnollisesti asynkroninen, koska ei ole varmaa, kuinka nopeasti Ilmastodieetti-rajapinta pystyy vastaamaan pyyntöön. Sen takia käsittelijäfunktio palauttaa lupauksen, mikä näkyy then()-metodissa. Kun vastaus lopulta saadaan, vastauksen rungon data tulostetaan konsoliin, muutetaan JSON-merkkijonoksi, ja res.send-lähettää sen käyttöliittymälle. Jos tulee virhetilanne, silloin virhe tulostetaan konsoliin. Kuva 9 esittää, miten palvelin saa http-pyynnön selainpuolelta ja käsittelee sen uudella http-pyynnöllä Ilmastodieetti-rajapinnalle.



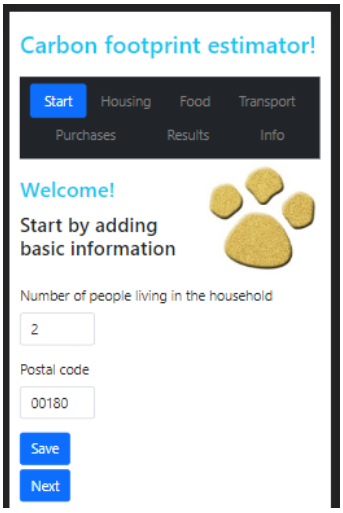
Kuva 9. Palvelin käsittelee käyttöliittymästä saadun pyynnön lähettämällä uuden pyynnön Ilmastodieetti-rajapinnalle.

Jotta saadaan tietoon käyttäjän elämän muiden osa-alueiden aiheuttamat hiilidioksidipäästöt, osa-alueita varten on omat http-kutsunsa käyttöliittymästä ja niitä vastaavat reitit sekä reittikohtaiset käsittelijäfunktiot palvelimessa. Asumisella, kulkuvälineillä liikkumisella, autolla ja ruoalla on myös omat http-pyyntönsä käyttöliittymästä palvelimeen ja palvelimesta Ilmastodieetti-rajapintaan.

5.2 Selainpuoli

Selainpuolella on toteutettu pyynnöt palvelimelle sekä käyttöliittymä ja sen tilanhallinta. Käyttöliittymässä on kyselylomakkeita käyttäjän elintavoista, käyttäjän syötteiden perusteella lasketut tulokset hiilidioksidipäästöistä ja lyhyt infoteksti sovelluksesta.

Käyttöliittymän näkymästä toiseen siirrytään React Routerin avulla. Reactiin ei kuulu reititystä näkymien vaihtamiseksi, mutta sitä varten voi asentaa react-router-dom-moduulin (74). React Routerissa polkua vastaa näytettävä komponentti, esimerkiksi polkua ”/” vastaa App-komponentti (kuva 10).



Kuva 10. Aloitussivun sisältö on toteutettu App-komponentissa.

Esimerkkikoodissa 12 näkyy React Routerin käytön lisäksi, kuinka käyttöliittymän JSX-komponentit ja JavaScript koodi yhdistetään render-metodilla root-ele-

menttiin ja sitä kautta html-rakenteeseen ja DOM:iin. Ulommaisena React-komponenttina on React Routerin BrowserRouter-komponentti usein näin käytetyn App-komponentin sijaan.

```
ReactDOM.render(
  <React.StrictMode>
      <BrowserRouter>
          <Routes>
            <Route path="/" element={<App />}>
                <Route path="/" element={<BasicInfo />} />
                <Route path="/housing" element={<Housing />} />
                <Route path="/food" element={<Food />} />
                <Route path="/transport" element={<Transport />} />
                <Route path="/purchases" element={<Purchases/>} />
                <Route path="/results" element={<Results />} />
                <Route path="/information" element={<Information />}
/>
            </Route>
          </Routes>
      </BrowserRouter>
  </React.StrictMode>,
  document.getElementById('root')
);
```

Esimerkkikoodi 12. Index.tsx-tiedostossa polut yhdistetään React-elementteihin React Routerin avulla.

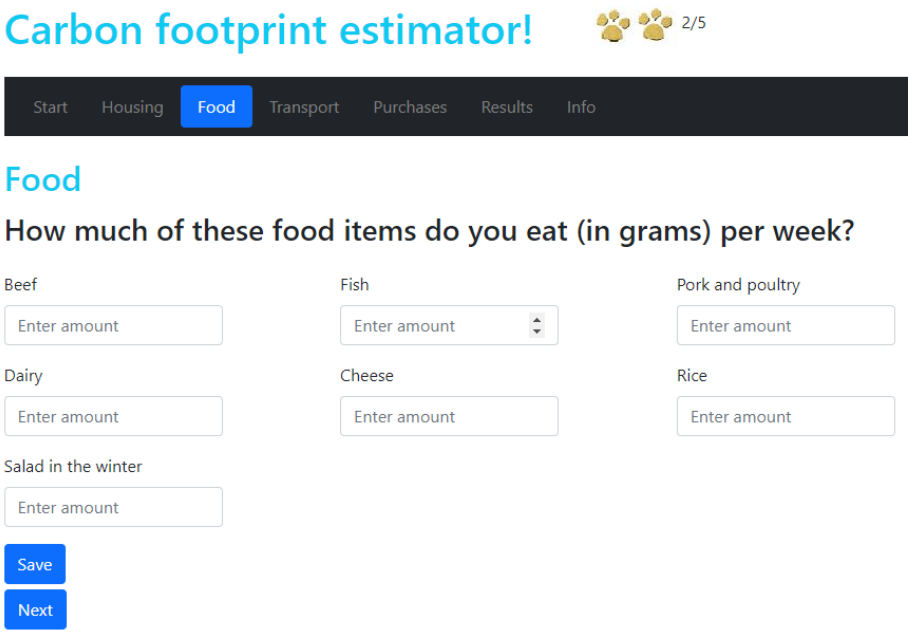
Käyttöliittymän ulkoasun toteutukseen on myös käytetty Bootstrap-kirjastoa ja React-Bootstrap-komponentteja. React Bootstrapissa Bootstrapin luokat on korvattu valmiilla React-komponenteilla, joita voi käyttää kuin itse tehtyjä komponentteja (75). Niille voi esimerkiksi antaa ominaisuuksia eli propseja. Esimerkkikoodissa 13 käytetään valmista Form.Group-komponenttia lomakkeen esittämiseen. Form.Group-komponentti saa propseina esimerkiksi luokan nimen, syötteen minimiarvon ja käyttäjän syötteen käsittelyä varten metodin. Siinä kysytään käyttäjällä summaa, jonka hän käyttää kenkien ostoon kuukausittain. Bootstrapin ja React-Bootstrapin avulla sovellus skaalautuu käytettäväksi myös tableteilta ja mobiililaitteilta.

```
<Form.Group className="mb-3" controlId="formPurchasesShoes">
    <Form.Label>Shoes</Form.Label>
    <Form.Control className="width-25px" type="number" place-
holder="Enter amount" min={0} value={this.state.shoes} on-
Change={this.handleShoesChange}/>
</Form.Group>}
```

Esimerkkikoodi 13. React-Bootstrap-komponentteja käytetään osana kyselylomaketta.

5.2.1 Kyselylomakkeet

Hiilijalanjälkilaskurissa on viisi sivua kyselylomakkeita käyttäjän elintavoista. Ensimmäinen kyselylomake on sovelluksen aloitussivu, ja siinä kysytään taloudessa asuvien ihmisten määrää ja postinumeroa. Toinen kyselylomake koskee asumista, kolmas ruokavaliota (kuva 11), neljäs matkustamista ja viides tavaroita ja hankintoja. Ensimmäisen sivun yleisiä kysymyksiä käytetään etenkin asumiseen liittyvien hiilidioksidipäästöjen laskemiseen. Kyselylomakkeet ovat napakankokoisia, ja tietokoneen tai kannettavan näytöltä katsoessa niitä ei tarvitse selailla ylös alas. Ensimmäinen kyselylomake sivu on toteutettu App-funktiokomponenttina. Muut kyselylomakkeet on toteutettu luokkakomponentteina.



Kuva 11. Ruokaan liittyvä kyselylomake on toteutettu luokkakomponenttina.

Kukin näistä neljästä viimeisestä kyselylomakkeesta on oma luokkakomponenttinsa, ja niillä on omat tilansa. Jokaista lomakeruutua vastaa käsittelijäfunktio, ja kun ruutu täytetään, käsittelijäfunktio asettaa annetun syötteen tilan muuttujaan talteen. Esimerkkikoodi 14:ssa punaista lihaa koskeva syöte käsitellään asettamalla syöte beef-tilan arvoksi. Luokkakomponenteissa käsittelijäfunktiot on sidottava konstruktorissa bind()-metodilla komponenttiin, jotta esimerkkikoodin 14 ”this” viittaa oikeaan komponenttiin ja kontekstiin (76).

```
handleBeefChange(event: React.ChangeEvent<HTMLInputElement>) {
        this.setState({beef: event.target.value});
    }
```

Esimerkkikoodi 14. Funktio käsittelee syötetapahtuman, ja komponentin tila muuttuu syötteen mukaiseksi. Luokkakomponentissa tilan muuttamisessa käytetään setState-metodia this-sanan kanssa.

Kun käyttäjä on täyttänyt lomakkeen ja tallentaa syötteensä klikkaamalla tallennusnappia, klikkaustapahtumaa vastaavaa käsittelijäfunktiota kutsutaan. Sen sisällä Mobxiä käyttävä ResultState-luokan resultStore-olio kutsuu esimerkiksi matkustusta koskevassa lomakkeessa asynkronista metodia loadTransportResults, joka pyytää palvelimelta matkustamiseen liittyvää hiilijalanjälkeä Axios-kirjaston avulla (esimerkkikoodi 15). Palvelin vastaavasti kutsuu Ilmastodieetti-rajapintaa, saa lopulta vastauksen ja välittää sen selainpuolelle. Metodi loadTransportResults sisältää Mobx-funktion runInAction, joka muuttaa sen sisällä olevan koodin toiminnoksi. Koska Mobx:ssä tilaa on muutettava aina toiminnon sisällä, on joko käytettävä runInAction-funktiota tai erillistä toimintoa tilan muuttamiseksi (77). RunInAction-funktion sisällä muutetaan resultStoren tilaa ja vastauksena saatu hiilijalanjälki asetetaan osaksi sitä suoraan ilman erillistä setState-funktiota.

```
@action
loadTransportResults = async (state: any) => {
     try {
          const promise = await axios.get(
               'http://localhost:4000/data/transport/bus/' + (state.bus
               ? state.bus : '0')
               + '/train/' + (state.train ? state.train : '0')
               + '/flights/' + (state.flights ? state.flights : '0')
               + '/cruises/' + (state.cruises ? state.cruises : '0')
          )
          runInAction(() => {
               this.resultsTransport = promise.data
          })
     } catch (e) {
          console.error(e);
     }
}
```

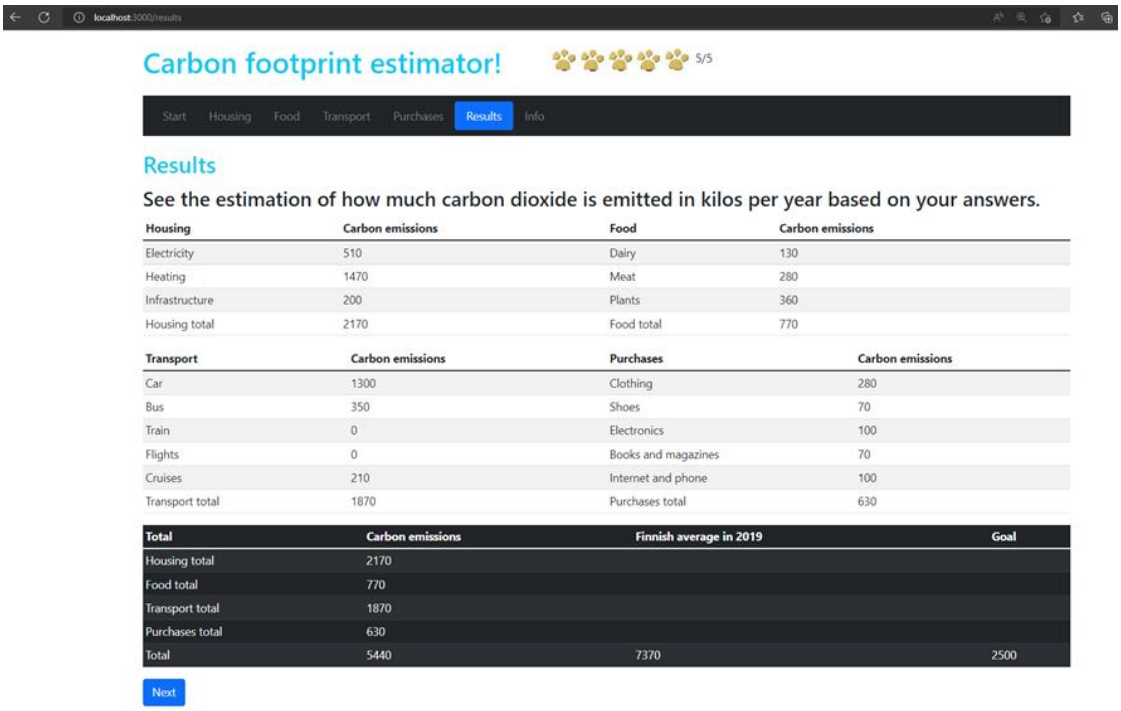
Esimerkkikoodi 15. Funktio lähettää asynkronisesti pyynnön palvelimelle matkustamiseen liittyvästä hiilijalanjäljestä. Parametriksi funktio saa käyttäjän lomakkeeseen tilana tallennetut käyttäjän syötteet. Lupauksena saatu vastaus tallennetaan osaksi resultStoren tilaa.

Jos hiilidioksidipäästötulokset tallennettaisiin erillisten kyselylomakekomponenttien tiloihin, niihin olisi vaikea päästä käsiksi, kun tulokset halutaan esittää erillisessä tulossivukomponentissa. Tila on periaatteessa komponentin sisäinen, mutta käyttämällä varastoa (store) tila voidaan jakaa komponenttien välillä. ResultStore-varaston avulla käyttäjän syötteet ja niiden perusteella Ilmastodieetin API:ssa laskettavat tulokset saadaan koottua samaan paikkaan. Lisäksi tulokset saadaan näytettyä niille varatussa omassa komponentissaan, kun niitä ei tallennetta kyselylomakekomponenttien tiloiksi.

5.2.2 Tulokset

Hiilijalanjälkitulokset esitetään React-Bootstrapin valmiiden taulukko-komponenttien avulla erillisessä Results-komponentissa. Tulokset on jaoteltu aihealueiden perusteella, ja ne vastaavat kyselylomakkeita. Niissä on myös eritelty kunkin aihealueen sisältö eri osiin, esimerkiksi ruokaa koskeva hiilijalanjälki on jaoteltu maitotuotteiden, lihan ja kasvisten hiilidioksidipäästöihin (kuva 12). Eri aihealueiden hiilidioksidipäästöjä on mahdollista vertailla toisiinsa. Käyttäjä näkee taulukosta, millä asioilla on suurin hiilijalanjälki. Samalla hän voi pohtia, mitä oman elämänsä osa-aluetta olisi helpointa muuttaa ja millä muutoksilla olisi

suurin vaikutus hiilijalanjäljen pienentämiseen. Taulukon lopussa näkyy myös suomalaisten keskimääräinen kokonaishiilijalanjälki vuonna 2019 (7370 kg/vuosi) sekä ideaalitavoite kokonaispäästöille (2500 kg/vuosi). Näiden lukujen lähteinä on käytetty Sitran elämäntapalaskuria (78).



Kuva 12. Tulokset esitetään taulukkoina.

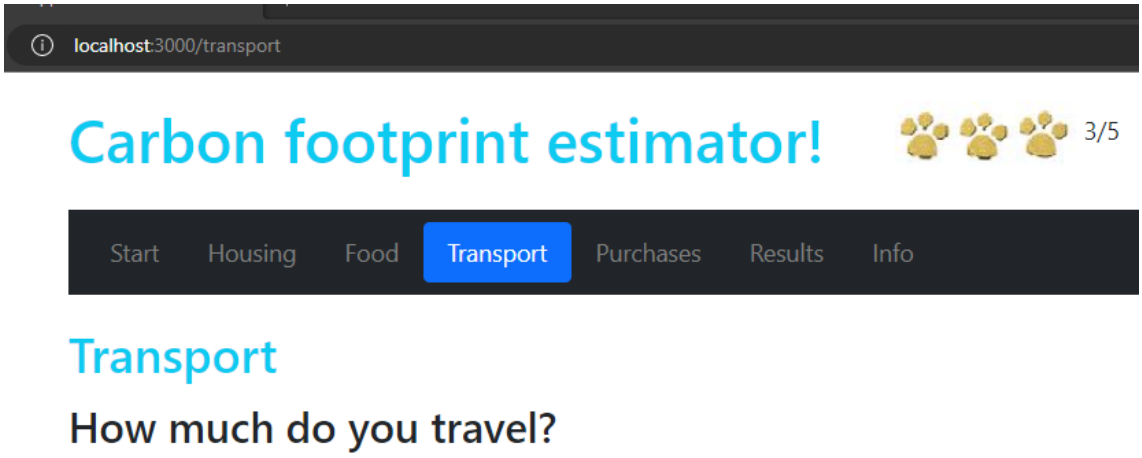
Tulokset on asetettu resultStore-olion tilaan http-kutsujen vastausten saannin jälkeen ja ne esitetään esimerkkikoodin 16 mukaisesti taulukossa.

```
<td>Electricity</td>
<td>{!!resultStore.resultsHousing.electricity
     ? Math.round(resultStore.resultsHousing.electricity / 10) * 10
     : "0"
}</td>
```

Esimerkkikoodi 16. Mikäli resultStore-olion electricity-muuttuja ei ole tyhjä, hiilidioksidiarvo pyöristetään ja esitetään taulukossa.

5.2.3 Pelillinen elementti

Pelillinen elementti on yksinkertaisesti kultaiset tassunjäljet, joita käyttäjä saa näkyville, kun hän etenee kyselylomakkeissa, kuten näkyy kuvassa 13. Heti kun täyttää aloitussivun ensimmäiset kysymykset ja siirtyy seuraavaan lomakkeeseen, saa ensimmäisen tassunjäljen. Tulokset-sivulla käyttäjä saa viidennen ja viimeisen tassunjäljen.



Kuva 13. Käyttäjä saa edetessään lisää kultaisia tassunjälkiä.

Tassunjäljet ovat erillisiä kuvia. Riippuen siitä, missä kohdassa käyttöliittymää käyttäjä on, näytetään käyttäjän edistystä vastaava kuva otsikon vieressä App-komponentissa. Esimerkkikoodissa 17 näkyy, että käyttäjän kulloinenkin olinpaikka tallentuu Reactin omalla useState()-koukulla tilan arvoksi. Alkuarvona on aloitussivua vastaava polku ”/”, mutta polun ja link-tilan muuttuessa setLink-funktiota kutsuttaessa tassunjälkien kuva vaihtuu ja käyttäjä saa lisää tassunjälkiä palkinnoksi laskurin täytön jatkamisesta.

```
const [link, setLink] = useState("/");
```

Esimerkkikoodi 17. Tassunjälkien määrä riippuu siitä, mikä polku vastaa käyttäjän olinpaikkaa.

6 Lopuksi

Insinöörityössä toteutui tavoitteena ollut hiilijalanjälkilaskuri ja tutustuminen verkkosovellusten toteuttamisen perusperiaatteisiin. Erityisesti tarkastelun kohteena olivat Node.js:n ja Reactin toiminta ja niihin liittyvät olennaiset käsitteet.

Käyttämällä hyväksi todettuja teknologioita ja tutustumalla niiden pääpiirteisiin oli mahdollista toteuttaa hiilijalanjälkilaskuri. Node.js:n asynkroninen tapahtumasilmukka mahdollisti tehokkaan palvelimen luonnin. Myös http-pyyntöjen lähettäminen Axiosin ja Expressin avulla sujuvoitti sovelluksen ohjelmointia ja auttoi nopeassa ja toimivassa kommunikaatiossa sovelluksen selainpuolen, palvelinpuolen ja Ilmastodieetin avoimen rajapinnan välillä datan saamiseksi käyttäjän näkyville. Lisäksi Reactin komponentteihin perustuvalla oli mahdollista luoda napakka, helppokäyttöinen ja dynaaminen käyttöliittymä. Reactin komponenteilla ja Mobx:n avulla Ilmastodieetistä saatu data esitettiin taulukoissa, jotka mahdollistivat selkeän vertailun. Lisäksi Reactin useState-koukkua käyttämällä mahdollistui pelillisen elementin käyttäminen eli tassunjälkien näyttäminen käyttäjän etenemisen mukaan.

Sovellusta voisi kehittää edelleen. Testauksen jälkeen sovellus olisi mahdollista julkaista ja ottaa yleiseen käyttöön Internetissä esimerkiksi Heroku- tai Netlify-alustalla. Lisäominaisuutena sovellukseen voisi myös liittää tietokannan ja käyttäjätilit, jolloin omat tulokset voisi tallentaa. Tällöin kuitenkin täytyisi olla tarkkana henkilötietojen säilyttämisen kanssa GDPR-asetuksen mukaisesti.

Toisena lisäkehityslinjana olisi myös mielenkiintoista laajentaa sovellusta laskemaan yksittäisen ihmisen hiilijalanjäljen sijaan perheiden, yhteisöjen, yritysten ja esimerkiksi kuntien hiilijalanjälkiä. Isojen yhteisöjen päätöksillä on usein enemmän vaikutusta kuin yksittäisen ihmisen Tallinnan matkalla tai talvisalaatin syönnillä. Datan avulla yhteisötkin saisivat tietoa vaikutuksestaan ilmastonmuutokseen ja voisivat tiedon perusteella muuttaa ainakin joitain osa-alueita toiminnassaan.

On tietysti kyseenalaista, miten paljon hyötyä yhdestä uudesta hiilijalanjälkilaskurista olisi. Ilmastonmuutoksen torjuminen on valtava monimutkainen pyrkimys, jonka saavuttamiseksi tarvitaan tekniikkaa ja ihmisiä.

Lähteet

1 Health and Environment Linkages Initiative. 2022. Verkkoaineisto. World Health Organization. <https://www.who.int/heli/risks/climate/climatechange/en/>. Luettu 25.2.2022.

2 Climate change and health. 2021. Verkkoaineisto. World Health Organization. <https://www.who.int/news-room/fact-sheets/detail/climate-change-and-health>. 30.10.2021. Luettu 25.2.2022.

3 Climate change impacts. 2021. Verkkoaineisto. National Oceanic and Atmospheric Administration. <https://www.noaa.gov/education/resource-collections/climate/climate-change-impacts>. 13.8.2021. Luettu 25.2.2022.

4 Torrent Tucker, Danielle. 2021. Climate change has caused billions of dollars in flood damages. Verkkoaineisto. Stanford Earth. <https://earth.stanford.edu/news/climate-change-has-caused-billions-dollars-flood-damages#gs.qzxu73>. 11.1.2021. Luettu 25.2.2022.

5 Lindsey, Rebecca, Dahlman Luann: 2022. Climate Change: Global Temperature. Verkkoaineisto. NOAA Climate.gov. <https://www.climate.gov/news-features/understanding-climate/climate-change-global-temperature>. 28.6.2022. Luettu 2.10. 2022.

6 What are the solutions to climate change? Verkkoaineisto. Greenpeace. <https://www.greenpeace.org.uk/challenges/climate-change/solutions-climate-change/>. Luettu 8.10.2022.

7 Climate change: Seven technology solutions that could help solve crisis. Verkkoaineisto. Sky News. <https://news.sky.com/story/climate-change-seven-technology-solutions-that-could-help-solve-crisis-12056397>. 12.10.2021. Luettu 8.10.2022.

8 Meenakshisundaram, Ashwinram. How can data sharing help combat climate change? Verkkoaineisto. Support Centre for Data Sharing. <https://eudatasharing.eu/news/how-can-data-sharing-help-combat-climate-change>. 6.7.2022. Luettu 8.10.2022.

9 Allwood, Julian: 2022. Technology will not solve the problem of climate change. Verkkoaineisto. Financial Times. <https://www.ft.com/content/207a8762-e00c-4926-addd-38a487a0995f>. 16.11.2021. Luettu 8.10. 2022.

10 Deutskens, Elisabeth, de Ruyter, Ko, Wetzels, Martin, Oosterveld, Paul. 2004. Response Rate and Response Quality of Internet-Based Surveys: An Experimental Study. Verkkoaineisto. Marketing Letters. <

https://www.researchgate.net/publication/5152940_Response_Rate_and_Response_Quality_of_Internet-Based_Surveys_An_Experimental_Study>. 1.2.2004. Luettu 4.3.2022.

11 Hartt, Maxwell, Hosseini, Hadi, Mostafapour, Mehrnaz. 2020. Game On: Exporing the Effectiveness of Game-based Learning. Verkkoaineisto. Planning Practice & Research. < https://web-s-ebscohost-com.ezproxy.metropolia.fi/ehost/pdfviewer/pdfviewer?vid=3&sid=5acf390c-0ff2-48a8-8cf1-cc7c905a2a77%40redis>. Lokakuu 2020. Luettu 3.10.2022.

12 HTTP. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP>. Päivitetty 13.9.2022. Luettu 9.10.2022.

13 HTTP. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/HTTP>. Päivitetty 6.8.2021. Luettu 9.10.2022.

14 TCP. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/TCP#cite_note-ien2-2>. Päivitetty 28.5.2022. Luettu 9.10.2022.

15 IP. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/IP>. Päivitetty 8.10.2022. Luettu 9.10.2022.

16 TypeScript is JavaScript with syntax for types. Verkkoaineisto. Wikipedia. <https://www.typescriptlang.org/>. Luettu 9.10.2022.

17 Software Framework vs Library. Verkkoaineisto. Geeks for Geeks. <https://www.geeksforgeeks.org/software-framework-vs-library/>. Päivitetty 11.7.2022. Luettu 9.10.2022.

18 Surve, Suraj. Why You Should Use React.js For Web Development. Verkkoaineisto. freeCodeCamp. <https://www.freecodecamp.org/news/why-use-react-for-web-development/> 18.2.2021. Luettu 9.10.2022.

19 Laszczak, Grzegorz. Why Choose Node.js? Verkkoaineisto. Medium. < https://medium.com/selleo/why-choose-node-js-b0091ad6c3fc> 26.1.2021. Luettu 9.10.2022.

20 Runtime Environment (RTE). Verkkoaineisto. Techopedia. <https://www.techopedia.com/definition/5466/runtime-environment-rte> 30.6.2020. Luettu 16.10.2022.

21 Patel, Priyesh. What exactly is Node.js?. Verkkoaineisto. Medium. <https://medium.com/free-code-camp/what-exactly-is-node-js-ae36e97449f5> 18.4.2018. Luettu 16.10.2022.

22 Syed, Basarat Ali, Bean, Martin. Beginning Node.js. E-kirja. Apress. <https://learning.oreilly.com/library/view/beginning-node-js/9781484201879/9781484201886_Ch02.xhtml> 2014. Luettu 17.10.2022.

23 Callback function. Verkkoaineisto. MDN Web Docs Glossary. <https://developer.mozilla.org/en-US/docs/Glossary/Callback_function> Päivitetty 21.9.2022. Luettu 18.10.2022.

24 Asynchronous. Verkkoaineisto. MDN Web Docs Glossary. <https://developer.mozilla.org/en-US/docs/Glossary/Asynchronous> Päivitetty 21.9.2022. Luettu 18.10.2022.

25 Emy, Njong. Asynchronous JavaScript – Callbacks, Promises, and Async/Await Explained. Verkkoaineisto. FreeCodeCamp. <https://www.freecodecamp.org/news/asynchronous-javascript-explained/> 20.6.2022. Luettu 23.10.2022.

26 Using Promises. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises> Päivitetty 13.9.2022. Luettu 24.10.2022.

27 Promise. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise> Päivitetty 24.9.2022. Luettu 24.10.2022.

28 Promise.prototype.then(). Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise/then> Päivitetty 3.10.2022. Luettu 24.10.2022.

29 await. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await> Päivitetty 11.10.2022. Luettu 24.10.2022.

30 async function. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function> Päivitetty 13.9.2022. Luettu 24.10.2022.

31 Kadam, Sarvesh. How Modular Programming Works in Node.js. Verkkoaineisto. FreeCodeCamp. <https://www.freecodecamp.org/news/modular-programming-nodejs-npm-modules/> 17.11.2020. Luettu 25.10.2022.

32 Evans, PJ. Node.js in Motion. Verkkoaineisto. Manning Publications. <https://learning.oreilly.com/videos/node-js-in-motion/10000MNLV201720/> 2018. Luettu 25.10.2022.

33 Agarwal, Mayank. Changing the npm start-script of Node.js. Verkkoaineisto. TutorialsPoint. <https://www.tutorialspoint.com/changing-the-npm-start-script-of-node-js> 20.5.2021. Luettu 25.10.2022.

34 Getting started. Verkkoaineisto. Axios. <https://axios-http.com/docs/intro> 20.5.2021. Luettu 25.10.2022.

35 An overview of HTTP. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview> Päivitetty 13.9.2022. Luettu 25.10.2022.

36 Uniform Resource Identifier. Verkkoaineisto. Wikipedia. <https://en.wikipedia.org/wiki/Uniform_Resource_Identifier> Päivitetty 23.10.2022. Luettu 25.10.2022.

37 McCann, Jacoby. Axios Library - HTTP Requests. Verkkoaineisto. Medium. <https://medium.com/@JacobyMcCann/axios-library-http-requests-a52a522b6f1c> 23.6.2022. Luettu 25.10.2022.

38 What is RESTful API? Verkkoaineisto. AWS. <https://aws.amazon.com/what-is/restful-api/> Luettu 25.10.2022.

39 XMLHttpRequest. Verkkoaineisto. JavaScript.Info. <https://javascript.info/xmlhttprequest> 14.5.2022. Luettu 25.10.2022.

40 XML. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/XML/> Päivitetty 1.8.2020. Luettu 25.10.2022.

41 Why Axios Can Be Used Both in Browser and Node.js. Verkkoaineisto. Medium. <https://javascript.plainenglish.io/why-axios-can-be-used-both-in-browser-and-node-js-5b88206f70f> 11.7.2022. Luettu 25.10.2022.

42 Express web framework (Node.js/JavaScript). Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs> Päivitetty 9.9.2022. Luettu 25.10.2022.

43 Smilga, John. Node.js and Express.js - Full Course. Verkkoaineisto. Youtube. <https://www.youtube.com/watch?v=Oe421EPjeBE> 1.4.2021. Katsottu 25.10.2022.

44 Express/Node introduction. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction#summary> Päivitetty 13.9.2022. Luettu 25.10.2022.

45 Routing. Verkkoaineisto. Express. <https://expressjs.com/en/guide/routing.html> Luettu 25.10.2022.

46 Väliohjelmisto. Verkkoaineisto. Wikipedia. <https://fi.wikipedia.org/wiki/V%C3%A4liohjelmisto> Päivitetty 10.4.2022. Luettu 30.10.2022.

47 Middleware in Express.js. Verkkoaineisto. Geeks for Geeks. <https://www.geeksforgeeks.org/middleware-in-express-js/> Päivitetty 19.7.2022. Luettu 30.10.2022.

48 Express.js Middleware. Verkkoaineisto. JavaTPoint. <https://www.javatpoint.com/expressjs-middleware> Luettu 30.10.2022.

49 Chiarelle, Andrea. Beginning React. Verkkokirja. Packt Publishing. < https://learning.oreilly.com/library/view/beginning-react/9781789530520/> 2018. Luettu 30.10.2022.

50 Document Object Model (DOM). Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model> Päivitetty 9.9.2022. Luettu 30.10.2022.

51 Introduction to the DOM. Verkkoaineisto. MDN Web Docs. < https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction> Päivitetty 29.9.2022. Luettu 6.11.2022.

52 Element. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/API/Element > Päivitetty 29.9.2022. Luettu 6.11.2022.

53 HTMLElement. Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/API/HTMLElement> Päivitetty 4.11.2022. Luettu 6.11.2022.

54 Components and Props. Verkkoaineisto. React. <https://reactjs.org/docs/components-and-props.html> Luettu 7.11.2022.

55 Gupta, Versha. React state management: What is it and why to use it?. Verkkoaineisto. Loginradius. <https://blog.loginradius.com/engineering/react-state-management/> Luettu 6.11.2022.

56 Thomas, Mark. React in Action. E-kirja. Manning Publications. <https://learning.oreilly.com/library/view/react-in-action/9781617293856/kindle_split_014.html> 2018. Luettu 12.11.2022.

57 Narayn, Hari. Just React!: Learn React the React Way. E-kirja. Apress. <https://learning.oreilly.com/library/view/just-react-learn/9781484282946/html/521019_1_En_3_Chapter.xhtml> 2022. Luettu 12.11.2022.

58 State. Verkkoaineisto. Hands on React. <https://hand-sonreact.com/docs/state> Luettu 12.11.2022i.

59 State and Lifecycle. Verkkoaineisto. React. <https://reactjs.org/docs/state-and-lifecycle.html> Luettu 12.11.2022.

60 Adhikary, Tapas. React Hooks Fundamentals for Beginners. Verkkoaineisto. FreeCodeCamp. <https://www.freecodecamp.org/news/react-hooks-fundamentals/> 15.3.2022. Luettu 12.11.2022.

61 Hooks at a Glance. Verkkoaineisto. React. <https://reactjs.org/docs/hooks-overview.html> Luettu 12.11.2022.

62 Using the State Hook. Verkkoaineisto. React. <https://reactjs.org/docs/hooks-state.html> Luettu 12.11.2022.

63 README. Verkkoaineisto. MobX. <https://mobx.js.org/README.html> Luettu 6.11.2022.

64 MobX Ten minute introduction to MobX and React. Verkkoaineisto. MobX. <https://mobx.js.org/getting-started> Luettu 13.11.2022.

65 The gist of MobX. Verkkoaineisto. MobX. <https://mobx.js.org/the-gist-of-mobx.html> Luettu 13.11.2022.

66 Abeyrathna, Rajitha. Redux VS MobX | Comparison. Verkkoaineisto. Medium. <https://medium.com/emblatech/redux-vs-mobx-comparison-d510c956ae27> 15.7.2021. Luettu 13.11.2022.

67 In-Depth Overview. Verkkoaineisto. Flux. <https://facebook.github.io/flux/docs/in-depth-overview/> Päivitetty 6.3.2022. Luettu 13.11.2022.

68 React stores. Verkkoaineisto Learn. <https://learn.co/lessons/react-stores/> Luettu 13.11.2022.

69 Ceddia, Dave. Immutability in React and Redux: The Complete Guide. Verkkoaineisto. Dave Ceddia. <https://daveceddia.com/react-redux-immutability-guide/#what-is-immutability/> 17.9.2018. Luettu 13.11.2022.

70 Ilmastodieetti API. Verkkoaineisto. <https://rajapinnat.ymparisto.fi/api/ilmastodieetti/index.html> Luettu 6.11.2022.

71 Cross-Origin Resource Sharing (CORS). Verkkoaineisto. MDN Web Docs. <https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS> Luettu 6.11.2022.

72 Understanding and Resolving CORS Error. Verkkoaineisto. Contentstack. <https://www.contentstack.com/docs/developers/how-to-guides/understanding-and-resolving-cors-error/> Luettu 6.11.2022.

73 Router. Verkkoaineisto. Express. <https://expressjs.com/en/4x/api.html#router> Luettu 6.11.2022.

74 React Router. Verkkoaineisto. W3Schools. <https://www.w3schools.com/react/react_router.asp> Luettu 13.11.2022.

75 React Bootstrap. Verkkoaineisto. React Bootstrap. <https://react-bootstrap.github.io/> Luettu 14.11.2022.

76 Varanasi, Gohit. Why React Event handlers Bind methods with “this”. Verkkoaineisto. Medium. <https://medium.com/weekly-webtips/why-react-event-handlers-bind-methods-with-this-51b121e0fad> 25.6.2020. Luettu 14.11.2022.

77 Mobx - runInAction() usage. Why do we need it?. Stackoverflow. <https://stackoverflow.com/questions/57271153/mobx-runinaction-usage-why-do-we-need-it> Päivitetty 19.1.2021. Luettu 14.11.2022.

78 Testaa, oletko uhka vai mahdollisuus. Verkkoaineisto. Sitra. <https://elamantapatesti.sitra.fi/> Päivitetty 19.3.2021. Luettu 14.11.2022.