Karelia-ammattikorkeakoulu
Tradenomi (AMK)
Tietojenkäsittely

Virtual showroomin front-endin toteutus A-Framea ja Reactia käyttäen

Jani Härkönen

Opinnäytetyö, syyskuu 2022

Karelia AMMATTIKORKEAKOULU	**OPINNÄYTETYÖ** **Syyskuu 2022** **Tietojenkäsittelyn koulutus** Tikkarinne 9 80200 JOENSUU +358 13 260 600

Tekijä(t)
Jani Härkönen

Nimike
Virtual showroomin front-endin toteutus A-Framea ja Reactia käyttäen

Toimeksiantaja
Karelia-ammattikorkeakoulu, 6190 Pohjois-Karjalan ja Japanin Naganon metsäbiotalousyhteistyön kehittäminen

Tiivistelmä

Tietotekniikan sekä yhteiskunnan digitalisaation kehittyessä yritykset ja organisaatiot kehittävät web-sivujensa graafista esitystä erilaisilla innovaatioilla, kuten virtuaalisilla esittelyillä (virtual showroom). Tässä opinnäytetyössä keskitytään Karelia-ammattikorkeakoululle toteutetun virtuaaliesittelyn front-endin sovellusarkkitehtuurin tärkeimpien komponenttien kuvaamiseen, ja niiden vertailuun vaihtoehtoisiin ratkaisuihin. Vertailussa keskitytään koodin selkeyteen sekä skaalautuvuuteen. Vertailun pohjalta oli tarkoitus muodostaa kehitysehdotuksia toimeksiantajalle ja mahdollisille jatkokehittäjille.

Työssä käsitelty virtuaaliesittely toteutettiin osana Pohjois-Karjalan ja Japanin Nagano-prefektuurin välistä yhteistyöhanketta. Sivusto koostui 360°-panoraamanäkymästä, johon aseteltiin hankkeeseen kuuluvien metsäbiotalousalan organisaatioiden klikattavia tietopaneeleja. Esittely ohjelmoitiin JavaScriptillä React-sovelluskehystä ja A-Frame-kirjastoa käyttäen.

Toteutettu sivusto vastasi pääasiassa toimeksiantajan asettamia vaatimuksia, mutta ohjelmistokomponenttien arkkitehtuureissa oli paljon puutteita skaalautuvuuden osalta. Verrokkiratkaisuja tarkastelemalla löytyi useita kirjastoja sekä menetelmiä, joita hyödyntäen koodin selkeyttä ja skaalautuvuutta voitaisiin parantaa.

Kieli	Sivuja
suomi	42

Asiasanat
virtual showroom, React, A-Frame, Javascript

Karelia
UNIVERSITY OF APPLIED SCIENCES

THESIS
September 2022
Degree Programme in Business Information Technology

Tikkarinne 9
FI 80200 JOENSUU
FINLAND
Tel. +350 13 260 600

Author(s)
Jani Härkönen

Title
Development of a Virtual Showroom Using A-Frame and React

Commissioned by
Karelia University of Applied Sciences, Forest Bioeconomy Development Co-operation (FBDC) 2020 - 2022 between North Karelia Region in Finland and Nagano Prefecture in Japan

Abstract

As information technology advances and society becomes more digital, companies and organizations are looking to improve the graphical representation of their websites with innovations such as virtual showrooms. In this thesis, the architectural components of a virtual showroom developed for the Karelia University of Applied Sciences are described and compared to alternative designs. The comparison focuses on the clarity and scalability of the code. The findings were used to form development suggestions for the commissioner as well as potential future developers.

The virtual showroom examined in the thesis was developed as a part of a cooperation initiative between North Karelia and Nagano prefecture of Japan. The website consisted of a 360°-panorama view containing clickable information panels for the organizations involved in the initiative. The showroom was programmed using JavaScript in conjunction with React-framework and A-Frame-library.

The website was in line with the requirements set by the commissioner. However, there were many issues regarding the scalability of the software components. The comparison showed that the alternative solutions utilized several different libraries and design patterns that could be used to improve the clarity as well as the scalability of the code base of the Karelia project.

Language	Pages
Finnish	42

Keywords
virtual showroom, React, A-Frame, JavaScript

1 Johdanto

Yhteiskunnan digitalisaation kiihtyessä, osittain koronapandemian vaikutuksesta, yritykset ja organisaatiot pyrkivät kehittämään web-sivujensa graafista esitystä erilaisilla innovaatioilla. Laitteiden ja selainten prosessointitehon kasvaminen sekä internetyhteyksien tiedonsiirtonopeuksien kehittyminen ovat mahdollistaneet teknisesti monimutkaisten web-sivujen, kuten virtuaaliesittelyjen (virtual showroom), hyödyntämisen kaupallisissa tarkoituksissa. Virtuaaliesittelyt ovat verkkosivuja, jotka hyödyntävät interaktiivisia 360°-panoraamanäkymiä organisaatioiden, tuotteiden ja palveluiden esittelyyn selainympäristössä.

Opinnäytetyössä käsitellään Karelia-ammattikorkeakoululle kehitettyä virtuaalista esittelyä. Esittelyn kehityksen valmistuttua huomasin useita mahdollisia kehityskohteita sivuston arkkitehtuurissa, ja halusin paneutua syvällisemmin virtuaaliesittelyjen toteutuksiin sekä vaihtoehtoisiin arkkitehtuuriratkaisuihin. Työn tavoitteena on käsitellä esittelyn front-endin – eli graafisen käyttöliittymän – eri elementtien sovellusarkkitehtuuria, tuoda esille vaihtoehtoisia toteutusratkaisuja sekä muodostaa kehitysehdotuksia arkkitehtuurin skaalautuvuuden parantamiseksi. Käsiteltävät elementit on valittu sen perusteella, kuinka yleisiä ne ovat virtuaalisissa esittelyissä. Jokaisen elementin kohdalla käydään läpi Karelian toteutuksessa käytetty arkkitehtuuri ja vaihtoehtoiset arkkitehtuuriratkaisut. Vaihtoehtoisia ratkaisuja analysoidaan ja verrataan Karelian toteutukseen. Ratkaisujen pohjalta pyritään tekemään kehitysehdotuksia Reactilla ja A-Framella tehdyn projektin jatkokehitystä varten.

Seuraavassa osiossa käsitellään tarkemmin Karelialle toteutettua sivustoa ja sen ominaisuuksia, kehityksessä käytettyjä menetelmiä ja sovelluskehyksiä sekä havainnollistetaan virtuaaliesittelyn käsitettä tarkemmin esimerkein. Viimeinen osio sisältää lyhyen yhteenvedon elementtien verrokkiratkaisuihin perustuvista arkkitehtuurin kehitysehdotuksista.

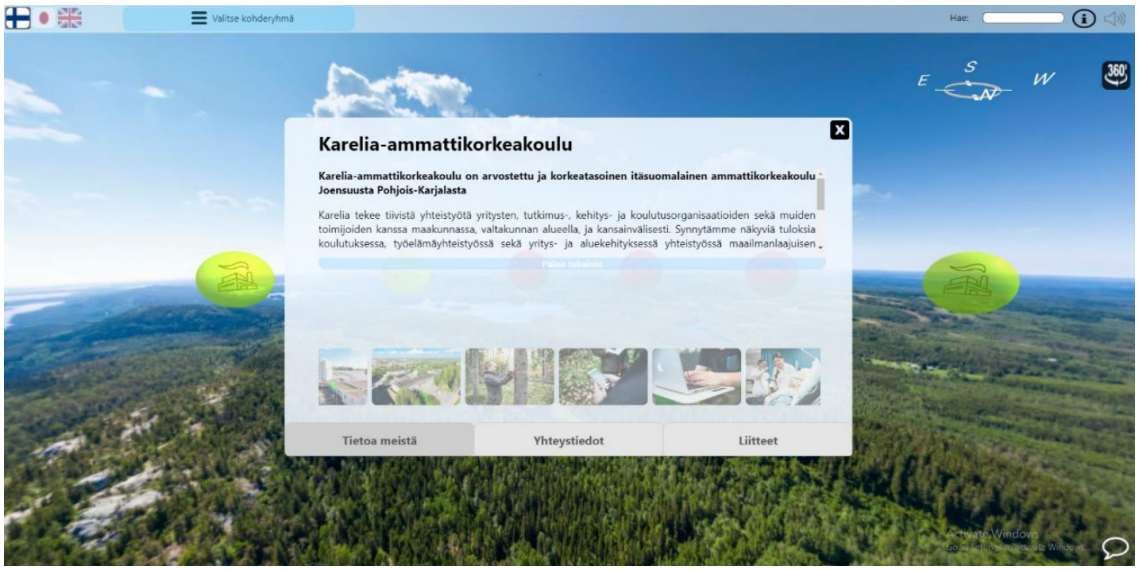
2 Sivuston ja toteutusmenetelmien kuvaus

2.1 Toteutuksen yleiskuvaus

Tarkastelun kohteena on virtuaaliesittely, joka toteutettiin Karelia-ammattikorkeakoululle osana Pohjois-Karjalan ja Japanin Nagano-prefektuurin välistä yhteistyöhanketta. Virtuaaliesittely on käsitteenä melko uusi eikä virallista määritelmää ole, mutta tyypillistä esittelyille on 360°-maisemakuva, jota käyttäjä voi tarkastella eri kulmista kameraa kääntäen. Lisäksi ympäristöön on yleensä asetettu vuorovaikutteisia elementtejä, esimerkiksi tietopaneeleja. Virtuaaliesittelyitä käytetään tyypillisesti tuotteiden, palveluiden tai yritysten esittelyyn. Esimerkiksi Mercedes-Benz (2020) hyödyntää virtuaalista ympäristöä Saudi-Arabiassa sijaitsevan autoliikkeen esittelyyn. Esittelyssä käyttäjä voi katsella ympärilleen hiirtä vetämällä ja liikkua ympäristössä siirtymispainikkeita klikkaamalla. Jokaisen liikkeessä olevan auton viereen on asetettu painike, jota klikattaessa avautuu auton esittelyvideon sisältävä tietoruutu. Avattu paneeli asettuu kolmeulotteisen näkymän päälle samoin kuin tavallinen HTML-elementeillä toteutettu modaali-ikkuna. (Mercedes-Benz 2020.)

Hankkeessa tehtävänä oli toteuttaa virtuaalinen esittely-ympäristö Pohjois-Karjalan alueella toimiville metsäbiotalousalan organisaatioille. Esittely-ympäristössä jokaisella organisaatiolla olisi oma, Koli-vaaran maisemakuvaan asetettu, esittelypallo (kuva 1). Palloa klikatessa avautuisi organisaation laajennettu esittely (kuva 2), joka sisältäisi organisaation lyhyen kuvauksen, kuvagallerian, yhteystiedot, sijainnin karttanäkymässä sekä mahdollisia ylimääräisiä liitetiedostoja ja -linkkejä. Sivusto oli tarkoitettu toimivaksi työpöytä- sekä mobiililaitteilla ja suunniteltu käännettäväksi pääasiassa kolmelle kielelle: suomeksi, englanniksi ja japaniksi. Lopullisessa toteutuksessa oli myös sivuston toimintaa selittävä opasteikkuna, asiakaspalvelu-chat sekä valikko ja hakukenttä organisaatioiden suodatusta varten. Opinnäytetyössä keskitytään 360°-näkymän, karttanäkymän ja kielituen toteutukseen sekä syvien tietorakenteiden yleiseen arkkitehtuuriin.

Kuva 1. Virtuaaliesittelyssä näytillä olevien organisaatioiden klikattavat esittelypallot on aseteltu panoraamamaisemaa vasten. Kuvassa on näkyvillä myös Karelia-ammattikorkeakoulun logo ja yhteystiedot (Kuva: Huusko 2022).



Kuva 2. Organisaation esittelypalloa klikattaessa avautuva laajennettu esittely (Kuva: Huusko 2022).

Sivusto kehitettiin käyttäen React-sovelluskehystä ja A-Frame-kirjastoa. React on käyttöliittymien kehitykseen suunniteltu avoimen lähdekoodin sovelluskehys, jota kehittää ja ylläpitää Meta (Facebook 2022). Mozillan MozVR-tiimin kehittämän web-pohjaisten virtuaalikokemusten rakentamiseen tarkoitetun A-Framen lähdekoodi on myös avoin (Mozilla Mixed Reality 2015). Esittelyn 360°-näkymä ja sen sisäiset vuorovaikutteiset elementit toteutettiin A-Framella, joka oli integroitu toimimaan Reactin kanssa. Käyttöliittymän kaksiulotteiset elementit toteutettiin Reactilla. Kehityksessä käytettiin rajatummin myös muita kirjastoja,

kuten OpenLayers-karttaratkaisua sekä React-kehitykseen suunniteltua Styled-components-kirjastoa (OpenLayers 2022; Styled-components 2022).

Kehitystyö toteutettiin Scrum-kehitysmenetelmää hyödyntäen; projektia kehitettiin pienin askelin iteraatioissa – sprinteissä – joiden päätteeksi tehtyjä muutoksia esiteltiin toimeksiantajalle. Sprinttien työtehtävät suunniteltiin ja merkittiin aina iteraation alussa Microsoftin (2022) Azuren DevOps-palveluun, josta kehitystiimin jäsenet sekä toimeksiantaja pystyivät seuraamaan tehtävien edistymistä. Projektin lopuksi sivuston arkkitehtuurista laadittiin kaavio sekä dokumentaatio, joissa kuvattiin tiedonkulkua ohjelmistokomponenttien välillä.

2.2 React-sovelluskehys

React on Metan kehittämä ja ylläpitämä avoimen lähdekoodin JavaScript-sovelluskehys, joka on suunniteltu selainpohjaisten sovellusten ja web-sivujen front-end-kehitykseen (React 2022a). React hyödyntää toiminnassaan komponenttipohjaisuutta, eli sovellusten näkymien nähdään koostuvan itsenäisistä, uudelleenkäytettävistä osista (React 2022b). Esimerkiksi näkymä on komponentti, joka sisältää toisia komponentteja, kuten näppäimiä. Kaikki näppäimet toimivat samalla logiikalla, mutta sisältävät erilaista tekstiä ja suorittavat erilaisia toimintoja. Kaikki näppäimet kuitenkin piirtävät painikkeen ja tekstin sekä suorittavat komennon käyttäjän klikatessa. React-sovelluksen käynnistyessä sovelluskehys piirtää kehittäjän toteuttamat komponentit selainikkunaan. React-sovelluksessa on juurikomponentti, joka renderöityessään piirtää kaikki sen sisällä olevat kehittäjän toteuttamat komponentit.

Reactissa kehittäjä voi määritellä kahdenlaisia komponentteja, luokkapohjaisia tai funktiopohjaisia. Karelian virtuaaliesittelyssä käytettiin funktiopohjaisia komponentteja. Funktiokomponentit ovat käytännössä JavaScript-funktioita, jotka ottavat syötteeksi komponentin toiminnan kannalta oleellista tietoa. Funktio palauttaa React-elementin, jonka sovelluskehys piirtää selainikkunaan (kuva 3). Palautetun React-elementin koostumukseen ja toimintaan vaikuttaa tyypillisesti funktiolle syötetyt arvot (kuva 4). React-sovelluksessa oleva juurikomponentti

(kuva 5) kutsuu renderöityessään kehittäjän toteuttamat funktiokomponentit, jotka palauttavat React-elementtinsä renderöitäväksi. (React 2022b.)

```
1  import React from "react"
2
3  export default function HelloWorldComponent(props) {
4      return(
5          <div>{props.message}</div>
6      )
7  };
```

Kuva 3. HelloWorldComponent palauttaa div-tagin, jonka sisälle on asetettu App-komponentin message-kenttään laittama teksti. Message-kentän tekstiin viitataan “props”-parametrilla.

```
1  import HelloWorldComponent from "./HelloWorldComponent";
2
3  function App() {
4    return (
5        <HelloWorldComponent message="Hello world" />
6    );
7  }
8
9  export default App;
```

Kuva 4. Juurikomponentin kutsuma App-funktiokomponentti palauttaa React-elementin, jonka React piirtää. App-komponentti sisältää HelloWorldComponentin, jolle syötetään ”message”-kenttä. HelloWorldComponent hyödyntää message-kenttää toiminnassaan.

```
1  import React from 'react';
2  import ReactDOM from 'react-dom/client';
3  import './index.css';
4  import App from './App';
5
6  const root = ReactDOM.createRoot(document.getElementById('root'));
7  root.render(<App />);
```

Kuva 5. Juurikomponentti, joka sisältää tyypillisesti yhden komponentin, jonka alle muut komponentit asettuvat.

Komponenteilla on yleensä tila (state), joka niiden on tallennettava muistiin. Esimerkiksi käyttäjän selatessa näkymiä ohjelman on muistettava, mitä näkymää piirretään ruudulle. Funktiokomponenteissa hyödynnetään usein useState-koukkuja (hook), joilla React ylläpitää komponenttien käyttämiä muuttujia (kuva 6). Koukku on funktio, jolla ohjelmistokomponentti lähettää tietoa rajapinnan takana olevalle komponentille. Tässä tapauksessa koukut lähettävät tietoa

Reactille. Koukku päivittää rajapinnan takana olevan komponentin laukaisten samalla tyypillisesti ylimääräisiä muutoksia ohjelmistossa, esimerkiksi näkymän uudelleenpiirron. Kun komponentin tila muuttuu useState-funktiota kutsuessa, esimerkiksi käyttäjän klikatessa näppäintä, React renderöi komponentin ja sen sisällä olevat komponentit uudelleen. Tila on tallennettava Reactin koukkuja käyttäen, sillä kaikki funktiokomponentissa määritellyt muuttujat toimivat samoin kuin normaaleissa JavaScript-funktioissa; muuttujat toimivat vain funktion rakenteen sisällä ja ovat olemassa vain funktion suorituksen ajan. Komponenttia renderöidessä uudelleen sen funktio kutsutaan jälleen, jolloin aiemmassa suorituksessa määritellyt muuttujat katoavat arvoineen. (React 2022c.)

```
1   import React, { useState } from "react";
2
3   export default function Example(props) {
4       const [someState, setSomeState] = useState(1);
5
6       const someFunction = () => {
7           setSomeState(someState + 1);
8       };
9   };
10
```

Kuva 6. UseState-koukku sisällytetään ensin React-kirjastosta. Tila asetetaan määrittelemällä ensin muuttuja, johon tila asetetaan (someState) sekä asettajafunktio (setSomeState), jolla tila voidaan muuttaa. SomeStaten tila on komponentin latautuessa 1. Esimerkissä näytetään myös asettajafunktion käyttö someFunctionissa, joka lisää someStaten arvoon yhden.

2.3 A-Frame-kirjasto

Karelian toteutuksessa virtuaaliympäristön 360°-näkymä luotiin A-Frame-kirjaston React-integraatiota hyödyntäen (Supermedium 2022). A-Frame on joulukuussa 2015 julkaistu web-pohjaisten virtuaalitodellisuuskokemuksien rakentamista varten kehitetty avoimen lähdekoodin sovelluskehys. A-Frame on rakennettu Three.js:n päälle, joka pyrkii olemaan helppokäyttöinen, kevyt, selainriippumaton ja yleiskäyttöön tarkoitettu 3D-kirjasto (mrdoob 2022). Sovelluskehyksen kehitti Mozillan MozVR-tiimi (Mozilla Mixed Reality 2015).

A-Framen toiminnassa korostuu järjestelmäriippumattomuus ja helppokäyttöisyys. A-Frame tarjoaa valmiita komponentteja, joiden avulla kehittäjä voi rakentaa 3D-ympäristön sisältävän sivuston melko lyhyessä ajassa ja lisätä siihen alkeellisia ominaisuuksia, esimerkiksi animaatioita tai yksinkertaisia vuorovaikutteisia elementtejä (A-Frame 2022a). A-Frame soveltaa entiteetti-komponentti-järjestelmä-ohjelmistoarkkitehtuuria, jossa ohjelmiston nähdään koostuvan osista, jotka voidaan jakaa kolmeen tyyppiin: entiteetteihin, komponentteihin ja järjestelmiin. Entiteetit ovat komponenttikokonaisuuksia, jotka toimivat ohjelmiston osien pohjana. Komponentit ovat moduuleja tai tietokokonaisuuksia, joita voidaan käyttää uudelleen eri entiteeteissä. Komponentteja käytetään entiteettien logiikan rakennuspalikoina. Järjestelmät ovat logiikkakokonaisuuksia, joiden puitteissa entiteetit toimivat. Järjestelmä siis sisältää ohjelmiston logiikan, joka hyödyntää entiteettejä, jotka koostuvat erilaisista ominaisuuksista, komponenteista. (A-Frame 2022b.)

A-Framessa entiteettejä kuvataan <a-entity>- tagilla. Entiteetteihin voidaan kiinnittää HTML-ominaisuuksina määriteltyjä komponentteja. Jokaiselle entiteetille on saatavilla tiettyjä A-Framen tarjoamia valmiita komponentteja, esimerkiksi “paikka” (kappaleen sijainti koordinaatistossa), “geometria” (kappaleen perusmuoto ulottuvuuksineen) ja “materiaali” (kappaleen pinnan materiaalin ominaisuudet). Kehittäjät voivat myös luoda räätälöityjä komponentteja, joiden toimintalogiikka on rekisteröity AFRAME.registerComponent-metodilla. Kaikki entiteetit kootaan <a-scene>-tagilla määritellyn näkymän alaisuuteen. Näkymän HTML-ominaisuuksilla voidaan määritellä käytössä olevat järjestelmät, jotka komponenttien tavoin rekisteröidään AFRAME.registerScene-metodilla. A-Framen React-integraatio toimii samalla tavalla kuin sen tavallinen versio, mutta tagien nimeäminen on hieman erilainen. Tagien nimet on muutettu yhteensopiviksi React-elementtien nimeämiskäytänteiden kanssa aloittamalla kaikkien elementtien nimet isolla alkukirjaimella ja poistamalla kaikki väliviivat, esimerkiksi <a-entity>-tagi on muutettu <Entity>-tagiksi (kuva 7). (A-Frame 2022b.)

```
return (
  <div className="View">
    <Scene
      device-orientation-permission-ui="enabled: false"
      cursor={{ mouseCursorStylesEnabled: "false", rayOrigin: "mouse" }}
      vr-mode-ui="enabled: false"
      enter-vr="none"
      raycaster="objects: .raycastable"
      background="color: #CECECE"
    >
    ...

    {/*Scene and camera*/}
    <Entity id="sky-winter"
            primitive="a-sky"
            background-transition-controller="enabled: false; index: 2;"
            src={"#" + backdrop[2]}
            events={{
                mousedown: () => { interruptSpin(); }
            }}
            className="raycastable"
            radius="50.4"
    />
    ...

    {Companies()}
  </Scene>
```

Kuva 7. 360°-näkymän React-komponentti, joka piirtää A-Framen näkymän. Scene-tagi noudattaa Reactin nimeämiskäytänteitä muodostaen näkymän pohjan, järjestelmän. Entiteetit lisätään Entity-tagilla, joka sisältää entiteetin koostumukseen vaikuttavat komponentit, kuten “primitive”. Entiteetille on asetettu myös räätälöity komponentti, “background-transition-controller", argumentteineen. Companies-funktio palauttaa näkyvillä olevien organisaatiopallojen entiteetit. Kuvan koodia on lyhennetty.

A-Framella toteutettu ympäristö koostuu aina <a-entity>-tagilla merkatuista entiteeteistä, jotka on asetettu <a-scene>-tagilla luotuihin kokonaisuuksiin. Yksinkertaisimmillaan ympäristö voi koostua muutamasta <a-scene>-tagien ympäröimästä <a-entity>-tagista. Reactissa <a-entity>- ja <a-scene>-tagit on nimetty <Entity>- ja <Scene>-tageiksi.

Monimutkaiset ympäristöt tarvitsevat usein ulkopuolisia tiedostoja, jotka liitetään ympäristöön, esimerkiksi pinnan materiaalina. A-Frame käyttää ulkopuolisista resursseista nimeä “asset”. Ulkopuolisia tiedostoja voidaan sisällyttää ympäristöön suorasti niiden URL-linkkiä käyttäen, mutta tyypillisesti resurssit ovat hyödyllistä ladata etukäteen suoritustehon parantamiseksi. Esilataus toteutetaan A-Framen resurssienhallintajärjestelmää käyttäen (kuva 8); kaikki esiladattavat tiedostot asetetaan <a-assets>-tagien väliin <a-asset-item>-tagilla ilmaistuna.

<a-asset-item>-tagista on erilaisia valmiita versioista, kuten <img>, joita voidaan käyttää tyypillisimpiä tietotyyppejä ladatessa. Esilataus varmistaa, etteivät ympäristön elementit piirry ruudulle ennen latauksen valmistumista. (A-Frame 2022c.)

```
{/*Asset loading*/}
<a-assets>
  <img id={backdrop[0]} src={imgSky} alt="Panorama view" />
  <img id={backdrop[1]} src={imgPanoramaAutumn} alt="" />
  <img id={backdrop[2]} src={imgPanoramaWinter} alt="" />
  <img id="missing-image" src={missingImage} alt="" />
  <img id="company-ball" src={matCompanyBall} alt="" />

  {/* Organization icons */}
  <img id="org-company" src={orgCompany} alt="" />
  <img id="org-education" src={orgEducation} alt="" />
  <img id="org-research" src={orgResearch} alt="" />
  <img id="ord-sights" src={orgSights} alt="" />
  <img id="org-partner" src={orgPartner} alt="" />
</a-assets>
```

Kuva 8. A-Framen resurssienhallintajärjestelmän käyttö Karelian projektissa.

3 360°-ympäristö

3.1 360°-näkymän arkkitehtuuri

Virtuaaliesittelyn kolmiulotteisen ympäristön toteutukseen käytettiin A-Framen React-integraatiota, aframe-react, sovelluskehyksien yhteistyön helpottamiseksi (Supermedium 2022). Näkymä muodostuu keskellä olevasta kamerasta, jota ympäröi 360°-panoraamakuva Kolista. Panoraama kuva vaihtuu pehmein siirtymin kolmen sekunnin tauoilla talvi-, kesä- ja syysmaisemien välillä. Ympäristöön on asetettu eri organisaatioita edustavia palloja, joita klikattaessa käyttäjälle avautuu organisaation lyhyt esittely, kuvagalleria, yhteystiedot, karttanäkymä sekä muita liitetiedostoja ja linkkejä sisältävä tietopaneeli. Jos käyttäjä vie hiiren organisaation pallon päälle, sen yläpuolelle ilmestyy näkymään upotettu tietotaulu, joka sisältää organisaation nimen, logon sekä yhteystiedot.

Käyttäjä voi tarkastella ympäristöään hiirtä vetämällä. Mikäli käyttäjä ei vaikuta ympäristöön hiirellä millään tavalla, 360°-näkymässä oleva kamera alkaa pyörimään hitaasti esitellen ympäristöä ja organisaatioita.

Näkymän koodi on jaettu kahteen osaan: React-komponentin sisäpuolelle, jossa sijaitsee näkymän piirtymisen sekä vuorovaikutuksen käsittelevä logiikka, ja React-komponentin ulkopuolelle, jossa sijaitsee pääasiassa A-Framen räätälöityjen komponenttien rekisteröinnit sekä niiden käyttämien muuttujien määritelmät. Näkymä hyödyntää A-Framen registerComponent-metodia luomaan räätälöityjä komponentteja kameran hitaalle kääntymiselle, maisemakuvan pehmeälle vaihtumiselle ja organisaation tietotaulun kääntämiselle kameraa kohti (kuva 9). Räätälöidyt komponentit sisällytetään React-komponentissa määriteltyihin entiteetteihin, esimerkiksi kameraan, tageja käyttäen (kuva 10).

```
// Set continuous camera rotation
AFRAME.registerComponent('constant-spin', {

    tick: function (t, dt) {
        if( isSpinning === true )
        {
            this.el.components['look-controls'].yawObject.rotation.y -= degToRad(spinSpeed * dt);
            spinSpeed = Math.min(spinSpeedMax, spinSpeed + spinSpeedAcceleration * dt);
        }
        else
        {
            if( isSpinStopped === false && spinResumeInterval == null )
            {
                spinResumeInterval = setInterval(() => {
                    if( isSpinStopped === false )
                    isSpinning = true;
                }, 5000);
            }
        }
    }
});
```

Kuva 9. Räätälöityä komponenttia ”constant-spin” käytettiin näkymän hitaaseen kääntämiseen.

```
<Entity
  id="camera"
  primitive="a-camera"
  wasd-controls-enabled="false"
  look-controls={"reverseMouseDrag: true; magicWindowTrackingEnabled: false;"}
  constant-spin
  compass-updater
  position={{
    x: 0,
    y: 2.25,
    z: 0
  }}
>
    { renderCompass() }
</Entity>
```

Kuva 10. Kuvassa 9 rekisteröidyn “constant-spin”-komponentin sisällytys kameraentiteettiin.

A-Framen tuottaman näkymän pohjana toimii <Scene>-tagi, jossa on perusasetusten lisäksi määritelty laitteen gyroskoopin ja VR-ilmoitusten deaktivointi sekä kursorin käyttö kosketinlaitteena. Seuraavaksi näkymän käyttämät kuvamateriaalit – panoraamakuvat, organisaatiopallojen tekstuurit ja organisaatiotyyppien kuvakkeet – on esiladattu A-Framen resurssienhallintajärjestelmää hyödyntäen. Samalla jokaiselle kuvalle on annettu ID, jolla siihen viitataan muualla koodissa. Taustakuvat on asetettu näkymään <a-sky>-tyyppisiä entiteettejä käyttäen. Jokaisen vuodenajan taustakuva piirretään samanaikaisesti, mutta kullakin hetkellä näytettävä maisema on täysin läpinäkymätön muiden ollessa näkymättömiä. Taustakuvat hyödyntävät “background-transition-controller"-komponenttia pehmeään siirtymään. Kamera on näkymän keskellä oleva <a-camera>- tyyppinen entiteetti, jonka kautta maisema piirtyy. Kamera käyttää “constant-spin”-räätälöityä komponenttia pyörimään käyttäjän ollessa paikallaan.

Lopuksi näkymään asetetaan oikeaan yläkulmaan kompassi sekä kameraa ympäröivät organisaatiot. Koska organisaatioiden määrä on riippuvainen suodatusasetuksista, niitä ei voida määritellä manuaalisesti HTML-dokumenttiin. Sen sijaan organisaatiot on lisättävä dynaamisesti erillisellä funktiolla, joka luo organisaatioiden pallot suodattimen mukaan kameran ympärille tasaisin välein asetettuna. Organisaatiopallo (kuva 11) koostuu <a-sphere>-tyyppisestä entiteetistä ja perusentiteetistä (<a-entity>), joka piirtää organisaation tietotaulun

tasoa käyttäen. Organisaatiopallo ja tietotaulu ovat niputettu yhteen React-sirpaleen (React.Fragment) alaisuuteen. Sirpale on Reactin tarjoama HTML-tyyppinen elementti, jolla useita HTML-elementtejä voidaan niputtaa yhteen, mikä on hyödyllistä, kun funktion halutaan palauttavan useita elementtejä, kuten organisaatiopallon tapauksessa (kuva 11). <a-sphere>-entiteetti sisältää <a-image>-entiteetin, joka piirtää organisaation tyyppiä edustavan kuvakkeen. Tietotaulu käyttää tekstikomponenttia piirtämään organisaation nimen ja yhteystiedot sekä <a-image>-entiteettiä piirtämään organisaation logon.

```
return (
  <React.Fragment key={"company-" + index}>
    {/* Clickable company element */}
    <Entity
      className="raycastable"
      key={"company-element-" + index}
      primitive="a-sphere"
      position={{
        x: pointAtDistDir(360 / companyData.length * comp.CompanyId, dist_from_camera).x,
        y: 1.75,
        z: pointAtDistDir(360 / companyData.length * comp.CompanyId, dist_from_camera).y,
      }}
      rotation={{ y: 270 - comp.CompanyId * (360 / companyData.length) }}
      radius="0.4"
      src={(color === "none") ? "#company-ball" : ""}
      material={(color !== "none") ? "color: " + color + ";" : ""}
      opacity="0.65"
      events={{
        click: () => {
          handleCompanyClick(comp.CompanyId);
          if( isSoundMuted === false )
          audioClick.play();
        },
        mouseenter: () => {
          setShowInfo(index);
        },
        mouseleave: () => {
          setShowInfo(-1);
        },
      }}
      animation="property: object3D.position.y; to: 2.0; dir: alternate; dur: 3000; loop: true"
    >
      {<Entity
        key={"org-type" + index}
        primitive="a-image"
        src={getOrgIconByCategory(comp).icon}
        position={{
          x: 0,
          y: 0,
          z: 0,
        }}
        width="0.5"
        height="0.5"
        alpha-test="0.2"
      />}
    </Entity>

    {/* Billboard */}
    <Entity
      look-at="#camera"
      key={"bboard-info" + index}
      geometry="primitive: plane; height: 1.2; width: 4"
      material="color: aliceblue"
      scale={{
          x: lerp(0.55, 1, arprc),
          y: lerp(0.65, 1, arprc)
      }}
      position={{
        x:
          pointAtDistDir(360 / companyData.length * comp.CompanyId, dist_from_camera).x -
          pointAtDistDir(360 / companyData.length * comp.CompanyId + 90, 1).x,
        y: 3.75,
        z:
          pointAtDistDir(360 / companyData.length * comp.CompanyId, dist_from_camera).y -
          pointAtDistDir(360 / companyData.length * comp.CompanyId + 90, 1).y
      }}
      visible={showInfo === index ? true : false}
      text={txt}
    >
      {/* Image container */}
      <Entity
        key={"bboard-image-container" + index}
        geometry="primitive: plane; height: 1.2; width: 2"
        material="color: aliceblue"
        position={{
          x: 3,
          y: 0,
          z: 0
        }}
        visible={showInfo === index ? true : false}
```

Kuva 11. Organisaatiopallon rakenne.

React-komponetti sisältää lisäksi muiden sivuston elementtien – esimerkiksi suodatusvalikon, chatin, kielivalikon ja organisaatioiden tietopaneelin – piirron, mutta näitä elementtejä ei renderöidä A-Framen näkymässä. Sen sijaan elementit asettuvat kolmeulotteisen näkymän päälle (kuva 12).

```
    {Companies()}
</Scene>
  { showStartOverlay === false &&
  <>
    {overlayOpen && (
      <Overlay
        companyData={clickedCompany}
        open={overlayOpen}
        close={closeOverlay}
        closeInfo={closeInfoPreview}
        tutorialPhaseSetter={setTutorialPhase}
        isSoundMuted={isSoundMuted}
      />
    )}

    <UpperMarginMenu
      setCategories={setCategories}
      setShowForm={setShowForm}
      categories={categories}
      setSearchInput={setSearchInput}
      setShowTutorialOverlay={setShowTutorialOverlay}
      companyData={companyData}
      handleCompanyClick={handleCompanyClick}
      toggleSpin={toggleSpin}
      isSpinning={spinState}
      toggleMuted={toggleMuted}
      isSoundMuted={isSoundMuted}
    />
```

Kuva 12. Kaksiulotteiset elementit asetetaan A-Framen näkymän päälle asettamalla React-elementit heti </Scene>-tagin jälkeen.

3.2 360°-näkymän vaihtoehtoiset toteutukset

Virtuaaliset esittely-ympäristöt ovat uutta teknologiaa, eikä kirjallisuutta A-Framella kehitettyjen virtuaaliesittelyjen teknisistä toteutuksista ole juurikaan tarjolla. Ongelma korostuu Reactin ja A-Framen yhteiskäytön toteutuksia tarkastellessa. Näistä syistä tässä osiossa keskitytään A-Framella kehitettyjen

virtuaalikiertueiden toteutusten tarkasteluun. Virtuaalikiertueet ovat pitkälti samanlaisia kuin virtuaaliesittelyt, mutta niiden käyttötarkoitus on usein erilainen; virtuaalikiertueita käytetään yleisesti sijaintien, esimerkiksi hotellien tai turistikohteiden, esittelyyn. Virtuaaliesittelyissä käyttäjällä ei tarvitse olla mahdollisuutta liikkua näkymien välillä, vaan kaikki esiteltävät kohteet voivat olla samassa tilassa. Sen sijaan virtuaalikiertueessa käyttäjän kyky liikkua on korostunut. Virtuaalikiertueiden toteutuksessa kehittäjät käsittelevät kuitenkin samoja teemoja kuin virtuaaliesittelyjen toteutuksessa: 3D-maailman katselu ja piirto, sisällön esittäminen virtuaaliympäristössä ja käyttäjän vuorovaikutus ympäristön kanssa. Koska esittely-ympäristöt ovat pitkälti samanlaisia, virtuaalikiertueiden toteutuksia voidaan hyödyntää myös virtuaaliesittelyjen kehityksessä.

Solange Gomes Santos ja Jorge C. S. Cardoso toteuttivat A-Framella virtuaalikiertueen Conimbrigan museosta, jonka kehitystä käsitellään heidän raportissaan “A-Frame experimentation and evaluation for the development of interactive VR: a virtual tour of the Conimbriga Museum”. Kiertue koostui museon 360°-panoraamakuvista, joihin asetettiin vuorovaikutteisia elementtejä. Käyttäjän oli mahdollista tarkastella museossa näytillä olevia arkeologisia esineitä sekä liikkua maisemien välillä. Kiertueen toteutuksessa ei käytetty Reactia, vaikka nykyisin kehityksestä poistuneen React-360-kirjaston käyttöä harkittiin projektin suunnitteluvaiheessa. Kehittäjät kuitenkin näkivät Reactin olevan “erikoistaito, jota ei ollut kaikilla web-ohjelmoijilla". (Santos & Cardoso 2019, 3.)

Virtuaalikiertueen panoraamakuvat esiladattiin A-Framen resurssienhallintajärjestelmää käyttäen. Maisemakuville annettiin oma ID, jolla siihen viitattiin koodissa. 3D-näkymään upotettiin kartta, jota hyödyntäen käyttäjä voi siirtyä kiertueen huoneiden välillä. Kartta hyödynsi räätälöityä komponenttia pysyäkseen aina samassa suunnassa kuin kamera sekä havaitsemaan käyttäjän painalluksia. Kartta toteutettiin niputtamalla yhden <a-entity>-tagin alle useita <a-image>-entiteettejä, jotka muodostivat kartan osa-alueet. Näkymässä olevien arkeologisten kappaleiden viereen asetettiin elementti, jota klikkaamalla käyttäjä pystyi avaamaan tietopaneelin, joka sisälsi lyhyen esittelyn esineestä. (Santos & Cardoso 2019, 6, 9—11)

Tietopaneeli hyödynsi räätälöityä komponenttia havaitsemaan painalluksen ja muuttamaan paneelin läpinäkyvyyden (opacity) käyttäjän klikatessa. Paneeli sulkeutuu, mikäli komponentti havaitsee käyttäjän katsovan poispäin laskemalla kulman paneelin ja kameran kiertokulman välillä. Kiertueen työpöytä versiossa hiiren sijainti tulkittiin käyttäjän katseeksi. Työpöytäversiossa tietopaneeli irrotettiin näkymästä ja toteutettiin kaksiulotteisena HTML-dokumenttiin virtuaalinäkymän päälle. <a-scene>-tagiin, johon näkymä pohjautui, kursori asetettiin kosketinlaitteeksi. Myös navigaatiovalikko toteutettiin HTML-dokumenttiin. (Santos & Cardoso 2019, 12—13, 14—15.)

Kehittäjät kohtasivat ongelmia tekstielementtien, esimerkiksi tietopaneelien, toteuttamisessa. Tekstien asettaminen elementteihin oli työlästä, sillä tekstikomponentille varatun alueen ulottuvuudet oli määriteltävä erikseen automaattisen määrittymisen sijasta. Esimerkiksi museon esineiden tietoikkunat olivat pitkälti räätälöityjä eripituisten tekstien takia. Myös visuaalisesti monimutkaisten elementtien toteuttamisessa havaittiin ongelmia; kartan pyöreät reunat oli toteutettava kuvankäsittelyohjelmalla tuotetulla tekstuurilla, sillä A-Frame ei tue CSS:n kaltaista tyylittelyä. (Santos & Cardoso 2019, 17.)

Mediumissa julkaistussa artikkelissaan, “How to create a Virtual Tour using A-Frame", Kumar Ahir (2018) kuvaa yksityiskohtaisesti yhdellä sivulla toimivan virtuaalikiertueen kehitystä A-Framella. Toteutus on yksinkertainen esimerkkisovellus, jossa käyttäjä voi tarkastella ympäristöä, jonka maisemana on lammen rannalta otettu 360°-panoraamakuva. Näkymään on upotettu klikattavia elementtejä, joiden avulla käyttäjä voi siirtyä muihin näkymiin. (Ahir 2018.)

Kiertue alkaa esilataamalla A-Framen resurssienhallintajärjestelmää käyttäen maisemakuvat antaen jokaiselle uniikin ID:n, jolla siihen viitataan koodissa. Näkymä pohjautuu <a-scene>-tagiin, johon kaikki näkymän elementit kuuluvat. Maisemakuvat on asetettu näkymään <a-sky>-tagilla, jotka viittaavat resurssienhallintajärjestelmässä määriteltyihin kuviin. (Ahir 2018.)

Seuraavaksi toteutettiin siirtymäpainikkeet, jotka upotettiin näkymään A-Frame-elementteinä. Painikkeita varten rekisteröitiin kaksi räätälöityä komponenttia: “spot” ja “hotspots”. Painikkeet muodostuivat “spot”-komponentin sisältävistä <a-image>-entiteeteistä, jotka oli koottu saman “hotspots”-komponentin sisältävän isäntäentiteetin alle. Jokaiseen painikkeeseen oli myös asetettu räätälöity “look-at”-komponentti sekä viittaus maisemasta, johon se johti. “look-at”-komponenttia käytettiin kääntämään entiteettiä kohti kameraa. Painiketta klikatessa seuraavan maiseman painike suurennettiin tavalliseen kokoonsa säätäen muiden painikkeiden koko nollaan. Myös maisema vaihdettiin painiketta klikattaessa. (Ahir 2018.)

3.3 360°-näkymän parantaminen

Karelian virtuaaliesittelyssä ja tarkastelluissa virtuaalikiertuetoteutuksissa oli useita samanlaisuuksia. Kaikki toteutukset hyödynsivät A-Framen tarjoamaa <a-scene>-entiteettiä näkymän pohjana, johon asetettiin maisemaa kuvaava 360°-panoraamakuva <a-sky>-entiteetillä. Koska toteutuksemme käytti A-Framen React-integraatiota, elementtien nimeämisessä noudatettiin Reactin vaatimuksia; A-Frame-elementtien nimissä ei käytetty väliviivoja, ja ne alkoivat isolla alkukirjaimella. Toteutuksissa käytetyt resurssit – kuten panoraamakuvat ja painikkeiden graafiset esitykset – esiladattiin A-Framen resurssienhallintajärjestelmällä. Lisäksi jokaiselle resurssille määriteltiin ID, jolla siihen viitattiin koodissa. Kuvien esilataamisella varmistettiin, että A-Framen näkymän piirtämisessä käytetyt resurssit olivat valmiita renderöitäväksi. Kuvat olisi ollut myös mahdollista linkittää näkymään tavanomaisesti "src"-ominaisuudella, mutta näkymän taustakuvat olisivat piirtyneet hitaasti useita suuria panoraamakuvia ladattaessa. Karelian toteutuksen näkymän perusrakenteen ja resurssienhallintajärjestelmän käytössä ei ole parannettavaa.

Tarkastelluissa virtuaalikiertueissa, kuten Karelian toteutuksessakin, hyödynnettiin räätälöityjä komponentteja näkymän entiteettien ohjaamiseen. Komponentit rekisteröitiin AFRAME.registerComponent-metodilla, jolloin niitä voitiin käyttää entiteeteissä samalla tavalla, kuin A-Framen peruskomponentteja.

Komponenttien käytössä ei ole pääasiassa poikkeavuuksia vertailtaviin toteutuksiin nähden, mutta Reactin käyttö projektissamme asetti komponenttien rekisteröinnille tiettyjä rajoituksia, joita muut toteutukset eivät joutuneet noudattamaan. Muissa toteutuksissa A-Framea käytettiin JavaScriptin kanssa ilman ylimääräisiä sovelluskehyksiä. Karelian toteutuksessa käytettiin funktiopohjaisia React-komponentteja, jotka renderöityessän piirsivät sisältämänsä A-Frame näkymän. Koska funktio suoritetaan uudelleen jokaisella renderöinnillä, kaikki funktiossa rekisteröidyt, räätälöidyt komponentit rekisteröitiin uudelleen. Ongelman ratkaisemiseksi A-Framen komponenttirekisteröinnit erotettiin React-elementin funktion ulkopuolelle, jolloin rekisteröinti toteutuu vain kerran. Karelian toteutuksessa kaikki A-Frame-komponentit rekisteröidään ja asetetaan näkymään samassa tiedostossa, mikä voi aiheuttaa ongelmia komponentteja lisätessä, sillä tiedoston sisältämä koodin määrä kasvaa hankaloittaen kehitystä ja skaalautuvuutta. Jatkokehityksessä on suositeltavaa pyrkiä jakamaan komponenttien rekisteröinti erillisiin tiedostoihin.

Kaikissa toteutuksissa hyödynnettiin entiteettien yhdistelyä isäntäentiteettien alaisuuteen kokonaisuuksien muodostamiseksi, kuten ympäristössä klikattavien entiteettien ryhmä, jossa ryhmään kuuluvat entiteetit yhdistettiin yhden entiteetin alle. Karelian toteutuksessa yhdistelyä on käytetty, mutta hyvin rajoitetusti. Yhdistely on toteutettu sekaisin Reactin ja A-Framen määritelmien mukaan. Kuten aiemmin esitetyssä kuvassa 9 on havaittavissa, kaikki organisaatiopallon entiteetit on niputettu React.Fragment-tagin alaisuuteen, mutta entiteettejä käytetään myös isäntäentiteetteinä. Skaalautuvuuden parantamiseksi olisi hyödyllistä käyttää ainoastaan entiteettejä, kuten muissa toteutuksissa, sillä entiteetteihin saatetaan joutua tulevaisuudessa viittaamaan tai lisäämään ominaisuuksia.

Muita yhtäläisyyksiä oli A-Framen virtuaalinäkymän vuorovaikutuksellisien elementtien ja tietoikkunoiden erottelu toisistaan. Klikattavat elementit lisättiin A-Framen tuottamaan näkymään, mutta klikattaessa avautuvat tietoikkunat asetettiin HTML-dokumentissa näkymän päälle erillisiksi kokonaisuuksiksi. Karelian toteutuksessa onnistuttiin erottelemaan virtuaaliympäristö ja tietopaneelit toisistaan selkeästi sekä toteuttamaan sujuva tiedon siirtyminen

osien välillä. Vaikka sivuston kehityksessä päätettiin käyttää A-Framen React-integraatiota, jatkokehityksen kannalta on suositeltavaa pyrkiä siirtymään A-Framen perusversion käyttöön. Siirtymistä suositellaan React-integraation Github-pakettivarastossa. Siirtymisen jättäminen toteuttamatta saattaa johtaa tulevaisuudessa yhteensopivuusongelmiin uusien React-versioiden kanssa tai korjaamattomiin virheisiin Reactin ja A-Framen yhteistoiminnassa, mikäli kehittäjät päättävät olla jatkamatta projektin kehitystä.

4 Kielituki

4.1 Kielituen arkkitehtuuri

Koska sivusto toteutettiin osana Japanin Naganon ja Pohjois-Karjalan välistä metsäbiotaloushanketta, yhdeksi vaatimukseksi nousi käännöstoiminnallisuuden toteutus. Sivusto oli tarkoitettu käännettäväksi suomeksi, japaniksi ja englanniksi, mutta kielituki toteutettiin siten, että käännöksiä voidaan lisätä mielivaltainen määrä. Kielen vaihtaminen tapahtuu sivun vasemmassa yläkulmassa olevia lippuja klikkaamalla (kuva 13). Kun käyttäjä klikkaa lippua, kielituki asettaa localStorageen tallennetun JSON-tyyppisen kielipaketin valittuun kieleen ja lataa sivun uudelleen. localStorage on selaimen käyttämä muistitila, jota web-sovellukset voivat käyttää tallentamaan tarpeellista tietoa ilman, että tieto katoaa sivua uudelleen ladattaessa tai sivujen välillä siirtyessä.

Kuva 13. Kielivalikko.

Kielituki toteutettiin tavallisena JavaScript-luokkana, sillä monet komponentit hyödynsivät käännöksiä, mikä vaati kielituen erottamista muusta koodista. Kielituki toimii omana kokonaisuutenaan taustalla React-komponenttien hakiessa siltä käännöksiä aina tarvittaessa. Kielituki-luokasta ei ole tarkoitus luoda instansseja. Tämän takia kaikki kielituen jäsenmuuttujat ovat staattisia, eli

saatavilla pelkkään luokkaan viittaamalla. Kielituki (kuva 14) sisällyttää aluksi kaikki tarvittavat kielipaketit, ja tallentaa niiden viitteet erillisiin muuttujiin, jotta React-komponentit voivat viitata kielipakkauksiin luokan ulkopuolelta. Seuraavaksi kielituki asettaa tällä hetkellä aktiivisena olevan käännöksen localStoragessa sijaitsevan kielipaketin mukaan. Mikäli localStoragesta ei löydy kielipakettia, oletuskielenä on suomi.

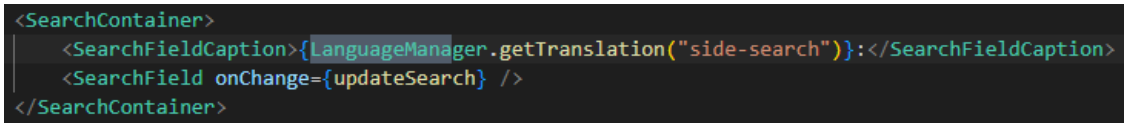
```
1   import IFIN from "./packs/FIN.json";
2   import IENG from "./packs/ENG.json";
3   import IJPN from "./packs/JPN.json";
4
5   class LanguageManager {
6
7       // Utilized when setting the language
8       static FIN = IFIN;
9       static ENG = IENG;
10      static JPN = IJPN;
11
12      // Default language
13      static siteLanguage = (
14                              localStorage.getItem("language") == null ||
15                              JSON.parse(localStorage.getItem("language")).LANG == null
16                          )
17                          ? this.FIN
18                          : JSON.parse(localStorage.getItem("language"));
```

Kuva 14. Kielituki sisällyttää kielipaketit ja asettaa kielen localStoragesta tai suomen, mikäli localStorageen ei ole tallennettu kieliasetusta.

Kielipaketit ovat JSON-tiedostoja, jotka sisältävät käännökset jokaiselle kielitukea hyödyntävälle sivuston tekstikentälle. Tiedostossa käännökset on asetettu viittaus-käännös-pareihin (kuva 15). React-komponentit hakevat käännöksiä hyödyntäen kielituen getTranslation-metodia (kuva 16), joka palauttaa käännöksen viittauksen perusteella. Tällöin React-komponenttien tarvitsee tietää vain viittaukset käyttämiinsä käännöksiin. Jokaisessa kielipaketissa on myös kenttä, johon on tallennettu kielen lyhenne. Lyhennettä käytetään tapauksissa, joissa kielipaketit on eroteltava toisistaan, esimerkiksi kielituen isCurrentLanguage-metodissa, jossa annettua kielipakettia vertaillaan tällä hetkellä aktiivisena olevaan pakettiin (kuva 17). Jokainen kielipaketti on rakenteeltaan samanlainen sisältäen samat kentät eri kielillä.

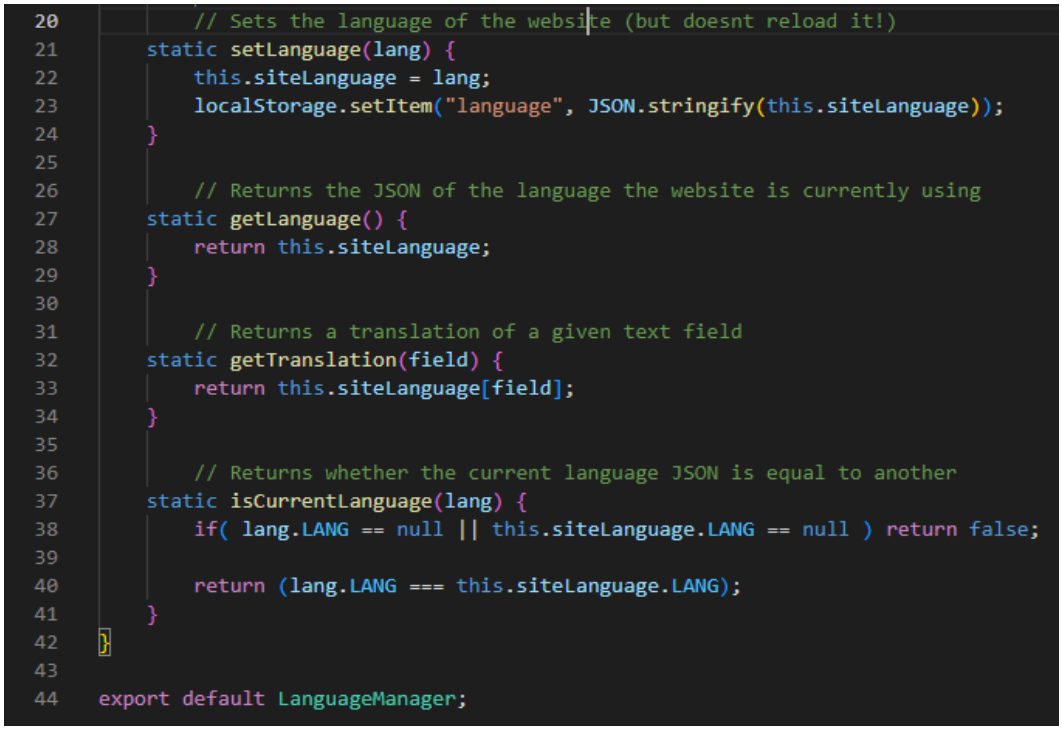
```
{
  "LANG": "FIN",

  "side-companies": "Yritykset",
  "side-education": "Tutkimus-, kehitys- ja koulutusorganisaatiot",
  "side-partners": "Muita metsäalan yhteistyöorganisaatioita ja toimijatahoja",
  "side-authorities": "Viranomaiset",
  "side-bioeconomy": "Biotalous:",

  "side-linkTypeVideo": "Video",
  "side-linkTypeFile": "Tiedosto",
  "side-linkTypeLink": "linkki",

  "side-other": "Nähtävää ja koettavaa",
  "side-attractions": "Nähtävää ja koettavaa",
  "side-general": "Yleistietoa",
  "side-addInfo": "Lisää yritystiedot",
  "side-all": "Kaikki",
  "side-categories": "Valitse kohderyhmä",
  "side-search": "Hae"
```

Kuva 15. Esimerkkinä suomen kielipaketti, josta käännökset haetaan avainta käyttäen. LANG-kenttää voidaan hyödyntää erottelemaan kielipaketit toisistaan koodissa.



```
<SearchContainer>
    <SearchFieldCaption>{LanguageManager.getTranslation("side-search")}:</SearchFieldCaption>
    <SearchField onChange={updateSearch} />
</SearchContainer>
```

Kuva 16. ”Etsi”-näppäin hyödyntää kielituen tarjoamia käännöksiä ”etsi”-tekstin piirtoon.



```
        // Sets the language of the website (but doesnt reload it!)
    static setLanguage(lang) {
        this.siteLanguage = lang;
        localStorage.setItem("language", JSON.stringify(this.siteLanguage));
    }

        // Returns the JSON of the language the website is currently using
    static getLanguage() {
        return this.siteLanguage;
    }

        // Returns a translation of a given text field
    static getTranslation(field) {
        return this.siteLanguage[field];
    }

        // Returns whether the current language JSON is equal to another
    static isCurrentLanguage(lang) {
        if( lang.LANG == null || this.siteLanguage.LANG == null ) return false;

        return (lang.LANG === this.siteLanguage.LANG);
    }
}

export default LanguageManager;
```

Kuva 17. Kielituen tarjoamat metodit.

4.2 Kielituen vaihtoehtoiset toteutukset

Opinnäytetyössään "Developing a restaurant web application using JavaScript libraries" Hien Nguyen käsitteli Reactilla luodun sivuston toteutusta, sovellusarkkitehtuuria ja suunnitteluvalintoja. Sivusto toteutettiin toimeksiantona vietnamilaista ruokaa tarjoilevalle Ravintola Fitille. Pyrkimyksenä oli luoda ravintolan brändiä mainostavat kotisivut, joista asiakkaat voivat tutustua ravintolaan sekä sen menuun, kampanjatuotteisiin, yhteystietoihin ja aukioloaikoihin. Kyseessä on tavallinen, yhdellä sivulla toimiva, yrityksen kotisivu, jossa käyttäjä voi siirtyä näkymien välillä navigaatiovalikon näppäimiä klikaten. Vaikka ravintola on suomalainen, sivustoon toteutettiin myös käännökset englanniksi, joilla pyrittiin luomaan ammattimainen kuva ravintolasta sekä parantamaan ravintolan sijoitusta hakukonetuloksissa; monikieliset sivustot asettuvat todennäköisemmin hakukoneen tulosten ensimmäiselle sivulle. (Nguyen 2019, 15—17, 25.)

Käännökset toteutettiin käyttäen i18next-sovelluskehyksen React-integraatiota, joka on sivujen kansainvälistämiseen tarkoitettu kielituki (i18next 2022a). i18next hyödyntää JSON-tiedostoja, joihin kehittäjä asettaa käännökset tietylle kielelle avain-käännös-pareina (kuva 18). Kehittäjän on mahdollista asettaa käännöksiä useiden avaimien alle. Käyttääkseen käännöksiä koodissa, sivusto alustaa i18nextin ensin käyttäen i18next.init-metodia, jossa voidaan asettaa esimerkiksi sivuston oletuskieli. Reactissa kehittäjä hyödyntää useTranslation-koukkua hakemaan oikean käännöksen JSON-tiedostosta avainta käyttäen (kuva 19). Käytössä olevan kielen voi vaihtaa i18next.changeLanguage-metodilla. (i18next 2022b; i18next 2022c.)

```
{
    "key": "value of key",
    "look": {
        "deep": "value of look deep"
    }
}
```

Kuva 18. Esimerkki käännöstiedostosta. I18nextissä nimiavaruuksia voidaan määritellä avaimilla, joiden alaisuuteen asetetaan avain-käännös-pareja (Kuva: i18next 2022b).

```
// having resources like this:
/*{
  "translation": {
    "very": {
      "deeply": {
        "nested": {
          "key": "here"
        }
      }
    }
  }
}*/
// you can define a keyPrefix to be used for the resulting t function
const { t } = useTranslation('translation', { keyPrefix: 'very.deeply.nested' });
const text = t('key'); // "here"
```

Kuva 19. i18nextin dokumentaatiossa annettu esimerkki useTranslation-koukun, ja sen palauttaman t-käännösfunktion käytöstä. Kuvassa kielipakkausta on havainnollistettu kommentin sisällä (Kuva: i18next 2022c).

Ravintola Fitin sivujen toteutuksessa kielituki alustetaan i18next.init-metodilla, mutta alustus toteutetaan palvelimen puolella. Oletuskieleksi asetetaan suomi, jonka jälkeen palvelin lähettää käännöstiedostot (JSON). Sivusto käyttää palvelimelta saatua i18next-oliota ja käännöstiedostoja asettamaan valitut käännökset. Kun käännöstiedostot on vastaanotettu ja i18next on alustettu, palvelimelta ei tarvitse hakea enää käännöksiä koskevaa tietoa. (Nguyen 2019, 41—44.)

Käännöstiedostot on eritelty kansioihin kielen perusteella ja jaoteltu erillisiin tiedostoihin niiden kategorioiden mukaan. Esimerkiksi menujen käännökset on asetettu omiin JSON-tiedostoihinsa ja yhteystiedot omiinsa. Käännöstiedostojen rakenne on yksikertainen, sillä ne muodostuvat ainoastaan avain-arvo-pareista (avain - käännös), eikä esimerkiksi ylimääräisiä avaimia ole hyödynnetty.(Nguyen 2019, 44—45.)

Kun sivusto on saanut tarvittavat tiedot palvelimelta, useTranslation-koukkua käytetään elementeissä, joihin tarvitaan käännökset. Koukussa ei olla määritelty oletuskäännöstä, mikäli käännöksen haussa tapahtuu virhe. Ravintola Fitin

toteutuksessa ei kuvailla, kuinka kielen vaihtaminen tapahtuu, mutta Reactin juurikomponentti on asetettu i18nextin tarjoaman I18nextProvider-komponentin sisään. I18nextProviderin kautta kaikki React komponentit voivat kutsua useTranslation- ja changeLanguage-koukkuja, jotka muuttavat I18nextProviderin tilaa. Kun React havaitsee tilan muutoksen, sivusto renderöidään uudelleen ilman uudelleenlatausta. (Nguyen 2019, 45.)

4.3 Kielituen parantaminen

Toteuttamassamme kielituessa ja sen implementaatiossa esiintyi useita yhtäläisyyksiä Nguyenin ratkaisun sekä i18nextin toimintaperiaatteen kanssa. Nguyenin toteutus valittiin vertailtavaksi, koska hänen toimeksiantonsa oli hyvin samanlainen kuin omamme: kaupalliseen esittelyyn tarkoitettu sivusto, joka sisältää erilaisia näkymiä, joiden tekstisisältö on käännettävissä muille kielille. Lisäksi Nguyenin opinnäytetyössä sivuston kehitystä on käsitelty samasta näkökulmasta, kuin tässä opinnäytetyössä käyden läpi ensin kehityksen työkaluja ja menetelmiä, sekä kuvaten sivuston toteutusta arkkitehtuuritasolla.

Emme kohdanneet juurikaan ongelmia oman kielitukemme kehityksessä ja integraatiossa projektiin, mutta suosittelen silti i18nextin käyttöön siirtymistä. i18nextin toimintaperiaate on hyvin samanlainen, kuin Karelian toteutuksen kielituen. Toteutuksemme kielituki on erotettu muusta ohjelmasta, kuten i18next Nguyenin toteutuksessa, koodin toiston vähentämiseksi ja skaalautuvuuden parantamiseksi. Mikäli kielituki olisi toteutettu osaksi jotain renderöintipuussa juurta lähellä olevaa komponenttia, komponentin olisi pitänyt antaa viittaus kielitukeen alikomponenteilleen, joiden olisi pitänyt antaa viittaus jälleen eteenpäin. Tällöin myös komponentit, joiden ei tarvitse hyödyntää kielitukea, olisivat silti joutuneet käsittelemään kielitukeen liittyvää koodia lisäten toistoa. Paras ratkaisu oli erottaa kielituki omaksi luokakseen, jonka React-komponentit pystyivät sisällyttämään osaksi koodiaan, mikäli kielitukea komponentissa tarvittiin.

Molemmat kielituet hyödyntävät ulkoisia JSON-tiedostoja käännösten tallentamiseen avain-käännös-pareina. Kun kielet on asetettu erillisiin tiedostoihin, ne saadaan erotettua muusta koodista parantaen ohjelman selkeyttä, varsinkin mikäli käännösten määrä kasvaa. Ratkaisu on myös skaalautuva, sillä uusien kielipakettien lisääminen, tai olemassa olevien poisto, ei vaadi muutoksia ohjelmakoodiin, ja ainoastaan pieniä muutoksia kielitukeen. Jos kielet olisi asetettu esimerkiksi yhteen tiedostoon, käännösten määrän kasvaessa, tiedoston navigoinnista olisi tullut haasteellista. Kielipakettien hakeminen ulkoiselta palvelimelta on myös helpompaa pieniä tiedostoja käyttäen.

Karelian toteutuksen kielituen toiminnallisuuksien lisäksi i18next tarjoaa muita hyödyllisiä toimintoja. i18nextin kielipaketeissa voidaan hyödyntää nimiavaruuksia, joilla avain-käännös-pareja voidaan kategorisoida. Tämä on hyödyllistä käännösten määrän lisääntyessä, sillä kielipakkauksiin voidaan muodostaa selkeä rakenne, jota kehittäjän on helppo seurata, ja johon muutoksia on helpompi tehdä. i18next mahdollistaa myös monikkomuotojen asettamisen yksittäisille sanoille tehden käännöksistä dynaamisempia (kuva 20), sillä saman sanan eri muotoja ei tarvitse tallentaa erikseen kielipakettiin. Lisäksi i18next-sovelluskehys on kevyt vieden vain 33 kilotavua muistia dokumentaation mukaan (i18next 2022d).

```
{
  "key_one": "item",
  "key_other": "items",
  "keyWithCount_one": "{{count}} item",
  "keyWithCount_other": "{{count}} items"
}
```

sample

```
i18next.t('key', {count: 0}); // -> "items"
i18next.t('key', {count: 1}); // -> "item"
i18next.t('key', {count: 5}); // -> "items"
i18next.t('key', {count: 100}); // -> "items"
i18next.t('keyWithCount', {count: 0}); // -> "0 items"
i18next.t('keyWithCount', {count: 1}); // -> "1 item"
i18next.t('keyWithCount', {count: 5}); // -> "5 items"
i18next.t('keyWithCount', {count: 100}); // -> "100 items"
```

Kuva 20. Kuvan yläosassa määritellyssä kielipakkauksessa asetetaan monikkomuoto “key”-kentälle. Monikkomuotoa voidaan käyttää kuvan alaosassa annetun esimerkin mukaisesti (Kuva: i18next 2022e).

Merkittävien yhtäläisyyksien vuoksi suosittelen i18next-sovelluskehyksen käyttöön siirtymistä. Koska oma kielitukemme on hyvin samanlainen, i18nextin integraatio olisi hyvin helppoa, sillä se sopeutuisi muiden ohjelmiston komponenttien kanssa hyvin. i18next on yhtä helppokäyttöinen, kuin kielitukemme, mutta tarjoaa enemmän toiminnallisuuksia, jotka ovat hyödyllisiä skaalautuvuuden kannalta. Lisäksi i18nextin vaihtaessa kieltä, sivuston ei tarvitse latautua uudelleen toisin kuin omassa toteutuksessamme.

5 Syvät tietorakenteet

5.1 Syvien tietorakenteiden arkkitehtuuri

React-komponentit voivat hyödyntää Reactin tarjoamia tiloja käyttämällä useState-metodia. Tilojen käyttö on hyödyllistä, kun komponentti on piirrettävä uudelleen jonkun muuttujan vaihtaessa arvoaan. React-komponentin alussa määritellään muuttujat, joiden muuttumista React tarkkailee, ja asetusmetodit,

joilla tila muutetaan. Asetusmetodia kutsuessa Reactille ilmoitetaan tilan mahdollisesta muuttumisesta, jolloin React tarkistaa, onko tila muuttunut aiemmasta arvosta. Tilojen käyttö on yksinkertaista, kun niissä käsiteltävät tietorakenteet ovat yksinkertaisia, esimerkiksi “setCounter(0);” (kuva 21). Usein tietorakenteet ovat kuitenkin monimutkaisia ja syviä.

```
1  import React, { useState } from "react";
2
3  export default function Example(props) {
4      const [counter, setCounter] = useState(0);
```

Kuva 21. Esimerkki yksinkertaisen tilan määrittelystä.

Projektia toteuttaessa on useasti käytetty monimutkaisia tiloja, joissa React ei ole havainnut tilan muutosta. 360°-näkymässä käyttäjällä on mahdollisuus suodattaa organisaatioita organisaatiotyypin perusteella. Näkymän React-komponentissa on määritelty tila, joka sisältää taulukon kaikista näkyvillä olevista organisaatioista, ja niiden tiedoista. Käytössä on taulukko, joka sisältää JSON-olioita, jotka sisältävät taulukkoja (kuva 22).

```
1  import React, { useState } from "react";
2
3  export default function Example() {
4      const [nestedState, setNestedState] = useState({
5          jsonField: {
6              arrayField: [
7                  {
8                      field: value
9                  },
10                 {
11                     field: value
12                 },
13                 {
14                     field: value
15                 }
16             ]
17         }
18     })
19 };
```

Kuva 22. Havainnollistava esimerkki monimutkaisesta tilasta. UseState-koukkua käyttäen tilaksi asetetaan JSON, jonka sisällä on “jsonField”-kenttä, johon on tallennettu toinen JSON, jossa on “arrayField”-kenttä, joka sisältää taulukon, joka sisältää JSON-olioita.

Mikäli tilassa oleva taulukko muuttuu – esimerkiksi alkiota poistettaessa – tilassa sijaitseva viite ei muutu. Koska tilassa oleva arvo ei muutu, React ei havaitse muutosta, eikä sivuston näkymä päivity. Projektissa syviä tietorakenteita käyttäviä tiloja päivittäessä tietorakenteesta tehdään kopio, johon halutut muutokset tehdään. Tämän jälkeen käytetään asetusmetodia asettamaan uusi tila, jolloin tilassa oleva viite muuttuu, minkä React havaitsee piirtäen komponentin uudelleen.

5.2 Syvien tietorakenteiden vaihtoehtoiset toteutukset

Syvien tietorakenteiden käyttöön liittyviä ongelmia käsitellään myös Reactin dokumentaatiossa. React suositteli alun perin “react-addons-update"-kirjastossa määriteltyä “update”-metodia, mutta dokumentaatioon on myöhemmin lisätty ohjaus “immutability-helper”-kirjastoon, jonka toimintaperiaate on hyvin samanlainen. Immutability-helper on Moshe Kolodnyn kehittämä ja ylläpitämä kirjasto, joka on suunniteltu auttamaan syvien ja monimutkaisien tietorakenteiden muuttamisessa. Kirjasto ei ole React-kehitystiimin virallinen julkaisu, eikä kirjastoa ole tarkoitettu ainoastaan Reactin kanssa käytettäväksi, vaikka Reactin tilat ovatkin yleisin kohde kirjaston käytölle. (React 2022e; Kolodny 2022.)

Immutability-helperin toimintaperiaate on yksinkertainen. Koska Reactin useState-koukku havaitsee muutoksen ainoastaan, kun uusi tietorakenne asetetaan, tietorakenne on ensin kopioitava erilliseen muuttujaan, johon muutokset tehdään. Uusi tietorakenne syötetään sitten useState-koukkuun, jolloin näkymä piirretään uudelleen. Ilman immutability-helperiä, jokainen tietorakenteen sisällä oleva tietorakenne on kopioitava Object.assign-metodia tai levitysoperaattoria (…) käyttäen, jonka jälkeen tehdään muutokset. Prosessissa käytetty koodi on äärimmäisen hankalasti luettavaa, varsinkin jos useisiin tietorakenteen kenttiin tehdään muutoksia. Kuvassa 23 (kuva 23) on immutability-helperin dokumentaatiossa annettu esimerkki syvän tietorakenteen kopioinnista ja samanaikaisesta muuttamisesta ilman ylimääräisiä kirjastoja. (Kolodny 2022.)

```
myData.x.y.z = 7;
// or...
myData.a.b.push(9);
```

```
const newData = Object.assign({}, myData, {
  x: Object.assign({}, myData.x, {
    y: Object.assign({}, myData.x.y, {z: 7}),
  }),
  a: Object.assign({}, myData.a, {b: myData.a.b.concat(9)})
});
```

Kuva 23. Kuvan yläosassa määritelty “myData”-olio sisältää monimutkaisen tietorakenteen, jossa on useita kenttiä. Mikäli yläosassa näytetyt muutokset haluttaisiin tehdä React-komponentin tilaan, alaosassa kuvattua koodia olisi normaalisti käytettävä (Kuva: Kolodny 2022).

Immutability-helper tarjoaa “update”-metodin, joka ottaa syötteeksi muokattavan tietorakenteen sekä muutokset JSON-muodossa. Muutoskomennot ja kentät, joihin muutoksia halutaan tehdä, kirjataan JSON-rakenteeseen. Update-metodi palauttaa kopion, johon annetut muutokset on tehty. Kirjastossa on saatavilla useita muutoskomentoja, jotka merkitään dollarimerkillä, kuten $set ja $push. Kuvassa 24 aiemmin ilman immutability-helperiä toteutettu tietorakenteen muuttaminen toteutetaan kirjaston avulla, jolloin koodista tulee paljon selkeämpää. (Kolodny 2022.)

```
import update from 'immutability-helper';

const newData = update(myData, {
  x: {y: {z: {$set: 7}}},
  a: {b: {$push: [9]}}
});
```

Kuva 24. Immutability-helperillä toteutettu olion kenttien muokkaus on huomattavasti selkeämmin kirjoitettu, kuin ilman kirjaston käyttöä (Kuva: Kolodny 2022).

5.3 Syvien tietorakenteiden käytön parantaminen

Karelian toteutuksessa syvien tietorakenteiden käyttöä on pyritty välttelemään, kun se on ollut mahdollista. Syviä taulukoita käsitellessä map- ja filter-metodit ovat usein riittäneet uuden taulukon asettamiseen setStatella, kuten 360°-näkymän. Map- ja filter-metodit ovat JavaScriptin taulukkoluokan (Array) perusominaisuuksia. Map-metodi iteroi taulukon läpi ja suorittaa jokaisen alkion kohdalla metodissa määritellyn funktion, joka palauttaa jonkin arvon taulukkoon, jonka map palauttaa (Mozilla 2022a). Filter-metodi iteroi taulukon läpi ja palauttaa kopion taulukosta, josta on poistettu kaikki metodissa annetun vaatimuksen täyttävät alkiot (Mozilla 2022b).

Joissain tapauksissa syvät tietorakenteet on eritelty useaan eri tilaan, jolloin kopioinnilta on vältytty kokonaan. Tämä kuitenkin johti monen tilan julistamiseen, joka haittasi koodin selkeyttä, kuten organisaatioiden tietoruuduissa olevassa kuvagalleriassa (kuva 25).

```
17  export default function CompanyThumbnail(props) {
18      const [displayedThumbnail, setThumbnail, displayedThumbnailRef] = useState(props.openImageIndex);
19      const [lastDisplayedThumbnail, setLastDisplayedThumbnail, lastDisplayedThumbnailRef] = useState(props
20      const [thumbnailSet, setThumbnails, thumbnailSetRef] = useState(props.thumbnails);
21      const [maxThumbnailsVisible, setMaxThumbnailsVisible, maxThumbnailsVisibleRef] = useState(Math.min(pr
22      const [thumbnailsStart, setThumbnailsStart, thumbnailsStartRef] = useState(props.startAt);
23      const [isMouseOverThumbnail, setMouseOverThumbnail] = useState(-1);
24      const [isAutoSwitchEnabled, setAutoSwitchEnabled] = useState(false);
25      const [currentAutoSwitchImage, setCurrentAutoSwitchImage, currentAutoSwitchImageRef] = useState(props
26      const [dragStart, setDragStart, dragStartRef] = useState({ x: 0, y: 0 });
27      const [mousePosition, setMousePosition, mousePositionRef] = useState({ x: 0, y: 0 });
28      const [previewPosition, setPreviewPosition, previewPositionRef] = useState(props.lastSavedPreviewPosi
29      const [isDragging, setDragging, isDraggingRef] = useState(false);
30      const [dragInterval, setDragInterval, dragIntervalRef] = useState(null);
31      const [lastDragResult, setLastDragResult, lastDragResultRef] = useState({ xStart: 0, xEnd: 0 });
32      const [galleryArea, setGalleryArea, galleryAreaRef] = useState(null);
33      const [thumbnailWidth, setThumbnailWidth, thumbnailWidthRef] = useState(150);
34      const [addedMovement, setAddedMovement, addedMovementRef] = useState({ current: 0, dec: 0 });
35      const [thumbnailOverflow, setThumbnailOverflow, thumbnailOverflowRef] = useState(props.overflow);
36      const [firstThumbnail, setFirstThumbnail, firstThumbnailRef] = useState(Math.floor(props.overflow / t
37      const [carouselLastActiveThumbnail, setCarouselLastActiveThumbnail, carouselLastActiveThumbnailRef] =
```

Kuva 25. Kuvagallerian tiloissa ei ole käytetty syviä tietorakenteita, vaan tilat on jaoteltu useisiin osiin.

Alun perin kuvagalleriassa määriteltyjä tiloja oli tarkoitus yhdistää, mutta tilojen muuttamisessa käytetty koodi sisälsi paljon toistoa ja oli hyvin hankalalukuista. Suosittelen immutability-helperin käyttöä varsinkin kuvagalleriassa tilojen määrän vähentämiseksi, sillä mahdolliset tilalisäykset komponenttiin vähentäisivät koodin selkeyttä entisestään. Vaihtoehtoisesti monimutkaisia tiloja hyödyntäviä komponentteja voidaan pyrkiä purkamaan erillisiin kokonaisuuksiin,

esimerkiksi kuvagallerian kuvakarusellissa kuvat voitaisiin toteuttaa komponentteina, jotka ovat erillään liikkuvasta pohjasta, johon kuvat asetetaan.

6 Kartan arkkitehtuuri

6.1 Karttanäkymän arkkitehtuuri

Virtuaaliesittelyssä näytillä olevien organisaatioiden osoitteen hahmottamiseksi, tietopaneelinäkymään tarvittiin kartta, joka näyttää organisaatioiden toimipisteet. Esittely käyttää kartan toteutukseen OpenLayers-kirjastoa, joka on avoimen lähdekoodin karttaratkaisu. OpenLayers on maksuton eikä veloita kehittäjiä sijaintikyselyistä. Lisäksi esittely hyödynsi Nominatim-osoitepalvelun tarjoamaa rajapintaa tilanteissa, joissa organisaatiot olivat toimittaneet osoitteen, mutta eivät toimipisteidensä koordinaatteja. (OpenLayers 2022; Nominatim 2022.)

Karttanäkymä piirretään React-komponentin mukana, mutta OpenLayers-kirjasto, jota käytetään ei ole React-implementaatio. Sen sijaan OpenLayersin tavallista JavaScriptiä hyödyntävää versiota on suosittu. Kartta haetaan React-komponentin latautuessa hyödyntämällä ensisijaisesti organisaatiotietokannasta saatuja koordinaatteja karttamerkkien asettamiseen. Mikäli tietokannassa oli saatavilla vain osoite komponentti lähettää kyselyn Nominatim-rajapintaan. Kyselyn onnistuessa komponentti saa vastaukseksi osoitteen koordinaatit, joita se käyttää hakemaan kartan ja asettamaan karttamerkit.

Kartan lisäys komponenttiin tapahtuu luomalla ensin uusi karttaolio (Map), jolle annetaan näkymän keskipiste, ulkoasun tyyppi sekä kohde -tagi React-komponentissa, johon kartta tullaan piirtämään. Kartan keskipiste lasketaan organisaation toimipisteiden koordinaattien keskiarvon perusteella, jolloin kartta näyttää kaikki toimipisteet. Kartan ulkoasuksi asetetaan TileLayer, jonka lähteenä toimii OSM. TileLayer hakee kartan tietoa “laatoittain”; karttatieto on asetettu ruudukkoon, joka on keskipisteen ympärillä, ja kun käyttäjä liikuttaa keskipistettä, näkymän ulkopuolelle jääneet ruudut poistetaan ja uudet ruudut tietoineen

ladataan. OSM, OpenStreetMap, on vapaasti käytettäviä karttoja tuottava palvelu (OpenStreetMap 2022). Kun kartan sisältävä React-komponentti piirretään, kartta etsii valmistuessaan kohde -elementin ja asettaa kartan toimintoineen siihen. React-komponentti piirtää karttamerkit, mutta OpenLayersin kartta asettaa merkit oikeille paikoilleen. Kartta piirretään uudelleen vain, jos “Yhteystiedot”-välilehti, jossa kartta on, ladataan uudelleen.

6.2 Karttanäkymän vaihtoehtoiset toteutukset

Toiminnallisessa opinnäytetyössään “Paikkatietorajapintojen ja karttakirjastojen hyödyntäminen ReactJS-sovelluksessa" Nestori Metsäranta tarkastelee erilaisten paikkatietorajapintojen ja karttakirjastojen hyödyntämistä React-sovelluksessa. Tarkasteltaviksi karttakirjastoiksi valittiin OpenLayers, Leaflet ja Mapbox GL, joita käyttäen toteutettiin pieni React-sovellus, jossa käyttäjä pystyi näkemään kartan, liikuttamaan karttaa ja zoomata karttanäkymässä sisään ja ulos. Ohjelmiston kehityksessä noudatettiin sovellettua vesiputousmallia aloittaen suunnittelulla, edeten toteutukseen sekä päättäen testauksella. Opinnäytetyön tavoitteena oli “päästä syvemmälle eri karttakirjastojen tarjoamiin vaihtoehtoihin ja vertailla niitä”. Keskityn tässä tekstissä ainoastaan OpenLayers-kirjaston toteutuksen esittelyyn, sillä kaikki opinnäytetyössä esitetyt sovellukset toimivat pitkälti samalla tavalla, ja OpenLayers-sovellus tuo parhaiten esille Karelian toteutuksessa hyödynnettäviä arkkitehtuuriratkaisuja. (Metsäranta 2021, 12—13.)

OpenLayers-kirjaston toteutuksen pohjana Metsäranta käytti Matthew Brownin Medium-artikkelia “How to Use OpenLayers Maps in React”, sillä dokumentaatiota ja neuvoja kirjaston integraatiosta React-sovellukseen oli tarjolla niukasti. Matthew Brown päätyi hyödyntämään artikkelissaan OpenLayersin tavallista JavaScript versiota, vaikka React-integraatio olikin saatavilla, koska integraation ei nähty olevan aktiivisessa kehityksessä, eikä kovin suosittu. Myös Metsärannan toteutuksessa käytettiin tavallista JavaScript versiota OpenLayersista. (Metsäranta 2021, 15, 18; Brown 2020.)

Toteutuksessa React-komponentti, joka piirtää kartan, sisältää kolme muuta React-komponenttia, jotka muodostavat karttanäkymän: <Map>, <Layers> ja <TileLayer>. <Map>-komponentin alaisuudessa on <Layers>-komponentin, joka sisältää kartan käyttämät ulkoasut; tässä tapauksessa <TileLayer>-komponentin. <Map>-komponentti sisältää kartan muodostuksen luoden karttaolion, jolle asetetaan Mapin isäntäkomponentilta saatu keskipiste, zoomaustaso sekä <Layers>-komponenttiin käärityt ulkoasut. Samalla kartalle määritellään -elementti, johon näkymä tullaan piirtämään kartan latauduttua. <TileLayer>-komponentti sisältää kartan ulkoasun määrittelyn. Kartta käyttää TileLayer-tyyppistä ulkoasua, joka hakee karttatietoa “laatoittain” näkymän keskipisteen ympäriltä sen paikan muuttuessa. Lähteenä TileLayer käyttää <Map>-komponentin isäntäkomponentilta saatua OSM-, OpenStreetMap, paikannusrajapintaa, jonka karttatieto piirretään ulkoasuun. Opinnäytetyössä ei käsitelty karttamerkkien asettelua. (Metsäranta 2021,19—20.)

6.3 Karttanäkymän parantaminen

Metsärannan toteutus valittiin vertailtavaksi, sillä opinnäytetyössä verrataan erilaisia karttaratkaisuja, joiden käyttöä harkittiin myös Karelian toteutuksessa. Jokaisella vertailtavalla karttakirjastolla toteutettiin Reactia hyödyntäen pieni karttanäkymä, joka oli hyvin samanlainen kuin oma toteutuksemme. OpenLayersin käyttöönotossa Metsäranta kohtasi samoja ongelmia kuin kehitystiimimme. OpenLayersin React-integraatioon ei ollut tarjolla juurikaan dokumentaatiota, ja saatavilla oli vain yksi Reactin kanssa yhteensopiva kirjasto, jonka suosio latausmäärän perusteella oli vähäinen. Epäsuositun kirjaston hyödyntäminen ei ole suositeltavaa, sillä sen kehitys tulevaisuudessa voi jäädä vähäiseksi, eikä ilmeneviä bugeja välttämättä korjata ajoissa, tai lainkaan. Molemmissa toteutuksissa päädyttiin käyttämään tavallista JavaScriptiä OpenLayersin integrointiin.

Opinnäytetyössään Metsäranta harkitsi GoogleMaps-karttakirjaston hyödyntämistä, mutta kirjasto hylättiin samoista syistä kuin Karelian toteutuksessa. GoogleMaps on Googlen kehittämä ja ylläpitämä karttakirjasto,

jota hyödynnetään esimerkiksi Googlen samannimisessä karttapalvelussa (Google, 2022a). GoogleMaps toimii samalla tavalla kuin OpenLayers, mutta tarjoaa enemmän toimintoja, kuten reitin laskun ja reittiohjeiden esityksen. Toisin kuin OpenLayers, GoogleMaps on maksullinen, minkä vuoksi GoogleMaps hylättiin Karelian toteutuksessa sekä Metsärannan toteutuksissa. GoogleMapsin karttarajapintaan tehtyjen kyselyiden kuukausihinnat tuhatta kyselyä kohden vaihtelevat $2—$14 välillä (Google, 2022b). Vaikka kulut eivät ole korkeat, rajapintaan rekisteröityminen ja kustannusten käsittely olisi aiheuttanut toimeksiantajan puolelta ylimääräisiä toimenpiteitä. Koska karttanäkymä oli melko yksinkertainen, OpenLayersin tarjoamat ilmaiset toiminnot nähtiin riittäviksi.

Metsärannan toteutuksessa karttanäkymä koostuu kolmesta komponentista: <Map>, <Layers> ja <TileLayer>. Ehdotan, että samaa rakennetta käytettäisiin myös Karelian toteutuksessa, jossa karttanäkymä toteutettiin kokonaisuudessaan yhteen React-komponenttiin. Mikäli karttaan lisättäisiin tulevaisuudessa erilaisia toimintoja, karttakomponentissa olevan koodin määrä kasvaisi haitaten ohjelman selkeyttä. Hajauttamalla karttanäkymän osia useampaan komponenttiin ratkaisun skaalautuvuus paranisi tilanteessa, jossa karttaan haluttaisiin lisätä useita ulkoasuja. Tällöin ulkoasut voitaisiin toteuttaa erillisinä komponentteina, jotka asetettaisiin <Layers>-komponentin alaisuuteen, jolloin <Map>-komponentti hakisi ne automaattisesti. Karelian ratkaisussa kaikki ulkoasut on määriteltävä samassa komponentissa, jolloin uusien ulkoasujen lisääminen merkitsisi uuden koodin lisäämistä karttakomponenttiin.

Vaikka Metsäranta kertoi opinnäytetyössään pettyneensä OpenLayers-kirjastoon React dokumentaation vähäisyyden takia, mielestäni OpenLayersin käyttöä voidaan silti jatkaa. Olettaen, että toimeksiantajan vaatimukset kartalle ovat pysyneet pitkälti samoina, OpenLayersilla toteutettu näkymä toimii edelleen toimeksiantajan vaatimusten mukaisesti. Tavallisella JavaScriptillä toteutettu karttaratkaisu toimii Reactin rinnalla odotetusti, mutta karttanäkymän skaalautuvuutta voidaan parantaa Metsärannan esittämällä komponenttien hajautuksella.

7 Pohdinta

Opinnäytetyössä käsitellyt osa-alueet on toteutettu mielestäni pääosin onnistuneesti, sillä ohjelmiston toiminnallisuudet täyttävät asiakkaan vaatimukset, ja komponenttien – kuten 360°-näkymän ja karttanäkymän – vastuualueet on selkeästi rajattu. Suurimmaksi ongelmaksi osoittautui komponenttien suuri koodimäärä, jota voitaisiin vähentää pilkkomalla komponentteja pienempiin komponentteihin tai erottelemalla komponenttien sisältämiä kokonaisuuksia erillisiin tiedostoihin, kuten 360°-näkymän räätälöityjä komponentteja rekisteröitäessä.

360°-näkymän toteutuksesta A-Framella oli saatavilla niukasti vaihtoehtoisia toteutuksia, mutta projektimme ja muiden toteutusten välillä oli useita yhtäläisyyksiä, esimerkiksi A-Framen resurssienhallintajärjestelmän hyödyntäminen. 360°-näkymän komponentissa on kuitenkin suuri määrä koodia, sillä A-Framen räätälöidyt komponentit rekisteröidään samassa tiedostossa. Suosittelen, että komponenttien rekisteröinti siirrettäisiin erilliseen tiedostoon koodin selkeyttämiseksi ja skaalautuvuuden parantamiseksi, mikäli komponentteja rekisteröidään enemmän tulevaisuudessa. Näkymän toteutuksessa hyödynnetään entiteettien yhdistelyä toistensa alaisuuteen, mutta Karelian toteutuksessa yhdistelytapa on epäjohdonmukainen; entiteettejä yhdistellään React-sirpaleilla ja tyhjillä A-Frame-entiteeteillä. Ehdotan, että vain yhtä yhdistelytapaa käytettäisiin, sillä kehittäjät saattavat viitata entiteetteihin tai sirpaleisiin tulevaisuudessa, tai lisätä niihin muita ominaisuuksia, joita ei ole mahdollista toteuttaa yhtenäisellä tavalla molemmille yhdistelytavoille. Suosittelen lisäksi siirtymistä A-Framen React-integraatiosta tavallisen JavaScriptin versioon. Integraation pakettivaraston ylläpitäjä ehdottaa siirtymistä tavallisen JavaScriptin versioon, mikä saattaa johtaa integraation ylläpidon loppumiseen tulevaisuudessa.

Kehitimme Karelian toteutukseen oman kielitukemme, jonka toimintaperiaate on pitkälti samanlainen kuin vaihtoehtoisessa toteutuksessa, jossa kielitukena

käytettiin i18next-sovelluskehystä. Koska molemmat toteutukset hyödyntävät ulkoisia käännöstiedostoja, jotka muodostuvat avain-käännös-pareista, siirtyminen i18nextiin on todennäköisesti hyvin helppo. Käännöstiedostoihin ei tarvitse tehdä muutoksia, ja kielitukea hyödyntäviin komponentteihin on tehtävä minimaalisia muokkauksia käännösten hakemiseksi. Suosittelen i18next-sovelluskehykseen siirtymistä, sillä i18next tarjoaa myös ylimääräisiä ominaisuuksia, kuten nimiavaruuksien ja monikkomuotojen käytön käännöstiedostoissa. I18nextin hyödyntäminen oman toteutuksen sijasta myös käytännössä ulkoistaisi kielituen kehityksen ja ylläpidon.

Karelian toteutuksessa hyödynnettiin OpenLayers-karttakirjastoa, jota käytettiin maksullisen GoogleMaps-kartan sijasta. Vaihtoehtoisessa toteutuksessa kehitetty OpenLayers-kartta luotiin pääasiassa samoja elementtejä käyttäen, mutta toisin kuin Karelian toteutuksessa, elementit oli jaoteltu erillisiin, pienempiin komponentteihin. Ehdotan vaihtoehtoisessa toteutuksessa esitettyä jakoa karttanäkymän arkkitehtuurin skaalautuvuuden parantamiseksi, mikäli karttaan halutaan tulevaisuudessa lisätä muita ominaisuuksia, kuten näppäimiä tai ylimääräisiä kerroksia (ulkoasuja). Opinnäytetyössä ei käsitelty GoogleMaps-rajapinnan käyttöä tämän maksullisuuden vuoksi. Mikäli toimeksiantaja on valmis maksamaan rajapintakutsuista, ehdotan GoogleMaps-rajapinnan harkitsemista ja tutkimista, sillä GoogleMaps tarjoaa suuremman määrän ominaisuuksia sekä aktiivisen ylläpidon.

Lopuksi ehdotan immutability-helper-kirjaston hyödyntämistä Karelian toteutuksessa syviä tietorakenteita käsiteltäessä. Vaikka tietorakenteita voidaan joskus käsitellä olemassa olevilla map- ja filter-metodeilla, tai tietorakenteita käyttävät komponentit voidaan pilkkoa pienempiin osiin, joissa kullekin osalle annetaan oma osuus tietorakennetta, on tilanteita, joissa syviä tietorakenteita on käytettävä, kuten kuvagallerian toteutuksessa. Immutability-helper-kirjasto helpottaa koodin luettavuutta syviä tietorakenteita muokattaessa.

Lähteet

A-Frame. 2022c. Asset Management System. https://aframe.io/docs/1.2.0/core/asset-management-system.html. 8.6.2022.

A-Frame. 2022b. Entity-Component-System. https://aframe.io/docs/1.2.0/introduction/entity-component-system.html. 8.6.2022.

A-Frame. 2022a. Introduction. https://aframe.io/docs/1.2.0/introduction/. 8.6.2022.

Ahir, K. 2018. How to create a Virtual Tour using A-Frame. Medium. 21.12.2018. https://medium.com/detaux/how-to-create-a-virtual-tour-using-a-frame-164941fea573. 8.6.2022.

Brown, M. 2020. How to Use OpenLayers Maps in React. Medium. 3.10.2018. https://medium.com/swlh/how-to-incorporate-openlayers-maps-into-react-65b411985744. 8.6.2022.

Facebook. 2022. react, GitHub. https://github.com/facebook/react. 8.6.2022.

Google. 2022a. Build awesome apps with Google's knowledge of the real world. https://developers.google.com/maps. 8.6.2022.

Google. 2022b. Pricing that scales to fit your needs. https://mapsplatform.google.com/pricing/. 8.6.2022.

Huusko, A. 2022. Kuvapainoitteisten verkkosivujen optimointi Node.js-ympäristössä. Karelia-ammattikorkeakoulu. Tietojenkäsittelyn koulutus. Opinnäytetyö. https://urn.fi/URN:NBN:fi:amk-2022052812722. 28.8.2022.

i18next. 2022d. Comparison to others. https://www.i18next.com/overview/comparison-to-others. 8.6.2022.

i18next. 2022b. Essentials. https://www.i18next.com/translation-function/essentials. 8.6.2022.

i18next. 2022a. Introduction. https://react.i18next.com. 8.6.2022.

i18next. 2022e. Plurals. https://www.i18next.com/translation-function/plurals. 8.6.2022.

i18next. 2022c. useTranslation (hook). https://react.i18next.com/latest/usetranslation-hook. 8.6.2022.

Kim, A. 2022. react-openlayers, npm. https://www.npmjs.com/package/react-openlayers. 8.6.2022.

Kolodny, M. 2022. immutability-helper, GitHub. https://github.com/kolodny/immutability-helper. 8.6.2022.

Mercedes-Benz. 2020. Mercedes-Benz KSA Showroom. https://www.mercedesbenzksa.com/vr-showroom. 11.8.2022.

Metsäranta, N. 2021. Paikkatietorajapintojen ja karttakirjastojen hyödyntäminen ReactJS-sovelluksissa. Hämeen ammattikorkeakoulu. Tietojenkäsittelyn koulutus. Opinnäytetyö. https://urn.fi/URN:NBN:fi:amk-2021121325641. 8.6.2022.

Microsoft. 2022. AzureDevOps. https://azure.microsoft.com/en-us/services/devops/. 8.6.2022.

Mozilla. 2022b. Array.prototype.filter().

https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/filter. 8.6.2022.

Mozilla. 2022a. Array.prototype.map(). https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Array/map. 8.6.2022.

Mozilla Mixed Reality. 2015. Introducing A-Frame: Building Blocks for WebVR. Mixed Reality Blog. 16.12.2015. Blogi. https://blog.mozvr.com/introducing-aframe/. 8.6.2022.

Nguyen, H. 2019. Developing a restaurant web application using JavaScript libraries. Vaasan ammattikorkeakoulu. Tietojenkäsittelyn koulutus. Opinnäytetyö. https://urn.fi/URN:NBN:fi:amk-2019112823195. 8.6.2022.

Nominatim. 2022. Open-source geocoding with OpenStreetMap data. https://nominatim.org. 8.6.2022.

OpenLayers. 2022. Overview. https://openlayers.org. 8.6.2022.

OpenStreetMap. 2022. OpenStreetMap. https://www.openstreetmap.org/about. 8.6.2022

React. 2022a. A JavaScript library for building user interfaces. https://reactjs.org/ . 8.6.2022.

React. 2022b. Components and Props. https://reactjs.org/docs/components-and-props.html. 8.6.2022.

React. 2022d. Hello World. https://reactjs.org/docs/hello-world.html. 8.6.2022.

React. 2022c. Hooks at a Glance. https://reactjs.org/docs/hooks-overview.html. 8.6.2022.

React. 2022e. Immutability Helpers. https://reactjs.org/docs/components-and-props.html. 8.6.2022.

Santos, S. & Cardoso, J. C. S. 2019. A-Frame experimentation and evaluation for the development of interactive VR: a virtual tour of the Conimbriga Museum. University of Coimbra. CISUC, DEI. Tekninen raportti. http://hdl.handle.net/10316/87028. 8.6.2022.

Styled-components. 2022. Getting started. https://styled-components.com. 8.6.2022.

Supermedium. 2022. aframe-react, GitHub. https://github.com/supermedium/aframe-react. 8.6.2022.