Opinnäytetyö (AMK)

Tieto- ja viestintätekniikan koulutus

2019

Saku Hirvonen

REACT-POHJAINEN VASTAAJAN NÄKYMÄ

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tieto- ja viestintätekniikka

2019 | 16 sivua

Saku Hirvonen

REACT-POHJAINEN VASTAAJAN NÄKYMÄ

Tämän opinnäytetyöprojektin tarkoituksena oli luoda REACT-tekniikkaan perustuvan loppukäyttäjän näkymän runko HARAVA-sovellukselle, joka korvaisi nykyisen vanhentuneen loppukäyttäjän näkymän. Tieto uuteen sovellukseen tuodaan REST-tyylillä erillisen sovelluksen kautta, ja koodi kehitetiin Visual Studio Code -ohjelmalla. Tavoitteena oli, että uusi sovellus on nopeampi ja käyttäjäystävällisempi, kuin aikaisempi sovellus. Projektin lopputulos oli karkea prototyyppi, jonka jatkokehitys jäädyttettiin toimeksiantajan asettaessa voimavaroja muihin projekteihin.

ASIASANAT:

REST,REACT,JavaScript, loppukäyttäjä.

BACHELOR´S / MASTER'S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Information and Communications Technology

2019 | 16 pages

Saku Hirvonen

REACT-BASED END USER VIEW

The goal of this thesis project was to create a base for a new REACT-based end user view for the HARAVA application, which would replace the old, outdated HARAVA end user view. The data was transferred with a REST style application and the code was created with Visual Studio Code. The new version should also be faster and more user friendly than the older version. The end result of the project was a rough prototype, which was stored for possible future development because the commissioning company allocated resources to other projects.

KEYWORDS:

REST,REACT,JavaScript,end-user.

SISÄLTÖ

1 JOHDANTO **6**

2 TEKNIIKAT JA TYÖKALUT **7**

2.1 REST-verkkoarkkitehtuurityyli 7

2.2 REACT-ohjelmointikieli 9

2.3 Visual Studio Code -ohjelmointialusta 10

3 PROJEKTIN OHJELMOINTI **11**

3.1 Havaintoja REACT ympäristön käyttöönotosta 12

3.2 Implementointi 13

4 PROJEKTIN LOPPUTULOKSEN POHDINTA **15**

LÄHTEET **16**

KUVAT

Kuva 1. Esimerkki REST-tiedonkeräämiskoodista 13~~14~~

Kuva 2. Sivu muodostetaan komponenteista kuvan mukaisesti 14

SANASTO

Angular	Googlen ylläpitämä JavaScript-ohjelmistokehys
Git	Versionhallintaohjelmisto
HTML	Hypertekstin merkintäkieli (HyperText Markup Language)
HTTP	Hypertekstin siirtoprotokolla (HyperText Transfer Protocol)
JavaScript	Netscapen kehittämä komentosarjakieli
JSON	JavaScript-kohteen merkintätapa (JavaScript Object Notation)
JSX	Lisäosa JavaScript kieliluetteloon (JavaScript XML)
REACT	Facebookin luoma JavaScript-kirjasto
REST	Edustuksellinen tilan muutos (Representational State Transfer)
Virtual DOM	Virtuaalinen dokumentti kohdemalli (Virtual Document Object Model)
XML	Laajennettava merkintäkieli (Extensible Markup Language)

1 JOHDANTO

Dimenteq, paikkatietopalveluihin erikoistuvia sovelluksia kehittävä yritys, tarvitsee uuden version sovelluksestaan Haravasta, koska vanha sovellus käyttää vanhaa tekniikkaa ja näyttää vanhanaikaiselta. Harava on pilvipalveluna toimiva kyselypalvelu, jonka avulla on mahdollisuus luoda erilaisia kyselyjä, jotka voivat käyttää karttapalveluita. Sovelluksen modernisoinniksi päätettiin käyttää tekniikoita REST ja REACT tiedonsiirron nopeuttamiseksi ja vastaajan näkymän koodin modularisoinniksi ja modernisoimiseksi.

REACT:lla luotu modulaarisempi koodi mahdollistaa koodin helpomman muokkaamisen tulevaisuudessa. REACT on komponentteihin pohjautuva Javascript-kirjasto. Sovellus saa tietonsa REST:iin pohjautuvasta verkkoarkkitehtuurista. Tämän opinnäytetyön tarkoituksena on luoda esikoisversio uudesta vastaajan näkymästä ilman muotoilua. Vastaajan näkymällä tarkoitetaan Haravan kyselyn vastaajan käyttöliittymää. Projektin lopputuloksen perusteella päätetään jatketaanko projektia loppuun asti vai pitäydytäänkö vanhan version päivittämisessä ja kehittämisessä.

2 TEKNIIKAT JA TYÖKALUT

Tässä luvussa tehdään selväksi projektin oleellisimmat tekniikat ja työkalut.

2.1 REST-verkkoarkkitehtuurityyli

Representational State Transfer (REST) on HTTP-protokollaan perustuva verkkoarkkitehtuurityyli, jolla saa siirrettyä tietoa rajapinnasta toiseen verkon kautta muuttumattomana ja tilattomasti. REST-termin määritteli ja esitteli Roy Fielding vuonna 2000 Kalifornian Irvinen yliopistossa väitöskirjassaan ”Architectural Styles and the Design of Network-based Software Architecture”.

REST:n kautta sovellus voi pyytää tietoa tietyllä osoitteella, jolloin REST-sovellus lähettää pyytäneelle sovellukselle tiedon esim. JSON-, XML- tai HTML-muodossa. Kaikki tarvittava tieto lähetetään samassa paketissa, mikä poistaa tarpeen usealle yhteyden otolle asiakkaan ja palvelimen välille. Tilaton tiedonlähetys mahdollistaa esimerkiksi kuvan piirtämisen näytölle rinnakkain itse resurssin lataamisen kanssa luo paremman käyttäjäystävällisyyden.

REST:n arkkitehtuurisiin ominaisuuksiin kuuluvat muun muassa: skaalautuvuus, yhtenäisen rajapinnan yksinkertaisuus, komponenttien muokattavuus, komponenttien välisen viestinnän näkyvyys, hyvä suoritus kyky komponenttien yhteistoiminnassa, komponenttien liikuteltavuus ja luotettavuus virheiden käsittelyssä. [1][2]

Skaalautuvuus mahdollistaa tuen suurelle määrälle eri komponentteja vähentämällä komponenttien yhdistäjien semantiikan monimutkaisuutta. Yhtenäisen rajapinnan yksinkertaisuus antaa monien eri ohjelmien jakaa toiminnallisuutta keskenään ja muodostaa oman standardinsa tietynlaisten sovellusten toiminnalle.

Komponenttien muokattavuus antaa mahdollisuuden muovata koodia tarvittavien muutoksien mukaiseksi ilman että koko järjestelmää pitää muokata. Komponenttien välisen viestinnän näkyvyys helpoittaa virheenkorjausta ja ohjelmointia.

Komponenttien liikuteltavuus mahdollistaa ohjelmointikoodin liikuttamista tiedon ohessa. Luotettavuus virheiden käsittelyssä vähentää koko järjestelmän aiheuttavan kaatu-

mista, kun virhe tapahtuu komponentissa. Tällöin virheellisesti toimiva komponentti kaatuu, kun muut komponentit jatkavat toimintaansa.

REST:n määrittelee myös seuraavat kuusi suuntaa antavaa arkkitehtuurista rajoitetta: asiakas-palvelin arkkitehtuuri, tilattomuus, tiedon varastoimisuus, tasoitettu järjestelmä, koodi vaatimuksen mukaan ja yhteneväinen rajapinta. [1]

Asiakas-palvelin arkkitehtuurissa asiakkaan järjestelmän ja palvelimen järjestelmän tulee toimia erillään toisistaan, mahdollistaen palvelimen toiminnan useamman erilaisen asiakkaan järjestelmän kanssa.

Tilattomuudessa asiakkaan tietoja ei säilytetä palvelimen puolella, vaan kaikki tarvittavat tiedot kulkevat saman kutsun mukana palvelimelle ja kutsun tila säilytetään vain asiakkaan päässä.

Tiedon varastoimisuus vähentää viestintää asiakkaan ja palvelimen väliltä, parantaen suorituskykyä, kun saman tiedon voi saada välimuistista asiakkaan omalta koneelta. On huomiotava, että kaikki pyynnöt eivät sovellu välimuisti käyttöön, esimerkiksi kun asiakkaan on saatava päivitettyä tietoa kannasta.

Tasoitettu järjestelmässä on välipalvelimia, joissa voidaan käsitellä tiedonsiirron turvallisuutta, tai palvelinten ylikuormitusta. Asiakas ei usein näe välipalvelimia kun tehdään pyyntöjä kantaan.

Koodi vaatimuksen mukaan. Palvelimet voivat myös väliaikaisesti lisätä tai muokata asiakkaan toimintaa lähettämällä valmiiksi aseteltua koodia liitännäisenä.

Yhteneväinen rajapinta on määritelty seuraavalla neljällä tavalla: Resurssin tunnistaminen pyynnöissä, resurssin manipulointi esityksessä, Itsensä selittävät viestit ja hypermedia sovellusten tilan moottorina. Eli resurssit tulee pystyä tunnistamaan osoitteistaan, jokaisessa resurssissa tulisi olla tarpeeksi tietoa mukana mahdollistaakseen resurssin poistamisen tai muokkaamisen. Jokaisessa pyynnössä tulisi olla tarpeeksi tietoa siitä, miten sitä käsitellään asiakkaan päässä ja asiakkaan pään pitää pystyä löytämään tarvittavat toiminnot ja resurssit, joita palvelimen antamat linkit tarvitsevat.

Nämä rajoitteet vähentää tapoja, joilla palvelin voi käsitellä ja vastata asiakkaan pyyntöihin. Jos järjestelmä ei sisällä näitä rajoitteita, sitä ei voida kutsua REST-tyyliseksi.

Pyynnöt asiakkaan ja palvelimen välillä toimii POST, GET, DELETE, PUT, HEAD, OPTIONS ja PATCH HTTP käskyillä. GET hakee tietoa annetulla osoitteella tietystä resurssista. POST antaa tietoa käsiteltäväksi tiettyyn resurssiin. PUT päivittää tietyn resurssin. HEAD hakee resurssin metatiedot. DELETE poistaa tietyn resurssin. OPTIONS palauttaa tuetut HTTP metodit ja PATCH päivittää vaillinaisesti resurssin. [3]

2.2 REACT-ohjelmointikieli

REACT on javascript kirjasto, joka on kehitetty nettikäyttöliittymien rakentamiseen. Yleisesti sitä käytetään puhelinsovellusten rakentamiseen. REACT:n kehitti Facebookille työskentelevä insinööri Jordan Walke. Facebook käytti REACT:a jo vuonna 2011 sovelluksessaan, ja se julkaistiin avoimeksi koodiksi 2013. REACT Native, joka on luotu älypuhelin sovellusten ohjelmointiin, julkaistiin 2015. [6]

REACT:n huomioitavia ominaisuuksia ovat mm. yhdensuuntainen tiedon sitominen ominaisuuksiin, virtuaalinen dokumentin kohdemalli (Virtual DOM) ja JSX. Ensimmäisessä tapauksessa ylemmän luokan komponentti siirtää ominaisuuksia alemman luokan komponentille, tieto tulee yhtenä muuttumattomana kohteena (Javascript object). Toisessa tapauksessa sovellus muodostaa automaattisesti koodin mukaisen näkymän avatulle verkkosivulle jokaisen koodimuutoksen tallennuksen jälkeen ja nopeuttaa ohjelmointia poistamalla rakennusvaiheen, joka pitää tehdä useammassa muussa ohjelmointikielessä. [4]

JSX (JavaScript eXtension) on laajennettu Javascript-kielen syntaksi, eli alkuperäiseen Javascript kieleen on lisätty uusia komentoja. Se muistuttaa ulkomuodoltaan HTML koodia, mutta se mahdollistaa eri komponenttien mudostamisen kehittäjille tutuilla syntakseilla.

REACT:n toiminta perustuu erilaisiin komponentteihin. Komponentit muodostavat annetusta tiedosto kohteita, jotka voivat olla vaikka painettavia nappeja tai karttaruutuja. Sovellus kutsuu näitä komponentteja halutussa järjestyksessä tai tietyllä tavalla rakentaen käyttäjän näkemän käyttöliittymän. Sovellus päivittää jatkuvasti vastaajan näkymää muuttaen sitä resurssin muuttuessa. [4]

2.3 Visual Studio Code -ohjelmointialusta

Yrityksellä ei ollut tarjota mitään erityistä sovellusta, jolla REACT:a pystyy ohjelmoida ketterästi, joten kevyen tutkimuksen jälkeen päädyttiin käyttämään helposti ja vapaasti saatavilla olevaan Visual Studio Code-ohjelmointialustaan. Visual Studio Code on Microsoftin kehittämä lähdekoodin muokkaussovellus. Se julkistettiin vuonna 2015 [5].

Kyseisellä sovelluksella pystyy ohjelmoiman useilla eri kielillä, ja se on erittäin suosittu kehityssovellus. Sovelluksen vahvuuksiin kuuluu sen vahva muokattavuus ja laajennettavuus. Git, joka on koodinversionhallintaohjelma, on komennoiltaan yhteensopiva Visual Studion kanssa, sillä Git-komennot on rakennettu valmiiksi sovellukseen. Visual Studio Code osoittautui sopivaksi ohjelmointialustaksi projektille.

3 PROJEKTIN OHJELMOINTI

Projektin tilaajat päättivät, että työssä tulisi käyttää REACT javascript -kirjastoa ohjelmoinnin pohjana yrityksen aikaisemman kokemuksen perusteella. Yrityksen vanhemmat työntekijät totesivat, että se olisi parempi tulevaisuuden kannalta, koska koodista tulisi moduulaarisempi. Tämä mahdollistaisi yksittäisten osien ohjelmointia erillään toisistaan. Näin muokkaamalla yhtä osaa ei tule hajoitettua koodia toisilta osilta. REST-arkkitehtuuri otettiin käyttöön projektia varten, koska sen avulla sovelluksen käyttämä tieto saadaan jatkossa siirrettyä myös tämän projektin ulkopuolisille rajapinnoille ketterästi. REST-rajapinta rakennettiin tämän projektin yhteydessä omana projektinaan, mutta sen lähettämää tietoa kyselyjen sisällöstä käytettiin projektissa.

Projektin sovelluksen on tarkoitus saada JSON-muodossa kyselyn sisältö, joka on luotu erillisellä Harava-ylläpitäjän sovelluksella. Tämä sisältää mm. kysymyssivujen määrän, kysymysten määrän sekä erilaiset täytetiedostojen osoitteet. Kun vastaaja on vastannut kyselyyn, vastaukset kerätään ja lähetetään takaisin Harava-ylläpitäjän sovellukseen REST:n avulla JSON-muodossa. Tämä tieto tallennetaan kantaan myöhempää analyysia varten. Projektin tieto täytyy siis muuttaa lopuksi muotoon, jota REST rajapinta ymmärtää.

Toisena tavoitteena projektilla on luoda mahdollisimman minimalistinen käyttöliittymä käyttäjäystävällisyyden parantamiseksi. Saman käyttöliittymän tulisi näyttää ja toimia yhtähyvin puhelimella ja pöytäkoneella.

Projekti aloitettiin tutkimalla mahdollisuutta luoda uusi käyttöliittymä vastaajille uudella tekniikalla. Tutkimukseen osallistui useampi työntekijä, ja tuloksia käytettiin suuntaa antavana ohjeena projektityöläisille siitä, miten projektin tulisi todennäköisesti näyttää. Projektiin asetettiin yksi työntekijä (Opinnäytetyön tekijällä), jolla ei ollut aikaisempaa kokemusta REACT:sta tai REST:sta, ja vain vähäistä kokemusta javascript ohjelmoinnista. Projekti opettikin paljon kyseisistä tekniikoista ja paljon muutakin ohjelmoinnin hyvistä tavoista ja käytännöistä.

3.1 Havaintoja REACT ympäristön käyttöönotosta

REACT:n perusteet onnistuu opettelemaan REACT:n omilta nettisivuilta löydetyn tutoriaalin läpikäynnillä. Visual Studio Code osoittautui käteväksi alustaksi aloittelevalle ohjelmoijalle sen yksinkertaisuuden ja muokattavuuden ansiosta. REACT koodiin tehdyt muutokset saa nähtyä nopeasti ja helposti. Jokaisen koodin tallennuksen jälkeen palvelin päivittää näkymän verkkosivulla automaattisesti. Samalla jos koodissa on ongelmia, virheviesti näkyy verkkosivulla.

REACT:ssa on tarkoitus luoda komponentteja erilaisista koodin osista modulaarisen koodauksen tyylin. Olisi hyvä ennen koodauksen aloitusta miettiä etukäteen mitä erilaisia moduuleja projektiin tulisi tehdä, Ensikertalaisen virhe on luoda kaikki koodi samaan tiedostoon. REACT:iin saa internetistä paljon valmiiksi tehtyjä komponentteja, joiden käyttöä tosin tulisi harkita mahdollisten tietoturva ja omistajuus ongelmien takia. Erityisesti jos tehtävää sovellusta myydään, kannattaa nämä komponentit itse tehdä.

REACT ohjelmoinnin omaksuminen sujuu huomattavasti nopeammin jos työorganisaatiossa olisi muita kielentaitajia ohjeistamassa. REACT:a tarjoaa myös kattavan tutoriaalin omilla sivuillaan, jonka avulla pystyy nopeasti aloittaa oman nettisivun kehittämisen.

3.2 Implementointi

Projektin koodin kehittäminen oli hankalaa. Verkkosivu sisältö perustui REST:n kautta tulleeseen JSON tietoon, jonka tiedonrakenne ja avainsanat muuttuivat projektin aikana. Koodi (Kuva 1.), jolla tieto haettiin REST-sovelluksesta, joka muutti muotoaan useasti. Prototyypissä ei vielä kehitetty automatisaatiota, vaan tiedonhakuun oleelliset muuttujat pidettiin vakiona testaamista varten.

```
componentDidMount = ()=>{           //Rest JSON:n hakeminen
  if(this.state.ladannut !== true){
    const username = 'saku.hirvonen@dimenteq.fi';
    const apiKey = '4K9sdad135oIDFA3';
    const infoId = 1133;
    const today = new Date();
    const apiKeyHash = SHA1(apiKey + '&' + username + '&' + today.toISOString().split('T')[0]).toString();
    fetch('http://localhost:8080/HaravaRest/kysely/?tiedonhankinnanId=' + infoId + '&apiKey=' + apiKeyHash).then(function(response){
      return response.json();
    }).then(data =>{this.setState({id :  data.tiedonhankinnanId,
                nimi :  data.kyselynNimi,
                Kielet :  data.kyselynKielet,
                valittukieli : data.kyselynValittuKieli,
                Kyselysivut : data.kyselynSivut,
                sivujarjestys : data.kyselynSivujenIdt,
                kyselynLogo : data.kyselynLogonPolku,
                sivunumero : data.kyselynSivujenIdt.length,
                kiitossivu : data.kiitossivunId,
                ladannut : true});
    })
    }
  };
```

Kuva 1. Esimerkki REST-tiedonkeräämiskoodista

Sovelluksen eri komponentit muodostavat lopullisen näkymän (Kuva 2.) riippuen REST:n kautta saadusta tiedosta. Kaikki eri sivutyypit alustettiin projektiin, mutta sisältöä jokaiseen ei pystynyt vielä muodostamaan.

```
render() {

    return (
    <div className="App">
      <div className="App-header">
        <img src={this.state.kyselynLogo} className="Applogo" alt="logo" />
        <h1>{this.state.nimi}</h1>
        <h2>{this.state.mainheader[this.state.kielinumero]}</h2>
      </div>
      <div className="Language-button" onClick={() => this.kielisyys(this.state.kielet[this.state.kielinumero])}>
      <button className="Language">
        {this.state.buttonText[this.state.kielinumero]}
      </button>
    </div>
      <br></br>
      <div className="pagecontent">
        {this.sivusisalto(this.state.NykySivu, this.state.ladannut, this.state.ladannut2)}
      </div>
    <div className="NextPage-button" >
      {this.sivunvaihtotakas(this.state.NykySivu,this.state.ladannut)} {this.sivunvaihtoeteen(this.state.NykySivu,this.state.ladannut)}

      </div>
        <div className="SubQuestion-button">
          </div>
      <div className="PageNumber">
        {this.sivujenmaara(this.state.ladannut,this.state.sivunumero,this.state.NykySivu)}
        </div>
  </div>
  );
  }
}

export default App;
```

Kuva 2. Sivu muodostetaan komponenteista kuvan mukaisesti

Ohjelmoitavia kyselyn komponentteja olivat mm. eri kysymyssivut ja eri kysymystyypit. Muotoilua ei lisätty ja kyselyiden karttakomponenttia ei kehitetty tätä projektia varten, koska projektin omistaja ei ollut päättänyt mitä karttamoottoria käytettäisiin, ja miltä lopputuotteen tulisi näyttää.

4 PROJEKTIN LOPPUTULOKSEN POHDINTA

Valmistunut prototyyppi mahdollistaa Harava-kyselyjen esittämisen vastaajille vähemmällä koodilla ja nopeammin. Koodia ei ole muotoiltu, koska yrityksessä ei olla päätetty visuaalisesta ulostulosta. Projektin jatkokehitys jäädytettiin yrityksen toimesta valmistumisen jälkeen. Yritys fuusioitui toiseen isompaan yritykseen, jonka toimintamallissa käytetään Angular Javascript-kirjastoa REACT:n sijaan.

Harava sovelluskin on jäänyt uudessa yrityksessä kehityksessä taka-alalle tärkeämpien projektien takia, joten tämä projekti jää yrityksen tiedostoihin odottamaan tulevaa mahdollista kääntämistä Angular-kielelle. Projektin kehittäjä oppi paljon Javascript-kielistä projektin aikana, ja yleisesti uuden sovelluksen koodin kehittämisestä, aikataulutuksesta, haasteista ja hallinnoinnista.

LÄHTEET

[1] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. Chapter 5 [Viitattu 21.2.2019](Sähköinen). Saatavissa https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm

[2] Fielding, Roy Thomas. *Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000. Chapter 2 [Viitattu 21.2.2019](Sähköinen). Saatavissa https://www.ics.uci.edu/~fielding/pubs/dissertation/net_app_arch.htm

[3]Google Developers. Intro to REST. 2008. [Viitattu 21.2.2019](Sähköinen). Saatavissa https://www.youtube.com/watch?v=YCcAE2SCQ6k

[4] Facebook inc. Reactjs main page.[Viitattu 21.2.2019](Sähköinen) Saatavissa https://reactjs.org

[5] John Montgomery. BUILD 2015 News: Visual Studio Code, Visual Studio 2015 RC, Team Foundation Server 2015 RC, Visual Studio 2013 Update 5. 2015. [Viitattu 17.3.2019](Sähköinen). Saatavissa https://devblogs.microsoft.com/visualstudio/build-2015-news-visual-studio-code-visual-studio-2015-rc-team-foundation-server-2015-rc-visual-studio-2013-update-5/

[6] TXJS. Pete Hunt | TXJS 2015. 2015. [Viitattu 17.3.2019](Sähköinen) Saatavissa https://www.youtube.com/watch?v=A0Kj49z6WdM