Haaga-Helia
ammattikorkeakoulu Oy

Hallittavan infonäyttö-ratkaisun kehittäminen Craneworks Oy Ltd:lle

Suzan Kinnunen

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2019

Haaga-Helia
ammattikorkeakoulu Oy

Tiivistelmä

Tekijä(t)
Suzan Kinnunen

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Raportin/Opinnäytetyön nimi

Hallittavan infonäyttö-ratkaisun kehittäminen Craneworks Oy Ltd:lle

Sivu- ja liitesivumäärä
28 + 5

Opinnäytetyönä toteutetaan työnantajalle, eli Craneworks Oy Ltd:lle, Javascript-pohjainen hallittava infonäyttöratkaisu, jota esitetään mainosnäytöllä. Ohjelma tehdään jo valmiina olevan ravintola-menuohjelman pohjalta, joka on jo käytössä mediasoittimella. Ohjelman uusi versio toimii mediasoittimen sijaan mainosnäytöllä.

Uusi ohjelma ohjelmoidaan Javascriptin React.js –sovelluskehystä (framework) käyttäen. Ohjelma on samankaltainen kuin mikä aiemmin oli käytössä, mutta yksinkertaisempi eli riisuttu versio, jossa on vähemmän toimintoja. Valmista ohjelmaa esitetään LG:n webOS Signageen tarkoitetussa näytössä. Koska näyttöä ei ole saatavilla, sama lopputulos saavutetaan käyttämällä sen sijaan LG:n digital signage -näytön emulaattoria.

Työn teoriaosuudessa selvennetään työn opinnäytetyön aikana käytettyjä teknologioita, ja avataan siinä vastaan tulleita asioita. Empiirisessä osassa kuvataan työn eteneminen alusta loppuun asti, eli LG:n sivustoihin tutustumisesta emulaattorilla toimivaan ohjelmaan asti.

Lopputuloksena syntyy ravintolan menuohjelma, jossa näkyy kuva ravintolan menusta. Navigointipalkista uuden kielen valitessa kuva vaihtuu uuteen kuvaan, joka vastaa valittua kieltä.

Asiasanat
Medianäyttö, digital signage, React.js, webOS

Sisällys

1 Johdanto

Tämän opinnäytetyön tuloksena syntyi Craneworks Oy Ltd:lle hallittava React.js:ää käyttäen tehty Javascript-pohjainen ravintolan menu-ohjelma. Ohjelma pyörii LG:n webOS Signageen tarkoitetulla emulaattorilla. Menu-ohjelma tehdään Craneworksilla jo olemassa olevan ohjelman pohjalta, joten sen back-end on siten olemassa eikä siihen tarvitse koskea. Erona edelliseen ohjelmaan on se, että edellinen ohjelma vaatii toimiakseen näytön ja erillisen mediasoittimen, kun uudempi versio taas toimii mainosäytöllä, joka ei tarvitse erillistä tietokonetta SoC-sirunsa ansiosta. Opinnäytetyönä valmistunut uusi versio ei tule korvaamaan vanhaa, mutta sitä voi muokata käytettäväksi muissa samankaltaisissa tilanteissa.
Valmiilla menu-ohjelmalla on valmiudet toimia LG:n webOS Signage-näytöillä. Kuitenkin fyysisen LG:n näytön puutteessa samaa asiaa ajaa projektin aikana webOS Signage-emulaattori.

1.1 Sanastoa

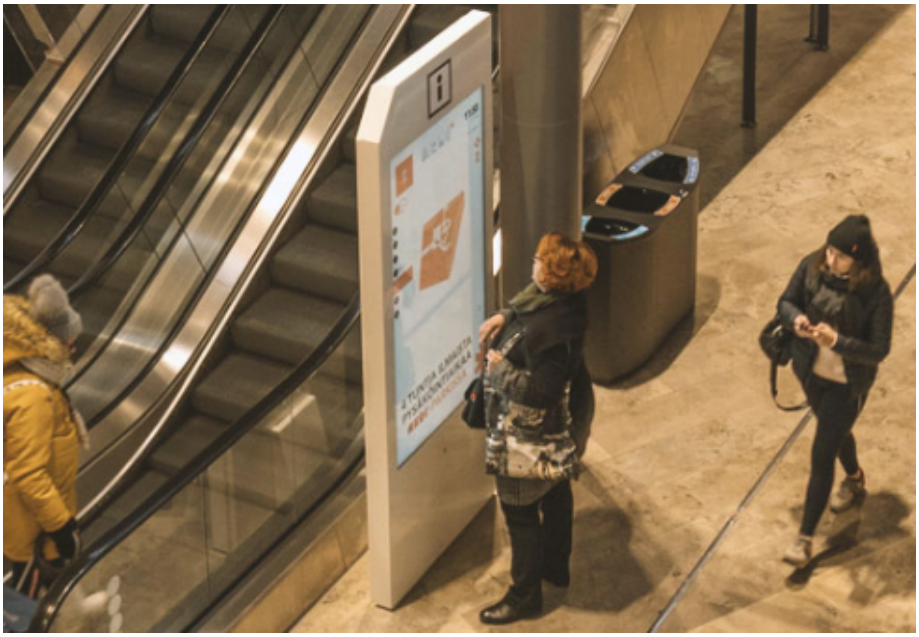
Digital signage	Julkisissa tiloissa näytettäviä digitaalisia mainoksia tai opasteita.
Mediasoitin/ Mediaplayer	Pieni tietokone, joka medianäytön yhteydessä toistaa sille määrättyä materiaalia.
Medianäyttö	Näyttö jossa esitetään mainoksia tai infoa.
Toteemi	Mediasoitin, medianäyttö ja niille tarkoitettu runko muodostavat yhdessä toteemin.
Soc-sirullinen näyttö	System-on-chip, järjestelmä sirulla. Pienoiskokoon pakattu tietokone joka löytyy näytön sisältä.
Wayfinding	Kolmiulotteinen karttasovellus joita löytää julkisista paikoista.
Front-end	Ohjelman näkyvä osa, eli se jonka voi nähdä esim. Verkkoselaimessa kun katsoo sivuja.
Back-end	Ohjelman osa, jossa data sijaitsee ja josta sitä voi mahdollisesti muokata.
Template	Mainos tai infopohja jonka sisältöä voidaan vaihtaa vastaamaan asiakkaan toiveita.

React.js	React.js on Javascript-kirjasto jonka Facebook on luonut.
Javascript	Javascript on ohjelmointikieli.
Build/ **npm build**	Projektin muodostaminen loppukäyttäjälle jaettavaan muotoon.
Code splitting	Koodin jakautuminen eri tiedostoihin build-vaiheessa.
Minify/ **Minifointi**	Kun tiedostosta poistetaan kaikki turha kuten välit ja kommentit. Koodin pienentäminen.

1.2 Craneworks yrityksenä ja sen palvelut

Craneworks Oy Ltd on vuonna 2004 perustettu täysin suomalaisten omistama yritys, joka toteuttaa asiakkailleen yksilöllisiä Digital Signage ja Wayfinding-ratkaisuja.

Craneworks suunnittelee ja valmistaa ohjelmistoja, mutta asiakkaille voidaan toimittaa myös näyttöjä tai Wayfinding-toteemeja ilman Craneworksin tekemää sisältöä. Sekä Digital Signage-näyttöjä, että Wayfinding toteemeja käytetään mm. kauppakeskuksissa ja laivoissa. Toteemit ovat suurikokoisia infonäyttöjä kuten kuvasta 1 voi nähdä.



Kuva 1. Wayfinding toteemi. (Digital Signage, Craneworks)

Wayfindingilla viitataan jo nimensäkin perusteella karttaopastusjärjestelmään. Isoissa tiloissa joissa navigoimiseen tarvitsee apua, saattaa löytyä Wayfinding toteemi. Craneworksin Wayfinding-toteemeja löytyy toistaiseksi kauppakeskuksista kuten Kamppi ja Itis, sekä

Silja Linen laivoilta. Wayfinding kuuluu Craneworksin BCN Dynamic Wayfinding järjestelmään. Toteemi on jalustaan kiinnitetty, itsenäisesti seisova info-näyttö joka sisältää kolmiulotteisen karttaohjelman. Toteemi näyttää sijainnin kartalla ja sillä voi hakea nopeimmat reitit olinpaikasta kauppakeskuksen sisällä oleviin liikkeisiin.

Digital signagella tarkoitetaan näkyville paikoille ja asiakkaiden kulkureiteille asetettuja erikokoisia näyttöjä eli ruutuja, joissa esitetään mainoksia. Digital Signage ja Wayfinding voivat toimia myös yhdessä siten, että Wayfinding toteemilla esitetään mainoksia niin kauan, kunnes asiakas koskee ruutua, jolloin esille tulee kartta. Digital signagea voidaan käyttää sekä sisätiloissa että ulkona. Näytöt voivat olla kooltaan hyvinkin isoja, ja ne luovat myös aina jonkin verran valoakin, esimerkiksi ulkona talven pimeydessä. Silloin ne myös kiinnittävät enemmän huomiota.

1.3 Opinnäytetyön aiheen valitseminen

Opinnäytetyön aiheen valitsin siten, että lähestyin itse Craneworksia, sillä minua kiinnosti olisiko heillä tarjota opinnäytetyöhön aiheita. Olin viettänyt puolet työharjoittelustani Craneworksilla, ja tutustunut siellä ollessani paremmin ohjelmointikielinä Javascriptiin ja HTML:ään, joten projektin aloittaminen tuntui mielenkiintoiselta. Opinnäytetyöaiheen Craneworksilta löytyminen oli mielestäni muutenkin hyvä idea, koska olin jo siellä työharjoittelussa ja oman osaamisen taso oli helpompi ottaa huomioon aihetta valitessa. Erilaisia vaihtoehtoja oli alunperin noin kolme, ja lopulta opinnäytetyön aiheeksi valittiin vanhan menu-ohjelman pohjalta tehtävä uusi menu-ohjelma joka tulisi mediasoittimen sijaan esitettäväksi LG:n soc-sirullisella webOS näytöllä.

Ollessani työharjoittelussa Craneworksilla, tulin tutustuneeksi vanhaan menu-ohjelmaan, vaikka en silloin suoraan tehnytkään sen parissa töitä. Työharjoitteluni aikana muokkasin React.js:ää käyttäen tehtyjä asiakkaiden pyytämiä templateja. Templatet ovat valmiiksi Craneworksin palvelinpuoleen, eli back-endiin yhteyttä ottava mainospohja joiden sisältö on osittain myös asiakkaan vaihdettavissa back-endin hallintajärjestelmästä.

Opinnäytetyöni aiheen valinta tuli siitä, että olin työharjoitteluni aikana ehtinyt jo tutustua hetken vanhaan menu-ohjelmaan, joten se ei ollut täysin tuntematon. Kyseisestä ohjelmasta tarvittiin uusi yksinkertaisempi ja kevyempi Javascript-pohjainen versio. Uusi versio ei kuitenkaan tulisi vanhan tilalle, vaan se suunniteltiin alusta lähtien tehtäväksi SoC-sirulliselle näytölle, eli näytölle jonka sisällä on pieni tietokone sirun muodossa, ja sen sisältö olisi myöhemmin muutettavissa. Se kuitenkin ottaisi yhteyttä samaan back-endiin, tosin tässä tapauksessa back-endin kopioon, kuin alkuperäinenkin menu-ohjelma, joten siten

ne toimivat samalla tavalla. Valmis ohjelma on siis SoC-sirullisella näytöllä esitettävä Javascript-pohjainen ohjelma, jonka back-end on jo olemassa. Ohjelma näyttää ravintolan ruokalistan, ja sen sisältöä pystyy vaihtamaan navigointipalkista kieltä vaihtaen.

Käytin opinnäytetyön päätyökaluna React.js:ää, sillä olin kiinnostunut tekemään sitä käyttäen kokonaisen projektin, mikä oli asia mitä en ennen ollut tehnyt. Työharjoitteluni aikana ja jo aikaisemmin koulussa olin päässyt tutustumaan Reactiin jonkin verran mutta kokonaista projektia en tähän mennessä ollut vielä koskaan tehnyt. Vanhan menu-ohjelman nähneenä minua myös kiinnosti se, että pystyisinkö tekemään uuden projektin paljon yksinkertaisemmin ja samalla selventämään koodia.

1.4 Tavoitteet ja rajaukset

Tämän opinnäytetyön tarkoituksena oli tehdä uusi yksinkertaisempi versio jo valmiina olevasta ohjelmasta. Opinnäytetyön tuloksena syntyi Javascript-pohjainen menu-ohjelma, joka toimii SoC-sirullisella mainosnäytöllä. Vanhempi versio on myös Javascript-pohjainen ja tehty React.js:ää käyttäen mutta uudesta versiosta olisi silti eri tavalla hyötyä, eikä sitä ole tarkoitettu tulemaan vanhan tilalle.

Craneworksille tästä opinnäytetyöstä on hyötyä siksi, että SoC-sirullisella näytöllä toimiva ratkaisu poistaisi tarpeen erilliseltä mediasoittimelta, eli tietokoneelta, sillä SoC-siru periaatteessa on se tietokone-osa, mutta siruun tiivistettynä ja sijaitsee näytön sisällä. Mediasoittimella tarkoitetaan tässä tapauksessa pienikokoista laatikonmallista tietokonetta joka toistaa sille määrättyä materiaalia, ja näyttöön yhdistettynä muodostaa mainosnäytön. Kun ohjelma pyörii SoC-sirulla, se tarvitsee vain mainostaulun tai näytön. Näytön voi sijoittaa paljon vapaammin eri paikkoihin, kun ei tarvitse huolehtia myös tietokonelaitteesta. Esimerkiksi Wayfinding-toteemi joita käytetään kauppakeskuksissa näyttämään mainoksia ja karttoja, koostu mediasoittimesta, kosketusnäytöstä ja rungosta joka sitoo sen kaiken yhteen, vaatii aina tietynlaisen tilan ja sen asentaminen on aikaa vievää. Pelkän näytön asentaminen kokonaisen toteemin sijaan olisi nopeampaa ja saattaisi myös tulla edullisemmaksi. Tulevaisuudessa joissain tapauksissa mediasoitinratkaisuissa siirrytään SoC-sirullisiin ratkaisuihin sen sijaan että käytettäisiin medianäyttöä ja mediasoitinta jos pelkkä näyttö riittää.

Ratkaisulla, eli menu-ohjelmalla on jo olemassa oleva back-end joka sisältää hallintasivun, jota mm. mahdolliset asiakkaat käyttäisivät muokatakseen sovelluksen sisältöä. Ohjelma on ulkonäöltään hyvin samankaltainen edeltäjänsä kanssa.

Projektin tarkoituksena on siis tehdä front-end Javascript-pohjaiselle mainosnäyttöratkaisulle joka ei tarvitse erillistä tietokonelaitetta. Projektissa riittää, että siinä saa näkyville yhden ravintolan ruokalistan monen ruokalistan sijaan. Koska ohjelmasssa on jo olemassa back-end, sitä ei tarvitse tehdä uudestaan. Samaa backendia käytetään monissa muissakin projekteissa. Projektin back-endia voidaan siis käyttää muokkaamatta, ja sillä pystytään tuottamaan dataa eri tarkoituksiin.

1.5 Tietoperusta

Opinnäytetyön tekemisessä käytettiin apuna samassa yrityksessä työharjoittelun aikana saatua Wesbos React.js videokurssin materiaaleja, työharjoittelun aikana valmistettuja templateja, sekä edellistä menu-ohjelmaa ulkonäön yhtenäisenä pitämiseen. Kirjallisia lähteitä ei käytetty juuri ollenkaan, mutta internetistä löytyvää React-dokumentaatiota, React-tutoriaaleja sekä artikkeleita että foorumeita hyödynnettiin uuden oppimiseen, asioiden varmistamiseen ja työn etenemiseen.

Opinnäytetyönä valmistunut menu-ohjelma toteutettiin React.js:ää, Javascriptia, CSS:ää ja HTML:ää käyttäen. Myös edellinen menu-ohjelma oli toteutettu sitä käyttäen, kuten myös templateissa jota olin muokkaillut työharjoitteluni aikana. Saman kielen valitsemiseen vaikutti mielenkiintoni oppia käyttämään Reactia paremmin, sillä olin tullut sen kanssa aikaisemminkin tekemisiin. Ajattelin myös, että edellisestä menu-ohjelmasta olisi helpompi saada ote kun molemmissa käytettäisiin Reactia.

Työssäoppimisen aikana kerätyn kokemuksen ansiosta pystyin tekemään ohjelman käyttäjäliittymän (ulkonäön) sellaiseksi mitä haettiin, eli edellisen ohjelman näköiseksi. En kuitenkaan ollut tätä ennen tehnyt kokonaista omaa projektiani alusta loppuun, joten kun tielleni tuli hankalia kohtia, sain apua ja ehdotuksia siitä, miten kannattaisi edetä. Tukea toimeksiantajalta opinnäytetyön tekemisessä sain Craneworksin puolesta vastuu-ohjaajaltani aina kun sitä pyysin.

Opinnäytetyönä valmistuva ohjelma tehtiin Craneworksille, ja sen sisällön päivittäminen ja ohjelman ylläpito jää heille. Mediasoitinratkaisu jää Craneworksin BitBucketiin eli Githubin kaltaiseen versionhallintajärjestelmään johon tallennetaan projekteja. Kun LG:n näyttö on saatavilla, menu-ohjelma laitetaan Craneworksin aulaan demonstraationa siitä, minkälaisia digital signage ratkaisuja Craneworks tarjoaa.

2 Teoriaosuus

Menu-ohjelman kirjoittamisessa käytettiin React.js:ää, HTML:ää, ja kehitysympäristönä (IDE, Integrated Development Environment) toimi WebStorm mistä lopullinen tuote siirrettiin webOS:n omaan kehitysympäristöön. WebOS:n kehitysympäristössä ohjelma on tarkoitus saada esitettäväksi SoC-sirulliselle näytölle.

Opinnäytetyön aiheena on mediasoitinratkaisu joka toimisi LG:n näytöllä, sen sisällä olevalla SOC (System-on-Chip)-sirulla. Valmis ohjelma toimisi siis itsenäisesti näytöllä. Miksi päädyimme tähän ratkaisuun? Edellinen menuohjelma toimi kuitenkin mediasoittimella. Alusta asti uusi menuohjelma oli tarkoitus tehdä näytölle, mutta miksi?

2.1 Mikä on mediasoitin?

Tässä yhteydessä mediasoitin on pieni tietokone, jolla voidaan toistaa sille määrättyä materiaalia. Tietokoneeseen yhdistetään kaapelilla näyttö, jotta itse materiaali saadaan näkyville. Jos mediasoitinjärjestelmän tarvitsee toimiakseen verkkoyhteyden, se pitää yhdistää internetiin samoin tavoin kuin tavallinenkin tietokone, eli joko langattomasti tai fyysisesti ethernet-kaapelilla. Mediasoittimella on yleensä oma käyttöliittymänsä, jonka avulla sille määrättyä sisältöä toistetaan. Jos tuotettua sisältöä varastoidaan, sen voi tehdä palvelimelle, pilvipalveluun tai kiintolevylle. (Eteläaho, 2016)

Mediasoittimia on erilaisia mutta tässä yhteydessä puhun digital signagessa käytettävistä mediasoittimista, jotka näyttöön yhdistettyinä muodostavat kokonaisuuden, joka tunnetaan info- tai mainosnäyttönä. Digital signage-mainosnäytöt ovat erikokoisia, yleensä ostoskeskuksista ja kauppojen lähettyviltä löytyviä näyttöjä joissa esitetään digitaalisia mainoksia.

Digitalisaation myötä ja mainontaan vaaditun laitteiston muututtua halvemmaksi on voitu siirtyä vanhoista mainosjulisteista ja banderolleista mediasoittimiin ja mainosnäyttöihin. Ne ajavat samaa asiaa kuin julisteetkin, mutta sama asia hoituu niillä tehokkaammin hallintamahdollisuuden ansiosta. (Hirvonen, 2015)

Ohjelma jonka pohjalta teen oman opinnäytetyöni, on infonäyttöratkaisu. Se tarkoittaa sitä, että edellinen ohjelma, eli menuohjelma toimii tällä hetkellä mediasoittimella, joka on kytketty kiinni kosketusnäyttöiseen medianäyttöön. Näyttöjä joissa menuohjelma toimii, on tällä hetkellä asennettuna Tallink Silja Linen ravintoloiden viereen näyttämään ravintoloiden ruokalistoja.

2.1.1 Digital Signage Craneworksilla

Teknologia kehittyy jatkuvasti ja digitaaliset mainokset ovat nykyaikaa. Julkisissa tiloissa löytyy nykyään usein myös digital signage ratkaisuja, liittyivät ne sitten mainontaan tai informaation jakamiseen, usein molempiin.

Craneworksille digital signage merkitsee monta asiaa. Sillä edistetään myyntiä, markkinointia ja viestintää. Sitä voidaan myös käyttää viihdyttämiseen, tiedon visualisoimiseen, brändin imagon vahvistamiseen ja tunnelman luontiin. Digital signage on kirkkaita etäohjattavia näyttöpintoja jotka ovat hyvin tärkeä väline vahvassa vaikuttamisessa. Se on tehokasta massamediaa, joka toimii tehokkaimmin paikoissa missä on paljon ihmisiä, kuten ulkotilat, ostoskeskukset, risteilyalukset, koulukampukset ja sairaalat. Näyttöjen on tarkoitus kiinnittää huomiota, ja päivitettävällä sisällöllä voidaan ohjata asiakkaita tarjouksien perässä erilaisiin kauppoihin. Kaikkia näyttöjä ei tietenkään käytetä mainostarkoitukseen, vaan esimerkiksi Craneworksin Wayfinding toimii opastuskarttana.
(Craneworks, 2019)

Suurin osa Craneworksin Digital Signage ratkaisuista toimii mediasoittimella. Käytössä saattaa olla toteemi, joka koostuu mediasoittimesta, kosketusnäytöstä ja rungosta. Opinnäytetyöni perustana toiminut menuohjelma-ratkaisu käyttää kosketusnäyttöä joka on kytketty mediasoittimeen joka toistaa materiaalia. Toteemeille ominainen runko-osa puuttuu siitä silloin kokonaan, sillä se on kiinni seinässä.

2.1.2 Mainosnäytön etu mediasoittimen verrattuna

Oma opinnäytetyötäni ei tehty mediasoittimelle vaan mainosnäytölle, missä se mediasoittimen sijasta olisi omalla SoC-sirullaan joka on integroitu, eli yhdistetty itse näyttöön. Miksi tähän ratkaisuun päädyttiin?

Soc, eli system-on-chip tarkoittaa yksinkertaisesti järjestelmää sirulla. Se on kuin pienoiskokoon pakattu tietokone, mikä mahdollistaa mm. sen, että kun koko järjestelmä on sirulla, elektronisten laitteiden koko saadaan pienemmäksi. Elektroniikan pienentyneen koon voi huomata parhaiten älypuhelimissa, tableteissa ja kannettavissa tietokoneissa jotka eivät ole enää läheskään yhtä suurikokoisia ja painavia verrattuna niiden alkuaikoihin. (Tietokonemaailma, 2019)

Mitä hyötyä SoC-sirullisesta näytöstä siis olisi? SoC-sirullisissa mainosnäytöissä on sekä hyviä että huonoja puolia verrattuina medianäyttöihin joiden materiaali pyörii mediasoittimella.

Soc-siru ajaa saman asian mitä mediasoitin ajaa. Tämä eliminoi kokonaan tarpeen erilliseltä tietokone-osalta eli mediasoittimelta. Tämä puolestaan johtaa jo yhteen tärkeimmistä hyödyistä, eli siihen että SoC-sirullisen näytön ansiosta saadaan paljon enemmän vapauksia siihen, minne se voidaan asentaa. Jo pelkän mediasoittimen puuttuminen ratkaisusta mahdollistaa sen, että näyttö vie vähemmän tilaa jo sen takia että mediasoitinta ei tarvitse piilottaa minnekään. Tämä tekee näytöstä nopeamman asentaa, ja samalla myös kustannustehokkamman.

2.2 React.js

React. on Facebookin kehittämä Javascript-kirjasto, joka auttaa käyttöliittymien luomisessa. Reactilla voidaan tehdä yhden sivun applikaatioita ja mobiiliaplikaatioita. Yhden sivun applikaatiolla tarkoitetaan esimerkiksi applikaatiota, joka ei lataa uutta sivua vaan muuttaa vain jo olemassa olevan sivun sisällön.

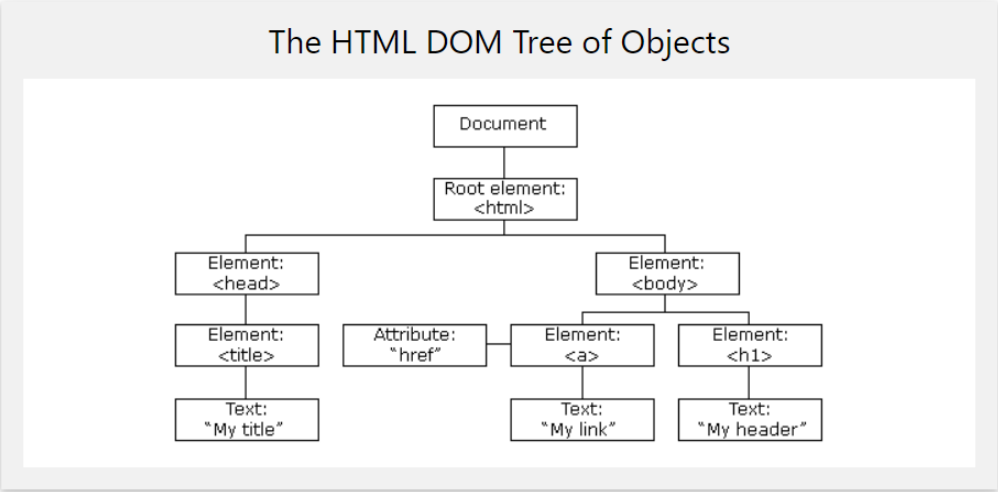
Vuonna 2018 julkaistussa artikkelissaan, ”What is React and why should we use it?” Nitin Pandit kuvailee Reactin mahdollistavan suurien web applikaatioiden datan vaihtamisen ilman että sivua on ladattava uudestaan. Hän kuvailee Reactia sanoin: nopea, skaalattava ja yksinkertainen.

2.3 React.js:n hyödyt

Kieliä on monenlaisia, mutta React.js:llä on omat etunsa, mikä myös vaikutti siihen, miksi se valittiin tämän opinnäytetyön kieleksi.

2.3.1 Virtual Dom

React.js on tunnetusti nopea. Nopeus johtuu osittain ainakin siitä, miten se päivittää dokumenttioliomallinsa, eli DOM:n (Document Object Model). Honkasen “ReactJS”-nimisessä opinnäytetyössä selitetään DOM olevan ohjelmointirajapinta HTML-dokumenteilla, joten HTML-tiedosto joudutaan ensin jäsentelemään DOM-malliksi ennen kuin sen voi renderöidä eli saada näkyviin selaimessa. Kuvasta 2 pystyy hahmottamaan paremmin miltä DOM näyttää.



Kuva 2. The HTML DOM tree of Objects (DOM, w3schools)

Honkanen myös kertoo, että kun Reactilla tehdyn sovelluksen data muuttuu, sen sijaan että koko käyttöliittymä päivitettäisiin, vain muuttuneet komponentit päivitetään mikä tekee Reactista hyvin nopean. Tämän tekee mahdolliseksi se, että React tekee DOM-mallista virtuaalisen kopion ja pitää sen muistissa. Muutosten tapahduttua Reactin on helppo tehdä muutokset oikeaan DOM-malliin. Näin koko DOM-mallia ei tarvitse päivittää ja selaimen renderöinti sujuu nopeammin. (Honkanen, 2017)

2.3.2 Komponentit

Reactin hyviin puoliin kuuluu myös se että esim. käyttöliittymän osat jaetaan eri komponenteiksi, mikä mahdollistaa sen, että niitä voi helposti käyttää useamman kerran, ja se helpottaa koodin siistinä pitämistä.

Koska React on komponenttipohjainen, joten se koostuu monesta eri komponentista. Komponentit ovat uudelleenkäytettäviä, mikä tarkoittaa sitä, kun sellainen on tehty, sen voi kutsua käytettäväksi HTML-tagia muistuttavalla tavalla, esim: <MyComponent />. (Negi, 2017)

Komponentti on periaatteessa Javascript luokka tai funktio, joka palauttaa ns. tietyn osan kokonaisuudesta. Esimerkiksi omassa opinnäytetyössäni mm. menu sisällöstä on tehty oma komponenttinsa. Tämä komponentti on luokallinen, sillä se sisältää muuttuvaa dataa, joka on erikseen noudettava back-endista. Koska kaiken voi tehdä komponenteiksi, ne pitää vain lopuksi kerätä yhteen toimivaksi kokonaisuudeksi, josta on helppo nähdä mistä osista ohjelma koostuu.

React.js:llä on monia hyötyjä, joista osa myös vaikuttaa siihen miksi valitsin sen ohjelmointikielekseni. Tapa millä React.js päivittää vain muuttuneet komponentit nopeuttaa toimintaa ja on hyödyllinen kun kyseessä on ohjelma, jossa muuttuu vain yksi alue kun navigointipalkin painikkeita painetaan. Myös edellinen ohjelma oli tehty React.js:ää käyttäen, ja työharjoitteluni aikana olin myös opiskellut ja käyttänyt pelkästään Reactia. Kielen valinta ei siis ollut kovin vaikea päätös, mutta se ei myöskään ollut itsestäänselvyys sitä päätettäessä. Sovelluskehykseksi harkittiin myös muita Javascript –sovelluskehyksiä kuten Vue.js.

2.3.3 React-kehittäjätyökalut

Selaimista löytyy kehittäjätyökalut (developer tools), Google Chromesta “more tools” painikkeen alta. Kehittäjätyökalujen avulla on helpompi hahmottaa mistä elementeistä tarkasteltava sivu koostuu, ja sillä pystyy myös lisäämään ja muuttamaan omalla näytöllään valitsemansa sivun tyyliä. Reactia käyttäessä on mahdollisuus ladata ainakin Google Chrome selaimeen lisäosa “React Developer Tools”. Se lisää uuden osan normaaleihin kehittäjätyökaluihin joka aukeaa kohdasta “React”. Valitsemalla tämän kohdan, voi nähdä React-sovelluksen hierarkiaa, sekä selvittää komponenttien tilan ja propsit, eli datan mitä niillä milläkin hetkellä on. (Top 10 Advantage of Using React, 2017)

Omassa opinnäytetyössäni React kehittäjätyökalujen käyttäminen tuli ensimmäistä kertaa vastaan silloin, kun halusin nähdä polun millä data siirtyi vanhassa menu-ohjelmassa. Tarvitsin tietoa siihen, että voisin käyttää samaa polkua omassa menu-ohjelmassani. Koska Reactin developer toolsista näkyi mitä propsit (React-komponenttien mukana siirtyvä data) sisälsivät, datan jäljittäminen ja oikean polun löytäminen helpottui huomattavasti. Liitteessä 5 näkyy polkuja joita etsin.

2.4 WebOS Signage

WebOS Signage on tällä hetkellä LG:n omistama alusta, joka on tarkoitettu mm. digitaalisessa mainonnassa käytettäville näytöille, ja tarkemmin ottaen SoC-siruilla toimiville LG:n näytöille. LG:n näytöille ominaista on sisällönhallintajärjestelmä (Content Management System) joka antaa käyttäjän toistaa, editoida ja aikatauluttaa digitaalisen soittolistan käyttämättä erikseen tietokonetta sitä varten, vaan se onnistuisi näytön omalla kaukosäätimellä. WebOS on web-keskeinen avoimeen lähdekoodiin perustuva alusta, ja koska se on järjestelmäriippumaton, sillä voi tehdä projekteja joita voidaan toistaa monella eri koneella kuten digital signage-näytöt, Windows-koneet, Android-tabletit tai puhelimet, sekä Applen tuotteet. WebOS näytöissä on sisäänrakennettu mediasoitin, eli SoC-siru ja ne tarvitsevat vain internet yhteyden toimiakseen. (LG, 2019)

Ne ovat siis valmiita medianäyttöjä, jotka tarvitsevat vain sisällön. Craneworksilla oli tiedossa heti alusta alkaen, että uusi menu-ohjelma tehtäisiin LG:n näyttöä varten.

2.4.1 WebOS kehitysympäristö ja näytön emulaattori

Kun ensimmäinen oma webOS Signage-applikaatio aloitetaan, tutustutaan webOS:n omaan kehitysympäristöön, webOS IDE:n. Kehitysympäristön saa ladattua LG:n webOS Signage developer-sivuilta, mistä myös löytyy ohje applikaation tekemiseen. Ohjeessa kerrotaan mitä kaikkea tarvitaan uuden ohjelman tekemiseen, eli webOS Signage IDE (kehitysympäristö), webOS Signage Emulator (v4.0.6). Käytimme emulaattoria sillä näyttöä ei ollut saatavilla. Näiden lisäksi ladattiin vielä webOS Signage CLI, joka on webOS terminaali.

WebOS:n kehitysympäristö on siitä kätevä, että se olettaa, että emulaattori otetaan käyttöön. Siksi emulaattorille löytyy oma kuvakkeensa, ja yhteyden ottaminen siihen on tehtävissä helposti oikea-klikkaamalla emulaattorin kohdalta ja valitsemalla heti ensimmäinen vaihtoehto joka on “Connect”. Yhteys kuitenkin syntyy vain, jos emulaattori on myös klikattu erikseen päälle. Näin webOS kehitysympäristössä tehtyä ohjelmaa voi kokeilla “näytölle”, eli emulaattorille.

2.5 Käyttöliittymä

Käyttöliittymä (UI, User Interface) on asiakkaalle näkyvä näkymä. Kun käyttöliittymiä suunnitellaan, on yleensä tavoitteena se että käyttöliittymä on yksinkertainen ja nopea käyttää. FixUIn artikkelin “Mikä Ihmeen käyttöliittymäsuunnittelu?” -nimisen artikkelin mukaan käyttöliittymän suunnittelussa pitää huomioida muutama asia. Siinä on otettava huomioon käyttäjien lähtökohdat, huomion kiinnittäminen malleihin ja asioiden yksinkertaisena pitäminen.

> *“Hyvällä käyttöliittymäsuunnittelulla pyritään siihen, että käyttäjä pystyy mahdollisimman tehokkaasti saavuttamaan tuotteen tai palvelun loppupäämäärän.” (FixUI, 2018)*

Koska opinnäytetyöni oli tarkoitus näyttää samanlaiselta kuin sitä edeltänyt sovellus, on siis tulkittavissa, että se on todettu sen verran hyväksi, ettei sitä tarvitse muuttaa. Käyttöliittymä on yksinkertainen ja väreiltään sinivihreä. Värit ovat suoraan asiakkaan brändivärеistä valittu, joten niistä voidaan päätellä se että sinistä on haluttu käyttää sen takia että se antaa mm. luotettavan ja rauhallisen vaikutelman. (Felix, 2019).

Käyttöliittymän yksinkertaisuus ja mallit joihin ihmiset ovat tottuneet (UI pattern) kävelevät

myös käsi kädessä. Mallilla tarkoitetaan tässä yhteydessä tapoja joihin ihmiset ovat tottuneet ja hämmentyvät jos totuttu asia ei käy toteen.

Menuohjelmassa on navigaatiopalkki oikealla, mikä johtuu mielestäni siitä, että suurin osa ihmisistä on oikeakätisiä. Täten on luonnollista käyttää oikeaa kättään, kun valitsee kielen millä haluaa ruokalistansa nähdä. Kun kieli-nappia on painettu, on myös totuttu siihen, että nappi muuttuisi jollain tavalla, yleensä väriltään tummemmaksi. Se auttaa jo parissa asiassa. Siinä että tietää millä sivulla on, eli mitä on painanut, ja siinä että näkee ohjelman toimivan. Jos napille ei painettaessa tapahdu mitään, helposti ihmetellään, menikö komento perille? Tekikö ohjelma mitään? Varsinkin kun ainut asia mikä vaihtuu on menun teksti.

Sivuanimaatiot ovat myös asia jotka tekevät käyttöliittymäkokemuksesta miellyttävän. Tässä tapauksessa, kun kieli-nappia painaa, uusi menu ei avaudu salamannopeasti, vaan tapahtuu häivytysefekti (fade-in transition). Tällöin kun painiketta on jo painanut, saa samantien tiedon siitä että jotain tapahtuu. Vaikka sivun päivittymisen pitäisi olla toivottua, liian nopealla transitiolla on tässä tilanteessa päinvastainen vaikutus. Jos transitio tapahtuu liian nopeasti, sitä ei välttämättä huomaa heti.

2.6 Tiedonvälitys

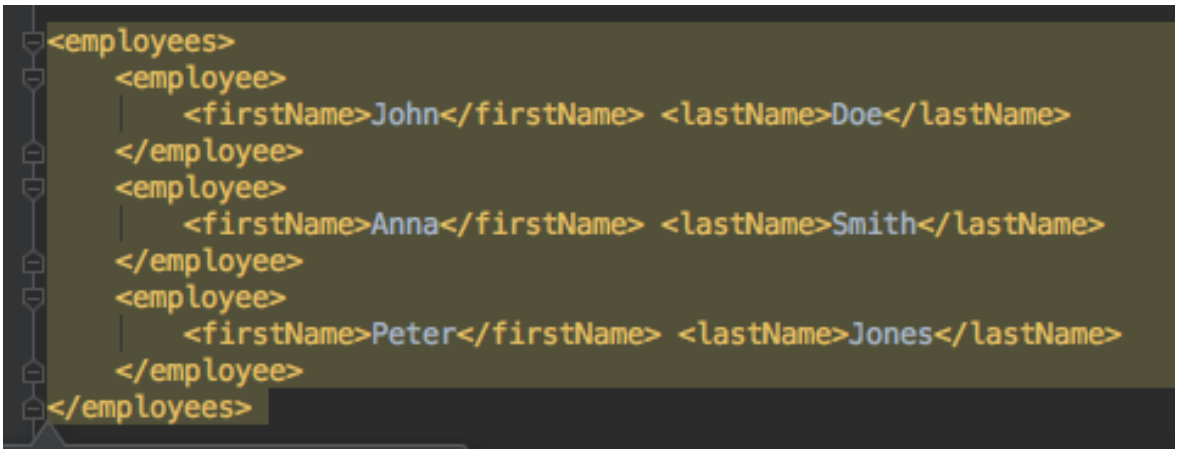
Opinnäytetyön aikana kohtasin kaksi eri tapaa välittää tietoa. JSON (Javascript Object Notation) ja XML (eXtensible Markup Language).
JSON (JavaScript Object Notation) on yksinkertainen tiedostomuoto, jota käytetään tiedonvälityksessä. Se sisältää taulukkomuodossa dataa jotka on eroteltu pilkuilla. Koska JSON on syntaksiltaan samanlainen kuin Javascript-objektit, Javascript-ohjelma pystyy muuttamaan saamansa JSON datan Javascript-objekteiksi. (W3schools) Kuva 3 havainnollistaa miltä JSON kirjoitettuna näyttää.

JSON esimerkki:

```
{"employees":[
 { "firstName":"John", "lastName":"Doe" },
 { "firstName":"Anna", "lastName":"Smith" },
 { "firstName":"Peter", "lastName":"Jones" }
]}
```

Kuva 3. Malli JSON:sta (mukaillen w3Schools, Introduction to JSON)
Myös XML (eXtensible Markup Language) käytetään kun halutaan hakea dataa web-palvelimelta. Oikeastaan JSON ja XML on tehty suorittamaan samoja asioita, mutta JSON:ia

pidetään parempana. Verrattuna JSONiin, XML on hitaampi ja se sisältää enemmän tekstiä. Saman asian pystyy ilmaisemaan vähemmällä JSON-muodossa. (Wyse, 2014) JSON:in ja XML:n eron pystyy havainnollistamaan parhaiten kuvasta 3 ja 4.



Kuva 4. Malli XML:stä (mukaillen w3Schools, JSON vs XML)

Suurimpia eroja kahden eri tavan välillä on se, että JSON on lyhyempi, sitä on helpompi ja nopeampi kirjoittaa ja siinä voi käyttää taulukko-listoja (array-listoja). XML pitää jäsentää (parse) XML-jäsentäjällä kun JSON:ille taas riittää sitä varten vain Javascript-funktio.

Opinnäytetyössäni yksi tärkeimmistä ideoista on se, että siinä käytetään jo valmiina olevaa back-endiä. Data, jota siihen käytetään, on haettavissa XML-tiedostosta, jota puolestaan päivitetään ohjelman hallintajärjestelmän kautta. Käytössämme on siis XML-URL joka sisältää dataa kuten käytössä olevat kielet, sekä menuohjelman muun sisällön, eli ruokalista kuvat. Kuvia voidaan päivittää hallintajärjestelmän kautta, jolloin ne myös päivittyvät XML-tiedostoon.

Opinnäytetyö ohjelman tekemisen aikana kävi myös ilmi, että Javascriptin json() –metodia käyttämällä, saadaan lupaus, joka jäsentää xml-datan JSON-muotoon. (MDN web docs. 2019)

3 Menu-ohjelman tekemisen vaiheet

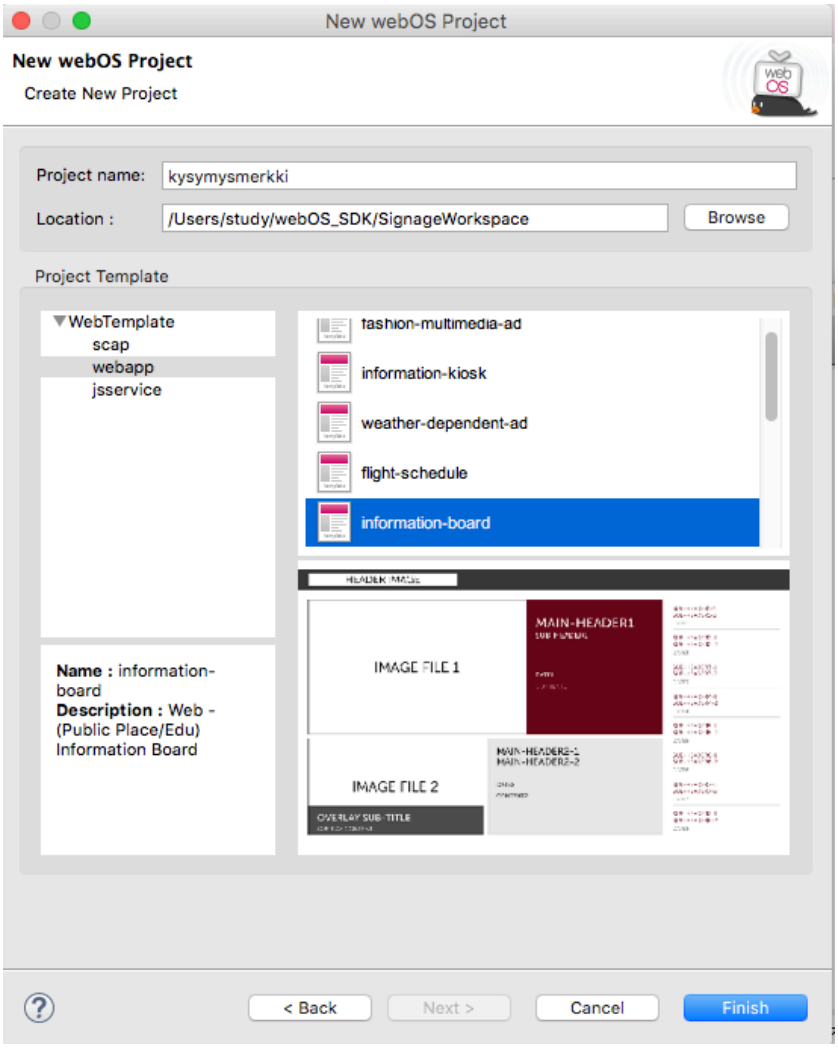
Tämän osan on tarkoitus antaa vaihe vaiheelta kuva menu-ohjelman valmistumisesta, eli siitä miten se tehtiin alusta loppuun saakka.

3.1 LGn WebOS Signage Developer -sivuihin tutustuminen ja tarvittavien osien lataaminen

Ennen kuin luotiin projektia tulevalle infonäyttöratkaisulle, sain tehtäväksi perehtyä LG:n webOS Signage Developer sivustolla julkaistuun dokumentointiin ja selvittää miten päätyä haluttuun lopputulokseen, eli menu-ohjelmaan, jota käytettäisiin LG:n näytöllä.
LG:n signage sivusto on suuri ja moniosainen, joten siellä on helppo eksyä erityyppisten digital signage -ratkaisujen pariin, kuin minkä parissa oli tarkoitus työskennellä.

Lukemalla sivustoa selvisi että applikaation tekemiseen webOS:illa tarvitsisimme ainakin webOS Signage IDE:n ja webOS Signage Emulator (v4.0.6). Vapaan LG näytön puutteessa emulaattori oli hyvä korvike. Emulaattorin idea on toimia valmiin LG:n näytön tavoin. Kun emulaattori-ohjelman avaa sitä voi kontrolloida kaukosäätimellä kuten TV:tä. Samalla ladattiin myös webOS Signage CLI, joka on webOS terminaali.

Aloitin webOS IDE:n tutkimisen sillä, että tein LG:n ohjetta apuna käyttäen uuden projektin. Huomasin heti, että jos halusin tehdä uuden webOS projektin, kuten ohjeessa sanottiin, tyhjää pohjaa ei saanut ollenkaan vaan oli pakko valita joku webOS:n valmiiksi tehdyistä templateista, eli mallikappaleista joihin olisi tarkoitus vain lisätä haluamansa tekstit. Mikään templateista ei näyttänyt lähellekään sellaiselta mitä halusin tehdä, mutta kokeilin niistä kuitenkin muutamaa nähdäkseni miten tällaiset ohjelmat toimivat. Kuvasta 5 näkyy yksi valittavista templateista, mikä havainnollistaa hyvin mitä osia siinä on tarkoitettu muutettavan.



Kuva 5. Uuden projektin aloittaminen webOS IDE:llä (Kinnunen)

Kokeilin muutamien LG:n omien esimerkkiohjelmien käyttämistä ja muokkaamista. Kokeiltuani ensin webOS:ia päätin, että applikaation tekeminen WebStormissa olisi helpompaa sillä silloin pystyisin katsomaan työn etenemistä selaimesta, kun webOSissa ei äkkiseltään löytynyt sellaista mahdollisuutta, ja se tuntui hitaalta verrattuna WebStormiin joka mm. tallensi kaikki muutokset automaattisesti. LG:n sivuilla kuitenkin kerrottiin, että webOS IDE tukee sekä Javascriptia, että HTML:ää kielinä, joten tulin siihen tulokseen, että React.js:llä tehty ohjelma olisi webOS IDE:n tukema. Kysyin tähän myös mielipidettä työkaverilta, ja päädyimme samaan tulokseen.

3.2 Edellisen mediasoitinratkaisun tutkiminen

Edellinen menu-ohjelma jonka perusteella opinnäytetyöni tein on Reactia käyttäen tehty ohjelma, joka toimii mediasoittimella. Menuohjelmassa saman ravintolan sisältä löytyy kahdesta eri ruokavalikoimasta, a la carte ja menu nordic. Kaikki ruokalistat ovat tarjolla

eri kielillä. Kieltä vaihdetaan oikealla olevan navigaatiopalkin kielinappeja painamalla, ja ravintola vaihtoehdot löytyvät myös navigaatiopalkista.

Edellisen menu-ohjelman tutkimiseen meni yllättävän paljon aikaa jo tässä vaiheessa sekä vielä lisää myöhemmin. Ohjelma oli iso, tutkittavaa oli paljon ja ensitöikseni yritin saada sitä edes toimimaan selaimella, missä kestikin yllättävän kauan. Toimimattomuuden syyksi paljastui myöhemmin, että melkein jokaiseen tiedostoon oli tullut muutoksia. Käytössäni oli vanhempi versio, jota olin käyttänyt harjoitteluaikanani, joten lopulta jouduin vetämään koko projektin uudestaan Craneworksin BitBucketista, eli paikasta minne projektit tallennetaan. Tämä uusi versio toimi ilman ongelmia.

Halusin tutkia edellistä applikaatiota koska ajattelin alun perin, että ohjelma minkä itse tekisin, tulisi lopulta olemaan yhtä suuri ja mahdollisesti monimutkainen. Selvitin mitä osia vanhassa ohjelmassa oli ja yritin parhaani mukaan ymmärtää mitä kaikkea se itseasiassa teki. Vanhan applikaation tutkimiseen sain myös apua samassa huoneessa olevilta työntekijöiltä silloin kun en ymmärtänyt jotain. Edellisen menu-ohjelman koodissa ei ollut lainkaan kommentointia, joten sitä oli senkin takia haastava lukea.

3.3 Käyttöliittymän suunnitteleminen

Käyttöliittymä (UI, User Interface) on asiakkaalle näkyvä näkymä. Se oli todettu hyväksi jo edellisen menu-ohjelman kohdalla. Sinisen ja vihreän eri sävyt määräytyivät suoraan niistä väreistä mitä asiakkaan brändin värikokoelmasta on saanut valita. Uusi menu-ohjelma on yhden sivun (single-page) ohjelma, mikä tarkoittaa sitä, että ohjelmassa on yksi sivu. Uusi menu-ohjelma haluttiin tehdä näyttämään samalta kuin edeltäjänsä, joten itselleni ei tältä alueelta jäänyt paljoakaan suunniteltavaa. Päädyin ottamaan vain pieniä vapauksia kuten logon ja päivämäärän aivan hieman suuremmaksi jättämisen, sillä ne näyttivät paremmilta niin.

3.4 Käyttöliittymän tekemisen aloittaminen

Käyttöliittymän tekemisen aloitin tekemällä uuden React app -projektin WebStormiin. Projekti oli tässä vaiheessa vain paikallinen (local), eli se sijaitsi vain omalla koneellani, eikä sille ollut vielä omaa paikkaa Craneworksin BitBucketissa. Ensimmäisen version tein kokonaan neonväreillä HTML:ää käyttäen, saadakseni tuntuman siihen mitä osia tarvitsisin tehdäkseni menu-ohjelman. Suurimmaksi ongelmakseni muodostui tässä vaiheessa yllättäen navigointipalkin kieli-painikkeet joita en osannut toteuttaa. Tässä vaiheessa koko projektini oli App.js-tiedoston sisällä eikä jaettuna komponentteihin.

3.5 Käyttöliittymän tekeminen uudestaan React.js:ää käyttäen

Koska olin jo viettänyt paljon aikaa käyttöliittymän osien suunnittelemisessa mutta halusin tehdä sen tällä kertaa enemmän “reactmaisemmaksi”, aloitin projektin siistimisen ja muuttamisen oikeaan suuntaan. Siistin pois turhia elementtejä mitä en tarvinnut ja muutin navigointipalkin omaksi komponentikseen. Kopioin sinne aikaisemmin app.js tehdyn navigointipalkin ja lisäsin sen App.js:ään komponenttinä. Sisällölle tein myöhemmin oman komponentin, mutta tässä vaiheessa se oli vielä app.js:n sisällä.

3.6 Vaatimusten määrittely

Toiminnallisuutta lähdin selvittämään siitä, miten edellinen menu-ohjelma oli tehty. Vaatimusten määrittely uudelle menu-ohjelmalle tapahtui alusta lähtien siten, että otin opinnäytetyölle annetut rajaukset huomioon. Tiedossani oli se, että lähtisin aivan aluksi vain tekemään käyttöliittymää (UI), joka näyttäisi mahdollisimman paljon edelliseltä menu-ohelmalta. Kaikki perustoiminnallisuudet mitä vanhassakin ohjelmassa oli käyttöliittymää katsellessa, tuli löytyä myös uudesta ohjelmasta. Edellinen projekti kuitenkin sisälsi paljon ylimääräisiä ratkaisuja, joita ei tarvittu uudessa menu-ohjelmassa. Sen takia päätin lähteä liikkeelle ottamatta edellisen projektin koodista niinkään mallia. Vain ne asiat mitkä näin itse vanhaa menu-ohjelmaa käyttäessäni piti luoda uudelleen.

Ennen toiminnallisuuden tekemistä, pidimme vielä palaverin missä varmistettiin mitä kaikkea uuden menu-ohjelman oli tarkoitus tehdä rajaukset huomioon ottaen. Listalle vaatimuksista päätyi lopulta se, että ohjelma näyttää vain yhden ravintolan ruokalistat. Navigointipalkin kielinappia painamalla lista näkyisi eri kielellä. Ohjelmassa valittaviksi kieliksi valittiin suomi, ruotsi ja englanti, sillä ne olivat yleisimmin käytetyt kielet. Ruokalistat olisivat kuvia, jotka asiakas saisi itse laittaa menu-ohjelman hallintajärjestelmän sivulta, eli ohjelman backe-endista. Opinnäytetyöstä oli rajattu pois monelle ravintolalle tehtävä versio, mutta home-painike jätettiin paikoilleen jatkokehitystä ajatellen, ja se sovittiin näyttävän pääkielenä toimivan suomen ruokalistan.

3.7 Ulkonäön valmistuminen

Ensimmäinen versio uudesta menuohjelmasta näytti hyvin samanlaiselta kuin mitä vanhakin versio oli. Saatuani ohjelman ulkonäön samanlaiseksi kuin edellisessä menu-ohjelmassa, seuraava vaihe oli toiminnallisuus. Aloitin navigointipalkin kieli-napeista. Jotta sivu latautuisi uudestaan eri ruokalistalla, React-projektiin piti lisätä reititin.

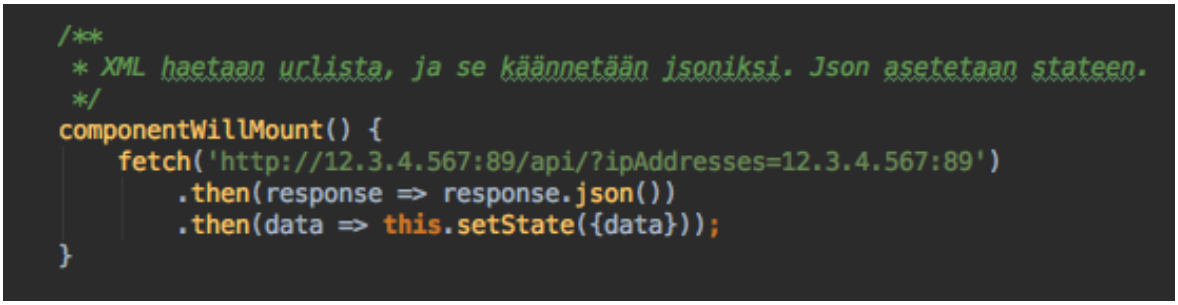
Etsin hyvän tovin vanhasta menu-ohjelmasta tietoa siitä, minkälaiseen reititin-ratkaisuun siinä oli päädytty, mutta sen löydettyäni en ymmärtänyt miten se toimi. Tästä seurasi pari päivää ”Routing in react.js” -hakusanalla googlailemista ja työkaverin kanssa asian yhdessä miettimistä. Kun toimiva ratkaisu löydettiin, laitoin tyylitiedoston, eli CSS:n puolelle kuvia jotka kovakoodasin reitittimeen nähdäkseni latautuisivatko sivut oikein. Kun reititin näytti toimivan kuten pitikin, eli näin sivun vaihtuvan, seuraavaksi tuli kuvien lataaminen ohjelman ruokalista-alueelle menu-ohjelman hallintajärjestelmän kautta. Menu-ohjelman ulkonäön näkee *liitteestä numero 2*.

3.8 XML-URL

Tärkein idea menu-ohjelman takana on se, että sen sisältöä pystyy vaihtamaan ohjelman hallintajärjestelmän kautta. Tärkeäksi tämän teki se, että menu-ohjelmalla oli jo oma backendinsa, mitä sen tuli käyttää, ja hallintajärjestelmä on osa sitä. Hallintajärjestelmä on tehty sen takia, että asiakkaat voivat itse päivittää menu-ohjelman ruokalistoja lataamalla hallintajärjestelmään jpg- tai png-kuvatiedostot ruokalistoista. Nämä tiedot applikaatio sitten hakee ja näyttää. Näin Craneworksin ei siis itse tarvitse päivittäin vaihtaa tietoja, vaan applikaation vaihtuva sisältö, eli tässä tapauksessa menulistat, ovat täysin asiakkaan hallittavissa.

Edellisellä menu-ohjelmalla oli käytössään URL, josta ladataan XML-tiedosto. Se sisälsi ravintolat ja kuvat mitä menu-ohjelmassa käytettiin. Tähän tiedostoon pystyy vaikuttamaan juuri hallintajärjetelmässä, jolloin sen sisältö vaihtuu.

Lähdin liikkeelle sillä oletuksella, että JSON:ia olisi helpompi tulkita, joten joutuisin kääntämään XML-tiedoston ensin JSON-muotoon. Etsin ratkaisua pitkään ja myös työkaverin avustuksella oikeastaan vain internetistä Googlea apuna käyttäen, mutta vaikka onnistuimmekin löytämään monia kysymyksiä asiasta, emme onnistuneet löytämään ratkaisuja, jotka olisivat toimineet. Hakusanat mitä käytin olivat “converting data from xml to json React.js” ja tulokset eivät olleet sellaisia jotka olisivat toimineet. Päädyin kokeilemaan mm. xml2json jäsentäjää, jonka löysin npmj.com-sivustolta, mutta poistin sen lopulta tutkittuani aikaisempaa menu-ohjelmaa, ja tultuani siihen tulokseen, että ainakaan siellä ei oltu käytetty ylimääräisiä npm-ohjelmia. Aivan lopuksi ja osittain jo asian suhteen siltä päivältä luovuttaneina, päätimme työkaverin kanssa googlata vielä ”How to fetch data in React?”. Selasimme sivua ihan vain nähdäksemme, oliko missään haettu URL:lla dataa, ja vastaus löytyikin heti ensimmäisessä linkissä, joka sattui olemaan Robin Rwieruch -nimisen ohjelmistoinsinöörin blogi. Kuvassa 6 näkee, kuinka sivustolta löydettyä vastausta sovellettiin.



Kuva 6. Back-endiin otetaan yhteys fetch-funktiolla. (mukaillen Rwieruch 2018)

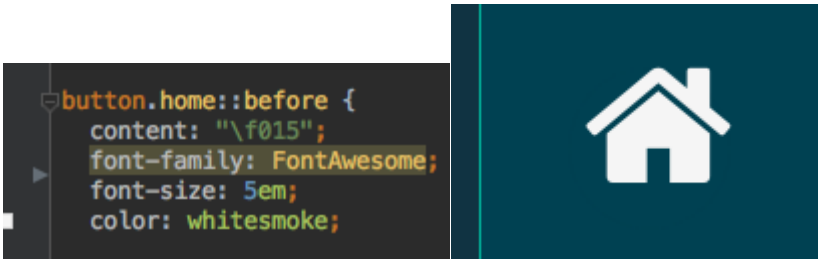
Tämä ratkaisu oli se joka lopulta toimi. Olimme ajatelleet asiaa liian hankalasti ja olettaneet että tarvitsisimme monta riviä koodia kääntääksemme XML-datan JSON-muotoon. Niin tässä pienessä pätkässä kuitenkin tapahtuukin. Fetch-funktiolla haetaan data, ja siitä saadaan vastaus (response). Json() metodia käyttämällä saadaan lupaus joka joka jäsentelee tuloksen JSON:ksi. (MDN web docs, 2019)

Kun datan hakeminen selvisi, lähdin selvittämään, miten saisin ruokalistat näkymään ohjelman ruokalista-osiossa nyt kun yhteys back-endiin oli saatu ja menu-ohjelman hallintajärjestelmästä laitettu data käytössäni. Selaimessa kehittäjäympäristöä apuna käyttäen etsin vanhasta menu-ohjelmasta React -painikkeen alta miten data siirtyi. Pystyin käyttämään sitä apuna kun koetin saada ruokalistat näkyville. Oikean reitin varmistaminen tapahtui lopulta console.log -toiminnolla. Sen avulla sain helposti oikean reitin selville, ja vielä tarkistettua sen kehittäjätyökalut console-ikkunan alta, ennen kuin kieli-painikkeille määrättiin reitit.

Aluksi jokaisella navigaation painikkeelle tehtiin omat polkunsa (path), eli reitti mitä seurata, jotta oikea ruokalista löytyisi. Liitteestä 5, 5.1 näkee miltä polku alunperin näytti. Kun sain sen toimimaan, työkaverini ehdotti tekemään reitityksen niin, että tulevaisuudessa olisi helppo lisätä esim. uusi kielipainike navigaatiopalkkiin ilman sen suurempaa koodin kirjoittamista. Tämä tarkoittaisi sitä että sen sijaan että jokaisella napilla olisi koodattuna erikseen oma polkunsa, tekisin ratkaisun jossa jokaisella kielellä olisi oma tunnus, jolle löytyisi sitä vastaava tunnus XML-tiedostosta. Näin uuden painikkeen lisäämällä ja sille tunnuksen lisäämällä pystyy lataamaan uuden ruokalistan helposti näkyville, kunhan tunnus on tiedossa. Tunnuksen selvittämiseksi tekee helpoksi se, että XML-tiedostossa jokaisella kielellä oli jo oma ID:nsä. Liitteestä 5, 5.2 näkee siistimmän ratkaisun. Ruokalistat haetaan myös XML-urlista ja ne tulevat näkyviin taustakuvana ruokalista-osioon.

3.9 Käyttöliittymän muokkaaminen toiminnallisuuden valmistuttua

Toiminnallisuuden valmistuttua lähdin erkanemaan hieman alkuperäisen menu-ohjelman ulkonäöstä. Alun perin ideana oli vain saada “Home” -painikkeeseen FontAwesome-fontilla kuvake, sillä niin oli tehty myös aikaisemmassa menu-ohjelmassa. Kopioin edellisen menu-ohjelman tiedostoista fonttitiedostot, ja lisäsin ne tyylitiedostooni, eli CSS -tiedostoon. Näin sain käyttöön myös kaikki kuvakkeet, jotka tulevat FontAwesome fontin mukana. ”Home” -painikkeen kuvan esille saaminen ei kuitenkaan ollut niin yksinkertaista kuin olin olettanut. FontAwesomen sivuilta löysin oikean koodin “/f015” jonka lisäämällä CSS:ään kyseisen painikkeen kohdalle, kuvan olisi pitänyt näkyä. Niin ei kuitenkaan tapahtunut, joten lähdin tutkimaan edellistä menu-ohjelmaa selainta kehittäjätyökaluilla. Valitsin tavan jolla pystyy tarkastelemaan elementtejä klikkaamalla niitä, ja huomasin ”home” -painikkeen kohdalla sanan “::before” jota ei kuitenkaan löytynyt edellisen projektin tyylitiedostosta. Kokeilin lisätä sen omaan CSS-tiedostooni, ja vihdoinkin ”home” -painike tuli näkyviin. Kuvassa 7 ja 8 voi nähdä home-painikkeen kuvakkeen ja miten se tehtiin.



Kuva 7. ”Home” -painike CSS-tiedostossa (Kinnunen)
Kuva 8. ”Home” -painike itse ohjelman navigointipalkissa.

Tämä oli asia joka toimi ilman että ymmärsin lainkaan miksi, joten lähdin selvittämään sitä jälkeenpäin. Sain selville, että ”::before” käytetään CSS:ssä kun halutaan lisätä jotain elementin eteen. Se tulee silloin ensimmäisenä näkyviin. (MDN web docs, 2019)

Saatuani ”home” -painikkeen kuntoon, tulin siihen tulokseen, että se näytti paremmalta kuin aikaisemman menu-ohjelman ”home” -painike jonka ympärillä oli vielä suurempi laatikon muotoinen painikealue. Tämä näytti siltä kuin pelkästään itse valkoinen kuvake olisi painike. Todellisuudessa tekemäni painikkeen ympärillä oli myös laatikon muotoinen painikealue, mutta sen väri oli vain määrätty samanväriseksi kuin navigaatiopalkki. Pohdin työkaverin kanssa, näyttäisivätkö painikkeet paremmilta, jos ne olisivat pyöreät laatikoiden sijaan ja että tekstin sijasta kielinapeissa näkyisikin maiden liput? Perustelumme tälle oli se, että jokaisessa kielessä ei käytetä samanlaisia aakkosia. Uudessa menu-ohjelmassa olevilla kielillä siitä ei olisi ollut ongelmaa, mutta tulevaisuutta ajatellen pohdimme, että esimerkiksi aasialaisen asiakkaan olisi helpompi tulkita lippuja, kuin lukea painikkeiden

tekstit. Vanhat painikkeet teksteillä ovat kuitenkin edelleen tallessa koodissa, ja ne saa helposti vaihdettua taas takaisin.

Uudet painikkeet näkee liitteestä 4. Haasteena tässä oli se, etten aluksi saanut asetettua painiketta linkin sisälle, ja tulkitsin virheellisesti, ettei niin voinut tehdä. Oikeasti en pystynyt tekemään niin ajatteluvirheen tai kirjoitusvirheen takia. Hetkeksi lähdin siis pohtimaan navigointinappuloiden toteuttamista tekemällä kuva-tiedostoista linkkejä, mutta palasin takaisin oikealle reitille.

Uudet painikkeet toteutettiin lopulta siten että painikkeen normaali kulmien pyöreysaste (radius) muutettiin CSS-tiedostossa siten että niistä tuli ympyröitä. CSS-tiedostoon asetettiin vain yleiset ohjeet painikkeen tulevaan taustakuvaan, jotta se tulisi näkyviin nätisti keskitettynä ja niin että se peittäisi koko painikkeen. Itse kuvat asetettiin suoraan jokaisen painikkeen kohdalle HTML-koodiin. Liitteestä 6 näkee, kuinka tämä on toteutettu.

3.10 Projektin siirtäminen webOS kehitysympäristöön

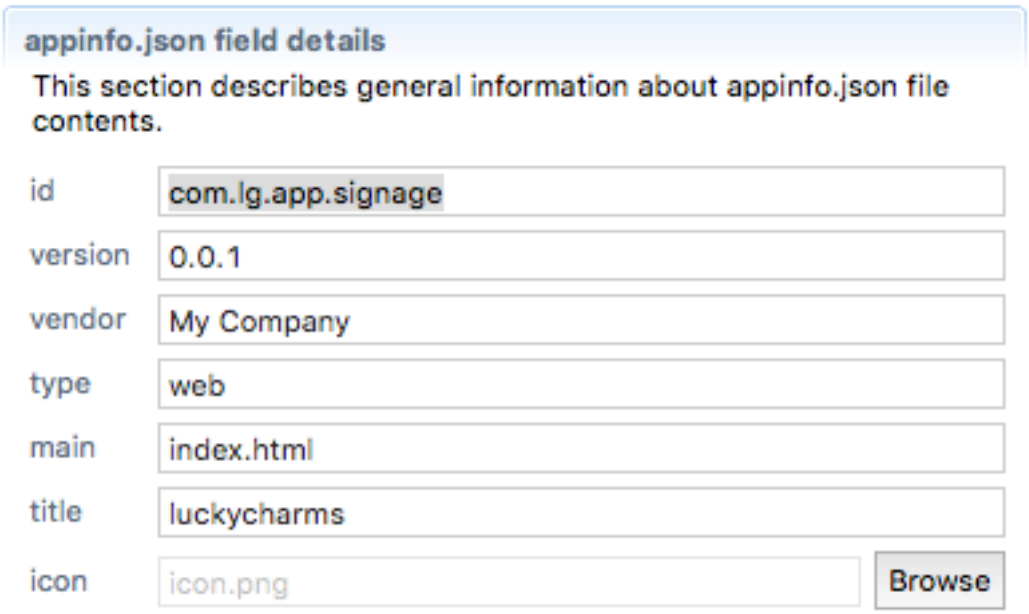
Kun uudessa menu-ohjelmassa oli valmiina toiminto ja ulkonäkö, tuli aika siirtää projekti webOS-kehitysympäristöön. Näin voisin tehdä siitä webOS-applikaation, ja siten saada sen näkymään emulaattorilla. Pakkasin valmiin projektin WebStormissa, eli tähän asti käyttämässäni kehitysympäristössä komennolla ”npm build”, joka tekee build-nimisen kansion missä koko projekti sijaitsee. Se on siten helpompi siirtää lisäkehitykseen toisia projekteja varten. Build tarkoittaa projektin muodostamista loppukäyttäjälle jaettavaan muotoon.

Heti ensimmäinen ongelma jonka kohtasin jälleen kerran oli se, että uuden projektin tekeminen oli hankalaa. Olin jo aikaisemmin kokeillut dokumentointiin tutustuessani oman projektin tekemistä, ja saanut sillä yksinkertaisen “Hello World” tekstin näkyviin emulaattorissa. Nyt oli kuitenkin tarkoitus siirtää kokonainen projekti ja se hankaloitti hieman asioita.

WebOS IDE:llä täytyy tehdä projekti siinä tehtäville sovelluksille, mutta jos haluaa uuden webOS projektin, on pakko valita yksi yhdestätoista templatesta, eli valmiista mallista. Tyhjiä alustoja ei ole, joten valitsin taas yhden tarjotuista pohjista. Tutkin yhdessä työkaverini kanssa uuden projektin kansioita ja päädyimme poistamaan kaiken sisällön, korvaten ne build-kansion sisällöllä, jonka olimme React-projektista saaneet. Tämä ei toiminut, sillä tiedostot eivät tulleet lainkaan näkyviin kehitysympäristön päivityttyä. Jäljelle jäi tyhjiä

kansioita mikä viittasi siihen, että webOS saattoi haluta siinä olevien projektien noudattavan tiettyä puurakennelmaa. WebOS:n dokumentaatiota tutkimalla sain selville, että tiedostojen ei ollut välttämättä sittenkään pakko olla tietyssä järjestyksessä.

Vanha projekti poistettiin tilaa viemästä ja uusi tehtiin samalla tavalla tilalle. Tällä kertaa katsoimme jokaisen kansion sisällön ja poistimme niitä sitä mukaan mitkä eivät olleet tärkeitä. Lopuksi projektiin jäi jäljelle enää neljä tiedostoa joita emme enää viitsineet poistaa ja joista yksi oli appinfo.json. Appinfo.json on siitä tärkeä, että se sisältää metadataa, eli datan kuvausta, mitä käytetään webOS Signagessa sovelluksen tunnistamiseen ja ohjelman ajamiseen. Sen on sijaittava jokaisen webOS Signage ohjelman juuressa, tai muuten sovellus ei yksinkertaisesti tule näkymään näytöllä tai emulaattorissa. Kuvassa 9 havainnollistetaan miltä appinfo.json näyttää, ja mitä tietoja se haluaa.



Kuva 9. webOS kehitysympäristön appinfo.json (Kinnunen)

Kun template-projektista oli poistettu kaikki tiedostot joita ei tarvittu, niiden tilalle kopioitiin projektin build-kansion sisältö. WebOS IDE:ssä päivittämällä tiedostonäkymän tiedostot tulivat näkyviin, mikä tarkoitti sitä, että webOS-projektin käynnistämistä pystyisi kokeilemaan. Projektin pystyi käynnistämään, mutta päällä oleva emulaattori näytti tässä vaiheessa vain mustaa ruutua. Miten siis edetään tällaisessa tilanteessa? WebOS IDE:ssä on mahdollisuus valita “Debug As” -> “webOS Application”. Tätä varten emulaattorin tulee olla päällä. Debugging, eli virheiden löytäminen toimii webOS Signagessa siten, että debugging avaa Google Chrome -selaimen, mistä aukeaa näkymä kehittäjätyökaluihin.

Täältä valitaan “console”, milloin saa näkyville virheilmoitukset siitä mikä asia estää webOS:ia toimimasta. Kuvasta 10 voi nähdä ne virheet jotka tässä vaiheessa sain.

Kuva 10. webOS debug-virheilmoitukset. (Kinnunen)

Kuvan virheilmoituksesta voi tulkita sen, että vaikka kaikki tiedostot olivat paikallaan, webOS ei osannut löytää niitä. Syy siihen arveltiin johtuvan siitä, että kun “npm build” komentoa käytetään tekemään build-kansio, koodi pakataan ja se jakautuu eri osiin (code splitting). React.js:ssä tästä syntyy tiedostoja, jotka on pakattu kuvassa 10 näkyviin “chunk”-tiedostoihin. Koska webOS:in arveltiin olevan kykenemätön lukemaan näitä tiedostoja, käännyttiin Googlen puoleen hakusanoin “disabling javascript chunks” ja “disabling code splitting in create react”. Tätä kautta löytyneet ohjeet olivat tapoja saada code splitting ja minifiointi pois ylikirjoittamalla projektin konfiguraatio. Minifiointi tarkoittaa sitä, että tiedostosta poistetaan kaikki turha, mm. välit ja kommentit, ja muuttujien nimet lyhennetään. Silloin koodi lyhenee pienentäen samalla tiedoston kokoa. Mikään näistä tavoista ei projektissa toiminut, minkä huomasi lukuisista virheviesteistä, joita WebStormin terminaaliin tuli ja siitä ettei ”npm build” enää luonut uutta build-kansiota. Näitä virheviestejä koetettiin selvittää useamman ihmisen voimin, mutta lopulta päätimme vaihtaa lähestymistapaa.

Ongelma oli siinä, että react-scripts (käynnistys- ja paketointiskripti) on tahallisesti rajoitettu helppokäyttöisyyden vuoksi. Kun React-projekti tehdään, lähes kaikki konfiguraatiot ovat staattisia ja kovakoodattuja. Konfiguraatioita ei voinut ylikirjoittaa tai muuttaa, joten lopuksi päädyttiin siihen, että projekti työnnettiin talteen BitBucketiin, minkä jälkeen sille päätettiin tehdä ”eject”-toiminto (apukirjastojen tuonti lähdekoodiin). Projektin kirjaston sen hetkinen tila kirjoitetaan package.json-tiedostoon, eikä sitä enää voi automaattisesti päivittää. Create-react-app ei kuitenkaan enää hallinnoinut konfiguraatioita, joten sitä pystyi nyt muuttamaan. Kun tähän tilaan oli päästy, buildin asetuksia lähdettiin etsimään suoraan projektista. Koska konfigurointi oli vapaasti muokattavissa, koodin minifoinnin pystyi ottamaan pois käytöstä, sillä sen ajateltiin auttavan virheiden ratkaisemisessa.

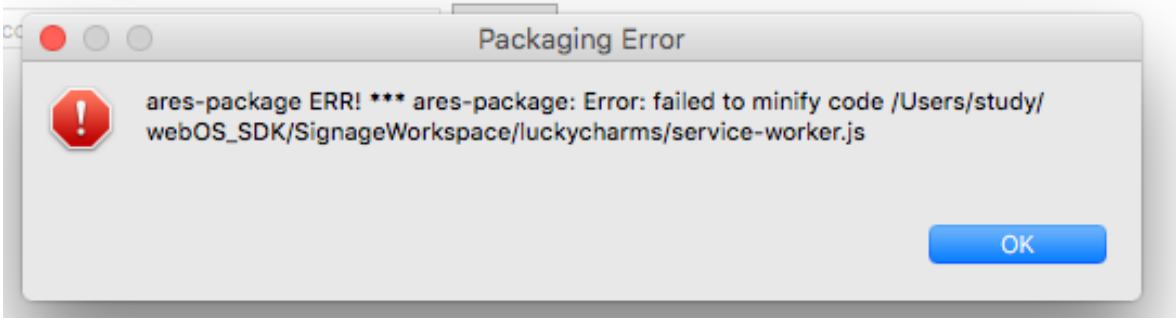
Kun etsii koko projektista sanaa “minif”, saa esille kaikki tiedostot joissa tuota sanaa on käytetty. Tämä valittiin sen takia, että ei ollut tiedossa missä muodossa sana esiintyisi koodissa. Tällä tavoin sai selville, että webpack.config.json -nimisessä tiedostossa oli käytetty yhtenä rivinä “minify: {“ ja klikkaamalla haun tulosta pääsi kyseiseen tiedostoon.

Kaikki mikä koodissa näkyi merkattuna todeksi “true”, muutettiin päinvastaiseksi eli false. Kuvasta 11 näkee lopputuloksen.

Kuva 11. pois käytöstä otettu minify. (Kinnunen)

Tästä samasta tiedostosta löytyivät myös code splitting -asetukset jotka kommentoitiin kokonaan pois. Tämä tarkoittaa sitä, että koodi otetaan pois käytöstä, mutta se näkyy vielä tiedostossa. Webstormiin muodostuneesta build-kansiosta pystyi tarkistamaan, ettei koodi ollut enää minifoitua ja eri osiin jaettuna.

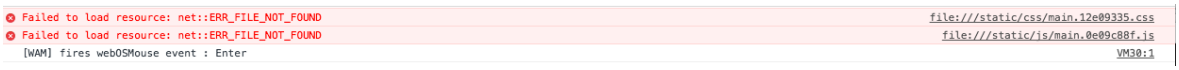
Aivan lopuksi “npm build” ajettiin vielä uudestaan, ja uudet build-kansiot siirrettiin vanhojen tilalle. Nyt kun projektin käynnisti webOS:issa, ruutuun ilmestyi välittömästi uusi virheviesti jonka voi nähdä kuvasta 12.



Kuva 12. pakkausongelma, puuttuva minifiointi (Kinnunen)

Minifiointi oli jäänyt vahingossa päälle käynnistäessäni webOS-projektia. Sitä pystyi projektin käynnistyksen yhteydessä muuttamaan itse. Kun projektin ajaa webOS:issa päälle painamalla vihreää “Run” painiketta, tai menemällä reittiä “Run As” -> “webOS Application”, webOS antaa mahdollisuuden muuttaa konfiguraatioitaan. Kohdan “Packaging type” eli pakkaustyylin alta pystyy valitsemaan joko “minify” tai “non-minify”, minkä jälkeen otetaan asetukset käyttöön painamalla “Apply” ja aivan viimeiseksi “Run”. Muutin siis

webOS:in kautta konfiguraatiota valitsemalla non-minify, siinä missä aiemmin olin koettanut käynnistää projektin “minify” asetuksilla. Sain projektin päälle, ja laitoin debugin taas päälle. Seurasi uusi virheviesti jonka voi nähdä kuvasta 13.



Kuva 13. Virheviesti joka kertoo ettei tiedostoja vieläkään löydy. (Kinnunen)

Koodin jakautuminen ja aiemmin minifointikin oli saatu pois, mutta webOS ei edelleenkään löytänyt oikeita tiedostoja. Tämä johtui siitä, että se ei löytänyt oikeita polkuja. Tästä syystä kokeilimme webOS:issa muuttaa HTML-tiedostoa hieman. Projektin index.html tiedostosta löytyy rivi, jossa CSS-tiedostoon viitataan linkillä. Tässä kohdassa näkyy polku, jonka eteen lisättiin piste. Piste muuttaa linkin polun relatiiviseksi projektin juurikansioon nähden. Tämä tarkoittaa sitä, että polku alkaa samasta tiedostosta missä index.html on, eli se alkaa projektin juuresta. Polun muuttamisen jälkeen debug-toiminto laitettiin taas päälle. Yksi virheviesteistä oli kadonnut mistä tulkittiin, että loputkin virheviesteistä poistuvat muuttamalla polkuja. Tätä varten palattiin takaisin WebStorm-kehitysympäristöön ja package.json:iin lisättiin kohta “homepage”:”./”, joka muutti kaikkien polkujen osoitteet. Tämän jälkeen tehtiin vielä viimeisen kerran uusi build-kansio, joka siirrettiin webOS-projektiin. Tällä kertaa debug ei näyttänyt ongelmia ja alunperin Webstormissa tekemäni ohjelma tuli viimeinkin näkyviin emulaattoriin. Ohjelman ruokalistat kuitenkin puuttuivat vielä, mutta ne tulivat näkyviin heti kun emulaattorille määriteltiin verkkoyhteys jota se käyttää, aivan kuten oikeassakin näytössä.

3.11 Emulaattorin näyttöasetukset.

Vaikka ohjelma näkyi tässä vaiheessa emulaattorilla vielä väärinpäin, jolloin puolet siitä hävisi pois näkyvistä. Tämä ei ollut toivottavaa, mutta oli helposti korjattavissa emulaattorin kautta. Koska emulaattori toimii kuin oikea tv-näyttö, siihen kuuluu kaukosäädin. Kun kaukosäätimestä painaa “Home”-painiketta, saa auki näkymän jossa voi muuttaa näytön asetuksia. Valikosta valitaan heti ensimmäinen painike, jonka alta paljastuu näytön yleiset asetukset. Täältä mennään kohtaan “display”, ja muutetaan “screen rotation” 270-asteella, jolloin projekti näkyy näytöllä siten miten se näkyisi oikealla näytöllä.

4 Pohdinta

Tässä luvussa pohditaan, miten opinnäytetyössä päästiin tavoitteisiin, miten aikataulutus sujui, muutamia jatkokehitysideoita, ja mitä opin tämän projektin aikana.

4.1 Tavoitteiden saavutus

Opinnäytetyön tavoitteena oli tehdä Craneworks Oy Ltd:lle Javascript-pohjainen menu-ohjelma, joka toimii LG:n soc-sirullisella kosketusnäytöllä. Ohjelma toimii rajausten mukaisesti eli se muistuttaa menu-ohjelmaa jonka pohjalta se tehtiin, ja navigointipalkista kieltä vaihtamalla siinä vaihtuu ruokalista. Ruokalistoja pystyy vaihtamaan hallintajärjestelmän puolelta, milloin ne päivittyvät myös ohjelmassa. Projektin valmistuttua lisätuloksena sain myös selvitettyä lopullisesti sen, että webOS-ohjelman voi ohjelmoida toisella kehitysympäristöllä ja siirtää sen jälkeen webOS:in omaan kehitysympäristöön.

4.2 Projektista opittua

Opin mielestäni projektin aikana lukemaan paremmin Javascriptia, HTML:ää ja hahmottamaan paremmin miten React.js toimii. Ehdin tehdä codecademyn React.js kurssit ja katsomaan opiskelumateriaaliksi annetut WesBos React-tutoriaali-videot joista oli myös paljon apua siinä vaiheessa, kun lähdin tekemään navigointipalkin toimivuutta. Ensimmäiseksi React-projektikseni opinnäytetyö onnistui mielestäni hyvin, vaikka tarvitsin välillä enemmän apua. Yksi tärkeistä asioista mitä mielestäni projektin aikana opin oli sanasto mitä kertyi projektia tehdessä, kuten build, code splitting, minify ja system on a chip.

Opin että virheviestien lukeminen on turhauttavaa, mutta vaikka ne eivät suoraan kertoisi mikä on vialla, ne useimmiten kertovat missä vika on. Kun kehitysympäristössä katsoo consoleen tulevia ilmoituksia, ne kertovat melko tarkasti missä tiedostossa ongelma on, tai antavat virheviestin jonka googlaamalla saattaa löytyä foorumeilta ratkaisuja kyseisiin ongelmiin. Opin periaatteessa sen, että punainen virheviesti on parempi kuin se, ettei tule virheviestiä mutta mikään ei tunnu toimivan. Uusi virheviesti on oikeastaan melkein ilon aihe, sillä siitä tietää että on päässyt taas askeleen eteenpäin.

Yksi asia jota olin jo aikaisemmin tehnyt, oli työskentely yhdessä muiden kanssa. Projektin aikana opin mielestäni kommunikoimaan paremmin ja pyytämään apua kun tarvitsin sitä. Toisten kanssa työskentelyn aikana täytyy mielestäni myös uskaltaa kysyä mitä tapahtuu jos on pudonnut kärryiltä. Asioiden selittäminen ja oppiminen on paljon helpompaa siinä vaiheessa kun sitä tehdään, kuin tehdä jokin asia tietyllä tavalla ja vasta sen jälkeen miettiä, että miten asia toimii.

Projektin aikana tulin oppineeksi myös asioita kuten sen, että jos projektin konfigurointia ei voi muuttaa, projektille voi tehdä “eject”-toiminnon, jotta siihen pääsee käsiksi. Silloin sitä ei voi tosin enää päivittää automaattisesti, mutta sillä oman projektini kohdalla BitBucketiin tehtiin “ejektoidulle” projektille oma “Git branch”, niin projektin vanhan version voi “git pull”-komentoa käyttämällä saada käyttöönsä, päivittää, suorittaa eject ja muuttaa konfiguraatiot taas. Näin projektin pystyisi taas päivittämään automaattisesti.

4.3 Jatkokehitysideat

Itse sovelluksessa en ole tyytyväinen siihen, etten ehtinyt saada siihen kunnolla animaatioita, kun kieltä vaihdetaan. Mielestäni sovellus muistuttaa paljon edellistä menu-ohjelmaa mutta siinä voisi tehdä pieniä esteettisiä muutoksia, että se näyttäisi yhtä hienolta. Mielestäni navigointipalkin sisältöä voisi keskittää pystysuunnassa vähän enemmän, etteivät kaikki painikkeet löytyisi ylhäältä.

Toinen asia mihin haluaisin palata, on Menu Nordic -ruokalistojen haku. Periaatteessa ruokalistat olisi voinut hakea täysin samalla tavalla kuin kieletkin, mutta koska “Menu Nordic” ei ole kieli, halusin pitää ne erillään.

Myös siitä oli puhetta, että jatkossa olisi järkevää käyttää projekteissa enemmän koodin sisällyttämistä html-tiedostoon (inline). Jos näin tehtäisiin niin kaikki koodi koottaisiin index.html-tiedostoon, mikä tekisi päivittämisestä ja jakelusta helpompaa.

4.4 Aikataulutus

Tämän projektin aikana tulin huomanneeksi, että aiktaulussa pysyminen oli minulle hyvin hankalaa, vaikka olen tottunut suorittamaan tehtäviä ajoissa. Vaikka minulla oli monta kuukautta aikaa tehdä projektia, loppupäässä tuli silti hirveä kiire. Mielestäni en osannut kunnolla arvioida kuinka pitkään tiettyjen asioiden tekemisessä minulla menisi, ja aliarvioin senkin kuinka pitkään opinnäytetyön kirjoittamisessa menisi. Aikataulussa oli siis aluksi tarpeeksi aikaa, mutta päätyi lopulta olemaan hyvin tiukka.

Yksi suurimmista syistä miksi aikatauluni petti oli se, että olin sen verran innokas aloittamaan uutta projektia, etten malttanut istua alas ja kirjoittaa opinnäytetyön teoriapohjaa kunnolla silloin kun siihen olisi marraskuun aikana ollut paljon aikaa. Olin myös ymmärtänyt väärin teoriapohjan tärkeyden opinnäytetyössä, joten se oli aluksi vain sivun pituinen. Käytin silloin aikani mieluummin React-tutoriaalien tutkimiseen, ja HTML-pohjan kanssa projektin ulkonäön muokkaamiseen. Tutkin myös vanhaa menu-ohjelmaa aivan liian

kauan, eikä siitä lopulta edes ollut niin paljon hyötyä muistettuani, että uudessa menu-ohjelmassa saisin tehdä ratkaisut miten halusin, eikä vanhasta tarvinnut ottaa lainkaan mallia. Vietin edellisen menu-ohjelman tutkimisen parissa ainakin muutaman viikon minkä olisin voinut käyttää suoraan joko opinnäytetyön kirjoittamiseen tai omassa projektissa etenemiseen. Aikataulun venymiseen vaikutti osittain myös ajoittainen oman osaamisen puute. Paljon aikaa meni asioiden dokumentaation tutkimiseen ja ongelmien ratkaisujen etsimiseen Googlesta.

Lähteet

Craneworks. 2018.
Luettavissa: https://www.craneworks.fi/.
Luettu: 03.12.2018

React.
Luettavissa: https://reactjs.org/.
Luettu: 24.02.2019

Pandit Nitin. What is ReactJS and why should we use it? 2018
Luettavissa: https://www.c-sharpcorner.com/article/what-and-why-reactjs/.
Luettu: 24.02.2019

Technopedia. 2019. System on a Chip (SoC)
Luettavissa: https://www.techopedia.com/definition/702/system-on-a-chip-soc.
Luettu: 24.02.2019

React. 2019. Components and Props
Luettavissa: https://reactjs.org/docs/components-and-props.html.
Luettu: 01.03.2019

Negi, M. 2017. React Components Explained.
Luettavissa: https://codeburst.io/react-components-explained-96718311f20b.
Luettu: 01.03.2019

Atto E. 2018. Kagga J. ”Understanding React Components”.
Luettavisa: https://medium.com/the-andela-way/understanding-react-components-37f841c1f3bb.
Luettu: 01.03.2019

MDN web docs. 2019. CSS ::before (:before) pseudo-element
Luettavissa: https://developer.mozilla.org/en-US/docs/Web/CSS/::before.
Luettu: 01.03.2019

Codecademy. 2019. The Virtual DOM
Luettavissa: https://www.codecademy.com/articles/react-virtual-dom.
Luettu: 19.04.2019

Honkanen J. 2017. “ReactJS”
Luettavissa: https://www.theseus.fi/bitstream/handle/10024/138247/Honkanen_Joni.pdf?sequence=1.
Luettu: 07.05.2019

Hirvonen, P. 2015. Medianäyttöjen etävalvonta.
Luettavissa: https://www.theseus.fi/bitstream/handle/10024/94531/Hirvonen_Pekka.pdf?sequence=1.
Luettu: 07.05.2019

Berg, J. 2018. Moderni web-sovellus hierojalle.
Luettavissa: https://www.theseus.fi/bitstream/handle/10024/142191/Berg_Janne.pdf?sequence=1&isAllowed=y.
Luettu: 07.05.2019

W3schools. 2019. What is the HTML DOM?
Luettavissa: https://www.w3schools.com/whatis/whatis_htmldom.asp.
Luettu: 07.05.2019

2Kmediat.com. 2000. Mikä on DOM?
Luettavissa: https://www.2kmediat.com/dhtml/dokumenttimalli.asp.
Luettu: 07.05.2019

Blog. DA-14. “Top 10 Advantages of using react.js”
Luettavissa: https://da-14.com/blog/its-high-time-reactjs-ten-reasons-give-it-try.
Luettu 07.05.2019

Brandnews. Värien psykologia ja merkitys viestinnässä
Luettavissa: http://brandnews.fi/varien-psykologia-merkitys-viestinnassa/.
Luettu: 8.05.2019

Felix, H. 2019. How colors influence people: the psychology of color in business marketing.
Luettavissa: https://www.lifehack.org/398377/how-colors-influence-people-the-psychology-of-color-in-business-marketing.
Luettu: 16.05.2019

MDN web docs. Body.json()
Luettavissa: https://developer.mozilla.org/en-US/docs/Web/API/Body/json.
Luettu: 13.05.2019

Rwieruch. 2018. How to fetch data in React
Luettavissa: https://www.robinwieruch.de/react-fetching-data/.
Luettu: 13.05.2019

Bincl, S. 2015. Centre liikekeskuksen digital signage
Luettavissa: https://www.theseus.fi/bitstream/handle/10024/91722/Bincl_Suzana.pdf?sequence=2.
Luettu: 15.05.2019

Tietokonemaailma. 2019. Mikä on soC (järjestelmä sirulla)?
Luettavissa: https://rivervalleygetaway.com/fi/smartphone/1435-simple-questions-what-is-a-soc-system-on-a-chip.html.
Luettu: 15.05.2019

Lauanne, M. 2012. Digitaalisen medianäytön hyödyntäminen markkinointiviestinnässä: Case Fazer.
Luettavissa: https://www.theseus.fi/bitstream/handle/10024/47420/Lauanne_Marja.pdf?sequence=1&isAllowed=y.
Luettu: 15.05.2019

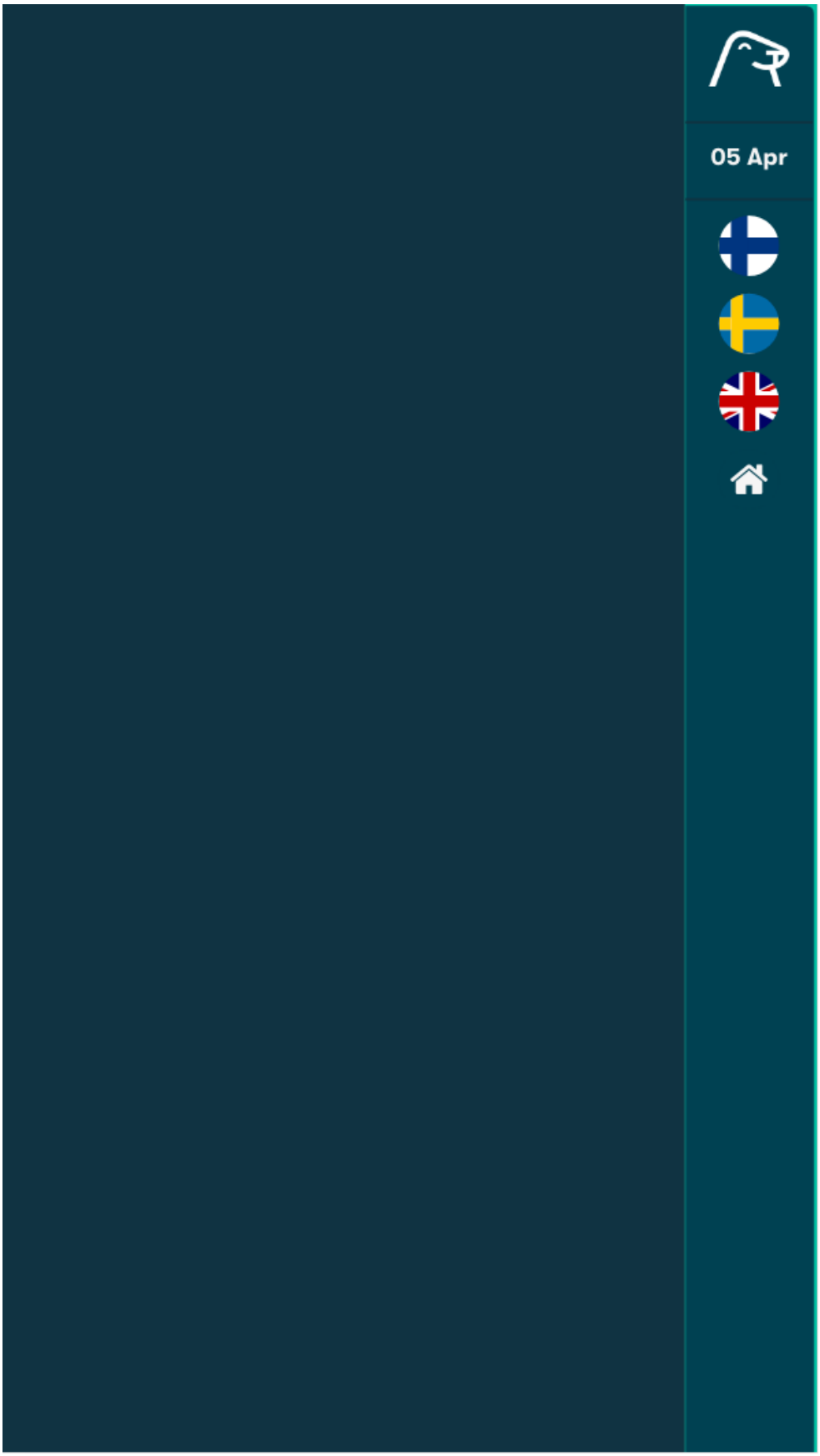
LG. 2015. Benefits of LG webOS
Katsottavissa: https://youtu.be/mcMHnCaUsuk.
Katsottu: 15.05.2019

Liitteet

Liite 1 – Valmis käyttöliittymä joka näyttää samalta kuin edeltäjänsä

Liite 2 - Toiminnallisuus

Liite 3 – Kielipainikkeet korvattu maiden lippujen kuvilla.



Liite 4

```
<content>
    <Switch>
        <Route path='/fi' render={() => <Fi image={menu.fi[0].image}/>}/>
        <Route path='/sv' render={() => <Sv image={menu.sv[0].image}/>}/>
        <Route path='/en' render={() => <En image={menu.en[0].image}/>}/>
    </Switch>
</content>
```

5.1 Vanha reititinratkaisu

```
/**
 * Uudelleennimetään menus-objekti. Luodaan tyhjä array johon asetetaan reitti ja kuvatiedot for-loopissa.
 */
const menus = this.state.data.restaurants[0].menus
var routes = []
for (var key in menus) {
    routes.push({path: '/' + key, image: menus[key][0].image})
}

/**
 * Käydään arrayn kaikki rivit läpi ja muodostetaan React komponentti jossa on oikeat reitit ja kuvat.
 */
const RouteMenu = ({routes}) => (
    routes.map(route => (
        <Route key={route.path} path={route.path} render={() => <Menu image={route.image}/>}/>
    ))
);
```

5.2 Uudempi reititinratkaisu

Liite 5 – lippujen kuvat painikkeissa

```
<div className="languagebox">
    <ul>
        <li>
            <Link to='/fi'>
                <button className="languageButton" style={{backgroundImage: "url('./images/fi_FI.svg')"}}>
                    {/*Suomi*/}
                </button>
            </Link>
        </li>
```