Nostolaitteen moderni käyttöliittymä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

HAMK, Tieto- ja viestintätekniikka

Kevät, 2019

Henri Tulus

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tieto- ja viestintätekniikka
Riihimäki

Tekijä Henri Tulus **Vuosi** 2019

Työn nimi Nostolaitteen moderni käyttöliittymä

Työn ohjaaja/t Petri Kuittinen

TIIVISTELMÄ

Työn tavoitteena oli kehittää uusi käyttöliittymäsovellus siltanosturille. Työssä verrattiin erilaisia tunnettuja JavaScript-sovelluskehyksiä, valittiin niistä sopivin ja rakennettiin sen avulla prototyyppimalli käyttöliittymästä.

Uuden käyttöliittymän ideana oli parantaa sen suorituskykyä, helpottaa käyttöliittymän kehitystä tulevaisuudessa ja lisätä siihen mobiiliyhteensopivuus. Näiden tavoitteiden saavuttamiseksi verrattiin JavaScript-sovelluskehyksiä.

Opinnäytetyössä verrattiin viittä erilaista sovelluskehystä. Vertailun pohjalta selvisivät erityisesti vanhojen ja modernien sovelluskehysten erot arkkitehtonisissa malleissa ja projektin rakenteessa. Tärkeimmäksi kriteeriksi nousi sovelluskehyksen suosio, jonka perusteella valittiin React-sovelluskehys.

Työn lopputuloksena syntyi käyttöliittymäprototyyppi, joka pystyy hakemaan nostolaitteen parametreja ja näyttämään ne dynaamisesti websivulla. Prototyypin pohjalta voidaan hyvin lähteä jatkokehittämään lopullista käyttöliittymää.

Avainsanat JavaScript-sovelluskehykset, Web-tekniikat, Käyttöliittymä

Sivut 44 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Information and communication Technology
Riihimäki

Author Henri Tulus **Year** 2019

Subject Industrial cranes modern user interface

Supervisors Petri Kuittinen

ABSTRACT

The aim of the thesis project was to develop a new user interface application for an industrial crane. In the thesis project different JavaScript frameworks were compared, the most suitable one was picked and a protype model of the user interface was built.

The idea behind the new user interface was to improve its performance, to make it easier to be developed in the future and to add mobile integration there. To achieve these goals different JavaScript frameworks were compared.

Five different frameworks are compared in this thesis. Based on the comparison it was found that the key differences between these frameworks were architectural models and project structure. The most important criterion here was popularity, based on which the React-framework was chosen to be used in the thesis project.

The outcome of the thesis project was a user interface prototype, which could retrieve hoist parameters and display them dynamically on the webpage. Based on the prototype, the user interface can be further developed.

Keywords JavaScript frameworks, Web techniques, User interface

Pages 44 pages

SISÄLLYS

1 JOHDANTO

Työ on tehty Konecranesille, joka on maailman johtavia nostolaitteiden valmistajia (Konecranes, n.d.). Konecranes rakentaa nostolaitteita muun muassa satamiin, hallirakennuksiin sekä tuotannonaloille kuten auto- ja metallialalle (Konecranes, n.d.). Nostolaitteissa on teräsosia, jotka muodostavat nosturin rungon ja nostavat osat, sähkölaitteet, jotka mahdollistavat nosturin toiminnan sekä käyttöliittymä, jonka avulla nostolaitetta ohjataan.

Yksi graafista käyttöliittymää käyttävä Konecranesin nostolaite on siltanosturi, joka on tuote nimeltään SMARTON. Tämän tyyppisiä nostolaitteita ohjataan erillisellä ohjaimella, jossa on yksinkertainen graafinen käyttöliittymä ja joystickit nostolaitteen ohjaukseen. (Konecranes, n.d.) Kuvassa 1 on siltanosturi ja siihen käytettävä radio-ohjain.

Kuva 1. Konecranes-siltanosturi ja ohjain (Konecranes, n.d.) (Konecranes, n.d.)

Modernit nostolaitteet käyttävät graafista käyttöliittymää. Käyttöliittymät kehitetään esimerkiksi nostolaitteita huoltavalle henkilöstölle. Nykyisessä nostolaitteen käyttöliittymässä voi muuttaa nosturin parametreja ja tarkkailla nostolaitteen tilaa. Virheet, hälytykset ja tapahtumat näkyvät käyttöliittymässä viesteinä.

Yksi Konecranesin hallinostureissa käytettävistä käyttöliittymistä on nimeltään WebKey. Se on ohjelmoitu webtekniikoita hyödyntävän sovelluskehyksen avulla, toisin sanoen WebKey on kehitetty selaimessa toimivaksi nettisivuksi. Opinnäytetyön aiheena on rakentaa uusi käyttöliittymä kyseiselle nostolaitteelle. Työn tavoitteena on verrata erilaisia JavaScript-sovelluskehyksiä, valita niistä sopivin ja rakentaa käyttöliittymän prototyyppi.

Työn tutkimuskysymyksenä on: Mikä JavaScript-sovelluskehys toimii parhaiten kyseiselle käyttöliittymälle? Sovelluskehyksen valinnalle tärkeitä kriteerejä ovat muun muassa sovelluskehyksen nopeus ja käytettävyyden helppous. Tarkoituksena on siis vertailla erilaisia JavaScript-sovelluskehyksiä ja tutkia, mikä niistä sopii parhaiten uuteen käyttöliittymään.

Lopullisena tuotoksena on synnyttävä prototyyppikäyttöliittymä, jossa käytetään valittuja rakennuspalikoita. Konecranesin käyttöliittymät käyttävät Konecranesin omaa värimaailmaa ja tyyliä, joita käytetään myös prototyyppikäyttöliittymässä. Tärkeänä ominaisuutena prototyyppikäyttöliittymässä on nostolaitteen parametrien kutsuminen. Ainakin yhtä nostolaitteen parametria on kutsuttava ja muutettava. Lopullinen tulos on prototyyppiversio uudesta käyttöliittymästä, josta sitä voidaan lähteä jatkokehittämään.

2 KÄYTTÖLIITTYMÄN TARVE

Konecranesin nykyinen WebKey-käyttöliittymä on käyttökelpoinen, mutta siinä on havaittu parannuskohteita. Tämän kappaleen alla käydään läpi käyttöliittymän tarve ja ne asiat, joita uuden käyttöliittymän on parannettava.

2.1 Alustatuki ja suorituskyky

Nykyinen käyttöliittymä toimii selainpohjaisena, ja se on suunniteltu toimimaan parhaiten pöytäkoneissa. Käyttöliittymästä on kehitetty mobiiliversiota. Mobiiliversio on nykyisestä käyttöliittymästä täysin erillinen sovellus, minkä takia sekä nykyistä käyttöliittymää että sen mobiiliversiota täytyy kehittää erikseen. Tämän ongelman ratkaisemiseksi uusi käyttöliittymä rakennetaan hybridimalliksi. Hybridimalli on sovellusmalli, jossa käyttöliittymä on suunniteltu toimimaan sekä pöytäkoneilla että mobiililaitteilla (Cowart, 2012).

Yksi parannettavista asioista on käyttöliittymän suorituskyky. Käyttöliittymä on ajoittain hidas ja huonosti reagoiva. Huono suorituskyky näkyy sekä käyttäjän selatessa käyttöliittymää että ajettaessa automaatiotestejä. Käyttöliittymän latausajat voivat nousta kymmeniin sekunteihin tai jopa minuutteihin. Huono suorituskyky johtaa myös sisällön latausongelmiin

navigoitaessa toiselle sivulle, jolloin edellisen sivun sisältö ladataan uudelle sivulle.

2.2 Käytettävyys

Yksi tärkeimmistä vaatimuksista on helppokäyttöisyys. Käyttöliittymä rakennetaan alustavasti huollon henkilöille. Nostolaitteen huoltohenkilöt ovat vastuussa nostolaitteen ylläpidosta ja korjauksesta. Käyttöliittymän suunnittelun lähtökohta lähtee siitä ajatuksesta, että sen käyttö vaivatonta nostolaitetta huoltavalle henkilöstölle.

Nostolaitteen konfiguroinnin ja vianetsinnän avulla sitä huoltavat henkilöt saavat tietoa laitteesta. Nykyisessä käyttöliittymässä on sivut aktiivisille hälytyksille sekä hälytyshistorialle. Hälytykset antavat tietoa nostolaitteen vikatiloista ja tapahtumista. Asetukset-sivulla voi muuttaa nostolaitteen parametrejä. Parametrien avulla voidaan muuttaa nostolaitteen tilaa. Uudessa käyttöliittymässä on oltava toiminnallisuudet vähintään näihin toimintoihin.

2.3 Tekniset vaatimukset

Nostolaitteilla on tietyt tekniset vaatimukset perustuen käyttöliittymän nopeuteen, nostolaitteen sijaintiin ja testaukseen.

Yksi kriteeri käyttöliittymälle on toimivuus mobiililaitteissa. Kehittyneissä maissa älypuhelimien omistusprosentti voi olla jopa yli 80% (Chaffey, 2018). Mobiililaitteilla oli vuonna 2016 noin 2 miljardia käyttäjää (Krumins, 2015). Toisin sanoen suuri osa ihmisitä omistaa jonkinlaisen mobiililaitteen. Mobiililaitetuella varustettua käyttöliittymää voisi näin käyttää suurin osa käyttöliittymää hyödyntävistä henkilöistä. Tärkeää olisi yhteensopivuus kaikissa erilaisissa laitteissa riippumatta siitä, mitä käyttöjärjestelmää laite käyttää. Käyttöliittymää voisi näin myös käyttää teoriassa lähes missä tahansa mobiililaitteessa, jolloin nostolaitteen vianetsintään voisi käyttää lähes mitä laitetta tahansa, kuten vaikkapa omaa puhelinta.

Nostolaitteet voivat sijaita paikoissa, joissa on hidas Internet-yhteys tai ei Internet-yhteyttä ollenkaan. Tämän vuoksi käyttöliittymän on toimittava offline-tilassa. Käytännössä tämä tarkoittaa, että JavaScript sovelluskehyksen ja JavaScript-kirjastojen tiedostojen on oltava ohjelmistossa mukana. Linkitys JavaScript-kirjastoihin tapahtuu usein HTML-linkin avulla palvelimessa sijaitsevaan tiedostoon. Offline-vaatimuksen takia käyttöliittymässä ei myöskään käytetä pilvipalveluita.

Modernin käyttöliittymän yksi piirre on, että sitä voidaan haluttaessa testata automaattisesti. Käyttöliittymä on rakennettava niin, että testiautomaatio olisi mahdollista toteuttaa vaivattomasti. JavaScript-sovelluskehyksessä on oltava tuki automaatiotestauksen toteutukseen.

Muita tärkeitä kriteerejä on, että käyttöliittymän olisi käytettävä mahdollisimman vähän resursseja nostolaitteen omalta tietokoneelta. Uuteen käyttöliittymään on otettava huomioon myös tietoturva. Silloin huomioon otettavia asioita ovat muun muassa HTTPS-salaus ja autentikointi.

3 KÄYTTÖLIITTYMÄN KEHITYKSESSÄ KÄYTETYT TEKNIIKAT

Tässä luvussa käydään läpi työssä käytettyjä tekniikoita. Ohjelmistokehityksessä voi valita useista ohjelmointikielistä, työkaluista ja ohjelmointikirjastoista. Projektissa kehitetään selainpohjainen käyttöliittymä. Käytännössä tämä tarkoittaa, että käyttöliittymä rakennetaan Web- ja Ajax-tekniikoilla. Kaikki koodiesimerkit tässä kappaleessa on otettu koodieditorista.

3.1 Kuvaus- ja ohjelmointikielet

Web-sovelluskehityksessä käytetty kuvauskieli on HTML (Hyper-Text Markup Language). Ensimmäinen HTML-versio HTML 1.0 julkaistiin vuonna 1992 SGML-standardin (Standard Generalized Markup Language) pohjalta. HTML kehitettiin muiden Web-tekniikoiden ohella CERN:ssä Tim Berners-Leen toimesta helpottamaan datan siirtoa ja käsittelyä fysiikkalaboratoriossa. HTML, HTTP-protokolla ja URL (Uniform Resource Locator) kehitettiin tällöin CERN:n sisäiseen käyttöön, mutta näistä muodostui myöhemmin Internetin infrastruktuurin pohja. World Wide Web Consortium (W3C) standardoi HTML-kielen vuonna 1996 HTML 3.2-päivityksessä. (Møller & Schwartzbach, 2006) W3C hallitsee nykyäänkin HTML- ja CSS-standardeja (w3schools, n.d.).

HTML-kielen rakenne koostuu tageista, jotka esittävät sisältöä websivulla. Tageilla on alkutagi ja lopputagi, joiden väliin tulee haluttu sisältö. Kuvassa 2 käytetään -tagia kirjoittamaan tekstiä. Alkutagin jälkeen kirjoitetaan haluttu teksti, ja lopputagi sulkee sisällön.

```
<p>Tekstiä websivulla.</p>
```

Kuva 2. HTML-tagi

Kuvassa 3 on yksinkertainen esimerkki websivusta, jossa <h1>-tagi määrittelee sivulle otsikon ja määrittelee leipätekstin. Sivu määritellään websivuksi <html>-tagin avulla. Sivulla on ylätunnus <head>, jossa määritellään

sivun metatietoja, kuten merkistö ja sivun otsikko. Itse sivun sisältö määritellään <body>-tagin sisällä. Kuvassa 4 on websivulla näkyvä sisältö.

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>HTML sivu</title>
    </head>
    <body>
        <h1>HTML Sivu</h1>
        <p>Tekstiä websivulla.</p>
    </body>
</html>
```

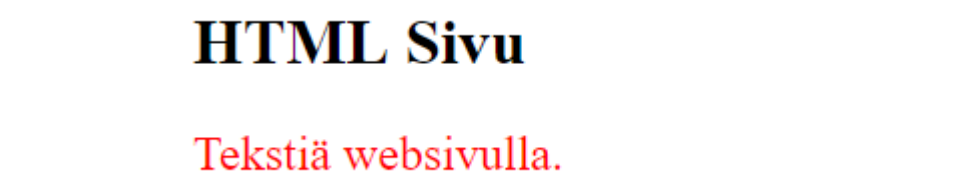
Kuva 3. Yksinkertainen HTML-sivu

Kuva 4. Kuvan 3 websivun sisältö

Alkuperäisessä HTML 1.0-versiossa ei ollut mahdollista muuttaa websivun tyyliä. Cascading Style Sheets (CSS) on tyyliohje HTML:n tyyppisille kuvauskielille, joka standardoitiin vuonna 1997 HTML 4.0 -versiossa. (Møller & Schwartzbach, 2006) CSS:n avulla voidaan muokata muun muassa sivun fonttikokoa, taustaväriä ja ryhmittelyä. HTML ja CSS on eroteltu niin, että HTML hoitaa sivun sisällön ja CSS hoitaa sivun tyylin. Kuvassa 5 tyylitetään aiemmasta HTML-esimerkistä ”Tekstiä websivulla.” kohta CSS:än attribuuttien avulla. Tekstin font-size -attribuutti määrää fonttikoon 24 pikselin kokoiseksi, ja color-attribuutti asettaa tekstin värin punaiseksi. CSS:än valitsimeksi on valittu p, jolloin tyylitys tapahtuu muuttamalla kaikkien <p>-tagien tyyli. Kuvassa 6 on websivulla näkyvä muutos.

```
p {
    font-size: 24px
;   color: red;
}
```

Kuva 5. Tyyli CSS:llä kuvan 3 HTML-sivuun



Kuva 6. Muutos sivun tyyliin

HTML ja CSS määrittävät sivun sisällön ja tyylin. Toiminnallisuutta websivulle haluttaessa käytetään ohjelmointikieltä. Ensimmäisenä selaimia tukevana interaktiivisuuden mahdollistavana ohjelmointikielenä julkaistiin vuonna 1995 JavaScript (Peltomäki & Nykänen, 2006). Suurin osa sivustoista käyttää JavaScriptia, jolle on kehitetty ja kehitetään lukuisia websivujen toiminnallisuutta lisääviä kirjastoja ja sovelluskehyksiä.

JavaScriptiä käytetään useimmiten webselaimissa. JavaScript-koodi pystyy vuorovaikuttamaan käyttäjän kanssa, hallitsemaan selainta ja muokkaamaan selaimessa näkyvää sisältöä. JavaScript on tulkattava ohjelmointikieli. Tämä tarkoittaa, että lähdekoodi käännetään tietokoneelle ymmärrettäväksi pieni osa kerrallaan. JavaScriptin muuttujat ovat dynaamisesti määritettyjä. Tämä eroaa tyypitetyistä ohjelmointikielistä niin, ettei niiden muuttujien tyyppejä tarvitse erikseen määritellä. (Flanagan, 2006)

JavaScriptin täytyy löytää HTML-elementit websivulta jonkin rakenteen avulla. DOM (Document Object Model) on tapa esittää websivun elementit rakenteena. Toisin sanoen se on HTML-dokumentin ohjelmointirajapinta. DOM:ia voidaan käyttää millä tahansa ohjelmointikielellä. JavaScript käyttää DOM-rakennetta löytääkseen websivun elementit. (Peltomäki & Nykänen, 2006) JavaScriptin käytöstä websivulla esimerkkinä kuvassa 7 on HTML-tiedostoon upotettu pätkä JavaScript-koodia. Button-tagin sisällä document.getElementById-metodi hakee demo-nimisen elementin DOM-rakenteen avulla ja asettaa sen arvoksi sen hetkisen ajan. Kuvassa 8 on websivulla näkyvä sisältö.

```
<html>
    <head>
        <meta charset="UTF-8">
        <title>HTML sivu</title>
    </head>
    <body>
        <h1>HTML Sivu</h1>
        <p id="demo">Tekstiä websivulla.</p>
        <button
            type="button"
            onclick="document.getElementById('demo').innerHTML = Date()">
            Click me to display Date and Time.
        </button>
    </body>
</html>
```

Kuva 7. JavaScript-koodipätkä HTML-sivulla

Kuva 8. Kuvan 7 websivun sisältö

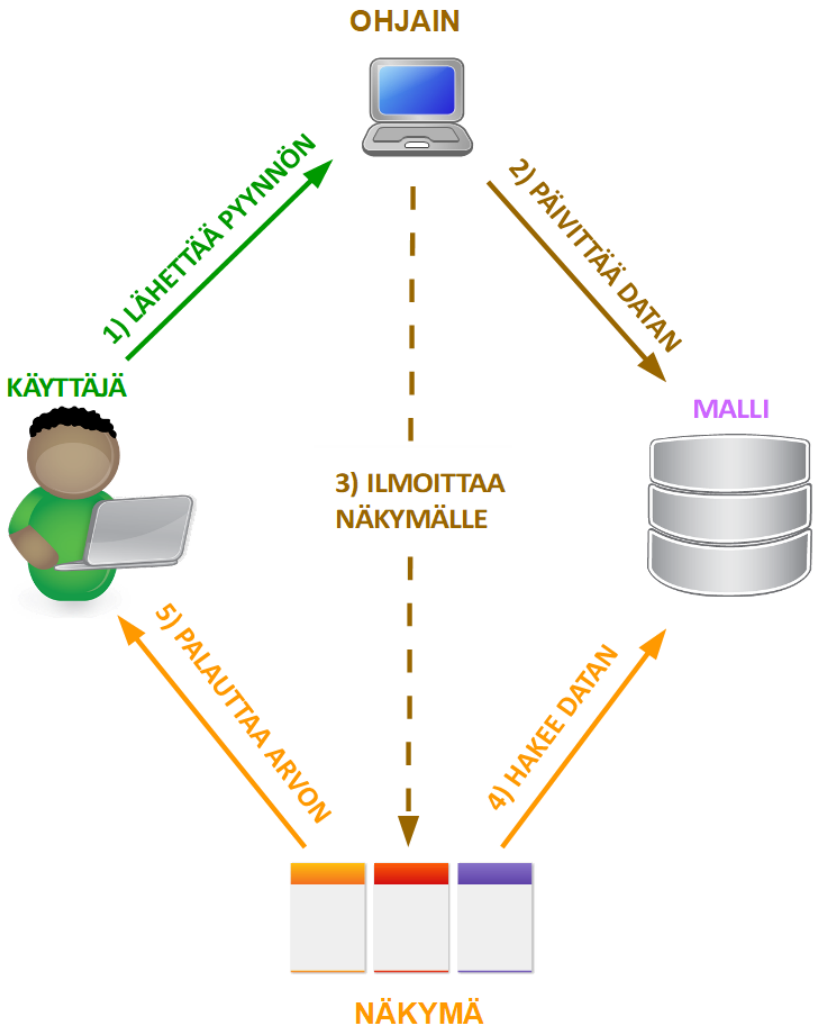
Dynaamisten sivujen saavuttamiseksi käytetään AJAX-tekniikkaa. Käytännössä AJAX (Asynchronous JavaScript and XML) on erilaisten tekniikoiden ja teknologioiden yhdistelmä. AJAXiin sisältyy JavaScript, XML-kuvauskieli ja DOM. AJAX on pohjimmiltaan standardi, jonka avulla voidaan tehdä websivulla palvelimelle asynkronisia pyyntöjä taustalla. Tämä tarkoittaa, ettei sivua tarvitse ladata kokonaan uudestaan joka kerta, kun halutaan näyttää sivulla uutta tietoa. AJAX hakee tiedon XMLHttpRequest-olion avulla. XMLHttpRequest-olio on kaikissa moderneissa selaimissa tuettu JavaScript-objekti, joka pystyy välittämään tiedonhakupyyntöjä eli HTTP-pyyntöjä palvelimelle. (Peltomäki & Nykänen, 2006)

3.2 Arkkitehtoniset mallit ohjelmistosuunnittelussa

Arkkitehtoniset mallit ovat ratkaisuja usein sovelluskehityksessä esiintyviin ongelmiin. Tässä kappaleessa käydään läpi käyttöliittymien kehityksessä usein käytettyjä malleja.

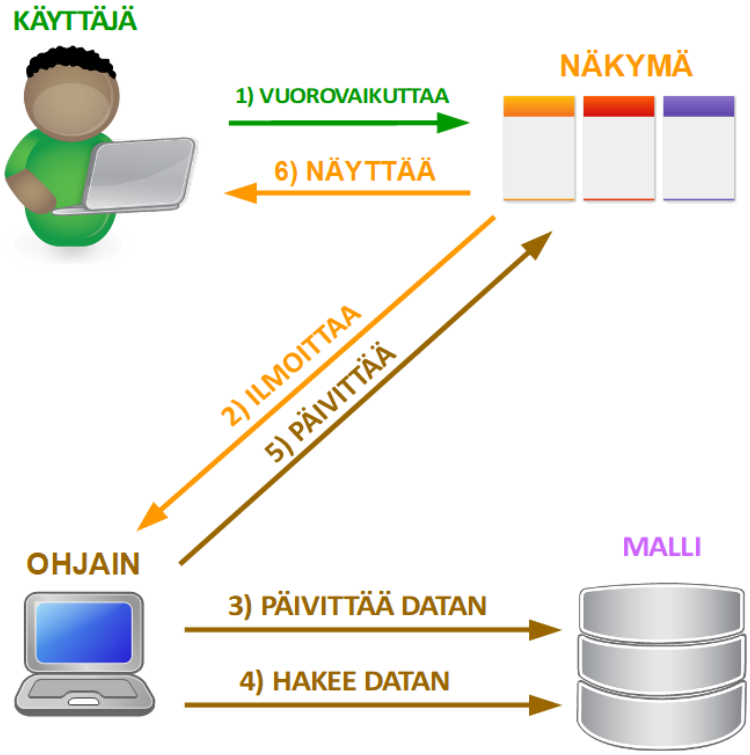
Usein malleista puhuttaessa puhutaan informaation jakamisesta eri osiin. Isoja projekteja on vaikea hallita, jos ohjelman logiikka ja data ovat yhdessä tai jos kaikki tieto on fyysisesti samassa tiedostossa. Näiden arkkitehtonisien mallien on tarkoitus jakaa tieto eri tavoin, jolloin ohjelman kehittämisestä ja ylläpidosta tulee helpompaa. Informaation jakamisella on myös se hyvä puoli, että jaettua koodia on helpompi käyttää uudelleen. Moni kehittäjä voi kehittää ohjelmaa samaan aikaan, jos he työskentelevät eri osien parissa.

MVC (Model-View-Controller) jakaa informaation kolmeen osaan: malliin, näkymään ja ohjaimeen. Malli hallitsee ohjelman dataa, informaatiota ja logiikkaa. Näkymä on käyttäjälle näkyvä osa applikaatiosta. Ohjain hallitsee käyttäjän syötteen ja muuttaa sen käskyiksi ohjelman mallille. (Wikipedia, n.d.b) Kuvassa 9 on malli MVC:n toiminnasta. Kuvan nuolet eivät kuvaa datan kulkusuuntaa, vaan ne kuvaavat kunkin osan tekemää pyyntöä.



Kuva 9. MVC-Malli (Bashir, 2014)

MVP (Model-View-Presenter) on MVC:n muunnos. Se kehitettiin helpottamaan yksikkötestausta. MVP:ssä malli on datan sisältävä ohjelman osa. Näkymä pitää sisällään käyttöliittymän näkyvän osuuden, ja se myös hoitaa käyttäjän syötteen. Näyttäjä (Presenter) hoitaa kommunikaation näiden kahden osan välillä. Näyttäjä hakee informaation mallista ja muuttaa sen oikeaan muotoon näkymälle. (Wikipedia, n.d.f) Kuvassa 7 on malli MVP:n toiminnasta. MVP-mallit voivat olla melko erilaisia. Esimerkiksi joissain malleissa näkymä-osa ei tee muuta kuin välittää käyttäjän syötteen näyttäjälle. (Block, 2008) Kuvassa 10 on malli MVP:n toiminnasta. Kuvan nuolet eivät kuvaa datan kulkusuuntaa, vaan ne kuvaavat kunkin osan tekemää pyyntöä.



Kuva 10. MVP-Malli (Bashir, 2014)

MVVM:n (Model-View-ViewModel) tarkoitus on erottaa graafisen käyttöliittymän kehitys mallin ja bisneslogiikan kehityksestä. Näkymä-osa on vastuussa dataobjektien muunnoksesta sellaiseen muotoon, että niitä on helppo esittää ja hallita. Malli esittää MVVM:ssä joko dataosaa tai ohjelman nykyistä tilaa. MVC- ja MVP-mallien tavoin näkymä on käyttäjän näkemä osa ohjelmasta. NäkymäMalli-osa käyttää sitojaa (binder). Sitoja on käytännössä automaattinen päivitys mallin ja näkymän välillä. NäkymäMalli on hyvin samankaltainen MVP:n näyttäjä-osaan. Ne eroavat toisistaan niin, että MVP:n näyttäjä hallitsee näkymän, kun taas NäkymäMalli sitoo mallin ja näkymän suoraan toisiinsa. (Wikipedia, n.d.c)

3.3 **JavaScript-sovelluskehykset**

Tässä kappaleessa käydään läpi erilaisia hyvin tunnettuja sovelluskehyksiä.

Pienikokoiset websovellukset rakennetaan usein SPA–arkkitehtuurimallin (Single-Page Application) avulla. Tällainen websovellus päivittää sivun dynaamisesti lataamatta koko sivua uudestaan joka kerta, kun sivulla muutetaan jokin arvo. Kaikki tiedot – HTML, CSS ja JavaScript tiedostot – haetaan single-page applicationissa yhdellä latauksella. (Wikipedia, n.d.d)

Isoissa yrityksissä kehitettäviä suurempia websovelluksia kehittää tiimi. Suurissa tiimeissä on useita kehittäjiä. Jotta kaikilla tiimin jäsenillä olisi käsitys sovelluksen toiminnasta, halutaan siihen tietty rakenne ja funktionaalisuudet. Tämän takia käytetään usein sovelluskehystä.

JavaScript-sovelluskehyksen tarkoitus on luoda standardoitu tapa rakentaa websovelluksia. Sovelluskehys on kuin sovelluksen luuranko. Sovelluskehys vähentää kehittäjän kuluttamaa aikaa matalatasoisien toiminnallisuuksien selvittämiseen. Esimerkiksi sivujen dynaaminen päivitys hoituu sovelluskehyksen toimesta. Kehittäjä voi näin käyttää aikansa sovelluksen vaatimuksien kehitykseen.

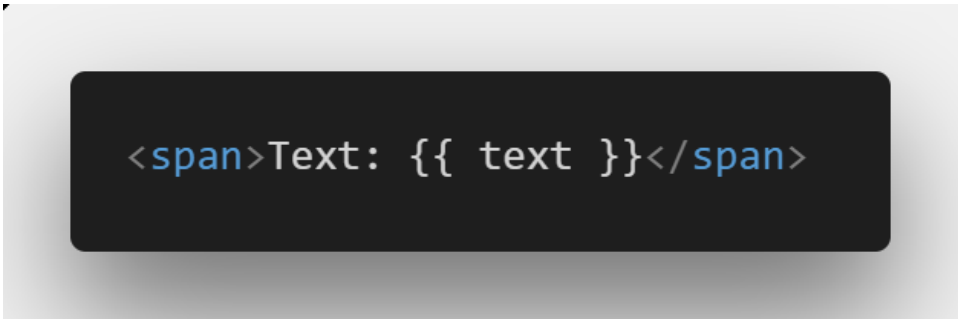
Sovelluskehykset ovat hyvin samankaltaisia JavaScript-kirjastoihin nähden, mutta niillä on muutama eroavuus. Sen sijaan, että kehittäjä kontrolloisi sovelluskehystä, sovelluskehys ohjaakin kehittäjää. Sovelluskehystä voi myös laajentaa käyttämään käyttäjän omaa funktionaalisuutta, kun taas kirjastoihin on tarkasti määritelty asiat, joita ne tekevät. (Kumar, 2015)

3.3.1 Vue

Vue on Evan Youn kehittämä JavaScript-sovelluskehys (Vuejs, n.d.a). Se on julkaistu vuonna 2014 (Github, n.d.g). Vue on progressiivinen sovelluskehys, tarkoittaen että sen avulla voidaan kehittää websovelluksia ilman ylimääräisiä JavaScript-kirjastoja ja -riippuvuuksia. Vue perustuu löyhästi MVVM-malliin (Vuejs, n.d.b). Vue käyttää muita JavaScript-kirjastoja tiettyjen toiminnallisuuksien saavuttamiseen. Nämä kirjastot ovat virallisesti Vuen tukemia.

Vue käyttää virtuaalista DOM-ympäristöä, mikä tarkoittaa, että käyttäjä päivittää Vue-instanssin, ja Vue-instanssi päivittää DOM:n. Vue käyttää komponentteja, jotka ovat reaktiivisia ja koottavia. Reaktiiviset komponentit päivitetään vain, jos ne luodaan samalla kun Vue-instanssi on luotu. Komponentit toimivat Vuessa käytännössä HTML-tageina. Vuessa komponenteilla voi rakentaa useista HTML-tageista koostuvan pätkän sisältöä ja käyttää sitä yksittäisenä HTML-tagina tiedostossa. Tällöin komponentteja voi uudelleenkäyttää hyvin samoin kuin funktioita JavaScriptissä. (Vuejs, n.d.c)

Syntaktisesti Vue käyttää normaalia HTML-tiedostoa tietojen esittämiseen ja omaa vue-päätteistä tiedostomuotoa komponenttien muokkaamiseen. Kuvassa 11 käytetään mustache-tagia näyttämään text-ominaisuus. Mustache-tagin sisällä text-objekti kuvaa arvoa, jonka muuttuessa teksti päivitetään myös dynaamisesti websivulla. (Vuejs, n.d.c)



Kuva 11. Vue ”mustache” syntaksi

Kuvassa 12 käytetään direktiiviä. Direktiivejä kuvataan v- -etuliitteellä ja ne sijoitetaan HTML-alkutagin sisälle. Direktiivit ovat JavaScript-määritelmiä. Arvon muuttuessa sivulla direktiivi tekee oman muutoksensa. Kuvan 9 direktiivi on v-bind, joka sitoo HTML-tagin sisällön direktiivin argumenttiin. Direktiivillä on argumentti href, joka määritellään kaksoispisteen jälkeen. Argumentti muuttaa HTML-attribuutin arvoa. Direktiivejä voivat myös olla esimerkiksi muista ohjelmointikielistä tutut if- ja for-lausekkeet.

Kuva 12. Vue direktiivi

Lisäksi direktiivit voivat kuunnella tapahtumia. Vue-komponenteille voi kirjoittaa metodeja, jotka toimivat kuten ohjelmointikielien funktiot. Kuvassa 13 v-on:click-direktiivi kuuntelee napin painallusta. Nappia painettaessa suoritetaan reverseMessage-metodi, joka kääntää merkkijonon järjestyksen.

```
<button v-on:click="reverseMessage">Reverse message</button>

<script>
    var app = new Vue({
        el: '#app',
        data: {
            message: 'Hello Vue.js'
        },
        methods: {
            reverseMessage: function() {
                this.message = this.message.split('').reverse().join('')
            }
        }
    })
</script>
```

Kuva 13. Vue metodien toiminta.

3.3.2 React

React on Facebookin ylläpitämä sovelluskehys. Se julkaistiin vuonna 2013 (Github, n.d.d). Reactin alkuperäinen kehittäjä on Jordan Walke. React toimii täysin JavaScriptissa.

React-sovellus muodostuu täysin komponenteista. Reactin komponenteilla on tila, jonka avulla ne voivat muistaa arvoja. Kuvassa 14 luodaan Square-niminen komponentti Reactin omasta luokasta React.component. Rakentajassa (constructor) määritellään komponentin muuttuja value lausekkeessa this.state. State eli tila kuvaa ohjelman tilaa, johon voi tallentaa arvoja, tässä esimerkissä arvona on value-niminen muuttuja. Komponentille määritellään onClick-tapahtuma, joka kuuntelee klikkausta. Klikatessa elementtiä määritetään valuen arvoksi X this.setState:n avulla. (Reactjs, n.d.b)

```
class Square extends React.Component {
    constructor(props) {
      super(props);
      this.state = {
        value: null,
      };
    }

    render() {
      return (
        <button
          className="square"
          onClick={() => this.setState({value: 'X'})}
        >
          {this.state.value}
        </button>
      );
    }
  }
```

Kuva 14. Reactissa luotu komponentti (Reactjs, n.d.b)

Komponentteja voi luoda createElement() -metodilla, joka on hyvin samantyylinen DOM:n createElement() -metodiin. Kuitenkin suurin osa React-kehittäjistä käyttää JSX:ä, joka on Reactin syntaksilaajennus JavaScriptiin. JSX kutsuu React.createElement:iä, kun se luo elementin sivulle ja palauttaa JavaScript-objektin. Kuvassa 15 muuttuja element luodaan JSX-formaatissa. JSX toimii käytännössä kuin HTML-kuvauskieli, mutta se sisältää kaiken JavaScriptin funktionaalisuuden. (Reactjs, n.d.a)

```
function formatName(user) {
    return user.firstName + ' ' + user.lastName;
}

const user = {
    firstName: 'Harper',
    lastName: 'Perez'
};

const element = (
    <h1>
        Hello, {formatName(user)}!
    </h1>
);

    ReactDOM.render(
    element,
    document.getElementById('root')
);
```

Kuva 15. JSX-formaatti (Reactjs, n.d.a)

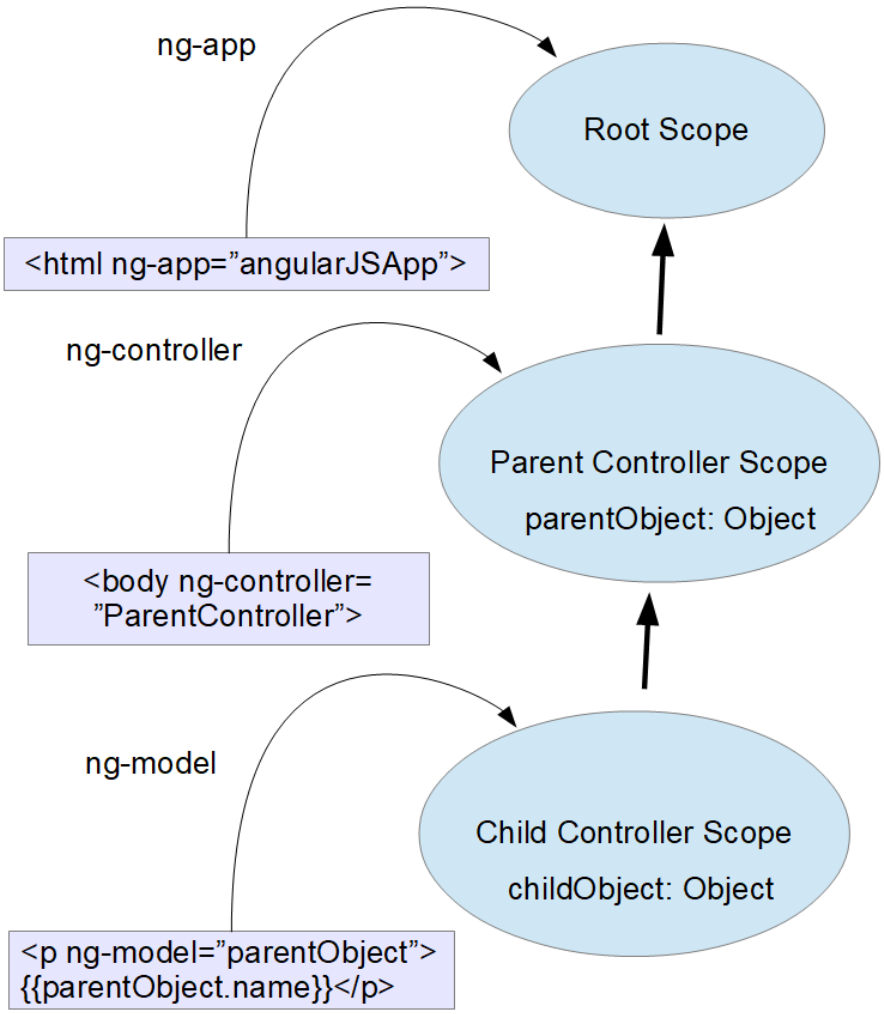
3.3.3 AngularJS

AngularJS, tai toisin kutsuttuna Angular 1, on Googlen kehittämä ja ylläpitämä sovelluskehys. AngularJS:n kehitys alkoi vuonna 2009 ja 1.0-versio valmistui vuonna 2012 (Github, n.d.b). AngularJS on kyseisen sovelluskehyksen ensimmäinen versio ja yksi ensimmäisistä sovelluskehyksistä.

AngularJS on syntaktisesti ja käsitteellisesti hyvin samankaltainen kuin Vue. AngularJS käyttää MVC-arkkitehtuuria. AngularJS:ssä näkymä näytetään mallinteen (template) avulla. Mallinne on käytännössä sivun HTML-tiedostopohja. AngularJS:ssä voi sitoa muuttujia Vuen tavoin mustache-tagilla. AngularJS:ssä on samantyyppisiä direktiivejä kuin Vuessa. AngularJS:n direktiiveissä on ng- liite. Direktiivit ovat AngularJS:ssä ainoita objekteja, jotka hakevat tietoa DOM:sta. (AngularJS Docs, n.d.)

AngularJS:n muuttujilla on näkyvyysalue samoin kuin esimerkiksi Javan muuttujilla. Näkyvyysalue voidaan käsittää laskevana puurakenteena, jossa ylhäällä olevat arvot, muuttujat ja funktiot näkyvät vain alemmille oksille. Alempana olevat objektit eivät kuitenkaan näy ylöspäin. Kuvassa 16 näkyy, miten AngularJS:ssa Root Scope on ohjelman alin näkymä. Käytännössä tämä on HTML-tiedostossa ensimmäinen tagi, eli sivun HTML-kieleksi määrittelevä <html>-tagi. Parent Controller Scope on tässä kuvassa oma näkyvyysalueensa, jossa parentObject-objekti näkyy kaikille näkymän alla oleville näkymille, tässä tapauksessa Child Controller Scopelle. Child

Controller Scopessa oleva childObject ei näy sen yläpuolella oleville näkyvyysalueille.



Kuva 16. AngularJS:n mallinteen ja mallin suhde näkyvyysalueilla (Angular Docs, n.d.)

3.3.4 Angular

Angular on julkaistu vuonna 2016 (Github, n.d.a). Sen virallinen kehittäjä on Google. Angular, tai toisin sanottuna Angular 2+, on täysin uudelleenkirjoitettu AngularJS. Angular käyttää TypeScriptia. TypeScript on tiettyä

JavaScriptin syntaksia käyttävä tyyppivarmennuskieli, eli sen avulla voidaan varmentaa muuttujien tyyppi tyypittömissä kielissä kuten JavaScriptissa. Angularilla käytetään MVC- tai MVVM-arkkitehtuuria. Tässä projektissa ei vertailla Angular 2-versiota, koska se on vanhentunut versio. Sen sijaan vertailu tehdään Angular 6 -versiolla, joka on julkaistu 3.5.2018 (Angular, n.d.b).

Angularilla kehitys tapahtuu määritellyssä työtilassa. Työtila sisältää projektin tiedostot. Angularin komponentit määrittävät näkymät. Komponentit käyttävät palveluita, jotka sisältävät näkymään liittymätöntä funktionaalisuutta, kuten datan haun tietokannasta. Angularin komponentit jaetaan HTML-, CSS- ja TypeScript-tiedostoihin. HTML-tiedostot sisältävät HTML:n, mustache-tagit ja sidonnat. CSS-tiedostot sisältävät CSS-koodin. TypeScript-tiedostoissa ladataan HTML-mallinteet ja annetaan niille funktionaalisuutta.

3.3.5 Backbone

Backbonen ensimmäinen versio julkaistiin vuonna 2010 (Backbone, n.d.). Backbone on MVC-arkkitehtuurilla toimiva sovelluskehys. Backbone käyttää MVC-konventiota kuitenkin vain löyhästi arkkitehtuurissaan. Siinä malli hoitaa datan käsittelyn lisäksi serverikutsut. Backbone mukana tulevan Underscore-kirjaston mallinne toimii näkymänä. Backbonen oma näkymä toimii ohjaimena, joka hoitaa toiminnot ja käyttöliittymän näytön. (Jindle, 2012)

Backbone hyödyntää monia JavaScript-kirjastoja sovelluskehyksen rakentamisessa. Backbone on riippuvainen Underscore.js-kirjastosta, joka rakentaa Backbonen näkymän. Tämän lisäksi jQuery hoitaa Backbonen DOM-manipulaation. (Backbone, n.d.) Käytännössä tämä tarkoittaa, että Backbone tarvitsee ainakin nämä JavaScript-kirjastot toimiakseen täydellisesti sovelluskehyksenä.

3.4 **Kehitysympäristö ja työkalut**

3.4.1 Visual Studio Code

Visual Studio Code on Microsoftin kehittämä ilmainen lähdekoodieditori. Lähdekoodieditorit antavat kyvyn muokata koodikieltä helpommin. Coden tapauksessa tämä tarkoittaa parannettua virheenetsintää eli debuggausta sekä tukea Git-versionhallinnalle, syntaksin korostukselle, automaattiselle koodin täytölle ja refraktoroinnille. Codessa on tuki usealle kymmenelle koodikielelle. Se tukee HTML-, CSS- ja JavaScript-kieliä, joita kaikkia käytetään projektissa. Visual Studio Codea hallitaan käskypaletin avulla, joka on komentorivikäyttöliittymä. (Wikipedia, n.d.e)

Visual Studio Coden ominaisuuksia voi parantaa laajennuksilla (Wikipedia, n.d.e). Tässä projektissa käytetään Polacode-laajennusta, jolla voi ottaa siistin kuvan lähdekoodista. Kaikki työn koodipätkät on otettu Polacode-laajennuksella. Lisäksi JavaScript-sovelluskehyksiin käytetään jokaisen kehyksen omaa laajennusta, joka tuo Codeen automaattisen koodin täytön kyseisille sovelluskehyksille.

3.4.2 CLI-käyttöliittymä

Osassa sovelluskehyksistä on oma Command Line Interface-käyttöliittymä. Tämä on komentorivillä toimiva käyttöliittymä sovelluskehyksen hallintaan (Wikipedia, n.d.a). Sovelluskehyksien CLI:stä voidaan luoda uusia projekteja, lisätä niihin uusia komponentteja ja hallita lisäosia. Lisäksi niillä voi suorittaa projektin selaimessa.

Vuessa on CLI-käyttöliittymä. Vuen CLI:n mukana tulee graafinen käyttöliittymä, josta voi hallita lisäosia, liitäntöjä ja kääntää projektin. (CLI Vuejs, n.d.) CLI:n avulla luodussa projektissa on mukana Babel-kääntäjä. Babel on JavaScript-kääntäjä, joka muuntaa JavaScriptin uudet ECMAScript-versiot ymmärrettäväksi vanhoille selaimille (Babeljs, n.d.).

Angular toimii vain CLI:n avulla, eli Angularin projektinhallinta hoidetaan kokonaan komentoriviltä (Angular, n.d.a). Siinä on samat toiminnallisuudet kuin Vuen CLI:ssä. Angular käyttää TypeScriptia, jota käytetään tyyppivahvistuksiin. TypeScript eroaa Babelista niin, että Babel yksinkertaisesti kääntää uudet ECMAscript-formaatit vanhaan formaattiin, jolloin sovellus toimii vanhoissa ECMAscript-ympäristöissä. TypeScriptia taas käytetään muuttujien tyyppivahvistuksiin. (Marty, 2016)

Muista sovelluskehyksistä Reactilla on oma CLI:nsä (Github, n.d.c). Siinä on samat perusominaisuudet kuin muissakin CLI:issä. Backbonella ja AngularJS:llä ei ole virallisesti tuettuja CLI-käyttöliittymiä.

4 JAVASCRIPT-SOVELLUSKEHYSTEN VERTAILU

4.1 Vertailtavat asiat

Sovelluskehyksen valinta tehdään luvussa 2 läpikäytyjen kriteerien perusteella. Sovelluskehyksistä verrataan ensin niiden mobiilitukea. Mobiilitukeen kuuluvat muun muassa natiiviapplikaatioiden kehitystuki. Sen jälkeen verrataan sovelluskehysten modulaarisuutta ja suorituskykyä. Sovelluskehykset sisältävät omia tiedostojaan ja lisäosia. Sovelluskehysten tiedostoista verrataan niiden koot. Toiset sovelluskehykset ovat suositumpia kuin toiset, jolloin niille löytyy enemmän tukea ja resursseja. Lisäksi verrataan sovelluskehysten testiautomaatiotukea.

4.1.1 Mobiilituki

Mobiilituki on tärkeä osa uutta käyttöliittymää. Mobiiliyhteensopivuutta ajateltaessa on valittava websovelluksen ja natiiviapplikaation välillä. Websovellus on selaimessa toimiva sovellus, joka on rakennettu pöytäkoneille suunnatuilla sovelluskehyksillä. Natiiviapplikaatio rakentaa applikaation käyttäen mobiililaitteen omia käyttöliittymäkomponentteja.

Natiiviapplikaation avulla saadaan otettua mobiililaitteen käyttöliittymästä kaikki funktionaalisuus irti, mutta eri käyttöliittymät eivät ole yhteensopivia toistensa kanssa. Hyvä esimerkki tästä on Applen iOS- ja Googlen Android-käyttöjärjestelmät. Lisäksi natiiviapplikaatiot ovat suorituskyvyltään tehokkaampia.

Websovellus toisaalta toimii missä tahansa tietokoneessa tai mobiililaitteessa, jossa on selain. Tällaiseksi dynaamiseksi rakennettua websovellusta kutsutaan hybridimalliksi. Hybridimallien käytön on ennustettu lisääntyvän paljon tulevina vuosina (Kotilainen, 2019).

Modernit sovelluskehykset on ensisijaisesti kehitetty pöytäkoneille sovelluskehitykseen. Kuitenkin useilla sovelluskehyksillä on omat natiiviversionsa.

Vuelle ja Angularille on julkaistu omat mobiilipohjaiset Native-versionsa. Native-versiot käyttävät Vuen ja Angularin syntaksia, mutta kääntävät sovelluksen mobiilissa toimivaksi natiiviapplikaatioksi. Vue Native ja Angular Native käyttävät NativeScriptia natiivin sovelluksen rakentamiseen. NativeScript on sovelluskehys natiivien mobiiliapplikaatioiden rakentamiseen. NativeScript hoitaa mobiiliapplikaatioksi muunnon ja webkehitykseen keskittyvät Vue ja Angular hoitavat itse käyttöliittymän kehityksen.

Muista sovelluskehyksistä React käyttää omaa React Native versiotaan, jonka syntaksi on Reactia. Reactin natiiviapplikaatio ei ole NativeScriptia käyttävä sovelluskehys, vaan se on kehitetty suoraan mobiiliyhteensopivaksi.

Backbone-sovelluskehystä käytetään usein jQuery Mobile -kirjaston kanssa rakentamaan websovelluksia. Backbonea itsessään ei ole kehitetty mobiiliyhteensopivaksi, vaan sitä käytetään jQueryn kanssa websovellusten kehittämiseen. Backbone tarjoaa sovelluskehyksen rakenteen ja funktionaalisuudet, kun taas jQuery tarjoaa kosketusnäyttö- ja mobiilitoiminnallisuudet. (JQuerymobile, n.d.) AngularJS:n mobiiliyhteensopivuus on samankaltainen Backboneen verrattuna. AngularJS käyttää myös samoin jQuery Mobilea websovellusten kehittämiseen mobiililaitteille (Mobileangularui, n.d.).

Mobiiliyhteensopivuutta katsottaessa pitää päättää, halutaanko sovellukseen websovellus vai natiiviapplikaatio. Websovellus on todennäköisesti

tulevaisuudessa suosiotaan kasvattava tapa tehdä mobiilisovelluksia. Etuna websovelluksille on myös toimivuus kaikissa käyttöjärjestelmissä ja laitteissa. Natiiviapplikaatiot taas saavat kaiken funktionaalisuuden irti mobiililaitteesta, mutta ne eivät toimi pöytäkoneilla ja epäsopivilla käyttöliittymillä. Sovelluskehykset, joilla voi kehittää natiiviapplikaatioita, käyttävät oman sovelluskehyksensä syntaksia, jolloin edellinen kokemus sovelluskehyksestä voi olla hyvä syy valita tietty sovelluskehys.

4.1.2 Modulaarisuus

Sovelluskehityksessä ohjelmat jaetaan osiin. Ohjelman osia voidaan yhdistää ja erottaa halutun määrän mukaan. Tätä kutsutaan modulaarisuudeksi. Sovelluskehityksessä on monia erilaisia tekniikoita halutun modulaarisuuden saavuttamiseksi.

React-ohjelmat suunnitellaan Single Responsibility Principlen mukaan. Sen mukaan jokainen ohjelman osa on vastuussa vain yhdestä asiasta. React-sivulla jokaisen komponentin ympärille voi visuaalisesti piirtää neliö, joka sisältää yhden React-komponentin. Komponentit jaetaan kätevästi datamallin mukaan, jolloin jokainen komponentti sisältää yhden data-arvon. Esimerkiksi sivulla oleva lista on yksi komponentti, ja jokainen listan arvo on komponentti. (Reactjs, n.d.d)

Samoin Angular käyttää Single Responsibility Principleä (Angular, n.d.c). Angular ja AngularJS käyttävät komponentteja. Angularien komponentit on jaettu useisiin tiedostoihin, jotka viittaavat toisiinsa MVC-mallin mukaan.

Vue käyttää hyvin samantyyppisiä komponentteja kuin React. Molemmissa käytetään JavaScriptia komponenttien rakentamiseen ja molempia komponenttityyppejä voidaan uudelleenkäyttää.

Backbonen löyhästi MVC-malliin perustuvassa arkkitehtuurissa näkymä toimii myös ohjaimena. Näkymä käsittelee käyttöliittymästä alun perin tulleet tapahtumat. Lisäksi malli hoitaa datan hallinnan lisäksi serverikutsut. Tällöin Backbone ei ole yhtä modulaarinen kuin muut sovelluskehykset.

4.1.3 Suorituskyky

Sovelluskehysten suorituskykyä verrattaessa on käytetty Stefan Krausen (Krause, Stefan, n.d.) tekemää suorituskykytestiä. Testi suorittaa valituilla sovelluskehyksillä operaation, kuten luo 1000 taulukkoriviä, ja mittaa, kuinka nopeasti sovelluskehys suoriutui tehtävästä.

Sovelluskehyksien elementeillä voi olla avainkenttä (keyed), joka viittaa suoraan DOM:ssa sijaitsevaan webelementtiin. Avainkentätön (non-

keyed) versio järjestää elementit omalla tavallaan, mikä tekee sovelluskehyksen toiminnasta nopeamman. Tämä saattaa kuitenkin aiheuttaa muiden JavaScript-kirjastojen viittaamaan väärin elementteihin, jolloin websivulla saatetaan näyttää vääriä arvoja. (Krause, Stefan, 2018)

Kuvassa 17 on avainkentälliset versiot sovelluskehysten operaatioista. Suurin ero on Vuen rivien vaihto-operaatiossa, jossa Vue on noin viisi kertaa nopeampi muihin sovelluskehyksiin verrattuna. Kuitenkin Vuen sivun osittainen päivitys on selkeästi hitaampi muihin sovelluskehyksiin verrattuna. AngularJS on myös selkeästi hitaampi poistamaan taulukon rivejä.

Kuvassa 18 on verrattu avainkentättömien sovelluskehysten tekemiä operaatioita. Suurimmasta osasta operaatioita sovelluskehykset toimivat samalla nopeudella, mutta suurin ero on edelleen Vuen osittaisessa sivunpäivityksessä. Lisäksi AngularJS on edelleen hitaampi poistettaessa taulukon rivejä.

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed
create rows Duration for creating 1000 rows after the page loaded.	182.1 ± 7.6 (1.0)	185.2 ± 10.2 (1.0)	180.5 ± 7.3 (1.0)	225.5 ± 10.7 (1.2)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	158.8 ± 2.7 (1.0)	161.2 ± 2.7 (1.0)	157.3 ± 2.0 (1.0)	201.4 ± 6.4 (1.3)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	156.4 ± 9.8 (2.3)	68.8 ± 3.7 (1.0)	81.9 ± 2.7 (1.2)	90.0 ± 5.8 (1.3)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	10.6 ± 2.0 (1.0)	7.9 ± 4.3 (1.0)	10.3 ± 2.1 (1.0)	9.7 ± 1.3 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	20.0 ± 2.9 (1.0)	105.8 ± 1.8 (5.3)	106.5 ± 1.9 (5.3)	106.8 ± 1.1 (5.3)
remove row Duration to remove a row. (with 5 warmup iterations).	54.2 ± 2.2 (1.2)	47.1 ± 3.0 (1.0)	49.6 ± 0.8 (1.1)	50.4 ± 0.8 (1.1)
create many rows Duration to create 10,000 rows	1,603.2 ± 34.8 (1.0)	1,693.9 ± 70.1 (1.1)	1,935.4 ± 33.6 (1.2)	2,126.4 ± 71.4 (1.3)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	342.5 ± 6.0 (1.4)	243.3 ± 6.3 (1.0)	268.6 ± 6.9 (1.1)	305.7 ± 11.6 (1.3)
clear rows Duration to clear the table filled with 10.000 rows.	191.9 ± 6.1 (1.1)	263.9 ± 3.0 (1.5)	175.4 ± 4.1 (1.0)	419.2 ± 6.9 (2.4)
slowdown geometric mean	**1.17**	**1.27**	**1.27**	**1.54**

Kuva 17. Avainkentälliset operaatiot

Name	angular-v6.1.0-non-keyed	react-v16.4.1-non-keyed	vue-v2.5.16-non-keyed
create rows Duration for creating 1000 rows after the page loaded.	177.9 ± 7.7 (1.0)	184.7 ± 7.4 (1.0)	182.0 ± 8.6 (1.0)
replace all rows Duration for updating all 1000 rows of the table (with 5 warmup iterations).	59.2 ± 2.6 (1.0)	61.4 ± 2.6 (1.0)	67.2 ± 2.1 (1.1)
partial update Time to update the text of every 10th row (with 5 warmup iterations) for a table with 10k rows.	66.4 ± 2.6 (1.0)	81.3 ± 2.4 (1.2)	169.7 ± 23.4 (2.6)
select row Duration to highlight a row in response to a click on the row. (with 5 warmup iterations).	7.0 ± 3.0 (1.0)	10.4 ± 2.3 (1.0)	11.2 ± 2.8 (1.0)
swap rows Time to swap 2 rows on a 1K table. (with 5 warmup iterations).	13.5 ± 4.4 (1.0)	14.8 ± 4.5 (1.0)	13.8 ± 1.5 (1.0)
remove row Duration to remove a row. (with 5 warmup iterations).	31.8 ± 3.3 (1.0)	37.8 ± 1.5 (1.2)	38.3 ± 1.5 (1.2)
create many rows Duration to create 10,000 rows	1,647.2 ± 47.6 (1.0)	1,945.2 ± 24.4 (1.2)	1,581.5 ± 48.4 (1.0)
append rows to large table Duration for adding 1000 rows on a table of 10,000 rows.	243.8 ± 5.1 (1.0)	267.8 ± 5.5 (1.1)	350.2 ± 13.8 (1.4)
clear rows Duration to clear the table filled with 10.000 rows.	257.4 ± 2.2 (1.5)	175.3 ± 1.6 (1.0)	192.0 ± 9.5 (1.1)
slowdown geometric mean	**1.05**	**1.09**	**1.21**

Kuva 18. Avainkentättömät operaatiot

Sovelluskehysten käynnistysoperaatioita verrataan kuvissa 19 ja 20. Ladattaessa sovelluskehyksiä Angular on selkeästi hitain. Sekä avainkentällisessä että -kentättömässä testissä Angular oli vähintään kaksi kertaa hitaampi lataamaan sovelluskehyksen kaikki tiedostot. Lisäksi Angular häviää muille sovelluskehyksille myös muissa käynnistysoperaatioissa. AngularJS oli

myös selkeästi hitaampi Vueen ja Reactiin nähden. Vue oli nopein käynnistysoperaatioissa.

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	2,252.7 ± 0.2 (1.0)	3,278.5 ± 4.0 (1.5)	2,477.6 ± 0.6 (1.1)	2,854.0 ± 0.7 (1.3)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	55.1 ± 1.5 (1.0)	235.2 ± 8.5 (4.3)	65.6 ± 2.3 (1.2)	124.6 ± 2.0 (2.3)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	420.6 ± 67.5 (1.0)	679.3 ± 5.3 (1.6)	466.0 ± 3.9 (1.1)	530.6 ± 2.5 (1.3)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	215,445.0 ± 0.0 (1.0)	365,497.0 ± 0.0 (1.7)	251,915.0 ± 0.0 (1.2)	328,568.0 ± 0.0 (1.5)

Kuva 19. Avainkentälliset käynnistysoperaatiot

Name	angular-v6.1.0-non-keyed	react-v16.4.1-non-keyed	vue-v2.5.16-non-keyed
consistently interactive a pessimistic TTI - when the CPU and network are both definitely very idle. (no more CPU tasks over 50ms)	3,273.2 ± 5.4 (1.5)	2,477.4 ± 0.2 (1.1)	2,252.7 ± 0.2 (1.0)
script bootup time the total ms required to parse/compile/evaluate all the page's scripts	227.6 ± 9.7 (4.0)	65.9 ± 1.1 (1.2)	56.9 ± 0.8 (1.0)
main thread work cost total amount of time spent doing work on the main thread. includes style/layout/etc.	673.3 ± 13.1 (1.6)	431.3 ± 65.2 (1.0)	456.3 ± 1.5 (1.1)
total byte weight network transfer cost (post-compression) of all the resources loaded into the page.	365,492.0 ± 0.0 (1.7)	251,926.0 ± 0.0 (1.2)	215,440.0 ± 0.0 (1.0)

Kuva 20. Avainkentättömät käynnistysoperaatiot

Kuvissa 21 ja 22 verrataan sovelluskehysten muistien käyttöä. Angular käyttää selkeästi eniten muistia ladattaessa sivua ja luodessa taulukkorivejä. AngularJS käyttää myös enemmän muistia kuin React ja Vue. Vue pärjää parhaiten muistin käytössä.

Name	vue-v2.5.16-keyed	angular-v6.1.0-keyed	react-v16.4.1-keyed	angularjs-v1.7.2-keyed
ready memory Memory usage after page load.	2.7 ± 0.2 (1.0)	6.0 ± 0.0 (2.2)	2.8 ± 0.2 (1.0)	3.4 ± 0.2 (1.2)
run memory Memory usage after adding 1000 rows.	7.1 ± 0.0 (1.1)	9.8 ± 0.0 (1.5)	6.7 ± 0.0 (1.0)	10.9 ± 0.0 (1.6)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	7.2 ± 0.0 (1.0)	9.8 ± 0.0 (1.4)	7.6 ± 0.0 (1.1)	10.9 ± 0.0 (1.5)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	7.2 ± 0.0 (1.0)	10.1 ± 0.0 (1.4)	7.9 ± 0.0 (1.1)	11.4 ± 0.1 (1.6)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	3.0 ± 0.0 (1.0)	6.4 ± 0.0 (2.1)	3.8 ± 0.0 (1.3)	3.9 ± 0.0 (1.3)

Kuva 21. Avainkentälliset muistiallokaatiot

Name	angular-v6.1.0-non-keyed	react-v16.4.1-non-keyed	vue-v2.5.16-non-keyed
ready memory Memory usage after page load.	6.0 ± 0.1 (2.2)	2.8 ± 0.2 (1.0)	2.7 ± 0.2 (1.0)
run memory Memory usage after adding 1000 rows.	9.8 ± 0.0 (1.5)	6.7 ± 0.0 (1.0)	7.1 ± 0.0 (1.1)
update eatch 10th row for 1k rows (5 cycles) Memory usage after clicking update every 10th row 5 times	9.8 ± 0.0 (1.4)	7.6 ± 0.0 (1.1)	7.1 ± 0.0 (1.0)
replace 1k rows (5 cycles) Memory usage after clicking create 1000 rows 5 times	9.8 ± 0.0 (1.4)	10.8 ± 0.1 (1.5)	7.1 ± 0.0 (1.0)
creating/clearing 1k rows (5 cycles) Memory usage after creating and clearing 1000 rows 5 times	6.4 ± 0.0 (2.1)	3.8 ± 0.0 (1.3)	3.0 ± 0.0 (1.0)

Kuva 22. Avainkentättömät muistiallokaatiot

Mitatuissa metriikoissa ja sovelluskehyksissä keskimäärin parhaiten pärjäsivät React ja Vue. Vue on muihin sovelluskehyksiin verrattuna hitaampi vain päivitettäessä osaa sivusta, mutta se on nopea kaikissa muissa metriikoissa. React on nopea tai vähintään keskinopea muihin sovelluskehyksiin verrattuna kaikissa metriikoissa. Angular ja AngularJS vastasivat toisiaan nopeudeltaan. AngularJS:ssä nopeasti ja hitaasti toimivat testit toimivat vastaavasti samalla tavoin Angularissa, mutta vahvistuneena. Angularin molemmat versiot toimivat nopeasti muutettaessa sivun sisältöä, mutta olivat hyvin hitaita sivun latausvaiheessa ja ne käyttivät paljon muistia.

4.1.4 Sovelluskehysten koko

Sovelluskehykset ovat loppujen lopuksi JavaScriptilla ohjelmoituja rakenteita. Niiden tiedostojen koko vaihtelee. Tässä projektissa sovelluskehyksen koko vaikuttaa käyttöliittymän latausnopeuteen. Myös niiden mukana tulevat JavaScript-kirjastot ovat erilaisia. Eri lisäosat tuovat eri hyötyjä projektiin. Lisäosien määrä voi kuitenkin tehdä projektin vaikeammin ymmärrettävämmäksi tai hitaammaksi. JavaScript minified -tiedostojen koko käydään läpi taulukossa 1.

Taulukko 1. Sovelluskehysten tiedostojen koko (Github, n.d.e)

Sovelluskehykset	Mittaus 2019	Lähde 2017
Vue	91KB	63KB
React	133KB	144KB
AngularJS	173KB	143KB
Backbone	23KB	-

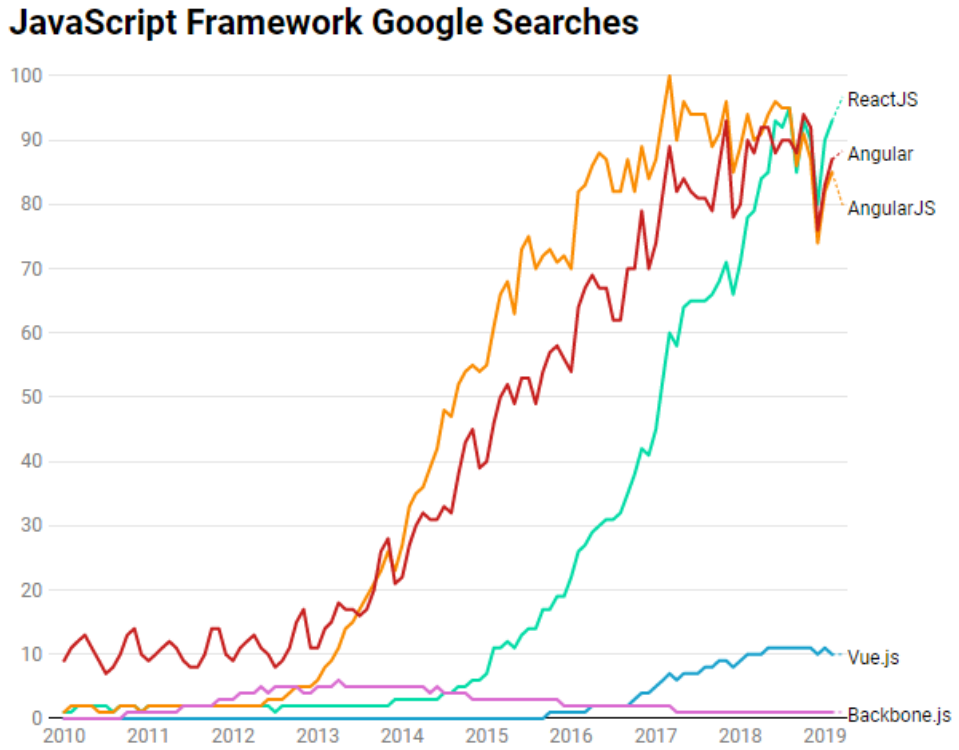
Arvot mitattiin lataamalla minified-kokoiset tiedostot sovelluskehysten nettisivuilta sekä käyttämällä lähdettä. Angular on tällä hetkellä pelkästään CLI-käyttöliittymällä toimiva sovelluskehys, joten sitä ei verrata tässä. Lähteessä ei ollut tutkittu Backbone-kirjastoa ollenkaan.

Vertailluista kirjastoista Backbone on selvästi kevyin. Vue on toiseksi ja React kolmanneksi kevyin. AngularJS on kevyistä asennuksista suurin. Verrattuna vuoteen 2017 Vuen koko on kasvanut noin 30KB. React on keventynyt hieman, kun taas AngularJS:n koko on kasvanut.

4.1.5 Yleisyys

Sovelluskehyksiä takaavat usein isot IT-yritykset. Esimerkiksi React on Facebookin kehittämä ja ylläpitämä. Lisäksi sovelluskehyksen ympärillä työskentelevä yhteisö kehittää resursseja sovelluskehykseen. Tämän otsikon alla vertaillaan sovelluskehysten suosiota ja sitä, minkälaista tukea niille on saatavilla.

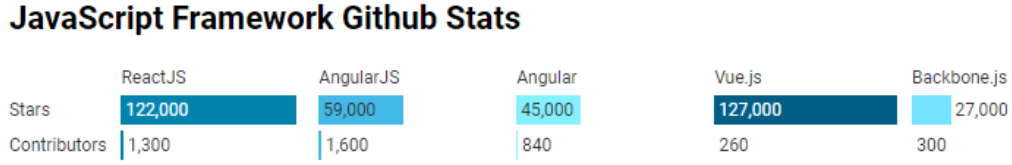
Kuvassa 23 on vertailtu Google Trendsistä saatuja hakumääriä kullekin sovelluskehykselle. Sovelluskehyksistä haetuimmat ovat React, AngularJS ja Angular. Vuella on uusimpana sovelluskehyksenä suhteellisesti vähän hakuja. Backbonen hakumäärä on marginaalinen. Mahdollisesti kuitenkin AngularJS on voinut saada enemmän hakuja, koska se on lähes samanniminen Angularin kanssa.



Kuva 23. Googlessa haetuimmat JavaScript-sovelluskehykset (Google, 2019)

Kuvassa 24 kerätyt tiedot ovat kunkin sovelluskehyksen Github-sivulta. Githubissa arvostetuimmille sovelluskehyksille annetaan tähtiä. Suosituimmat ovat Vue 127 000 ja React 122 000 tähdellä. Angularin molemmilla versioilla on alle puolet tästä, ja Backbonella on vain 27 000 tähteä.

AngularJS:llä on eniten sovelluskehystä vapaaehtoisesti kehittäneitä henkilöitä eli edistäjiä, noin 1600. Tämä johtuu todennäköisesti siitä, että AngularJS on näistä sovelluskehyksistä vanhin, joten sillä on ehtinyt olla eniten kehittäjiä. Reactilla ja Angularilla on myös paljon kehittäjiä, joka johtuu osittain Facebookin ja Googlen suurista kehitystiimeistä. Vue ja Backbone ovat lähtökohdiltaan yhden henkilön kehittämiä, joten niillä ei ole yhtä paljon edistäjiä.



Kuva 24. JavaScript-sovelluskehyksien tilastotiedot Githubissa (Github, n.d.f)

Taulukossa 2 on arvioitu sovelluskehysten keskimääräisiä viikoittaisia npm-latauksia ajalta helmikuu 2018 – helmikuu 2019. Latausmäärät haettiin kyseisien sovelluskehysten npm-sivulta. Ohjelmistorekisteri npm:stä suosituin sovelluskehys on React 3 miljoonalla viikoittaisella latauksella. Angularilla on alle puolet sen npm-latauksista. Vuella on noin puoli miljoonaa viikoittaista latausta, kun taas Backbonella on noin 250 000 latausta viikossa.

Taulukko 2. Sovelluskehysten keskimääräiset viikoittaiset lataukset

Vue	500 000 latausta
React	3 miljoona latausta
Angular	1,2 miljoonaa latausta
Backbone	250 tuhatta latausta

Taulukossa 3 on verrattu löydettyjä työpaikkoja suosituista suomalaisista työpaikkahakusivustoista. Työpaikkasivustoista tehtiin haku sovelluskehyksen nimellä ja katsottiin, kuinka monta työpaikkaa sivustoilla oli. Haettuja sivustoja olivat Duunitori, Monster, TE-palvelut, Oikotie-työpaikat ja Rekrytointi. Reactilla oli eniten avoimia työpaikkoja. Angularilla oli myös lähes sata tämän hetkistä työpaikkaa auki. Vuella ja Angularilla oli noin 35 paikkaa kummallakin. Backbonella oli viisi työpaikkaa avoinna.

Taulukko 3. Työpaikat suosituista työpaikkahakusivustoista 19.2.2019

Vue	34
React	170
Angular	97
AngularJS	35
Backbone	5

4.1.6 Testiautomaatio

Websovellukset testataan bugien ja ei-halutun toiminnan varalta. Sovelluksia kehitettäessä testausprosessi halutaan automatisoida, koska uusia versioita sovelluksesta voidaan tehdä nopeasti. Tällöin aikaa kuluisi paljon pelkästään testaukseen.

Projektissa tärkeimpinä testattavina asioina ovat yksikkötestaus ja käyttöliittymätestaus. Yksikkötestaus tarkoittaa sovelluksen osan tai osien testausta käyttäen dataa ja osien funktionaalisuuksia. Käyttöliittymätestauksessa testataan itse käyttöliittymää, esimerkiksi painetaan käyttöliittymässä olevaa näppäintä ja varmistetaan, että käyttöliittymä päivittyi oikein. Sovelluskehyksistä tulisi löytyä tuki ainakin näiden tyyppisille testauksille.

Vue Test Utils on Vuen virallisesti tukema testikirjasto. Vue Test Utilsin avulla voidaan liittää haluttu Vue-komponentteja sisältävä JavaScript-tiedosto kirjastoon. Testaus-kirjaston avulla voidaan sitten testata Vue-komponentin arvoja, syötteitä ja tulosteita. Vue Test Utils voi etsiä websivulta HTML-elementtejä ja testata niiden olemassaoloa, interaktiivisuutta tai arvoja. (Vue test utils, n.d.) Lisäksi Vue tukee Jest- ja Mocha-testikirjastoja, joita käytetään yksikkötestaamiseen (Vuejs, n.d.d).

React Test Utils on samantyyppinen testikirjasto kuin Vue Test Utils. Niillä on samat funktionaalisuudet ja ne eroavat käytännössä vain syntaksiltaan. Facebook suosittelee Jest-testisovelluskehystä käytettäväksi Test Utilsin rinnalla. (Reactjs, n.d.c)

Angularia kehittävä tiimi on kehittänyt Karman, joka on selaimessa toimiva testikirjasto. Käytettäväksi sovelluskehykseksi Angular suosittelee Jasmine-testisovelluskehystä. Jasmine kykenee yksikkötestaukseen, jonka lisäksi se pystyy näyttämään testausraportit webselaimessa. (Angular, n.d.d)

Backbone ei suosittele tai tue virallisesti mitään testikirjastoa, mutta usein Backbone-sovelluksien testaukseen suositellaan Mochaa tai jQueryn testauskirjastoa QUnitia. Mocha ja QUnit kykenevät yksikkötestaukseen.

4.2 **AngularJS ja Angular**

Angular kehitettiin AngularJS:n pohjalta. Siksi Angularissa on tiettyjä muutoksia AngularJS:ään verrattuna. Ensimmäinen suuri ero on TypeScriptin käyttö. AngularJS:n käsitteet näkyvyysalue ja ohjain eivät ole olemassa Angularissa. Sen sijaan Angular käyttää hierarkkisesti järjesteltyjä komponentteja. Komponentti Angularissa on direktiivi, jolla on mallinne. Angular on näin hyvin samanlainen tässä tapauksessa Reactiin ja Vueen. Sekä Angular että AngularJS ovat avoimeen lähdekoodiin pohjautuvia sovelluskehyksiä.

Syntaktisesti AngularJS:ssä kaksisuuntainen sidonta luodaan ng-model -direktiivillä ja yksisuuntainen sidonta ng-bind -direktiivillä. Angular käyttää pelkästään ngModel -direktiiviä, mutta sen voi valita yksi- tai kaksisuuntaiseksi sidonnaksi joko [] tai [()] -merkkien sisään kirjoitettuna. Sovelluskehysten syntaksissa on myös pienempiä muutoksia, kuten AngularJS:n direktiivi ng-repeat on korvattu Angularissa koodillisesti tutummalla ngFor-direktiivillä. (Dziwoki, 2019)

Angular on modulaarisempi. Suuri osa runkofunktionaalisuudesta on siirretty eri moduuleihin. Tämän takia Angular on nopeampi ja kevyempi ydinfunktionaalisuudeltaan. Tämän lisäksi Angularilla on oma CLI-käyttöliittymänsä helpottamaan projektinhallintaa. (Dziwoki, 2019)

4.3 Valittu sovelluskehys

Mobiilitukea ja modulaarisuutta verrattaessa modernit sovelluskehykset ovat selvästi parempia. Näitä ovat React, Vue ja Angular, joilla on tuki omaan mobiiliyhteensopivaan Native-versioonsa. Nämä sovelluskehykset ovat hyvin modulaarisia, sillä niiden perustoiminnallisuudet on jaettu pakkauksiin ja ne käyttävät modulaarisia malleja eli komponentteja. Samoin modernit sovelluskehykset integroivat testiautomaation suoraan sovelluskehyksen tukemiin Test Utils -kirjastoihin. Vanhemmat sovelluskehykset AngularJS ja Backbone taas eivät sisällytä mobiilitukea ja testiautomaatiota suoraan sovelluskehyksiin, vaan ne ratkaisevat nämä muiden JavaScript-kirjastojen avulla. Vanhemmat sovelluskehykset eivät myöskään ole yhtä modulaarisia.

Suorituskyvyssä parhaiten pärjää Vue. React on kuitenkin hyvin lähellä Vuen suorituskykyä. Angularit taas eivät pärjää yhtä hyvin, erityisesti verrattaessa käynnistysoperaatioita ja muistin käyttöä. Sovelluskehyksien ko'oista pienin on Backbonella. Seuraavana järjestyksessä ovat Vue, React ja Angular.

Näillä metriikoilla tutkittuina ei saada vielä tehtyä lopullista valintaa. Esimerkiksi suorituskyky on tärkeä metriikka sovelluskehyksen valinnalle, mutta kaikista tärkein on sovelluskehyksen suosio. Suosituimmilla sovelluskehyksillä on enemmän resursseja, kuten dokumentointia ja tutoriaaleja, ja osaavia työntekijöitä. Käyttöliittymän tulevaisuutta mietittäessä halutaan sillä olevan osaavia kehittäjiä vielä vuosienkin päästä.

React on sovelluskehyksistä selvästi suosituin. Se on haetuin ja ladatuin sovelluskehys. React on Githubissa Vuen ohella arvostetuimpia sovelluskehyksiä. Sillä on myös eniten avoimia työpaikkoja. Angularit ovat seuraavaksi suosituimpia sekä latauksissa, työpaikoissa että hauissa mitattuina. Kuitenkin Angularilla kehitystä tehneet ihmiset eivät pitäneet niistä yhtä paljon. Githubin tähtien määrää katsottaessa ne eivät ole yhtä arvostettuja. Vue on uusimpana sovelluskehyksenä suhteellisen tuntematon ja vähän käytetty verrattuna Reactiin ja Angulareihin, mutta sitä käyttäneet kehittäjät arvostivat sen helppokäyttöisyyttä. Backbone on sovelluskehyksistä vähiten käytetty.

Sovelluskehyksistä valitaan React. Tärkein kriteeri valinnalle on sen suosio, mutta React on myös muilla metriikoilla vertailukelpoinen sovelluskehys. React käyttää virtuaalista DOM-ympäristöä, joka on moderneissa sovelluskehyksissä käytetty tekniikka. Virtuaalinen DOM-ympäristö takaa sen, että Reactin suorituskyky on hyvä. React käyttää sovellusten rakentamiseen komponentteja, jotka ovat moderni tapa kehittää websovelluksia sovelluskehyksillä. Reactilla on suuri käyttäjäkunta ja kehittäjätiimi, jotka tarjoavat hyvän dokumentaation ja tuen React-kehitykselle. Reactin tulevaisuus näyttää vankalta, sillä se on suosituin webkehitykseen käytetty sovelluskehys.

5 KÄYTTÖLIITTYMÄN SUUNNITTELU

Tässä luvussa käydään läpi käyttöliittymän suunnitteluun liittyviä asioita. Lopullinen tavoite on rakentaa prototyyppimalli uudesta käyttöliittymästä Reactilla.

5.1 Sisältö

Käyttöliittymässä on oltava ainakin neljä sivua.

- Etusivu
- Hälytys-sivu
- Hälytyshistoria-sivu
- Asetukset-sivu

Etusivulle sijoitetaan nostolaitteen yksilöiviä tietoja kuten sen ID. Lisäksi etusivulla näytetään perustietoja nostolaitteen tilasta sekä kuva nosturista.

Hälytys-sivut näyttävät nostolaitteen virhetiloja. Hälytyksillä on oma luokkansa, joka voi olla vika, hälytys tai tapahtuma. Lisäksi niillä on virhetapahtuman kuvaus ja muita tietoja. Hälytys-sivulla näytetään aktiiviset hälytykset. Niitä voi klikata, jolloin näytetään lisätietoja hälytyksestä. Hälytyshistoria-sivulla näytetään menneet hälytykset.

Asetukset-sivulla voi muuttaa nostolaitteen parametreja. Prototyyppiin ohjelmoidaan yksi nostolaitteen parametri, jota voidaan muuttaa ja päivittää dynaamisesti.

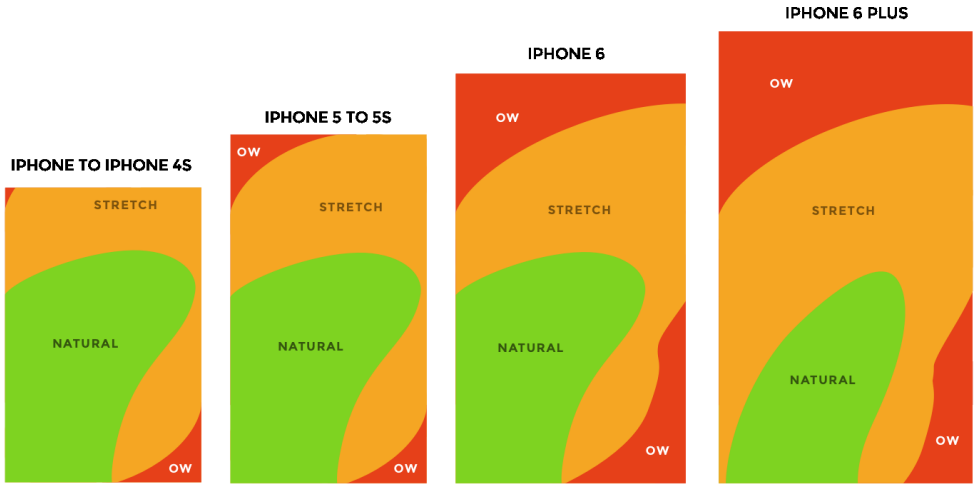
5.2 Käyttökokemus

Käyttöliittymässä käytetään Konecranesin ulkoasua. Esimerkiksi sivun värit asetetaan Konecranesin tyylin mukaan. Kuitenkin sivun suunnittelussa voidaan miettiä erilaisia ratkaisuja. Mobiilikäyttöliittymissä käytetään usein sivun vasemmasta yläkulmasta aukeavaa hampurilaisvalikkoa.

Hampurilaisvalikkoja voi suunnitella eri tavoin. Usein valikko sijoitetaan vasempaan yläkulmaan, josta sitä painettaessa avautuu navigointivalikko. Valikko voi avautua vasemmalta oikealle tai ylhäältä alas. Etuna vasempaan yläkulmaan sijoittamisessa on tunnettu sijainti valikolle ja selkeä käyttöliittymä.

Kuvassa 25 näytetään, miten peukalo sijoittuu älypuhelimen näytölle. Jos valikko sijoitetaan sivun ylälaitaan, joutuu käyttäjä käyttämään kahta kättä sivun navigointiin. Valikko voidaan sijoittaa näytön alakulmaan, jolloin saa-

daan aikaan parempi käyttökokemus. Käyttäjä tarvitsee näin vain älypuhelinta pitävän käden käyttöliittymässä navigoimiseen. Haittapuolena tässä on kuitenkin se, että käyttäjät ovat tottuneet siihen, että valikko sijaitsee sivun ylälaidassa. Sivua on näin vaikeampi hahmottaa, kun sivun sisältöalue sijaitsee sivun ylälaidassa. Lisäksi Konecranesin muut käyttöliittymät on rakennettu yläkulmassa sijaitsevilla valikoilla, jolloin alalaitaan sijoitettu valikko muuttaisi tuttuja käytäntöjä liian paljon.



Kuva 25. Peukalon sijoittuminen älypuhelimen näytölle (Hurff, n.d.)

5.3 Muut suunniteltavat asiat

Käyttöliittymästä tehdään hybridimalli. Tämä saadaan aikaan Bootstrap-kirjastolla, joka on websivun tyylin muokkausta helpottava JavaScript-kirjasto. Bootstrap-kirjastoa käyttää lähes 20 % kaikista websivuista (W3Techs, 2019). Kirjasto jakaa sivun 12 osaan, joiden avulla sivulle voidaan ryhmitellä elementit halutulla tavalla. Lisäksi kirjastolla voidaan muokata HTML-elementtien tyyliä sen omilla CSS-luokilla.

Muita käytettäviä kirjastoja on Bootstrap-kirjastossa käytettävä Popper-kirjasto, jonka avulla voidaan helposti sijoittaa vihjelaatikoita. Tämän lisäksi käytetään jQuery-kirjastoa. Se antaa suuren määrän funktionaalisuuksia websivun muokkaamiseen JavaScriptissä.

Yksikkötestaukseen React suosittelee React Test Utilsin käyttöä Jest-testauskirjaston kanssa. Jest kykenee myös tilannekatsaustestaukseen, jossa varmistetaan käyttöliittymän tila verrattuna aiemmin tallennettuun tilakatsaukseen. Konecranesin nykyinen käyttöliittymätestaus toteutetaan Katalon-testaussovelluksella. Käyttöliittymätestaukseen voidaan jatkaa Katalonin käyttöä.

6 KÄYTTÖLIITTYMÄN KEHITYS

Työn käytännön tavoitteena on rakentaa prototyyppi käyttöliittymästä. Se rakennetaan Reactilla. Käyttöliittymään tulee luvussa 5 läpikäydyt asiat.

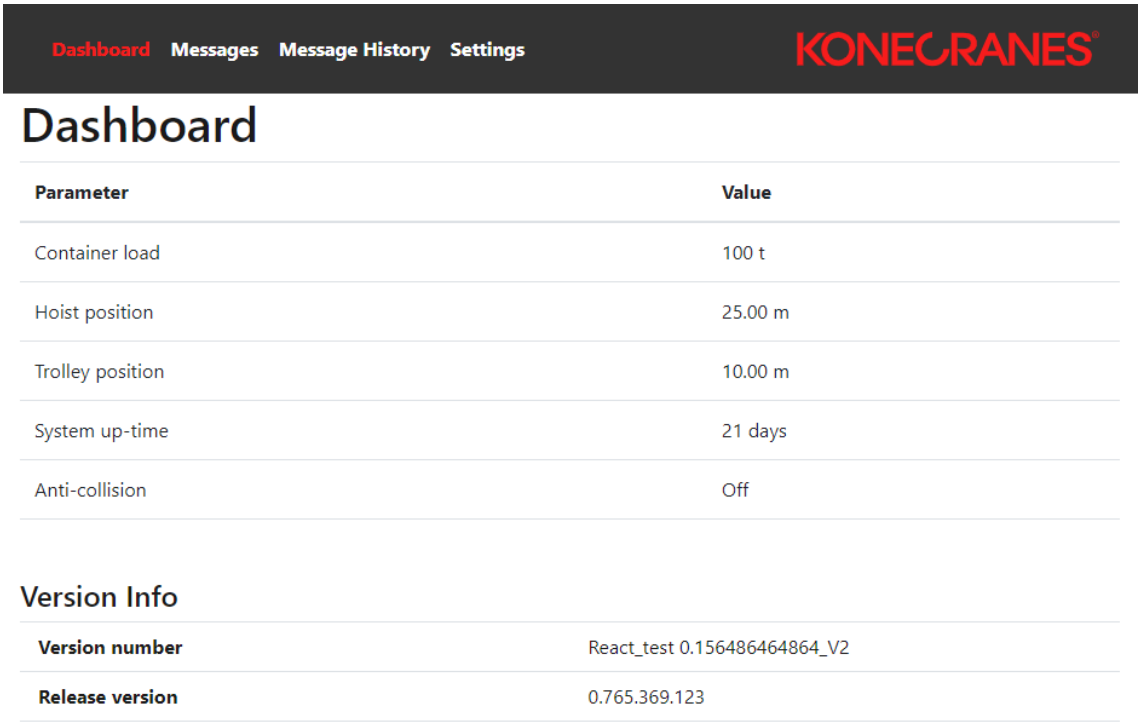
6.1 React ja riippuvuudet

Reactia käytettäessä suositellaan luomaan sovelluksen pohja npm-pakkauksella create-react-app, joka luo React-sovelluksen ja siihen liittyvät riippuvuudet automaattisesti. Kuitenkin tässä projektissa ei tarvittu läheskään kaikkia create-react-appin mukana tulevia riippuvuuksia, joten turhien toiminnallisuuksien poistamiseksi asennus tehtiin manuaalisesti.

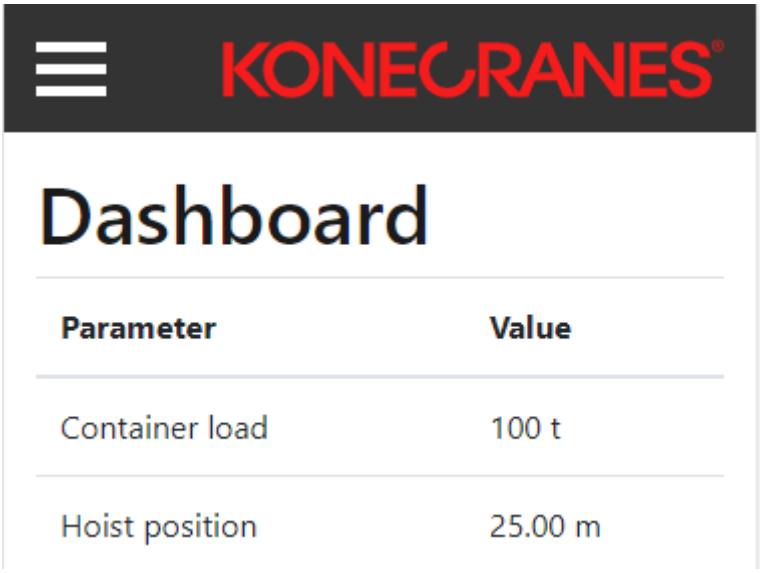
Tärkeimpinä riippuvuuksina asennettiin itse React ja sen tärkeimmät osat react-dom ja react-router-dom, joita käytetään DOM:in manipuloimiseen ja sivujen navigointiin vastaavasti. Visuaalisina riippuvuuksina lisättiin Bootstrap, Popper ja jQuere. Tärkeinä taustalla toimivina ja avustavina riippuvuuksina ovat Babel, Webpack ja erilaisten tiedostojen latauskirjastot, kuten css-loader, joka nimensä mukaisesti antaa toiminnallisuuden ladata css-tiedostoja JavaScript-koodissa.

6.2 Sivun pohja

Käyttöliittymän pohja muodostuu kahdesta osasta: header- ja content -osasta. Header-osa on sivujen navigoimiseen käytettävä yläpalkki. Se muuttuu dynaamisesti hampurilaisvalikoksi mobiililaitteita käytettäessä. Content-osa on itse sivun sisältö, josta löytyy muun muassa nostolaitteen parametrejä. Kuvassa 26 näkyy käyttöliittymän etusivu, jossa näkyy tekoparametrejä. Kuvassa 27 on saman sivun mobiiliversio. Bootstrap-kirjaston avulla muutetaan käyttöliittymän näkymää näyttökoon mukaan. Käytännössä tämä tapahtuu Bootstrapin CSS-luokkien avulla.



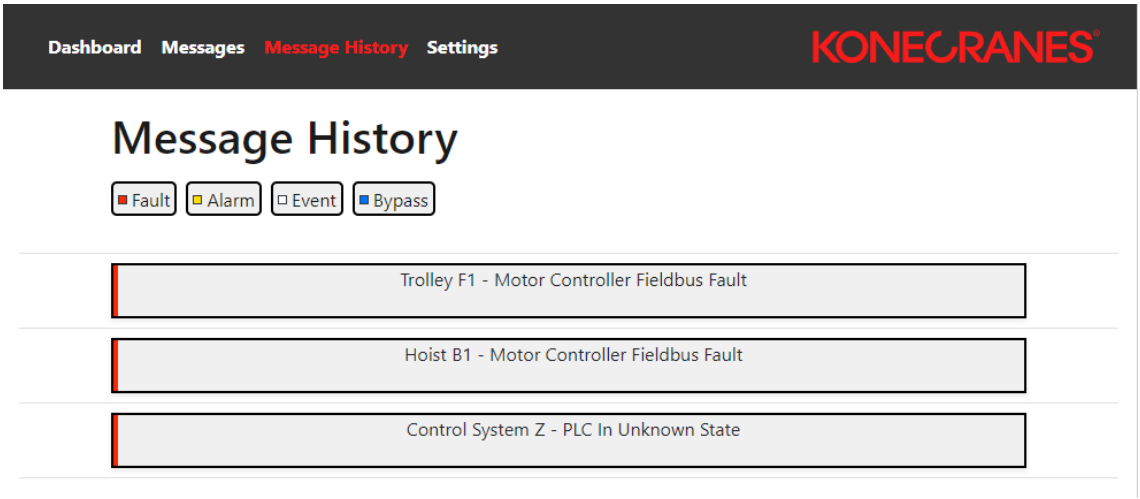
Kuva 26. Käyttöliittymän etusivu



Kuva 27. Mobiiliversio käyttöliittymän etusivusta

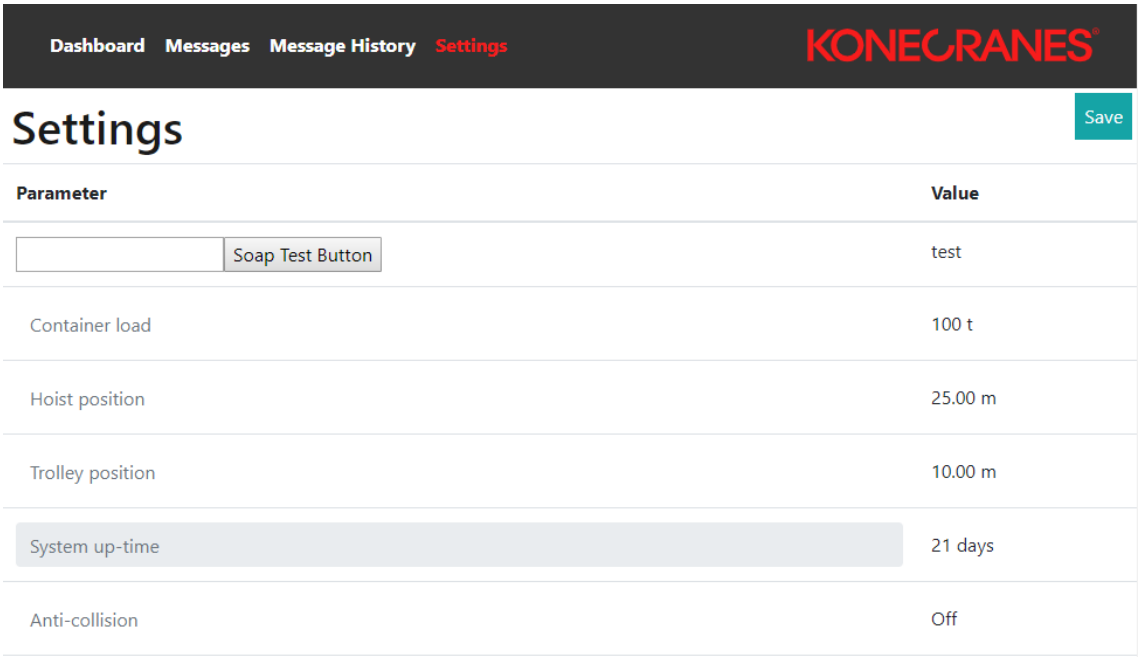
6.3 Navigoitavat sivut

Hälytys- ja hälytyshistoria-sivuilla on aktiiviset ja menneet hälytykset vastaavasti. Kuvassa 28 näkyy hälytyksien tyyppiä vastaavat värit otsikon alla. Kuvan alaosassa on kolme vikahälytystä.



Kuva 28. Hälytyshistoria-sivu

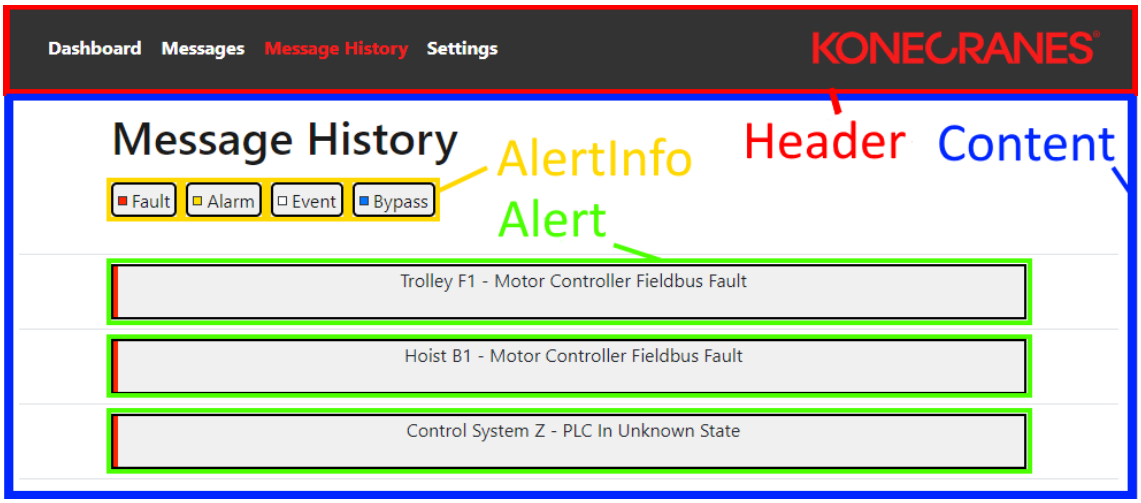
Kuvassa 29 asetukset-sivulla on tekoparametrejä ja -valikoita. Lisäksi sivulla voi kirjoittaa Soap Test Write -näppäimellä tekstikentän sisältö nostolaitteen yhdelle parametrille.



Kuva 29. Asetukset-sivu

6.4 React-komponentit

React-projekteissa sivun sisältö jaetaan komponentteihin. Kuvassa 30 on jaettu hälytyshistoria-sivu komponentteihin. Header- ja content -osiot määrittävät sivun osiot. Alert-komponentit ovat hälytyksiä, jotka hakevat hälytyksen tiedot nostolaitteelta. AlertInfo-komponentit näyttävät hälytyksien värejä vastaavat tyypit. Fyysisesti projektissa jokainen komponentti on sijoitettu omaan tiedostoonsa.



Kuva 30. Sivun React-komponentit

6.5 Nostolaitteen rajapinnan kutsuminen

Nostolaitteen parametreja kutsutaan XMLHttpRequest-kutsulla. Kuvassa 31 on koodiesimerkki nostolaitteen parametrin kutsumisesta. Ensin XMLHttpRequest-olio luodaan muuttujaksi. Pyyntöön lisätään siihen tarvittavat otsikot setRequestHeader-metodeissa. Pyyntö avataan open-metodilla ja lähetetään send-metodilla. Pyynnön loppuessa siirrytään onreadystatechange-funktioon, jonka sisällä tarkistetaan, palautettiinko pyyntö hyväksytysti. Tämän jälkeen pyyntö parsitaan DOMParser-objektin avulla, joka pystyy parsimaan XML-objekteja. Lopuksi haettu tai kirjoitettu arvo tallennetaan Reactin tilaan.

```
// The POST request
var xmlhttp = new XMLHttpRequest();

xmlhttp.onreadystatechange = function () {
    if (xmlhttp.readyState == 4 && xmlhttp.status == 200) {
        var response = xmlhttp.responseText

        var xmlParser = new DOMParser();
        var xmlDoc = xmlParser.parseFromString(response,"text/xml");

        var value = xmlDoc.getElementsByTagName(value)[0].childNodes[0].nodeValue;
        this.setState({ SOAPReturn: value })
    }
}.bind(this);

xmlhttp.open('POST', IP-address, true);
xmlhttp.setRequestHeader('Content-Type', 'application/soap+xml');
xmlhttp.setRequestHeader("SOAPAction", SOAPaction);
xmlhttp.send(envelope);
```

Kuva 31. XMLHttpRequest pyyntö JavaScriptissa

7 LOPPUTULOS JA JATKOKEHITYS

Opinnäytetyön tavoitteina oli verrata erilaisia sovelluskehyksiä, valita niistä sopivin ja rakentaa prototyyppi käyttäen tätä sovelluskehystä. Kaikki tavoitteet saavutettiin. Verrattavia sovelluskehyksiä olivat React, Vue, Angular, AngularJS ja Backbone. Sovelluskehyksiä verrattiin erilaisilla metriikoilla ja vertailun avulla saatiin valittua sopivin sovelluskehys. Vertailtavien asioiden perusteella paras sovelluskehys Konecranesin käyttöliittymään oli React-sovelluskehys. Vertailtaessa sovelluskehyksiä parannettavana asiana olisin voinut vielä vertailla lisää eri tekijöitä, kuten esimerkiksi syntaksin ja projektin rakenteen selkeyttä ja tukea erilaisille JavaScript-kirjastoille.

Käytännön osuudessa päästiin tavoitteisiin, eli käyttöliittymästä rakennettiin toimiva prototyyppi Reactilla. Yhtenä isona ongelmana kehitystyössä oli erilaisten riippuvuuksien versioyhteensopivuus. Webkehityksessä käytettävät työkalut kehittyvät nopeaa vauhtia, jolloin eri riippuvuuksien yhteensopivuus ei ole aina kovin selvä. Monet vain alle vuoden vanhat React-tutoriaalit olivat vanhentuneet. Lisäksi Reactin npm-paketin nimitys oli muuttunut uusimmissa versioissa, mikä vaikeutti pakkauksien yhteensopivuuksien ymmärtämistä entisestään. Loppujen lopuksi oikeat versiot jokaisesta riippuvuudesta kuitenkin löytyivät.

Yksi tärkeistä saavutetuista tavoitteista oli yhteensopivuus mobiililaitteisiin. Prototyyppikäyttöliittymä skaalautuu mobiililaitteille. Käyttöliittymää on helppo kehittää sekä pöytäkoneille että mobiililaitteille, sillä kahden eri version kehittämisen sijaan muutetaan käyttöliittymässä vain muutamia

tyyliluokkia. Kehitysaika paranee prototyyppikäyttöliittymässä teoriassa näin puoleen entisestään.

Kehitystyössä olisin voinut parantaa JavaScript-kirjastojen hyödyntämistä ja käyttöä heti alussa. Projektin alussa en hyödyntänyt Bootstrap-kirjastoa täysin, koska olin tottunut tekemään sivun tyylin CSS-tyyliohjeilla. Myöhemmin kehitystyössä kuitenkin tajusin voivani hyödyntää kirjastoa enemmän, jolloin muutin osan sivun tyyleistä käyttämään Bootstrap-luokkia. Tähän meni kuitenkin aikaa hukkaan, sillä sama työ tuli tehtyä kahteen kertaan.

Jatkokehitettäessä käyttöliittymää voidaan etsiä parannuksia käyttäjille, eli käytännössä nostolaitetta huoltaville henkilöille. Käyttöliittymän käyttökokemusta voisi parantaa suunnittelemalla sivujen sisällön asettelu paremmin. Käyttöliittymän lopullisessa versiossa parametrien asettamiselle laitettiin erilaisia vaihtoehtoja, kuten esimerkiksi käyttäjän täyttämä tekstikenttä tai lista vaihtoehdoista. Käyttöliittymän ulkoasussa käytettiin Konecranesin värejä ja tyylejä, mutta suunniteltaessa itse käyttöliittymää voidaan miettiä erilaisia vaihtoehtoja.

Työssä opin paljon sovelluskehyksien rakenteesta, toiminnasta ja tarkoituksesta. Erityisesti sovelluskehyksistä opin, miten ne auttavat websovellusten kehityksessä, esimerkiksi miten sovelluskehykset helpottavat työskentelyä tiimissä. Käyttöliittymää kehitettäessä pääsin tutustumaan ja käyttämään erilaisia JavaScript-kirjastoja, joiden käytöstä opin paljon. Lisäksi perustietoni ja -taitoni HTML:stä, CSS ja JavaScriptistä vahvistuivat. Opin paljon myös muista tekniikoista, kuten XML-kuvauskielestä ja websovelluksessa käytettävistä arkkitehtonisista malleista.

LÄHTEET

Angular Docs. (n.d.). AngularJS Templates. Haettu 25. 4 2019 osoitteesta Angular Docs: https://docs.angularjs.org/tutorial/step_02

Angular. (n.d.a). CLI Command Reference. Haettu 25. 4 2019 osoitteesta Angular: https://angular.io/cli

Angular. (n.d.b). Releases. Haettu 15. 2 2019 osoitteesta Angular: https://angular.io/guide/releases

Angular. (n.d.c). Style Guide. Haettu 13. 2 2019 osoitteesta Angular: https://angular.io/guide/styleguide

Angular. (n.d.d). Testing. Haettu 20. 2 2019 osoitteesta Angular: https://angular.io/guide/testing

AngularJS Docs. (n.d.). AngularJS Templates. Haettu 5. 2 2019 osoitteesta AngularJS: https://docs.angularjs.org/guide/concepts

Babeljs. (n.d.). What is Babel? Haettu 25. 4 2019 osoitteesta Babeljs: https://babeljs.io/docs/en/index.html

Backbone. (n.d.). Backbone.js. Haettu 15. 2 2019 osoitteesta Backbone: http://backbonejs.org/

Bashir, A. (12. 9 2014). What are MVP and MVC and what are the differences. Haettu 25. 4 2019 osoitteesta Stackoverflow: https://stackoverflow.com/a/25816573

Block, G. (2008). What are MVP and MVC and what is the difference? Haettu 5. 2 2019 osoitteesta Stackoverflow: https://stackoverflow.com/a/101561

Chaffey, D. (11. 7 2018). Mobile Marketing Statistics compilation. Haettu 7. 2 2019 osoitteesta Smart insights: https://www.smartinsights.com/mobile-marketing/mobile-marketing-analytics/mobile-marketing-statistics/

CLI Vuejs. (n.d.). Overview. Haettu 25. 4 2019 osoitteesta CLI vuejs: https://cli.vuejs.org/guide/

Cowart, J. (14. 6 2012). What is a Hybrid Mobile App? Haettu 25. 4 2019 osoitteesta Telerik: https://www.telerik.com/blogs/what-is-a-hybrid-mobile-app-

Dziwoki, M. (19. 9 2019). What's the difference between AngularJS and Angular? Haettu 5. 2 2019 osoitteesta Gorrion: https://gorrion.io/blog/angularjs-vs-angular

Flanagan, D. (2006). JavaScript: The Definitive Guide. Teoksessa D. Flanagan, JavaScript: The Definitive Guide (ss. 1-2). O'Reilly Media, Inc. Haettu 29. 1 2019 osoitteesta

https://books.google.fi/books?hl=fi&lr=&id=k0CbAgAAQBAJ&oi=fnd&pg=PT6&dq=javascript&ots=O3pzmmeAwT&sig=_Omkn8SSbfXa7_55_2mlwTlF-Xc&redir_esc=y#v=onepage&q&f=false

Github. (n.d.a). Angular Changelog. Haettu 15. 2 2019 osoitteesta Github: https://github.com/angular/angular/blob/master/CHANGELOG.md

Github. (n.d.b). AngularJS Changelog. Haettu 15. 2 2019 osoitteesta Github: https://github.com/angular/angular.js/blob/master/CHANGELOG.md

Github. (n.d.c). create-react-app. Haettu 25. 4 2019 osoitteesta Github: https://github.com/facebook/create-react-app

Github. (n.d.d). Facebook Changelog. Haettu 15. 2 2019 osoitteesta Github: https://github.com/facebook/react/blob/master/CHANGELOG.md

Github. (n.d.e). Framework sizes. Haettu 8. 2 2019 osoitteesta Github: https://gist.github.com/Restuta/cda69e50a853aa64912d

Github. (n.d.f). Front-end JavaScript frameworks. Haettu 8. 2 2019 osoitteesta Github: https://github.com/collections/front-end-javascript-frameworks

Github. (n.d.g). Vue Releases. Haettu 15. 2 2019 osoitteesta Github: https://github.com/vuejs/vue/releases

Google. (8. 2 2019). JavaScript Framework Google Searches. Haettu 8. 2 2019 osoitteesta Google Trends: https://trends.google.fi/trends/explore?date=2010-01-01%202019-02-13&q=%2Fm%2F012l1vxv,%2Fm%2F0j45p7w,Angular,%2Fg%2F11c0vmgx5d,%2Fm%2F0h94450

Hurff, S. (n.d.). How to design for thumbs in the era of huge screens. Haettu 1. 3 2019 osoitteesta Scotthurff: http://scotthurff.com/posts/how-to-design-for-thumbs-in-the-era-of-huge-screens

Jindle, Y. (24. 12 2012). Is backbone js really an MVC? Haettu 6. 2 2019 osoitteesta Stackoverflow: https://stackoverflow.com/a/14978913

JQuerymobile. (n.d.). jQuery Mobile, Backbone.js and Require.js. Haettu 8. 2 2019 osoitteesta JQuerymobile: http://demos.jquerymobile.com/1.4.4/backbone-requirejs/

Konecranes. (n.d.). Ohjain. Haettu 22. 2 2019 osoitteesta Konecranes: https://www.konecranesusa.com/equipment/overhead-cranes/smart-features-for-overhead-cranes/smart-controller

Konecranes. (n.d.). Siltanosturi. Haettu 22. 2 2019 osoitteesta Konecranes: https://www.konecranes.com/equipment/overhead-cranes/open-winch-cranes/smartonr-open-winch-crane

Konecranes. (n.d.). Teollisuudenalat. Haettu 25. 4 2019 osoitteesta Konecranes: https://www.konecranes.com/fi/teollisuudenalat

Konecranes. (n.d.). The crane with brain - SMARTON. Haettu 25. 4 2019 osoitteesta Konecranes: https://lv.konecranes.lv/sites/default/files/download/the_crane_with_a_brain_-_smarton.pdf

Konecranes. (n.d.). Tietoa. Haettu 25. 4 2019 osoitteesta Konecranes: https://www.konecranes.com/fi/tietoa

Kotilainen, S. (2019). JavaScript valtaa maailman. Tivi, 19-23.

Krause, Stefan. (27. 9 2018). JS web frameworks benchmark – Round 8. Haettu 19. 2 2019 osoitteesta Stefankrause: https://www.stefankrause.net/wp/?p=504

Krause, Stefan. (n.d.). Results for js web frameworks benchmark – round 8. Haettu 19. 2 2019 osoitteesta Stefankrause: https://stefankrause.net/js-frameworks-benchmark8/table.html

Krumins, C. (13. 10 2015). Smartphone Ownership, Usage And Penetration By Country. Haettu 7. 2 2019 osoitteesta Smsglobal: https://thehub.smsglobal.com/smartphone-ownership-usage-and-penetration

Kumar, S. (25. 5 2015). Difference Between Library and Framework. Haettu 5. 2 2019 osoitteesta c-sharpcorner: https://www.c-sharpcorner.com/UploadFile/a85b23/framework-vs-library/

Marty. (25. 5 2016). What are the main differences between Babel and TypeScript. Haettu 25. 4 2019 osoitteesta Stackoverflow: https://stackoverflow.com/a/37430993

Mobileangularui. (n.d.). Start learning Mobile Angular UI. Haettu 8. 2 2019 osoitteesta Mobileangularui: http://mobileangularui.com/docs/

Møller, A.;& Schwartzbach, M. (2006). An Introduction to XML and Web Technologies. Teoksessa A. Møller;& M. Schwartzbach, An Introduction to XML and Web Technologies (ss. 3-6). Addison-Wesley.

Peltomäki, J.;& Nykänen, O. (2006). Web-selainohjelmointi. Docendo Finland Oy.

Reactjs. (n.d.a). React JSX. Haettu 25. 1 2019 osoitteesta Reactjs: https://reactjs.org/docs/introducing-jsx.html

Reactjs. (n.d.b). React Tutorial. Haettu 25. 1 2019 osoitteesta Reactjs: https://reactjs.org/tutorial/tutorial.html

Reactjs. (n.d.c). Test Utilities. Haettu 20. 2 2019 osoitteesta ReactJS: https://reactjs.org/docs/test-utils.html

Reactjs. (n.d.d). Thinking in React. Haettu 14. 2 2019 osoitteesta ReactJS: https://reactjs.org/docs/thinking-in-react.html

w3schools. (n.d.). HTML intro. Haettu 29. 1 2019 osoitteesta w3schools: https://www.w3schools.com/html/html_intro.asp

W3Techs. (1. 3 2019). Usage of JavaScript libraries for websites. Haettu 1. 3 2019 osoitteesta W3Techs: https://w3techs.com/technologies/overview/javascript_library/all

Wikipedia. (n.d.a). Command-line interface. Haettu 25. 4 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Command-line_interface

Wikipedia. (n.d.b). Model-view-controller. Haettu 5. 2 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller

Wikipedia. (n.d.c). Model-view-viewmodel. Haettu 5. 2 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93viewmodel

Wikipedia. (n.d.d). Single-page application. Haettu 31. 1 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Single-page_application

Wikipedia. (n.d.e). Visual Studio Code. Haettu 25. 4 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Visual_Studio_Code

Wikipedia. (n.d.f). Model-view-presenter. Haettu 5. 2 2019 osoitteesta Wikipedia: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93presenter

Vue test utils. (n.d.). Vue Test Guides. Haettu 8. 2 2019 osoitteesta Vue test utils: https://vue-test-utils.vuejs.org/

Vuejs. (n.d.a). Team. Haettu 15. 2 2019 osoitteesta Vuejs: https://vuejs.org/v2/guide/team.html

Vuejs. (n.d.b). The Vue Instance. Haettu 19. 2 2019 osoitteesta Vuejs: https://vuejs.org/v2/guide/instance.html

Vuejs. (n.d.c). Vue Guide. Haettu 25. 1 2019 osoitteesta Vuejs: https://vuejs.org/v2/guide/

Vuejs. (n.d.d). Unit Testing. Haettu 8. 2 2019 osoitteesta Vuejs: https://vuejs.org/v2/guide/unit-testing.html