Oskari Tiala

ALUSTARIIPPUMATON MOBIILI-KEHITYS REACT NATIVELLA

Tekniikka
2019

VAASAN AMMATTIKORKEAKOULU
Tietotekniikka

TIIVISTELMÄ

Tekijä	Oskari Tiala
Opinnäytetyön nimi	Alustariippumaton mobiilikehitys React Nativella
Vuosi	2019
Kieli	suomi
Sivumäärä	31
Ohjaaja	Pirjo Prosi

Tässä opinnäytetyössä oli tavoitteena havainnollistaa mobiilikehitystä React Native -teknologialla sekä vertailla sitä perinteiseen mobiilikehitykseen natiivikielillä, kuten Java (Android) ja Swift/Objective-C (iOS). Opinnäytetyössä suoritettu vertailu natiiviteknologiaan on tehty Android-kehittäjän näkökulmasta.

Referenssityönä toteutettiin alustariippumaton mobiilisovellus, joka käyttää olemassa olevaa REST-rajapintaa. Referenssityön toimeksiantaja oli ketterään ohjelmistokehitykseen ja liiketoiminnan digitalisoimiseen erikoistunut Haltu Oy, ja tilaajana maanrakennus- ja murskausalan tuotteisiin erikoistunut Saranco Oy.

Taustatutkimuksena tälle opinnäytetyölle tutustuttiin erilaisiin olemassa oleviin ratkaisuihin, jotka pyrkivät mahdollistamaan alustariippumattoman kehityksen erilaisten mobiilikäyttöjärjestelmien välillä.

Työn lopputuloksena syntyi vaatimusmäärittelyn mukainen mobiilisovellus sekä Android- että iOS-alustalle.

Avainsanat Mobiilikehitys, JavaScript, React, React Native

VAASAN AMMATTIKORKEAKOULU
VAASA UNIVERSITY OF APPLIED SCIENCES
Tietotekniikka

ABSTRACT

Author	Oskari Tiala
Title	Cross-platform mobile development with React Native
Year	2019
Language	Finnish
Pages	31
Name of Supervisor	Pirjo Prosi

The objective of this thesis was to demonstrate using React Native in cross-platform mobile development, and to compare it to developing applications in native languages, such as Java for Android, and Swift/Objective-C for iOS. All comparisons made during this thesis is made from the view of an Android developer.

As a part of this thesis, a reference application was developed under Haltu Oy, an agile software house specializing in digitalization of business. The client was Saranco Oy, which is a company that specializes in products of civil engineering and construction.

As background research, different types of solutions for cross-platform mobile development were investigated.

As a final product, a mobile application communicating with an existing REST API was implemented according to the requirements specification. The final product was targeted for both iOS and Android platforms.

Keywords Mobile development, JavaScript, React, React Native

TERMIT JA LYHENTEET

Redux	Redux on ennustettavissa oleva tilakirjasto JavaScript-sovelluksille.
Action	Actionit ovat datakuormia, jotka välittävät tietoa sovelluksen Storeen.
Reducer	Reducerit määrittävät, miten sovelluksen tila muuttuu Storeen lähetettyjen Actioneiden toimesta.
Store	Store säilyttää sovelluksen globaalia ja keskitettyä tilapuuta.
Renderöinti	Prosessi, jossa näytölle piirretään kuva tai komponentti

SISÄLLYS

KUVIO- JA TAULUKKOLUETTELO

1 JOHDANTO

Mobiilisovellusmarkkinat ovat kasvava ala, jonka liikevaihdon arvioidaan yltävän jopa 188,9 miljardiin Yhdysvaltain dollariin vuonna 2020 /1/. Alaa hallitsevat kahden suurimman toimijan omistamat käyttöjärjestelmät. Vuoden 2018 toisella kvartaalilla 88 % kaikkien mobiilipäätelaitteiden käyttöjärjestelmistä oli Googlen Androideja, kun taas n. 12 % laitteista käyttivät Applen iOS-käyttöjärjestelmää /2/.

Markkinoiden jakautuminen luo sovelluskehittäjille ja ohjelmistotaloille luonnollisesti paineita tarjota sovelluksia molemmille käyttöjärjestelmille.

Monialustaisen mobiilikehityksen ongelmana ovat pitkään olleet käyttöjärjestelmien eroavaisuudet; Android-käyttöjärjestelmän sovellukset kirjoitetaan pääsääntöisesti Javalla tai Kotlinilla, kun taas iOS-käyttöjärjestelmän sovellukset kirjoitetaan Swiftillä tai Objective-C:llä. Alustojen välillä ei ole lähtökohtaisesti yhteistä kieltä, minkä lisäksi kielet ovat luonteeltaan erilaisia, joten tarvittavien resurssien ja kehittäjien määrä kasvaa.

React Native pyrkii ratkaisemaan ongelman mahdollistamalla mobiilisovellusten kehittämisen eri alustalle JavaScript -ohjelmointikielellä.

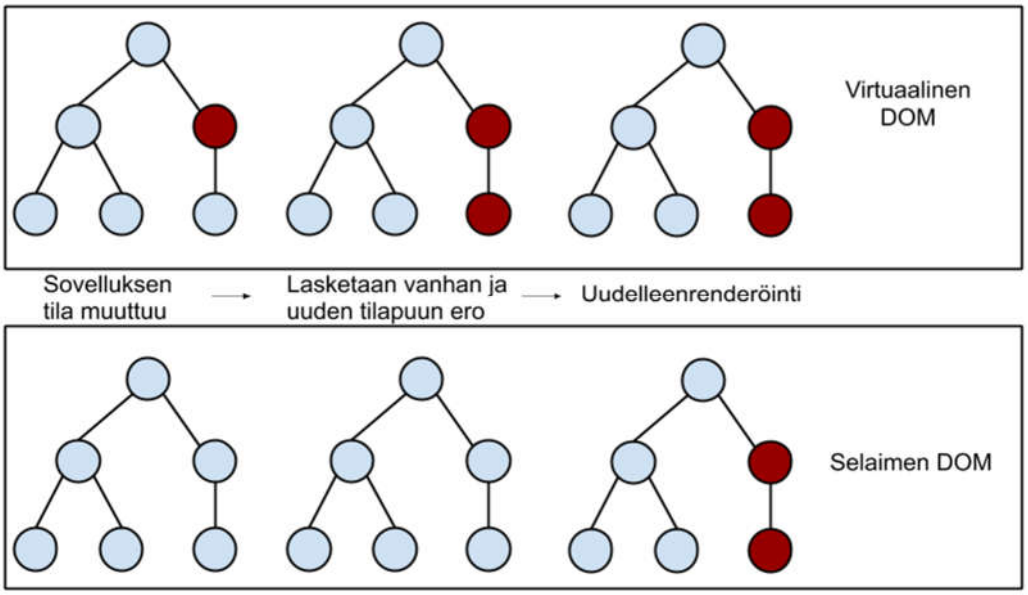
Työn tavoitteena on verrata React Nativella kirjoitettuja sovelluksia natiivisovelluksiin kehittäjän sekä yrityksen näkökulmasta. Referenssityönä toteutetaan asiakkaalle olemassa olevan web-sovelluksen REST-rajapintaa käyttävä mobiilisovellus asiakkaan kanssa laaditun vaatimusmäärittelyn mukaan. REST-rajapinta on toteutettu Django-sovelluskehyksellä sekä Django REST framework -kirjastolla. Opinnäytetyön keskiössä on erityisesti mobiilisovellus, React Nativen toimintaperiaatteet sekä sen vertautuminen mobiilisovelluksien kehittämiseen natiivikielillä. Vertailu on tehty Android-kehittäjän näkökulmasta, mutta alustariippumattoman kehityksen ollessa kyseessä samat asiat pätevät myös esimerkiksi iOS-käyttöjärjestelmässä.

2 REACT

Ennen React Nativen esittelemistä on mainittava muutama sana Reactista, johon React Native pohjautuu. React on JavaScript-kirjasto, jolla kirjoitetaan web-sisällölle käyttöliittymiä sekä toiminnallisuuksia. Avaintermi React-filosofiassa on komponentti. Reactissa on tapana jakaa käyttöliittymän osat uudelleenkäytettäviin komponentteihin, joille usein kuljetetaan tietoa komponentin ulkopuolelta. Tieto voidaan kuljettaa komponentille ns. *propsina* (engl. Prop, properties). /3, 5/

Oletuksena React-projektin sisällä kulkeva tieto säilytetään kunkin komponentin sisäisessä tilassa, ja mikäli esimerkiksi päänäkymän tulee olla tietoinen valikkokomponentin tilasta, tilaa kuljetetaan valikkokomponentilta eteenpäin päänäkymälle. /4, 5/

Yksi Reactin merkittävimpiä etuja on virtuaalinen DOM (Document object model, Kuva 1), joka laskee ulkoasun muuttuessa tehokkaimman tavan tehdä muutokset DOMiin. Uudelleenpiirtämiseen pystyy toki vaikuttamaan manuaalisesti, mutta oletuksena uudelleenpiirtäminen tapahtuu niille komponenteille, joiden sisäinen tila, tai sille annettu data (props) muuttuu. /6, 7/

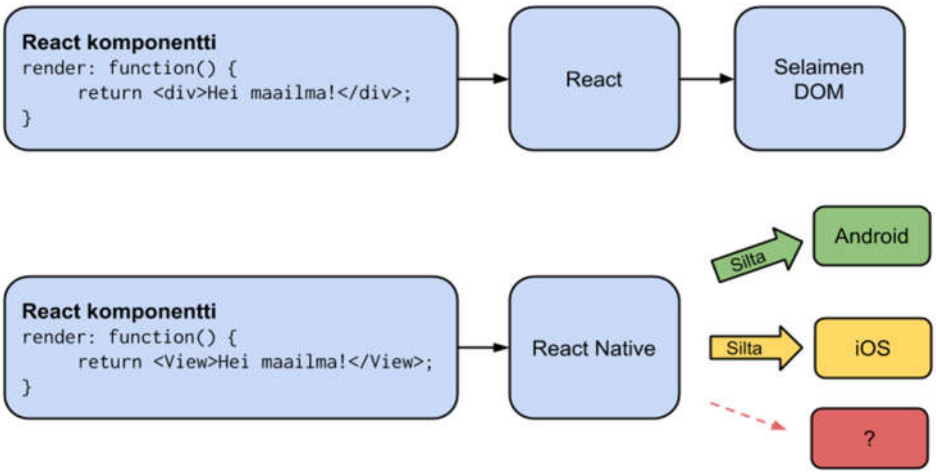


Kuva 1. Virtuaalisen DOMin toimintaperiaate

Kuvan 1 punaiset solmut kuvaavat DOM:in solmuja (engl. Node), joiden tila on muuttunut. React vertaa virtuaalisen DOM:in aiempaa versiota uuteen versioon sekä laskee muutokset näiden kahden version välillä, piirtäen uudelleen varsinaiseen DOM:iin ainoastaan ne solmut, joiden tila on muuttunut, täten säästäen laskentatehoa perinteiseen DOM-malliin verrattuna, jossa koko dokumenttipuu uudelleenpiirretään jokaisen päivityksen jälkeen. /6, 7/

3 REACT NATIVE

React Native on Facebookin kehittämä, vuonna 2015 julkaistu ja esitelty JavaScript-sovelluskehys, joka pyrkii ratkaisemaan perinteistä monialustakehitystä koskevia ongelmia. React Nativella sovellukset kirjoitetaan kummallekin käyttöjärjestelmälle käyttäen React-kirjastoa. /8/

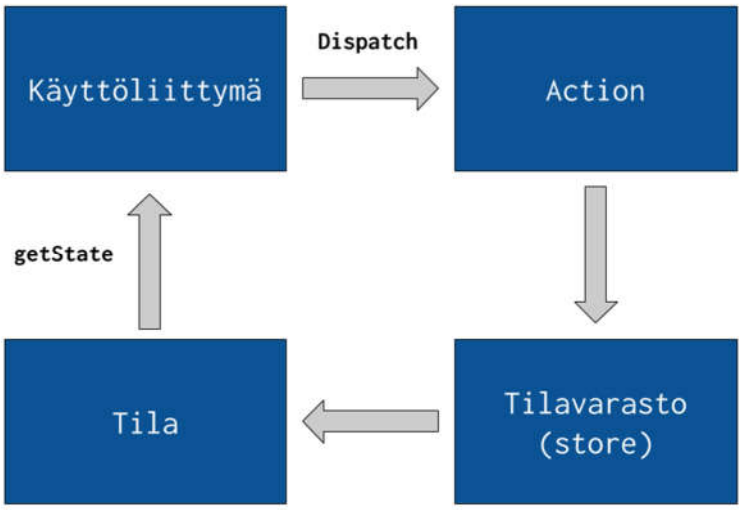


Kuva 2. Komponenttien renderöinti Reactissa ja React Nativessa

React Native -sovellukset kirjoitetaan JavaScriptilla ja JSX:llä, joka on XML-tyyppinen merkintäkieli. Koodi käännetään sovellukseksi halutun käyttöjärjestelmän tukemaan muotoon. Sovellus suoritetaan päätelaitteen, eli useimmiten puhelimen tai tabletin, JavaScript-moottorissa. Päätelaitteella JavaScript-moottorissa ajettava koodi kutsuu React Native -sillan kautta natiivirajapinnan komponentteja (**Kuva 2**.) jonka ansiosta React Nativella toteutetut sovellukset sekä näyttävät, että tuntuvat natiiveilta. /9, 10/

3.1 Tilanhallinta

Oletuksena myös React Nativessa yleinen tilanhallinta tapahtuu *komponenttien sisäisessä tilassa* (component state), eikä muilla komponenteilla ole toistensa tilaan pääsyä, ellei tilaa viedä eksplisiittisesti komponenteilta eteenpäin. Komponenttien lukumäärän ja projektin koon kasvaessa tilan hallinnoimisesta saattaa tulla työlästä, joten monimutkaisissa järjestelmissä on suotavaa käyttää ulkoista tilanhallintakirjastoa. Redux (**Kuva 3.**) on esimerkki tällaisesta kirjastosta. Reduxissa sovelluksen tilaa hallitaan globaalissa ja keskitetyssä tilavarastossa, eli *storessa.* Toisin kuin komponenttien sisäisessä tilassa, Reduxissa mikä tahansa komponentti pystyy storeen yhdistämällä saada pääsyn globaaliin tilaan. Reduxin säilyttämä tilaobjekti on *muuttumaton* (immutable), eli tilaa päivitetään luomalla uusi tilaobjekti, joka mahdollistaa ns. aikamatkustamisen tilojen välillä, joka on arvokas ominaisuus muun muassa vikatilanteiden selvittämisessä. /10, 11/.



Kuva 3. Tiedon kulku Reduxissa

```
import React from 'react'
import { render } from 'react-dom'
import { Provider } from 'react-redux'
import { createStore } from 'redux'
import rootReducer from './reducers'
import App from './components/App'

const initialState = {
    user: {
        isAuthenticated: false,
        username: null,
        fieldErrors: {}
    },
    token: null,
    errors: null,
};

const store = createStore(rootReducer, initialState, applyMiddleware(logger));

render(
    <Provider store={store}>
        <App />
    </Provider>,
    document.getElementById('root')
)
```

Kuva 4. Storen luominen

Store luodaan Reduxin createStore-metodilla (**Kuva 4.**). Metodin ainoa pakollinen argumentti on reducer-funktio, joka kuvailee, miten sovelluksen tila muuttuu. Store hyväksyy ainoastaan yhden reducer-funktion, joten mikäli arkkitehtuurisista tai muista syistä halutaan luoda useampi reducer-funktio, ne voidaan yhdistää käyttämällä combineReducers -metodia. Alustava tila, eli preloadedState, on metodin valinnainen argumentti, joka kuvan 3 esimerkissä luo storeen rakenteen, jossa säilytetään käyttäjän sekä istunnon tietoja. Tämä on hyödyllinen esimerkiksi sovelluksissa, joissa halutaan säilyttää istunnon tietoja; palvelimelle tallennetut tiedot voidaan noutaa uudelleen istunnon alkaessa sekä asettaa alustavaan tilaan. createStore-metodin toinen valinnainen argumentti on enhancer, eli tehostajafunktio, jolla voidaan lisätä storeen kolmannen osapuolen lisätoiminnallisuuksia, kuten lokitus tai *väliohjelmisto* (middleware).

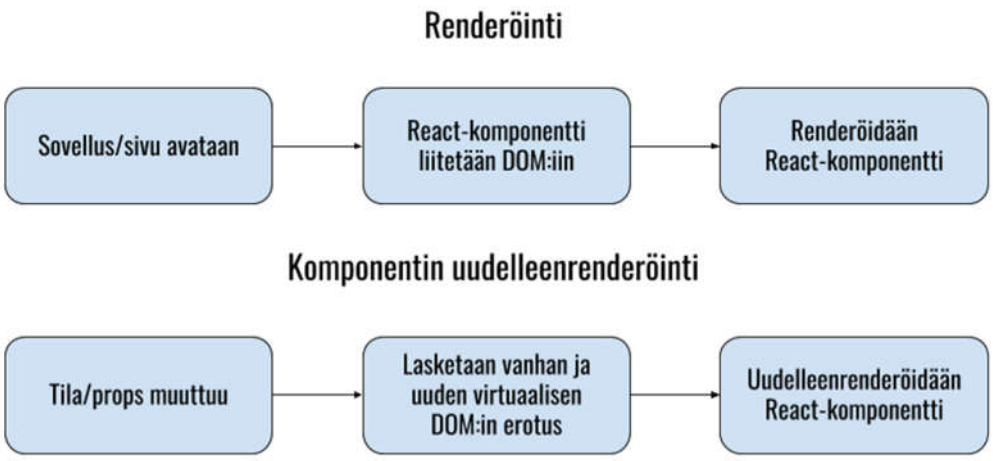
Ainoa tapa muuttaa Reduxissa oikeaoppisesti tilaa on lähettää storeen action storen dispatch -metodilla. Action on storeen tietoa kuljettava datakuorma. Actionit koostuvat actionin tyypistä, joka on merkkijonovakio, joka ei suoranaisesti kerro mikä storen tilassa muuttuu, vaan kuvaa ainoastaan mitä tapahtui. Tyypin lisäksi tarvitaan myös action creator, joka on yksinkertainen funktio, joka palauttaa actionin tyypin, sekä mahdollisen eteenpäin vietävän datakuorman.

Kutsutut actionit käyvät aina rekursiivisesti läpi kaikki reducerit; se mikä reducer käsittelee actionin, riippuu actionin tyypistä. Actioneita välitetään dispatch-metodilla, joka vaatii argumentiksi actionin sekä mahdollisen datan, eli nk. payloadin.

Reducerit ovat funktioita, jotka määrittelevät kuinka sovelluksen tila muuttuu, perustuen reducerin vastaanottamaan actioniin. Reducerit ottavat argumentteina sovelluksen nykyisen tilan sekä actionin, ja niiden perusteella joko luo ja palauttaa uuden tilan, tai palauttaa vanhan tilan.

3.2 Renderöinti

Renderöinnin elinkaari React Nativessa (**Kuva 5**.) on hyvin samankaltainen, kuin Reactissa, sillä erotuksella, että React Nativen renderöinti on riippuvainen käytetystä sillasta (**Kuva 2.**).



Kuva 5. React Nativen renderöinnin elinkaari

React Nativen komponenttien (**Kuva 6.**) olennainen osa on render-metodi, jonka täytyy palauttaa JSX-elementti. Render-metodi toimii ohjeistuksena natiivirajapintaa kutsuvalle sillalle siitä mitä ruudulla halutaan näyttää. /10/

```
export default class GreetingsComponent extends Component {
    render() {
        return (
            <View>
                <Text>Hello {this.props.name}!</Text>
            </View>
        );
    }
}
```

Kuva 6. React Native -komponentti

Kuvan 5 esimerkissä luodaan GreetingsComponent-niminen komponentti, jonka <View>-elementti ohjeistaa Android-alustalla kutsumaan Androidin natiivia näkymäkomponenttia (android.view). Näkymän sisällä on <Text>-elementti, joka puolestaan kutsuu Androidin natiivikomponenttia tekstinäkymälle (TextView). Text-elementissä käytetään myös jo aiemmin mainittua propsia (**Kuva 7.**), joka mahdollistaa tässä tapauksessa nimen kuljettamisen parent-komponentilta GreetingsComponent-komponentille.

```
<GreetingsComponent name='Oskari' />
```

Kuva 7. React Native -komponentin käyttäminen sovelluksessa

React Nativen yhteydessä puhutaan usein DOM:ista, vaikka teknisesti sillä ei tässä kontekstissa tarkoiteta samaa DOM:ia kuin selaimessa ja Reactissa. React Nativessa poikkeavaa on se, että React Native hallinnoi käyttöliittymää arkkitehtuurikerroksilla, jotka päättävät kuinka näkymät tulisi renderöidä, sekä pyrkivät optimoimaan käyttöliittymän muutoksia. Sovelluksen käynnistyessä UIManager pyytää JSX-elementtien perusteella vastaavat natiivikomponentit. UIManagerin käskyjen perusteella luodaan niin kutsuttuja *varjosolmuja* (Shadow nodes). Varjosolmut itsessään eivät renderöi näytölle mitään, vaan ovat pelkkä kuvaileva esitys siitä, mitä pitäisi renderöidä. Varjosolmujen perusteella luodaan varsinaiset käyttöliittymäelementit käyttäen Yoga-moottoria. Yoga on C-ohjelmointikielellä kirjoitettu avoimen lähdekoodin alustariippumaton sijoittelumoottori, jonka tehtävänä React Nativen renderöinnissä on päättää mihin käyttöliittymäkomponentit piirretään, komponenttien oletustyylien, sekä annettujen tyylimäärittelyjen perusteella. /10, 13, 12/

4 TAPAUSTUTKIMUS: MOUKARI

4.1 Projektin osapuolten esittely

Tässä luvussa käydään läpi varsinaisen referenssityön taustoja, sekä toteutusta. Toteutus-osuudessa käytettyjä esimerkkikoodeja on saatettu muuttaa tai obfuskoida tietoturvasyistä.

4.1.1 Haltu Oy

Haltu Oy on Tampereella sijaitseva ohjelmistoyritys, joka on erikoistunut web-palveluiden ja mobiilisovellusten kehittämiseen sekä palvelinten ylläpitoon. Ketterän ohjelmistokehityksen lisäksi Haltu Oy tarjoaa UX-konsultointia ja palvelumuotoilua. Haltun asiakasprojekteista noin 50 % koostuu mobiilisovelluksista.

Mobiilikehitystä Haltulla tehdään pääasiassa React Nativella, joka opinnäytetyön kirjoittamishetkellä koetaan olevan kustannustehokkain vaihtoehto kehittää alustariippumattomia mobiilisovelluksia.

Toimeksiantaja toivoo opinnäytetyöltä uusia näkökulmia sekä parempaa ymmärrystä React Nativesta, siitäkin huolimatta, että Haltulla on referensseinä sekä ylläpidossa, useita kymmeniä React Native -projekteja.

4.1.2 Saranco Oy

Saranco Oy on maanrakennus- ja murskausalan tuotteisiin erikoistunut yritys. Sarancon tuotteisiin kuuluvat mm. seulaverkot sekä Moukari, joka on tarkoitettu työmaiden tehtävien hallintaan, laitteiden ja osien seurantaan, varastointiin sekä raportointiin.

4.2 Tekniikoiden esittely

Seuraavassa kappaleessa esitellään teknologiat, joilla referenssiprojekti toteutettiin.

4.2.1 Taustajärjestelmä

Referenssityön backend-toteutus on tehty käyttäen Django-ohjelmistokehystä sekä Django REST framework -kirjastoa, joka on suunniteltu REST-rajapintojen toteutukseen. Django on Python-ohjelmointikielen web-sovelluskehys, joka mahdollistaa nopean kehittämisen muunmuassa Object Relational Mappingin ansiosta, joka toimii abstraktiona sovelluksen ja tietokannan välillä mahdollistaen tietokantaobjektien käsittelyn Python-objekteina.

Django REST framework -kirjasto tarjoaa valmiita työkaluja, joita voidaan käyttää REST-rajapintojen toteuttamiseen. Näitä ovat mm:

- Automaattinen verkkoselaimella selattava rajapinta
- Autentikaatio, mm. Oauth
- Datan *serialisoiminen* (serializing) haluttuun muotoon

Kaikki edellämainituista olisi myös toteutettavissa pelkällä Django-ohjelmistokehyksellä, mutta kyseisen kirjaston käyttäminen vähentää työtunteja huomattavasti.

Sekä Djangon, että Django REST frameworkin ominaisuudet ovat täysin muokattavissa.

4.2.2 Mobiilisovellus

Moukarin mobiilisovellus on tehty React Native -ohjelmistokehyksellä. React Native oli luonteva valinta, sillä projektin budjetti oli rajallinen, ja mobiilisovellus tarvittiin sekä iOS- että Android-alustoille.

Sovelluksen navigointiin ja reititykseen eri näkymien välillä käytettiin React Navigation -kirjastoa.

Kommunikointi REST-rajapinnan kanssa on toteutettu axios-nimisellä kirjastolla, jota käytetään HTTP-pyyntöjen lähettämiseen, sillä ainakin projektin toteutushetkellä JavaScriptin Fetch-rajapinnalla HTTP-pyyntöjen käsittely oli projektiryhmän mielestä tarpeettoman monimutkaista verrattuna axiosiin. Sittemmin kyseistä Fetch-rajapintaa on kehitetty, ja tilanne saattaa olla muuttunut.

Tilanhallinta on toteutettu käyttäen Redux-kirjastoa, sillä valmiin projektin tilapuun koko ennakoitiin olevan niin suuri, että tilanhallinnan käsitteleminen komponentin tiloissa arveltiin olevan ongelmallista. Datan synkronointi offline-tilasta palvelimelle toteutettin Reduxin persist-lisäosalla.

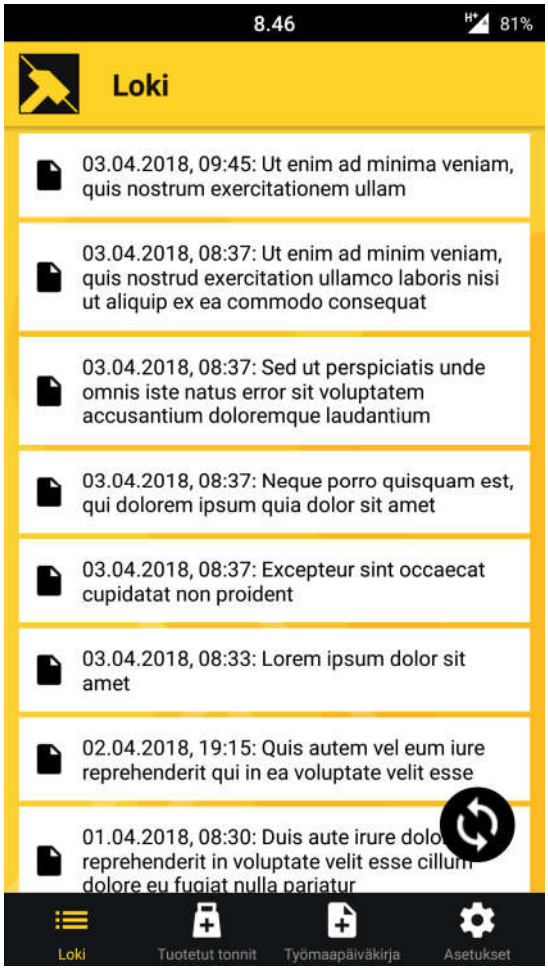
Aikaobjektien parsiminen, validoiminen ja muokkaaminen toteutettiin Moment.js-kirjastolla, joka on JavaScript-maailmassa eräänlainen De Facto -standardi kyseisiin käyttötarkoituksiin.

4.3 Referenssityön esittely

4.3.1 Esittely

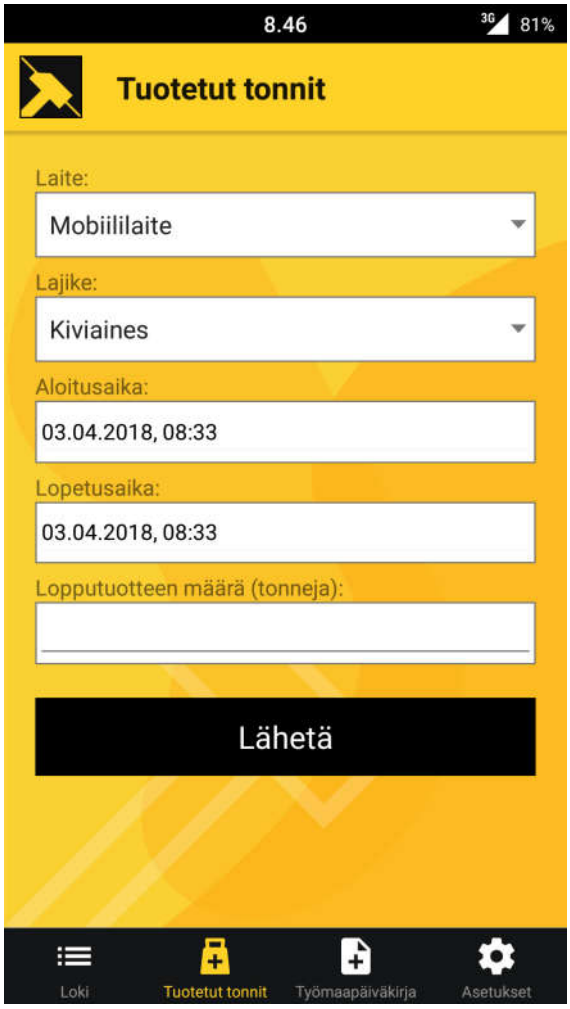
Moukari on Saranco Oy:n tuote, joka on tarkoitettu työmaiden toiminnan tehostamiseen. Referenssityössä toteutettiin kyseiseen olemassaolevaan järjestelmään mobiilisovellus, jota tässä osiossa kutsutaan *mobiilimoukariksi*.

Mobiilimoukarin avulla yritys voi hallinnoida ja seurata kiviainesseuloja sekä niiden vaikutusta tuotantoon. Mobiilimoukariin syötetyn tiedon perusteella käyttäjät voivat nähdä esimerkiksi kuinka tietyn raekoon tuotanto laskee, ja sen perusteella tehdä päätöksiä esimerkiksi uuden seulan hankinnasta.



Kuva 8. Mobiilimoukarin lokinäkymä testidatalla

Mobiilimoukari-sovellus koostuu kolmesta päänäkymästä. Kirjautumisen jälkeen käyttäjä ohjataan lokinäkymään (**Kuva 8.**), joka näyttää aikaleimoilla varustetun listauksen työmaalla tuotetuista kiviainesmääristä, sekä työmaapäiväkirjamerkinnöistä. Lokimerkinnät päivitetään automaattisesti, mutta kelluvasta synkronointinapista (FAB, Floating action button) painamalla käyttäjä voi pakottaa sovelluksen hakemaan rajapinnasta uudet lokimerkinnät.



Kuva 9. Mobiilimoukarin tuotetut tonnit -näkymä

Tuotetut tonnit -näkymässä (**Kuva 9.**) käyttäjä voi merkitä järjestelmään työmaalla kullakin laitteella tuotetun kiviaineksen määrän, lajikkeen, sekä aikaikkunan.

Kuva 10. Mobiilimoukarin työmaapäiväkirja-näkymä

Työmaapäiväkirja-näkymässä (**Kuva 10.**) voidaan syöttää vapaita merkintöjä työmaata koskevista asioista, joista muiden työntekijöiden tulisi olla tietoisia. Tällaisia voivat olla esimerkiksi työtä haittaavat sääolosuhteet, tai muut työn edistymiseen vaikuttavat tapahtumat.

Kuva 11. Mobiilimoukarin asetukset-näkymä

Asetukset-näkymästä (**Kuva 11.**) käyttäjä voi valita pudotusvalikosta työmaan. Saatavilla olevat työmaat riippuvat käyttäjän oikeuksista, ja hänelle lisätyistä työmaista Moukarin taustajärjestelmässä. Näkymässä on myös ilmoitus käyttäjälle sovelluksen käännöstoiminnallisuudesta.

4.3.2 Toiminnallinen suunnittelu ja määrittely

Moukari-projektin mobiiliosuuden suunnitteluun osallistui asiakkaan edustajien lisäksi kaksi ohjelmistokehittäjää, projektipäällikkö, graafikko sekä käyttäjäkokemuksen suunnittelija, jonka tehtävänä oli yhteistyössä kehittäjien kanssa suunnitella ja parantaa projektin aikana sovelluksen käyttäjäkokemusta (UX, user experience) annetun käyttäjäkuvauksen ja yleisten hyvien käytäntöjen perusteella.

Asiakkaan toiveina mobiilisovellukselle oli kevyt ja selkeä käyttöliittymä, joka tarjoaa heti ensi silmäyksellä hyvän kuvan työmaan tilanteesta. Vaatimuksena oli myös offline-toiminnallisuus, sillä työmailla verkko on paikoitellen huono, eikä internet-yhteys ole aina saatavilla.

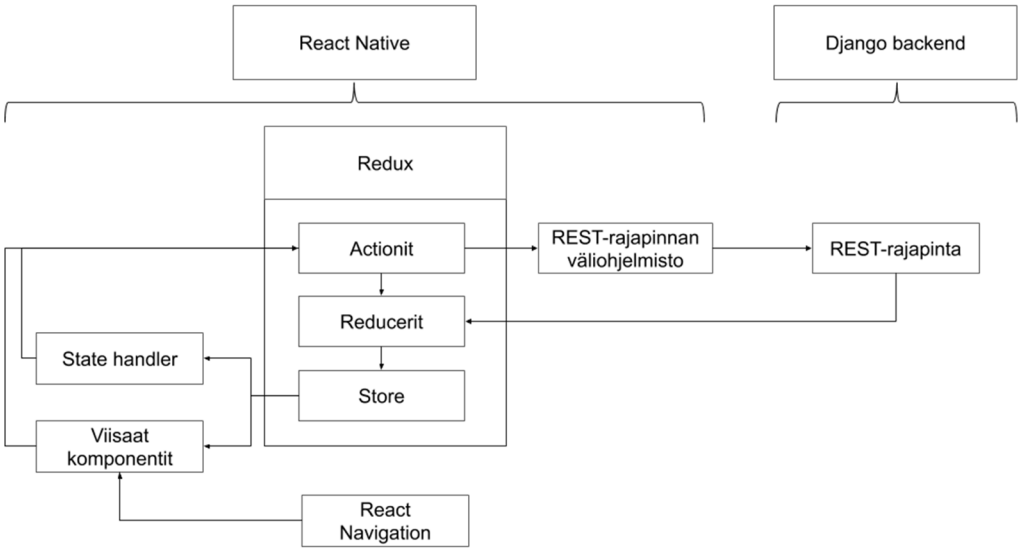
Käyttöliittymän pääväreiksi valittiin musta ja keltainen. Käyttöliittymän suunnittelussa pyrittiin ottamaan huomioon helppo käytettävyys, sillä mobiilisovelluksen ei haluttu häiritsevän loppukäyttäjän varsinaista työtä.

Toiminnallisuuksien osalta merkittävin ominaisuus sovellukselle oli tuotettujen tonnien syöttö, jonka avulla pystyi kirjaamaan kullakin laitteella tuotettujen kiviainesten lajikkeen sekä lopputuotteen määrän. Sovelluksella täytyi myös pystyä lisäämään työmaapäiväkirjamerkintöjä, joiden avulla voidaan jakaa tärkeää tietoa työmaan sisällä. Sekä tuotetut tonnit, että päiväkirjamerkinnät näytetään lokinäkymässä, joka sisältää listan merkintöjä sekä aikaleimat. Sovellukseen tarvittiin myös asetukset-näkymä, josta voitiin valita työmaa.

Mobiilisovellusta kehitettiin kiinteässä yhteistyössä asiakkaan kanssa, viikoittaisissa tapaamisissa käytiin läpi edistyminen sekä seuraavat tavoitteet. Projektin arkkitehtuurin (**Kuva 12.**) suunnitteluun osallistui koko tiimi. Arkkitehtuuria kehitettiin olemassa olevan järjestelmän rajoitteiden puitteissa suunnittelupalavereissa ennen varsinaisen kehitysvaiheen aloittamista.

4.3.3 Toteutusratkaisut

Mobiilimoukarin toteutuksessa pyrittiin noudattamaan React-ideologiaa; komponentit jaettiin ns. tyhmiin ja viisaisiin komponentteihin. Komponenteista pyrittiin tekemään uusiokäytettäviä, eli tyhmiä komponentteja, jotka eivät itse käsittele dataa, vaan sen esittämä data annettiin sille ulkopuolelta propseina.



Kuva 12. Karkea kuvaus mobiilimoukarin arkkitehtuurista

Moukarin kaikki kommunikaatio taustajärjestelmän kanssa tapahtuu actioneiden (**Kuva 13.**) kautta. Taustajärjestelmän rajapintaa varten kirjoitettu väliohjelmisto käsittelee actioneiden määrittelemät kutsut, sekä muodostaa valmiin HTTP-pyynnön ja hoitaa pyynnön autentikoinnin taustajärjestelmään. Actionit voivat myös toimittaa dataa suoraan reducereille, esimerkiksi sovelluksen tilan muuttuessa.

```
export function setCredentials(credentials) {
  return function(dispatch) {
    dispatch({ type: SET_CREDENTIALS, payload: credentials });
  };
}

export function login(credentials, navigation) {
  return {
    [API]: {
      method: POST,
      url: `${API_URL}/<login_endpoint_url>/`,
      types: [LOGIN_REQUEST, LOGIN_SUCCESS, LOGIN_FAIL],
      data: credentials
    }
  };
}

export function logout(token, navigation) {
  return {
    [API]: {
      method: POST,
      url: `${API_URL}/<logout_endpoint_url>/`,
      types: [LOGOUT_REQUEST, LOGOUT_SUCCESS, LOGOUT_FAIL],
      data: token,
      next: 'Login',
      navigation: navigation
    }
  };
}
```

Kuva 13. Esimerkki sovelluksen actioneista, jotka kutsuvat REST-rajapintaa

Sovelluksen offline-tuki implementoitiin siten, että luotiin tilanhallinnalle oma JavaScript-moduuli (*state handler*), joka pitää sovelluksen ja palvelimen datan synkronoituna. State handler kuuntelee sovelluksen kaikkia actioneita sekä niiden tyyppejä, ja tekee päätöksiä vertailemalla uusia actioneita aiemmin tapahtuneisiin actioneihin. State handler toteutettiin Reduxissa käytettävällä kolmannen osapuolen redux-persist -kirjastolla, joka mahdollistaa Reduxin globaalista tilapuusta datan tallentamisen päätelaitteen välimuistiin (AsyncStorage). Lähes koko Reduxin

tilapuu tallennettiin välimuistiin, jolloin sovellusta oli mahdollista käyttää myös offline-tilassa. State handler -moduuli myös kuunteli päätelaitteen verkkoyhteyden tilaa, ja kun laitteella oli taas verkkoyhteys käytettävissä, välimuistiin tallennettu data synkronoitiin palvelimen tietokantaan, ja kyseinen data merkittiin Reduxin tilapuussa synkronoiduksi.

Taustajärjestelmän REST-rajapinnan *päätepisteet* (endpoint) jouduttiin eriyttämään omaksi mobiilirajapinnakseen, muun muassa autentikointieroavaisuuksien takia, jotta *taaksepäin yhteensopivuus* (backward compatibility) voitiin taata.

Kehittämisessä käytettiin Git versionhallintajärjestelmää, sekä yksinkertaistettua Gitflow-variaatiota /14/; ominaisuudet kehitettiin omissa git-haaroissaan, ja kun ominaisuudet katsottiin olevan valmiita, ne katselmoitiin ja testattiin, jonka jälkeen ne haarat yhdistettiin develop-haaraan. Julkaisua valmistellessa develop haara yhdistettiin master-haaraan, josta tehtiin julkaisu, ja johon lisättiin *semanttisen versioinnin* (SemVer, Semantic Versioning) /15/ mukainen *tunniste* (tag).

5 TULOKSET

5.1 Vertautuminen perinteiseen mobiilikehitykseen

Merkittävin fundamentaalinen ero esimerkiksi Java-kielellä kirjoitetun ja React Nativella -sovelluskehyksellä kirjoitetun sovelluksen välillä on tapa, joilla sovelluksia ajetaan päätelaitteella. Java-koodi käännetään tavukoodiksi, jonka Java-virtuaalikone (JVM, Java virtual machine) tulkitsee. Androidin kehitystyökaluihin (SDK, software development kit) kuuluva DEX-kääntäjä kääntää Javan tavukoodin Dalvik-tavukoodiksi, jonka Androidin Dalvik-virtuaalikone kääntää binäärimuotoiseksi laitekoodiksi (”natiivikoodi”), jota suoritetaan päätelaitteella.

React Nativella kirjoitettua JavaScript-koodia ei missään vaiheessa käännetä natiivikoodiksi, vaan sovellusta ajetaan päätelaitteen JavaScript-moottorissa, josta käskytetään natiivirajapinnan käyttöliittymäkomponentteja.

Teoriassa tämä tarkoittaa sitä, että Android-sovelluksien kehittäminen Javalla vaatii koodin muuttamisen jälkeen sovelluksen uudelleenkääntämisen ja päivittämisen puhelimelle. React Nativessa puolestaan on mukana ”hot reloading” -toiminto, joka pitää sovelluksen päätelaitteella käynnissä, mutta injektoi päätelaitteelle ajon aikana tehdyt muutokset, tehden React Native -sovelluksien kehittämisestä tietyssä mielessä nopeampaa.

Toinen merkittävä ero kehittäjälle on tyylittely. Android tarjoaa kehittäjälle graafisen käyttöliittymän, johon käyttöliittymäkomponentteja voi lisätä drag and drop -tyylillä, sekä muokata niiden sijaintia ja tyylejä. React Nativessa sen sijaan esimerkiksi komponenttien sijainti ja tyylittelyt määritetään CSS-tyyppisillä tyyliohjeilla. React Nativen tyyliohjeet poikkeavat CSS:stä siten, että ne kirjoitetaan camelCasella, eli esimerkiksi ohje ”border-radius” muuttuu React Nativessa muotoon ”borderRadius”.

Vaikka React Nativella kehitetyt sovellukset vastaavatkin ulkonäöltään ja tuntumaltaan natiivisovelluksia, React Native -sovellusten suorituskyky ei ole optimoinneista huolimatta aivan natiivisovellusten tasolla. Myös React Nativen ylimääräiset abstraktiokerrokset saattavat projektista riippuen mahdollistaa suorituskykyongelmien kasaantumisen, sekä tehdä vianselvityksestä hankalaa.

Yrityksen näkökulmasta merkittävä ero on se, että monialustaisen sovelluksen kehittäminen natiivikielillä vaatii sekä Androidin, että iOSin teknologioiden osaamista. Kullakin alustalla käytettyjen kielten ero on siinä määrin merkittävä, että ei voida olettaa Java-kehittäjän osaavan myös vaikkapa Objective-C:tä. Näiden kielien osaajia on myös alalla vähemmän kuin JavaScript-osaajia; Stack Overflown suorittaman kyselyn vastanneista n. 70 % kertoi käyttävänsä JavaScript-ohjelmointikieltä, kun taas Javalla vastaava lukema oli n. 45 %, Objective-C:llä 7 %, sekä iOS-alustan toisella tuetulla kielellä, Swiftillä, lukema oli n. 8 %. /16/

Kun otetaan huomioon myös se, että React Nativella voidaan kehittää useammalle alustalle sovellukset yhteisellä koodikannalla, voidaan perustellusti väittää, että ainakin suurimmissa osassa tapauksista useammalle alustalle vaadittava sovellus on kustannustehokkaampaa kehittää käyttäen React Native -ohjelmointikehystä.

5.2 Johtopäätökset

Työn tavoitteena oli havainnollistaa mobiilikehitystä React Native -teknologialla sekä vertailla sitä mobiilikehitykseen natiivikielillä. Taustatyön ja referenssiprojektin kehityksen myötä ymmärrys React Nativesta, Android-sovelluksista sekä mobiilikehityksestä kasvoi.

Opinnäytetyöstä toimitettiin asiakkaalle valmis sovellus molemmille pyydetyille alustoille, sekä kattava dokumentaatio. Sovellukselle asetetut määrittelyt ja tavoitteet täyttyivät.

Työn aikana kerrytetyn tiedon perusteella vaikuttaisi siltä, että joistain ongelmakohdista huolimatta React Native on erittäin hyödyllinen ja kustannustehokas ohjelmistokehys, kun mobiilisovellusta tarvitaan useammalle kuin yhdelle alustalle.

LÄHTEET

/1/ https://www.statista.com/statistics/269025/worldwide-mobile-app-revenue-forecast/

Viitattu 8.5.2019

/2/ https://www.statista.com/statistics/266136/global-market-share-held-by-smartphone-operating-systems/

Viitattu 8.5.2019

/3/ https://reactjs.org/

Viitattu 9.5.2019

/4/ https://reactjs.org/docs/faq-state.html

Viitattu 9.5.2019

/5/ https://reactjs.org/docs/components-and-props.html

Viitattu 9.5.2019

/6/ https://reactjs.org/docs/faq-internals.html

Viitattu 9.5.2019

/7/ https://programmingwithmosh.com/react/react-virtual-dom-explained/

Viitattu 11.5.2019

/9/ https://facebook.github.io/react-native/docs/tutorial

Viitattu 16.5.2019

/10/ Bonnie, E. 2017. Learning React Native. Toinen painos. California. O'Reilly Media, Inc.

/11/ https://redux.js.org/api/store

Viitattu 16.5.2019

/12/ https://stackoverflow.com/questions/41804855/does-react-native-have-a-virtual-dom

Viitattu 8.5.2019

/13/ https://www.youtube.com/watch?v=8va9prUqjnA

Sjölander, E. React Native, The Native Bits. 2019. Wroclaw, Poland.

Viitattu 8.5.2019

/14/ https://fi.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

Viitattu 16.5.2019

/15/ https://semver.org/

Viitattu 16.5.2019

/16/ https://insights.stackoverflow.com/survey/2018

Viitattu 16.5.2019