Ville-Pekka Sarkkinen

UNIVERSAALIN SOVELLUKSEN TUOTANTOYMPÄRISTÖN KEHITYS

Opinnäytetyö
CENTRIA-AMMATTIKORKEAKOULU
Tieto- ja viestintätekniikka koulutusohjelma
Kesäkuu 2020

TIIVISTELMÄ OPINNÄYTETYÖSTÄ

Centria-ammattikorkeakoulu	**Aika** Kesäkuu 2020	**Tekijä/tekijät** Ville-Pekka Sarkkinen
Koulutusohjelma Tieto- ja viestintätekniikka		
Työn nimi UNIVERSAALIN SOVELLUKSEN TUOTANTOYMPÄRISTÖN KEHITYS		
Työn ohjaaja Sakari Männistö		**Sivumäärä** 19
Työelämäohjaaja Juho Muuraiskangas		

Tämän opinnäytetyön aiheena oli suunnitella ja toteuttaa kehitysympäristö, jolla kyetään tuottamaan ja jakelemaan yhteisestä ohjelmakoodista sovellus mahdollisimman monelle alustalle. Alustoina olivat Android, iOS, Windows, Linux ja macOS. Aiheeseen kuului myös tehdä pieni sovellus tuotantoympäristöllä. Tavoitteena on, että sovelluksen tuottaminen ja jakelu yhteisestä lähdekoodista voidaan automatisoida tuotantoympäristön avulla.

Ensimmäisessä osassa käsitellään sovelluksen kannalta tärkeimpiä tekniikoita, joiden avulla mahdollistettiin yhteisen lähdekoodin käyttö. Toisessa osassa esitellään sovellus ja kerrotaan sen kehittämisestä. Kolmannessa osassa esitellään tuotantoympäristön kannalta tärkeimpiä tekniikoita ja kuvaillaan tuotantoympäristön tekoa.

Tuotantoympäristön teko onnistui, Tuotantoympäristöllä kyetään tuottamaan ja jaella universaali sovellus.

Asiasanat
Android, Electron, Fastlane, iOS, Linux, macOS, React, React Native, React Native For Web, TravisCI, Windows.

ABSTRACT

Centria University of Applied Sciences	**Date** June 2020	**Author** Ville-Pekka Sarkkinen
Degree programme Information Technology		
Name of thesis DEVELOPMENT OF UNIVERSAL APPLICATION DEVELOPMENT ENVIRONMENT		
Instructor Sakari Männistö		**Pages** 19
Supervisor Juho Muuraiskangas		

Subject of this thesis is to design and produce a development environment, which enables the development and distribution of an application to as many platforms as possible. The platforms were: Android, iOS, Windows, Linux and macOS. Subject also included small application using this development environment. The aim is to automate applications development and distribution from one source code.

In the first part of this thesis the most important technologies for the application are introduced. The second part introduces the application and describes its development. The third part introduces the most important technologies for the development environment and describes its development.

Production of the development environment was successful. Development enviroment enables development and distribution of universal applications.

Key words
Android, Electron, Fastlane, iOS, Linux, macOS, React, React Native, React Native For Web, TravisCI, Windows

KÄSITTEIDEN MÄÄRITTELY

POC	Proof Of Concept on tietyn asian tai idean osoittaminen toteuttamiskelpoiseksi
MVP	Minimum viable product on tuote, joka saavuttaa asiakkaan minimivaatimukset
Html	Hypertext Markup Language on web-sivujen rakennetta kuvaava kieli
DOM	Html DOM eli Document Object Model on puu-olio, jonka avulla html renderöidään netti-sivulle
TestFlight	Applen sovelluskaupan testiversio testissä oleville sovelluksille
Push-notifikaatio	Notifikaatio laitteeseen, joka voidaan luoda joko sovelluksen sisällä tai ulkoisen palvelun avulla
Provisioning-profiili	Applen metadataa

TAULUKOT

1 JOHDANTO

Opinnäytetyöni tarkoituksena oli tutkia, miten onnistuu sovelluksen kehittäminen useammalle alustalle yhteisellä lähdekoodilla, ja kehittää yksinkertainen sovellus. Työssäni etsin vastauksia seuraaviin kysymyksiin: Miten kehittäminen onnistuu? Mitä tekniikoita se vaatii? Onko yhteisen lähdekoodin useammalle alustalle tekemisestä hyötyä? Sovelluksena toimi asiakasvastaavien sparraus käyttöliittymä. Sovellusta kehitettiin Alfame Systems Oy:lle. Sovelluksen tarkoitus oli tuoda asiakasvastaaville olennaista tietoa viikottaisiin palavereihin.

Asiakasvastaavat on jaettu useampaan ryhmään, jotka tapaavat viikottain. Ryhmien kokoonpanot muuttuvat viikottain, minkä tarkoituksena on jakaa tietoa ja taitoa eri ryhmien välillä. Olennaista tietoa asiakasvastaaville on tieto projekteihin käytetyistä tunneista ja kauppojen tilat.

Aluksi esittelen käyttämiäni tekniikoita ja sen jälkeen kuvailen toteutusta. Lopuksi arvioin työssä saatuja tuloksia verrattuna tavoitteisiin.

2 KÄYTETYT TEKNIIKAT

Koska opinnäytetyön tarkoituksena oli kehittää ja luoda tuotantoympäristö, joka mahdollisti sovelluksen tuottamisen useaan ympäristöön, piti käytettyjen tekniikoiden tukea tätä tavoitetta. Tässä osassa esittelen sovelluksen kannalta tärkeimpiä tekniikoita, jotka mahdollisti yhteisen lähdekoodin käytön kaikissa ympäristöissä.

2.1 React

React on deklaratiivinen JavaScript-kirjasto, joka on tarkoitettu käyttöliittymien rakentamiseen. Deklaratiivinen tarkoittaa, että kehittäjä kuvaa koodilla, minkälaisen lopputuloksen hän haluaa, ja deklaratiivinen kirjasto toteuttaa tämän. (React 2020a.) Tässä luvussa esittelen Reactin kättöön liittyvät tärkeimmät asiat ja kuvaan Reactin liittyvät yleisimmät kirjastot, jotka perustuvat Reactiin. Lopuksi kuvaan vielä kirjastoa, joka on keskeisessä osassa tässä opinnäytetyössä.

2.1.1 JSX

React-koodi on JavaScriptiä. Reactin perusfunktio on React.createElement, jonka parametreihin kuvataan, miltä elementin halutaan näyttävän, ja se palauttaa React-elementin. React-elementit ovat objekteja, jotka kuvaavat, mitä käyttöliittymän halutaan näyttävän (React 2020b). Kuvassa 1 on koodiesimerkki React.createElement-funktion käytöstä.

```
1  import React from 'react';
2
3  const element = React.createElement(
4      'h1',
5      {className: 'boldTitle'},
6      'Hello, world!'
7      );
8
```

KUVA 1. Esimerkki React.createlement-funktiosta

React.createElement voi mennä sekavaksi, koska sen sisään voi laittaa lapsia sisäisellä React.createElement-funktiolla. JSX on syntaksin laajennus Javascriptille. JSX-koodia on helpompi lukea kuin puhdasta JavaScriptiä. (React 2020b.) Kuvassa 2 on kuvassa 1 oleva koodi JSX-muodossa.

```
1  import React from 'react';
2
3  const element = (
4          <h1 className="boldTitle" >
5              Hello, world!
6          </h1>
7      );
8
```

KUVA 2. Perus JSX-esimerkki.

JSX sekoittaa JavaScriptiin käyttöliittymää kuvaavia elementtejä, kuten html-elementtejä mukailevat div ja h1. Vaikka React-elementti h1 näyttää samanlaiselta kuin html-vastaava h1, se ei kuitenkaan ole html-elementti. Esimerkkinä erosta on html-atribuutti class, joka on React-elementissä ”className”. Kuvassa 3 on esimerkki JavaScriptistä React-elementin sisällä.

```
1  import React from 'react';
2
3  const techsInUse = ['React', 'React Native'];
4
5  const element = (
6          <div>
7              We are using:
8              <ul>
9                  {techsInUse.map(tech => (
10                     <li>{tech}</li>
11                     )
12                 )}
13             </ul>
14         </div>
15     );
```

KUVA 3. Esimerkki JavaScriptistä JSX:n sisällä

Kuva 3 esittää miten Javascriptiä kirjoitetaan JSX:n sisällä. JavaScript eroitetaan React-elementeistä aaltosuluilla ({, }). Kuvassa 3 Javascript-funktio ”map” ottaa listan, lisää jokaisen sanan ympärille li-elementit ja palauttaa uuden listan. Näin ei tarvitse kirjoittaa listan sisältöä koodiin.

2.1.2 Virtual DOM ja rekonsiliaatio

Keskeisessä osassa Reactia on Virtual DOM, jonka tarkoituksena on välttää turhaa näkymän päivittämistä. Virtual DOM on olio muistissa, joka vastaa näkymää. Virtual DOMin nimi on harhaanjohtava, koska sillä viitataan selaimessa toimivaan DOMiin, vaikka Reactia voi käyttää muissakin ympäristöissä kuten mobiili. (React 2020c.)

Kun React-komponentin tila päivittyy, se palauttaa puu-olion, joka esittää, miltä komponentin ja sen lasten pitäisi näyttää. React-rekonsiliaatio on prosessi, joka vertaa tätä päivitystä Virtual DOMin kanssa ja päättää, mitä osia pitäisi päivittää. Ilman optimointia päivitysalgoritmin kompleksisuus olisi verrannollinen elementtien määrän kolmanteen potenssiin $O(n^3)$. Optimointisääntöjen avulla kompleksisuus vähenee tasolle O(n). (React 2020d.)

Ensimmäinen sääntö viittaa React -elementtien tyyppeihin. Jos elementin tyyppi on päivityksessä vaihtunut, rekonsiliaatio ei vertaile sitä ja olettaa, että se on muuttunut kokonaan. Tästä syystä React-elementtien tyyppien vaihtamista pitäisi välttää, jos näkymä kuitenkin pysyy samanlaisena.

Toinen sääntö viittaa listoihin. Toistettaville elementeille pitäisi antaa ainutlaatuinen key-atribuutti, jotta React pystyy tunnistamaan, mitkä elementit listalta ovat muuttuneet. Kuvassa 4 näkyy aikaisemmin kuvassa 3 näkynyt esimerkki muokattuna palvelemaan toista sääntöä.

```
import React from 'react';

const techsInUse = [{id: 1, name: 'React'}, {id: 2, name: 'React Native'}];

const element = (
        <div>
            We are using:
            <ul>
                {techsInUse.map(tech => (
                    <li key={tech.id} >{tech.name}</li>
                    )
                )}
            </ul>
        </div>
    );

```

KUVA 4 Kuvan 3 esimerkki muokattu palvelemaan rekonsiliaation toista sääntöä.

Kuvassa 4 map-funktio toistaa li-elementtiä, jonka sisällä näkyy teknologian nimi. Jotta koodin saisi sopimaan toisen säännön kanssa, pitää uloimmalle toistettavalle elementille antaa yksilöllinen key-atribuutti, kuten kuva 4 näyttää. Key-atribuutin arvon pitää olla yksilöllinen, eikä se saa vaihtua. Paras arvo key-atribuutille on datasta otettu id-arvo. Jos ei ole muuta mahdollisuutta, map-funktio antaa toisena parametrinä listan osan järjestysnumeron, jota voi käyttää key-atribuuttina. Järjestysnumeron käyttämistä ei suositella, koska se ei toimi hyvin, jos listaan lisätään tai sieltä poistetaan elementtejä. (React 2020d.)

React-rekonsiliaatio on uudelleen kirjoitettu arkkitehtuurilla nimeltä React fiber. React Fiber-arkkitehtuuri antaa rekonsiliaatiolle mahdollisuuden priorisoida renderöintiä, pysäyttää komponentin renderöinti, jatkaa aikaisemmin pysäytettyä komponentin renderöintiä ja hylätä komponentin renderöinti.
Renderöintien prioriteetissa voi esimerkiksi priorisoida käyttäjän syötteen tärkeämmäksi kuin API:lta palaavan datan renderöinti. Tällä voi välttää nykimistä esimerkiksi käyttäjän kirjoittaessa tekstikenttään, koska käyttäjän muutokset tiloihin ovat aina tärkeämpiä päivityksiä. (Clark 2016.)

2.1.3 ReactDOM

ReactDOM sitoo Reactin ja selaimen DOMin yhteen. ReactDOM kuului aikaisemmin Reactiin, mutta se erotettiin, koska Reactia voi käyttää muuallakin kuin selaimessa, kuten mobiilissa React Nativen kanssa. Monissa React-applikaatioissa ei käytetä ReactDOMia muuhun kuin applikaation sitomiseen

DOMiin ReactDOM.render-funktiolla. ReactDOM tarjoaa myös mahdollisuuden käsitellä DOMin osia mutta tämä tapahtuu Reactin ulkopuolella, eikä sitä suositella. (Raghav 2019.)

2.1.4 React Native

React Native on Facebookin kehittämä kirjasto, joka avattiin avoimeksi lähdekoodiksi kaksi vuotta Reactin jälkeen. Kirjaston tarkoituksena on tuoda Reactin edut mobiili-kehitykseen. Mobiilikehitys normaalisti tehdään joko Java, Kotlin (Android) tai Objective-C, Swift (iOS) ohjelmointikielillä. Eri kielistä johtuen applikaatiot, jotka ovat molemmilla ympäristöillä ja jotka toimivat molemmissa ympäristöissä, edellyttävät omat alustakohtaiset lähdeohjelmakoodit. React Native kääntyy molemmille ympäristöille, joten yksi lähdekoodi riittää. Mobiili-kehityksen ongelmana on myös, että kaikki muutokset edellyttävät sovelluksen uudelleengenerointia lähdekoodista. Tästä johtuen pienenkin muutoksen näkyminen ruudulla voi viedä useita minuutteja riippuen koodin koosta. Koska React Native on JavaScript-koodia, se pystyy päivittymään uusilla lähdekoodeilla sekuneissa. (Occhino 2015.)

Mobiili kehityksessä on jo aikaisemmin kokeiltu JavaScriptin avulla toimivaa yhteistä kehitystä, mm. WebView-komponentin avulla, joka on perjaatteessa nettisivu applikaation sisällä. Ongelmana tässä lähestymistavassa on, että se ei ole natiivi-toteutus. Se ei voi käyttää mobiilin natiiveja komponentteja, kuten kameraa, puhelinta ja albumia. React Nativella ei käytetä Reactin komponentteja, vaan React Nativen komponentteja, jotka sitten muutetaan vastaaviksi natiivi-komponenteiksi. (Occhino 2015.) Taulukko 1 näyttää eri komponentteja ja niiden vastaavuudet eri ympäristössä.

TAULUKKO 1. Komponentit eri ympäristöissä. (mukaillen React 2020e)

React Native	Android	iOS	Web
<View>	<ViewGroup>	<UIView>	
<Text>	<TextView>	<UITextView>	<p>
<Image>	<ImageView>	<UIImageView>	<img>
<ScrollView>	<ScrollView>	<UIScrollView>	
<TextInput>	<EditText>	<UITextField>	<input type=”text”>

2.1.5 React Native For Web

Opinnäytetyöni applikaatio-osuuden koodi kirjoitettiin suurimmalta osin React Nativella, mutta React Nativen koodi ei toimi muualla kuin mobiili-ympäristössä. Saman applikaation kirjoittaminen uudelleen Reactilla menisi kokonaan ohi tämän opinnäytetyön tarkoituksesta. Ratkaisuna oli React Native For Web-kirjasto. React Native For Web-kirjasto kääntää React Nativen komponentin web-vastaavaksi, joka kelpaa web-ympäristössä.

2.2 Electron

Tähän mennessä on applikaation kehitys yhdellä koodilla mahdollistettu ympäristöissä Web, Android ja iOS. Windows, Linux ja macOS applikaatioihin tarvitaan vielä yksi kirjasto lisää. Electron on kirjasto, joka on rakennettu Chromiumin ja Node.js avulla. Electronin avulla html ja Javascript koodi, joksi React-applikaatio kootaan, voidaan käyttää pöytäkoneessa omana sovelluksenaan. (Lord 2020.)

3 SOVELLUS

Opinnäytetyön alkuperäisenä tavoitteena oli kehittää Asiakasvastaava sparraus ui-sovellus Alfame Systems OY:lle. Aihe kuitenkin kehittyi koskemaan tuotantoympäristöä ja itse sovellus jäi POC/MVP-tasolle. Tässä osiossa esittelen sovelluksen tarkoitusta, toimintaa ja oleellista tietoa.

3.1 Asiakasvastaavat ja viikkosparraus-palaveri

Asiakasvastaava on asiakkaan yhteyshenkilö Alfamella. Asiakasvastaavan tarkoituksena on pitää yllä kommunikaatiota asiakkaisiin ja tunnistaa ja myydä lisäkehitystä asiakkaille. Asiakasvastaavat käyvät läpi asiakkaiden palautteita ja varmistavat, että nykyinen toteutus vastaa asiakkaan vaatimuksia. Palavereissa voidaan myös tunnistaa, jos asiakasyhteys on hiljenemässä ja miten siihen voidaan reagoida. Asiakasvastaavat pitävät kerran viikossa viikkosparraus-palaverin. Palaverit pidetään 3–5 hengen ryhmissä, ja niissä käydään läpi nykyisten asiakkaiden tilaa myyntiprosessissa. Ryhmien kokoonpanot ja roolit vaihtuvat viikottain, minkä tarkoituksena on varmistaa, että kaikki asiakasvastaavat ovat tekemisissä kaikkien muiden kanssa. Vaihtuvat roolit ja kokoonpanot varmistavat, että uudet ideat ja menetelmät pyörivät jokaisen asiakasvastaavan ulottuvilla.

3.2 Sovelluksen tarve ja ratkaisu

Viikkosparraus-palaverien vaihtuvista kokoonpanoista johtuen on asiakasvastaavien hankala pysyä kartalla muiden vastaavista projekteista ja asiakkuuksista. Palavereihin on varattu puoli tuntia aikaa tutustua näihin Planmillin ja HubSpotin välityksellä. Sovelluksen oli tarkoitus hakea ja esitellä viikkosparrauspalaverissa tarvittavia tietoja ryhmittäin asiakasvastaavan suosimalle alustalle. Näin asiakasvastaavien ei tarvitsisi käyttää aikaa palaverissa tai ennen palaveria tutustumiseen muiden asiakasvastaavien vastuualueiden tietoihin.

3.3 Robottimanageri

Robottimanageri on järjestelmä, joka hallitsee mm. viikkosparraus-palaverien järjestäytymistä. Robottimanageri hoitaa ryhmien kokoonpanot. Robottimanagerin käyttöliittymä näyttää vain ryhmien kokoonpanot. Asiakasvastaava sparraus ui oli jatkokehitystä robottimanagerille.

3.4 Integraatiot

Asiakasvastaaville tuotava tieto oli jaettu kahteen osuuteen: nykyisten asiakkaiden myyntiin liittyvää tietoa HubSpotista ja projektien kulusta kertovaa tietoa Planmillistä. Integraatio-osuus ei kuulunut opinnäytetyöni alueeseen, vaan integraation oli tarkoitus tulla eri tekijältä.

3.4.1 HubSpot Customer Relationship Management -järjestelmä

HubSpot on CRM eli Customer Relationship Management -järjestelmä. CRM:n tarkoituksena on tuoda yhteydenpito asiakkaaseen useamman henkilön saataville. CRM:ssä on asiakkaiden yhteystiedot, asiakkaan ja yrityksen myynnin välinen yhteydenpito ja myynnin tilanne asiakkaan kohdalla. (HubSpot, 2020.) HubSpotista oli tarkoitus hakea asiakasvastaavien alla olevia liidejä ja diilejä. Liidi on asiakkaan yhteystiedot ja alustava yhteydenpito. Diili on myyntiprosessissa oleva kohde, johon kuuluvat myynti, tarjoukset ja kilpailu.

3.4.2 PlanMill Enterprise Resource Management -järjestelmä

PlanMill on ERP eli Enterprise Resource Management -järjestelmä. ERP-järjestelmä hallitsee projekteja, toimeksiantoja ja niiden laskutusta. (PlanMill 2020.) PlanMillista oli tarkoitus tuoda asiakasvastaavien vastuulla olevien projektien työtuntitietoja, kuten kirjatut tunnit, hyväksytyt tunnit ja toimeksiannoilla jäljellä olevat tunnit.

3.5 Sovelluksen toteutus

Sovelluksen ympäristöriippumattomuus aiheutti sovelluksen koodauksessa jonkin verran rajoitteita. Tässä osassa kuvaan hieman eri versioiden eroavaisuuksia ja ongelmia. Sovelluksen toteutuksen aloitin käyttämällä create react appia (CRA), jossa on jo valmiiksi konfiguroitu projektipohja ja joka vastasi web ja Electron-versioiden asetuksista ja koonnista. Mobiiliversion tein normaalin React Native-projektin samaan kansioon CRA-projektin kanssa. Ympäristömuuttujiin minun piti lisätä ”SKIP_PREFLIGHT_CHECK=true”, koska CRA ja React Native käyttivät joistain kirjastoista eri versioita. Tämän avulla CRA:n skriptit ei valittaneet eri versioista. Jouduin myös väliaikaisesti siirtämään CRA:n .gitignore ja package.json tiedostot ennen React Nativen projektin aloitusta, koska se olisi ylikirjoittanut nämä. React Nativen aloituksen jälkeen yhdistin projektien .gitignore ja package.json tiedostot.

Index.js-tiedosto on normaalisti CRA ja React Native projektien koodin aloituskohteena, niin myös tässäkin tapauksessa. Index-tiedosto CRA:ssa sitoo sovelluksen htmlDOMiin ReactDOMin avulla ja React Nativen puolella se rekisteröi sovelluksen Appregistryyn. Näiden tiedostojen piti olla erillisiä, koska CRA:n index-tiedostossa oli yksi rivi enemmän. Index-tiedostot olivat myös eri kansioissa, CRA:n versio oli src -kansiossa kuten muukin koodi, mutta React Native versio oli projektin juuressa. Tämä johtui siitä, että CRA:n ja React Native projektin koontikoodit etsivät index-tiedostoja näistä paikoista. Aluksi muokkasin React Nativen koodia niin, että se osaisi etsiä myös React Nativen index-tiedostoa src-kansiosta. Kuitenkin päätin perua muokkauksen, koska se ei olisi toiminut automaattisessa tuotantoympäristössä.

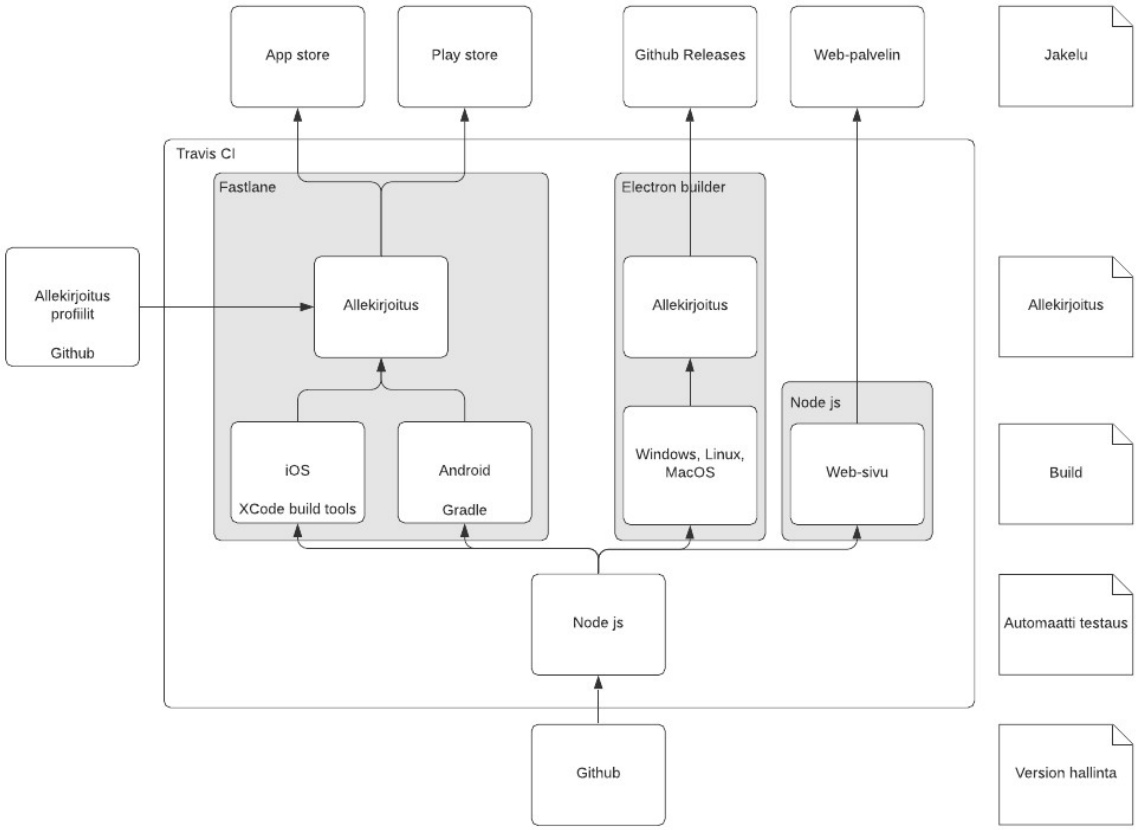
CRA-versiossa toimii Webpack-työkalu, joka kokoaa osista koostuvan ohjelmakoodin tarvittavine oheisresursseineen ajoympäristössä tarvittavaan muotoon. Halusin Webpackin toimivan myös React Native-versiossa. Tähän tarkoitukseen löysin kirjaston, joka kykeni lisäämään Webpackin React Native projektiin. Kuitenkin päädyin virheeseen, jota en yrityksesistäni huolimatta saanut korjattua. Jouduin hylkäämään Webpackin React Native-versiossa ja käyttämään React Nativen normaalisti käyttämää metro-builderia.

Itse koodin eroavaisuuksiakin löytyi, vaikka sovellus olikin pieni. Electron tarvitsee main-tiedoston, jossa on electronin toimintaan liittyvää koodia, kuten mitä tapahtuu, kun sovelluksen sulkee, ja mistä electron löytää Reactista kootun koodin. Navigointiin käytin React router-kirjastoa, jossa oli navigointi paketit Reactille ja React Nativelle. Koska molempien apit olivat samoja, sen käyttö onnistui.

Navigoinnin paketeiden tuonnin koodille piti tehdä hieman muutosta normaalista, että projekti osaisi käyttää oikeaa pakettia riippuen ympäristöstä. Väärän paketin käyttö johtaa virheisiin, koska paketin tavoittelemat natiivit API:t eivät ole ympäristössä.

4 TUOTANTOYMPÄRISTÖ

Opinnäytetyön pääasiallisena aiheena oli tuotantoympäristön suunnittelu, jonka tarkoituksena oli mahdollistaa kehitys ja jakelu useamman alustan sovellukselle. Tuotantoympäristön haluttiin toimivan nk. ”yhden napin painalluksella”. Tuotantoympäristöön tulisi kuulua versionhallinnan, automaattisen testauksen, sovelluksen kokoamisen, allekirjoituksen ja jakelun. Tässä osiossa käyn läpi tuotantoympäristön suunnittelua ja toteutusta. Alapuolella on kuvio tuotantoympäristöstä. Kuvio 1 on jaoteltu pystysuunnassa tuotantoympäristön vaiheiden perusteella, joiden nimet ovat oikeassa laidassa ja vastaavat tämän opinnäytetyön seuraavia otsikoita. Vaakasuunnassa kuvio 1 on jaoteltu ylimmän tason jakelukohteiden perusteella.



KUVIO 1. Tuotantoympäristön kaavio

4.1 Versionhallinta

Tuotantoympäristön versionhallintaan valitsin Github-versionhallintajärjestelmän. Syinä valintaan olivat oma kokemus järjestelmästä ja Githubin ja Travis CI:n välinen integraatio. Vaihtoehtoisia järjestelmiä olisi olleet Gitlab ja Bitbucket. Githubissa repositori oli jaettu kolmeen haaraan: master, staging ja production. Master-haara oli kehityshaara, johon meni uusimmat koodit. Staging-haara oli testi haara, joka kävi läpi testin, kokoamisen, allekirjoituksen ja jaon App store ja Google play -palveluiden testausjakeluihin. Production-haara kävi läpi testin, kokoamisen, allekirjoituksen sekä kaikkien versioiden jakelut.

4.2 Continuous Integration ja Continuous Deployment

Continuous Integration eli CI tarkoittaa jatkuvaa prosessia, jossa automaattisesti hoidetaan koodin testaus ja testaus, kun esimerkiksi versionhallintaan tuodaan uutta koodia. Continuous deployment eli CD tarkoittaa prosessia, johon kuuluu CI-prosessin lisäksi myös kootun sovelluksen jakelu. (Brent Laster, 2018.) CI/CD-palveluna päädyin käyttämään Travis CI-palvelua. Syinä valintaan olivat intergaatio Githubin kanssa ja palvelun ilmainen open source-versio.

Travis CI tarjoaa virtuaalikoneita, joissa itse CI/CD-prosessi tapahtuu. Virtuaalikoneita ohjataan YAML-kielellä kirjoitetulla .travis.yml-tiedostolla koodin juuressa. Tiedostolla voidaan ohjata useampia töitä, jotka suoritetaan toistensa jälkeen tai maksullisessa versiossa samanaikaisesti. Asetuksissa voidaan määrätä, mitä käyttöjärjestelmää, ohjelmistoja ja versioita halutaan käyttää. Käyttöjärjestelmän valinta voi olla tärkeä, koska esimerkiksi iOS- ja macOS-sovellusten koonti ja allekirjoitus voidaan tehdä ainoastaan macOS-käyttöjärjestelmällä. (Travis CI 2020.) CI/CD prosessi käy läpi 12 vaihetta, joihin annetaan ohjeistus .travis.yml-tiedostolla. Vaiheet ovat lisäosat, välimuisti, ennen asennusta, asennus, ennen koodia, koodi, ennen välimuistia, koodin onnistumisen tai epäonnistumisen jälkeen, ennen lähetystä, lähetys, lähetyksen jälkeen ja koodin jälkeen.

4.3 Fastlane

Fastlane on työkalupakki, joka tehtiin helpottamaan Applen tunnetusti hankalan ekosysteemin automatisointia iOS-kehityksessä. Fastlanella voi myös automatisoida Android-ekosysteemin. Taulukossa 2 on esitelty Fastlanen tarjoamat työkalut ja niiden selitykset. (Katz 2018.)

TAULUKKO 2. Fastlanen tarjoamat työkalut ja niiden selitykset.

Työkalu	**Selitys**
iOS	
Deliver	Kuvakaappauksien, metadatan ja applikaation lataaminen App Storeen.
Snapshot	Automaattisesti ottaa applikaatiosta kuvakaappauksia eri ruutu-kooilla ja kielillä, vastaamaan App Storen vaatimuksia.
Frameit	Kuvakaappauksien liittäminen iOS-laitteiden raameihin.
Pem	Push-notifikaatio-profiilien hallinta.
Sigh	Provisioning-profiilien hallinta.
Produce	Uuden applikaation luominen iTunes Connect ja Developer Portal-palveluihin
Cert	Koodin allekirjoitus-sertifikaattien automaattinen luominen ja hallinta.
Scan	iOS ja macOS applikaatioiden testaus.
Gym	iOS applikaatioiden koonti.
Match	Provisioning-profiilien ja allekirjoitus-sertifikaattien synkronisointi.
Pilot	TestFlight-testaajien hallinta.
Boarding	Luo testaajien kirjaantumiselle sivun.
Android	
Supply	Android-applikaatioiden ja niiden metadatojen päivitys Google Play Storessa
Screengrab	Automaattisesti ottaa applikaatiosta kuvakaappauksia eri ruutu-kooilla ja kielillä.

Vaikka Fastlane on tarkoitettu helpoittamaan Applen ekosysteemin kanssa toimimista, toi se silti virheitä ja vaikeuksia Fastlanen lävitse. Joitakin asioita, joita Fastlane tarjoaa, minun piti tehdä itse Applen ekosysteemissä, koska Fastlane epäonnistui. Fastlanea ohjasin kirjoittamalla fastfile-tiedoston, johon pystyin laittamaan eri polkuja, kuten testaus ja tuotanto, jotka veivät Android, iOS ja macOS sovellukset joko testi- tai tuotanto jakeluun, riippuen siitä, mille git-haaralle koodi vietiin.

Fastlanea pystyy käyttämään omalla kehityskoneella, mutta se tarjoaa myös integraatiota CI/CD-palveluiden kanssa. Travis CI:n oli tarkoitus käyttää Fastlanea, mutta joitakin kerran tehtäviä taskeja Travis ei pystynyt suorittamaan, joten ne suoritin manuaalisesti.

4.4 Automaattinen testaus

Travis CI käynnisti automaattisesti testauksen aina kun koodia tuli mille tahansa haaralle. Työ asensi Linux-virtuaalikoneelle Node js:n ja ajoi yksikkö- ja integraatio-testit. Epäonnistuessaan työ esti muiden vaiheiden töiden alkamisen.

4.5 Koonti

Koonti oli jaettu kolmeen eri työhön, jotka vastasivat eri alustojen koonnista. Android- ja iOS-sovellukset koottiin suoraan React Native-koodista Fastlane-työkalupakin avulla. Web-versio koottiin React Native-koodista react native for web-kirjaston avulla.

Windows-, Linux- ja macOS- sovelluksen koonti erosi muista, koska ne koottiin kahteen kertaan. Ensimmäisenä koottiin React-koodista puhdas JavaScript-koodi, joka oli web-sivun koodi. Sen jälkeen otettiin mukaan Electron-spesiifinen kooditiedosto ja koottiin se Electron-builder-työkalun avulla.

4.6 Allekirjoitus

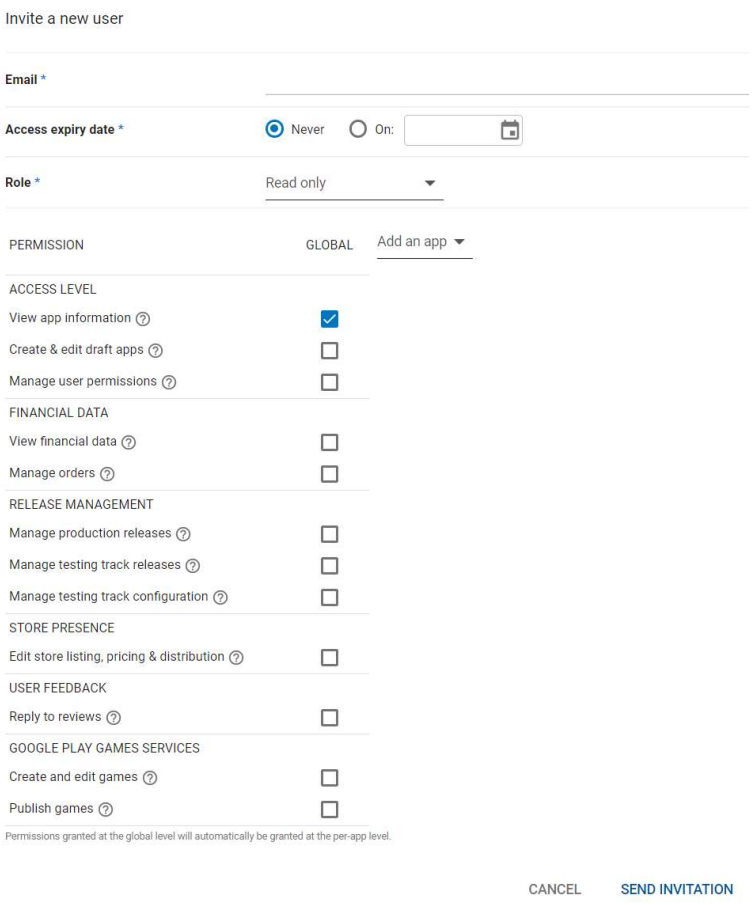
Sovelluksen allekirjoituksen on tarkoitus suojella sovelluksen tekijää ja käyttäjää. Käyttäjän alusta tarkistaa allekirjoituksen avulla, että tekijä on luotettu ja että sovelluksen päivitys on tullut samalta tekijältä. App store ja Google play store ei salli sovellusten lataamista palvelimille ilman allekirjoitusta.

4.7 Jakelu

Jakeluun liittyi paljon asiaa johtuen siitä, että sovelluksia oli kuusi kappaletta ja puolet sovelluksista meni eri jakeluihin. Tässä osiossa käyn läpi eri sovellusten jakeluita.

4.7.1 Android-sovelluksen jako Google Play-palvelussa

Android-jakeluun käytettiin Google Play-kauppaa. Play-kaupan hallinnoimiseen käytetään Google Play Console-sivustoa. Sivuston käyttämiseen tarvitaan Google Developer-käyttäjä, jonka saa 25 dollarin kertamaksulla. Google Play Consolella voi hallita useampia käyttäjiä ja sovelluksia ja tarkastella sovellusten statistiikkaa. Käyttäjiä voi kutsua ja antaa niille oikeuksia. Seuraavassa kuvassa (KUVA 5) näkyy käyttäjän kutsumislomake.



KUVA 5 Käyttäjän kutsuminen Google Play Consolessa.

Kutsumisen lisäksi Consolessa voi käyttää Service Account-käyttäjää. Service Account-käyttäjät saavat oikeuden Google Play Console API:n käyttöön, joten ne on tarkoitettu automaattiseen käyttöön, kuten CD:n käyttöön. Google Play Console ei voi luoda käyttäjiä, vaan se piti tehdä Google Cloud Platformissa. Luotaessa Service Accountia on mahdollista luoda salainen avain, jota mm. Fastlane

käyttää tunnistautumiseen Google Console API:lle. Tein Fastlanelle oman Service Accountin, salaisen avaimen ja annoin sille Release Management-oikeudet.

Google Play Console tarjoaa julkaisuun neljä eri tietä: julkaisu, avoin testi (beta), suljettu testi (alfa) ja sisäinen testi. Ennen kuin voin julkaista, minun piti tehdä uusi sovellus Consolessa ja asettaa sille pakolliset tiedot. Fastlane ei pysty julkaisemaan ensimmäistä sovellusta, joten tämä piti tehdä käsin. Käytin Fastlanea omalla koneella luodakseni kootun ja allekirjoitetun sovelluksen. Latasin sovelluksen Google Play Consoleen.

Jokaisen sovelluksen version latauksen jälkeen Play Console käy läpi sovelluksen testauksen ja antaa siitä raportin. Testi on jaettu viiteen osioon: stabiilisuus, suorituskyky, esteettömyys, kuvankaappaukset ja turvallisuus. Stabiilisuus-testissä sovellusta testataan 10 laitteella, että se käynnistyy eikä kaadu. Suorituskykytestissä robotti käyttää sovellusta ja mittaa mm. muistin käyttöä. Kuvankaappausosiossa robotti ottaa kuvankaappauksia sovelluksesta erikokoisilla ja resoluutioisilla laitteilla, joista voidaan tarkastaa, että sovellus näyttää oikealta eri laitteissa. Testin mentyä läpi se voidaan julkaista. Ensimmäisen julkaisun jälkeen Fastlane voi suorittaa julkaisut.

4.7.2 iOS-julkaisu App Store-palvelussa

iOS-sovelluksen jakeluun käytettiin Applen App Store-palvelua. Jotta App Storessa pystyi julkaisemaan, minun piti liittää appleId-käyttäjä Apple Developer Programiin. Apple Developer Program vaati D-U-N-S-numeroa ja 100 dollaria vuodessa. D-U-N-S-numero eli Data Universal Numbering System on The Dun & Bradstreetin yrityksille tarjoama uniikki numero, jonka avulla voi etsiä perustietoja yrityksestä (Dun & Bradstreet 2020). Hain D-U-N-S-numeroa, koska Alfamella ei ollut semmoista. D-U-N-S-numeron saatuani hain Apple Developer Programiin. Kun Apple oli tarkastanut esimieheltäni, että minulla oli oikeus liittyä yrityksen nimissä, pääsin vihdoinkin sisään.

Apple Developer Program tarjoaa julkaisuoikeuden ja statistiikkaa. Kuten Android puolella iOS-puolella on myös mahdollisuus hallita useampia sovelluksia ja käyttäjiä. Toisin kuin Google Play Consolessa, Apple-puolella Fastlane pystyi suorittamaan ensimmäisen julkaisun.

4.7.3 Windows, macOS ja Linux jakelu GitHub-releasessa

Mahdollisuuksina Windows-, macOS- ja Linux- sovellusten jakelulle harkitsin seuraavia vaihtoehtoja: Windows store (windows ainoastaan), oma jakelu omalla palvelimella ja GitHub releases. Windows storen hylkäsin, koska halusin pitää kaikki versiot samassa jakelussa. Oman jakelun hylkäsin, koska Alfamella ei ollut omaa jakelupalvelua. Päädyin valitsemaan Github releases-palvelun, koska se oli jo olemassa ja se onnistui helposti Travis CI:llä.

5 YHTEENVETO

Tämän opinnäytetyön tarkoituksena oli luoda tuotantoympäristö, jonka avulla kykeni kehittämään sovelluksen yhteisellä lähdekoodilla useampaan ympäristööön ja luomaan sillä POC/MVP-sovellus. Opinnäytetyön aiheen vaihtuminen mobiilisovelluksesta tuotantoympäristön toteutukseen oli hyvä muutos. Olin jo ennestään tehnyt melko paljonkin React- ja React Native- koodausta, ainoa uusi asia pelkän sovelluksen tekemisestä olisi ollut sovellusten jakelu. Uudella aiheella sain mukaan myös Electron, React Native For Web ja CI/CD-prosessin, jotka eivät olleet minulle muutoin kuin nimeltä tuttuja.

Tuotantoympäristön kehittäminen ja sen avulla sovelluksen kehittäminen onnistui kohtuullisen sujuvasti. Sovelluksen aloitus ja teko ei tuntunut ihan luistavalta kehitykseltä, kuten sovelluksen toteutuksen osiossa kuvailin. Tuotantoympäristön tärkeimpinä tekniikoina olivat React-native ja React yhteisen lähdekoodi perheen kielet, React Native for web näiden yhdistämiseen ja Travis CI tai vastaava CI/CD-palveluautomatisoinnin suorittamiseen. Hyötyinä yhteisen lähdekoodin kehittämisestä usealle alustalle on, että versiot vastaa toisiaan eikä resursseja tarvitse välttämättä yhtä paljon.

Uskon, että tämän kaltaisella kehityksellä on tulevaisuus, koska universaalille sovelluksille on aina ollut kysyntää. React Native For Web on jo tuotantokäytössä useammallakin isolla yhtiöllä, mikä on monesti hyvä merkki. Myös Expo, joka on Create React Appin vastaava React Nativen puolella, on alkanut tukemaan React-toteutusta React Native For Web-kirjaston avulla. En käyttänyt Expoa, koska web-tuki tuli Expoon sen jälkeen, kun olin jo työn tehnyt.

LÄHTEET

Clark, A. 2016. React Fiber Architecture. Saatavissa: https://github.com/acdlite/react-fiber-architecture Viitattu 29.4.2020

Duns & Bradstreet. 2020. Dun & Bradstreet D U N S® Number. Saatavissa: https://www.dnb.com/duns-number.html Viitattu 15.03.2020

HubSpot. 2020. What is a CRM. Saatavissa: https://www.hubspot.com/growth-stack/what-is-crm Viitattu 12.2.2020

Katz, D. 2018. What is Fastlane. Saatavissa: https://subscription.packtpub.com/book/application_development/9781788398510/1/ch01lvl1sec11/what-is-fastlane Viitattu 3.5.2020

Laster, B. 2018. What is CI/CD. Opensource. Saatavissa: https://opensource.com/article/18/8/what-cicd Viitattu 23.2.2020

Lord J. 2017. Essential Electron. Saatavissa: https://jlord.us/essential-electron/ Viitattu 3.5.2020

Occhino, T. 2015. React Native: Bringing modern web techniques to mobile. Saatavissa: https://engineering.fb.com/android/react-native-bringing-modern-web-techniques-to-mobile/ Viitattu 3.5.2020

PlanMill. 2020. ERP. Saatavissa: https://www.planmill.com/fi/erp/ Viitattu 13.2.2020

Raghav, R. 2019. React vs. React-DOM. Saatavissa: https://medium.com/programming-sage/react-vs-react-dom-a0ed3aea9745 Viitattu 30.4.2020

React. 2020a. React docs. Saatavissa: https://reactjs.org Viitattu 25.4.2020

React. 2020b. Introducing JSX. Saatavissa: https://reactjs.org/introducing-jsx.html Viitattu 27.4.2020

React. 2020c. Virtual DOM and Internals. Saatavissa: https://reactjs.org/docs/faq-internals.html Viitattu 28.4.2020

React. 2020d. Reconciliation. Saatavissa: https://reactjs.org/docs/reconciliation.html Viitattu 28.4.2020

React. 2020e. Core Components and Native Components. Saatavissa: https://reactnative.dev/docs/intro-react-native-components Viitattu 3.5.2020

Travis CI. 2020. Documents. Saatavissa: https://docs.travis-ci.com/ Viitattu 23.2.2020