Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Samuli Lindroos

Sopimustietokannan hallintatyökalun React Native -prototyyppi

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

20.5.2021

Tekijä Otsikko Sivumäärä Aika	Samuli Lindroos Sopimustietokannan hallintatyökalun React Native -prototyyppi 25 sivua 20.5.2021
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Mediatekniikka
Ohjaaja	Lehtori Ilkka Kylmäniemi

Insinöörityössä perehdyttiin mobiilisovelluksen prototyypin kehittämiseen ja sen teknologioihin ja tehtiin prototyyppi React Native -kehitysympäristössä. Prototyypin ideana oli toimia sopimustietokannan hallintatyökaluna. Käyttäjän tuli pystyä kirjautumaan tililleen, hallitsemaan oman yrityksensä sopimuksien uusimista ja selaamaan omia sopimuksiaan. Prototyypin idea tuli työn tilanneelta yritykseltä.

Prototyypin backendinä toimii SQLServer-tietokanta ja ASP.net-palvelin. Frontend tehtiin Facebookin tuottamalla React Native -kirjastolla, joka pohjautuu React-kirjastoon, eli prototyyppi kehitettiin komponenttipohjaisena sovelluksena. Työssä tutkittiin myös, miten mobiilisovelluksien hybridikehittäminen toimii, ja sitä hyödyntämällä prototyyppi voitiin kehittää sekä Android- että iOS-alustoille.

Työn tuloksena saatiin kehitettyä sovelluksen prototyyppi, johon saatiin kehitettyä kaikki työn tilaajan haluamat toiminnallisuudet.

Prototyypin kehityksessä ei ilmennyt suurempia ongelmia, joten insinöörityön koodausosuus hoitui alun teknologioiden opiskelun jälkeen hyvin. Kehittäminen sujui helposti React-kehitysympäristön entuudestaan osaamisen myötä, mutta mobiilikehittämisessä tuli esille paljon uutta, esimerkiksi web- ja mobiilikehittämisen eroista, ja mobiilisovelluskehityksen parhaita käytäntöjä.

Avainsanat	React, React Native, mobiilikehittäminen, protototyyppi

Author Title Number of Pages Date	Samuli Lindroos Creating a agreement database handling tool with React Native 25 pages 20 May 2021
Degree	Bachelor of Engineering
Degree Programme	Information and communications technology
Professional Major	Media technology
Instructor	Ilkka Kylmäniemi, Senior Lecturer

This thesis consists of creation of a mobile application prototype in React Native environment and going through the required technologies to pull it off. The idea of the prototype is to function as an agreement bank handling tool. It has to offer the user a login, function to scroll through the user's business' agreements, and offer a functionality to renew them. The idea for the prototype came from the employer.

Thesis goes through the used tools in mobile development, in back- and frontend. Backend consists of SQLServer database and ASP.net server. Front end is created using Facebook's React Native -library, which is based upon React-library. This means that the prototype will be coded in React component style. The thesis also goes through how hybrid mobile development works, and how it is used to produce a prototype to iOS and Android platforms.

As a result of this thesis a functional React Native prototype was developed, which included all the functionalities that were desired.

The process of creating the prototype went smoothly without any big hiccups. So the coding went easy after learning the technologies. The fact that React development was known to me beforehand, was a big help, but I also learned a lot about mobile development.

Keywords	React, React Native, mobile development, prototype

Sisällys

Lyhenteet

OOP Object oriented programming. Oliomallin mukainen ohjelmointiparadigma.

JS (JavaScript) Internetin ohjelmointikieli. Alun perin kehitetty tukemaan web-sivustojen toiminnallisuutta.

HTML Hypertext Markup Language. Internetsivun ”selkäranka”.

CSS (Cascading Style Sheets) HTML-tiedoston ulkoasun ja asemoinnin kattava kieli.

ES2016 ECMAScript 2016. Uusin JavaScript-standardi.

React Facebookin tuottama frontend-JavaScript-kirjasto.

React Native Facebookin tuottama mobiilikehityskirjasto. Perustuu React-komponentteihin ja kääntyy iOS- ja Android-alustoille.

JSX (JavaScript XML) Reactissa käytössä oleva JavaScript-syntaksi.

Java Ohjelmointikieli, jota käytetään natiiviohjelmien kehittämiseen Androidille.

Application programming interface (API) Backendin osa, joka vastaa frontendin ja tietokannan välisestä kommunikaatiosta.

DOM (Document Object Model) Tapa, jolla HTML-tiedoston rakenne on selaimen luettavissa.

VDOM Virtual DOM Reactin ylläpitämä virtuaalinen, ”näkymätön”, versio DOM:sta.

1 Johdanto

Insinöörityön tarkoituksena on tutkia React Native -teknologian soveltuvuutta yrityksen mobiilikehitykseen. Lopputuotoksena kehitetään prototyyppi jo olemassa olevasta web-sovelluksesta mobiiliversiona React Native -teknologiaa hyödyntäen. Mobiilikehitys oli minulle työtä aloitettaessa entuudestaan tuntematon kehitysympäristö, mutta React, johon React Native pohjautuu, ja sen komponenttipohjaisuus olivat jo tuttuja.

Työssä tarkastellaan mobiilisovellusten kehitykseen liittyviä teknologioita ja työn toteutuksessa käytettyä kehitysympäristöä ja kuvataan prototyypin toiminnan perusteet. Prototyypin backend kehitettiin minulle uusilla teknologioilla, mutta niiden toimintamalli oli tuttu.

Prototyypin tarkoituksena on tarjota käyttäjälle sovellus, jonka avulla hän voi hallita yrityksensä sopimuksia asiakasyrityksen kanssa. Sinne tulee pystyä kirjautumaan, näkemään omat sopimuksensa helposti selattavassa muodossa ja tarjoamaan mahdollisuus sopimuksien uusimiseen. Prototyypin idea tuli työn tilanneelta yritykseltä.

Työn salassapitovelvoitteiden takia kuvista ja teksteistä on jätetty pois kaikki velvoitteiden sisältämät osat.

2 Mobiilisovellusten teknologiat ja kehitysympäristöt

2.1 Mobiilisovellusten taustatietoa

Mobiilimarkkinat ovat kasvaneet alkuajoistaan, ja niiden liikevaihdon arvellaan kasvavan 10,5 miljardista (2016) 25 miljardiin dollariin vuoteen 2024 mennessä (Mobile Marketing Makert 2019). Käyttäjiä on samalla aikavälillä tullut 2,1 miljardista 2,87 miljardiin (The mobile app market analysis: current and future trends 2017). Mobiilisovellusmarkkinoiden arvellaan kasvavan 106,27 miljardista (2018) 407 miljardiin dollariin vuoteen 2026 mennessä (Mobile Application Market Size Expected to Reach USD 407.31 Billion by 2026 – Valuates Reports 2020). Ennusteet ovat hurjia, sillä mobiilisovellusmarkkinoiden kasvun ennustetaan olevan lähes 400 % kahdeksassa vuodessa.

Jatkuvaa kasvua voi selittää elämäntyylimuutoksella, kun modernit älypuhelimet tulivat markkinoille. Älypuhelimet toivat mukanaan jatkuvan yhteyden normalisoinnin, mikä tarkoittaa, että oletusarvoisesti voi saada yhteyden kehen tahansa vain nappia painamalla, valittua kommunikaatioväylää pitkin. Tulevaisuuden ennustuksia markkinoiden kasvusta tukevat tekoälyn ja virtuaalitodellisuuden ja lisätyn todellisuuden kasvavat käyttötarkoitukset sekä big datan analytiikan tärkeys. Ennustusten mukaan suurin kasvu markkinoilla on pienten ja keskisuurien yritysten joukossa. Tätä perustellaan markkinoinnin mahdollisuuksilla mobiiliympäristössä. Mobiilikehittämisestä on tullut helpompaa, jolloin pienemmätkin yritykset saavat isomman hyödyn mobiilimarkkinoinnista. (Mobile Marketing Makert 2019.)

Älypuhelinmarkkinoiden käyttöjärjestelmät jakautuvat periaatteessa kahteen vaihtoehtoon: 72 % markkinoiden älypuhelimista toimii Android-käyttöjärjestelmällä ja 27 % taas Applen iOS-käyttöjärjestelmällä (Mobile Operating System Market Share Worldwide 2021). Tämä tarkoittaa sitä, että natiivi mobiilisovellus tulee kehittää molemmille alustoille toimivaksi. Mobiilisovelluksen kehittäminen Android-käyttöjärjestelmälle vaatii Javan tai Kotlinin osaamista, kun taas iOS-alustalle kehitetty sovellus on kirjoitettu Applen Swift-ohjelmointikielellä tai harvemmin Objective-C:llä. Tämä jako vaatii yritykseltä kehittäjiä, jotka joko erikoistuvat toiseen näistä tai opiskelevat tarvittavat taidot kehittääkseen sovelluksia molemmille alustoille.

Tässä insinöörityössä käytetty React Native mahdollistaa niin sanotun hybridikehittämisen molemmille alustoille. Tämä tarkoittaa, että sovelluksen koodaaminen tehdään samaa ohjelmointikieltä hyödyntäen tehtiin sovellusta sitten kummalle tahansa alustoista. Tätä teknologiaa hyödyntämällä yritys voi palkata mobiilikehittäjän, joka kykenee kehittämään sovelluksen, joka kääntyy molemmille alustoille. React Native -hybridikehitys säästää siis yritykseltä rahaa, koska kehittäjien ei tarvitse olla erikoistuneita yhteen käyttöjärjestelmään. Hybridikehityksessä syntyvä sovellus vastaa toiminnallisuuksiltaan natiivia sovellusta, sillä React Native -koodi kääntyy natiiviksi koodiksi ”konepellin alla”.

2.2 Frontend ja backend

Niin web- kuin mobiilisovellukset jaetaan frontendiin ja backendiin. Frontend tarkoittaa käyttäjälle näkyvää osaa ja sen toimintoja, kun taas backend tarkoittaa sovelluksen perustaa, jota hyödyntämään frontend on suunniteltu. Backend pitää sisällään yleensä tietokannan, sen API:n ja esimerkiksi palvelinasetukset. Frontend koostuu web-sovelluksissa pääasiassa HTML-, CSS- ja JavaScript-kokonaisuudesta. HTML määrittelee oliot, jotka sivulla piirtyvät, CSS määrittelee asemoinnin ja värit, ja JavaScript määrittelee sivuston toiminnallisuudet. JavaScript mahdollistaa http-pyyntöjen lähettämisen internetin välityksellä backendiin, jossa on päätepisteet kyseiselle pyynnölle. Näistä lisää tietoa luvussa 4.1. Backend suorittaa kysytyn pyynnön ja palauttaa frontendiin pyynnön mukaisen vastauksen.

2.3 Frontend-teknologiat

JavaScript

JavaScriptiä pidetään koko internetin ohjelmointikielenä, ja täten se onkin maailman laajimmin levinnyt ohjelmointikieli. Internetselaimet, mobiilipuhelimet ja tabletit käyttävät JavaScriptiä näyttäessään käyttäjälle moderneja internetsivustoja. JavaScript mahdollistaa sivuston elementtien toiminnallisuudet. Esim. käyttäjä voi tehdä http-pyyntöjä tietokantapalvelimelle tiedon hakua tai varastointia varten ja järjestää sitä haluamiensa kriteereiden mukaan. JavaScript on tyypittämätön korkean tason ohjelmointikieli, joka soveltuu niin funktionaaliseen kuin olio-ohjelmointiin.

JavaScript on kehitetty vuonna 1995 Netscape-nimisen yrityksen toimesta (Aston 2015). Nykyään JavaScript-nimi on rekisteröity Oraclelle. JavaScriptin standardisoitu muoto kulkee nimellä ECMAScript. Todellisuudessa kaikki käyttävät JavaScript-termiä tarkoittaessaan ECMAScriptiä. JavaSciptiä kehitetään jatkuvasti, ja uusin standardisoitu versio on ECMAScript 2020, mutta suurimmat muutokset ovat tulleet ES5:n (2015) mukana, jolloin JavaScriptiin tuli nuolifunktiot ja lupaukset.

JavaScriptin sisäinen API mahdollistaa esim. numeroiden, tekstin ja listojen manipulaation. Reititys, tiedon tallennus ja graafiset elementit jäävät JavaScriptin ulkopuolelle, jollei näihin erikoistuneita kirjastoja oteta käyttöön. Niiden käyttö vaatii lisäominaisuuksia alustalta, jossa JavaScript ajetaan. Tiedon tallennusta varten tulee olla yhteys tietokantapalvelimeen tai tulee käyttää laitteen sisäistä muistia. Reitittämistä varten on useita kirjastoja, jotka helpottavat kehittäjän työtä mahdollistamalla http-pyyntöjen suorittamisen.

React

React on Facebookin vuonna 2011 aloittama ja vuonna 2013 julkaistu komponenttipohjainen JavaScript-kirjasto (Banks & Porcello 2020), joka on vuoden 2019 jälkeen ollut markkinoiden suosituin JavaScript-kirjasto. React on suunniteltu helpottamaan käyttäjänäkymien kehittämistä, sillä React mahdollistaa omien, uudelleenkäytettävien ja tilansa muistavien komponenttien rakentamisen. Reactin ympärille on kehitetty erinäisiä lisätyökaluja. Niistä suurin osa on julkaistu vuosina 2015 ja 2016 (Banks Alex & Porcello Eve 2020). Lisätyökalujen käyttötarkoituksina on laajentaa React-kirjaston toiminnallisuuksia, esim. React Router mahdollistaa sivuston alasivujen välillä siirtymisen kehittämisen ja Reduxin, jonka avulla komponenttien tilanhallinta helpottui huomattavasti. Vuonna 2019 React julkaisi ”hookit”, joiden avulla komponenttien tilanhallinta helpottui ilman lisätyökaluja, ja tämän julkaisun jälkeen erittäin suosittu Redux jäi hieman taka-alalle, sillä suuri osa sen tuomista työkaluista oli sisällytetty Reactiin.

React on käytössä monella Fortune 500 -listalla olevalla yrityksellä. Esim. Uber, Twitter ja Airbnb käyttävät React-kirjastoa omien sovelluksiensa kehityksessä. Reactin kilpailevat kirjastot ovat AngularJS ja VueJS. Angular on Googlen kehittämä, ja Vue on yksityishenkilön nimeltä Evan You kehittämä avoin kirjasto.

Reactin komponenttipohjaisuus tarkoittaa sitä, että React-kirjasto antaa kehittäjälle mahdollisuuden luoda omia uudelleenkäytettäviä, uniikin tilansa muistavia ja uniikit ominaisuutensa omaavia komponentteja. Tämä onnistuu Reactin useState()-funktion ja komponentille annettujen props (properties) -arvojen avulla. UseState()-funktio liittää komponenttiin tilan, jonka päivittyessä tarvittavat osat komponentista piirtyvät selaimessa uudelleen. Tämä tuo helpotusta käyttäjän käyttämälle laitteelle, sillä piirto-operaatio on erittäin resurssi-intensiivinen, jolloin sen tarpeen minimoiminen tuo suuria tehokkuusparannuksia.

Selain piirtää komponentit DOM:iin (Document Object Model). React taas pitää kirjaa DOM:sta Virtual DOM:n (VDOM) muodossa. Virtual DOM on DOM:sta erillinen, Reactin ylläpitämä, virtuaalinen versio DOM:sta, ja se ei näy käyttäjälle. Tämä mahdollistaa DOM:n päivityksen vain osissa, joissa on eroavaisuuksia virtual DOM:n ja DOM:n välillä, eikä täten koko DOM:a tarvitse piirtää uudelleen.

React Native

React Native on Facebookin kehittämä avoimen lähdekoodin JavaScript-kirjasto, joka on suunnattu hybridimobiilikehitystä varten niin iOS- kuin Android-laitteille. React Native -sovellusta voi siis kirjoittaa JavaScript (JSX) -syntaksilla, ja se kääntyy natiivikoodiksi molemmille alustoille. React Native pohjautuu React-kirjastoon ja pyrkii tuomaan sen funktionaalisuudet mobiilikehitykseen. Ennen React Nativea oli muitakin hybridikehitystyökaluja, esim. Ionic, joka mahdollisti kehityksen iOS- ja Android-ympäristöihin, mutta React-kehittäjien määrän noustessa nousi myös kysyntä mobiilikehitykseen samalla kirjastolla. Tämä lisäsi React Nativen suosiota, ja se onkin nykyään yksi suosituimmista hybridimobiilikehitys-kirjastoista. React Nativen kehitystiimin ajatusmaailma on ”learn once, write anywhere”, mikä viittaa JavaScriptin monipuolisuuteen. (Abhishek & Paul 2019.)

VDOM toimii React Nativessa samaan tapaan kuin Reactissa, mutta VDOM:n ja DOM:n erojen laskun jälkeen React Native kutsuu laitteen natiivi-API:a palan nimeltä silta välityksellä. Silta muuttaa React Native -koodin kyseisen alustan natiivikoodiksi. iOS-käyttöjärjestelmällä tämä tarkoittaa olio C:tä ja Androidilla taas Javaa. Tämä toiminnallisuus saavutetaan käyttämällä React Nativen järjestelmäriippumattomia komponentteja. Esim. <View>-komponentti kääntyy sillan välityksellä iOS-ympäristössä <UIView>-muotoon ja

Android-ympäristössä <View>-muotoon. Suuri osa React Nativen komponenteista pohjautuu HTML-komponentteihin, syntaksieroavaisuuksilla.

JSX (JavaScript XML) on ECMAScriptin laajennos, jota React Native käyttää syntaksinaan. Se yhdistää logiikkaa ja mark-up-kieltä Tämä tarkoittaa, että sen ulkoasu on hyvin lähellä HTML-kieltä, mutta pitää sisällään JavaScriptin toiminnallisuudet.

2.4 Backend-teknologiat

SQLServer

Projektin tietokantana päädyttiin käyttämään SQL Server -kantaa. SQL Server on Microsoftin kehittämä relaatiotietokannan hallintajärjestelmä. Tämä järjestelmä oli minulle tuntematon, mutta se valittiin käyttöön siksi, että työn tilaaja käytti sitä.

Kannan sisältö tuli työn tilaajan esittelyprojektista. Sisältönä on monia sopimuksia, joita yritysten on tarkoitus pystyä uusimaan ennen takarajan umpeutumista tässä työssä tehdyn sovelluksen avulla. Tietokannan täyttämiseen käytettiin ”migraatioskriptiä”, jonka SQL Server osasi tehdä demoympäristöstä, ja tämä skripti kopioitiin sovelluksen kantaan.

ASP.Net

Tietokannan päälle täytyi rakentaa itse backend, jonka kautta sovellus pystyy kommunikoimaan tietokannan kanssa. Tähän käytettiin Microsoftin kehittämää ASP.Net-ohjelmointikieltä. Kieli oli minulle täysin uusi, mutta se oli melko helppo opetella tämän työn vaatimalle tasolle. Backendiin tehtiin muutama päätepiste, joiden avulla kannasta pystyy pyytämään sopimuksia yrityksen nimellä ja muuttamaan niiden status-kentän arvoa. Päätepisteen tarkoituksena on toimia välikätenä sovelluksen ja tietokannan välillä. Näistä lisää luvussa 4.1.

Ngrok

Ngrok on komentorivityökalu, joka mahdollistaa lokaalin portin avaamisen Ngrokin palvelinten kautta internetiin. Tämä helpottaa kehitystyötä mahdollistamalla API-kutsut mobiilisovelluksesta ilman kehitysympäristön verkkoasetuksien muuttamista. Ngrok antaa käyttäjälle URL-osoitteen, jonka kautta backendin palvelimen portti on auki koko internetille.

Ngrokin käyttö vaatii rekisteröitymisen sen palveluun. Tämän jälkeen käyttäjä saa tunnistevälineen (authentication token), joka kirjoitetaan komentorivikomennon avulla Ngrokin parametritiedostoon.

2.5 Kehitysympäristö

Insinöörityön kehitysympäristön käyttöjärjestelmänä toimi Windows 10. Koodin kirjoittamiseen käytettiin Visual Studio 2019 -ohjelmaa. Sovelluksen testaamiseen käytettiin Android Studion AVD-emulaattoria ja Google Pixel 3a- ja iPhone 7 -puhelimia.

Visual Studio 2019

Projektin aikana koodin kirjoittamiseen ja projektin hallintaan käytetty sovellus oli Visual Studio 2019. Se on Microsoftin kehittämä koodin editointityökalu, joka mahdollistaa koodin selkeän lukemisen ja editoinnin, projektin tiedostorakenteen hallinnan ja koodin linttauksen, eli koodin lukua helpottavien sääntöjen automaattisen toteutuksen, haluttujen asetusten mukaan. Olen aiemmin käyttänyt koodin kirjoittamiseen Visual Studio Code -nimistä ohjelmaa, joka on ilmainen, mutta yritys, jolle insinöörityö tehtiin, käyttää Visual Studio 2019 -ohjelmaa, joka on maksullinen yritystason kehitysympäristö.

Android Studio ja AVD Manager

Android Studio on IntelliJ IDEA -koodieditoriin pohjautuva kehitysympäristö, joka on suunnattu Android-sovellusten kehittämiseen. Se pitää sisällään Gradle-ohjelmaan pohjautuvan rakennusfunktionaalisuuden, joka mahdollistaa eri tarkoituksiin soveltuvien

asennuspakettien rakentamisen. Vaadittujen pakettien listaus ei ole Android Studio -projekteissa package.json-tiedostossa, vaan build.gradle-nimisessä tiedostossa.

AVD Manager (Android Virtual Device) on Android Studion sisällä oleva Android-ympäristön emulointityökalu. Sen avulla voidaan emuloida laajaa valikoimaa erinäisiä Android-laitteita. Sen avulla voi testata koodimuutokset sovelluksessa heti niiden tallennuttua.

Git

Projektin versionhallinta toteutettiin alalla jo vakiintuneeseen tapaan Git-komentoriviohjelman avulla. Git on ilmainen avoimen lähdekoodin ohjelma, joka mahdollistaa projektien versionhallinnan, joko lokaalisti kehityskoneella tai sen avulla voi työntää koodikannan johonkin ulkopuoliseen palvelimeen tallennettuun koodivarastoon. Git mahdollistaa myös projektin haarautuksen ja monen kehittäjän koodin yhdistämisen. Useimmiten Gitiä käytetään GitHub-palvelun yhteydessä, joka varastoi käyttäjän koodikannan omille palvelimilleen. Tämän insinöörityön yhteydessä yrityksen, ja täten myös tämän projektin, koodikanta tallennettiin Microsoft Azure -palveluun.

Node.js ja NPM

Node.js on JavaScript-pohjainen ajoympäristö, joka mahdollistaa JavaScript-koodin suorittamisen muuallakin kuin selainympäristössä. Tämä koodin suoritus tapahtuu samaa V8 -moottoria käyttäen kuin Chrome-selaimessa on käytössä. Nodea käytetään useimmiten Express.js -kirjaston kanssa luomaan helppokäyttöinen web-palvelin.

NPM on Noden mukana tuleva pakettienhallintajärjestelmä. Sen avulla voidaan asentaa lähes kaikki tarvittavat kirjastot ja lisäkomponentit projektiin. NPM-komennot tapahtuvat komentorivin kautta, ja paketin latauduttua NPM lisää sen ja sen version projektin packages.json-tiedostoon. Packages.json-tiedoston avulla sovellus saadaan asennettua eri kehitysympäristöihin helposti yhdellä NPM-komennolla, joka lataa kaikki tarvittavat paketit sovelluksen toiminnan varmistamiseksi. Tämän takia on tärkeää pitää packages.json-tiedostossa olevien pakettien vaaditut versiot ajan tasalla.

Puhelimet

Sovelluksen testaukseen käytettiin Google Pixel 3a (Android 11)- ja iPhone 7 (iOS 14.4.2) -älypuhelimia. Puhelimiin ladattiin Expo-sovellus, jonka kautta testaus suoritettiin.

2.6 React Native -kehityksen kaksi tapaa

React Native -kehitys voidaan aloittaa kahdella eri tavalla. Ensimmäinen, helpompi, tapa on käyttää avoimen lähdekoodin Expo-sovelluskehystä, ja toinen tapa on käyttää React Nativen alustariippuvaisia komponentteja.

Expo

Expo on alusta, jonka avulla kehitetään, rakennetaan ja julkaistaan universaaleja React-sovelluksia. Expo on rakennettu React Nativen ympärille, ja se tarjoaa helpommat metodit natiiveihin toiminnallisuuksiin, esim. laitteen kameran käyttöön sovelluksessa.

Expon käytön aloittamiseksi kehittäjän tulee asentaa Expo CLI -komentorivityökalu, minkä jälkeen projekti alustetaan komentorivikomennolla ”expo init *projektin nimi*”. Komento luo projektin tiedostorakenteelle selkärangan, jonka päälle on helppo kehittää omaa sovellustaan. Sovellusta voi testata avaamalla Expo CLI:n mukana tuleva Metro Bundler komennolla ”npm start”. Tämä käynnistää http-palvelimen, joka tarjoaa sovellusta Expo-mobiilisovellukseen. Sovelluksen saa auki älypuhelimella asentamalla Expo-sovelluksen puhelimeen ja lukemalla sillä ”npm start”-komennon jälkeen komentoriville ilmestyvän QR-koodin (kuva 1). Tämä edellyttää molempien laitteiden olevan samassa verkossa tai Expon tunneliominaisuuden hyödyntämistä. QR-koodin lukemisen jälkeen Expo kääntää ja rakentaa koodin, minkä jälkeen sovellus on näkyvissä ruudulla. Sovellus päivittää automaattisesti muutokset koodista, jolloin niiden tarkastelu kehitystyössä on nopeaa. (Walkthrough 2021.)

Kuva 1. Esimerkki Expon luomasta QR-koodista.

Expon avulla tehtyjen sovellusten testaus on mahdollista Expo-sovelluksen avulla. Loppukäyttäjällä ei kuitenkaan yleensä ole laitteessa tätä sovellusta asennettuna, joten Expo tarjoaa mahdollisuuden luoda ”standalone APK”. APK on tiedosto, joka pitää sisällään kaiken sovelluksen tarvitseman tiedon. Tämä APK voidaan julkaista Google Play Storessa ja Apple Storessa, jolloin loppukäyttäjä voi käyttää sitä ilman Expo-sovellusta. (Walkthrough 2021.) Expon ”expo publish” -komennon avulla voi päivittää sovelluksen osia, ilman että se pitäisi poistaa ja lisätä uudestaan sovelluskauppoihin.

Expoa käyttäessä kehittäjä on rajoitettu käyttämään vain Expo CLI:n ja React Nativen sisältämiä komponentteja, koska Expo ei kokoa natiivikoodia projektia luodessa (Limitations 2021). Yleisimpiin sovelluskehityksen kohteisiin nämä riittävät mainiosti. Jos kuitenkin projektin kulun aikana todetaan, etteivät Expon komponentit riitä, voidaan Exposta luopua, mikä tarkoittaa, että projekti muutetaan Expo-sovelluksesta tavalliseen React Native -sovellukseen, jolloin Expon rajoittavuudet mutta myös helpottavat osat poistuvat. Tämä prosessi on helppo, ja sen voi myös peruuttaa.

React Native CLI

Sovelluksia voi myös kehittää Android Studiolla, jos halutaan Android-sovellus, tai Xcodella, jos halutaan iOS-sovellus. Android-sovelluksia voidaan kehittää käyttöjärjestelmäriippumattomasti, mutta iOS-sovelluksien kehitys Xcoden avulla vaatii macOS-tietokoneen.

Kehitettäessä mobiilisovelluksia Android Studiolla tulee kehittäjän asentaa koneelleen Node.js:n lisäksi Python, JDK (Java Development Kit), React Native CLI ja tietenkin itse Android Studio. Android Studion mukana asennetaan Android SDK ja joko virtualisoitu Android-käyttöjärjestelmä tai fyysinen Android-laite.

Kehitettäessä mobiilisovelluksia tulee kehittäjän asentaa Xcode ja Xcode Command Line Tools Mac App Storesta. Jos sovellusta halutaan testata iOS-simulaattorissa, tulee myös se asentaa Xcoden preferences-valikosta. Vaihtoehtoisesti voidaan käyttää fyysistä iOS-laitetta. (Setting up the development environment 2021.)

3 Mobiilisovelluksen prototyyppi

3.1 Backendin toteutus

Sovelluksen backendin toteutus alkoi teknologioiden valinnasta, jossa päädyttiin käyttämään SQLServer-kantaa ja ASP.Net-backendiä. Tämä koko paketti siirrettiin Microsoftin Azure-pilveen.

Sain työn tilaajalta kannan rungon ja tilaajan demoprojektista myös sisällön kantaan. Nämä oli helppo siirtää insinöörityöprojektin kantaan käyttämällä migraatioskriptejä, jotka hoitivat työn lähes itsestään. Kun kanta oli saatu kokoon, oli aika alkaa toteuttaa itse backendiä. Backendin palvelin saatiin pystyyn käyttämällä ASP.Net API -vakiotekstiä pohjana, jonka päälle suunniteltiin ja kirjoitettiin itse päätepisteet, sopimuksien mallit ja muu logiikka. Kieli oli minulle entuudestaan tuntematon, mutta parin päivän opiskelun ja dokumentaation selailun myötä se alkoi avautua ja sain kiinni siitä, mitä olin tekemässä.

Päätepisteiden tehtävänä on vastaanottaa pyyntö sovellukselta, toteuttaa pyynnön tarvitsemien tietojen haku kannasta ja palauttaa tämä tieto oikeassa muodossa sovellukselle. Tämän projektin toiminta vaati kaksi päätepistettä. Toinen hakee kannasta tietyn yrityksen omistamat sopimukset, ja toinen hallinnoi näiden sopimusten tilan arvoa. Päätepisteiden toimintaa testattiin tässä vaiheessa Postman-ohjelman avulla. Postman mahdollistaa http-pyyntöjen lähettämisen backendiin samaan tyyliin, kuin itse sovellus tulee tekemään. Pyyntöön liitetään halutun yrityksen tunnus, jonka avulla backend hakee kannasta tarvittavat tiedot ja palauttaa ne JSON-muodossa. Kuvassa 2 on esimerkki backendin päätepisteestä.

```
[Microsoft.AspNetCore.Mvc.HttpGet("{link}")]
public List<Agreement> Get(String link)
{
    using (var context = new AgreementRenewalDBContext())
        try
        {
            return context.Agreements.Where(ag => (ag.Link == link && ag.Status == 1)).ToList();

        } catch (Exception e)
        {
            Console.WriteLine("Error: " + e.Message);
            return null;
        }
}

/*
```

Kuva 2. Esimerkki päätepisteestä.

Pyynnöt oli tässä vaiheessa lähetettävä Ngrokin palvelimien kautta, jotta saatiin simuloitua pyynnön tuloa eri verkosta, kuin missä backend oli käynnissä. Pyynnöt siis lähetettiin Postmanistä Ngrokin palvelimelle, joka ohjasi liikenteen takaisin kehityskoneessa olevaan oikeaan osoitteeseen ja porttiin, jonka kautta sai yhteyden backendiin.

Kun backend oli saatu toimimaan halutulla tavalla, oli aika siirtyä kehittämään itse sovellusta, mutta tämä ei tarkoittanut, että työ backendin parissa olisi ollut valmista. Kun itse sovellusta oli saatu kehitettyä siihen asti, että sen avulla pystyi testaamaan päätepisteiden toiminnallisuutta, alettiin suunnitella backendin siirtoa Microsoftin Azure-pilvipalveluun. Azure-pilvipalvelu mahdollistaa tietokannan ja backendin ajon Microsoftin palvelimilla. Backendin siirto Azureen oli minulle aivan tuntematon konsepti, sillä en ollut tätä aikaisemmin työskennellyt pilvipalveluiden kanssa kehityspuolella. Sain tähän apua työn tilaajalta ja saimme backendin Azureen työpäivän sisällä. Backendin siirto Azureen oli siis melko vaivatonta, ja suuri osa siitä on automatisoitua. Siirron jälkeen sovelluksen http-pyynnöt ohjataan uuteen Azuren osoitteeseen. Tämä tarkoittaa, että Ngrokia ei enää tarvita projektissa.

3.2 Frontendin toteutus

Frontendin suunnittelu lähti käyntiin opiskelemalla, miten React Native eroaa tavallisesta Reactista, joka oli jo entuudestaan tuttu. Pääkohdat ovat samat, molemmat ovat komponenttipohjaisia kirjastoja, toinen on vain suunnattu mobiilisovelluksien kehittämiseen, kun toinen taas web-sovellusten kehittämiseen.

React Nativessa on mobiiliympäristöön ominaisia komponentteja, jotka kääntyvät iOS- ja Android-ympäristöissä kyseisen ympäristön natiiveiksi komponenteiksi. Esim. Button-komponentilla on erilaiset ominaisuudet iOS- ja Android-ympäristöissä. iOS Button vaatii lisäparametrejä, jos haluaa nappiinsa esim. taustan. Androidissa taas tausta on määriteltynä Button komponentin parametreissä suoraan.

Frontendin ulkoasusta luotiin karkea luonnos, jonka tarkoituksena oli olla apuna komponenttien vaatiman logiikan suunnittelussa ja komponenttien asemoinnissa. Sovelluksen tarkoituksena on tarjota kirjautumissivu, johon syötetään halutun yrityksen tunnus. Tämän jälkeen sovelluksen tulee listata kyseiselle yritykselle kuuluvat sopimukset helposti luettavassa muodossa. Kirjautumissivun ja -lomakkeen luonnin jälkeen suunnittelin sovelluksen tilan hallintalogiikkaa. Mietin, miten saan eri sivuilta tehdyt valinnat uusittavien sopimusten osalta pysymään muistissa, vaikka käyttäjä liikkuisikin eri sivujen välillä useasti ennen lopullista päätöstä vahvistaa valittujen sopimusten uusinta. Päädyin kirjoittamaan etusivulle React staten, joka piti sisällään listaa uudistukseen halutuista sopimuksista. Tämän tilan hallintafunktio ”porattiin” jokaiseen syvemmällä olevaan komponenttiin, mikä mahdollisti listan muokkauksen kaikilta sitä vaativilta sivuilta sovelluksen sisällä. ”Poraus” oli mahdollista, sillä React Native antaa mahdollisuuden syöttää ”child”-komponentille propseja, jotka voivat olla funktioita tai arvoja esim. listamuodossa. ”Child”-komponentti osaa täten hyödyntää sille annettuja propertyjä, ja niiden avulla muokata React statea, joka on ”parent”-komponentin sisällä.

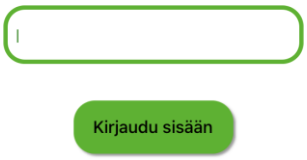
3.3 Näkymät

React Nativen komponentit pohjautuvat kaikki View-komponenttiin, jonka sisältö piirretään laitteen ruutuun. Toisin sanoen jokainen sovelluksen näkymä on oma View-komponenttinsa. Tässä insinöörityössä tehdyssä sovelluksessa on viisi näkymää: Kirjautuminen, Koti, Lisätiedot, Luokkatiedot ja Sopimuksen uusimisen vahvistus.

Kirjautuminen

Sovelluksen ensimmäinen näkymä on Kirjautuminen-näkymä (kuva 3), joka tulee vastaan heti, kun sovellus käynnistetään. Näkymän tarkoituksena on tarjota käyttäjälle lomake, jonka avulla sovellukseen syötetään kirjautumistunnus. Lomaketta klikatessa

avautuu laitteen oma näppäimistö, jolla kirjautumistunnus kirjoitetaan. Tunnuksen syötön jälkeen käyttäjän tulee painaa Kirjaudu-nappia, joka ohjaa käyttäjän sovelluksen Koti-näkymään.



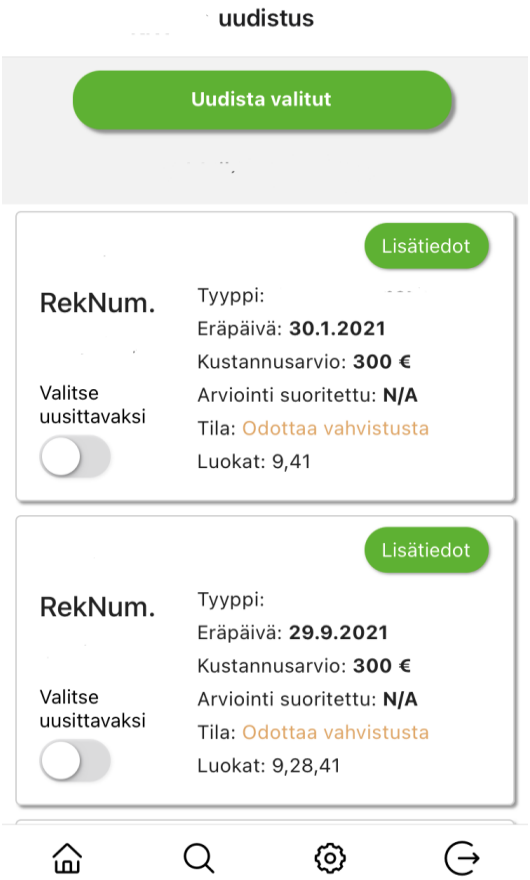
Kuva 3. Kirjaudu-näkymä.

Koti

Koti-näkymän (kuva 4) tarkoituksena on listata kirjautumistunnuksen omistuksessa olevat sopimukset käyttäjälle helposti luettavassa muodossa. Näkymää ladattaessa käyttäjä näkee pyörivän latausindikaattorin. Latauksen aikana sovellus lähettää http-pyynnön backendin päätepisteeseen kirjautumistunnuksen kanssa. Backend hakee tietokannasta sopimukset, jotka vastaavat kirjautumistunnusta, ja palauttaa ne listattuna JSON-muodossa.

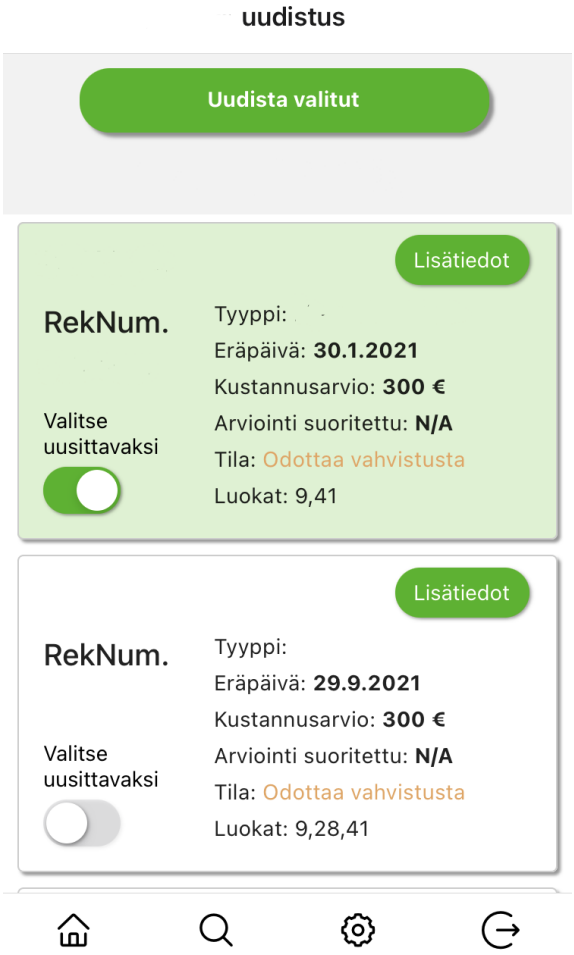
Näkymän koodissa on React Hook, joka ajetaan, kun backend on palauttanut sopimusten listan. Tämä Hook pitää sisällään silmukan, joka palauttaa jokaista sopimusta kohden Sopimus-komponentin. Nämä komponentit sijoitetaan Koti-näkymässä olevaan Scroll View -komponenttiin. Scroll View on muuten sama kuin View, mutta sitä voi vierittää vertikaalisesti. Sopimus-komponentti asemoi tietokannasta tulleen tiedon käyttäjälle

helposti luettavaksi laatikoksi, mahdollistaa kyseisen sopimuksen valinnan uudistamista varten, pitää sisällään tilamuuttujan valinnasta ja tarjoaa napin sopimusten lisätietojen avausta varten.



Kuva 4. Koti-näkymä.

Kuvasta 4 näkee Koti-näkymän ja sen sisällä olevat Sopimus-komponentit. Sopimuksen valinta uusimiseen on toteutettu React Nativen Switch-komponentilla, joka toimii tavallisen napin tavoin, mutta vaihtaa väriään, kun se on valittu (kuva 5).



Kuva 5. Valintaesimerkki.

Kuvasta 5 näkee, kuinka Switch-komponentti ja valitun Sopimus-komponentin värit vaihtuvat, kun se on valittuna uusimista varten.

Kuva 6 pitää sisällään useEffect Hook, joka ajetaan, kun Koti-näkymän lataus on valmis tai jos Sopimus-komponentin isSelected-tilamuuttujan arvo vaihtuu. Hookin sisältö vaihtaa kyseisen Sopimus-komponentin taustan väriä riippuen siitä, onko se valittuna uusimiseen vai ei. Kun Sopimus-komponentti on valittuna, muuttuu tausta valkoisesta vihreäksi.

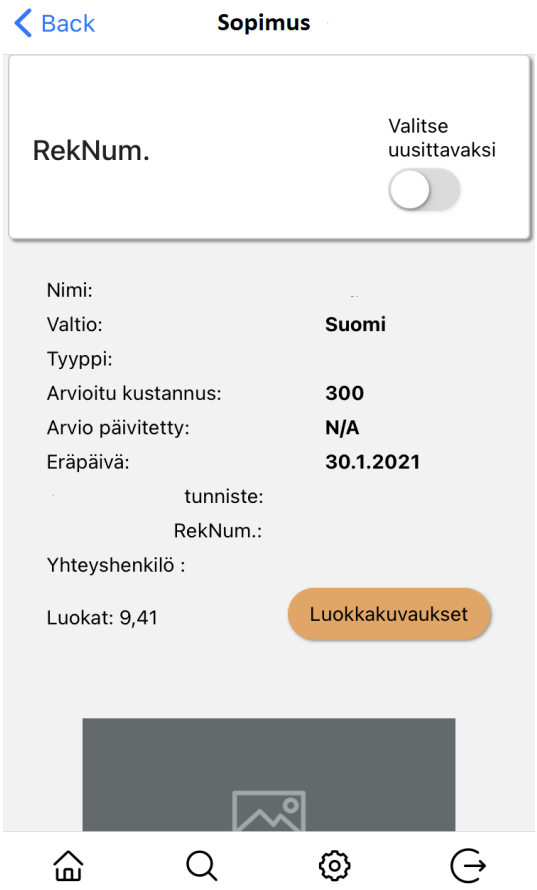
```
useEffect(() => {
    if (isSelected === true) {
        setSelectionBackgroundColor("#dbf2d0");
    } else setSelectionBackgroundColor("#fff");
}, [initDone, isSelected])
```

Kuva 6. useEffect-esimerkki.

Lisätiedot

Kun käyttäjä haluaa nähdä lisää tietoja sopimuksesta, tulee painaa ”Lisätiedot”-nappia. Tämä avaa Lisätiedot-näkymän (kuva 7). Lisätiedot-näkymän on tarkoituksena nimensä mukaan näyttää käyttäjälle enemmän tietoa, kuin Koti-näkymän Sopimus-komponenttiin mahtuu. Näkymän yläosassa on karsittu versio Sopimus-komponentista, jotta myös Lisätiedot-näkymästä pystyy valitsemaan sopimuksen uudistukseen. Tämän alla on listattuna useita tietoja sopimusta koskien. Myös sopimuksille mahdollisesti kuuluva kuva näkyy Lisätiedot-näkymässä. Jokaisella sopimuksella on omat sopimusluokkansa, joiden selitystä varten on toteutettu Luokkakuvaukset-näkymä.

Lisätiedot-näkymä saa Koti-näkymästä sopimuksen tiedot propseina. Tämä mahdollistaa jokaisen uniikin sopimuksen tietojen listauksen ilman uutta http-pyyntöä backendiin.



Kuva 7. Lisätiedot

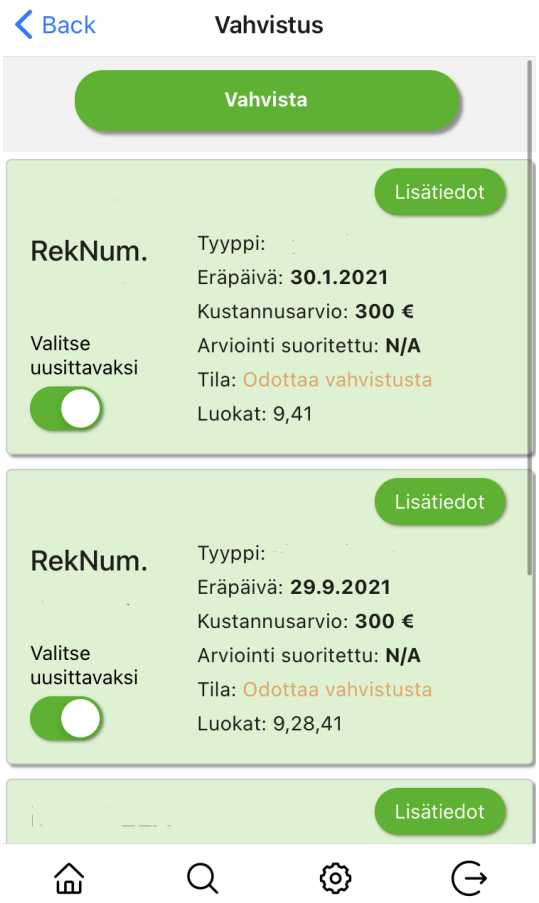
Luokkatiedot

Luokkatiedot-näkymä on tässä projektissa vain ns. ”placeholder”-näkymä. Näkymä on toteutettu kirjoittamalla tekstiä View-komponentin sisään. Sen tarkoituksena on vain havainnollistaa, että luokkatiedot vaativat oman näkymän niiden selityksille. Näitä selityksiä ei ole toteutettu, joten näkymä on kuvan 8 mukainen.

Kuva 8. Luokkatiedot.

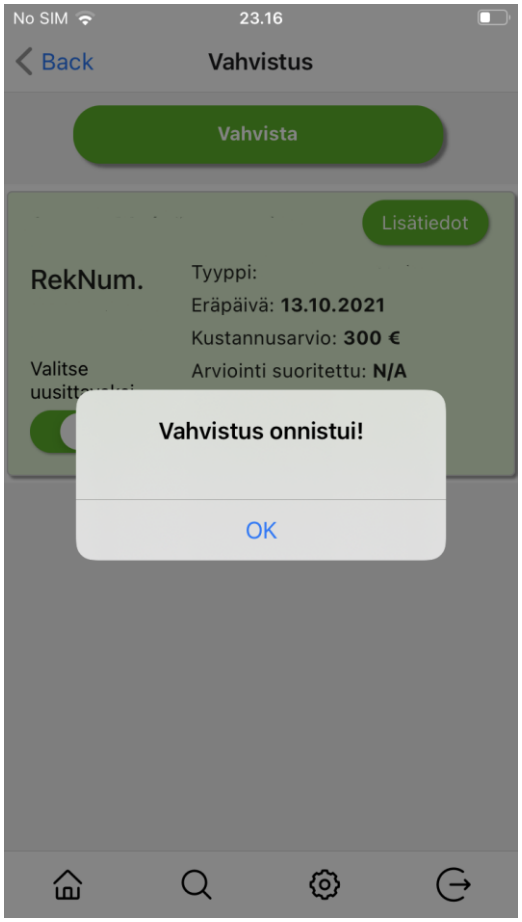
Sopimuksen uusimisen vahvistus

Viimeinen näkymä on Sopimuksen uusimisen vahvistus -näkymä (kuva 9). Sen tarkoituksena on listata käyttäjän uusimiseen valitut sopimukset samaan tyyliin kuin Koti-näkymässä. Sivun yläreunassa on ”Vahvista”-nappi, jonka painaminen lähettää backendiin http-pyynnön, jonka mukana lähtee valittujen sopimusten uniikit id:t (kuva 11). Backend muuttaa näiden sopimusten Status-muuttujan arvoa ykkösestä kakkoseksi, jolloin niitä ei enää näy Koti-näkymässä. Tämän muutoksen olisi tarkoitus jatkossa laukaista jonkinlainen prosessi sopimuksen uusimista varten, mutta se on jätetty tämän työn ulkopuolelle.



Kuva 9. Uusimisen vahvistus.

Ennen Vahvista-napin painallusta käyttäjä pystyy vielä poistamaan sopimuksia uudistuksesta, jos näin haluaa. Kun Vahvista-nappia painaa, ja jos Status-muuttujan arvon vaihto onnistuu backendin puolella, antaa näkymä tästä vahvistuksen käyttäjälle alertin muodossa (kuva 10).



Kuva 10. Uusimisen vahvistus-alert.

```
const updateStatus = async (props) => {
    try {
        const id = props;
        const resp = await axios.put(APIURL + "/api/agreement/updatestatus/" + id);
        if (resp = 203) {
            return 203;
        } else {
            console.log("Error updating status:")
            return null;
        }
    } catch (e) {
        console.log("Error: " + e.message)
    }
}
```

Kuva 11. Funktio, joka lähettää http-pyynnön sopimuksien statuksen muutosta varten.

3.4 React Navigation -kirjasto

Jotta käyttäjä pystyy kulkemaan eri näkymien välillä, tuli sovellukseen asentaa React Navigation. Tämän paketin avulla määritellään sovellukseen Stack- ja BottomTab-navigaatiot.

Stack

Sovelluksen juuressa (app.js tiedostossa) määritellään ns. Stack, johon on listattu jokainen näkymä ja sen sisällään pitämä komponentti (kuva 12). React Navigation pitää sisällään React Hookin, jota kutsutaan useNavigation()-funktiokutsulla. Tämä mahdollistaa navigate-metodin kutsun, joka mahdollistaa sovelluksen näkymien välillä siirtymisen. Esim. kuvissa vasemmassa yläreunassa näkyvä ”Back”-nappi on Stack Navigationin luoma (kuvat 9 ja 10). Toisin sanoen Stack-navigaatio on pino näkymiä, joiden välillä sovellus pystyy liikkumaan.

```
const Stack = createStackNavigator();
<Stack.Navigator initialRouteName="Kirjaudu">
    <Stack.Screen name="Kirjaudu" component={Login} options={{ headerShown: false }} />
    <Stack.Screen name="Sopimuksien uudistus" component={Home} options={{ headerLeft: null }} />
    <Stack.Screen name="Sopimus" component={AgreementInfo} />
    <Stack.Screen name="Vahvistus" component={Confirmation} />
    <Stack.Screen name="Luokkatiedot" component={ClassDescriptions} />
</Stack.Navigator>
```

Kuva 12. Esimerkki Stack-navigaatiokomponentista.

BottomTab

BottomTab Navigation (kuva 13) on toiminnaltaan hieman erilainen. Se määritellään jokaiseen näkymään erikseen ja näkyy näissä näkymissä sovelluksen alareunassa olevana palkkina (kuvat 7, 8, 9, 10). Tähän palkkiin voidaan määritellä erinäköisiä toiminnallisuuksia, mutta tässä sovelluksessa toteutettiin Kirjaudu ulos- ja Koti-toiminnot. Koti-toiminto vie Koti-näkymään, ja Kirjaudu ulos kirjaa käyttäjän ulos ja vie takaisin kirjautumissivulle.

```
const Tab = createBottomTabNavigator();
<Tab.Navigator>
    <Tab.Screen name="Kirjaudu" component={Login} />
    <Tab.Screen name="Sopimus" component={AgreementInfo} />
    <Tab.Screen name="Vahvistus" component={Confirmation} />
    <Tab.Screen name="Luokkatiedot" component={ClassDescriptions} />
</Tab.Navigator>
```

Kuva 13. Esimerkki yksinkertaisesta BottomTab-navigaatiokomponentista.

4 Yhteenveto

Insinöörityön tarkoituksena oli kehittää työn tilaajalle sopimuksienhallintatyökalun mobiiliversion prototyyppi käyttämällä React Native -teknologiaa, ja samalla perehdyttiin hybridimobiilikehityksen perusteisiin. Insinöörityön tavoitteisiin päästiin. Hybridimobiilikehitykseen perehdyttiin, jotta saatiin kokonaisvaltainen kuva prototyypin kehittämisen vaateista, ja siihen liittyvien teknologioiden perusteet tuli opeteltua uutena taitona. React Native -teknologiaa hyödyntäen saatiin aikaan mobiilisovelluksen prototyyppi, joka voidaan asentaa niin iOS- kuin Android-käyttöjärjestelmille. Prototyypin halutut toiminnallisuudet saatiin kaikki toteutettua, mutta jatkokehittämisvaraa on, jos sovelluksesta halutaan oikea tuote.

Mobiilisovellusmarkkinat ovat suuressa kasvussa, joten mobiilikehittämisen tulevaisuuden näkymät ovat työllistymisen kannalta erittäin hyvät. Suurin osa markkinoista on Android-käyttöjärjestelmän omaavia laitteita ja toiseksi suurin osa on iOS-laitteita. Näiden käyttöjärjestelmien sovelluskehitys eroaa toisistaan niin ohjelmointikielessä kuin kehitysympäristössä. Tätä helpottamaan Facebook on luonut React Native -kirjaston, jonka avulla voidaan kehittää sovelluksia molemmille käyttöjärjestelmille. React Native -kirjasto pohjautuu toiminnaltaan Facebookin kehittämään React-kirjastoon. Tämä tarkoittaa, että React-ympäristöstä tuttu komponenttipohjainen kehitys onnistuu myös React Nativessa. Mobiilisovelluksen runko on hyvin samankaltainen kuin tavallisen web-sovelluksen. Se koostuu frontendistä ja backendistä. Frontend kuvaa kaikkea, mitä käyttäjä näkee, kun taas backend kaikkea taustalla tapahtuvaa koodin suoritusta.

Kehitysprosessi sujui kohtalaisen vaivattomasti, koska aluksi opiskeltiin vaadittavat teknologiat perusteellisesti ja entuudestaan tuttu React-teknologia kääntyi React Native -kehittämiseen helposti. Backendissä käytössä olevat ohjelmointikielet olivat uusia, mutta niiden toiminnan perusteet olivat entuudestaan tuttuja. Backendin toteutus ja pilveen siirto sujuivat molemmat vaivattomasti. Pieniä haasteita aiheutui koronatilanteen takia, sillä vaikka kommunikaatio työn tilaajan kanssa toimi, tuo etätyöskentely aina omat haasteensa.

Lähteet

Aston, Ben. 2015. A Brief history of JavaScript. Verkkoaineisto. <https://medium.com/@_benaston/lesson-1a-the-history-of-javascript-8c1ce3bffb17>. Luettu 15.02.2021

Banks, Alex & Porcello, Eve. 2020. Learning React. 2nd Edition. O'Reilly Media.

Horstmann, Cay S. 2020. Modern JavaScript for the Impatient. Addison-Wesley Professional

Limitations. 2021. Verkkoaineisto. Expo. <https://docs.expo.io/introduction/why-not-expo/>. Luettu 15.02.2021.

Mobile Application Market Size Expected to Reach USD 407.31 Billion by 2026 – Valuates Reports. 2020. Verkkoaineisto. Valuates Reports. <https://www.prnewswire.com/news-releases/mobile-application-market-size-expected-to-reach-usd-407-31-billion-by-2026---valuates-reports-301053918.html>. Luettu 18.3.2021.

Mobile Marketing Market. 2019. Verkkoaineisto. Marketsandmarkets. <https://www.marketsandmarkets.com/Market-Reports/mobile-marketing-market-246836146.html>. Luettu 18.03.2021.

Mobile Operating System Market Share Worldwide. 2021. Verkkoaineisto. Statcounter. <https://gs.statcounter.com/os-market-share/mobile/worldwide>. Luettu 18.3.2021

Nalwaya, Abhishek & Paul, Akshat. 2019. React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications. O'Reilly Media

Setting up the development environment. 2021. Verkkoaineisto. React Native. <https://reactnative.dev/docs/environment-setup>. Luettu 15.2.2021.

The mobile app market analysis: current and future trends. 2017. Verkkoaineisto. Railwaymen. <https://medium.com/@Railwaymen_org/the-mobile-app-market-analysis-current-and-future-trends-6e119a58e306>. Luettu 18.3.2021.

Walkthrough. 2021. Verkkoaineisto. Expo. <https://docs.expo.io/introduction/walkthrough/>. Luettu 15.2.2021.

Welcome to the Visual Studio IDE. 2019. Verkkoaineisto. Microsoft. <https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide?view=vs-2019>. Luettu 6.4.2021.