TAMPEREEN
AMMATTIKORKEAKOULU

MOBIILISOVELLUKSEN TOTEUTTAMINEN REACT NATIVE -OHJELMISTOKEHYKSELLÄ

Konsta Salminen

Opinnäytetyö
Toukokuu 2018
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tieto- ja viestintätekniikka
Ohjelmistotekniikka

SALMINEN KONSTA
Mobiilisovelluksen toteuttaminen

Opinnäytetyö 22 sivua, joista liitteitä 0 sivua
Toukokuu 2018

Tässä opinnäytetyössä käydään läpi mobiilisovelluksen toteutusta Android- ja iOS-laitteille React Native -ohjelmistokehystä käyttäen sekä REST-ohjelmointirajapinnan toteutusta Drupal-ohjelmistokehyksellä. Mobiilisovelluksen tarkoitus on toimia karsittuna versiona asiakasyrityksen verkkosivustosta.

Työn tuloksena oli testiympäristössä toimiva Android-mobiilisovellus ja sitä tukeva Drupalilla toteutettu REST-ohjelmointirajapinta.

Asiasanat: mobiili, android, ios, drupal, rest

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Information and Communication Technology
Software technology

SALMINEN KONSTA:
Implementation of a Mobile Application

Bachelor's thesis 22 pages, appendices 0 pages
May 2018

This thesis goes through the implementation of a functioning mobile application for Android and iOS devices by using React Native framework and Drupal framework for a REST application programming interface (API). The function of the mobile application is to be a stripped-down version of a customer's website.

The result of this project was an Android application and a Drupal REST API that functions in a test environment.

Key words: mobile, android, ios, drupal, rest

SISÄLLYS

1 JOHDANTO

Projektin toimeksiantona oli luoda mobiilisovellusversio toimeksiantajan asiakasportaali verkkosivustosta, jossa yrityksen asiakkaat voivat muun muassa seurata yritystensä erinäisiä kirjanpitoon liittyviä lukuja ja laskutusta. Toimeksiantaja oli Gylling Accounting Oy, joka on Tampereella, Valkeakoskella, Hämeenlinnassa ja Parkanossa toimiva talous- ja henkilöstöhallinnon kehittäjä yritys.

Tässä raportissa kerrotaan mobiilisovellusprojektissa käytetyistä teknologioista ja miten sovellus on toteutettu. Projektin tavoitteena oli luoda kevennetty versio asiakkaan verkkosivustosta mobiilisovelluksena. Projektiin kuului itse mobiilisovelluksen ja tarvittavien palvelinrajapintojen toteuttaminen. Mobiilisovellukseen sisällytettiin vain muutamia tärkeimpiä verkkosivuston ominaisuuksia, kuten pankkitilien saldot, kassaennuste ja ostolaskujen hyväksyntä.

Projektissa käytetyt teknologiat olivat React Native mobiilisovelluksessa ja Drupal palvelinohjelmistokehyksenä. Projektin lähtökohtina olivat mobiilisovellus runko, johon oli valmiiksi toteutettu sovelluksen navigointijärjestelmä sekä rajapinta erilaisille verkkokutsuille, ja Drupalissa toimiva verkkosivusto, jonka pohjalta mobiilisovellus oli tarkoitus tehdä.

Mobiilisovelluksen toteutuksessa käytettiin mahdollisimman paljon React Nativen omia komponentteja ja kolmannen osapuolen komponentteja, jotka oli tehty toimimaan sekä Android- että iOS-alustoilla. Verkkorajapintojen toteutukseen käytettiin Drupalin tarjoamaa REST-moduulia ja sitä varten kehitettyä graafista käyttöliittymää sekä kolmannen osapuolen JWT-moduulia käyttäjän autentikointiin.

2 TEKNOLOGIAT

2.1 Mobiiliteknologia vaihtoehdot

Mobiilisovellusten kehitykseen löytyy useita eri teknologioita, jotka jakautuvat kolmeen eri luokkaan; natiivi, web ja hybridi. Natiiviteknologioita käytettäessä sovelluskehitys tapahtuu käyttöjärjestelmän valmistajan tekemillä työkaluilla, joita ovat esimerkiksi Android Studio Androidille ja Xcode Applen tuotteille. Käytetyt ohjelmointikielet myös vaihtelevat alustojen välillä, Javaa Androidille ja Objective-C:tä tai Swiftiä iOS:lle. Web-sovellukset ovat verkkosivuja, joita käytetään mobiililaitteiden selaimissa ja ne ovat usein kirjoitettu käyttäen HTML5:ttä. Hybriditeknologiat käyttävät nimensä mukaisesti osia sekä natiivi- että web-teknologiosta. Usein hybridisovellukset ovat verkkosivuja, jotka on käärittyjohonkin sovelluskehykseen, joka yhdistää verkkosivun laitteistorajapintoihin. Natiivi- ja hybridisovellukset ovat laitteeseen asennettavia sovelluksia. Eri teknologiatyyppien eroihin kuuluvat suorituskyky, käyttökokemus ja kehitystyö. (Budiu Raluca 2013)

Natiiviteknologioilla saadaan paras suorituskyky, koska natiivi ympäristöön on sisäänrakennettu kaikki laitteistorajapinnat ja ohjelmointikielenä käytetään suoraan yhteensopivaa kieltä. Hybriditeknologioissa vastaavasti joudutaan usein tulkitsemaan tai kääntämään jotain muuta ohjelmointikieltä natiiviksi ja sovelluksen laitteistorajapintojen välissä saattaa olla ylimääräinen kerros. Suorituskyky ero ei kuitenkaan ole niin suuri, että se pakottaisi aina käyttämään natiivia.

Natiiviteknologioilla myös voidaan saada aikaiseksi paras käyttökokemus, kun käytössä on käyttöjärjestelmän valmistajan omat työkalut, joissa käyttöliittymäelementit ja -toiminnot seuraavat alustakohtaisia normeja. Hybriditeknologioilla voidaan usein saada aikaiseksi lähes natiivia vastaava käyttökokemus jatkuvasti kehittyvien teknologioiden ansiosta.

Jos ollaan kehittämässä mobiilisovelluksia sekä Android- että iOS-alustoille kehitystyön määrä on lähes kaksinkertainen natiiveja teknologioilla verrattuna useimpiin hybrideihin,

koska suuri osa hybriditeknologioiden koodeista toimii molemmilla alustoilla samalla tavalla, kun taas natiivi koodeja joutuu kirjoittamaan omansa kummallekin alustalle. (Klimenko Anna 2018)

Tunnetuimpia hybriditeknologioita ovat Xamarin, Ionic ja React Native. Xamarinilla ohjelmointi tehdään C#-ohjelmointikielellä ja keskimäärin 75% koodista on samaa eri alustoilla. Xamarin kääntää koodin iOS:lle suoraan ARM assembly koodiksi ja Android koodia ajetaan käyttämällä ajonaikaista kääntämistä (Xamarin 2018). Ionicilla sovellusten kehitys tapahtuu käyttämällä HTML-, CSS- ja JavaScript-koodia. Ionic-sovelluksia ajetaan web-näkymässä (Ionic 2018). React Native sovellukset tehdään pääosin JavaScript-koodilla, jota ajetaan JavaScriptCore-virtuaalikoneella (Facebook 2018).

2.2 Käytetyt teknologiat

2.2.1 React Native

React Native on Facebookin kehittämä avoimen lähdekoodin ohjelmistokehys mobiilisovelluksille. React Native julkistettiin vuonna 2015, eli se on vielä melko uusi ohjelmistokehys. React Native pohjautuu Facebookin kehittämään React JavaScript-kirjastoon, jonka ensimmäinen julkaisu tapahtui vuonna 2013.

React Nativella voidaan käyttää suurelta osin samaa koodia sekä Android- että iOS-alustoilla. Ohjelmointikielenä käytetään JavaScriptiä. React Nativeen voidaan myös lisätä alustakohtaista koodia, joka on Android-alustoilla Javaa ja iOS-alustoilla Objective-C:tä tai Swiftiä. React Nativella tehty sovellus ei ole web-sovellus, jota ajetaan web-näkymässä kuin se olisi verkkosivu kuten joissain samankaltaisissa teknologioissa tehdään. React Native JavaScript-koodia ajetaan JavaScriptCore-virtuaalikoneella. React Native myös hyödyntää myös Androidin ja iOS:n omia käyttöliittymäelementtejä. Ohjelmistojen kehitys Android-laitteille onnistuu Windows-, Linux- ja macOS-käyttöjärjestelmillä ja iOS-laitteille vaaditaan macOS-käyttöjärjestelmä.

React Nativessa ja Reactissa käyttöliittymän ohjelmoinnissa käytetään JSX-syntaksia, joka mahdollistaa XML:n käytön JavaScript-koodissa ja se näyttää HTML-koodin kaltaiselta, missä HTML-tagit on korvattu React- komponenteilla. Kuvassa 1 on esitetty Hello World -sovelluksen React Native-koodi.

```
import React, { Component } from 'react';
import { Text } from 'react-native';

export default class HelloWorldApp extends Component {
  render() {
    return (
      <Text>Hello world!</Text>
    );
  }
}
```

KUVA 1 Hello World -koodi (https://facebook.github.io/react-native/docs/tutorial.html)

React Native-pakettiin kuuluu Babel JavaScript-kääntäjä, joka mahdollistaa uusimpien JavaScript-standardien mukana tulevien ominaisuuksien käytön, ja Jest-testityökalu JavaScriptille, joka on myös Facebookin kehittämä. Vakiona React Nativen pakettien hallintaan käytetään Node Package Manageria eli NPM:ää, mutta muitakin vaihtoehtoja löytyy kuten Yarn.

React Nativella ohjelmointi perustuu komponenttien luontiin. Komponenteilla voi olla oma tila eli state, joka on vain kyseisen komponentin käytettävissä ja se sisältää nimettyjä muuttujia, joita voidaan muuttaa. Komponenteille voidaan välittää propseja (properties), jotka ovat funktioiden parametrien kaltaisia. Komponenttien käyttöliittymä määritellään komponenttien render-funktioissa (kuva 1). React komponenteillä on myös elämänkaarimetodeja (engl. lifecycle method), joita kutsutaan esimerkiksi, kun komponentti asennetaan ja poistetaan. Komponenttien ulkoasuun voidaan vaikuttaa style nimisellä propilla, joka vastaanottaa objektin, joka sisältää CSS-koodin kaltaisia arvoja.

React Native on käytössä esimerkiksi seuraavissa mobiilisovelluksissa: Facebook, Instagram, Skype ja Airbnb.
(Facebook 2018)

2.2.2 Drupal

Drupal on avoimen lähdekoodin sisällönhallintajärjestelmä (CMS eli Content Management System). Drupalin ensimmäinen versio julkaistiin vuonna 2000 ja raporttia kirjoitettaessa uusin versio on 8.5.1.

Drupal on kirjoitettu PHP-ohjelmointikielellä ja yksinkertaisimmillaan Drupalin käyttö ei vaadi ohjelmointitaitoja, sivustoja voi hallita graafisen käyttöliittymän avulla. Drupal käyttää asetusten ja muiden tietojen tallentamiseen SQL-tietokantaa. Drupaliin löytyy monia yhteisön kehittämiä moduuleja ja kuka tahansa voi luoda ja jakaa omia moduulejaan. Moduulien hallintaan suositellaan Composer-riippuvuuksienhallintatyökalua, ja Drupalissa on myös oma komentorivirajapinta nimeltä Drush, jolla voi esimerkiksi kommunikoida moduulien kanssa ja suorittaa SQL-kyselyitä.

Drupal sivustojen kehitys tapahtuu PHP:llä ja Drupal 8 versiossa siirryttiin olio pohjaiseen tyyliin ja alettiin hyödyntämään osia Symfony-ohjelmistokehyksestä. Drupal tukee myös JavaScriptin käyttämistä sivuilla ja Drupal Core sisältää muun muassa jQuery-kirjaston. Drupal 8 toteuttaa PSR-4 standardin mukaisen pakettipohjaisen nimiavaruus automaattilatauksen, joka vaatii moduuleilta tietyn kansiorakenteen. Kun kansio rakenne on toteutettu oikein nimiavaruudet ovat käytettävissä koko sovelluksen laajuisesti. (Drupal 2017)

Esimerkkejä Drupalia käyttävistä yrityksistä: Tesla, NBC, Pfizer ja Princess Cruises. (Drupal 2018b)

3 TOTEUTUS

Ohjelmistojen toteutuksessa käytettyjä työkaluja olivat Atom ja Visual Studio Code ohjelmointiin tarkoitetut tekstieditorit, Android Studio emulaattori mobiililaitteiden emulointiin, Google Chrome vianetsintä työkalu, Git-versionhallintatyökalu ja Postman API testaustyökalu. Kehitykseen käytettiin Linux-käyttöjärjestelmää.

Projektin seurantaan käytettiin Odoo:ta ja projekti jaettiin osiin ohjelmistoon haluttujen toimintojen perusteella. Osiin kuului vaadittavat mobiili- ja palvelinohjelmistojen toteutukset.

3.1 Mobiili

Mobiilisovelluksen toteutus alkoi Pepperoni-sovelluspohjan päältä, mikä on Futuricen kehittämä pohja React Native-sovelluksille. Pepperonissa oli valmiiksi toteutettu autentikaatiorajapinta, navigaatiojärjestelmä, johon kuului myös päävalikko, ja Redux-moduuli. Pepperonissa oli valmiiksi asennettuna myös muita paketteja. Toteutuksessa käytettiin mahdollisuuksien mukaan valmiita avoimenlähdekoodin paketteja ja yksi pakettien valinta kriteereitä oli toimivuus sekä Android- että iOS-alustoilla. Pakettien hallintaan käytettiin pääosin Yarnia.

Sovelluksen kehitystä helpotti huomattavasti React Nativen tarjoama mahdollisuus nähdä koodin muutoksien vaikutukset suoraan emulaattorissa tai testilaitteessa, mikä nopeutti etenkin käyttöliittymän rakentamista. Molempiin kehityksen aikana käytettyihin tekstieditoreihin löytyi moduuleja React Native kehitystä varten, nämä moduulit tarjosivat esimerkiksi syntaksin tarkistusta ja koodi ehdotuksia.

3.1.1 Redux

Redux on JavaScript-moduuli, joka mahdollistaa koko sovelluksen laajuisen tilan käytön. Tämä tila on tavallinen objekti, joka on osa Reduxin store-objektia (myöh. store). Tilaa ei voi muuttaa suoraan vaan se tehdään lähettämällä action-objekti (myöh. action), jonka

täytyy sisältää vähintään tyyppi, joka on merkkijono. Yleensä actionit myös sisältävät datan, joka kirjoitetaan tilaan reducer-funktiolla. Actioneja tehdään kutsumalla funktioita, joista käytetään nimeä action creator, nämä funktion voivat ottaa mitä tahansa dataa parametrinä ja ne palauttavat actionin. Reducer-funktiot vastaanottavat edellisen tilan ja actionin ja palauttavat seuraavan tilan. Reducerit eivät suoraan muuta vanhaa tilaa vaan kopioivat sen ja tekevät actionien osoittamat muutokset ja sitten palauttavat tämän uuden tilan. Reducerit ovat myös osa storea.

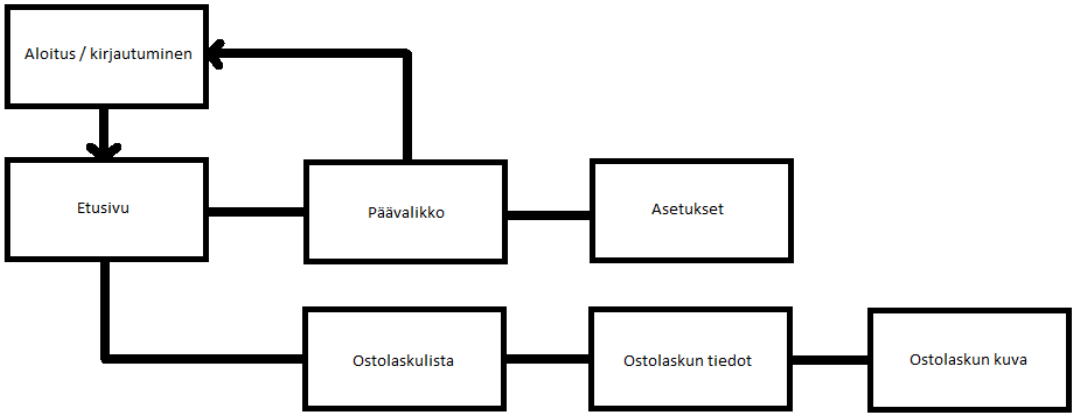
Storeen voi myös liittää väliohjelmistoja (engl. Middleware) storen luonnin yhteydessä, yksi tälläisistä väliohjelmistoista on esimerkiksi Redux Thunk, joka mahdollistaa asynkronisten action creatorien käytön. Väliohjelmistot vastaanottavat lähetetyt actionit ja palauttaa uudet actionit reducerille haluttujen toimintojen jälkeen. (Redux 2018)

3.1.2 Arkkitehtuuri

Toteutuksessa käytettiin Pepperonissa määriteltyä arkkitehtuuria, jossa komponentit jotka käsittelevät sovelluksen tilaa tehtiin tiedostorakenteessa erilleen uudelleenkäytettävistä komponenteista. Yksittäiset tilaa käsittelevät komponentit jaettiin kolmeen eri osaan, joista yhdessä tiedostossa oli itse komponentti, toisessa tiedostossa komponentti yhdistettiin Redux storeen ja kolmannessa tiedostossa oli komponentin actionit ja reducer-funktio. (https://github.com/futurice/pepperoni-app-kit/blob/master/docs/ARCHITECTURE.md)

Kuvassa 2 on esitetty sovelluksen navigaatiokartta. Ensimmäinen näkymä sisältää sisäänkirjautumisen. Onnistuneen sisäänkirjautumisen jälkeen sovellus siirtyy etusivulle. Päävalikko avautuu etusivun päälle ja se sisältää käyttäjän aktiivisen yrityksen valinnan, uloskirjautumis painikkeen ja asetuksiin siirtymispainikkeen. Etusivulla sijaitsevasta na-

pista voidaan siirtyä näkymään, jossa on lista avoimista ostolaskuista ja mahdollisuus hyväksyä laskuja tai siirtyä katsomaan yksittäisen laskun tarkempia tietoja. Yksittäisen laskun tiedoista voidaan avata pdf-kuva laskusta.



KUVA 2. Navigaatiokartta

3.1.3 Sovellus

Sovelluksen jokaisessa näkymässä käytettiin React Native Material UI -paketin työkalurivi-komponenttia. Komponenttiin oli mahdollista lisätä painikkeita ja otsikkotekstipainikkeille pystyi antamaan ikonin tai tekstin ja funktion, joka ajetaan painiketta painettaessa. Jokaiselle sivulle työkaluriville lisättiin otsikoksi näkymän nimi ja etusivulla (kuva 3) työkaluriville asetettiin kaksi painiketta, ensimmäinen avaamaan päävalikko ja toinen etusivulla näkyvien tietojen päivitykseen, muilla sivuilla työkaluriville asetettiin yksi painike näkymien sulkemiseen tai taaksepäin navigointiin.

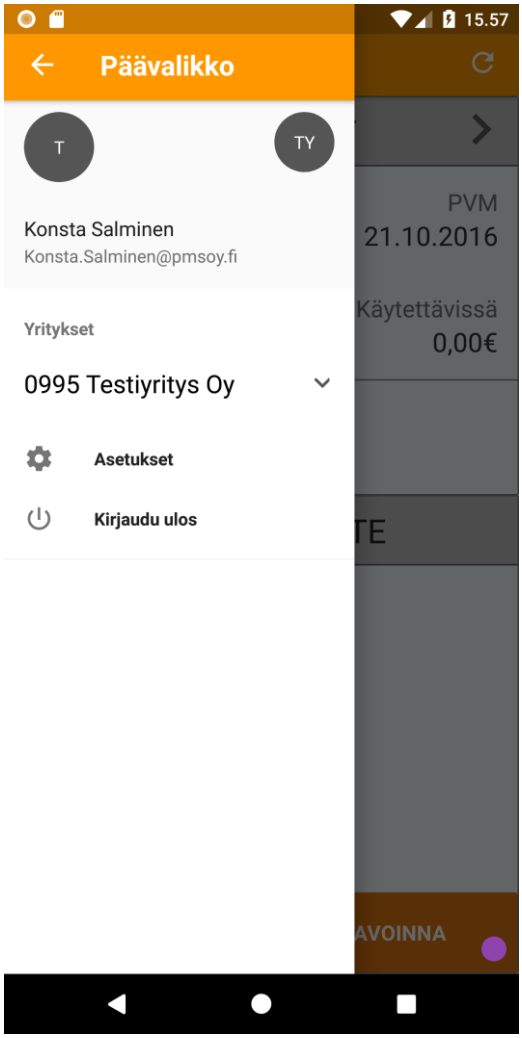
KUVA 3. Etusivu

Etusivun pankkitilit osio pystyttiin luomaan React Nativen omilla View-, Text- ja TouchableOpacity-komponenteilla sekä Icon- ja Button-komponenteilla React Native Material UI -paketista (jatkossa Materia UI). TouchableOpacity-komponenttien avulla voidaan asettaa muille komponenteille painallusfunktioita laittamalla haluttu komponentti TouchableOpacityn sisään kuvassa 4 esitetyn esimerkin tavalla. Pankkitilit otsikon sivuilla oleville nuoli ikoneille asetettiin painallusfunktiot, joilla voidaan selata läpi yrityksen pankkitilejä. Etusivun alaosaan toteutettiin kassaennuste-kuvaaja Victory Native -paketin avulla. Kuvaaja piirretään vain, jos sovellus on vastaanottanut kassaennustedataa.

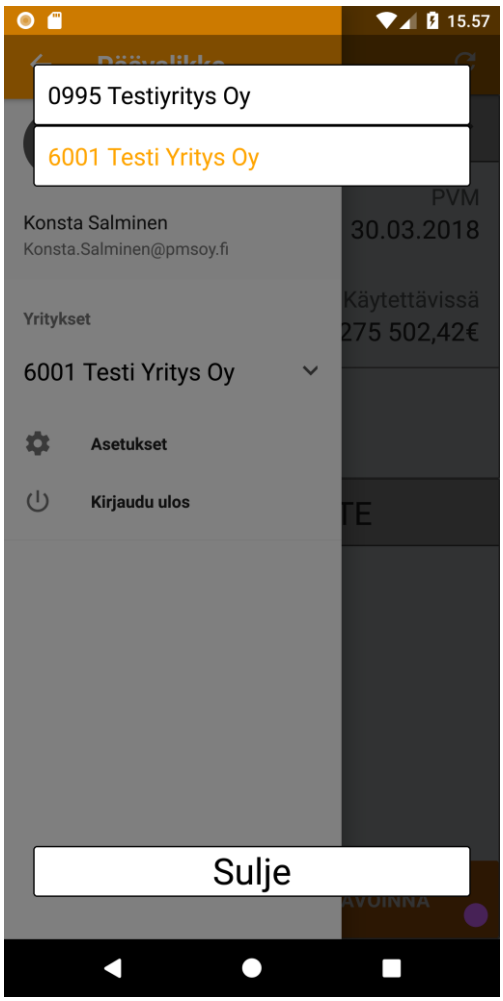
```
<TouchableOpacity onPress={this._onPressButton}>
  <Image
    style={styles.button}
    source={require('./myButton.png')}
  />
</TouchableOpacity>
```

KUVA 4. TouchableOpacity

Sovelluksen päävalikko (kuva 5) toteutettiin Matetial UI:n Drawer-komponentilla, joka käyttää muita saman paketin komponentteja ulkoasun määrittämiseen. Valikon yläosan palloilla käyttäjä voi vaihtaa aktiivista yritystä. Valikko pitää muistissa yhdestä kolmeen viimeksi käytettyä yritystä. Valikon keskiosassa sijaitsevaa yrityksen nimeä painamalla avautuu React Nativen Modal- ja FlatList-komponenteilla toteutettu yrityksen valinta ruutu (kuva 6). Yrityksen nimen alapuolella päävalikossa on painikkeet asetussivun avaamiseen ja uloskirjautumiseen.

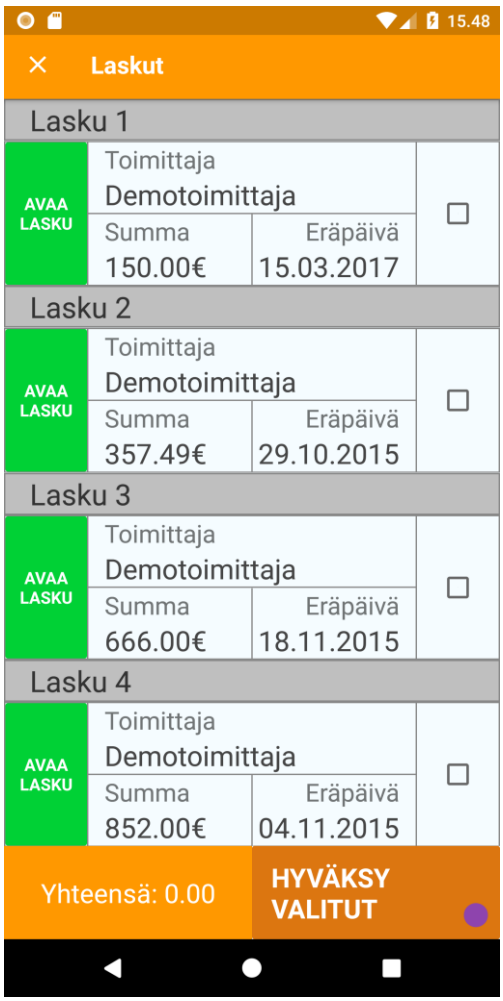


KUVA 5. Päävalikko



KUVA 6. Yritysvalinta

Etusivun alalaidassa sijaitsevasta näppäimestä avautuu näkymä (kuva 7), jossa on lista aktiivisen yrityksen ostolaskuista. Listan toteutukseen käytettiin FlatList-komponenttia. FlatList-komponentin kohteita varten täytyy sille antaa propsina funktio, joka palauttaa piirrettävän komponentin (kuva 8). Listan kohteissa käytettiin React Nativen View- ja Text-komponentteja sekä Material UI:n CheckBox- ja Button-komponentteja. Valittujen laskujen yhteissumma päivittyy näkymän vasempaan alareunaan. Oikean alareunan näppäintä painettaessa sovellus lähettää tarvittavat laskujen tiedot POST-pyynnössä palvelimelle, minkä jälkeen avoimet laskut haetaan uudelleen.

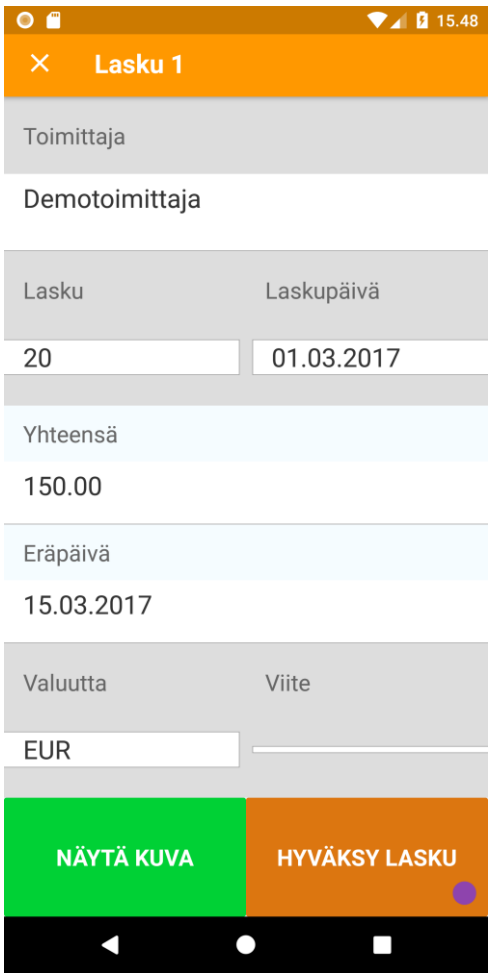


KUVA 7. Ostolaskulista

```
<FlatList
  data={[{key: 'a'}, {key: 'b'}]}
  renderItem={({item}) => <Text>{item.key}</Text>}
/>
```

KUVA 8. FlatList koodi

Ostolaskulistan kohteen Avaa lasku -näppäintä painettaessa sovellus siirtyy näkymään, jossa näytetään tarkemmin kyseisen laskun tietoja (kuva 9). Laskun tiedot näkymä toteutettiin React Nativen View- ja Text-komponenteilla sekä Material UI:n Button-komponenteilla. Oikeanpuoleista näppäintä painettaessa lasku hyväksytään ja sovellus palaa laskulista näkymään. Vasemmanpuoleinen näppäin avaa uuden näkymän, jossa näytetään laskun PDF-kuva. PDF-kuvan näyttämiseen käytettiin React Native PDF -paketin tarjoamaa komponenttia, jolle välitetään base64-enkoodattu merkkijono.



KUVA 9. Laskun tiedot

Sovelluksen kaikki verkkorajapintakutsut toteutettiin käyttäen Pepperonin valmiita funktioita. Kaikki rajapinnasta vastaanotetut tiedot tallennetaan Redux tilaan käyttäen asynkronisia action creatoreja, nämä action creatorit tekevät rajapintakutsun, joka palauttaa lupaus-objektin (engl. Promise), lupaus-objekteihin voidaan kiinnittää takaisinkutsufunktioita, joita kutsutaan, kun lupaus täytetään (resolve) tai hylätään (reject) (MDN web docs). Onnistuneen kutsun jälkeen asynkroniset action creatorit palauttavat tavallisen action creatorien tapaan actionin reducerille, jos kutsu epäonnistuu, virhe tulostetaan konsoliin.

Sovellukseen toteutettiin lokalisaatio hyödyntäen React Native I18n -pakettia. Lokalisaatiolle täytyi luoda kansio projektin juureen. Kansioon tehtiin yksinkertainen asetus kooditiedosto, jossa määriteltiin käytettävät kielet ja oletus kieli, ja erillinen kansio kielitiedostoille, joissa käännökset ovat JSON-objekteissa merkkijonoina. Kielen valintaa varten luotiin asetukset-näkymään (kuva 10) alasvetovalikko käyttäen React Nativen Picker-komponenttia. Kun sovellus käynnistetään ensimmäisen kerran lokalisaatiomoduuli tunnistaa laitteeseen asetetun kielen ja käyttää sitä.

KUVA 10. Asetukset

Sovelluksen itse tehdyille komponenteille luotiin yksikkötestejä käyttäen Jestiä. Testeissä hyödynnettiin Reactille tehtyä Enzyme-kirjastoa. Kuvassa 11 on esimerkki yhdestä thedystä snapshot testistä. Snapshot testissä komponentti luodaan ja sen tulostus tallennetaan JSON muodossa erilliseen tiedostoon. Testi perustuu siihen, että seuraavilla kerroilla kun testi ajetaan, tulostusta verrataan olemassa olevaan snapshottiin, jos eroja löytyy Jest ilmoittaa siitä ja näyttää muuttuneet osat joiden perusteella kehittäjä päättää onko muutokset haluttuja vai ei ja jos tulos on haluttun, snapshot voidaan luoda uudelleen ja tallentaa vanhan päälle.

```
describe("<InvoiceInfoView />", () => {
  it('should match snapshot', () => {
    const wrapper = shallow(
        <InvoiceInfoView
            navigation={{state: {params: {index: 0}}}}
            invoicesActions={InvoicesActions}
            openInvoices={invoices}
            activeCompanyId={'6001'}
        />,
        {
            context: {uiTheme}
        }
    );
    expect(wrapper).toMatchSnapshot();
  });
});
```

KUVA 11. Snapshot testi

3.2 Palvelin

Palvelin rajapinnat toteutettiin luomalla uusi moduuli, joka hyödyntää Drupal coren REST-moduulia. Yksittäiset rajapinnan osoitteet toteutettiin moduulin liitännäisinä (engl. Plugin). REST-rajapintaliitännäisten koodissa täytyi käyttää kuvan 12 mukaista huomautusta (engl. annotation) ennen luokan määritystä. Huomautuksen id määrittää resurssin koneluettavan nimen, label määrittää käyttöliittymässä näytettävän nimen ja uri_paths määrittää osoitteen resurssille. Osoitteen kaarisuluilla osoitetaan, että tämä osoitteen osa on muuttuja, joka on käytettävissä liitännäisen vastaus funktioissa. Rajapintojen eri määritykset tehtiin käyttäen REST-UI -moduulin tarjoamaa graafista käyttöliittymää.

```
/**
 * Provides a resource for database watchdog log entries.
 *
 * @RestResource(
 *   id = "dblog",
 *   label = @Translation("Watchdog database log"),
 *   uri_paths = {
 *     "canonical" = "/dblog/{id}"
 *   }
 * )
 */
```

KUVA 12. Drupal REST huomautus esimerkki

Koodissa määritellään funktiot eri kyselytyypeille, GET-kyselyä varten toteutetaan ”get” niminen funktio ja POST-kyselyä varten ”post” niminen funktio. Rajapintojen palauttamien datojen hakuun käytettiin jo olemassa olleita verkkosivustolle luotuja funktioita ja luokkia. Funktiossa on käytössä objekti, joka sisältää kyselyn datat, ja osoitteessa olevat siihen määritellyt muuttujat.

Rajapintojen autentikaatio toteutettiin käyttäen JSON Web Tokenia (JWT), jota varten Drupalissa käytettiin JWT moduulia. JWT:n hakuun luotiin uusi autentikaation tarjoaja moduuli (engl. authentication provider), jotta kirjautuminen mobiilisovelluksesta voitiin tehdä halutulla tavalla. Sisäänkirjautumista varten luotiin REST-rajapinta, joka vastaanottaa POST-kyselyssä käyttäjätunnuksen ja salasanan, jotka varmistetaan.

4 YHTEENVETO

Työn lopputuloksena on testiympäristössä toimiva mobiilisovellus, jossa on toteutettuna kaikki projektin aloitusvaiheessa määritellyt ominaisuudet. Sovellus ei ole vielä täysin julkaisuvalmis vaan se vaatii vielä testausta.

Mobiili sovellukseen valittu React Native -ohjelmistokehys toimi hyvin sovellusta Android-laitteille kehitettäessä, iOS-laitteille kehitystä ei tehty, koska siihen vaaditaan macOS-käyttöjärjestelmä, mutta sovellus luultavasti toimisi myös iOS-laitteilla, koska kehityksen aika valittujen kolmannen osapuolen moduulien valintakriteereihin kuului niiden toimivuus molemmilla alustoilla. Yksi React Nativen haasteista oli se, että uutena tekniikkana siihen julkaistiin kuukausittain uusi versio.

Palvelin ohjelmistokehyksenä Drupal on toimiva ratkaisu ja siihen löytyy paljon kehittäjäyhteisön luomaa materiaalia ja moduuleja. Toisinaan oli haasteellista löytää yhteisön luomista materiaaleista ja moduuleista ajantasaista tietoa.

Kokonaisuudessaan kehitystyö onnistui hyvin. Haluttujen ominaisuuksien vaatimien komponenttien ja moduulien löytäminen oli helppoa ja haluttu tulos saatiin usein ensimmäisenä valitulla ratkaisulla. Sovellusta tehdessä olisi ollut hyvä olla paremmat suunnitelmat siitä millainen käyttöliittymän tulisi olla.

Jatkokehitystä voisi tehdä tuomalla lisää verkkosivuston ominaisuuksia sovellukseen ja käyttökokemusta voisi vielä hioa.

LÄHTEET

Budiu Raluca. 2013. Mobile: Native Apps, Web Apps, and Hybrid Apps. https://www.nngroup.com/articles/mobile-native-apps/

Drupal. 2018a. Luettu 2.4.2018. https://www.drupal.org/

Drupal. 2018b. Luettu 2.4.2018. https://www.drupal.com/

Drupal. 2017. PSR-4 namaspaces and autoloading in Drupal 8. Luettu 11.4.2018. https://www.drupal.org/docs/develop/coding-standards/psr-4-namespaces-and-autoloading-in-drupal-8

Facebook. 2018. React Native. Luettu 2.4.2018. https://facebook.github.io/react-native/

Ionic. 2018. https://ionicframework.com/docs/

Klimenko Anna. 2018. Native vs Hybrid Mobile App Development: What to Choose in 2018?. https://greenice.net/native-vs-hybrid-mobile-app-development-choose/

MDN web docs. Using Promises. Luettu 2.4.2018. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Using_promises

Redux. 2018. Luettu 2.4.2018. https://redux.js.org/

Xamarin. 2018. https://www.xamarin.com/platform