SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

WEB-KEHITYS ANGULAR-TEKNIIKALLA

TEKIJÄ/T: Valtteri Korhonen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma/Tutkinto-ohjelma Tietotekniikan tutkinto-ohjelma	
Työn tekijä(t) Valtteri Korhonen	
Työn nimi Web-kehitys Angular-tekniikalla	
Päiväys 3.4.2020	Sivumäärä/Liitteet 27
Ohjaaja(t) Jussi Koistinen, Keijo Kuosmanen	
Toimeksiantaja/Yhteistyökumppani(t) Prasos Oy	

Tiivistelmä

Tässä opinnäytetyössä kehitettiin web-sovellus tilaajan palvelua käyttävien asiakkaiden tunnistamista varten. Tarkoituksena oli luoda sovellus, jossa asiakas voi tunnistautua joko tilaajan toiseen palveluun luodun käyttäjätunnuksen tai pankkitunnistautumisen avulla. Lisäksi työssä oli tavoitteena kehittää tilaajan olemassa oleva Angular-sovellus toimivammaksi mobiililaitteilla lisäämällä sovellukseen PWA-ominaisuuksia ja muuttamalla palvelun ulkoasua käyttäjäystävällisemmäksi mobiililaitteilla.

Työssä ensin toteutettiin tunnistautumissovellus Angular-tekniikalla. Seuraavaksi tilaajan sovellukseen toteutettiin PWA:n ydintoiminnot. Tämän jälkeen tutkittiin PWA:n mahdollistamia lisäominaisuuksia. Sovellukseen päätettiin toteuttaa QR- ja viivakoodiskanneri kameran avulla. Lopuksi suunniteltiin mobiiliulkoasun parantamiseksi tarvittavat muutokset ja toteutettiin ne.

Opinnäytetyön tuloksena saatiin toimiva web-sovellus yrityksen asiakkaiden tunnistamista varten. Lisäksi tilaajan sovellukseen toteutettiin sekä toiminnallisia että ulkoasumuutoksia parantamaan sivuston käytettävyyttä mobiililaitteilla. Opinnäytetyössä toteutettu sovellus ja lähes kaikki asiakkaan sovellukseen tehdyt muutokset ovat jo käytössä tuotantoympäristössä.

Avainsanat
Angular, PWA, TypeScript

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Valtteri Korhonen			
Title of Thesis Web Development with Angular Framework			
Date	3 April 2020	Pages/Appendices	27
Supervisor(s)			
Client Organisation /Partners Prasos Oy			

Abstract

The objective of this thesis was to create a web application for authenticating the identity of customers using the client organisation's service. The idea of the application is that the user can authenticate using the bank identification or account created in the organisation's other service. Another objective was to develop an existing web application more suitable for mobile users by adding PWA features and enhancing the user interface.

The project started with developing the identification application. Next step was to implement the core features of PWA to another application. After that additional features made possible by PWA were investigated. It was decided that Qr and barcode scanners using device's own camera were to be implemented. Finally, the changes for enhancing the user interface were planned and implemented.

The result of the thesis was a working web application for identifying customers. Additionally, the client's service was successfully updated with modern features and an enhanced user interface for mobile users. The application developed during this project and most of the updates to the existing application are already in use in production environment.

Keywords
Angular, PWA, TypeScript

SISÄLTÖ

1 JOHDANTO

Suoritin tutkintoon kuuluvat työharjoittelut Prasos Oy:ssä ja sain niiden jälkeen työpaikan yrityksestä. Sain ensimmäiseksi työtehtäväkseni harjoittelun päätyttyä toteuttaa web-sovellus asiakkaiden tunnistamista varten ja päätin ottaa sen opinnäytetyöni aiheeksi. Yritys oli edellisenä kesänä saanut virallisen maksulaitoksen toimiluvan Suomen finanssivalvonnalta, jonka seurauksena yritykseltä vaadittiin kaikkien ei-satunnaisten asiakkuuksien tunnistamista. Toteuttamani web-sovellus liittyy olennaisesti tähän prosessiin. Yritys omistaa useita kryptovaluuttojen vaihtoautomaatteja, Bittimaatteja, joiden avulla asiakkaat voivat vaihtaa käteistä rahaa bitcoineiksi tai bitcoineja käteiseksi. Kun asiakas asioi Bittimaatilla, häntä pyydetään tunnistautumaan web-sovelluksen avulla. Tunnistautumisen jälkeen asiakas saa vaadittavan QR-koodin osto- ja myyntitapahtumia varten. Kehittämässäni sovelluksessa asiakas pystyy tunnistautumaan joko pankkitunnuksilla tai yrityksen toisen palvelun, Coinmotionin, käyttäjätilin avulla.

Lisäksi opinnäytetyössä kehitettiin yrityksen pääpalvelu Coinmotionin mobiiliominaisuuksia. Coinmotion oli alun perin toteutettu ensisijaisesti tietokoneelle sopivaksi, joten sen käytettävyydessä mobiililaitteilla huomattiin parantamisen varaa. Opinnäytetyössä toteutettavaksi kokonaisuudeksi muodostui sivuston muuttaminen PWA-applikaatioksi ja sen mahdollistamien uusien ominaisuuksien kehittäminen. Lisäksi suunniteltiin ja toteutettiin koko sivuston ulkoasun muuttaminen selkeämmäksi ja käyttäjäystävällisemmäksi mobiilialustalla.

2 KÄYTETTÄVÄT TEKNIIKAT

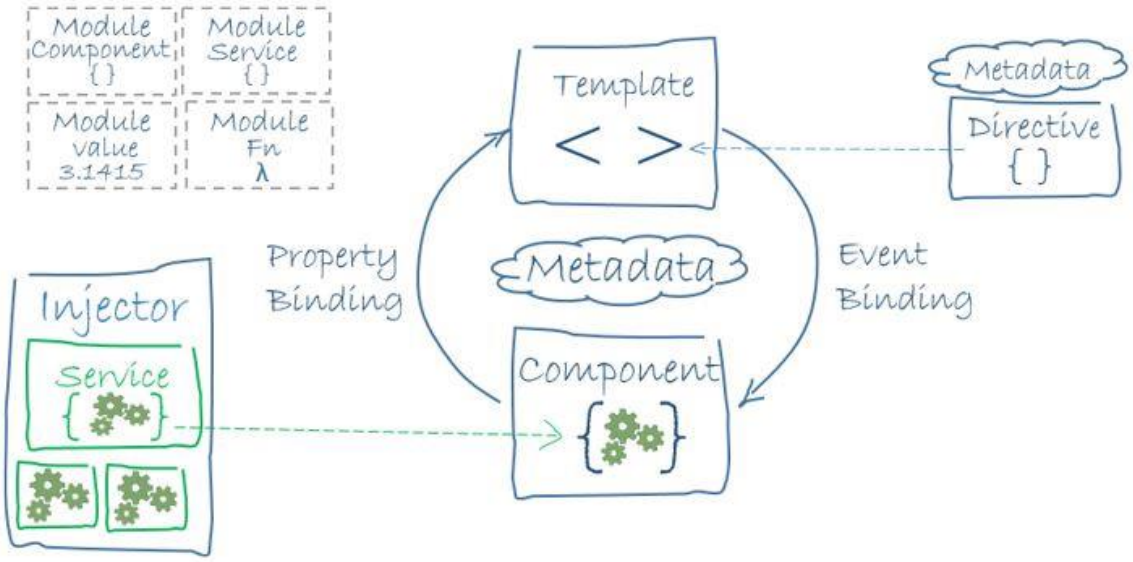
Opinnäytetyö toteutettiin Angular-ohjelmointialustalla. Yrityksellä oli ennestään Angularilla kehitettyjä sivustoja, joten oli luonnollinen valinta toteuttaa myös uusi tarvittava sivusto tutulla ja hyväksi todetulla tekniikalla.

2.1 Angular

Angular on ohjelmointialusta, jolla voi toteuttaa helposti erilaisia web-pohjaisia sovelluksia. Angular käyttää ulkoasun luomiseen HTML:ää ja toiminnallisuus toteutetaan TypeScriptillä.

Angularin perusrakennuspalikoita ovat NgModulet ja jokaisessa sovelluksessa on ainakin yksi tällainen moduuli, joka toimii sovelluksen juurena. Usein sovelluksessa kuitenkin on useampi moduuli, joiden avulla sovelluksen ominaisuudet toteutetaan. Moduulit voivat käyttää muiden moduulien toteuttamaa toiminnallisuutta tai jakaa omat toiminnot muiden moduulien käyttöön. Esimerkiksi jos sovelluksessa haluaa käyttää reititystä, voidaan sovellukseen tuoda Angularin tarjoama reititysmoduuli. Tällainen rakenne helpottaa hallitsemaan monimutkaisempia sovelluksia eristämällä toiminnot omiksi kokonaisuuksikseen.

Moduulit koostuvat komponenteista, jotka määrittävät kaikki näytöllä olevat näkymät. Jokaiseen komponenttiin kuuluu luokka, joka sisältää komponenttiin liittyvät tiedot ja toiminnallisuuden. Lisäksi luokkaan liitetään HTML-malli, joka määrittää komponentin ulkoasun. Mallissa voi käyttää myös Angularin määrittelemiä toimintoja, jotka muokkaavat HTML-elementtejä ennen kuin ne tulevat näytölle. Näiden avulla komponenttien ulkoasua voidaan muokata toimintalogiikan mukaan tai reagoida erilaisiin tapahtumiin, esimerkiksi kun käyttäjä painaa nappia. Komponentille voi myös määritellä omat tyylit css-tyylitiedoston avulla. Jos jotain tietoa tai toimintoja tarvitaan useammassa kuin yhdessä komponentissa avuksi voi luoda palveluluokan. Palvelu voidaan injektoida jokaiseen komponenttiin, jossa sen toteuttamia toimintoja tarvitaan. Tämän avulla komponentit pysyvät mahdollisimman pieninä ja selkeinä. (Google LLC, 2020)



Kuva 1 Angular sovelluksen rakenne. (Google LLC, 2020)

2.1.1 TypeScript

Typescript on Microsoftin kehittämä ohjelmointikieli, joka jatkaa Javascriptin toiminnallisuuttaa. Selaimet eivät osaa suorittaa Typescriptiä, joten se täytyy kääntää Javascriptiksi ennen suorittamista. Typesriptin voidaan ajatella siis olevan eräänlainen kerros Javascriptin päällä. Suurin ero Javascriptin ja Typesriptin välillä on se, että Typescriptin muuttujille ja olioille voi asettaa tyypin. Tämän ansiosta koodille muodostuu selvä rakenne ja se on helpommin luettavaa. Lisäksi kääntäjä automaattisesti huomaa enemmän koodissa olevia virheitä. (Gagliardi, 2020) Alla olevassa kuvassa on esimerkki Typescriptillä tehdystä luokasta.

```
interface Link {
  description: string;
  id: number;
  url: string;
}
```

Kuva 2 Typescript muuttujien tyyppi. (Gagliardi, 2020)

2.1.2 Sass

Sass on ohjelmointikieli, joka jatkaa css:än toiminnallisuutta. Sass-tiedostot täytyy kääntää css-tiedostoiksi ennen kuin selaimet osaavat suorittaa niitä. Sass:in päätarkoitus on vähentää tarvetta toistaa asioita koodissa. Hyödyllisimpiä ominaisuuksia tämän saavuttamiseen ovat muuttujat, sisäkkäiset lohkot ja moduulit. Muuttujaan voi määrittää esimerkiksi fontin

värin ja käyttää samaa muuttujaa kaikkialla. Näin väriä muutettaessa tarvitsee muuttaa vain yhden muuttujan arvo.

SCSS Sass

```scss
$font-stack:    Helvetica, sans-serif;
$primary-color: #333;

body {
  font: 100% $font-stack;
  color: $primary-color;
}
```

CSS

```css
body {
  font: 100% Helvetica, sans-serif;
  color: #333;
}
```

Kuva 3 Sass muuttujan käyttäminen. (Google Inc., 2020)

Sisäkkäiset koodilohkot vähentävät toistoa erityisesti, kun elementin sisällä on useita elementtejä, joille halutaan määrittää omat tyylit.

SCSS Sass

```scss
nav {
  ul {
    margin: 0;
    padding: 0;
    list-style: none;
  }

  li { display: inline-block; }

  a {
    display: block;
    padding: 6px 12px;
    text-decoration: none;
  }
}
```

CSS

```css
nav ul {
  margin: 0;
  padding: 0;
  list-style: none;
}
nav li {
  display: inline-block;
}
nav a {
  display: block;
  padding: 6px 12px;
  text-decoration: none;
}
```

Kuva 4 Sass koodilohkojen asettelu sisäkkäin. (Google Inc., 2020)

Moduulit mahdollistavat usean eri tiedoston käyttämisen. Näin eri osa-alueet voidaan laittaa omiin tiedostoihin ja siten koodin rakenne pysyy selkeänä. Tämän avulla voi esimerkiksi määrittää muuttujat yhdessä tiedostossa ja tuoda kyseinen tiedosto kaikkialle missä sitä tarvitaan. (Google Inc., 2020)

Kuva 5 Sass moduulirakenne. (Google Inc., 2020)

2.2 PWA

PWA eli Progressive Web App on web-sovellus, joka on kehitetty käyttäen sellaisia menetelmiä, että se voi käyttää sekä web-sovelluksen että natiivi mobiilisovelluksen toimintoja. PWA on siis vaihtoehto natiiville mobiilisovellukselle. PWA-sovellus on pohjimmiltaan vain web-sovellus eikä sellaista voi luoda yhdellä teknologialla, joten ei ole aina itsestään selvää, milloin sovellus on PWA. On olemassa joitakin pääominaisuuksia, joiden avulla voi todeta web sovelluksen olevan PWA-sovellus:

- Sovelluksen täytyy olla asennettavissa älypuhelimeen. Asennuksen jälkeen sovellus avautuu omaan ikkunaan ja sitä voi käyttää niin kuin tavallista sovellusta.

- Sovelluksen täytyy ilmoittaa tietoja kuten sovelluksen nimi ja ikoni web app manifestissa. Tämän avulla puhelimet osaavat luoda kuvakkeen aloitusnäytölle.

- Sovelluksen täytyy olla linkitettävissä URL-osoitteen avulla. Tällöin ei ole tarvetta sovelluskaupalle ja sovellus on todella helppo jakaa.

- Sovelluksen täytyy toimia ilman verkkoyhteyttä. Sovellus tallentaa mahdollisimman paljon tietoa laitteelle ja mahdollistaa siten toiminnan ilman verkkoyhteyttä. Jos jokin sivu vaati verkkoyhteyden, ulkoasu määritetään sen mukaan eikä näytetä selaimen omia virheilmoituksia.

- Sovellus voidaan kehittää käyttämään uusimpia toimintoja mutta se toimii silti hyväksyttävästi myös vanhemmilla laitteilla ja selaimilla.

- Sovellus käyttää hyväksi ilmoituksia. Sovellus voi aktivoida käyttäjiä ilmoittamalla päivityksistä ja uusista ominaisuuksista, vaikka käyttäjä ei sillä hetkellä olisi käyttämässä sovellusta tai laitetta.

- Sovellus on responsiivinen eli se näyttää hyvältä ja on helppo käyttää millä tahansa laitteella riippumatta näytön koosta.

- Sovellus on turvallinen käyttää. PWA-sovelluksissa yhteys palvelimelle on salattu käyttäen HTTPS-protokollaa, joka estää henkilökohtaisen datan päätymisen ulkopuolisille toimijoille.

PWA-sovelluksen ydintoiminnot ovat suhteellisen helppo toteuttaa ja siitä saatavat hyödyt web-sovellukseen verrattaessa ovat merkittäviä. Asennuksen jälkeen sovelluksen latausajat pienevät merkittävästi ja välimuisti poistaa latausajat lähes kokonaan. Sovellusta päivitettäessä välimuistiin tarvitsee ladata uudestaan vain muuttuneet tiedostot. PWA-sovelluksen käyttökokemus on todella lähellä natiivia mobiilisovellusta. Ilmoitusten avulla asiakkaita saadaan aktivoitua käyttämään sovellusta enemmän.

PWA-sovelluksen toimivuus riippuu täysin käytettävästä selaimesta. Vuonna 2020 tunnetuimmat selaimet ovat toteuttaneet suurimman osan PWA:n toimintaan tarvittavista ominaisuuksista. Safari on ainoa selain, jolla on rajattu yhteensopivuus sovelluksen lisäämiseen aloitusnäytölle ja ilmoitusten näyttäminen puuttuu kokonaan. Useilla selaimilla ominaisuudet ovat kuitenkin vielä kokeiluvaiheessa, joten erilaiset ongelmat ovat mahdollisia. (Mozilla Contributors, 2019)

3 TYÖKALUT

Opinnäytetyössä käytettiin apuna monenlaisia eri työkaluja helpottamaan kehitystä.

3.1 Npm

Npm on pakettienhallintaohjelmisto, jonka avulla voi helposti hallita web-kehityksessä tarvittavia Javascript-paketteja. Npm:ään kuuluu komentorivillä käytettävä käyttöliittymä, jonka kautta projektiin voi liittää rekisterissä olevia paketteja. Npm on maailman laajin ohjelmistorekisteri, joten sieltä löytyy todella paljon valmiiksi tehtyjä toimintoja. Esimerkiksi seuraavaksi käsiteltävä Angular cli asennetaan Npm:än kautta. (npm, Inc. and Contributors)

3.2 Angular CLI

Angular cli on komentorivityökalu, jolla voi luoda Angular-sovelluksia ja lisätä projektiin uusia rakennuspalikoita. Uusi sovellus luodaan ng new komennolla. Tämä komento luo sovelluksen rungon, jota voi lähteä muokkaamaan. Runko sisältää juurimoduulin, juurikomponentin ja tarvittavat konfiguraatiotiedostot. ng generate komennolla voi luoda uusia moduuleja, komponentteja, palveluita ja muita tarvittavia toimintoja. (Google LLC, 2020)

3.3 Visual Studio Code

Visual Studio Code on tekstieditori, jonka tarkoituksena on helpottaa koodin kirjoittamista ja lukemista. Editoriin on mahdollista asentaa lisäosia, jotka lisäävät erilaisia toimintoja kuten uusia ohjelmointikieliä tai selaimen kanssa toimiva virheenetsintä. Visual Studio Codessa on sisäänrakennettu yhteensopivuus TypeScriptin kanssa. Lisäksi koodin korostus ja automaattinen täyttö ovat Visual Studio Codessa huippuluokkaa.

3.4 Git

Git on avoimen lähdekoodin versionhallintajärjestelmä, joka on hyvin laajasti käytössä ohjelmistokehitysalalla. Gitin perusperiaate on, että ohjelmistokehittäjät voivat ladata tiedostot Git-repostitoriosta, muokata niitä omalla tietokoneellaan ja päivittää muokatut tiedostot repositorioon. Gitin suurin ero useimpiin muihin versionhallintajärjestelmiin on se, että ohjelmistokehitystä varten voi luoda päähaarasta poikkeavan kehityshaaran, johon tarvittavat muutokset voi tehdä. Muutoksia voi sen jälkeen tarkastella ja testata erillään päähaarasta. Kun muutokset on todettu hyviksi ja toimiviksi ne voi liittää takaisin päähaaraan. Eri haarojen välillä voi liikkua vapaasti. Tämä mahdollistaa monen eri ominaisuuden kehittämisen

yhtä aikaa. Alla oleva kuva havainnollistaa ohjelmistokehitystä Gitin avulla. (Scott Chacon and others, 2018)

Kuva 6 Esimerkki Git kehityshaaroista. (Scott Chacon and others, 2018)

4 ANGULAR-SOVELLUS

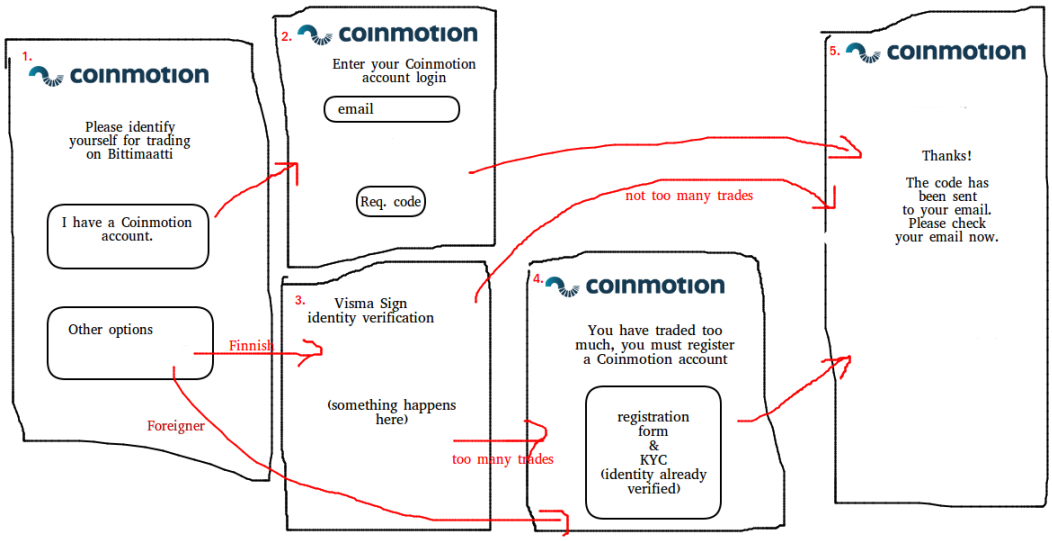
Tässä kappaleessa käydään läpi opinnäytetyössä toteutetun sovelluksen suunnittelu ja kehitys.

4.1 Suunnittelu

Tarvittavan sovelluksen toiminnot olivat suurimmilta osin valmiiksi suunniteltuna ja sovelluksella oli todella tiukka aikataulu, joten sovelluksen suunnittelu tämän opinnäytetyön sisällä jäi vähäiseksi. Työ päätettiin jo alkuvaiheessa tehdä Angularilla koska yrityksellä oli jo valmiiksi käytössä yksi Angular web-sovellus ja se oli todettu hyväksi ja toimivaksi tekniikaksi.

Sovelluksen ideana on, että käyttääkseen yrityksen Bittimaatti-palvelua, asiakkaan täytyy ensin tunnistautua kehitettävän sovelluksen avulla. Tunnistauduttuaan asiakas saa qr-koodin, jonka avulla asiakas pystyy asioimaan Bittimaatilla. Koodin avulla Bittimaatti osaa yhdistää tapahtuman tiettyyn asiakkaaseen.

Sovellukseen haluttiin tehdä kaksi eri tapaa, joilla asiakas voi tunnistautua. Ensimmäinen tapa on käyttää Coinmotion-käyttäjätiliä. Asiakas voi syöttää sovellukseen käyttäjätilin sähköpostin, ja saa sen jälkeen tunnistautumisen vahvistavan koodin sähköpostiinsa. Toinen tapa on tunnistautua suomalaisella pankkitunnuksella. Käyttäjän tunnistauduttua, sovellus näyttää tarvittavan koodin. Jos asiakas ei pysty tunnistautumaan kummallakaan tavalla, hänet ohjataan rekisteröitymään Coinmotioniin. Rekisteröidyttyään asiakas voi tunnistautua juuri luomansa tilin avulla. Alla olevassa kuvassa on ensimmäinen suunnitelma sovelluksen rakenteesta.



Kuva 7 Angular sovelluksen ensimmäinen suunnitelma

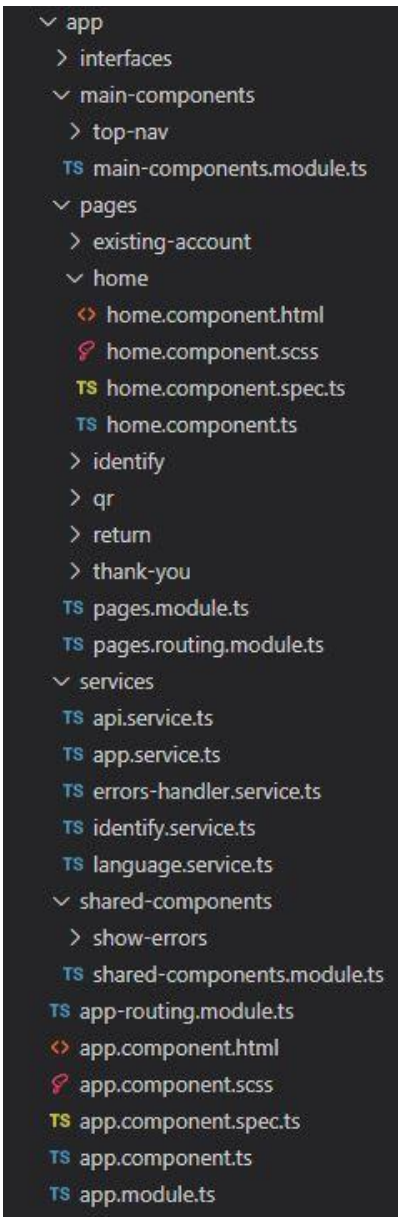
Työn alkuvaiheessa huomattiin, että osa sovelluksen toiminnoista oli jo toteutettu muualla ja niiden toteuttaminen uudestaan ei olisi järkevää. Suurin näistä oli Coinmotion tilin rekisteröinti. Aluksi oli tarkoitus tehdä Coinmotion-tilin rekisteröinti sovellukseen, mutta sen todettiin olevan liian työläs ylläpitää kahdessa paikassa. Suunnitelmaa päätettiin muuttaa siten että sovellus ohjaa asiakkaat Coinmotioniin rekisteröitymään.

4.2 Toteutus

Sovelluksen toteutus aloitettiin luomalla uusi Angular-projekti Angular CLI:n avulla. Sovellukseen tehtiin ensin tarvittavat sivut ja luotiin niille ulkoasu. Jotta sivustolle saatiin yhtäläinen tyyli Coinmotionin kanssa, sovellukseen tuotiin osa Coinmotionin tyylitiedostoista ja käytettiin mahdollisimman paljon näitä valmiita tyylejä. Alla olevassa kuvassa näkyy sovelluksen etusivu. Kaikki muutkin sivut ovat tyyliltään samanlaisia ja hyvin minimalistisia.

Kuva 8 Sovelluksen etusivu.

Lopuksi sivuille toteutettiin toiminnot ja reititys. Alla olevasta kuvasta voi tarkastella sovelluksen lopullista rakennetta.



Kuva 9 Angular sovelluksen rakenne.

4.2.1 Tunnistautuminen

Tunnistautuminen pankkitunnuksilla toteutettiin Nets-palvelun avulla. Tunnistautuminen avautuu iframe-elementtiin tunnistautumissivulla.

Kuva 10 Pankkitunnistautumisen etusivu.

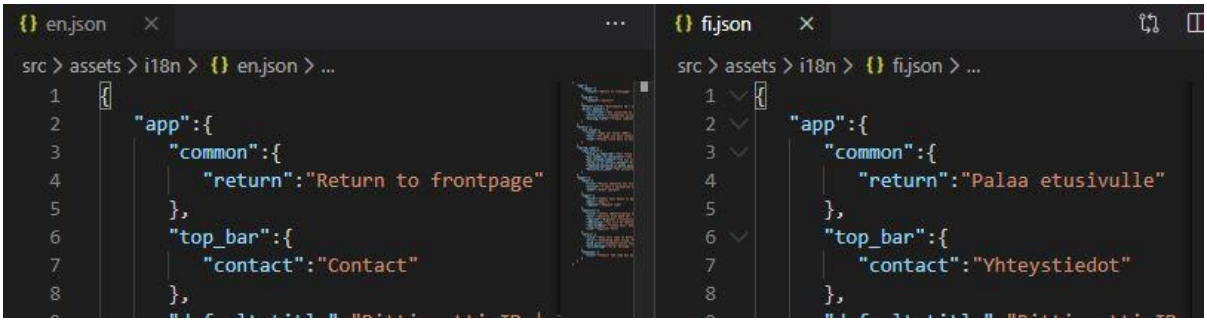
Kun tunnistautumisen suorittaa loppuun Nets ohjaa käyttäjän paluu-sivulle ja luo urlin query-osaan tiedot, joiden avulla asiakkaan tiedot saa haettua. Nämä tiedot lähetetään palvelimelle kutsumalla Coinmotionin rajapintaa, joka palauttaa merkkijonon, joka näytetään qr-koodina asiakkaalle.

```
this.identifyService.finalize({"params": {"code": this.params.code, "nonce": this.params.nonce, "state": this.params.state},
.subscribe(result => {
    if (result.cmidToken != null) {
      this.value = String(result.cmidToken);
    }
    else{
      this.error = true;
    }
  }, error => {
    window.console && window.console.log && console.log(error);
    this.error = true;
  }
);
```

Kuva 11 Tunnistautuneen asiakkaan käsittely.

4.2.2 Sovelluksen kieli

Sovellus haluttiin toteuttaa kaksikielisenä. Tätä varten sovellukseen tuotiin palvelu, jonka avulla kielen vaihtaminen oli helppo toteuttaa. Kaikki sovelluksen tekstit laitettiin käännöstiedostoihin, joista palvelu käy hakemassa tekstin käytössä olevan kielen mukaan.



Kuva 12 Käännöstiedostot.

Kielen valinta toteutettiin joka sivulla näkyvään yläpalkkiin. Kääntäjäpalvelun toimintaa hallitsemaan luotiin language.service-palvelu, jonne toteutettiin kaikki kieleen liittyvät toiminnot. Kaikki komponentit saavat palvelun kautta tarvitsemansa toiminnot. Tässä vaiheessa käytännössä ainoa komponenttien tarvitsema toiminto on kielen vaihtaminen mutta tällä periaatteella toimintoja on todella helppo lisätä. Palvelu tallentaa käytössä olevan kielen selaimessa olevaan keksiin, jonka avulla sovellus tietää automaattisesti käyttäjän kielen seuraavilla käyttökerroilla. Palvelun toteutus on nähtävillä alla olevassa kuvassa.

```
export class LanguageService {
  isTranslationsUpdated: BehaviorSubject<string> = new BehaviorSubject(this.translate.currentLang);

  constructor (private cookieService: CookieService,
               public translate: TranslateService
  ) {
    this.initLanguage();
  }

  switchLanguage (language: string) {
    document.documentElement.lang = language;
    this.setLanguageCookie(language);
    this.translate.use(language);
    this.isTranslationsUpdated.next(language);
  }


  setLanguageCookie (language: string) {
    if (language && window.self === window.top){
      const expiredDate = new Date();
      expiredDate.setDate(expiredDate.getDate() + 180);
      this.cookieService.delete('lang');
      this.cookieService.set('lang', language, expiredDate, '/');
    }
  }

  initLanguage (){
    if (this.cookieService.check('lang') && window.self === window.top) {
      var lang = this.cookieService.get('lang');
      this.translate.use(lang);
      document.documentElement.lang = lang;
      this.isTranslationsUpdated.next(lang);
    }
  }
}
```

Kuva 13 Language.service-palvelu.

4.2.3 Reititys

Sovelluksen reititys toteutettiin Angularin tarjoamalla reititysmoduulilla. Sivujen osoitteet ja ladattavat komponentit määritettiin omaan moduuliin ja välitettiin tiedot reititysmoduulille. Näin sovellus osaa näyttää oikean komponentin navigoitaessa tiettyyn osoitteeseen.

```
const Routes: Routes = [
  {path: '', component: HomeComponent},
  {path: 'login', component: ExistingAccountComponent},
  {path: 'identify', component: IdentifyComponent},
  {path: 'identify/return', component: ReturnComponent},
  {path: 'qr/:value', component: QrComponent},
  {path: 'thankyou', component: ThankYouComponent},
  {path: '**', redirectTo: ''}
];

export const PagesRouting: ModuleWithProviders = RouterModule.forChild(Routes);
```

Kuva 14 Sovelluksen reititys.

5 PWA JA MOBIILIKÄYTTÖLIITTYMÄ

Tässä kappaleessa käsitellään opinnäytetyössä toteutettu Coinmotionin kehitys. Coinmotion on kryptovaluuttojen vaihto- ja säilytyspalvelu. Työssä toteutettiin Coinmotionin muuttaminen PWA-applikaatioksi ja käyttöliittymän parantaminen mobiililaitteilla. PWA:n ansiosta palveluun olisi myös mahdollista lisätä useita käytettävyyttä parantavia ominaisuuksia.

5.1 Suunnittelu

Ominaisuuksien suunnittelu alkoi tutkimalla PWA:n toteuttamista Angularilla. Seuraavaksi otettiin selvää mitä ominaisuuksia PWA:n avulla sovellukseen voidaan luoda. Hyödyllisimmäksi mahdolliseksi ominaisuudeksi todettiin ilmoitusten näyttäminen asiakkaille. Nopeasti kuitenkin selvisi, että sen toteuttaminen vaatisi isoja muutoksia palvelimen päähän, joten tästä ominaisuudesta päätettiin luopua.

Kryptovaluuttalompakoiden osoitteita on todella helppo jakaa qr-koodien avulla, joten sellaisen skannaaminen kameralla todettiin hyödylliseksi ominaisuudeksi. Lisäksi selvisi, että se on helppo toteuttaa PWA:n avulla. Sovelluksella voi myös maksaa laskuja käyttäjätilillä olevilla varoilla. Tätä helpottamaan päätettiin myös tehdä laskun viivakoodin skanneri samaa toteutusta hyödyntäen.

Sovellus oli alun perin tehty pääasiassa tietokoneella käytettäväksi ja mobiiliulkoasun huomiointi oli kehitysvaiheessa jäänyt taka-alalle. PWA-toteutuksen yhteydessä päätettiin varmistaa, että koko sovellus toimii hyvin myös mobiililaitteilla. Tätä varten kaikki sovelluksen sivut käytiin läpi ja kirjattiin ylös muutostarpeet toteutusta varten. Suunnittelun avuksi selvitettiin hyväksi todettuja tapoja suunnitella mobiilisovelluksen ulkoasu. Suunnittelussa kiinnitettiin erityisesti huomiota seuraaviin asioihin:

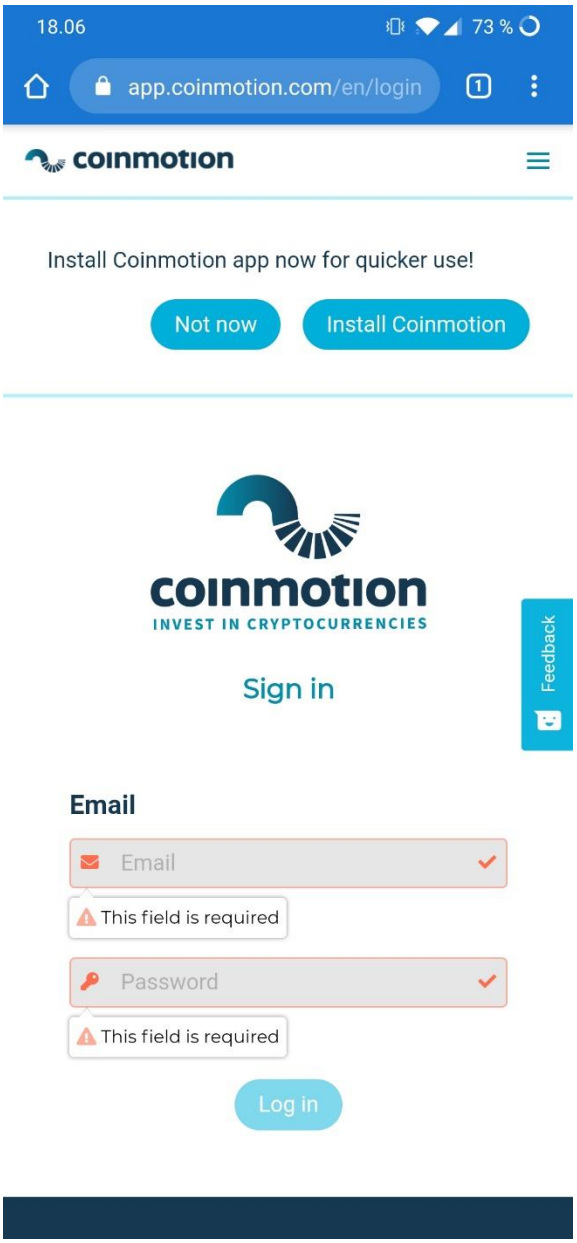
- Oleellisimman sisällön täytyy olla näkyvissä ilman skrollaamista.
- Klikattavat elementit ovat tarpeeksi suuria kosketusnäytölle.
- Ulkoasu näyttää hyvältä riippumatta näytön koosta.

5.2 Toteutus

5.2.1 PWA ominaisuudet

Toteutus aloitettiin lisäämällä PWA:n ydintoiminnot Angular-projektiin. Tämän sai helposti toteutettua yhdellä Angular cli komennolla. Seuraavaksi täytyi asettaa komennon luomille ominaisuuksille halutut asetukset ja lisätä tarvittavat ikonit. Coinmotionin Angular projekti ei ollut suoraan yhteensopiva PWA-ominaisuuksien luomisessa käytetyön työkalun kanssa. Tämän takia sovellukseen täytyi erikseen lisätä toiminto, jonka avulla luodut ominaisuudet saatiin käyttöön.

Seuraavaksi luotiin komponentti, jonka avulla asiakkaat voivat asentaa sovelluksen aloitus-näytölleen. Ilmoitus tulee näkyviin, kun selain toteaa sivuston olevan PWA-applikaatio.



Kuva 15 Sovelluksen asentaminen.

Sovellukseen luotiin myös ilmoitus päivityksistä. Jos sovellus huomaa uudemman version olevan saatavilla, sovellus ilmoittaa siitä ja pyytää päivittämään sivun, jolloin sovellus lataa päivitetyt tiedostot palvelimelta ja korvaa välimuistissa olevat vanhat tiedostot uusilla.

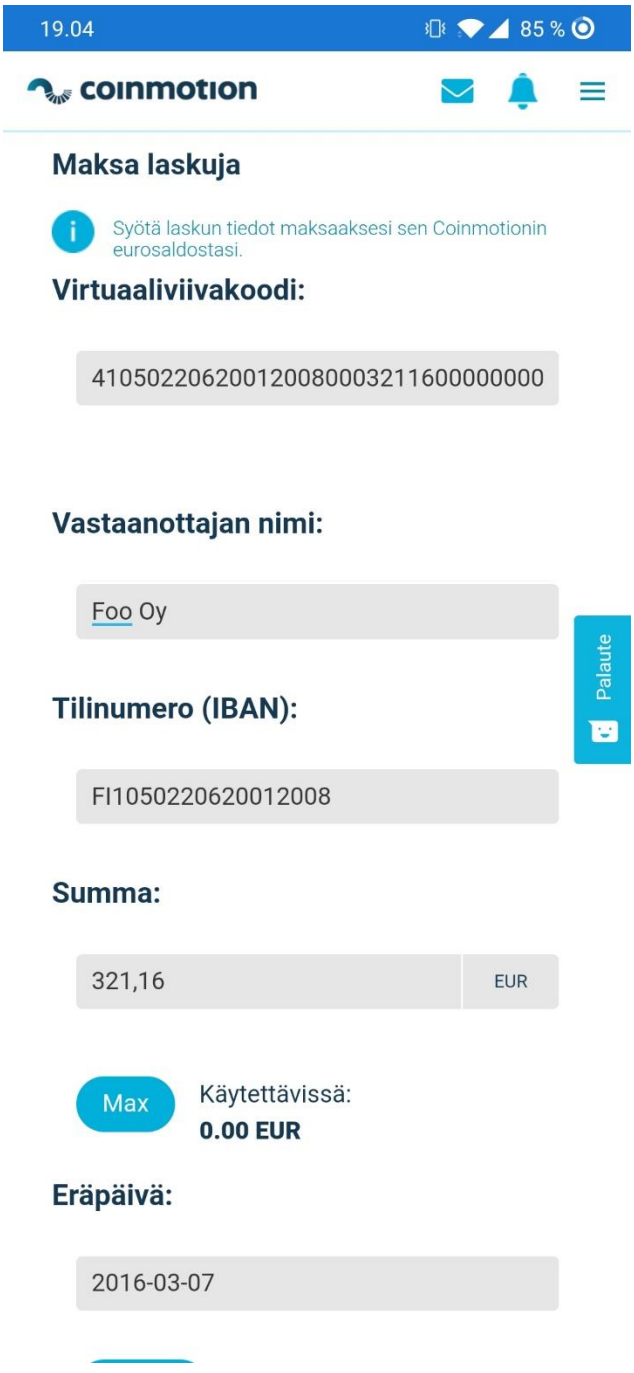
5.2.2 Kameran käyttäminen sovelluksessa

Qr- ja viivakoodien lukemista varten sovellukseen tuotiin kirjasto, jossa oli jo toteutettu tarvittavat ominaisuudet. Tämän avulla saatiin helposti luotua elementti, jonka sisällä näkyy kameran kuva. Tämän jälkeen toteutettiin toiminnot skannauksen aloittamiselle ja tuloksen käsittelemiselle. Alla olevassa kuvassa näkyy skannerin toimintojen toteutus. Käyttäjän painaessa skannaa nappia kysytään lupa käyttää laitteen kameraa ja sen jälkeen avataan skanneri. Kun koodi saadaan luettua, siitä kerätään tarvittavat tiedot ja asetetaan ne oikeisiin kenttiin.

```
onCodeResult(resultString: string): void {
  this.inputQr(resultString);
  this.checkDestinationTagInAddress();
  this.scanQr = false;
}

startScan(): void {
  this.userDevice.askForPermission().then(result => {
    if(result){
      this.scanQr = true;
    }
    else{
      this.appService.flashMessage.next({
        isActive   : true,
        title      : '',
        type       : 'info',
        message    : 'transfers.send.cryptocurrencies.scan_message',
        autoHideIn : 10000
      });
    }
  });
}

inputQr(qrResult: string): void{
  let rawQr = qrResult.split(":");
  let qrAddress = rawQr[rawQr.length-1];

  if(this.selectedCurrency.value==='xrp' || this.selectedCurrency.value==="xlm"){

    var combinedAddress = qrAddress.split("?");

    if(combinedAddress.length == 2){
      qrAddress = combinedAddress[0];
      let tag = combinedAddress[1].split("=")[1];
      this.formElement('destinationTag').setValue(tag);
    }
    else{
      this.formElement('destinationTag').setValue("");
    }
  }
```

Kuva 16 Qr-skannerin toiminnot.

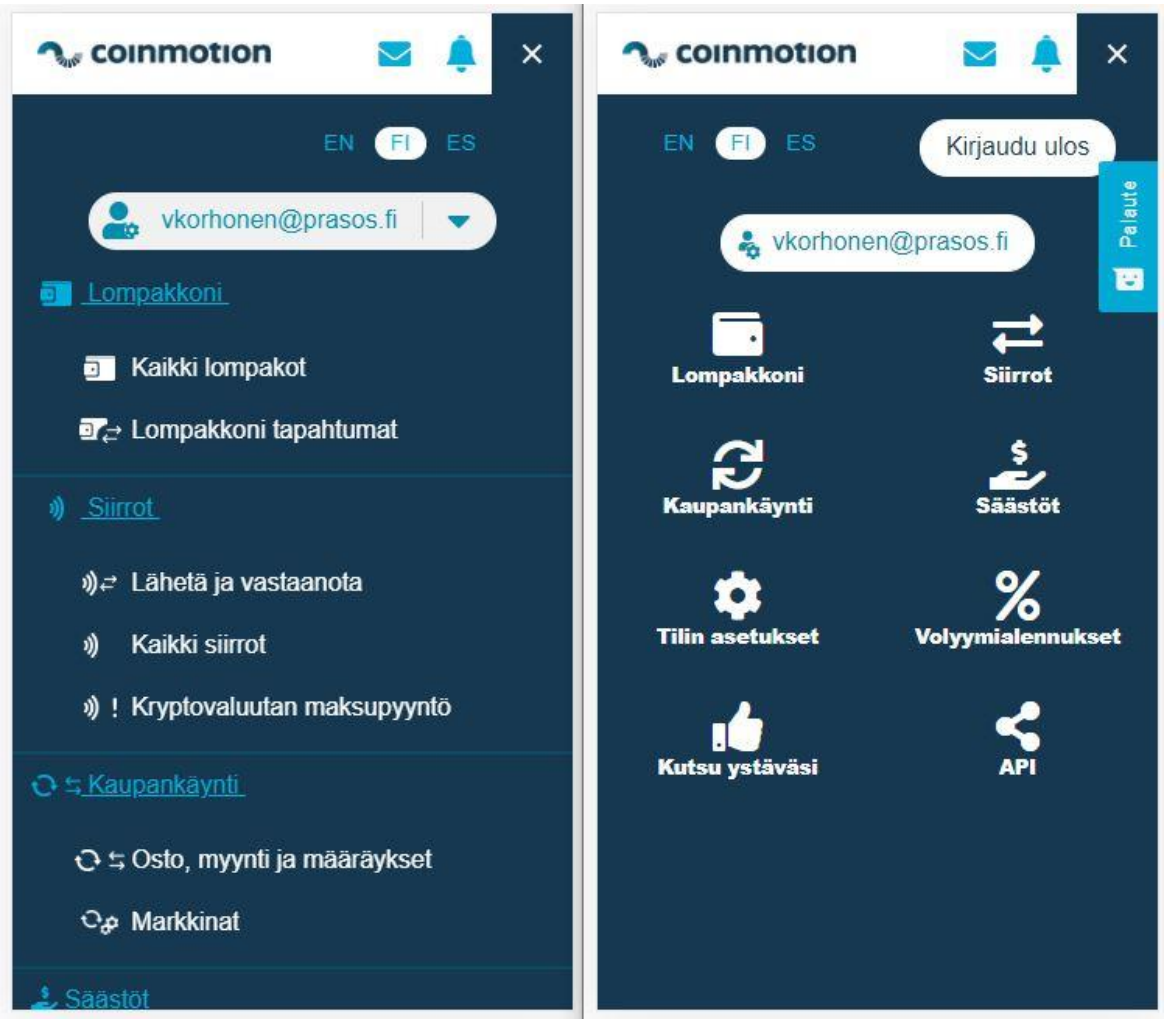
Samalla tavalla toteutettiin myös viivakoodiskanneri laskujen maksamista varten, mutta jostakin syystä se toimi todella epäluotettavasti eikä ongelmia saatu korjattua tämän opinnäytetyön aikana. Samaan yhteyteen tehtiin kuitenkin kenttä, johon laskun virtuaaliviivakoodin voi liittää ja nopeuttaa siten laskun maksamista.



Kuva 17 Laskun tietojen syöttäminen virtuaaliviivakoodin avulla.

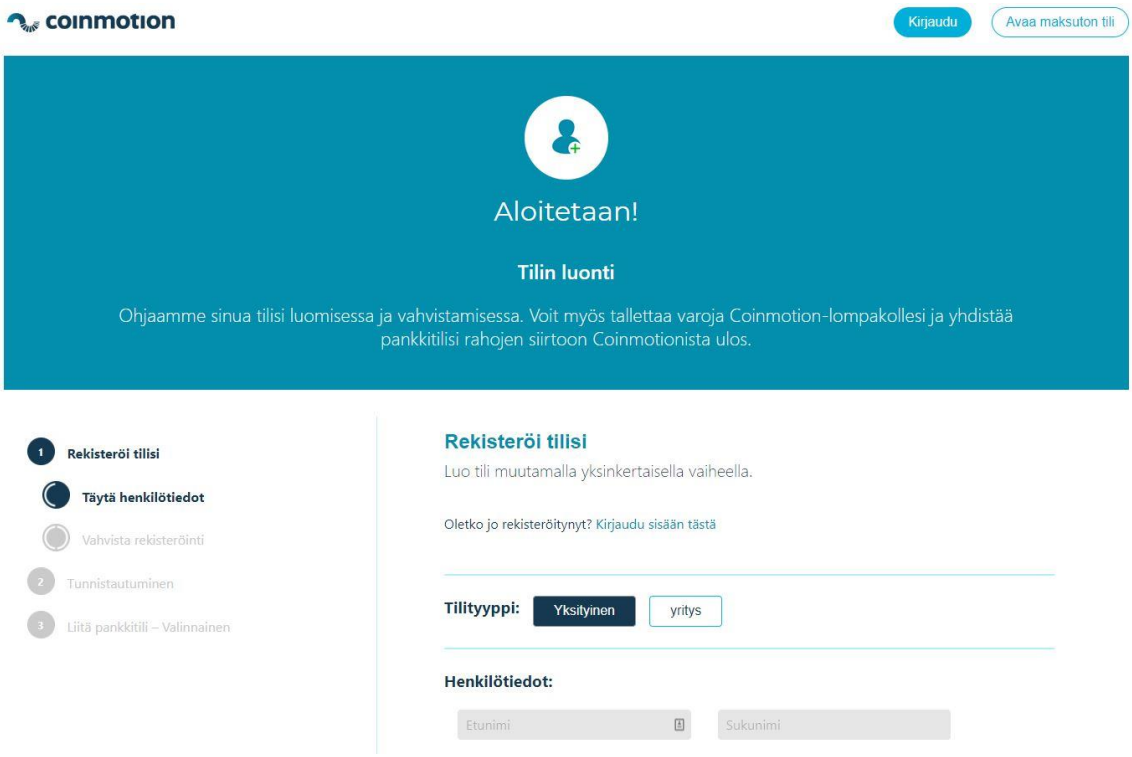
5.2.3 Ulkoasun toteutus

Kaikki ulkoasumuutokset toteutettiin pelkästään muuttamalla HTML-rakennetta ja tyylejä. Ensimmäisenä paranneltiin sivuston päävalikko. Valikon sisällä oleva valikko poistettiin ja siirrettiin siellä olevat napit selkeästi esille. Valikon rakennetta selkeytettiin poistamalla ylimääräiset alasivut. Näin valikosta saatiin todella selkeä ja kaikki toiminnot ovat nähtävillä yhdellä silmäyksellä.



Kuva 18. Päävalikko ennen ja jälkeen muutosten.

Seuraavaksi keskityttiin rekisteröintisivuun. Tällä sivulla oli erityisen hyvin nähtävissä, kuinka sivusto oli suunniteltu etupäässä isommalla näytöllä käytettäväksi. Sivulla on useita elementtejä antamassa lisätietoa rekisteröinnin vaiheista, mikä on toimiva ulkoasu tietokoneella, mutta mobiililaitteella ne vievät paljon tilaa ja hankaloittavat siten itse rekisteröitymistä. Alla olevista kuvista voidaan nähdä, kuinka eri tavalla ulkoasu toimii tietokoneella ja mobiililaitteella.



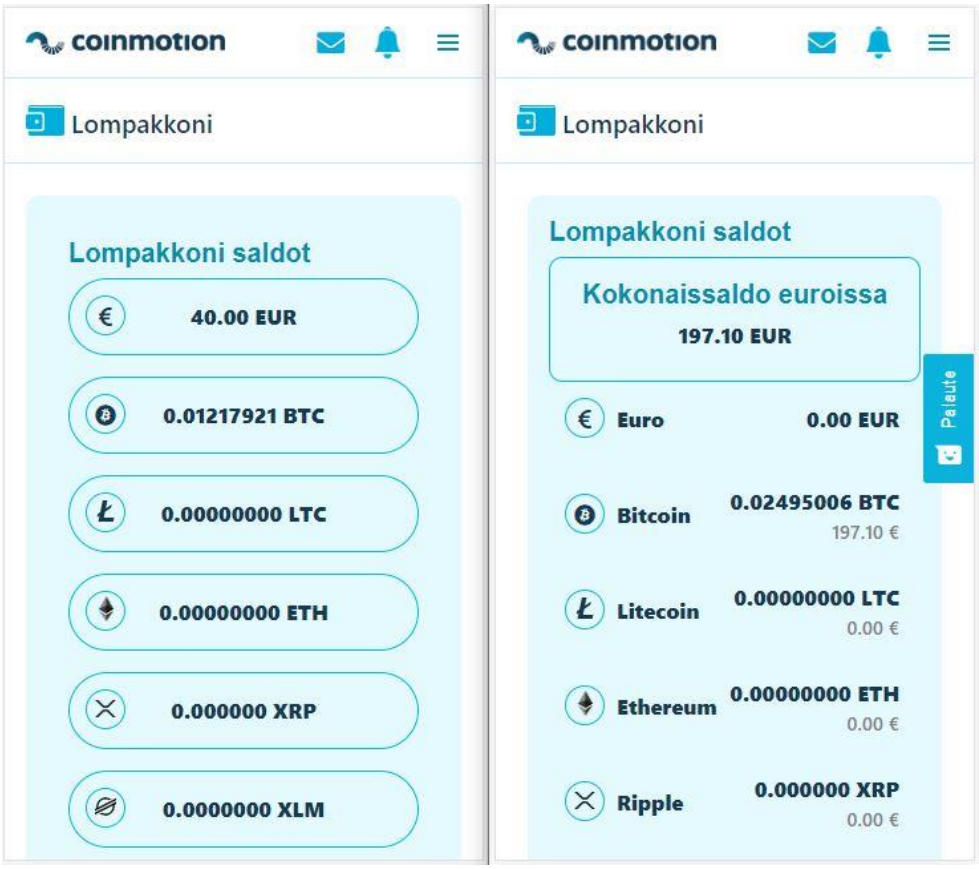
Kuva 19. Rekisteröintisivu tietokoneella.

Kuva 20. Rekisteröinti mobiililaitteella.

Tilaa vievät elementit piilotettiin tilin luontia lukuun ottamatta kaikissa rekisteröinnin vaiheissa. Sivun tärkein sisältö on näin heti näkyvissä ja koko rekisteröinti helpottuu huomattavasti.

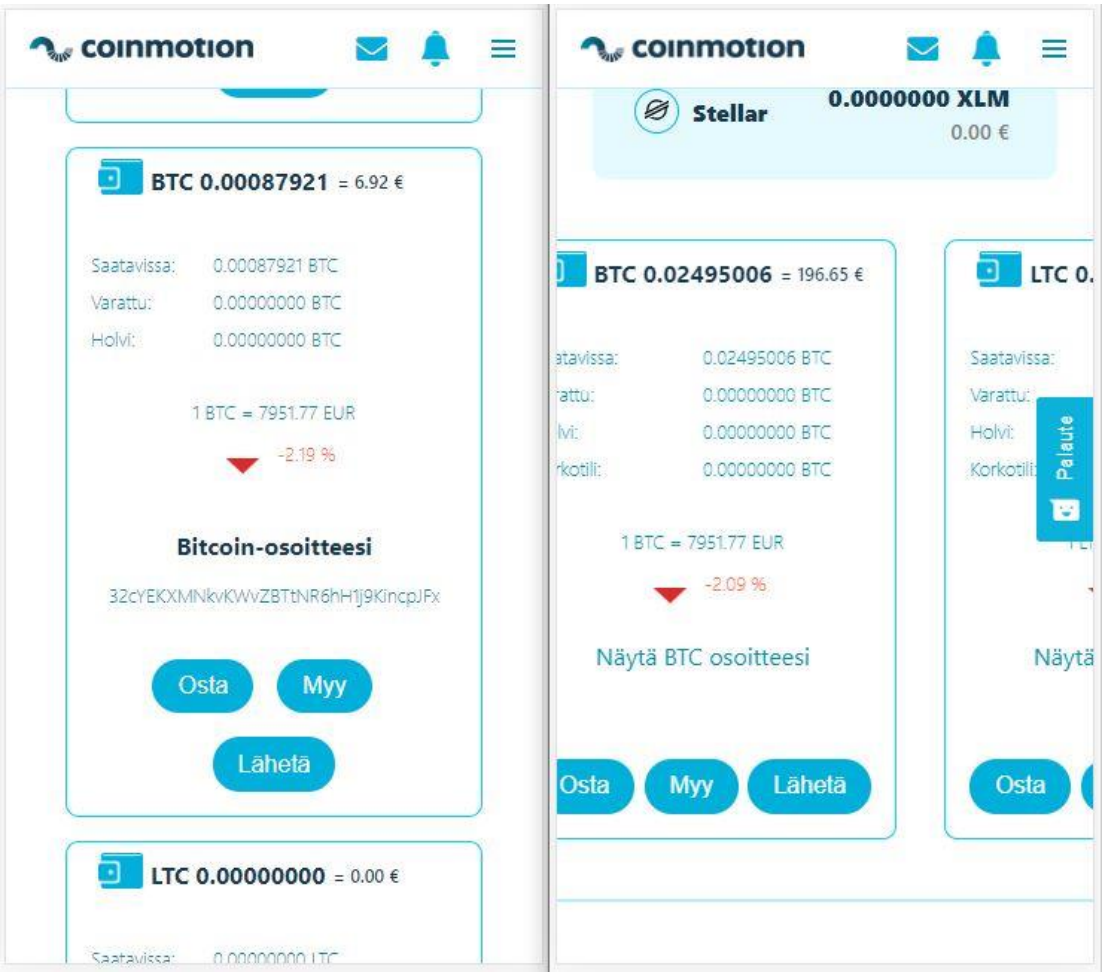
Kuva 21. Parannettu rekisteröintisivu.

Sovelluksen lompakko sivulle toteutettiin myös paljon muutoksia. Tilin yhteenveto osioon tuotiin lisää yksityiskohtaista tietoa tilin saldoista. Näin käyttäjän ei tarvitse skrollata sivua nähdäkseen yleiskuvan omista sijoituksistaan.



Kuva 22. Lompakko sivu ennen ja jälkeen muutosten.

Lisäksi lompakko sivulla olevat lompakot aseteltiin vierekkäin eikä allekkain. Tämän ansiosta käyttäjien ei tarvitse skrollata kaikkien lompakoiden ohi tarkastellakseen lompakkojen alapuolella olevaa sisältöä. Isommalla näytöllä tämä ei ole ongelma koska ruudulla on enemmän tilaa ja lompakot voidaan asetella leveämmiksi ja siten ne vievät korkeussuunnassa vähemmän tilaa.



Kuva 23. Käyttäjän lompakoiden asettelu ennen ja jälkeen muutosten.

Näiden muutosten lisäksi lähes jokaiselle sivulle tehtiin joitakin pieniä muutoksia. Useimmilla muutoksilla vähennettiin tyhjää tilaa elementtien ympäriltä tai tehtiin pieniä korjauksia siten, että ulkoasu sopii mobiililaitteelle.

6 POHDINTAA

Tämän opinnäytetyön aikana toteutettiin tilaajan vaatimusten mukainen web-sovellus asiakkaiden tunnistamista varten sekä suunniteltiin ja toteutettiin Angular-sovelluksen muuttaminen PWA-sovellukseksi. Projektilla oli todella tiukka aikataulu tunnistautumissovelluksen osalta ja siinä onnistuttiin pysymään. Tiukan aikataulun takia projektin suunnittelu jäi kuitenkin vähäiseksi ja suunnitelmaa jouduttiin muuttamaan kesken toteutuksen. Työ saatiin tästä huolimatta toteutettua aikataulussa, mutta paremmalla suunnittelulla koko projekti olisi mennyt sujuvammin.

Jälkeenpäin ajateltuna olisin käyttänyt enemmän aikaa Bittimaatti-sovelluksen suunnitteluun. Paremmalla suunnittelulla kaikki toteutukseen käytetty aika olisi saatu käytettyä vielä tehokkaammin hyväksi. Lisäksi tällöin olisi voinut paremmin tutkia millä tekniikalla sovellus olisi järkevintä toteuttaa. Vähäisen suunnittelun takia pidettiin lähes itsestään selvänä, että sovellus toteutetaan ennestään tutulla Angular-tekniikalla.

Projekti opetti minua Angular-sovellusten ja yleisesti myös web-sovelluksen kehittämisestä todella paljon. Projektin aikana Angularin erilaiset komponentit ja niiden käyttäminen tulivat hyvin tutuiksi. Lisäksi opin paljon PWA-sovelluksista ja web-sovellusten kehittämisestä mobiilialustalle. Uskon että näistä kaikista on paljon hyötyä työelämässä.

Tunnistautumissovellukseen saatiin luotua kaikki tarvittavat toiminnot ja se ajaa tarkoituksensa todella hyvin, joten sovellukselle ei ole tiedossa jatkokehitystarvetta. Sovelluksen ulkoasu on tarkoituksella mahdollisimman yksinkertainen, joten senkin kehittämistarve on vähäinen.

Coinmotion-sovellukseen taas jäi todella paljon kehitettävää. Toteutettu viivakoodinlukija tarvitsee jatkokehitystä ennen kuin sen voi laittaa tuotantoympäristöön. Lisäksi PWA-toimintojen ansiosta sovellukseen olisi mahdollista kehittää useita uusia ominaisuuksia, joista suurimpana voidaan mainita ilmoitusten näyttäminen käyttäjälle. Myös sovelluksen ulkoasun jatkokehityksen mahdollisuudet ovat rajattomat. Tämän projektin aikana vältettiin suuria ulkoasun muutoksia ja tavoitteena oli parantaa olemassa olevaa ulkoasua. Tätä voisi jatkokehittää tekemällä suurempia muutoksia parantamaan mobiilikäytettävyyttä tai jopa rakentaa alusta asti ensisijaisesti mobiilialustalle suunniteltu käyttöliittymä.

7 LÄHDELUETTELO

Abu Experience. 2017. 10 MOBILE UX DESIGN PRINCIPLES YOU SHOULD KNOW. [Online] 13. 6 2017. [Viitattu: 24. 2 2020.] http://uxbert.com/10-mobile-ux-design-principles.

Gagliardi, Valentino. 2020. TypeScript Tutorial For Beginners: Your Friendly Guide. [Online] 7. 2 2020. [Viitattu: 9. 2 2020.] https://www.valentinog.com/blog/typescript/.

Google Inc. 2020. Sass Basics. [Online] 2020. [Viitattu: 12. 2 2020.] https://sass-lang.com/guide.

Google LLC. 2020. Angular documentation. [Online] 2020. [Viitattu: 9. 2 2020.] https://angular.io/guide/architecture.

Google. 2019. Your First Progressive Web App. [Online] 2019. [Viitattu: 13. 2 2020.] https://codelabs.developers.google.com/codelabs/your-first-pwapp/#0.

Microsoft. Visual studio code documentation. [Online] [Viitattu: 14. 2 2020.] https://code.visualstudio.com/docs.

Mozilla Contributors. 2019. Introduction to progressive web apps. [Online] 2019. [Viitattu: 13. 2 2020.] https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps/Introduction.

npm, Inc. and Contributors. About npm. [Online] [Viitattu: 14. 2 2020.] https://docs.npmjs.com/about-npm/.

Scott Chacon and others. 2018. Git about. [Online] 2018. [Viitattu: 14. 2 2020.] https://git-scm.com/about.