

Viivakoodin lukeminen Ionic-ohjelmistokehyksellä



Ammattikorkeakoulututkinnon opinnäytetyö

Visamäki, Tietojenkäsittelyn koulutusohjelma

Kevät 2018

Karoliina Kunnas

Tietojenkäsittelyn koulutusohjelma
Visamäki

Tekijä	Karoliina Kunnas	Vuosi 2018
Työn nimi	Viivakoodin lukeminen Ionic-ohjelmistokehyksellä	
Työn ohjaaja/t	Tommi Saksa	

TIIVISTELMÄ

Opinnäytetyön toimeksiantajana oli Kauppavalmennus Oy, joka on valmentanut kauppvoja hävikin vähentämiseen vuodesta 2013. Yritys tarjoaa asiakkailleen sovelluksen, joka muun muassa kokoaa myynti- ja hävikkiraportit yhteen. Sovelluksen avulla voidaan puuttua hävikin perimmäisiin syihin ja korjata niitä. Sen web-versio on tehty AngularJS-ohjelmistokehyksellä ja hybridisovellus Apache Cordovalla. Ylläpidettävänä on siis kaksi koodipohjaa ja versiota.

Työn tavoitteena oli toteuttaa Kauppavalmennuksen sovelluksesta Ionic Framework -ohjelmistokehyksellä versio, jolla merkitään tyhjiä hyllypaikkoja. Sovellus toimii sekä web- että mobiilisovelluksena, joten useampaa versiota ei tarvita. Tarkoituksena oli tutkia, mitä ominaisuuksia Ionic tarjoaa ohjelmistokehittäjälle ja kuinka sillä tehty sovellus julkaistaan.

Työssä käsiteltiin sovelluksen toteutustekniikoita, kuten Ionic, Angular ja Apache Cordova. Lisäksi selvitettiin, voidaanko Kauppavalmennuksen sovelluksen ominaisuuksia tuoda helposti uuteen versioon. Lopputuloksena onnistuttiin kehittämään sovellus, joka täyttää sille annetut vaatimukset.

Avainsanat Ionic Framework, Angular, Apache Cordova, ohjelmointi, viivakoodit

Sivut 32 sivua

Degree Programme in Business Information Technology
Visamäki

Author	Karoliina Kunnas	Year 2018
Subject	Barcode scanning using Ionic Framework	
Supervisors	Tommi Saksa	

ABSTRACT

The client of this thesis was Kauppavalmennus Oy, which has trained grocery stores to reduce loss since 2013. The company offers an application that compiles sales and loss reports together. The application helps address the root causes of loss and correct them. The web version of the application is developed using AngularJS and the hybrid application was developed using Apache Cordova. Consequently, there are two codebases and versions to maintain.

The aim of this thesis was to develop version of Kauppavalmennus application using Ionic Framework. It is used for marking up empty shelves. Due to using Ionic the application runs both on web and mobile, therefore multiple codebases and versions are not needed. The purpose was to research what features Ionic offers and how the application is published.

The technologies used in the application, for example, Ionic Framework, Angular and Apache Cordova is discussed in this thesis. In the thesis was researched whether it would be easy to bring old features of the Kauppavalmennus application to Ionic version. The project was successful, and the set goals were achieved.

Keywords Ionic Framework, Angular, Apache Cordova, programming, barcodes

Pages 32 pages

SISÄLLYS

1	JOHDANTO.....	1
2	SOVELLUKSEN TOTEUTUSTEKNIIKAT	2
2.1	Viivakoodit.....	2
2.2	Hybridisovellus	3
2.3	Ionic Framework.....	4
2.3.1	Palvelut ja työkalut	4
2.3.2	Käyttöliittymäkomponentit ja sivupohjat	5
2.3.3	Ionic-projektin kansiorakenne.....	6
2.4	Apache Cordova	7
2.5	Angular	8
2.5.1	Arkkitehtuuri.....	10
2.5.2	MVC- ja MVW-arkkitehtuuri.....	11
2.6	Typescript.....	11
2.7	JSON Web Token	12
3	TOIMEKSIANNON TAVOITTEET	14
3.1	Hävikin hallinta.....	14
3.2	Kauppavalmennus-sovellus.....	15
4	SOVELLUKSEN TOTEUTUS.....	17
4.1	Kirjautuminen.....	17
4.2	Myymälän valitseminen	20
4.3	Sivunvalikko ja uloskirjautuminen.....	21
4.4	Viivakoodin lukeminen.....	22
5	SOVELLUKSEN JULKAISU WEB- JA MOBIILISOVELLUKSENA	28
6	YHTEENVETO	29
	LÄHTEET	30

1 JOHDANTO

Työn toimeksiantaja on Kauppavalmennus Oy, joka on valmentanut kauppvoja hävikin vähentämiseen vuodesta 2013. Yritys tarjoaa mobiilisovelluksen, joka on kauppojen jokapäiväinen työkalu. Sovelluksen avulla on mahdollista puuttua hävikin lisäksi myös tyhjien hyllypaikkojen perimmäisiin syihin ja korjata niitä.

Tämän opinnäytetyön tavoitteena on helpottaa ja nopeuttaa päivittäistavarakauppojen tuotetietojen käsittelyä sekä hävikin hallintaa ja vähentämistä. Työssä kehitetään mobiilisovellus, jonka avulla älypuhelimella voidaan käyttää tuotteiden viivakoodien lukemiseen. Työssä keskitytään siihen, että tyhjien hyllypaikkojen tuoteviivakoodit voidaan lukea ja saadut tiedot tallennetaan tietokantaan. Ionic-ohjelmistokehityksen avulla tehdään hybridimobiilisovellus, joka toimii sekä web- että mobiiliympäristössä. Lisäksi työssä selvitetään, olisiko Kauppavalmennus-sovelluksesta hankalaa tuoda ominaisuuksia Ionic-versioon. Valitsin aiheen, koska projektissa käytettävät tekniikat olivat minulle ennestään täysin tuntemattomia. Tämä oli erinomainen mahdollisuus oppia uutta.

Hävikille on aina olemassa syynsä, ja osa niistä voidaan nykytiedoilla tunnistaa ja osa myös käsitellä. Punalaputuksella kauppa pääsee helpommin eroon tuotteista, jotka uhkaavat päätyä hävikkiin. Hävikki ja punalaputus ovat yleisimpiä syitä tyhjille hyllypaikoille. Kauppojen ongelmana on myynnin vaihtelu ja sen seuraamisen vaikeus. Työntekijöillä ei ole nopeaa ja helppoa tapaa nähdä tietyn tuotteen varasto-, myynti- ja hävikkitietoja, jotta esimerkiksi tilausmäärät saataisiin optimoituja. Lisäksi tiedon kulku on hidasta, joten ongelmiin reagoidaan viikkojen tai jopa kuukausien viiveellä. (Ketola, Rikberg, Österlund 2017, 12.)

Opinnäytetyössä vastataan seuraaviin tutkimuskysymyksiin.

- Miten toteutetaan yksinkertainen viivakoodinlukija mobiilisovelluksena?
- Mitä ominaisuuksia Ionic-ohjelmistokehitys tarjoaa sovelluskehittäjille?
- Miten Ionic-ohjelma voidaan julkaista sekä web- että mobiilisovelluksena?

2 SOVELLUKSEN TOTEUTUSTEKNIIKAT

2.1 Viivakoodit

Viivakoodi on koneellisesti luettavissa olevaa informaatiota. Sitä voidaan lukea viivakoodinlukijalla tai ohjelmallisesti eli tulkitsemalla viivakoodista otettua kuvaa. Norman Woodland ja Bernard Silver keksivät viivakoodin vuonna 1949. Silver kuuli kauppiaan haaveilevan laitteesta, jolla voisi lukea tuotteen tietoja. Woodland ja Silver päättivät kokeilla morseaakkosten viivoja ja pisteitä tuotetietojen tallentamiseen. Pisteet olivat liian pieniä luettaviksi, joten he päättivät venyttää pisteet ohuiksi ja viivat paksuiksi viivoiksi. He valmistsivat viivakoodinlukijan 500 watin sähkölampusta. 1970-luvulla viivakoodit yleistyivät tuotteissa, kun viivakoodinlukija tuli markkinoille. (GS1 n.d.)

Aiemmin viivakoodit kuvasivat tietoa rinnakkaisten viivojen ja niiden välien suhteita muuttaen. Nykyään myös pisteiden, ympyröiden ja yleisesti ottaen sääntöpohjaisten kuvioden kautta tehty erimalliset symbolit luetaan viivakoodiksi. GS1-järjestelmä tarjoaa erilaisia viivakoodia jäsensensä käyttöön. GS1 on kehittänyt ja standardisoinut omia tarkemmin määriteltyjä viivakoodirakenteita, jotka soveltuvat etenkin toimitusketjun hallintaan. Myös niiden tietosisältö on standardisoitu. Kaupan tuotteiden EAN-viivakoodit on viivakoodien tunnetuin käyttökohde. Viivakoodien uudet sukupolvet lisäävät niiden käyttömahdollisuuksia, ja ne ovatkin yleistyneet myös muilla sektoreilla. (GS1 n.d.)

EAN/UPC-viivakoodit (Kuva 1) ovat suunniteltu pääasiassa vähittäiskaupan kassapisteiden suurivolyymiseen käyttöön. Niissä oleva tietomäärä on rajoitettu GS1-avainten ja yritysten sisäisten kauppatuotteiden tunnistamiseen. Ensimmäinen UPC-viivakoodi tuotettiin vuonna 1976, jolloin myös luettiin EAN-viivakoodi kassapisteellä ensimmäistä kertaa. (GS1 n.d.)



Kuva 1. EAN-13-viivakoodi (GS1 n.d.).

GS1-128-viivakoodia (Kuva 2) käytetään logistiikan sovelluksissa. Ne voivat sisältää minkä tahansa GS1-tunnisteavaimen tai -attribuutin. Niitä ei voida lukea monisädelukijoilla, joita käytetään muun muassa päivittäistavarakauppojen kassoilla. GS1-128-viivakoodissa voidaan käyttää GS1-sovellustunnuksia. Ne ovat 2-4 numeroisia tunnuksia, jotka kertovat muun muassa niihin liittyvän tiedon merkityksen. (GS1 n.d.)



Kuva 2. GS1-128-viivakoodi (GS1 n.d.).

GS1 DataMatrix -viivakoodi (Kuva 3) on ainut GS1:n 2D-viivakoodi. Tällä hetkellä se on käytössä muun muassa terveydenhoitosektorilla. DataMatrix voidaan lukea ainoastaan kameraskannerilla. Siinä voidaan käyttää GS1-sovellustunnuksia kuten GS1-128-viivakoodissa. DataMatrix mahdollistaa suuren informaatiomäärän tallentamisen pieneen tilaan. (GS1 n.d.)



Kuva 3. GS1 DataMatrix -viivakoodi (GS1 n.d.).

2.2 Hybridisovellus

Hybridisovelluksissa voidaan käyttää web-tekniikoita, kuten HTML, CSS sekä JavaScript, sekä natiiveja elementtejä. Se mahdollistaa laitteen kameran käytön viivakoodien lukemiseen. Hybridisovellukset ovat suosittuja, jos halutaan kehittää sovellus usealle eri alustalle tai hyödyntää ominaisuuksia, joita web-sovelluksessa ei ole. (Anand & Wasmer n.d, 15.) Hybridisovellukset asennetaan laitteeseen natiivisovelluksien tapaan (Telerik Developer Network 2015). Niitä ajetaan laitteen selainmoottorilla, johon luodaan selainnäkö (Ionic Framework n.d.). Se on kuin web-sivu ilman osoitepalkkia ja selaimen kontrolleja (Anand & Wasmer n.d, 15).

Hybridisovellukset eivät sovellu joka tilanteeseen. Natiivisovellus saattaa olla parempi vaihtoehto, jos sovelluksessa käytetään paljon natiiveja ominaisuuksia tai siltä vaaditaan erityisen paljon suorituskykyä. (Rownski 2012.) Esimerkiksi Facebookilla oli alun perin hybridisovellus. Yritys päätti kehittää sen tilalle natiivin iOS-mobiilisovelluksen vuonna 2012. Hybridisovelluksen ansiosta Facebook pystyi päivittämään sovelluksen useita kertoja päivässä. Osa ominaisuuksista kuten uutisvirta ja ystäväläistä tuotiin verkkosivustolta, ja natiiveista ominaisuuksista hyödynnettiin muun muassa laitteen kameraa. (Rownski 2012.) Facebookin uutisvirta on loputon lista erilaista sisältöä, kuten kuvia, linkkejä ja tekstiä. Dynaamisesti päivittyvällä listalla muistin ja vierityksen suorituskyvyn hallinta voi olla hankalaa

selainnäkyessä. (Anand & Wasmer n.d, 17.) Facebookin kehittämä natiivi iOS-sovellus on nopeampi ja tehokkaampi. Vieritys on sulavampaa ja uutisvirran sisältö latautuu välittömästi sovelluksen avaamisen jälkeen. (Rowniski 2012.)

2.3 Ionic Framework

Ionic Framework on avoimen lähdekoodin ohjelmistokehys (framework). Se on tarkoitettu hybridimobiilisovellusten kehittämiseen mobiilioptimoiduilla HTML-, CSS- ja JavaScript-komponenteilla. Ionic perustuu Angular-ohjelmistokehykseen, joten sillä kehitetyssä sovelluksessa voidaan käyttää mitä tahansa mobiilisovellukseen sopivia Angular-komponentteja. (Ravulavaru 2017, 46.) Ionic-sovelluksia ajetaan natiivisovellusten tapaan eikä laitteen selaimessa. Siihen tarvitaan natiivi paketoija, kuten Apache Cordova. (Ionic Framework n.d.a.)

Ionic on Adam Bradley'n, Max Lynchin ja Ben Sperry'n kehittämä. Alfaversio julkaistiin vuoden 2013 marraskuussa ja beetaversio maaliskuussa 2014. Lopullinen versio 1.0 julkaistiin toukokuussa 2015. (Ionic Framework 2018a.) Tammikuussa 2017 julkaistiin versio 2.0 ja huhtikuussa julkaistiin Ionic 3. (Ionic-Team 2017.)

2.3.1 Palvelut ja työkalut

Ionic Command Line Interface (CLI) on työkalu, joka tarjoaa monia hyödyllisiä komentoja. Sitä käytetään muun muassa Ionicin asentamiseen ja päivittämiseen. (Ionic Framework 2018a.) Pääasiallisesti Ionic-projektit luodaan Ionic CLI -työkalulla, joka perustuu Node.js-sovellusalueeseen. Sen käyttö vaatii Noden LTS-version (Long Term Support) asentamisen. (Ionic Framework 2018b.)

Ionic Pro on joukko palveluita ja ominaisuuksia, jotka helpottavat Ionic-sovelluksen kehittämistä. Se pohjautuu Git-versionhallintaan. Ionic Pro tallentaa koodin muutokset ja lisää ne pilvipalveluun. Sieltä sovellusta voidaan jakaa ja hallita yksinkertaisella Dashboard-käyttöliittymällä ja CLI-työkalulla. (Ionic Framework n.d.d.) Live Deploy -palvelun avulla voi päivittää sovelluksen reaaliaikaisesti eikä sovelluskaupan välityksellä. Ionic View on mobiilisovellus, jonka avulla sovellus voidaan jakaa, esimerkiksi testaa-jille, ilman sovelluksen laittamista sovelluskauppaan. Se on saatavilla iOS- ja Android-käyttöjärjestelmille. (Ionic Framework n.d.d.)

Ionic DevApp on ilmainen sovellus, jolla sovellusta voidaan testata helposti laitteessa. Se on saatavilla iOS- ja Android-käyttöjärjestelmille. DevApp-sovelluksen käyttöä varten tarvitaan uusimman version CLI-työkalusta, ja laite täytyy yhdistää samaan paikalliseen verkkoon. (Ionic Framework n.d.d.) DevApp-sovelluksessa ei ole natiiveja riippuvuuksia, joten sovelluksen testaamiseen laitteella ei tarvita esimerkiksi Android Studiota. Lisäksi siihen on

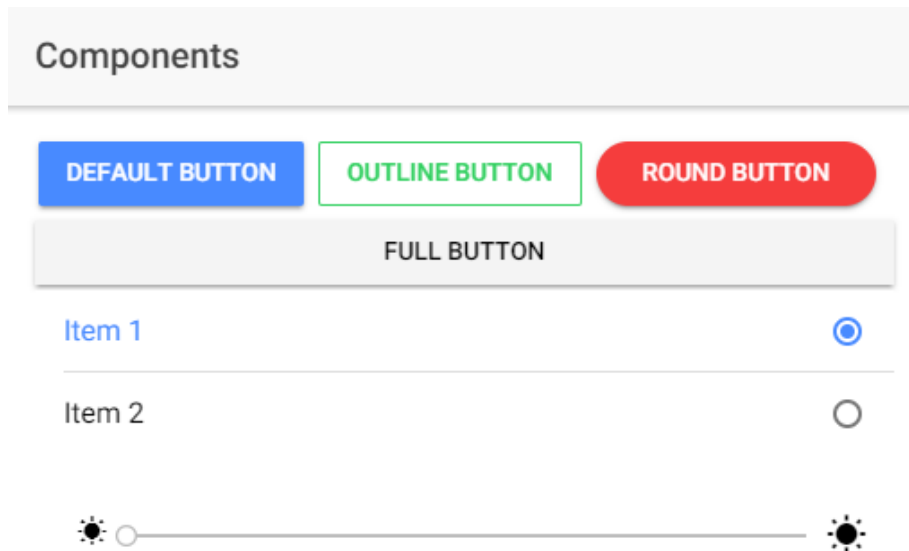
asennettu yleisimpiä natiiveja lisäosia, joten myös täysin natiiveja toiminnallisuuksia voidaan testata. LiveReload-toiminto päivittää muutokset heti niiden tekohetkellä. (Ionic Framework Blog 2017a.)

Ionic Creator on yksinkertainen ”Drag and Drop”-työkalu, jossa sovellus kehitetään visuaalisesti raahaamalla komponentit emulaattoriin. Sillä sovellus voidaan jakaa muille. Creatorissa on integroitu koodieditori, joten kustomoidun koodin lisäämiseen ei tarvita erillistä paikallista editoria. (Ionic Framework n.d.e.) Sillä voi tehdä oman toimivan sivupohjan, jonka voi ladata ja jatkaa ohjelmointia paikallisesti. (Ionic Framework Blog 2017b.) Ionic kehittäjät ovat kehittämässä Creatorista paikallisesti ladattavaa työkalua. Tavoitteena on, että ”Drag and Drop”-tilasta olisi mahdollista siirtyä saumattomasti lähdekoodin muokkaamiseen. Paikallisella työkalulla pysyisi käyttämään muun muassa emulaattoreita, Git-versionhallintaa ja CLI-työkalua. (Ionic Framework Blog 2017b.)

2.3.2 Käyttöliittymäkomponentit ja sivupohjat

Ionic ei korvaa JavaScript-ohjelmistokehystä, vaan yksinkertaistaa frontend-kehitystä eli selainpuolen web-kehitystä. Se tarjoaa erilaisia valmiita mobiilikäyttöliittymäkomponentteja ja sivupohjia. Ne ovat samanlaisia kuin natiivien iOS tai Android SDK:lla (Software development kit) kehitetyissä sovelluksissa. (Ionic Framework n.d.a, 2018b.) Ionic CLI hakee valitun sivupohjan Git-tietovarastosta (repository), ja yksilöi sen jokaiselle uudelle sovellukselle. Sivupohjissa on valmiiksi esimerkiksi sivuvalikko tai välilehdet. (Ionic Framework n.d.b.)

Käyttöliittymäkomponenttien avulla sovelluksen käyttöliittymä voidaan rakentaa nopeasti (Ionic Framework n.d.c). Niitä ovat esimerkiksi painikkeet, valintapainikkeet ja liukusäätimet (Kuva 4). Komponentit mukautuvat automaattisesti käyttöjärjestelmän mukaan. (Ravulavaru 2017, 72.)

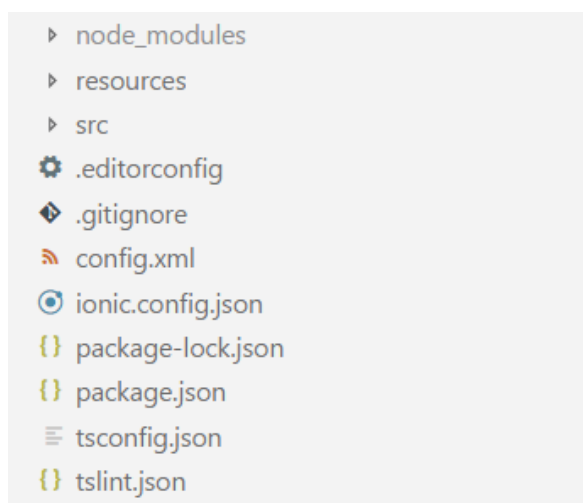


Kuva 4. Ionicin valmiita komponentteja.

Ionicissa on oma grid-järjestelmä, jonka avulla komponentit asettuvat tasaisesti vierekkäin. Se on ottanut vahvasti vaikutteita Bootstarpin grid-järjestelmästä. Grid muodostuu kolmesta osasta: taulukko (grid), rivi (row) ja sarake (column). Taulukko on ikään kuin säiliö riveille ja sarakkeille. Rivit asettavat sarakkeet tasaisesti riviin. Grid-järjestelmä perustuu 12 sarakkeen pohjaan, jossa on erilaiset raja-arvot eri kokoisille näytöille. (Ionic Framework 2017)

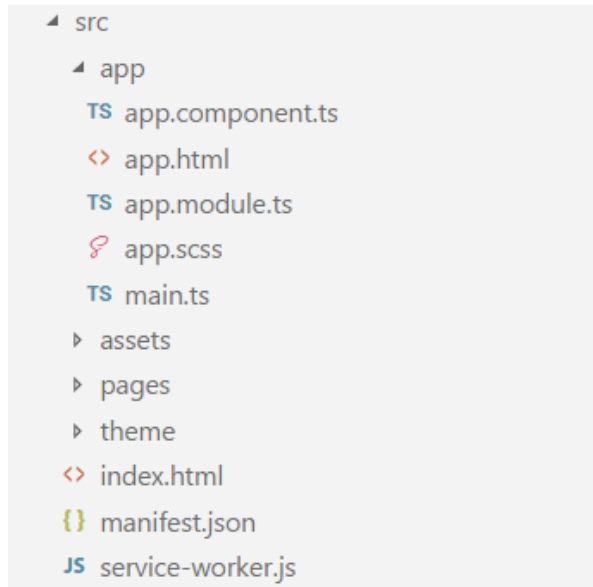
2.3.3 Ionic-projektin kansiorakenne

Kuvassa 5 on Ionic-projektin kansiorakenne. Sovelluksen lähdekoodi on src-kansiossa. Config.xml-tiedostossa on metatietoja, joita Cordova tarvitsee sovelluksen kääntämiseen eri käyttöjärjestelmille. Lisäksi projektiin kuuluu JSON-tiedostoja, jotka sisältävät muun muassa node-riippuvuuksiin ja TypeScriptin konfiguraatioon liittyvää tietoa. (Ravulavaru 2017, 58–59.)



Kuva 5. Ionic-projektin rakenne.

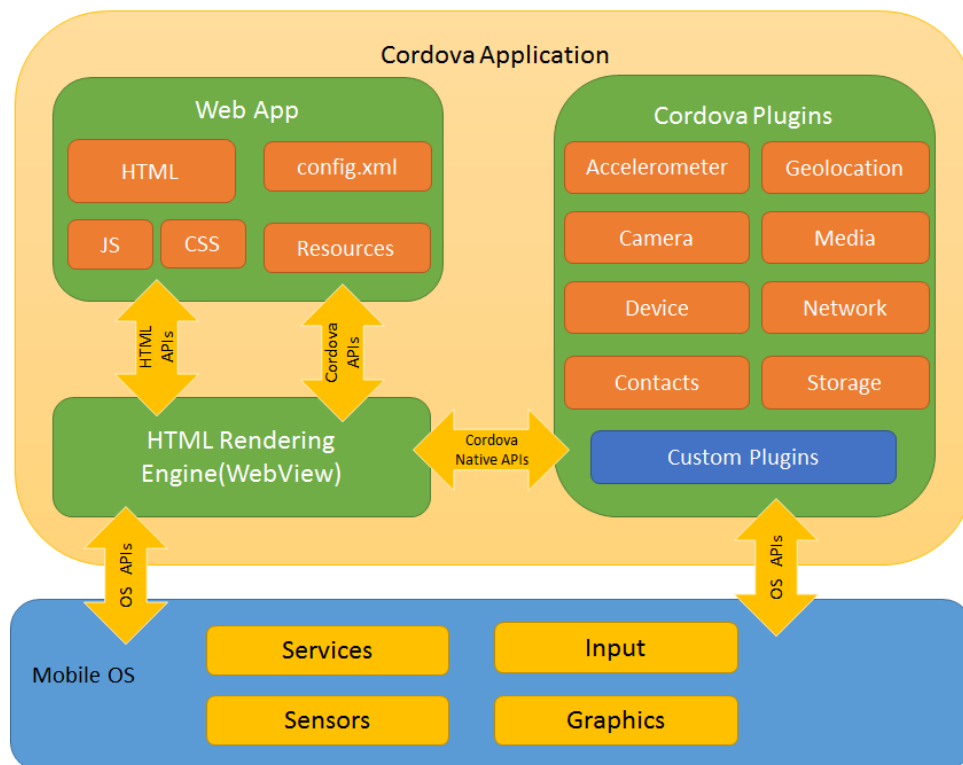
Src-kansio sisältää Ionic-sovelluksen HTML-, CSS- ja JavaScript-koodit (Kuva 6). App-kansiossa on sovelluksen alustustiedostot. Esimerkiksi @NgModule-moduuli määritetään app.module.ts-tiedostossa, ja sovelluksen juuriluokka on app.component.ts-tiedostossa. Pages-kansioon tallennetaan sovellukseen luodut sivut. Niillä on omat HTML-, TS- (TypeScript) ja tarvittaessa tyylien muokkaamiseen käytettävät SCSS-tiedostot. TS-tiedostossa määritetään sovelluksen sivun toimintalogiikka. (Ravulavaru 2017, 60–61.)



Kuva 6. Src-kansion sisältö.

2.4 Apache Cordova

Apache Cordova on avoimen lähdekoodin mobiilisovellusalausta, jota käytetään järjestelmäriippumattomien hybridisovellusten kehittämiseen. Se mahdollistaa mobiilisovellusten kehittämisen HTML-, CSS- ja JavaScript-tekniikoiden avulla. Sovellus kehitetään web-sivuna, ja Apache Cordova kääntää sen kullekin alustalle sopivaksi (Kuva 7). Se hyödyntää ohjelmointirajapintoja käyttäkseen laitteen ominaisuuksia, kuten kameraa, dataa ja verkon tilaa. (Apache Cordova 2016.)



Kuva 7. Näkymä Apache Cordovan sovellusarkkitehtuurista (Apache Cordova 2015).

Cordovassa käytetään lisäosia natiivin alustan kanssa kommunikointiin. Niiden avulla sovelluksessa voidaan käyttää laitteen toiminnallisuuksiin kuten kamera, GPS ja yhteystiedot. Cordovan omien lisäosien lisäksi on mahdollista käyttää myös kolmannen osapuolen lisäosia. Niiden avulla saadaan käyttöön ominaisuuksia, jotka eivät välttämättä ole saatavilla kaikilla alustoilla. Myös omien lisäosien kehittäminen on mahdollista. (Apache Cordova 2016.)

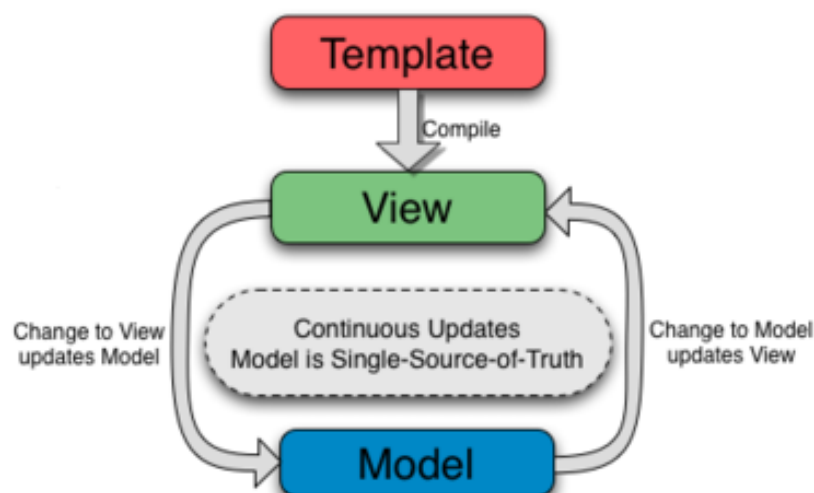
Cordova tarjoaa kaksi tapaa mobiilisovelluksen kehittämiseen. Laiteriippumattonta tapaa käytetään, kun halutaan ajaa sovellusta niin monella eri käyttöjärjestelmällä kuin mahdollista. Siinä käytetään Cordova CLI:tä (Command Line Interface) eli komentokehotetta. CLI kopioi web-ominaisuudet kullekin alustalle omiin alihakemistoihinsa ja tekee myös tarvittavat konfiguroinnit. Sillä voi myös lisätä lisäosia sovellukseen. Toinen tapa on alustakeskeinen eli: sovellus kehitetään vain yhdelle alustalle. Tätä tapaa käytetään erityisesti silloin, kun sovellusta halutaan muokata alemmillakin tasoilla. Alusta-keskeisessä toimintatavassa lisäosien asentamiseen tarvitaan erillinen Plugman-työkalu. (Apache Cordova 2016.)

2.5 Angular

Angular on Googlen ylläpitämä avoimen lähdekoodin JavaScript-ohjelmistokehys. Sen ovat luoneet Misko Hevery ja Adam Abrons vuonna 2009.

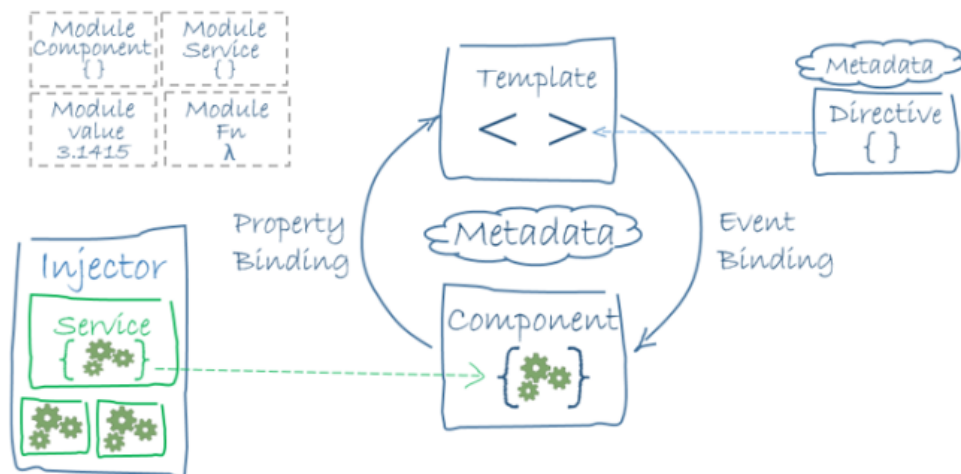
Vuonna 2010 Misko Hevery työskenteli Googlen Feedback-projektin parissa. Hän sai vähennettyä projektin koodirivien määrää huomattavasti Angularin avulla. Sitä käytetään nykyään useissa Googlen projekteissa. (Branas 2014, 8.) Angularin ensimmäisistä versioista käytettiin nimitystä AngularJS. Versiota 2 ja siitä uudempia kutsutaan Angulariksi. (Angular n.d.e.)

Angular-ohjelmistokehys laajentaa HTML-syntaksia direktiiveillä, joita käytetään DOM-elementtien manipulointiin. Angularin riippuvuusinjektio (dependency injection) ja datasidonta (data binding) vähentävät huomattavasti sovelluksen koodin määrää. (AngularJS 2018a.) Riippuvuusinjektio on mekaniikka, joka käsittelee komponenttien välisiä riippuvuuksia. Sen avulla sovelluksen osia luodaan ja välitetään niitä muihin sovelluksen osiin. (Angular n.d.a.) Riippuvuusinjektioilla luotu komponentti yhdistetään palveluihin, joita se tarvitsee. Ne on lisätty komponentin rakentajan parametreiksi. (Angular n.d.b.) Datasidonta tarkoittaa sitä, että data synkronoidaan automaattisesti mallin ja näkymän välillä (Kuva 8). Malliin tehdyt muutokset näkyvät välittömästi näkymässä ja toisinpäin. (AngularJS 2018b.)



Kuva 8. Kaksisuuntainen datasisidonta (AngularJS 2018b).

2.5.1 Arkkitehtuuri



Kuva 9. Angularin kokonaisarkkitehtuuri (Angular n.d.b).

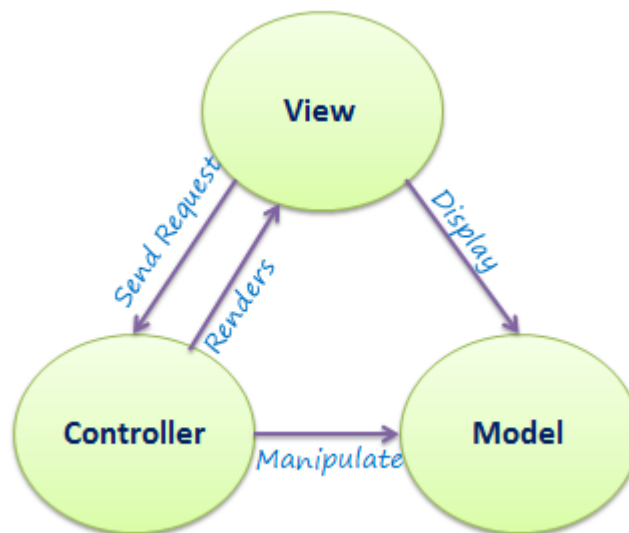
Angular-sovellus koostuu muun muassa HTML-kielellä kirjoitetusta dokumenttipohjasta, komponenteista ja palveluista (Kuva 9). Ne ovat modulaarisia eli niihin voi lisätä ja poistaa itsenäisiä osia ja kokonaisuuksia. Angularilla on oma moduulijärjestelmä `NgModules`. Moduuliluokissa käytetään `@NgModule`-decoratoria. Jokaisessa Angular-sovelluksessa on vähintään yksi `NgModule`-luokka eli juurimoduuli, jonka nimi on yleensä `AppModule`. (AngularJS 2018c.) Se konfiguroi injektorin ja kääntäjän. Lisäksi juurimoduuli yksilöi muun muassa moduulin omat komponentit ja direktiivit. (Angular n.d.d.) JavaScriptissä on oma moduulijärjestelmänsä, ja se on täysin erilainen kuin Angularin. (Angular n.d.b.) Angular-sovelluksissa ei ole päämetodia kuten useimmissa sovelluksissa, joten moduulit määrittävät, miten sovellus otetaan käyttöön. (AngularJS 2018c.)

Komponentteja eli luokkia käytetään näkymien hallintaan. Niissä määritellään näkymän toimintalogiikka. (Angular n.d.b.) `@Component`-decoratorilla luokka määritellään Angular-komponentiksi, ja sillä voidaan lisätä metatietoja. Niillä määritellään, miten komponentti käsitellään, ilmenneen ja käytetään ajonaikana. (Angular n.d.c.)

Angularissa on rakennetta ja ominaisuutta muokkaavia direktiivejä, joista lisätään merkinnät DOM-elementteihin. Rakennedirektiivit lisäävät, poistavat ja korvaavat DOM-elementtejä. Esimerkiksi `NgIf`-rakennedirektiivi lisää elementin, jos sille asetettu ehto toteutuu. Ominaisuutta muokkaavat direktiivit eivät lisää uusia elementtejä vaan muokkaavat jo olemassa olevia. Esimerkiksi `NgModel` on ominaisuutta muokkaava direktiivi, joka implementoi kaksisuuntaista datasisäntä. (Angular n.d.b.)

2.5.2 MVC- ja MVW-arkkitehtuuri

MVC-arkkitehtuuri jakaa sovelluksen kolmeen komponenttiin: malli (Model), näkymä (View) ja ohjain (Controller). Jokaisella komponentilla on oma tehtävänsä (Kuva 10). Malli huolehtii tiedon käsittelystä, kuten tietojen hakeminen ja tallentaminen tietokantaan. Näkymä huolehtii tietojen esittämisestä eli käyttöliittymästä. Ohjain käsittelee käyttäjän pyyntöjä ja muuntaa tiedon mallille tai näkymälle. (Tutorialspoint n.d.).



Kuva 10. MVC-arkkitehtuuri (Tutorials Teacher n.d.).

Aiemmin AngularJS-sovellusten arkkitehtuuri muistutti enemmän MVC-mallia, mutta muutoksien myötä se oli lähempänä MVVM-mallia (Model-View-ViewModel). Myöhemmin AngularJS julisti noudattavansa MVW-arkkitehtuuria (Model-View-Whatever), joka vastaa MVC-arkkitehtuuria. Mutta se ei rajoitu malli-, näkymä- ja ohjainkomponentteihin vaan ohjaimen sijasta valitaan kulloinkin sopiva vaihtoehto. Tyypillisesti Angular-sovellus koostuu näkymästä, mallista ja kontrollista, mutta siinä on myös monia muita tärkeitä komponentteja, kuten palvelut ja direktiivit. (Google+ 2012; Branas 2014.)

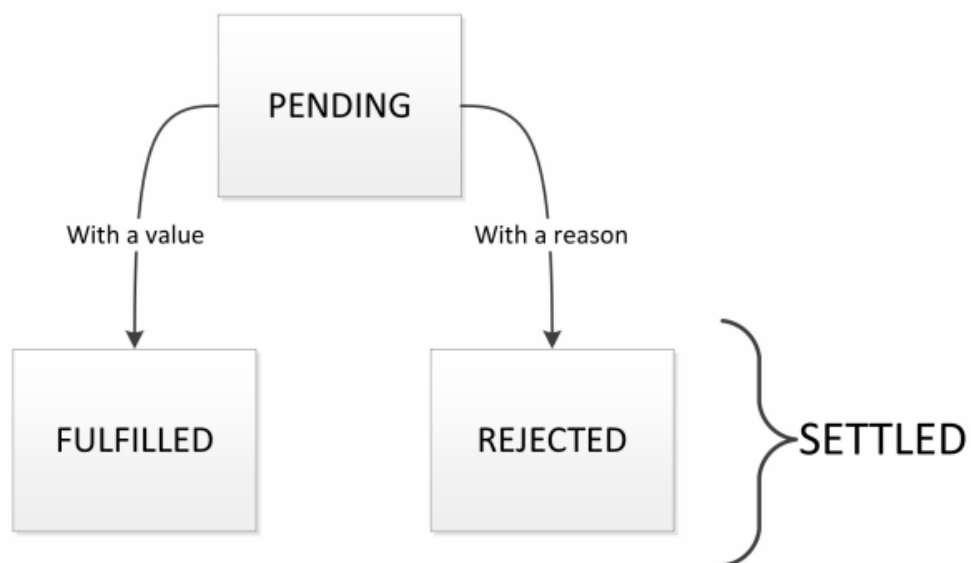
2.6 Typescript

TypeScript on Microsoftin kehittämä JavaScriptiin perustuva ohjelmointikieli. Se mahdollistaa luokkapohjaisen olio-ohjelmoinnin ja vahvan staattisen tyyppityksen JavaScriptissä. TypeScript julkaistiin vuonna 2012. Sen tarkoitus on helpottaa suurien ohjelmien kehittämistä. (Cleverism n.d.)

JavaScriptiä voidaan käyttää TypeScriptin kanssa, koska TypeScript käännetään JavaScriptiksi, jotta sitä voidaan ajaa esimerkiksi selaimessa. TypeScript käyttää ECMAScript 6 -standardin (ES6) ominaisuuksia kuten moduuleja, luokkia, vakioita, rajapintoja ja nuolifunktioita. (Telerik Developer Network 2017.)

TypeScript-kielessä ei tarvitse erikseen määritellä muuttujien tyyppejä. Se olettaa tyyppin olevan se, joksi muuttuja ensimmäisen kerran asetettiin (Ali Syed 2017, 10.) JavaScript on heikosti tyyppitetty ohjelmointikieli, joten muuttujien tyyppeihin liittyvät virheet saatetaan löytää vasta kun sovellus on jo otettu käyttöön. TypeScript ratkaisee ongelman antamalla virheilmoituksen jo koodia kirjoittaessa. (Telerik Developer Network 2017.)

TypeScript-kielessä on Promise-luokka, jonka avulla virheitä voidaan käsitellä (Ali Syed 2017, 68). Promise vastaa JavaScriptin Callback-metodia, mutta sen syntaksi on yksinkertaisempi ja virheiden käsittely on helpompaa. Promise-luokasta luodaan olio new-operaattorilla ja luokan rakentajalle välitetään sisäfunktio. Sen parametreiksi asetetaan resolve- ja reject-metodit. (Codecraft n.d.) Niitä käytetään Promise-olion tilan määrittämiseen. Se voi olla odottava, täytetty tai hylätty (Kuva 11). (Ali Syed 2017, 72.) Sisäfunktiossa suoritetaan haluttu prosessi, jonka jälkeen kutsutaan resolve-metodia. Reject-metodia kutsutaan virhetilanteissa. (Codecraft n.d.) Promise-oliolla on then-metodi, jolla se voidaan ketjuttaa. Siihen voidaan liittää catch-metodi virheiden käsittelyä varten. Jos then-metodi ei toteudu, kutsutaan catch-metodia. (Ali Syed 2017, 73–74.)



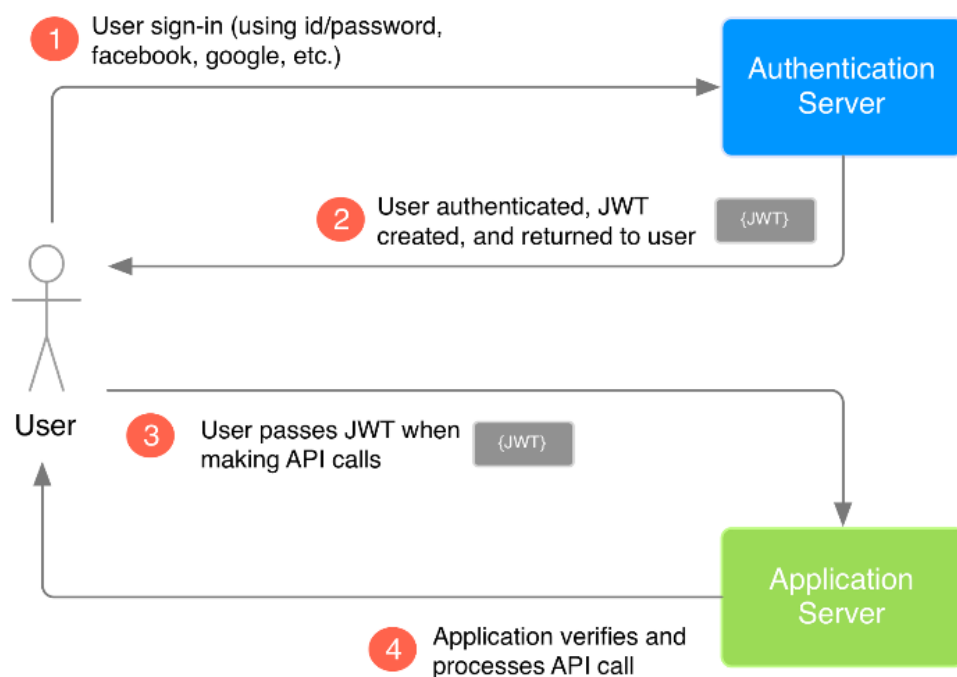
Kuva 11. Promise-olio on yhdessä kolmesta tilasta (Ali Syed 2017, 72).

2.7 JSON Web Token

JSON Web Token (JWT) eli käyttöoikeustietue on avoimen standardin menetelmä tiedon välittämiseen turvallisesti JSON-objektina. JWT-menetelmää käytetään yleisimmin käyttäjän autentikointiin, mutta myös muuhun tiedon siirtoon. JWT-käyttöoikeustietue koostuu otsikosta (header), sisällöstä (payload) ja allekirjoituksesta (signature). Otsikko sisältää käyttöoikeustietueen tyyppiin ja salaukseen liittyviä tietoja. Sisältöön lisätään väitteet (claims), jotka voivat olla rekisteröityjä, julkisia tai privaateja.

teja. Käyttäjän autentikoinnissa väitteisiin listataan käyttäjän tiedot. Allekirjoitus osa muodostuu salatusta osasta sekä Base64-muotoon koodatuista otsikosta ja väitteistä. Sitä käytetään käyttöoikeustietueen lähettäjän todentamiseen. Kokonaisuudessaan JWT muodostuu kolmesta merkkijonosta, jotka on yhdistetty toisiinsa pisteellä. (JWT n.d.)

Kun käyttäjä kirjautuu onnistuneesti tunnuksillaan, palautetaan JWT. Se tallennetaan laitteen muistiin, kun taas perinteiseen tapaan evästeet tallennettaisiin palvelimelle. Aina kun käyttäjä haluaa käyttää suojattua resurssia, täytyy pyynnön mukana lähettää myös JWT. Se lisätään tyypillisesti Authorization-otsikkoon, jossa käytetään Bearer-mallia. Silloin käyttäjätietoja ei koskaan tallenneta palvelimelle. Authorization-otsikosta varmistetaan, että käyttäjällä on pääsy suojattuun resurssiin (Kuva 12).



Kuva 12. JWT-menetelmän käyttö käyttäjän autentikointiin (Stecky-Efantis 2016).

3 TOIMEKSIANNON TAVOITTEET

Toimeksiantona oli toteuttaa Ionic-mobiilisovellus, jotta älypuhelinta voidaan käyttää tuotteiden viivakoodien lukemiseen. Siitä tehtiin Ionic-ohjelmistokehyksellä hybridisovellus, joka toimii sekä web- että mobiilisovelluksena. Sellaiset ominaisuudet, jotka toimivat vain älypuhelimella, piilotettiin websovelluksesta. Sovellukseen lisättiin käyttäjän autentikointi, viivakoodin luku -ominaisuus ja tyhjän hyllypaikan tallentaminen tietokantaan.

Kauppavalmennus-sovelluksen nykyinen web-versio on toteutettu AngularJS-ohjelmistokehyksellä. Siitä toteutettiin hybridisovellus Apache Cordovalla, jotta viivakoodeja voitiin lukea. Ongelmana kuitenkin oli, että sovelluksessa tarvittiin kaksi koodipohjaa ja versiota. Tässä työssä päädyttiin käyttämään Ionicia, koska se ratkaisee kyseisen ongelman. Kun ei tarvitse ylläpitää eri versioita, säästetään paljon aikaa ja rahaa.

3.1 Hävikin hallinta

Hävikki ei ole normaalia, sille on aina olemassa syy ja osa syistä voidaan nykytiedoilla tunnistaa. Valtaosa hävikistä johtuu virheistä, jotka jatkuvat, kunnes ne huomataan ja korjataan. Ainoastaan pieni osa johtuu satunnaisista syistä, esimerkiksi ennustamattomista myynninvaihteluista tai tunnistamattomista syistä. Kun hävikkiä syntyy, sen syy täytyy tutkia mahdollisimman pikaisesti ja selvittää mitä voidaan tehdä sen korjaamiseksi, sillä kaikki viive tarkoittaa lisää hävikkiä. Lisäksi mitä nopeammin hävikki saadaan tilastoitua, sitä paremmin muistetaan tilanne, jossa hävikki on syntynyt. Kun hävikin perimmäinen syy löydetään, siihen voidaan puuttua. (Ketola, Rikberg, Österlund 2017, 12–14.)

Yleisimpiä perimmäisiä syitä hävikin syntymiselle on myynninvaihtelu tai se, että automaatti tai ihminen tekee virheen. Esimerkiksi automaatti tilaa tuotetta liikaa, vaikka tuotteen saldo ja tilausasetukset olisivat oikein. Automaatin tilausarvo on voitu asettaa tilaamaan liian aikaisin tai liian paljon tavaraa. Jos ihmisellä ei ole riittävän hyviä työkaluja tai koulutusta tilausten tekemiseen, hän saattaa tilata tuotetta liikaa. Lisäksi tuotteen myyntierä eli pienin mahdollinen tilattava määrä saattaa olla liian suuri. Tuotteen myynti voi olla niin vähäistä, että sen pitäminen valikoimassa ei ole kannattavaa. Voi myös olla, että tuotetta on ensin myyty sen verran, ettei hävikkiä ole syntynyt ja sen jälkeen myynti on pudonnut. Kaikki nämä syyt johtavat siihen, ettei kaikkea ehditä tai saada myytyä, minkä seurauksena syntyy hävikkiä. Kaikki hävikki ei kuitenkaan ole huonoa, vaan riski kannattaa ottaa esimerkiksi uutuustuotteiden lanseerauksessa tai valikoiman laajentamisessa. (Ketola ym. 2017, 19–21.)

Kaupan ongelma on myynninvaihtelu, jonka vaikutuksia lieventämään on keksitty punalaputus. Sen avulla kauppa pääsee helpommin eroon tuot-

teista, jotka ovat päätyvässä hävikkiin. Punalaputus voi pienentää hävikkiä, mutta se ei kuitenkaan korjaa ongelmaa ja sen syitä. Hävikki ja punalaputus aiheuttavat katteen menetystä eli hukkaa. Lisäksi ne johtavat usein tyhjiin hyllypaikkoihin. (Ketola, Rikberg, Österlund 2017, 12.)

Kauppavalmennuksella on kehittänyt hävikin hallintaan metodin, joka perustuu lean-ajatteluun. Sillä yritetään löytää kaikelle hävikille perimmäiset syyt ja korjata ne. Ongelmaan on helppo puuttua, jos tuotteesta voidaan tunnistaa kaava, joka aiheuttaa hävikin syntymisen. Metodilla pyritään rakentamaan malleja hävikin perimmäisistä syistä, jotka pätevät koko ketjussa tai mahdollisesti koko alalla. Niiden avulla rakennetaan toimintatapoja, joiden avulla hävikkiä voidaan vähentää koko ketjussa. Myymälöiden tulee seurata hävikkiä säännöllisesti ja tarkasti sekä reagoida nopeasti tunnistettuun hävikkiin, jotta metodi toimisi. Kauppavalmennuksen visio on, ettei ole hävikkiä, punalaputusta tai hyllypuutteita. Hukkaa ei vähennetä siirtämällä ongelma muualle tai heikentämällä asiakaskokemusta. Visio ei ole realistinen vaan sen tarkoitus on antaa suunta, jota kohti edetään. (Ketola ym. 2017, 13.)

Metodia kokeiltiin 13 myymälässä (Ketola ym. 2017, 14). Tuloksien arvioitiin käytettiin 11 viikon jaksoa kokeilun lopussa. Hävikki väheni useimmissa tapauksissa, erityisesti osastoilla, joissa tuotteita tilataan automaattijärjestelmällä. Kokeilussa huomattiin tarve tilanteeseen sopivalle työkalulle etenkin osastoilla, joissa tuotteita tilataan käsin. Hävikin vähentäminen vaatii standardoitua, jatkuvaa ja säännöllistä työtä, jotta huomataan mikä toimii ja mikä ei. Kokeilun jälkeen suurin osa myymälöistä palasivat vanhoihin tapoihin, koska heillä ei ole työkaluja noudattaa Kauppavalmennuksen metodia. (Ketola ym. 2017, 23–24.)

3.2 Kauppavalmennus-sovellus

Kauppavalmennus Oy on kehittänyt mobiilisovelluksen, jota käytetään apuna hävikin vähentämisessä. Se on päivittäistavarakauppojen jokapäiväinen työkalu. Kauppavalmennus-sovellus kokoaa muun muassa kauppojen myynti- ja hävikkiraportit yhdeksi kokonaisuudeksi tuotekortteihin, jolloin työntekijän on helpompi löytää hävikin perimmäiset syyt. Sovelluksessa on erilaisia toimintoja, kuten hukan hallinta ja tilausavustaja.

Hukan hallinta -toiminto listaa työntekijälle tuotteet, joissa on paljon hukkaa. Lista on jaettu kolmeen osaan riippuen hukan määrästä. Paljon hukkaa aiheuttavat tuotteet on listattu punaisiin, vähäisemmät oranssiin ja vähiten aiheuttavat vihreisiin. Työntekijä käy listat läpi tuote kerrallaan aloittaen punaisista. Hän selvittää hukan aiheuttajan tuotekortissa näkyvien tietojen avulla ja suorittaa vaaditut toimenpiteet hukan vähentämiseksi. Työntekijän tulee kuitata tuote käsitellyksi, jolloin se siirtyy hukkalistalla vihreisiin. Tuotetta ei poisteta kokonaan listoilta, koska tehdyt toimenpiteet eivät välttämättä korjanneet ongelmaa. Jos hukkaa vielä kertyy, tuote siirtyy uudelleen punaisiin.

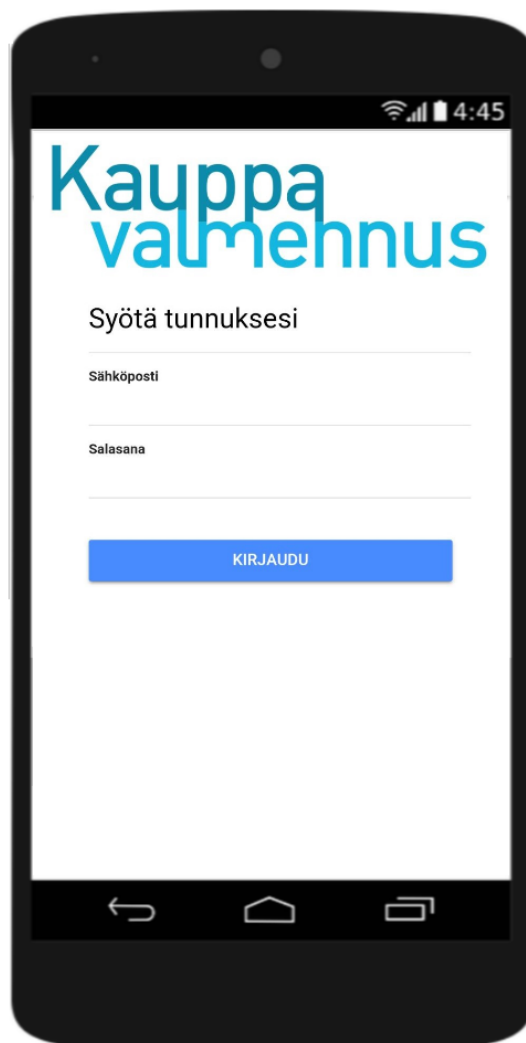
Tilausavustaja-toiminnon avulla työntekijä näkee valitun tuotteen tuotekortin, jossa on viikoittaiset myynnit ja hävikit. Ne on listattu taulukkoon, jossa hävikin syyt on värikoodattu. Harmaa tarkoittaa tyhjää hyllypaikkaa. Vihreä tarkoittaa, että tuote on ollut kampanjassa. Punainen tarkoittaa, että tuote on punalaputettu. Tuotekorttiin on lisätty myös tuotteen viivakoodi. Siitä se voidaan tarvittaessa lukea helposti toisella laitteella. Tuotekorttiin voi lisätä myös tuotetta koskevan muistutuksen, esimerkiksi tiedon tulevasta kampanjasta.

4 SOVELLUKSEN TOTEUTUS

Viivakoodin lukemista testattiin erillisellä prototyypillä ennen työn varsinaista aloittamista. Sovelluksessa noudatettiin MVC-mallia, jonka ansiosta samoja rajapintakutsuja voi käyttää useammassa sovelluksen osassa eri käyttötarkoituksiin. Ohjelmointi aloitettiin käyttäjän kirjautumisesta. Sen jälkeen siirryttiin Myymälänhallinta -näkömään. Siellä käyttäjä valitsee myymälän, jonka tietoja halutaan käsitellä. Viivakoodin pystyi kirjoittamaan tai skannaamaan älypuhelimella. Sovellus lähetti tuotekoodin tietokantaan, ja viimeksi lähetetyt tuotekoodit listattiin taulukkoon.

4.1 Kirjautuminen

Kirjautumisnäkömään (Kuva 13) ulkoasussa käytettiin valmiita Ionic-komponentteja, joten ne asettuvat ja skaalautuvat automaattisesti.



Kuva 13. Kirjautumisnäkömä.

Kirjautumisessa käytettiin Json Web Token -menetelmää (JWT) ja Kauppa-valmennuksen valmista ASP.NET WebAPI -rajapintaa. Käyttäjälle rekisteröitiin käyttäjätunnus rajapintakutsulla, jonka lähettämiseen käytettiin Postman-sovellusta.

LoginPage-luokassa kutsuttiin authService-luokassa olevaa login-metodia (Ohjelmakoodi 1), jolle välitettiin kirjautumistiedot parametrinä. Käyttäjänystötteisiin päästiin käsiksi Angularin NgModel-direktiivin avulla ja ne tallennettiin loginData-muuttujaan. Jos käyttöoikeustietueen palauttaminen onnistui, käyttäjä ohjattiin Myymälänhallinta -näkymään. Jos palauttaminen epäonnistui, näytettiin ponnahdusikkuna, jossa kerrottiin tunnuksen olevan virheelliset. PresentAlert-metodilla määritettiin ponnahdusikkunan sisältö.

```
login() {
  this.auth.login(this.loginData).then(() =>
  {
    this.navCtrl.setRoot(OrganizationsPage);
  },
  error =>
  {
    this.presentAlert()
  });
}

presentAlert() {
  let alert = this.alertCtrl.create({
    title: 'Kirjautuminen epäonnistui',
    subTitle: 'Virheellinen käyttäjätunnus tai salasana.',
    buttons: ['OK']
  });
  alert.present();
}
```

Ohjelmakoodi 1. LoginPage-luokan Login- ja presentAlert-metodien ohjelmakoodi.

Kirjautumista varten luotiin authService-luokka, jossa varsinainen käyttäjän autentikointi suoritettiin. Login-metodissa (Ohjelmakoodi 2) kutsuttiin rajapintaa, joka palautti käyttöoikeustietueen, jos tunnus ja salasana olivat oikein. Rajapintakutsuun lisättiin palvelinosoite sekä avainparit käyttäjätunnuksesta ja salasanasta. Kutsun otsikkoon asetettiin tietoa, jota käytetään käyttöoikeustietueen allekirjoituksessa ja salauksessa. Login-metodissa palautettiin Promise-olio, jolloin sen tilan perusteella toimittiin LoginPage-luokassa.

```
login(credentials)
{
  return new Promise((resolve, reject) =>
  {
    let data = "grant_type=" + this.grant_type
      + "&username=" + credentials.username
      + "&password=" + credentials.password;

    let headers = new HttpHeaders();
```

```

headers.append("Content-Type",
'application/x-www-form-urlencoded');

this.http.post(env.apiUrl + "/Token", data,
{headers}).subscribe(data => {
this.access_token = data["access_token"];
this.tokenService.setToken(this.access_token);
resolve(data);
},
error =>
{
reject("Epäonnistui " + error);
});
});
}

```

Ohjelmakoodi 2. AuthService-luokan Login-metodin ohjelmakoodi.

Sovellukseen tehtiin tokenService -luokka, jota käytettiin käyttöoikeustietueen ja käyttäjän valitseman myymälän tietojen tallentamiseen ja hakemiseen laitteen muistista. Siihen käytettiin Angular 2:n tarjoamaa local storage -palvelua.

Jotta jokaisella rajapintakutsulla ei tarvitsisi erikseen lisätä otsikkoon käyttöoikeustietuetta, sovellukseen lisättiin auth.Interceptor-luokka. Intercept-metodi (Ohjelmakoodi 3) pysäyttää rajapintakutsun ennen kuin se lähetetään palvelimelle. Sillä haettiin valitun organisaation ja käyttöoikeustietueen tiedot tokenService-luokasta. Käyttöoikeustietue lisättiin rajapintakutsun Authorization-otsikkoon. Jotta kutsua voidaan muokata, kloonataan alkuperäinen kutsu. Siihen muutettiin otsikko, ja sen jälkeen kloonattu kutsu lähetettiin rajapintaan.

```

intercept (req: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>>
{
this.organization = this.tokenService.getOrganization();
let token = this.tokenService.getToken();

const headers = new HttpHeaders({
'Authorization': 'Bearer ' + token,
'X-Organization': this.organization,
'Content-Type': 'application/json'
});

const cloneReq = req.clone({headers});

return next.handle(cloneReq);
}

```

Ohjelmakoodi 3. Intercept-metodin ohjelmakoodi

4.2 Myymälän valitseminen

Myymälänhallinta -näkymään (Kuva 14) listattiin kaikki käyttäjän myymälät. Niistä valitaan se, jonka tietoja halutaan käsitellä. Myymälän tunnisteen avulla tiedot saadaan tallennettua oikeaan paikkaan tietokannassa.



Kuva 14. Myymälänhallinta -näkymä.

OrganizationService-luokkaa käytettiin myymälöiden tietojen hakemiseen. Sinne luotiin getAllOrganizations-metodi, joka teki rajapintakutsun ja palautti listan myymälöistä. Kyseistä metodia kutsuttiin OrganizationPage-luokassa, ja sen palauttamat tiedot tallennettiin muuttujaan. Jotta myymälöiden nimet saatiin lisättyä Organizations-näkymän painikkeisiin (Ohjelmakoodi 4), käytettiin ngFor-direktiiviä. Se lisää näkymään niin monta painiketta kuin listassa on alkioita.


```

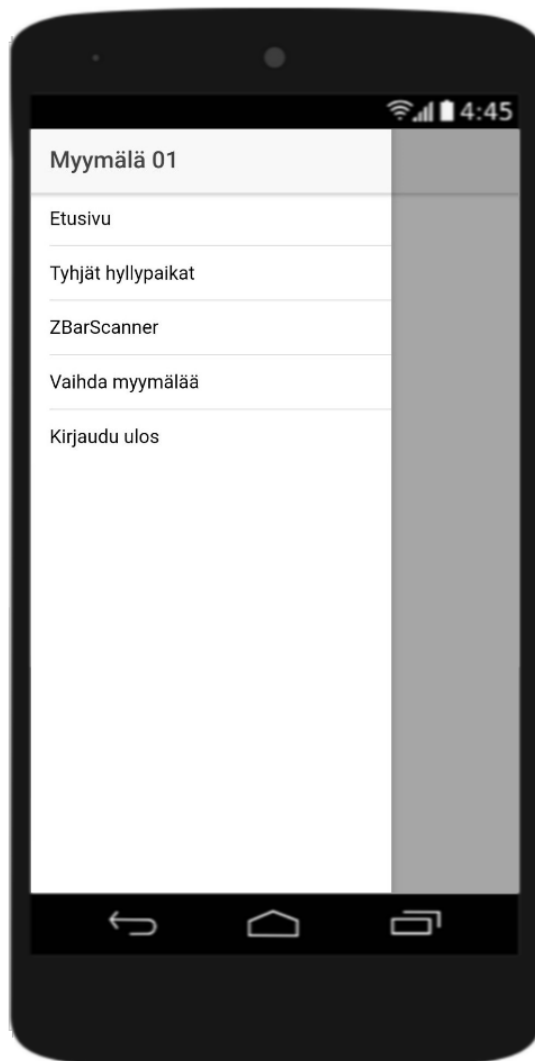
<ion-content padding>
  <ion-list>
    <ion-item no-lines *ngFor="let Organization of organiza-
tions">
      <button ion-button color="light" block padding
(click)="selectOrganization(Organization.Id)">
        {{Organization.Name}}
      </button>
    </ion-item>
  </ion-list>
</ion-content>

```

Ohjelmakoodi 4. Myymälöiden nimet lisättiin painikkeisiin ngFor-direktiivillä

Myymälä valittiin painiketta painamalla, jonka jälkeen käyttäjä ohjattiin sovelluksen etusivulle.

4.3 Sivuvalikko ja uloskirjautuminen



Kuva 15. Sivuvalikko.

Sovelluksessa käytettiin Ionicin sivupohjaa, jossa on valmiiksi sivuvalikko (Kuva 15). Sen sisältö määriteltiin NavController-luokan listalla. App.component.ts-luokkaan luotiin vastaava pages-lista, johon haettiin alkiot NavController-luokan listasta. App.html tiedostossa valikkoon luotiin painikkeet pages-listalla oleville objekteille (Ohjelmakoodi 5).

```
<ion-content>
  <ion-list>
    <button menuClose ion-item *ngFor="let p of pages"
      (click)="openPage(p)">
      {{p.title}}
    </button>
  </ion-list>
</ion-content>
```

Ohjelmakoodi 5. App.html ohjelmakoodi.

Erillinen luokka sisällön hallintaan luotiin, jotta valikon otsikkoon saatiin asetettua myymälän nimi. Siihen käytettiin Ionicin Events-palvelua, joka lähettää ja vastaanottaa tapahtumia (event). Tapahtumaa lähetettäessä myymälän valitsemisen jälkeen sille annettiin aihe, jota käytettiin sen vastaanottamisessa (Ohjelmakoodi 6) navCtrl.ts-luokassa. Sen yhteydessä asetettiin valikon otsikko.

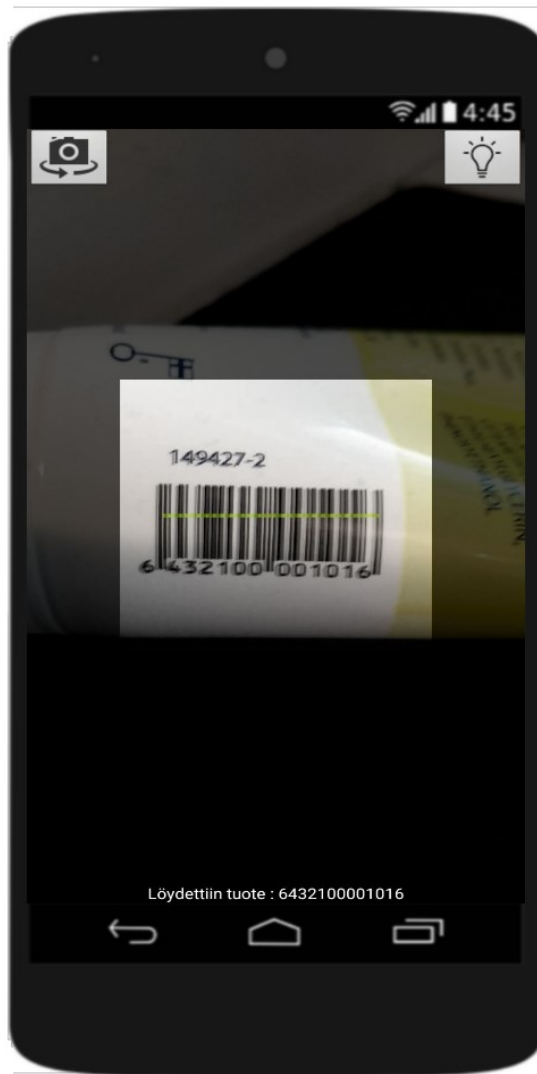
```
this.events.subscribe('organization:selectOrganization',
  ()=>
  {
    this.organizationService.getCurrentOrganiza-
tion().then((res)=>
    {
      this.currentOrganization = res["Name"];
    });
  });
```

Ohjelmakoodi 6. Lähetyksen vastaanottamisen ohjelmakoodi.

Valikkoon lisättiin sivujen lisäksi painike ulos kirjautumista varten. Kun painiketta painettiin, app.component.ts-luokassa kutsuttiin authService-luokan logout-metodia. Se tyhjensi laitteen muistiin tallennetut tiedot. Sen jälkeen käyttäjä ohjattiin kirjautumisnäkykseen.

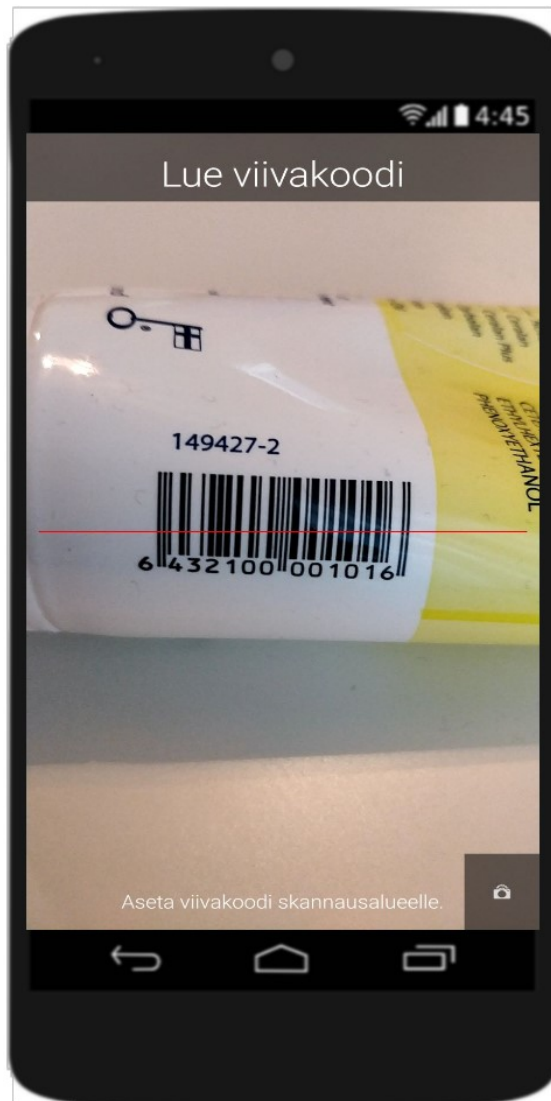
4.4 Viivakoodin lukeminen

Viivakoodin lukemiseen testattiin kahta eri lisäosaa. Molemmat avaavat laitteen kameran, skannaavat viivakoodin ja palauttavat tiedot automaattisesti.



Kuva 16. Barcode Scanner -lisäosan skannausnäkymä.

Barcode Scanner -lisäosa (Kuva 16) tukee Android-, iOS-, Windows- ja BlackBerry 10 -käyttöjärjestelmiä. Sillä voidaan lukea eri tyyppisiä viivakodeja riippuen käyttöjärjestelmästä. Useimmilla voidaan lukea muun muassa EAN8-, EAN13-, DataMatrix- ja QR-koodeja. Barcode Scanner asennettiin muutamalla CLI-komennolla. Lisäosa luo barcodeScanner-objektin, jossa on scan-metodi. Sillä on kaksi callback-metodia success ja fail. Success palauttaa objektin, jossa on viivakoodin tiedot, tyyppi ja tieto siitä, peruuttiko käyttäjä skannauksen. Scan-metodille voi halutessaan antaa parametriksi vapaaehtoiset asetukset, esimerkiksi salaman käyttö ja otsikko.



Kuva 17. ZBar-lisäosan skannausnäkyvä.

Sovelluksessa testattiin myös ZBar Scanner -lisäosaa (Kuva 17), koska se on nopeampi skannaamaan viivakoodin kuin Barcode Scanner. Sillä voidaan lukea 2d-koodeja, kuten QR-koodi ja DataMatrix. Niiden lisäksi voidaan lukea esimerkiksi EAN8-, EAN13- ja Code 128 -koodeja. ZBar tukee Android- ja iOS-käyttöjärjestelmiä. Se vaatii toimiakseen Ionicin Natiivin Camera-lisäosan, jotta laitteen kameraa voidaan käyttää. ZBar Scanneria käytetään samanlailla kuin Barcode Scanner -lisäosaa. Ainoa ero on se, että scan-metodille on annettava parametriksi lista asetuksista. Niitä on esimerkiksi ot-sikko, käytetäänkö etu vai takakameraa ja käytetäänkö salamaa.

Viivakoodin skannaamista varten luotiin barcodeScannerService-luokka, jossa käytettiin sekä Barcode Scanner- että ZBar-lisäosaa. Luokkaan luotiin kaksi scan-metodia, joiden toimintaperiaate on sama. Scan-metodilla (Ohjelmakoodi 7) luettiin viivakoodi, jonka jälkeen kutsuttiin getBarcode-metodia. Se palautti scannedBarcode-objektin, johon asetettiin aikaleima, paikka tuotekoodille, lähettämisen status ja tokenService-luokasta haettu

myymälän tunniste. Scan-metodissa asetettiin skannattu tuotekoodi tyhjäksi alustetun alkion tilalle. Scan-metodi palautti Promise-olion. Jos viivakoodin lukeminen onnistui, resolve-metodilla palautettiin scannedBarcode-objekti. Scan-metodille annettiin parametriksi lista asetuksista, joissa määritettiin muun muassa viivakoodin luku -tilassa näkyvä otsikko ja ohjeteksti.

```
this.barcodeScanner.scan(options).then((result) =>
{
  this.scannedBarcode = this.getBarcode();
  this.scannedBarcode['AggregateId'] = result.text;

  resolve(this.scannedBarcode);
},
error =>
{
  reject("Epäonnistui " + error);
});
```

Ohjelmakoodi 7. Barcode Scanner -lisäosan scan-metodin ohjelmakoodi.

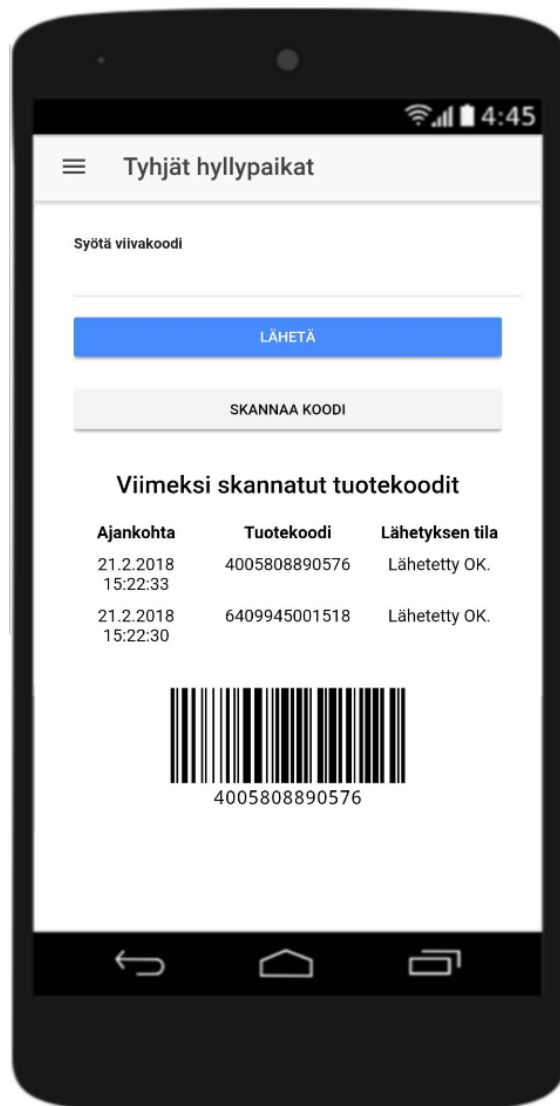
EmptyShelfScan-luokassa viivakoodin luettiin Barcode Scanner -lisäosalla tai käytettiin käsin kirjoitettua koodia. Scan-metodi kutsui barcodeScannerService-luokan scan-metodia. Kun viivakoodin tiedot saatiin, kutsuttiin sendScannedBarcode-metodia. Se tallensi scannedBarcode-objektin tiedot listaan, joka lähetettiin parametrinä emptyShelfService-luokan processSuccessScan-metodille. ZBar-lisäosaa varten luotiin oma ZBarScanner-sivu, jossa oli samanlaiset metodit viivakoodin lukemiseen ja lähettämiseen kuin emptyShelfScan-luokassa. ZBarScanner-näkymään lisättiin painike viivakoodin lukemista varten, ja viimeksi skannattujen tuotteiden tiedot listattiin kuten Tyhjä hyllypaikat -näkymässä.

Rajapintakutsu, joka lähetti tiedot tietokantaan, suoritettiin EmptyShelfService-luokassa olevalla processSuccessScan-metodilla (Ohjelmakoodi 8). Tietokantaan lähetettiin myymälän tunniste, skannauksen ajankohta ja tuotekoodi.

```
processSuccessScan(data) {
  return new Promise ((resolve, reject) =>
  {
    this.http.post(env.apiUrl +
      "/api/Product/MarkOutOfStock",
      JSON.stringify(data)).subscribe(data =>
      {
        resolve("Lähetetty OK.");
      },
      error =>
      {
        reject("Epäonnistui, ei uudelleenlähetystä. ");
      });
  });
}
```

Ohjelmakoodi 8. ProcessSuccessScan-metodin ohjelmakoodi.

Kun tiedot oli tallennettu tietokantaan onnistuneesti, emptyShelfScan-luokan sendScannedBarcode-metodissa lisättiin scannedBarcode-objektin tiedot lastScannedBarcodes-listaan.



Kuva 18. Tyhjät hyllypaikat -näkymä.

Viivakoodin luku -ominaisuutta ohjelmoidessa todettiin, että viivakoodi pitäisi voida syöttää myös käsin. Päivittäistavarakaupoissa voi olla heikko valaistus, ja hintalapuissa saattaa olla viivakoodi vain numeroina. Tästä syystä Tyhjät hyllypaikat -näkymään (Kuva 18) lisättiin tekstikenttä ja painike, jotta viivakoodi voitiin syöttää käsin. Viivakoodin käsin syöttämistä varten emptyShelfScan-luokkaan luotiin submitBarcode-metodi. ScannedBarcode-objektiin haettiin tiedot barcodeScannerService-luokan getBarcode-metodilta. Tuotekoodin tilalle asetettiin tekstikenttään syötetty arvo. Tyhjät hyllypaikat -näkymään listattiin viimeksi luettujen tuotekoodien ajankohta, tuotekoodi ja lähetysten tila. Tiedot saatiin emptyShelfScan-luokan lastScannedBarcodes-listasta.

Kauppavalmennus-sovelluksen vanhoista ominaisuuksista päätettiin kokeilla viivakoodin piirtämistä Tyhjät hyllypaikat -näkymään. Siihen käytettiin JsBarcode-kirjastoa, joka generoi viivakoodin. Se tukee useita eri viivakoodi muotoja, ja se toimii selaimessa ja Node.js-sovellusalustalla. JsBarcode-kirjasto asennettiin CLI-komennolla `npm install jsbarcode -save`. HTML-tiedostossa luotiin viivakoodille piirtoalue `svg`-tunnisteella. Jotta tunnisteeseen päästiin käsiksi `emphyShelfScan`-luokassa, käytettiin Angularin `@ViewChild`-decoratoria. Viivakoodi generoitiin JsBarcode-metodilla, jolle asetettiin parametreiksi HTML-tunniste ja skannattu tuotekoodi. Viivakoodi piirrettiin Tyhjät hyllypaikat -näkymään tietokantaan lähettämisen jälkeen.

5 SOVELLUKSEN JULKAISU WEB- JA MOBIILISOVELLUKSENA

Android-julkaisuversio generoidaan CLI-komennolla `ionic cordova build --release android`. Siinä käytetään `config.xml`-tiedostossa laadittuja asetuksia. Komento luo allekirjoittamattoman (unsigned) APK -tiedoston (Android application package). Se tulee allekirjoittaa sovelluksen allekirjoitusavaimella, joka luodaan JDK:n (Java Development Kit) `keytool`-komennolla. Sitä tarvitaan myös sovelluksen päivittämisessä. Allekirjoittamiseen voidaan käyttää JDK:n `jarsigner`-työkalua. APK:n optimoimiseen käytetään `zipalign`-työkalua, jolloin sovellus käyttää vähemmän muistia ja käynnistyy nopeammin. APK:n allekirjoittamiseen on myös muita tapoja. Jotta sovellus voidaan julkaista Google Play Storessa, täytyy rekisteröityä Google Play -kehittäjäksi. Sovellukselle kirjoitetaan perustiedot, kuten kuvaus. Sen jälkeen ladataan palveluun APK:n julkaisuversio.

Jotta sovellus voidaan julkaista iOS-käyttöjärjestelmälle, rekisteröidytään Apple-kehittäjäksi, ja XCode yhdistetään kehittäjätunnuksiin. Sovellukselle asetetaan App ID -tunnistetiedot, jotta sillä on pääsy palveluihin kuten Apple Play ja Wallet. iTunes Connect -palvelulla sovellus voidaan julkaista App Storessa tai jakaa sen betaversiota TestFlight-sovelluksella. Sovelluksen julkaisuversio generoidaan CLI-komennolla `ionic cordova build ios --release`. Sen jälkeen XCodella avataan `xcodeproj`-tiedosto, josta tarkistetaan muun muassa tunnistetietojen olevan oikein. Lisäksi valitaan mitä laitteita sovellus tukee. Sovelluksesta luodaan arkisto, jonka `build configuration` -asetukseksi asetetaan `release`. Sen jälkeen arkisto on valmis ladattavaksi App Storeen. iTunes Connect -palvelussa sovellukselle asetetaan tiedot, kuten kuvaus ja avainsanat. Build-osiossa valitaan XCodella ladattu arkisto. Muiden tietojen asettamisen jälkeen sovellus tallennetaan ja odotetaan Applen suorittamaa arviointia.

Ionic-sovelluksesta voidaan luoda Progressive Web App (PWA). Sen kehitti Google vuonna 2015. PWA on websivu, joka käyttää perinteisten websivujen ja natiivien mobiilisovellusten ominaisuuksia. Suurin etu sillä on se, että sitä voidaan käyttää myös offline-tilassa. Jotta Ionic-sovelluksesta saadaan PWA, siihen täytyy lisätä Web Manifest ja Service Worker. Web Manifestin avulla PWA ladataan automaattisesti käyttäjän aloitusnäyttöön. Manifest.json-tiedosto, johon listataan asennuksen vaatimukset, lisätään www-kansioon. Se yhdistetään `index.html`-tiedostoon koodilla:

```
<link rel="manifest" href="manifest.json">
```

Service Worker lisätään projektiin luomalla JS-tiedosto `www`-kansioon. Google tarjoaa useita valmiita Service Worker -skriptejä, joista haluttu kopioidaan suoraan luotuun JS-tiedostoon. Lopuksi JS-tiedosto lisätään `index.html`-tiedostoon. Sovelluksen julkaisuversio generoidaan komennolla `npm run ionic:build --prod`. Se luo projektiin `www`-kansion, jonka sisältö ladataan palvelimelle.

6 YHTEENVETO

Ionic on aloittelijalle helposti lähestyttävä, koska siitä löytyy paljon dokumentaatiota. Sen käytössä tarvitsee hallita useampi viitekehys, mutta niiden käytön oppii suhteellisen nopeasti. Lisäksi sovelluksen testaaminen laitteessa on helppoa. Ionicin tarjoamat työkalut vaikuttavat erittäin päteviltä, vaikka niitä ei tämän opinnäytetyön käytännön osuudessa hyödynnetty.

Opinnäytetyön tavoitteena oli kehittää Ionic-sovellus, jolla voidaan lukea tuotteiden viivakoodeja älypuhelimella. Toimeksiantaja halusi selvittää, onko nykyisestä Kauppavalmennus-sovelluksesta helppo tehdä Ionic-versio. Teoriassa tutustuttiin sovelluksen toteutustekniikoihin. Ionicista tuotiin esiin sen tarjoamia työkaluja, jotka helpottavat sovellusten kehittämistä.

Tavoitteessa onnistuttiin hyvin, ja sovellus täyttää sille asetetut vaatimukset. Sitä testattiin Android- ja iOS-käyttöjärjestelmillä onnistuneesti. Sovelluksessa kokeiltiin kahta eri lisäosaa viivakoodin lukemiseen. Testauksen yhteydessä todettiin, että Barcode Scanner -lisäosa soveltuu käyttötarkoitukseen paremmin. ZBar-lisäosa vaati asetusten muokkaamista toimiakseen eri iOS-versioilla, joten se päätettiin poistaa sovelluksesta.

Opinnäytetyössä haluttiin selvittää, olisiko Kauppavalmennus-sovelluksesta hankalaa tuoda vanhoja ominaisuuksia Ionic-versioon. Päätettiin kokeilla viivakoodin piirtämistä Tyhjät hyllypaikat -näkymään. Sen toteuttaminen oli helppoa, koska JsBarcode-kirjaston tuotiin Ionic-sovellukseen yhdellä komennolla ja import-syntaksilla. Kirjaston käyttäminen ei ollut vaikeaa.

Alussa uutta opittavaa oli todella paljon, joten sovelluksen kehittäminen oli melko hidasta. Haastavinta oli saada kirjautumisen logiikka toimimaan. Rajapintakutsujen tekeminen oli vaikeaa ilman opastusta, mutta sen jälkeen niiden tekeminen sujui ongelmitta. Opinnäytetyö oli opettavainen kokemus. Uutta opittavaa oli todella paljon, mikä teki prosessista erityisen mielenkiintoisen.

LÄHTEET

Ali Syed B. (2017.) *TypeScript Deep Dive*. Samurai Media Limited. Haettu 19.1.2018 osoitteesta

<https://basarat.gitbooks.io/typescript/>

Anand, I. & Wasmer, D. (n.d.). *Native vs Web vs Hybrid*. Haettu 19.1.2018 osoitteesta

<https://go.kinvey.com/native-web-hybrid-developers-ebook/>

Angular (n.d.a). *Angular Glossary*. Haettu 26.1.2018 osoitteesta

<https://angular.io/guide/glossary>

Angular (n.d.b). *Architecture Overview*. Haettu 30.1.2018 osoitteesta

<https://angular.io/guide/architecture>

Angular (n.d.c.) *Component*. Haettu 31.1.2018 osoitteesta

<https://angular.io/api/core/Component>

Angular (n.d.d.) *NgModules*. Haettu 14.2.2018 osoitteesta

<https://angular.io/guide/ngmodules>

Angular (n.d.e.) *AngularJS to Angular Quick Reference*. Haettu 5.3.2018 osoitteesta

<https://angular.io/guide/ajs-quick-reference>

AngularJS (2018a). *Introduction*. Haettu 25.1.2018 osoitteesta

<https://docs.angularjs.org/guide/introduction>

AngularJS (2018b). *Data binding*. Haettu 25.1.2018 osoitteesta

<https://docs.angularjs.org/guide/databinding>

AngularJS (2018c). *Modules*. Haettu 31.1.2018 osoitteesta

<https://docs.angularjs.org/guide/module>

Apache Cordova (2016). *Overview*. The Apache Software Foundation. Haettu 19.1.2018 osoitteesta

<https://cordova.apache.org/docs/en/latest/guide/overview/index.html>

Branas, R. (2014). *AngularJS Essentials*. Birmingham: Packt Publishing Ltd. Haettu 25.1.2018 osoitteesta

<https://www.packtpub.com/packt/free-ebook/angularjs-essentials>

Cleverism (n.d.). *TypeScript*. Haettu 25.1.2018 osoitteesta

<https://www.cleverism.com/skills-and-tools/typescript/>

Codecraft (n.d.). *Promises*. Haettu 1.2.2018 osoitteesta

<https://codecraft.tv/courses/angular/es6-typescript/promises/>

Google+ (2012). *MVC vs MVVM vs MVP*. Haettu 30.1.2018 osoitteesta
<https://plus.google.com/+AngularJS/posts/aZNVhj355G2>

GS1 (n.d.). *Viivakoodit*. Haettu 20.1.2018 osoitteesta
http://www.gs1.fi/content/download/4705/30095/file/1.4+viivakoodi-taulu_suomi.pdf

Ionic Framework (n.d.a). *All about Ionic*. Haettu 18.1.2018 osoitteesta
<https://ionicframework.com/docs/v1/guide/preface.html>

Ionic Framework (n.d.b). *CLI docs*. Haettu 20.1.2018 osoitteesta
<https://ionicframework.com/docs/cli/>

Ionic Framework (n.d.c). *Components*. Haettu 31.1.2018 osoitteesta
<https://ionicframework.com/docs/components/#overview>

Ionic Framework (n.d.d). *Welcome to Ionic Pro*. Haettu 31.1.2018 osoitteesta
<https://ionicframework.com/docs/pro/basics/welcome/>

Ionic Framework (n.d.e). *Creator*. Haettu 31.1.2018 osoitteesta
<https://ionicframework.com/pro/creator>

Ionic Framework (2017.) *Grid*. Haettu 23.2.2018 osoitteesta
<https://ionicframework.com/docs/api/components/grid/Grid/>

Ionic Framework (2018a). *Core Concepts*. Haettu 20.1.2018 osoitteesta
<https://ionicframework.com/docs/intro/concepts/>

Ionic Framework (2018b). *Installing Ionic*. Haettu 31.1.2018 osoitteesta
<https://ionicframework.com/docs/intro/installation/>

Ionic Framework Blog (2017a). *Announcing Ionic DevApp*. Haettu 31.1.2018 osoitteesta
<https://blog.ionicframework.com/announcing-ionic-devapp/>

Ionic Framework Blog (2017b). *Announcing Creator Ionic 3 Support (And what's coming next)*. Haettu 31.1.2018 osoitteesta
<https://blog.ionicframework.com/announcing-creator-ionic-3-support-and-whats-coming-next/>

Ionic-Team (2017). *Releases*. Haettu 14.2.2018 osoitteesta
<https://github.com/ionic-team/ionic/releases>

JWT (n.d.). *Introduction to JSON web Tokens*. Haettu 1.2.2018 osoitteesta
<https://jwt.io/introduction/>

Ketola, J., Rikberg, E. & Österlund, H. (2017). *Hävikin hallinnan tehostaminen päivittäistavarakaupan myymälöissä*. Haettu 17.1.2018 osoitteesta https://www.ptv.fi/fileadmin/user_upload/tiedostot/Julkaisut/Muut_julkaisut/Havikinhallinnan_tehostaminen_pt_kaupassa_2017.pdf

Ravulavaru A. (2017). *Learning Ionic - Second Edition*. Birmingham: Packt Publishing Ltd. Haettu 15.1.2018 osoitteesta https://www.packtpub.com/mapt/book/web_development/9781786466051

Rownski, D. (2012). *Why Facebook Ditched the Mobile Web & Went Native With its New iOS App*. Blogijulkaisu 23.8.2012. Haettu 25.1.2018 osoitteesta <https://readwrite.com/2012/08/23/how-facebook-ditched-the-mobile-web-went-native-with-its-new-ios-app/>

Stecky-Efantis, M. (2016). *5 Easy Steps to Understanding JSON Web Tokens (JWT)*. Haettu 1.2.2018 osoitteesta <https://medium.com/vandium-software/5-easy-steps-to-understanding-json-web-tokens-jwt-1164c0adfcec>

Telerik Developer Network (2015). *What is a Hybrid Mobile App*. Haettu 25.1.2018 osoitteesta <https://developer.telerik.com/featured/what-is-a-hybrid-mobile-app/>

Telerik Developer Network (2017). *What is TypeScript*. Haettu 25.1.2018 osoitteesta <https://developer.telerik.com/topics/web-development/what-is-typescript/>

Tutorialspoint (n.d.). *Basic MVC Architecture*. Haettu 30.1.2018 osoitteesta https://www.tutorialspoint.com/struts_2/basic_mvc_architecture.htm

Tutorials Teacher (n.d.). *MVC Architecture*. Haettu 30.1.2018 osoitteesta <http://www.tutorialsteacher.com/mvc/mvc-architecture>