Markus Heikkilä

AWS-pilvipalvelua käyttävän kassaohjelmiston toteuttaminen Android-laitteille

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikka

Insinöörityö

17.4.2018

Metropolia

Tekijä Otsikko Sivumäärä Aika	Markus Heikkilä AWS-pilvipalvelua käyttävän kassaohjelmiston toteuttaminen Android-laitteille 41 sivua 17.4.2018
Tutkinto	Insinööri (AMK)
Tutkinto-ohjelma	Tietotekniikan koulutusohjelma
Ammatillinen pääaine	Ohjelmistotekniikka
Ohjaajat	lehtori Juha Kämäri lehtori Jussi Alhorinne ohjelmistoarkkitehti Lasse Pajunen

Insinöörityön tarkoituksena oli toteuttaa Amazon Web Services -pilvipalvelua (AWS) käyttävä kassaohjelmisto Android-laitteille. Kassaohjelmisto tarvitsi erillisen maksupäätteen toimiakseen.

Kassaohjelmisto on toteutettu Java-ohjelmointikielellä, ja se käyttää AWS:n tarjoamia kirjastoja toimiakseen taustajärjestelmän kanssa. Kassaohjelmiston taustajärjestelmänä toimii AWS-pilvipalvelussa Lambda-funktio, joka on toteutettu JavaScript-ohjelmointikielellä Node.js-ajoympäristöön. Kassaohjelmisto käyttää tiedon tallentamiseen PostgreSQL-tietokantaa, joka sijaitsee AWS:n Relational Database -palvelussa (RDS). Tietokannan päivitysoperaatiot on toteutettu Scala-ohjelmointikielellä.

Insinöörityön alussa käydään läpi Android-käyttöjärjestelmän ja AWS-pilvipalvelun historiaa sekä muiden kassaohjelmistojen sisältämiä ominaisuuksia. Tämän jälkeen käydään läpi kassaohjelmiston toteuttamista Android-ympäristössä sekä sen käyttämiä AWS:n palveluita. Lopuksi insinöörityössä käydään läpi toteutetun kassaohjelmiston käyttöä sekä sitä, kuinka tarvittavat kassaohjelmiston ominaisuudet on toteutettu.

Avainsanat	Android, AWS, Java, Javascript, PostgreSQL

Author Title Number of Pages Date	Markus Heikkilä Implementing a point of sale application which uses AWS cloud service for Android devices 41 pages 17 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Professional Major	Software Engineering
Instructors	Juha Kämäri, Senior Lecturer Jussi Alhorinne, Senior Lecturer Lasse Pajunen, Software Architect

The subject of this thesis was to implement a point of sale software, which uses Amazon Web Services cloud service, for Android devices. The software needed a separate payment terminal to work.

The point of sale software is implemented with Java programming language and it uses AWS libraries to work with the backend. The point of sale software's backend is a Lambda function which operates in AWS cloud service and it is made with JavaScript programming language in Node.js runtime. The point of sale software uses PostgreSQL database to store data, which runs on AWS Relational Database -service (RDS). The databases migrations are implemented with Scala programming language.

In the start of this thesis we will go through Android operating system and AWS cloud service history and the features in other point of sale software's. After this we will go through implementing the point of sale software in Android environment and the services it uses from AWS. At the end of this thesis we will go through the usage of the implemented point of sale software and how the required features are implemented.

Keywords	Android, AWS, Java, Javascript, PostgreSQL

Sisällys

Lyhenteet

APK	Android Application Package. Android-laitteille tehtyjen sovellusten tiedostotyyppi.
ASCII	American Standard Code for Information Interchange. Tietokonemerkistö, joka sisältää amerikanenglannin sisältämiä merkkejä.
AWS	Amazon Web Services. Amazonin pilvipalvelu, joka tarjoaa monipuolisia työvälineitä ja palveluita sovellusten tekemiseen.
CIA	Central Intelligence Agency. Yhdysvaltojen keskustiedustelupalvelu, joka analysoi tiedustelutietoja ulkomaisista hallituksista, järjestöistä ja yksilöistä.
HTML	Hypertext Markup Language. Kuvauskieli, jonka avulla kuvataan hyperlinkkejä sisältävää tekstiä.
HTTP	Hypertext Transfer Protocol. Protokolla, jota sovellukset käyttävät tiedonsiirtoon.
IaaS	Infrastructure as a Service. Palvelimien ja palvelinsalien ulkoistaminen.
IBM	International Business Machines Corporation. Teknologiayritys, joka tarjoaa useita eri IT-palveluita, kuten pilvipalveluita.
IoT	Internet of Things. Internetverkon laajentuminen laitteisiin ja koneisiin, joita voidaan hallita internetverkon yli.
JSON	JavaScript Object Notation. Ohjelmistokielestä riippumaton tietomuoto.
mPOS	Mobile Point of Sale. Älypuhelimilla, tableteilla tai muulla langattomalla laitteella toimiva kassa.
PaaS	Platform as a Service. Palvelusalustan ulkoistaminen.
REST	Representational State Transfer. Arkkitehtuurimalli ohjelmointirajapintojen toteuttamiseen.

SaaS	Software as a Service. Ohjelmiston hankkiminen palveluna.
TCP	Transmission Control Protocol. Tietoliikenneprotokolla, jonka avulla lähetetään tietoa laitteiden välillä.
USB	Universal Serial Bus. Sarjaväyläarkkitehtuuri oheislaitteiden liittämiseksi toiseen laitteeseen.
XML	Extensible Markup Language. Metakieli, jolla määritellään rakenteellisia merkkauskieliä.

1 Johdanto

Tässä insinöörityössä käydään läpi kassaohjelmistojen yleisiä ominaisuuksia, kassaohjelmiston toteuttamista Android-käyttöjärjestelmää käyttäville laitteille sekä sen liittämistä Amazon Web Services -pilvipalveluun (AWS).

Työssä tehty kassaohjelmisto on toteutettu pienkauppiaita varten ja tehty käytettäväksi vain Android-laitteille. Pienkauppiaat ovat yleisesti ottaneet vastaan vain käteismaksuja, ja tarve korttimaksujen vastaanottamiseen on kasvussa. Pienkauppiaat haluavat myös käyttää jo omistamiaan laitteita eivätkä halua ostaa erillistä kassaa. Pienkauppiailla ei myöskään ole mahdollisuutta tehdä omaa kassaa sekä ylläpitää tarvittavia palvelimia.

Android-laitteen lisäksi kassaohjelmisto vaatii toimiakseen maksupäätteen sekä internetyhteyden, jotta sillä pystytään käyttämään kaikkia toteutettuja ominaisuuksia. Maksupääte ja kassaohjelmisto kommunikoivat keskenään joko Bluetooth- tai Transmission Control Protocol -yhteydellä (TCP) riippuen maksupäätteen ominaisuuksista.

Kassaohjelmistoon liittyy myös taustajärjestelmä, joka toimii AWS-pilvipalvelussa. Taustajärjestelmään tallennetaan kassasta tulevia tietoja, kuten käyttäjän tiedot tai kassalla tehdyt tapahtumat. Kassaohjelmisto kommunikoi taustajärjestelmän kanssa Hypertext Transfer Protocol -protokollan (HTTP) avulla ja välittää tietoa JavaScript Object Notation (JSON) -muodossa.

Työn alussa käydään läpi Android-käyttöjärjestelmän ja AWS:n historiaa. Tämän jälkeen tutustutaan kassaohjelmistojen yleisiin ominaisuuksiin. Sen jälkeen käydään läpi oman kassaohjelmiston toteuttamista Android-laitteille. Työssä käydään myös läpi kassaohjelmiston käyttämiä AWS:n palveluita ja näiden hyötyjä. Lopuksi työssä esitellään toteutetun kassaohjelmiston käyttöä sekä sitä, kuinka tarvittavat kassaohjelmiston ominaisuudet on toteutettu.

2 Android-käyttöjärjestelmän historiaa

Android on Linuxiin pohjautuva käyttöjärjestelmä, jonka nykyisestä kehityksestä vastaa Google. Androidia käytetään mm. puhelimissa, tableteissa, rannekelloissa ja jopa autojen stereoissa. Sen kehitys sai alkunsa lokakuussa 2003, kun neljä henkilöä (Rich Miner, Nick Sears, Chris White ja Andy Rubin) perustivat Android Inc. -nimisen yhtiön Palo Altossa Kaliforniassa. [1.]

Alun perin Androidin käyttötarkoitus oli parantaa digitaalisten kameroiden käyttöjärjestelmiä. Vuonna 2004 Android Inc. näytti sijoittajilleen, kuinka digitaalinen kamera voisi ottaa langattomasti yhteyttä tietokoneeseen, jos siihen oli asennettu Android. Kameran omistajat voisivat myös ottaa yhteyden tietokoneellaan Android Datacenter -nimiseen palveluun, johon he pystyisivät tallettamaan valokuviaan. Android Inc. päätti kuitenkin siirtää katseensa matkapuhelinten käyttöjärjestelmiin, sillä digitaalisten kameroiden markkinat olivat laskussa. [1.]

Vuonna 2005 Google osti Android Inc. yhtiön. Tällöin tehtiin myös päätös käyttää Linux-käyttöjärjestelmää Androidin pohjana. Tämä tarkoitti sitä, että Google pystyisi tarjoamaan Android-käyttöjärjestelmää ilmaiseksi muille matkapuhelinten kehittäjille. Google olikin sitä mieltä, että se saisi tehtyä enemmän tuottoa tarjoamalla Androidilla erilaisia sovelluksia ja palveluita sen sijaan, että yrittäisi myydä Android-käyttöjärjestelmää itsessään. [1.]

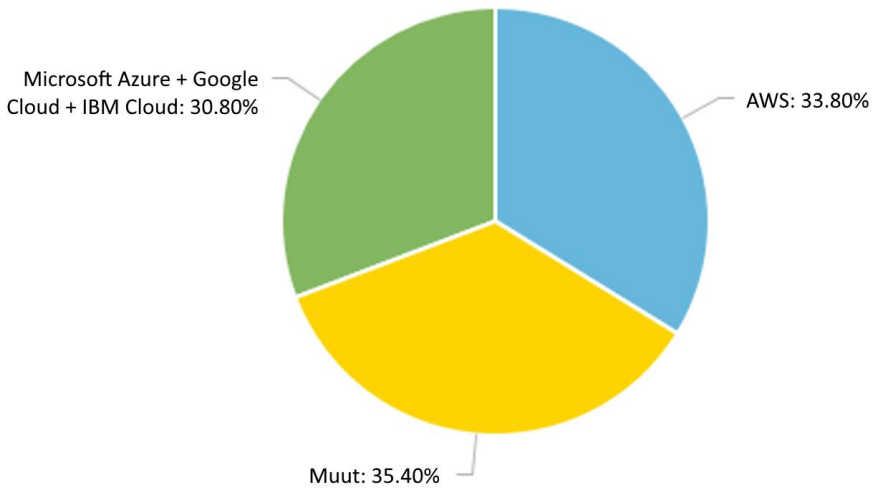
Vuonna 2007 Googlen kilpailija Apple julkaisi ensimmäisen version iPhone-puhelimestaan. Tämä pakotti Googlen vastaamaan tähän omalla julkaisullaan Androidista, jonka ensimmäinen beeta-versio julkaistiin 5. maaliskuuta 2007. Google ei kuitenkaan saanut Android-käyttöjärjestelmää käyttävää puhelinta markkinoille ennen syyskuuta 2008. [1.]

Google jatkaa Androidin kehitystä edelleen ja julkaiseekin usein uusia versioita Android-käyttöjärjestelmästä.

3 Amazon Web Services -pilvipalvelun historiaa

Amazon Web Services on Amazonin tarjoama jatkuvasti kehittyvä pilvipalvelu. AWS tarjoaa runsaasti erilaisia palveluja perinteisten mallien mukaan joko infrastruktuurina (Infrastructure as a Service (IaaS)), sovellusalustana (Platform as a Service (PaaS)) tai ohjelmistona (Software as a Service (SaaS)).

AWS julkaistiin Amazonin toimesta vuonna 2006, ja se on onnistunut muuttamaan koko IT-alan käyttämään enemmän pilvipalveluita kuin ennen. AWS on suurin kaikista IaaS-palveluntarjoajista, mutta joutuu kilpailemaan muiden isojen nimien kanssa, kuten Microsoft Azuren, Google Cloudin ja International Business Machinesin Cloudin (IBM Cloud). Canalys yhtiön mukaan vuonna 2017 AWS:ssää käytti 33,8 % kaikista IaaS-palveluiden käyttäjistä, kun taas yhteensä laskettu luku Microsoft Azuren, Google Cloudin ja IBM Cloudin käyttäjistä oli 30,8 %. Kuvasta 1 näemme myös muiden pilvipalveluiden käyttäjien määrän. Viime vuosina AWS:n kilpailijat ovat kuitenkin lisänneet keskittymistä pilvipalveluihinsa, ja he ovatkin merkittävä uhka kasvavassa markkinassa. [2, s. 1.]



Kuva 1. IaaS-palveluiden käytön jako vuonna 2017

AWS julkaistiin oikeastaan ensimmäisen kerran jo vuonna 2002, jolloin sen tarkoitus oli antaa kehittäjille mahdollisuus sisällyttää Amazon.com -sivun ominaisuuksia omille sivustoilleen. [2, s. 1.]

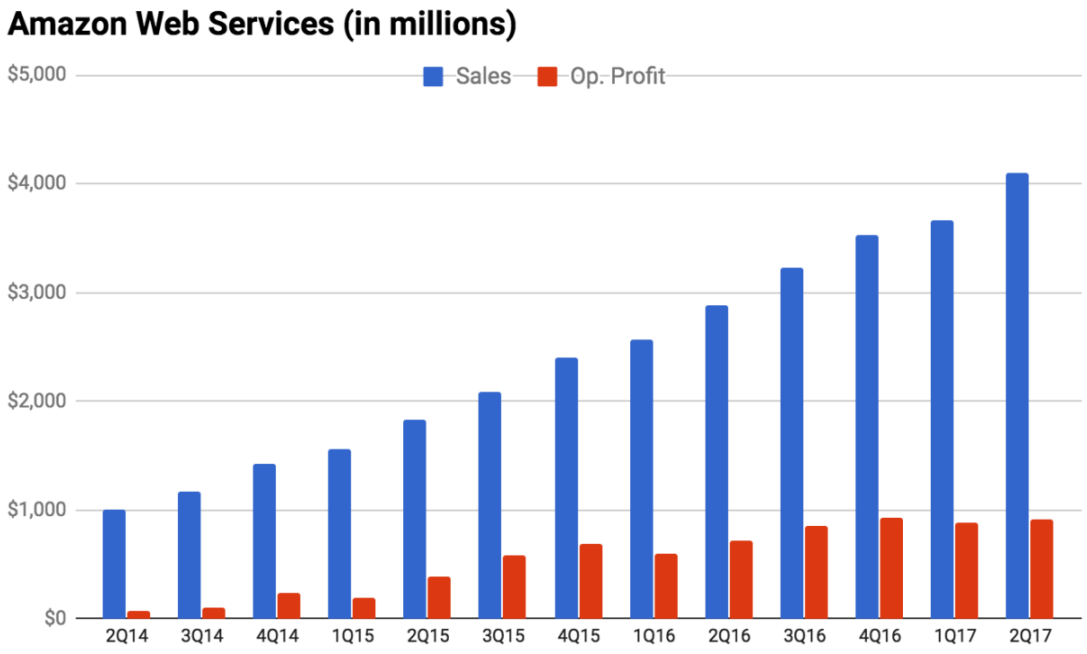
Vuonna 2006 AWS julkaisi ensimmäiset versiot pilvipalveluistaan, joka mahdollisti kehittäjien tehdä muitakin sovelluksia käyttäen Amazonin infrastruktuuria. Ensimmäiseksi AWS julkaisi S3-palvelun, joka mahdollistaa tiedostojen tallentamisen pilveen. Tämän jälkeen julkaistiin EC2-palvelu, joka on palvelinten vuokraus- ja hosting-palvelu. Vuosien varrella EC2-palveluun on lisätty erilaisia instanssivaihtoehtoja, joita valitsemalla voi saada esim. eri määrän muistia tai levytilaa. Molemmat S3- ja EC2-palvelut ovat suosittuja AWS:n tarjoamia palveluita vielä tänäkin päivänä. [2, s. 2.]

Kaksi vuotta myöhemmin vuonna 2008 kilpailu kasvoi Googlen liityttyä markkinaan mukaan tarjoamalla omia pilvipalveluitaan käyttäjille. Kilpailu jatkoi kasvuaan vuonna 2009 Microsoftin liityttyä mukaan Azure-pilvipalvelullaan. Azure on nykyään suurin AWS:n kilpailija. [2, s. 3.]

Vuonna 2012 AWS piti ensimmäisen kehittäjäkonferenssin Las Vegasissa, jossa haluttiin korostaa AWS:n kasvavaa yhteisöä. Konferenssia pidetään nykyään vuosittain ja siihen rekisteröityy jopa 30000 henkilöä. [2, s. 5.]

AWS ei yleensä rakenna asiakkailleen omia infrastruktuureja, mutta vuonna 2013 AWS oli halukas tekemään yksityisen pilvisopimuksen Central Intelligence Agencyn (CIA) kanssa. Tämä sopimus oli noin 600 miljoonan dollarin arvoinen. AWS joutui kilpailemaan tästä sopimuksesta IBM-yhtiön kanssa. Sopimuksen solmiminen vahvisti AWS:n asemaa kilpailijoiden keskuudessa, ja IBM väittikin, että Amazonilla ei ole vahvaa historiaa onnistuneista muutosprojekteista Yhdysvaltojen hallitusten kanssa, kuten heillä itsellään. [2, s. 6.]

Vuonna 2015 Amazon julkaisi ensimmäisen kerran tiedot siitä, kuinka paljon AWS toi heidän yhtiöllensä liikevaihtoa. Vuonna 2014 AWS saavutti 4,6 miljardin dollarin liikevaihdon ja odotti luvun kasvavan jopa 49 prosentilla vuonna 2015, joka tarkoittaisi 6,2 miljardin dollarin liikevaihtoa. Vuoden 2016 alussa Amazonin toimitusjohtaja Jeff Bezos lupasi, että AWS:n liikevoitto kasvaisi 10 miljardiin dollariin nopeammin kuin koskaan. Vuoden loppuun mennessä AWS:n liikevoitto olikin kasvanut jopa 12,2 miljardiin dollariin. Kuvasta 2 näemme AWS:n liikevaihdon vuosien 2014 ja 2017 välillä. [2, s. 8.]



Kuva 2. AWS:n liikevaihto (miljoonissa dollareissa) [3]

AWS:llä on aina ollut suuri määrä datakeskuksia ympäri maapalloa, mutta vuonna 2016 he pyrkivät tarjoamaan enemmän alueellisia datakeskuksia, sillä asiakkaat ovat enemmän huolissaan oman datansa turvallisuudesta. Joulukuussa 2016 he julkaisivatkin Ison-Britannian alueella sijaitsevan datakeskuksen. Kilpailijat kuten Microsoft ja Google seurasivat nopeasti ja julkaisivat omia datakeskuksiaan samoilla alueilla. [2, s. 10.]

Nykyään AWS tarjoaa melkein jopa 100 erilaista pilvipalvelua asiakkailleen. Palvelut ovat monimuotoisia ja vaihtelevatkin paljon. AWS tarjoaa mm. seuraavia palveluita:

- tietojenkäsittelypalvelut
- tiedontallennuspalvelut
- analytiikkapalvelut
- verkkopalvelut
- mobiilipalvelut
- kehitystyökaluja
- hallintatyökaluja
- IoT-palvelut
- tietoturvapalveluita.

Nykyään AWS on keskittynyt enemmän palvelimattomaan tietojenkäsittelymalliin sekä koneoppimisen työkaluihin. [2, s. 12.]

Palvelimettomassa tietojenkäsittelymallissa ei nimensä mukaan käytetä mitään tiettyä palvelinta pyörittämään sovellusta, jolla olisi staattiset resurssit käytettävänä vaan palvelimia käynnistetään sitä mukaan, miten sovellus vastaanottaa kyselyitä. Palvelimaton tietojenkäsittelymalli helpottaa kehittäjiä huomattavasti, sillä heidän ei tarvitse pitää huolta palvelimesta itsestään vaan pilvipalvelun tarjoaja hoitaa kaiken taustalla. Tämä helpottaa mm. skaalautuvuuden hallintaa, sillä palvelittomassa mallissa voidaan esim. ruuhka-aikoina pyörittää useita eri palvelimia vastaamaan kyselyihin. Palvelimattomassa tietojenkäsittelymallissa pilvipalvelun tarjoaja laskuttaa sen käyttäjää kyselyiden määrän perusteella. [4.]

Palvelimattomassa tietojenkäsittelymallissa on kuitenkin myös huonot puolensa. Jos sovellus ei ole vastaanottanut kyselyitä vähään aikaan, saattaa palvelimen käynnistäminen kestää joskus jopa 10-15 sekuntia. Tällöin puhutaan ns. kylmäkäynnistysilmiöstä. Kehittäessä sovelluksia palvelimattomaan tietojenkäsittelymalliin on tämä otettava usein huomioon. Kehittäjät joutuvat myös kirjoittamaan sovelluksensa siten, että he eivät voi luottaa sovelluksen tilaan, sillä se voi sovelluksen ajon aikana vaihtua. Palvelimattomassa tietojenkäsittelymallissa kehittäjillä ei myöskään ole täyttä kontrollia omasta ohjelmistosta, kuten heillä olisi oman palvelimen kanssa. Tällöin kehittäjät ovat lukinneet itsensä käyttämään tietyn pilvipalvelun tarjoajan palveluita. [4.]

4 Kassaohjelmistojen ominaisuudet

Google Play Kaupasta löytyy useita erilaisia kassaohjelmistoja etsimällä niitä hakusanalla ”mPOS” (Mobile Point of Sale). Kaikki kassaohjelmistot näyttävät ulkoapäin hieman erilaisilta, mutta niiden sisältämät perusominaisuudet ovat periaatteessa kaikissa samat. Jotkut saattavat tosin olla helppokäyttöisempiä, ja niillä on vähemmän ominaisuuksia, mutta toiset ovat taas vaikeakäyttöisempiä ja rikkaampia ominaisuuksiltaan.

4.1 Perusominaisuudet

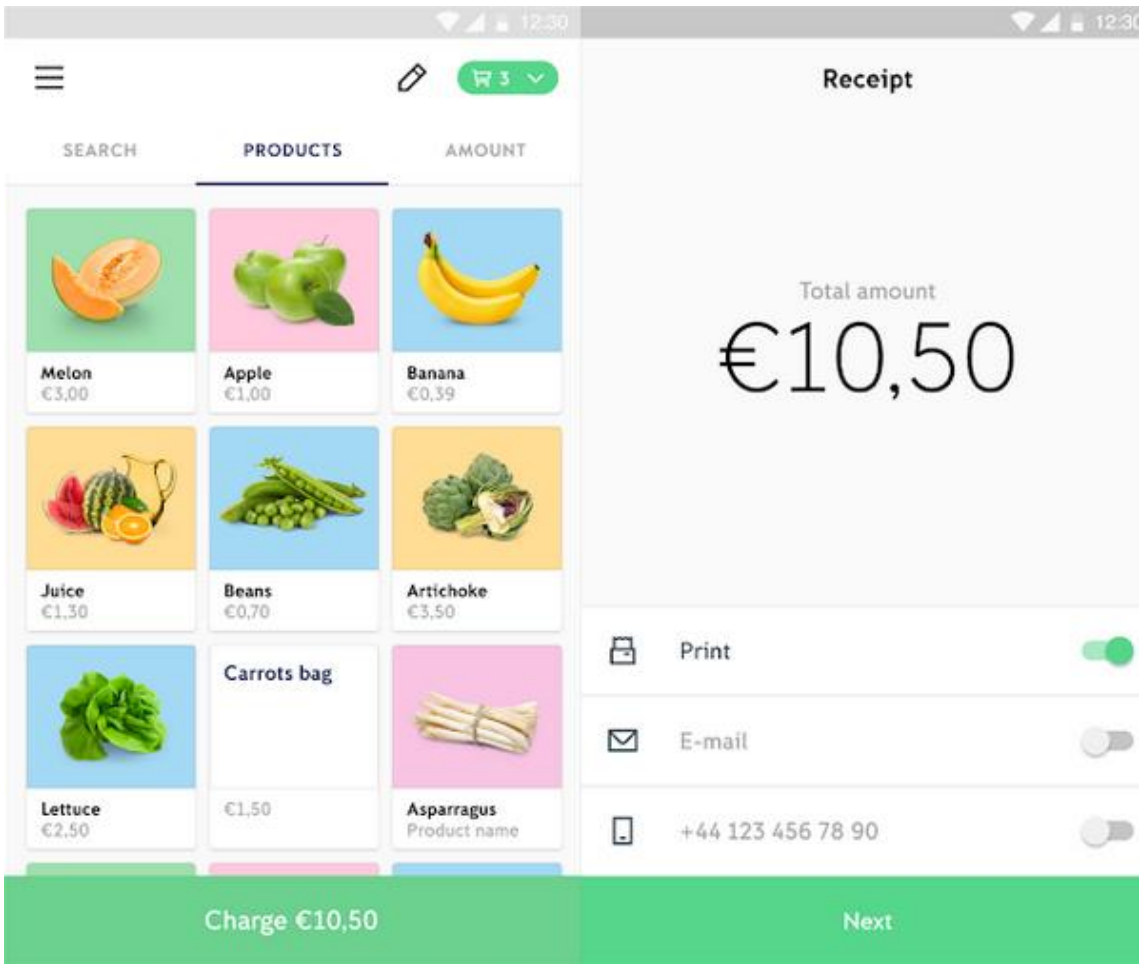
Kaikissa kassaohjelmistoissa on perusominaisuuksia, jotka määrittelevät sen kassaohjelmistoksi. Nämä ominaisuudet ovat jokaisessa kassaohjelmistossa samat, mutta niiden ulkonäkö saattaa vaihdella. Esimerkiksi kuvassa 3 nähdään iZettle Go -kassaohjelmiston päänäkymän ulkonäkö.

Jokaisessa kassaohjelmistossa on mahdollisuus suorittaa ostotapahtumia syöttämällä manuaalinen hinta. Tällöin joko lisätään manuaalinen tuote ostoskoriin, jos sellainen on olemassa, tai aloitetaan ostotapahtuma sillä hinnalla, mikä on syötetty sovellukselle.

Ostoprosessi on kaikissa kassaohjelmistoissa, mutta joissakin hyväksytään vain käteismaksut. Toisissa taas voidaan hyväksyä tämän lisäksi myös korttimaksut. Jos korttimaksut hyväksytään, tarvitaan kassaohjelmiston lisäksi jonkinlainen maksupääte.

Ostotapahtuman jälkeen myyjälle ja asiakkaalle on tarjottava kuittia. Jos kassaohjelmistolla on lisänä maksupääte, voidaan se tulostaa sen avulla. Jos maksupäätettä ei ole, näytetään kuitti kassaohjelmiston näytöllä tai se voidaan lähettää sähköpostitse tai tekstiviestitse.

Näiden ominaisuuksien lisäksi kaikissa kassaohjelmistoissa on tapahtumahistoria, josta voidaan nähdä vanhat ostotapahtumat. Lisäksi voidaan myös listata hyvitys- ja peruutustapahtumat, jos kassaohjelmisto tukee niitä. Tapahtumista voidaan nähdä ainakin oston hinta ja ajankohta, mutta riippuen kassaohjelmistosta voi tapahtumasta nähdä myös lisätietoja esim. tuotteet tai kortin tyypin, jolla osto on maksettu.



Kuva 3. iZettle Go -kassaohjelmiston päänäkymä ja kuitin tulostamisnäkymä. [5]

4.2 Lisäominaisuudet

Joissakin kassaohjelmistoissa on lisäominaisuuksia, jotka tuovat käyttäjälle enemmän valinnanvaraa ohjelmiston käytössä. Näitä ominaisuuksia ei löydy jokaisesta kassaohjelmistosta, mutta monet ovat kuitenkin ottaneet niitä käyttöön.

Useimmissa kassaohjelmistoissa ensimmäinen ominaisuus, mikä tulee vastaan, on sisäänkirjautuminen. Tämän monet tekijät ovat ratkoneet samalla tavalla, jossa käyttäjä rekisteröityy kassaohjelmistoon joko sähköpostilla tai puhelinnumerolla ja liittää tunnukseensa salasanan. Joskus kassaohjelmiston kirjautumiseen tarvitaan vain aktivoimiskoodi, joka toimii tunnuksena. Jotkut kassaohjelmistojen tekijät eivät kuitenkaan vaadi sisäänkirjautumista. Tällöin kassaohjelmistoon tallennettavat asetukset ja ostotapahtumat ovat vain yhdellä laitteella käytössä eikä niitä ole tallennettu taustajärjestelmässä toimivaan tietokantaan.

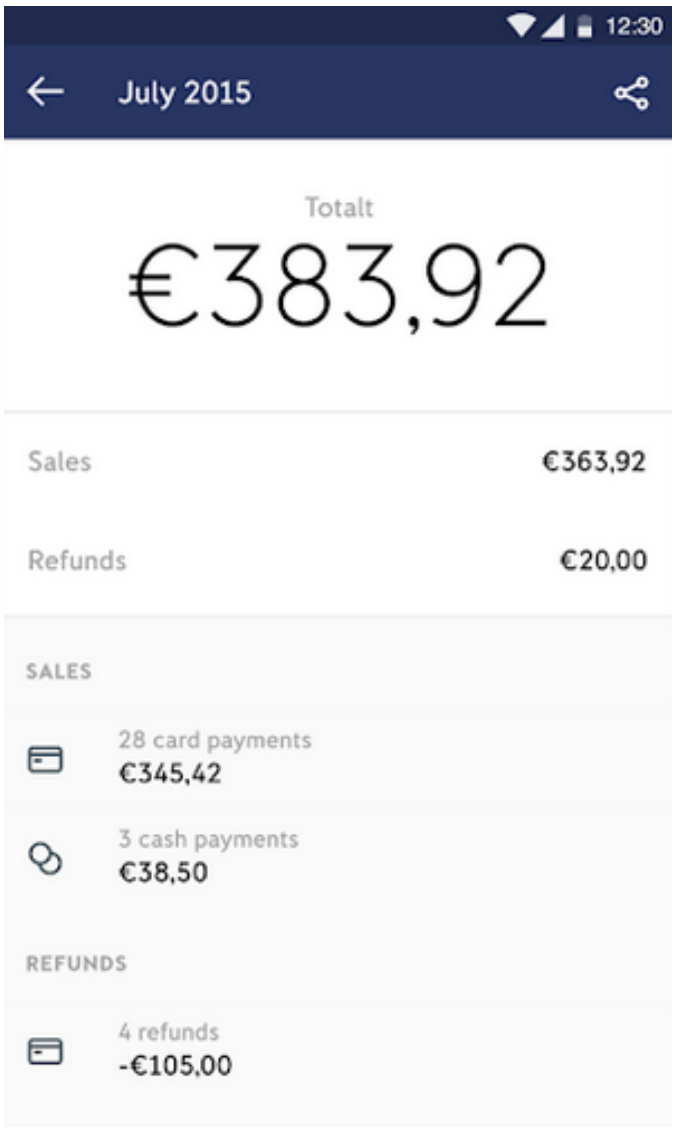
Kassaohjelmistoissa on usein jonkinnäköinen tuotelista, johon voi lisätä omia tuotteita. Näitä tuotteita voi myös poistaa tai muokata. Tuotteiden lisäksi on usein ostoskori, johon tuotteita voi lisätä ennen kuin aloitetaan suorittamaan ostoprosessia.

Jotkin kortilla tehdyt maksut saattavat vaatia kortinhaltijaa allekirjoittamaan myyjän kuitin. Jos kassaohjelmiston käyttäjällä on kuitenkin käytössä maksupääte, jossa ei ole tulostinta, ei kuittia ole mahdollista allekirjoittaa. Tällöin kassaohjelmisto voi pyytää kortinhaltijaa allekirjoittamaan kuitin maksupäätteen näytöllä.

Jos kassaohjelmistolla on käytössä maksupääte, voi ohjelmistolla olla ominaisuus hyvittää tai peruuttaa maksuja. Peruuttaminen on mahdollista vain hetken ostotapahtuman jälkeen, kun taas hyvityksen voi tehdä milloin vain. Näistä tapahtumista myyjä sekä asiakas tarvitsevat kuitin niin kuin ostotapahtumasta.

Joissakin kassaohjelmistoissa voi olla käyttäjiä, joilla on erilaiset käyttöoikeudet. Pääkäyttäjä voi esimerkiksi tehdä alikäyttäjiä, joilla on rajatut oikeudet. Nämä käyttäjät jakavat tuotelistat ja asetukset muuten pääkäyttäjän kanssa, mutta niillä voi olla esimerkiksi vain oikeus tehdä ostotapahtumia.

Lisäominaisuuksia voi olla näiden lisäksi myös muita, esimerkiksi kuvassa 4 nähdään iZettle Go -kassaohjelman kuukausittainen raporttinäkymä.



Kuva 4. iZettle Go -kassaohjelmiston kuukausittaisen myynnin raporttinäkymä. [5]

5 Kassaohjelmiston kehittäminen

Tässä työssä tehtyä kassaohjelmistoa lähdettiin jatkokehittämään jo olemassa-olevasta kassaohjelmistosta. Kassaohjelmistoon haluttavia ominaisuuksia kehitetään ja parannetaan jatkuvasti.

5.1 Android-sovelluskehitys

Android-sovellukset rakentuvat erilaisista komponenteista. Sovelluksen voi tehdä käyttämällä vain yhtä tiettyä komponenttia, mutta sovellus voi myös rakentua näiden komponenttien yhdistelmästä. Komponentteja on neljää erilaista:

- Activity
- Service
- Broadcast Receiver
- Content Provider.

Jokaisella komponentilla on oma tarkoituksensa, ja niiden elinkaari eroaa toisistaan siten, miten ne luodaan ja tuhotaan. [6.]

Activity-komponentit ovat näkymiä, jotka ovat vuorovaikutuksessa käyttäjän kanssa. Sitä edustaa yksi näkymä, jossa on käyttöliittymä. Tässä työssä aktiviteetteja käytetään mm. kirjautumisnäkymässä, päänäkymässä ja tuotelistanäkymässä. Aktiviteetin käyttöliittymä luodaan Extensible Markup Language -tiedostolla (XML). [6.]

Aktiviteetteja voidaan kutsua myös sovelluksen ulkopuolelta. Esimerkiksi jokin kamerasovellus voisi kutsua sähköpostisovelluksen aktiviteettia, jos halutaan lähettää kuva sähköpostilla. [6.]

Service on komponentti, jota voidaan ajaa sovelluksen taustalla. Tällä komponentilla ei ole käyttöliittymää, minkä kanssa se voisi olla vuorovaikutuksessa käyttäjän kanssa. Service-komponenttia voidaan käyttää esimerkiksi tekemällä verkkokyselyitä tai soittamalla musiikkia taustalla, kun käyttäjä käyttää toista sovellusta. Jokin toinen komponentti, kuten aktiviteetti, voi luoda palvelukomponentin ja olla vuorovaikutuksessa sen kanssa. Tässä työssä tehdyssä kassaohjelmistossa ei käytetty Androidin Service-komponentteja. [6.]

Broadcast Receiver -komponentti mahdollistaa vastaamisen järjestelmän tuottamiin viesteihin. Esimerkkinä järjestelmän viesteistä on mm. laitteen käynnistyminen tai akun loppuminen. Broadcast Receiver -komponenttiin ei liity käyttöliittymää. Nämä komponentit voivat myös vastaanottaa järjestelmän viestejä, vaikka itse sovellus ei olisi käynnissä. [6.]

```
public class BootUtil extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        if (Intent.ACTION_BOOT_COMPLETED.equals(intent.getAction())) {
            Intent bootIntent = new Intent(context, SigninActivity.class);
            bootIntent.addFlags(Intent.FLAG_ACTIVITY_NEW_TASK);
            context.startActivity(bootIntent);
        }
    }
}
```

Esimerkkikoodi 1. Kassaohjelmistossa käytetty Broadcast Receiver -komponentti

```
<receiver android:name=".util.BootUtil">
     <intent-filter>
          <action android:name="android.intent.action.BOOT_COMPLETED"/>
     </intent-filter>
</receiver>
```

Esimerkkikoodi 2. BOOT_COMPLETED -viestin vastaanottaminen

Tässä työssä Broadcast Receiver -komponenttia käytettiin ainoastaan käynnistämään kassa kirjautumisnäkymään automaattisesti laitteen käynnistyttyä. Komponentti oli tehty Esimerkkikoodin 1 mukaisesti. Järjestelmän käynnistysviestin vastaanotto määriteltiin AndroidManifest.xml-tiedostossa esimerkkikoodin 2 mukaisesti. Esimerkkikoodissa 2 nähdään, että kun sovellus vastaanottaa BOOT_COMPLETED -viestin Androidilta, käynnistää se BootUtil-luokan.

Content Provider on komponentti, joka hallinnoi sovelluksen tietoja. Content Provider mahdollistaa sovelluksen kommunikaation sen tietokannan kanssa. Sen hallinnoima tieto voi sijaita missä tahansa, mihin sovelluksella on oikeus, esimerkiksi SQLite-tietokannassa tai internetissä. Mikä tahansa sovellus voi kutsua muiden sovelluksien Content Provider -komponentteja ja hallita niiden tietokantaa, kunhan niillä on tarvittavat oikeudet. Tässä työssä tehdyssä kassaohjelmistossa ei käytetty Androidin Content Provider -komponentteja. [6.]

Näiden peruskomponenttien lisäksi Androidilla on myös useita pienempiä komponentteja, joista sovellus voi koostua. Esimerkkinä on kassaohjelmistossa käytetyt DialogFragment-komponentit, jotka ovat erikoisempia Fragment-komponentteja. Fragment on osa sovelluksen käyttöliittymää tai käyttäytymistä, jonka voi sijoittaa aktiviteetin sisälle. DialogFragment esittää käyttöliittymässä pienen dialogi-ikkunan, jonka kanssa käyttäjä voi olla vuorovaikutuksessa. Kuvassa 11 nähdään esimerkki DialogFragmentista. [7; 8.]

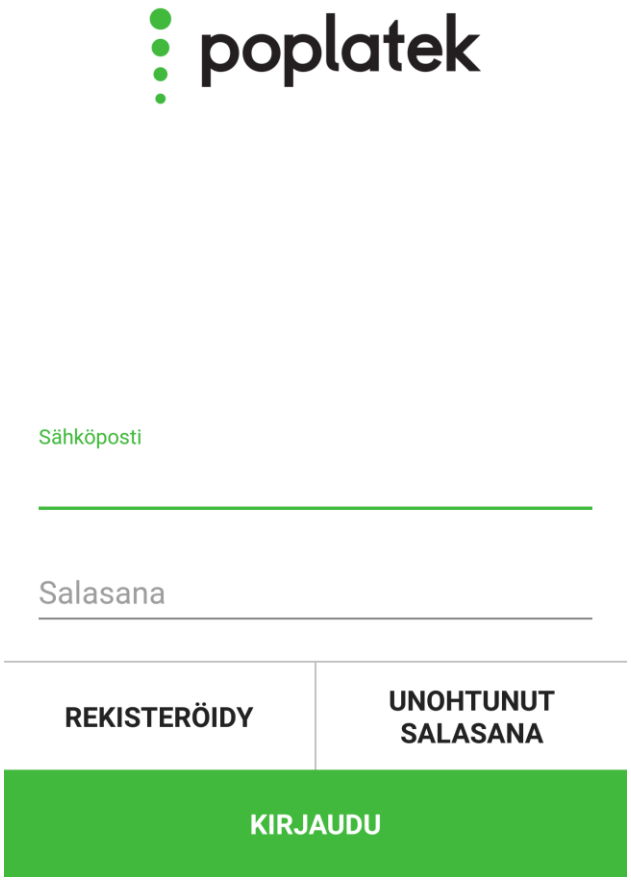
Androidille voidaan kehittää sovelluksia perinteisesti Java- ja C++-ohjelmointikielillä, mutta nykyään sovelluksia voidaan myös kehittää uudemmalla Kotlin-ohjelmointikielellä. Kotlin-tuki Androidille julkaistiin virallisesti vuonna 2017. Java on suosituin ohjelmointikieli Android-sovellusten kehittämisessä. Kotlin on ohjelmointikieli, jota käytetään kehittäessä Androidille sovelluksia. Monet Android-sovellusten kehittäjät ovatkin siirtyneet kirjoittamaan koodia Kotlinilla Javan sijaan. Kotlinia voi kuitenkin myös käyttää Javan ja C++:n kanssa samanaikaisesti eikä jo olemassa olevia sovelluksia täydy kokonaan uudelleen kirjoittaa Kotlin pohjalle. Analytiikkayhtiö Realm ennustaa, että Kotlinia käytettäisiin Javaa enemmän Android-sovellusten tekemisessä vuoteen 2018 mennessä. Tässä työssä tehtyä kassaohjelmaa kehitettiin Java-ohjelmointikielellä. [9; 10.]

5.1.1 Kassaohjelmiston Android-sovelluksen perusrakenne

Tässä työssä tehty kassaohjelmiston Android-sovellus rakentuu seuraavista osista:

- sisäänkirjautumisnäkymä
- rekisteröintinäkymä
- unohtunut salasana -näkymä
- asetukset-näkymä
- päänäkymä, jossa voi lisätä tuotteita ostoskoriin ja tehdä ostoja
- maksuhistorianäkymä
- tuotekatalogin muokkausnäkymä.

Näiden päänäkymien lisäksi ohjelmiston taustalla toimii säikeitä, jotka mm. pitävät yhteyttä yllä maksupäätteeseen sekä kommunikoivat taustajärjestelmän kanssa. Kuvassa 5 näemme kassaohjelmiston Android-sovelluksen sisäänkirjautumisnäkymän.



Kuva 5. Kassaohjelmiston sisäänkirjautumisnäkymä.

Kassaohjelmiston Android-sovellus koostuu pääosin erilaisista Androidin Activity-, Fragment- ja DialogFragment-komponenteista. Kun käyttäjä avaa sovelluksen, käynnistetään BaseActivity-luokka, jota käyttää kaikki sovelluksen aktiviteetit. BaseActivity-luokka välittää tietoja mm. yhteydestä maksupäätteeseen muille aktiviteeteille. Esimerkkikoodista 3 nähdään BaseActivityn onCreate()-funktio, jossa asetetaan erilaisia arvoja käytettäväksi muualla sovelluksessa.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
    getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN|WindowMan-
ager.LayoutParams.SOFT_INPUT_STATE_ALWAYS_HIDDEN,
            WindowManager.LayoutParams.FLAG_FULLSCREEN);
    getWindow().getDecorView().getRootView().setSystemUiVisibility(View.SYS-
TEM_UI_FLAG_HIDE_NAVIGATION|View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY);

    if (getResources().getBoolean(R.bool.is_portrait_layout)) {
        mIsPortraitLayout = true;
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_PORTRAIT);
    } else {
        setRequestedOrientation(ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE);
    }

    mPtManager = PaymentTerminalManager.INSTANCE;
    mPtManager.setListener(this);
    mHttpManager = HttpManager.INSTANCE;
    mHttpManager.setListener(this);
    mMposDbManager = MposDbManager.INSTANCE;
    mMposDbManager.setListener(this);
    mMposDbHelper = MposDbHelper.getInstance(getApplicationContext());
    mTransactionUploadService = TransactionUploadService.getInstance(getAppli-
cationContext());

    mReconnectingDialog = new AlertDialog.Builder(this)
            .setTitle(getString(R.string.error_title_connection_lost))
            .setMessage(getString(R.string.label_connection_reconnecting))
            .setIcon(android.R.drawable.ic_dialog_alert)
            .setPositiveButton(R.string.label_cancel, new DialogInterface.On-
ClickListener() {
                public void onClick(DialogInterface dialog, int which) {
                    disconnect();
                }
            })
            .setCancelable(false)
            .create();
}
```

Esimerkkikoodi 3. BaseActivity-luokan onCreate()-funktio.

BaseActivity-luokassa asetetut arvot ovat:

- mPtManager hoitaa sovelluksen yhteyden maksupäätteeseen.
- mHttpManager hoitaa pyynnöt taustajärjestelmän Representational State Transfer -rajapintaan (REST).
- mMposDbManager ja mMposDbHelper hoitavat sovelluksen lokaalin tietokannan.
- mTransactionUploadService hoitaa lokaaliin tietokantaan tallennettujen transaktioiden viennin AWS-pilvipalvelussa olevaan tietokantaan.

```
public enum PaymentTerminalManager implements Connection.Listener {
    INSTANCE;

    public static Locale POS_LOCALE;
    private Connection mConnection;

    public void setListener(Context context) {
        mListener = (Listener) context;
        POS_LOCALE = context.getResources().getConfiguration().locale;
    }

    public void createTCPConnection(String address) {
        if (!isConnecting() && !isConnectionAlive()) {
            mConnection = new TCPConnection(this, address);
        }
    }

    public void createBluetoothConnection(BluetoothDevice device) {
        if (!isConnecting() && !isConnectionAlive()) {
            mConnection = new BluetoothConnection(this, device);
        }
    }

    public void closeConnection(CloseReason closeReason) {
        if (isConnecting() || isConnectionAlive()) {
            mConnection.close(closeReason);
        }
    }
}
```

Esimerkkikoodi 4. PaymentTerminalManagerin maksupääteyhteyden muodostamiseen liittyvät funktiot.

Sovelluksessa on useita eri aktiviteetteja. Aktiviteetit ovat usein eri näkymiä, mutta niitä käytetään sovelluksessa myös muiden aktiviteettien apuna. Esimerkiksi sovelluksen päänäkymä käyttää MainActivity-aktiviteettia, joka käyttää apunaan BasePurchaseActivity-aktiviteettia. Aktiviteettien jakaminen moneen eri osaan helpottaa sovelluksen kehittämistä.

Päänäkymässä voidaan aloittaa ostosprosessi, joka avaa DialogFragment-komponentin päänäkymän päälle. Tämän komponentin avaaminen nähdään esimerkkikoodissa 5. Avattu dialogi nähdään kuvassa 11. Kaikkien sovelluksessa käytettävien DialogFragment-komponenttien avaaminen tapahtuu samalla tavalla kuin esimerkkikoodissa 5.

```
protected void startPaymentDialog(boolean continueActivePayment) {
    int amount = continueActivePayment ? Transaction.getTotalAmount(mActive-
Transaction.purchaseItems) : mShoppingCart.getTotalPrice();
    Currency currency = Currency.getValue(continueActivePayment ? mActive-
Transaction.currency : mCurrency);

    FragmentTransaction transaction = getSupportFragmentManager().beginTrans-
action();
    Fragment prev = getSupportFragmentManager().findFragmentByTag(Pay-
mentDialogFragment.TAG);

    if (prev != null) {
        transaction.remove(prev).commit();
```

```
        transaction = getSupportFragmentManager().beginTransaction();
    }

    mPaymentDialogFragment = PaymentDialogFragment.newInstance(amount, cur-
rency, isConnectionAlive(), continueActivePayment);
    mPaymentDialogFragment.show(transaction, PaymentDialogFragment.TAG);
}
```

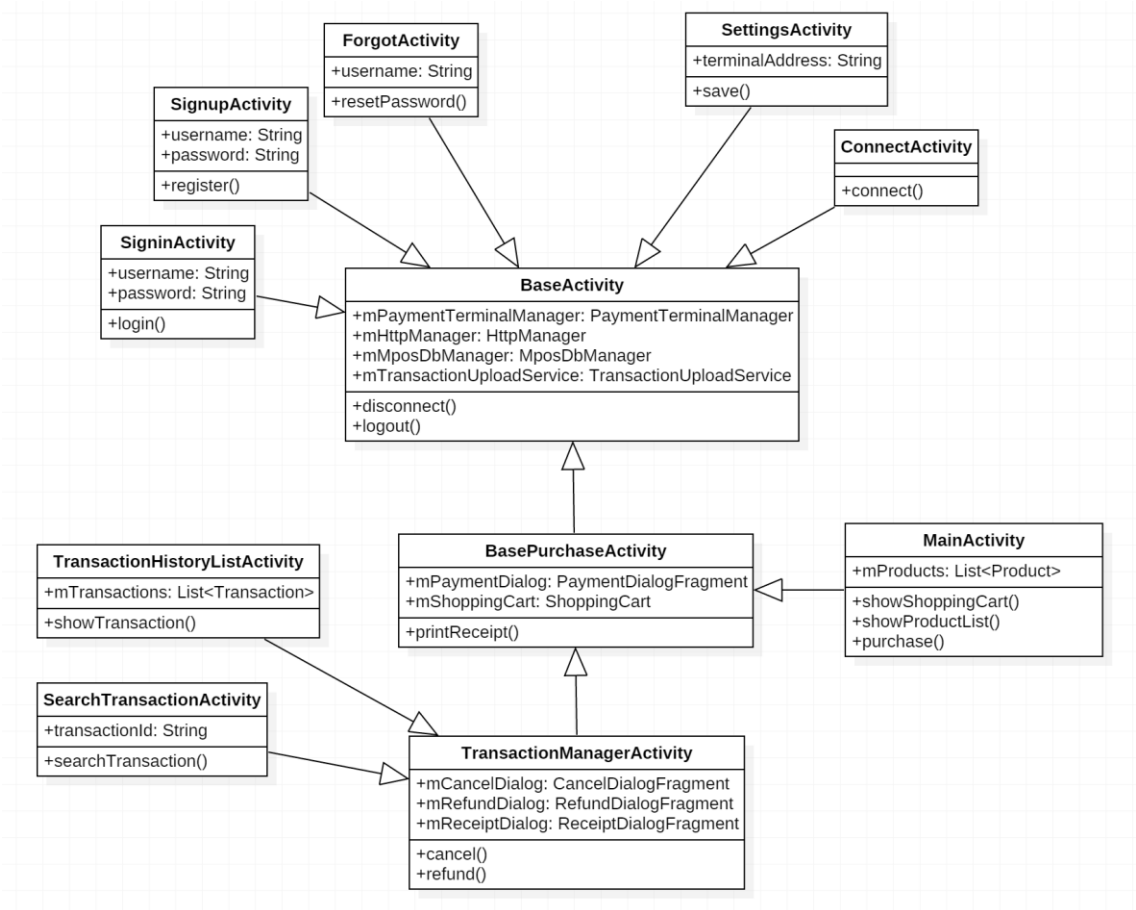
Esimerkkikoodi 5. Ostoprosessissa käytettävän DialogFragment-komponentin avaaminen.

Ostoprosessin toteuttamiseen käyttöliittymässä tarvitaan siis seuraavat komponentit:

- BaseActivity
- BasePurchaseActivity
- MainActivity
- PaymentDialogFragment.

BaseActivity on pohjimmainen komponentti ja PaymentDialogFragment päällimmäinen komponentti. Kaikki sovelluksen näkymät rakentuvat samankaltaisesti aktiviteeteista ja mahdollisesta DialogFragment-komponentista:

- BaseActivity
- mahdolliset apuaktiviteetit
- näkymän oma aktiviteetti
- mahdollinen DialogFragment-komponentti.



Kuva 6. Kassaohjelmiston yksinkertaistettu rakenne.

Kuvassa 6 nähdään kassaohjelmiston yksinkertaistettu rakenne luokkakaavion avulla. Ylhäällä vasemmalla näemme aktiviteetit, joihin on pääsy ennen sisäänkirjautumista. Käyttäjä voi rekisteröityä SignupActivityssä, kysyä unohdetun salasanan palautusta ForgotActivityssä sekä kirjautua sisään SigninActivityssä. Sisäänkirjautumisen jälkeen avataan ConnectActivity, josta voi joko avata SettingsActivityn tai ottaa yhteys maksupäätteeseen. SettingsActivityssä voi käyttäjä muokata ja tallettaa eri asetuksia, kuten maksupäätteen osoitteen. Kun yhteys on saatu maksupäätteeseen, avataan MainActivity, joka koostuu BasePurchaseActivitystä. Tässä näkymässä voidaan hallinnoida tuotteita sekä tehdä ostoja. BaseActivityssä oleva mTransactionUploadService yrittää viedä 5 minuutin välein lokaaliin tietokantaan jääneitä transaktioita AWS:ssä sijaitsevaan tietokantaan.

Kuvassa 6 näkyvässä TransactionHistoryListActivityssä listataan kaikki käyttäjän transaktiot, joista voidaan tehdä hyvityksiä ja palautuksia. Lisäksi transaktioita voidaan

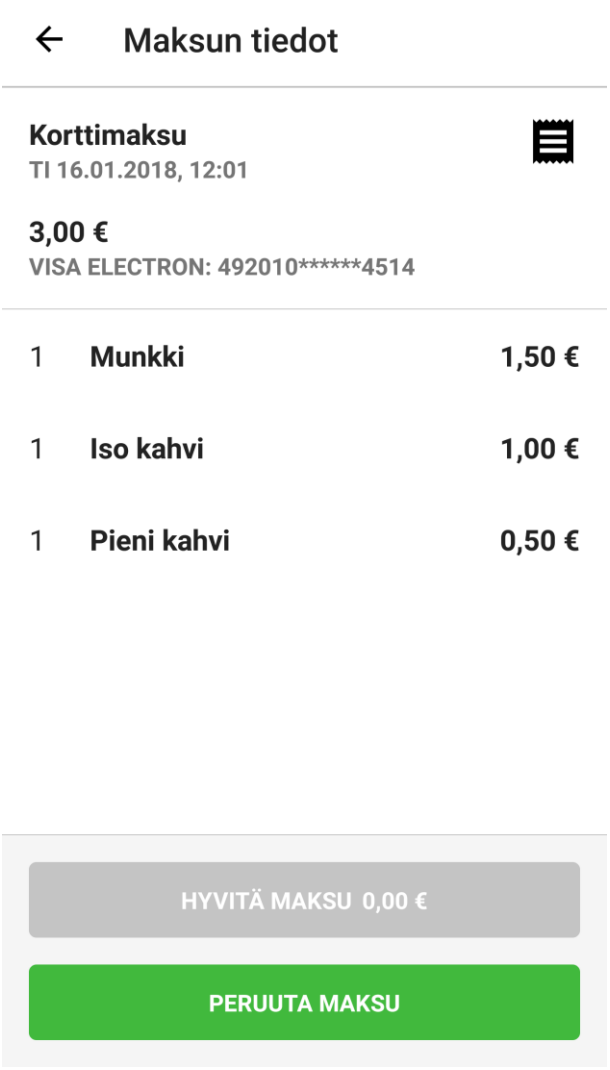
myös etsiä sen tunnisteen avulla SearchTransactionActivityssä. Nämä aktiviteetit koostuvat TransactionManagerActivityssä, joka hoitaa hyvityksen ja palautuksen DialogFragment -komponentit.

Transaktiovarmuuden varmistamiseksi kaikki ostokset tallennetaan lokaaliin SQLite-tietokantaan. Kun ostot aloitetaan maksupäätteessä, joudumme ottamaan huomioon mahdollisen yhteyden katkeamisen, jolloin lokaaliin tietokantaan tallennetaan keskeneräinen transaktio aloitetusta ostosta. Jos yhteys katkeaa ja saamme sen muodostettua uudelleen, voimme kysyä maksupäätteeltä kesken jääneen transaktion tilaa. Transaktio voi olla joko edelleen kesken, onnistunut tai keskeytetty.

```
public void insertActiveTransaction(Context context, Transaction transaction)
{
    MposDbHelper mposDbHelper = MposDbHelper.getInstance(context);
    SQLiteDatabase db = mposDbHelper.getWritableDatabase();

    Gson gson = new Gson();
    String externalData = gson.toJson(transaction.externalData);
    String purchaseItems = gson.toJson(transaction.purchaseItems);

    ContentValues values = new ContentValues();
    values.put(MposContract.ActiveEntry.COLUMN_NAME_TERMINAL_ID, transac-
tion.terminalId);
    values.put(MposContract.ActiveEntry.COLUMN_NAME_TRANSACTION_TYPE, transac-
tion.transactionType.name());
    values.put(MposContract.ActiveEntry.COLUMN_NAME_PAYMENT_METHOD_TYPE,
transaction.paymentMethodType.name());
    values.put(MposContract.ActiveEntry.COLUMN_NAME_EXTERNAL_DATA, exter-
nalData);
    values.put(MposContract.ActiveEntry.COLUMN_NAME_CURRENCY, transaction.cur-
rency);
    values.put(MposContract.ActiveEntry.COLUMN_NAME_PURCHASE_ITEMS, pur-
chaseItems);
    values.put(MposContract.ActiveEntry.COLUMN_NAME_CUSTOMER_RECEIPT, transac-
tion.customerReceipt);
    values.put(MposContract.ActiveEntry.COLUMN_NAME_MERCHANT_RECEIPT, transac-
tion.merchantReceipt);

    Log.d(TAG, "Inserting active transaction to local db...");
    db.insert(MposContract.ActiveEntry.TABLE_NAME, null, values);
}
```

Esimerkkikoodi 6. Keskeneräisen transaktion tallettaminen lokaaliin tietokantaan MposDbManager-luokan avulla.

Onnistuneet maksut tallennetaan ensin lokaaliin tietokantaan, ennen kuin ne viedään AWS:ssä sijaitsevaan tietokantaan. Tällöin varmistetaan transaktioiden tallennus. Kun olemme varmoja, että transaktiot on tallennettu AWS:ssä sijaitsevaan tietokantaan, voimme poistaa ne lokaalista tietokannasta. Kuvasta 7 näemme onnistuneen maksun tiedot. Maksun tiedot haetaan AWS:ssä sijaitsevasta tietokannasta.



Kuva 7. Onnistunut maksu, joka nähdään maksuhistorianäkymässä.

Kassaohjelmistoa kehitettiin 2-4 hengen ryhmässä noin 7 kuukauden ajan. Insinöörityön tekijä kehitti kassaohjelmistoon uusia ominaisuuksia, pääasiassa Android-sovellukseen. Näihin ominaisuuksiin kuului rekisteröinti ja sisäänkirjautuminen ja sen yhdistäminen AWS-pilvipalvelussa toimivaan Cognito-palveluun. Työn tekijä oli myös vastuussa tuotteiden muokkaamisesta, osto-, hyvitys- ja peruutusoperaatioista, joiden piti toimia maksupäätteen kanssa, sekä kuittien lähettämisestä sähköpostina tai SMS-viestinä. Tämän lisäksi työn tekijä oli vastuussa lokaalista tietokannasta ja sen varmistamisesta, että lokaaliin tietokantaan jääneet transaktiot yritetään viedä säännöllisin väliajoin AWS:ssä sijaitsevaan tietokantaan. Muut ryhmän jäsenet olivat vastuussa kuittien tulostamisesta maksupäätteen avulla, maksupääteyhteyden parantamisesta sekä käyttöliittymän parantamisesta. Ryhmä avusti toisiaan tarpeen mukaan.

5.1.2 Yhteys maksupäätteeseen

Kassaohjelmisto voi ottaa yhteyden erilliseen maksupäätteeseen joko Bluetooth- tai TCP-yhteydellä.

Maksupäätteen kanssa kommunikointi toimii JSON-RPC-menetelmällä. Kassaohjelmisto voi pyytää maksupäätteeltä tietoja tietyillä pyynnöillä. Nämä pyynnöt lähetetään joko TCP- tai RFCOMM-yhteydellä. Yhteyden ylläpitämiseen lähetetään maksupäätteelle _Keepalive-pyyntö. _Keepalive-pyyntö ei ota vastaan mitään parametreja useimmista muista kutsuista poiketen. Muita yhteyden ylläpitämiskutsuja ovat _Info, _Error, _CloseReason ja _Sync. [11.]

JSON-RPC-viestit koostuvat alussa olevasta Hex-koodatusta kahdeksanmerkkisestä numerosarjasta, jonka jälkeen on kaksoispiste. Kaksoispisteen jälkeen on JSON-muotoinen American Standard Code for Information Interchange -viesti (ASCII). Lopuksi viestissä on vielä uusi rivi. [11.]

```
 00000055:{"jsonrpc":"2.0","method":"ExampleMethod",
"params":{"argument":"value"},"id":"req-1"}
```

Esimerkkikoodi 7. Esimerkki JSON-RPC-viestistä maksupäätteelle. [11]

Ostosta tehdessä maksupäätteelle lähetetään Purchase-kutsu. Tämän kutsun parametrien perusteella aloitetaan maksupäätteessä ostoprosessi. Kaikki Purchase-kutsuun vaadittavat parametrit nähdään Esimerkkikoodista 8.

```
{
      "jsonrpc": "2.0",
      "id": "1516092376397",
      "method": "Purchase",
      "params": {
            "api_key": "API_KEY",
            "timestamp": "2018-01-16T08:46:16.397000Z",
            "protocol_version": 1,
            "pos_id": 123,
            "sale_location_code": "POS21",
            "cashier_code": "USER1",
            "cashier_language": "fi",
            "amount": 100,
            "currency": "EUR",
            "receipt_id": 999,
            "sequence_id": 456,
            "bypass_pin": false,
            "forced_authorization": true
      }
}
```

Esimerkkikoodi 8. Kassaohjelmiston tekemä Purchase-kutsu maksupäätteelle yhdellä eurolla.

Maksupääte lähettää oston onnistuessa kassaohjelmistolle JSON-viestin, jossa on mukana transaktioon liittyviä tietoja mm. käytetystä kortista, transaktion tunnisteesta, kuittien sisällöstä jne. Maksupääte sisältää lähettämissään viesteissään ”response_to” -kentän, jonka avulla selviää, mihin kutsuun se vastaa.

5.1.3 Koodin kääntäminen

Androidille tehdyt koodit käännetään Android Application Package -tiedostoiksi, jotka ajamalla Android-laitteessa saadaan kaikki sovelluksen resurssit käyttöön niin kuvista kirjoitettuun koodiin. Useilla Android-kehitystyökaluilla tämä käännös voidaan ajaa automaattisesti, jolloin sovellusten kehittäminen on helpompaa.

Tässä työssä on käytetty kehitystyökaluna Android Studio -ohjelmaa, jolla voidaan ajaa automaattisesti tehty sovellus Android-laitteessa. Tämä vaatii laitteen kiinnittämisen ensin tietokoneeseen, usein Universal Serial Bus -johdon (USB) avulla, mutta laite voidaan myös yhdistää esim. langattoman verkon kautta.

5.2 Versionhallinta

Kassaohjelmiston versionhallintaan käytettiin Git-versionhallintaohjelmistoa. Git on yleisimmin käytetty moderni versionhallintaohjelmisto. Alkuperäisen Git-ohjelmiston kehitti Linus Torvalds vuonna 2005, mutta Git on nykyään avointa lähdekoodia ja sitä kehitetään jatkuvasti. [12.]

Git on muihin versionhallintaohjelmistoihin verrattuna nopea ja turvallinen. Kun Git määrittelee eroavaisuuksia versioiden välillä, ei se ota huomioon tiedostojen nimiä, kuten jotkut muut versionhallintaohjelmistot, vaan Git huomioi eroavaisuudet itse tiedoston sisällön perusteella. [12.]

Projektien lähdekoodin turvallisuuteen Git käyttää SHA-1-salausalgoritmia. Tiedostojen sisällöt ja suhteet muihin tiedostoihin ja kansioihin, versiot, tagit sekä kaikki projektin muutokset on suojattu tällä salausalgoritmillä. Kaikki projektiin tehdyt muutokset tarkistetaan SHA-1-tarkistussummalla. Tarkistussumma on 40-merkkinen merkkijono, ja se

koostuu heksadesimaalimerkeistä. Tämän salauksen avulla projektien historia pysyy suojattuna, eikä historiaa pystytä muuttamaan ilman, että siitä jäisi jokin jälki. [12.]

Tässä työssä käytettiin Gitin lisäksi verkkosivustoa Github, johon projekti tallennettiin. Github on verkkosivusto, jota voidaan pitää säilytyspaikkana Gittiä käyttäville projekteille. Github tarjoaa käyttäjilleen graafisen käyttöliittymän. [13.]

5.3 Jatkuva integraatio

Kassaohjelmiston toteutuksessa käytettiin Travis CI -nimistä jatkuvan integraation palvelua.

Jatkuvalla integraatiolla tarkoitetaan prosessia, jossa automatisoidaan tehdyn tuotteen koonti ja testaus. Yhdistämällä jatkuva integraatio Gitin kaltaiseen versionhallintaan voidaan jokaisesta koodin muutoksesta koota erillinen versio. Näin voidaan varmistaa, että muutokset eivät riko tuotetta. Jatkuva integraatio myös helpottaa kehittäjiä toimimaan ryhmässä sekä kehittämään pienempiä kokonaisuuksia, joita halutaan yhdistää tuotteeseen isojen kokonaisuuksien sijaan. [14; 15.]

Travis CI -palvelu kopioi Githubissa olevan projektin omaan virtuaaliseen ympäristöön, jonka jälkeen se tekee koodille haluttuja tehtäviä sekä testejä. Travis CI voi mm. rakentaa Android-sovelluksesta APK-tiedoston. Jos jokin tehtävistä tai testeistä jostain syystä epäonnistuu, Travis CI pitää koontia rikkinäisenä. Jos kaikki kuitenkin onnistuu, on koonti mennyt läpi. Koonnin jälkeen tuote voidaan esimerkiksi julkaista Google Play Kaupassa. [15.]

Travis CI -palvelun tehtävät määritellään .travis.yml-tiedostossa, joka sijaitsee projektin päähakemistossa. Tiedostossa voi määritellä esimerkiksi. mitä käyttöjärjestelmää ympäristö tulee käyttämään ja minkälaisia eri komponentteja projektin koonti vaatii. [15.]

Tässä työssä tehty kassaohjelmisto julkaistiin Google Play Kauppaan käyttämällä fastlane-työkalua. Jos Travis CI -palvelun tekemä koonti oli mennyt läpi onnistuneesti, lähetettiin tuotettu APK-tiedosto Google Play Kauppaan fastlane-työkalun avulla.

Kassaohjelmiston julkaiseminen tehtiin aina, kun projektin päähaaraan tuli muutoksia. Päähaarasta tehdystä APK-tiedostosta julkaistiin alfa-versio Google Play Kauppaan.

Tällöin voitiin valita kassaohjelmiston viimeisimmän version testaajat sisäisesti. Jos kassaohjelmisto oli toimiva, voitiin se manuaalisesti julkaista beeta-versiona Google Play Kaupassa. Tällöin kassaohjelmiston sai ladata sellaiset henkilöt, joilla oli beeta-versioon oikeus.

6 Amazon Web Services -pilvipalvelun käyttö

Tässä työssä AWS-pilvipalvelusta käytettiin kuutta sen tarjoamaa palvelua:

- Amazon Cognito
- Amazon API Gateway
- AWS Lambda
- Amazon Relational Database Service (RDS)
- Amazon Simple Notification Service (SNS)
- Amazon Simple Email Service (SES).

Insinöörityön tekijä oli vastuussa Amazon Cogniton sekä SNS- ja SES-palveluiden käytöstä. Muut kassaohjelmiston tekijät olivat vastuussa API Gatewaystä, AWS Lambdasta sekä RDS:ssä toimivasta tietokannasta. Näihin kuului myös tietoturvan hallinta. Ryhmän jäsenet avustivat toisiaan tarpeen mukaan.

6.1 Amazon Cognito

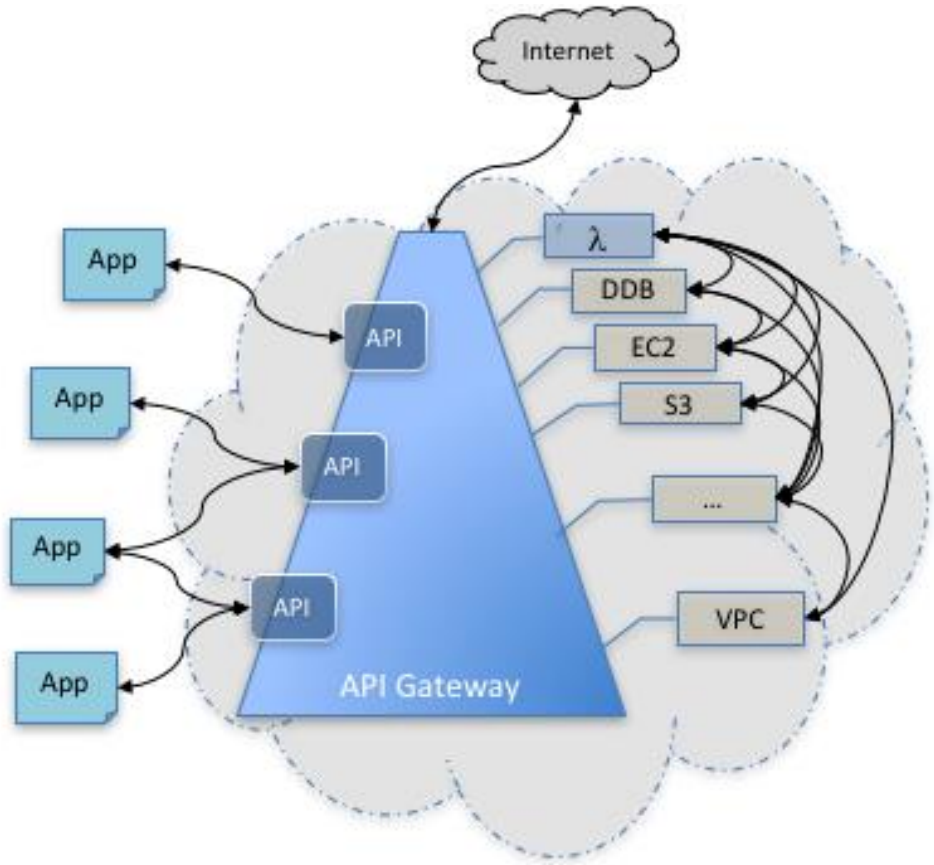
Amazon Cognito on AWS:n palvelu, jonka avulla voi hallita sovelluksen käyttäjiä rekisteröinnin ja kirjautumisen avulla. Amazon Cogniton avulla voidaan myös hallita käyttäjien oikeuksia eri sovelluksiin. Lisäksi Amazon Cogniton avulla voi tallentaa käyttäjien tietoja lokaalisti, jolloin sovellukset voidaan tehdä toimiviksi offline-tilassa. Käyttäjien tiedot voidaan synkronoida eri laitteiden välillä, jolloin käyttökokemus on aina samanlainen, vaikka käyttäisikin eri laitetta. [16.]

Tässä työssä tehdyssä kassaohjelmistossa Amazon Cognitoa käytettiin hallinnoimaan käyttäjiä. Amazon Cognito on helppo liittää Androidille kehitettyyn sovellukseen, sillä Amazon on julkaissut siitä oman Android-kirjaston.

Käyttäjä voi rekisteröityä kassaohjelmistoon sähköpostiosoitteella. Rekisteröinnin yhteydessä sähköpostiosoite täytyy varmistaa oikeaksi. Varmistuskoodi lähetetään käyttäjälle sähköpostiin. Tämä varmistuskoodi on 6-merkkinen numerojono, joka käyttäjän täytyy syöttää kassaohjelmistoon rekisteröinnin yhteydessä. Amazon Cognito hoitaa varmistuskoodin lähettämisen.

6.2 Amazon API Gateway

Amazon API Gateway on AWS:n palvelu, joka mahdollistaa kehittäjien tehdä, ylläpitää, monitoroida ja turvata erilaisia ohjelmointirajapintoja. Amazon API Gatewayn avulla voidaan tehdä ohjelmointirajapintoja, jotka ottavat yhteyttä muihin AWS:n palveluihin ja helpottavat näin sovelluksien kehittämistä. Amazon API Gatewayn voidaan kuvitella olevan pääväylä internetistä tulevilta pyynnöiltä. Kaikkien pyyntöjen on siis tultava Amazon API Gatewayn kautta, joka vie ne muiden palveluiden käytettäväksi kuvan 8 mukaisesti. [17.]



Kuva 8. Amazon API Gateway -ohjelmien ja internetin välillä. [17]

Tässä työssä Amazon API Gatewaytä käytettiin siltana kassaohjelmiston ja Lambda-funktion välillä. Näin pystyttiin varmistamaan, että rajapintoja kutsuva käyttäjä on kirjautunut sisään oikealla käyttäjällä.

6.3 AWS Lambda

AWS Lambda on pilvipalvelu, jossa koodia voi ajaa ilman, että joudutaan hallinnoimaan erillisiä palvelimia. Tämä helpottaa ohjelmistojen kehittäjiä, sillä heidän ei tarvitse välittää palvelinten koosta tai niiden skaalautuvuudesta, vaan AWS Lambda hoitaa kaiken tämän taustalla. AWS Lambdalla skaalautuvuus ei ole ongelma, sillä sen sisällä olevaa koodia ei ajeta tietyssä palvelimessa vaan sillä on käytössä koko AWS:n infrastruktuuri. Tämän avulla AWS Lambdalla voi ajaa minkälaista koodia tahansa, otti sovellus vastaan sitten muutamia pyyntöjä päivässä tai tuhansia muutamassa sekunnissa. [18.]

Tässä työssä AWS Lambdaa käytettiin vastaanottamaan kassaohjelmistolta HTTP-pyyntöjä. Pyyntöjä käytettiin tietokannasta luvun ja kirjoittamisen avuksi.

AWS Lambdassa ajettava koodi on kehitetty JavaScriptillä Express-sovelluskehystä käyttäen. Express on minimaalinen ja joustava Node.js-sovelluskehys, joka helpottaa etenkin rajapintojen kirjoittamista. Vaikka Express onkin itsessään hyvin minimaalinen, ovat monet kehittäjät kehittäneet omia väliohjelmistoja auttaakseen muita kehittämään esimerkiksi käyttäjien kirjautumisia. [19.]

Kaikki AWS Lambdaan vastaanotettavat HTTP-pyynnöt validoidaan, jotta voimme olla varmoja, että pyynnössä on kaikki tarvittavat tiedot oikeassa muodossa. Validointiin käytetään avuksi express-validator-väliohjelmistoa. Pyyntö voidaan hylätä, ellei se läpäise validointia. Tällöin käyttäjälle annetaan virhe, josta käy selville, että se oli validointivirhe. Jos pyynnön validointi menee läpi, annetaan sille oikeus päästä käsiksi tietokannassa olevaan tietoon.

AWS Lambdassa oleva funktio tarjosi kassaohjelmiston Android-sovellukselle REST-rajapinnan, jota vasten se pystyi tekemään pyyntöjä. REST-rajapinta koostui seuraavanlaisesti:

```
GET  /api/v1/ping
GET  /api/v1/catalog
POST /api/v1/catalog/product/add
```

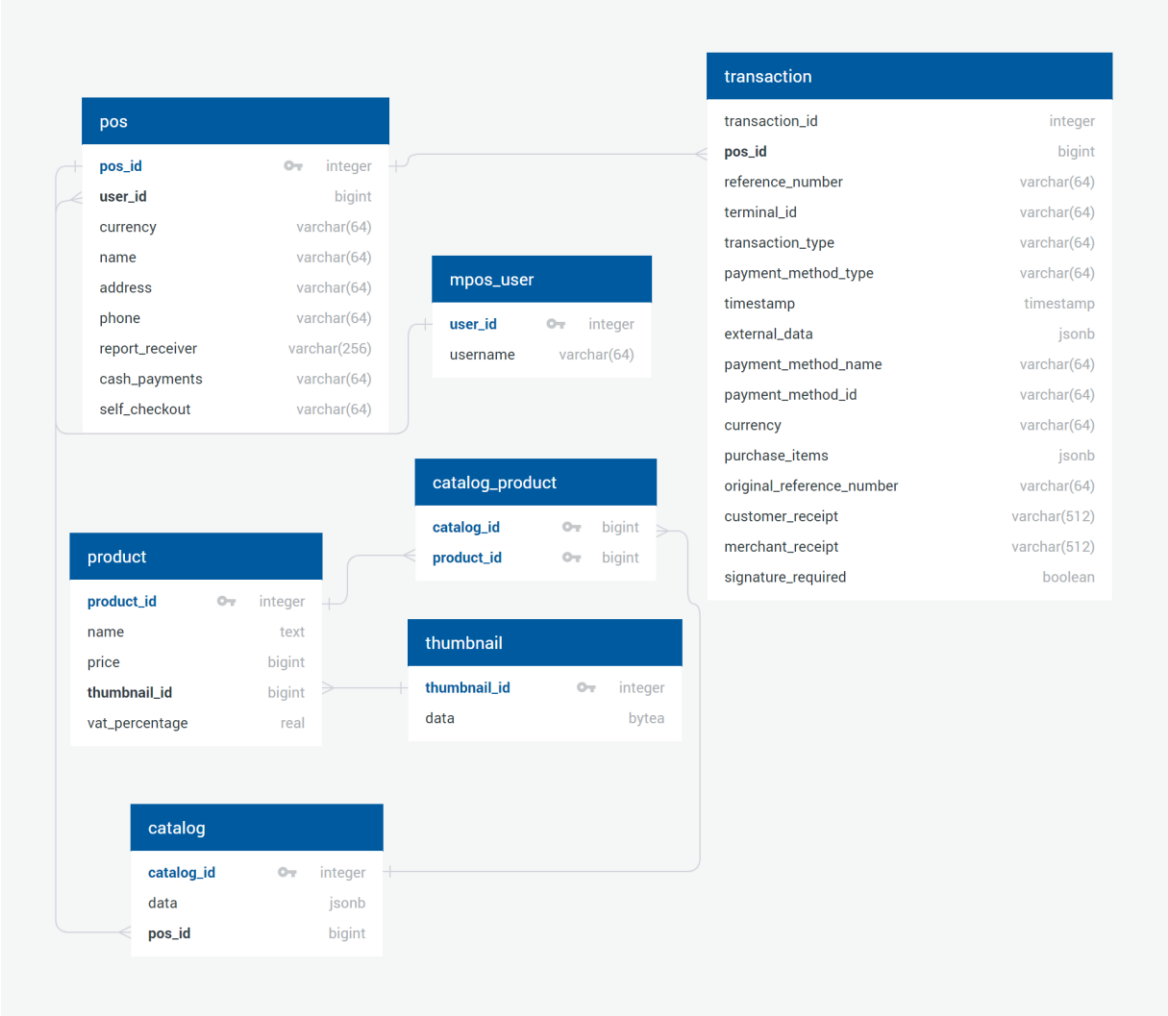
```
POST /api/v1/catalog/product/update
POST /api/v1/catalog/product/delete
GET  /api/v1/catalog/product/thumbnail/:id
GET  /api/v1/transaction
POST /api/v1/transaction
POST /api/v1/transaction/refund
POST /api/v1/transaction/cancel
GET  /api/v1/transaction/:referenceNumber
GET  /api/v1/pos
POST /api/v1/pos
GET  /api/v1/report/send
```

6.4 Amazon RDS

Amazon RDS on palvelu, jonka avulla voidaan perustaa relaatiotietokanta. RDS:n avulla voidaan hallinnoida sekä skaalata perustettua relaatiotietokantaa pilvessä. Amazon RDS eroaa normaalista palvelimesta mm. seuraavilla tavoilla [20]:

- Resurssien, kuten tallennustilan saatavuus on helpompaa.
- RDS hoitaa varmuuskopiot, ohjelmistopäivitykset, automaattiset vian havaitsemiset ja tarvittaessa palautukset.
- RDS tarjoaa asynkronisen toisen instanssin mitä voi käyttää ongelmatapauksissa.
- Lisäksi RDS tarjoaa ylimääräisiä lukukopioita alkuperäisestä tietokannasta, jos tarvitsee enemmän lukuskaalautuvuutta.

Tässä työssä tehdyssä kassaohjelmistossa Amazon RDS:sää käytettiin hallinnoimaan PostgreSQL-tietokantaa. Tietokantaan tallennettiin tietoja mm. käyttäjistä, heidän asetuksistaan, transaktioista ja tuotteista.



Kuva 9. Kassaohjelmiston tietokannan rakenne.

Kassaohjelmiston tietokanta koostuu 7 eri taulusta, kuten kuvasta 9 nähdään. Tietokannan mpos_user-taulussa on kassaohjelmistoon rekisteröityneen Amazon Cognito -käyttäjätunnus. Pos-taulussa viitataan kyseiseen käyttäjätunnukseen. Pos-taulu sisältää kaikki käyttäjätunnuksen asetukset, kuten tiedot mitä kuittiin tulostetaan. Jokaisella pos-taulun käyttäjällä on oma kataloginsa catalog-taulussa. Kassaohjelmiston käyttäjien katalogit koostuvat tuotteista, joihin viitataan catalog_product-taulussa. Product-taulu sisältää tuotteiden tiedot, joita ovat esimerkiksi nimi ja hinta. Tuotteilla voi myös olla kuva, joka tallennetaan thumbnail-tauluun binääritietona. Transaction-tauluun tallennetaan kaikki kassaohjelmistossa tehdyt onnistuneet transaktiot, kuten ostot, hyvitykset ja peruutukset. Transaktioista tallennetaan tietokantaan erilaista tietoa, esim. tuotteet tai kuittitiedot.

Tietokannassa olevien tietojen muokkaaminen onnistui Lambdassa olevan funktion avulla. Funktio on Node.js-ajoympäristössä pyörivä Express-palvelin. Esimerkkikoodista

9 nähdään, kuinka pos-taulun tietoja päivitetään. Funktion alussa validoidaan, että tuleva HTTP-pyynnön sisältää tarvittavat tiedot. Jos pyyntö ei pääse validointia läpi, palautetaan käyttäjälle HTTP-tilakoodi 400. Onnistuneesta tietokantamuutoksesta palautetaan käyttäjälle aina HTTP-tilakoodi 200.

```
module.exports.setPosSettings = (req, res) => {
  console.log('Received POST request for pos/')

  req.check('currency').isCurrency().withMessage('Invalid currency')
  req.check('cashPayments').isBoolean().withMessage('Cash payments has to be
boolean value')
  req.check('selfCheckout').isBoolean().withMessage('Self checkout has to be
boolean value')

  req.getValidationResult()
    .then((validationResult) => {
      if (validationResult.isEmpty()) {
        const body = req.body

        return req.db.none(
          `UPDATE pos SET
              currency=$1,
              name=$2,
              address=$3,
              phone=$4,
              report_receiver=$5,
              cash_payments=$6,
              self_checkout=$7
            WHERE pos_id=$8`,
          [
            body.currency,
            body.name,
            body.address,
            body.phone,
            body.reportReceiver,
            body.cashPayments,
            body.selfCheckout,
            req.posId
          ]
        )
          .then(() => res.sendStatus(200))
      } else {
        console.log('Failed to update pos settings - error: ', util.in-
spect(validationResult.array()))
        res.status(400).json({
          code: MessageCode.VALIDATION_ERROR
        })
      }
    })
    .catch(err => {
      console.log('Failed to update pos settings - error: ', err)
      res.status(500).json({
        code: MessageCode.INTERNAL_ERROR
      })
    })
}
```

Esimerkkikoodi 9. Pos-taulun päivitysoperaatio Lambda-funktion avulla.

6.5 Amazon SES ja SNS

Amazon SES on palvelu, jota käyttämällä voi lähettää ja vastaanottaa sähköpostiviestejä. SES-palvelun sähköpostiviestejä voidaan lähettää omista sähköpostiosoitteista sekä verkkotunnuksista. Tässä työssä SES-palvelua käytettiin lähettämään ostoksen kuitti asiakkaan sähköpostiin, jos käytössä oli maksupääte, jossa kuitin tulostaminen ei ole mahdollista. [21.]

Sähköpostikuitin lähettäminen pystyttiin toteuttamaan tässä työssä helposti AWS:n tarjoamien Android-kirjastojen avulla. Sähköpostin kuitin lähettäminen nähdään esimerkkikoodista 10. Sähköpostikuitin sisältö on HyperText Markup Language -kielen (HTML) mukaisesti tehty merkkijono, joka sisältää mm. tuotteet ja niiden hinnat, myyjän tiedot sekä ajankohdan.

```
private void sendEmailReceipt() {
    Destination destination = new Destination().withToAddresses(mReceiver);
    Content subject = new Content().withData(EMAIL_SUBJECT);
    Content htmlBody = new Content().withData(mReceipt);
    Body body = new Body().withHtml(htmlBody);
    Message message = new Message().withSubject(subject).withBody(body);

    SendEmailRequest request = new SendEmailRequest().with-
Source(EMAIL_FROM).withDestination(destination).withMessage(message);

    try {
        AmazonSimpleEmailServiceClient client = new AmazonSim-
pleEmailServiceClient(mCredentialsProvider);

        client.setRegion(Region.getRegion(Regions.fromName(BuildConfig.AWS_RE-
GION)));
        client.sendEmail(request);

        Log.d(TAG, "The email was sent succesfully to " + mReceiver);
    } catch (Exception ex) {
        Log.e(TAG, "The email was not sent to " + mReceiver, ex);
    }
}
```

Esimerkkikoodi 10. Sähköpostikuitin lähettämiseen käytetty funktio.

Amazon SNS on palvelu, jonka avulla voidaan hallinnoida erilaisten viestien lähettämistä. Tässä työssä SNS:ssää käytettiin ainoastaan SMS-viestien lähettämiseen. Viestejä käytettiin kuittien lähettämiseen, kuten SES-palvelussa. SNS-palvelulla julkaisijat voivat kuitenkin tehdä aiheita, joita tilaajat voivat tilata. Aiheiden avulla viestejä tai ilmoituksia voidaan lähettää vain sellaisille asiakkaille, jotka ovat sen tilanneet. Aiheen viestejä voidaan mm. lähettää sähköpostin tai SMS:n avulla. [22.]

7 Kassaohjelmiston Android-sovelluksen käyttäminen

Kassaohjelmistoa voidaan käyttää kaikilla Android-laitteilla, jotka käyttävät vähintään Androidin 5.0-versiota. Kassaohjelmiston voi ladata laitteelle Google Play Kaupasta, mutta se ei ole julkisesti jaossa. Jotta ohjelmiston saa näkyviin Google Play Kauppaan, on henkilön täytynyt ottaa yhteyttä kassaohjelmiston tekijöihin ja hänen on pitänyt ilmoittaa sähköpostiosoitteensa heille.

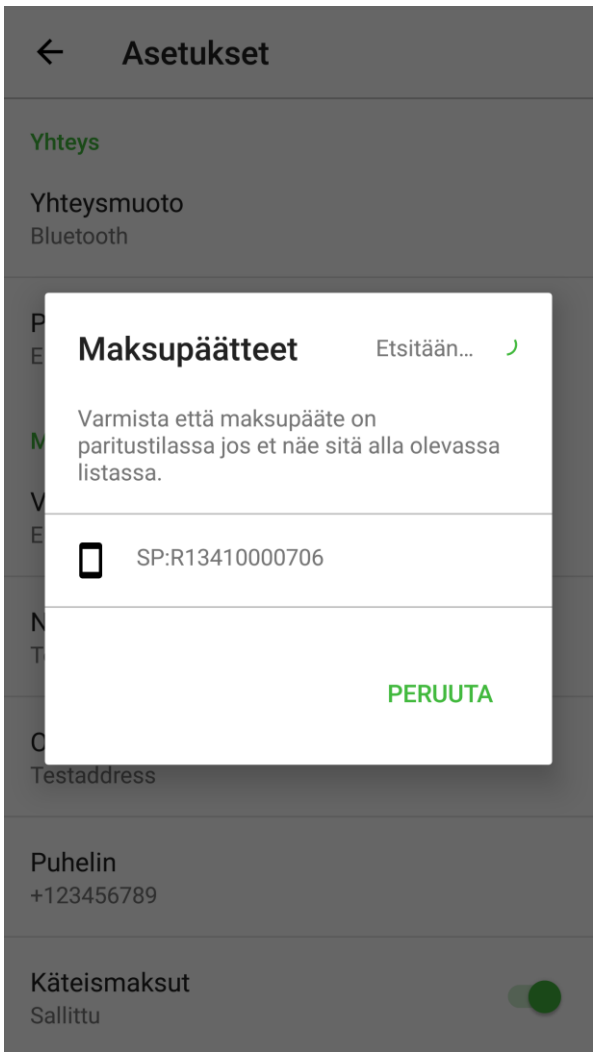
Jotta kassaohjelmistoa voisi käyttää, tarvitsee sitä käyttävän henkilön luoda käyttäjätunnus. Käyttäjätunnus voidaan luoda sähköpostilla, ja käytettävä sähköposti on varmennettava varmistuskoodilla, joka lähetetään käytettyyn sähköpostiosoitteeseen.

7.1 Yhteyden ottaminen maksupäätteeseen

Kun käyttäjä on kirjautunut sisään, pääsee hän yhdistämisnäkymään, josta voi ainoastaan päästä asetuksiin, kirjautua ulos tai ottaa yhteyden maksupäätteeseen.

Jotta maksupäätteeseen voitaisiin ottaa yhteyttä, käyttäjän tarvitsee muuttaa yhteysasetuksia.

Yhteysasetuksissa voidaan asettaa kassaohjelmiston käyttävän joko Bluetooth- tai TCP-yhteyttä. Jos käyttäjä valitsee Bluetooth-yhteyden, joutuu hän parittamaan Android-laitteen maksupäätteen kanssa ennen yhteyden muodostamista. Kuvasta 10 nähdään Bluetoothin paritusikkuna. Paritus laitteiden välillä joudutaan tehdä vain kerran. TCP-yhteyden valitsemalla käyttäjä joutuu manuaalisesti syöttämään maksupäätteen IP-osoitteen.

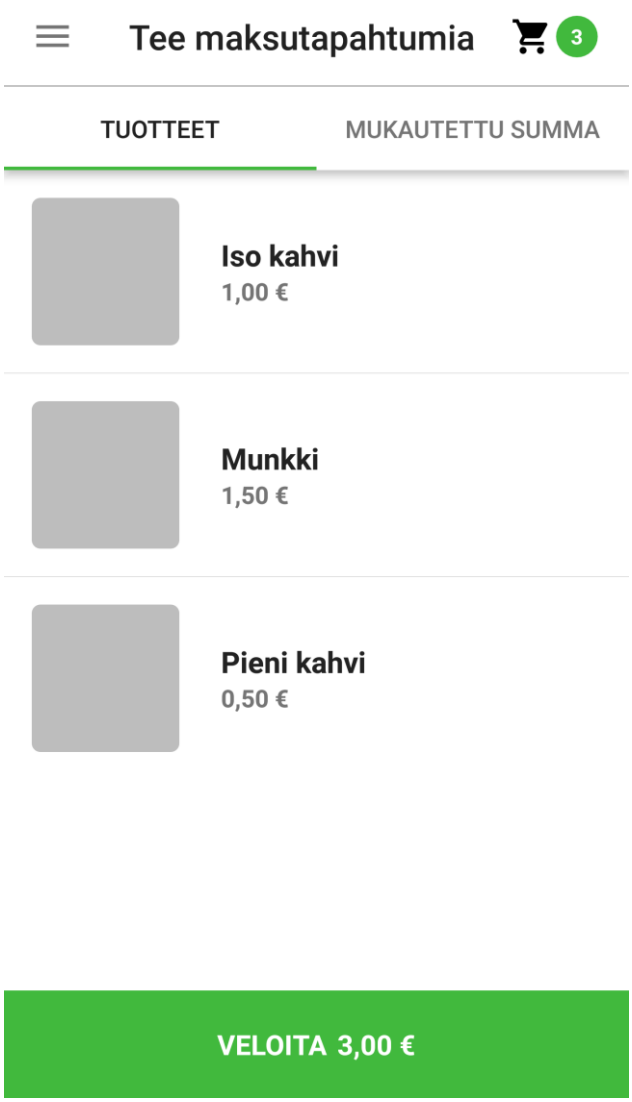


Kuva 10. Kassaohjelmisto Bluetooth-paritustilassa.

Tässä työssä tehtyä kassaohjelmistoa voi jatkossa käyttää myös ilman maksupäätettä.

7.2 Oston tekeminen

Oston tekeminen voidaan aloittaa päänäkymästä, joka nähdään kuvassa 11. Päänäkymässä on lista kauppiaan tuotteista sekä mukautetun summan syöttömahdollisuus. Näiden lisäksi päänäkymässä pääsee myös tarkastelemaan ostoskoria.



Kuva 11. Kassaohjelmiston sovelluksen päänäkymä, jossa ostoskorissa on 3 tuotetta.

Kassaohjelmiston käyttäjä lisää ensin tuotteita ostoskoriin, jonka jälkeen hän voi painaa veloita nappia. Veloita-nappi avaa kuvan 12 mukaisen Androidin DialogFragment-komponentin, joka tuodaan päänäkymän päälle.

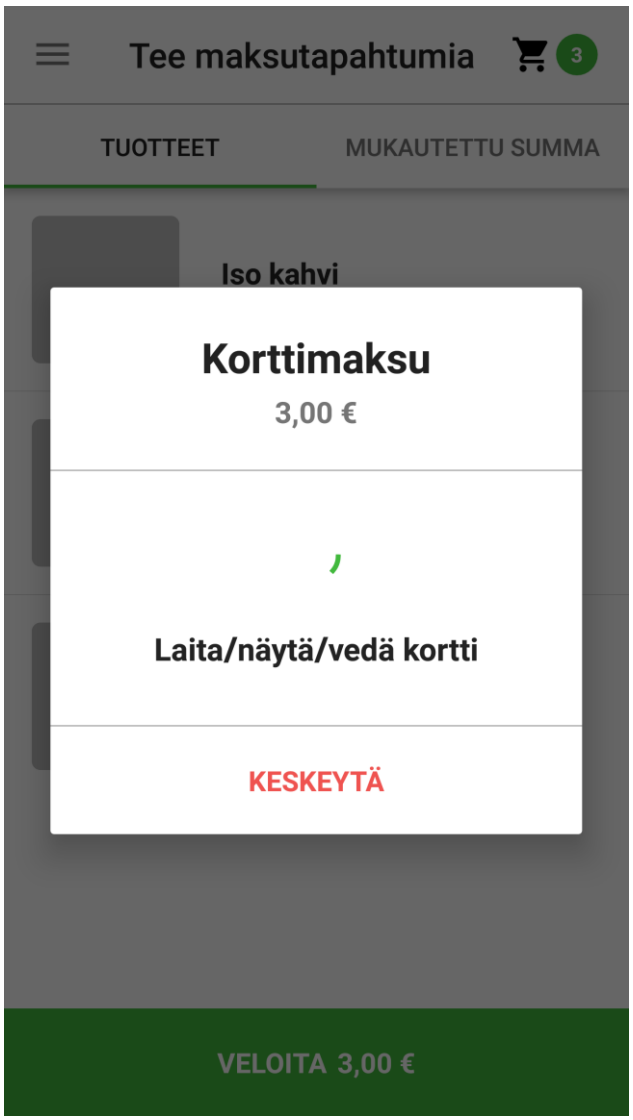
Kuva 12. Veloita-napin painaminen avaa DialogFragment-komponentin, jossa voi valita ostoksessa käytettävän maksutavan

Kun käyttäjä valitsee korttimaksun, kassaohjelmisto lähettää maksupäätteelle pyynnön aloittaa maksuprosessi ostoskorissa olevalla summalla. Tämän jälkeen maksupääte kertoo sovellukselle maksuprosessin tilasta PosMessage-viestillä. PosMessage-viesti on tarkoitettu myyjälle, ja se kertoo, mitä maksun maksajalta odotetaan. Viesti voi kertoa mm. maksajaa näyttämään korttia tai ottamaan se pois maksupäätteestä. Kuvassa 13 nähdään korttimaksun valinnan jälkeinen tila.

Jos käyttäjä valitsee käteismaksun, edetään suoraan kuvassa 14 olevaan tilaan.

```
public class PaymentDialogMethodFragment extends android.support.v4.app.Frag-
ment {

    @BindView(R.id.amount) TextView mAmountView;
    @BindView(R.id.btn_cash) LinearLayout mCashBtn;
    @BindView(R.id.btn_credit_card) LinearLayout mCardBtn;
    @BindView(R.id.btn_cancel) LinearLayout mCancelBtn;
    @BindView(R.id.cash_payment_view) LinearLayout mCashPaymentView;

    private PaymentDialogFragment.Listener mListener;

    public static PaymentDialogMethodFragment newInstance(String amount, bool-
ean isConnected) {
        PaymentDialogMethodFragment f = new PaymentDialogMethodFragment();
        Bundle args = new Bundle();
        args.putString(Constants.ARGS_KEY_AMOUNT.name(), amount);
        args.putBoolean(Constants.ARGS_KEY_IS_CONNECTED.name(), isConnected);
        f.setArguments(args);
        return f;
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
        View view = inflater.inflate(R.layout.fragment_payment_dialog_method,
container, false);
        ButterKnife.bind(this, view);
        boolean isConnected = getArguments().getBoolean(Con-
stants.ARGS_KEY_IS_CONNECTED.name());
        mCardBtn.setVisibility(isConnected ? View.VISIBLE : View.GONE);
        mAmountView.setText(getArguments().getString(Con-
stants.ARGS_KEY_AMOUNT.name()));
        return view;
    }

    @Override
    public void onAttach(Context context) {
        super.onAttach(context);
        if (context instanceof PaymentDialogFragment.Listener) {
            mListener = (PaymentDialogFragment.Listener) context;
        } else {
            throw new RuntimeException(context.toString() + " must implement
PaymentDialogFragment.Listener");
        }
    }

    @OnClick(R.id.btn_cancel)
    public void handleCancel() {
        mListener.dismissPaymentDialog();
    }

    @OnClick(R.id.btn_cash)
    public void handleCashPayment() {
        mListener.startCashPayment();
        mCardBtn.setVisibility(View.GONE);
        mCashBtn.setVisibility(View.GONE);
        mCancelBtn.setVisibility(View.GONE);
        mCashPaymentView.setVisibility(View.VISIBLE);
    }

    @OnClick(R.id.btn_credit_card)
    public void handleCardPayment() {
        mListener.startCardPayment();
    }
}
```

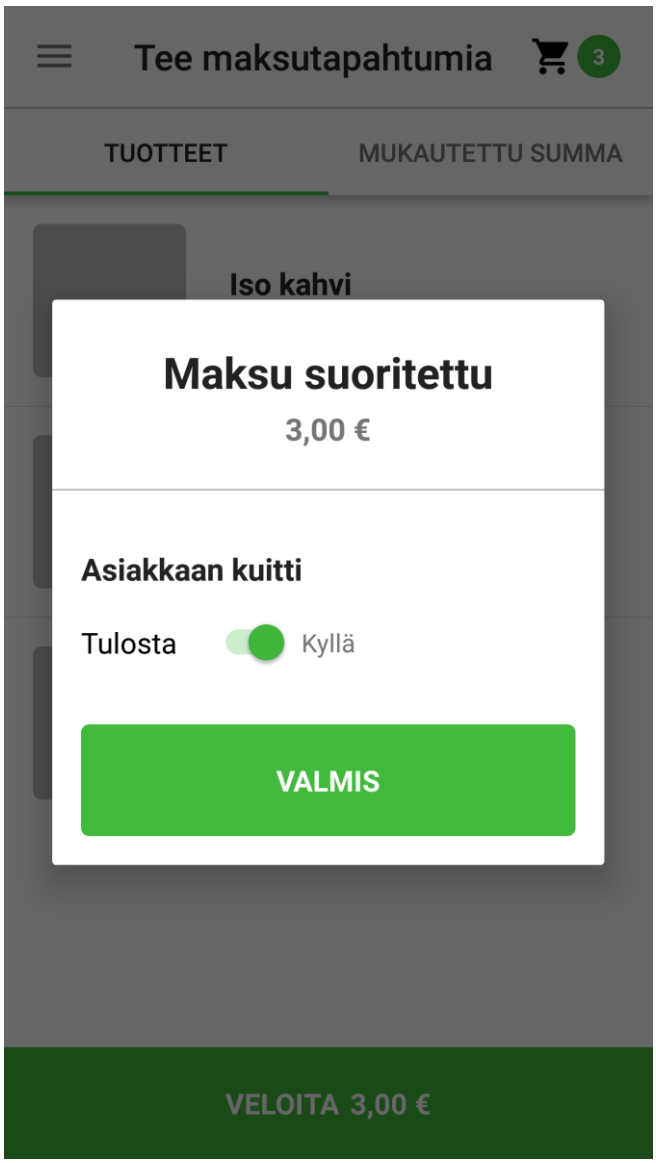
Esimerkkikoodi 11. Maksutavan valinnan Fragment-komponentti.



Kuva 13. Korttimaksu on käynnissä maksupäätteessä.

Kun maksu on suoritettu onnistuneesti maksupäätteessä tai käyttäjä on valinnut käteismaksun, edetään sovelluksessa tilaan, jossa voidaan tulostaa asiakkaan kuitti haluttaessa (kuva 14). Jos kassaohjelmistoa käyttää myyjä, tulostetaan hänelle automaattisesti erillinen kuitti maksun suorituksen jälkeen.

Jos maksupäätteessä tulostaminen ei ole mahdollista, voidaan kuitti lähettää käyttäjälle joko tekstiviestillä tai sähköpostilla. Myyjän kuitti joudutaan tällaisessa tilanteessa hakemaan maksuhistorian kautta. Maksuhistorian kautta voidaan aina lähettää tai tulostaa kuitit.



Kuva 14. Onnistuneen maksun jälkeen voi käyttäjä halutessaan tulostaa asiakkaan kuitin.

Onnistuneen oston jälkeen kassaohjelmisto tyhjentää ostoskorin ja palaa päänäkymään, joka nähdään kuvassa 11.

7.3 Tuotteiden muokkaaminen

Kassaohjelmistossa on mahdollista ylläpitää tuotekatalogia. Tuotteita voidaan lisätä ja poistaa sekä näiden lisäksi muokata jo olemassa olevia. Tuotteille voi antaa nimen, hinnan, arvonlisäveroprosentin sekä kuvan. Tuotteet tallennetaan AWS-pilvipalvelussa olevaan tietokantaan, jolloin samaa tuotekatalogia voi käyttää usealla eri laitteella.

Tuotteita muokattaessa siirrytään tuotteiden muokkausnäkymään, joka rakentuu eri aktiviteetista kuin päänäkymä. Aktiviteetti ajetaan startActivityForResult()-funktiolla, joka palattaessa päänäkymään palauttaa vastauksen tuotteiden muokkausaktiviteetista. Tällöin uutta tuotekatalogia ei tarvitse ladata taustajärjestelmästä palattaessa päänäkymään. Esimerkkikoodista 12 nähdään, kuinka tuotteiden muokkausaktiviteetti ajetaan.

```
@Override
public boolean onNavigationItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.nav_edit_catalog:
            Intent intent = new Intent(this, EditCatalogActivity.class);
            intent.putExtra(Constants.KEY_PRODUCTS.name(), GsonUtil.createIn-
stance().toJson(mProducts));
            startActivityForResult(intent, EDIT_CATALOG_CODE);
        }
    }
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data)
{
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == EDIT_CATALOG_CODE) {
        String products = data.getStringExtra(Constants.KEY_PRODUCTS.name());
        if (products == null) {
            mProducts = GsonUtil.createInstance().fromJson("[]", new TypeTo-
ken<List<Product>>(){}.getType());
        } else {
            mProducts = GsonUtil.createInstance().fromJson(products, new Type-
Token<List<Product>>(){}.getType());
        }
    }
}
```

Esimerkkikoodi 12. Tuotteiden muokkaamisen jälkeen palatessa päänäkymään tuotteita ei tarvitse hakea uudelleen taustajärjestelmästä.

8 Yhteenveto

Insinöörityössä kehitettiin kassaohjelmisto Android-käyttöjärjestelmää käyttäville laitteille. Kassaohjelmiston Android-sovellus tuotettiin Java-ohjelmointikielellä ja se yhdistettiin taustajärjestelmään, joka toimii AWS-pilvipalvelussa. Taustajärjestelmän tärkeimmät ominaisuudet ovat käyttäjien hallinnointi sekä tietokannan ylläpitäminen, jossa säilytetään tietoja mm. käyttäjästä, tuotteista sekä käyttäjän suorittamista ostoksista. Taustajärjestelmää käytettiin myös sähköposti- sekä SMS-kuittien lähettämiseen, jos kassan käytössä oli maksupääte, jolla kuitin tulostaminen ei ollut mahdollista.

Kassaohjelmisto valittiin toteutettavaksi Android-laitteille, sillä pienkauppiaat eivät halua maksaa ylimääräistä ohjelmistosta pyörittävästä laitteesta. Android-laitteita saa hankittua edullisemmin kuin muita käyttöjärjestelmiä käyttäviä laitteita. Pienkauppiaita varten valittiin AWS-pilvipalvelu taustajärjestelmän ylläpitämiseksi, sillä emme halunneet hoitaa omia palvelimia.

Kassaohjelmistoa kehitettiin 2-4 hengen ryhmässä noin 7 kuukauden ajan. Työssä jatkettiin jo olemassa-olevaa kassaohjelmistoa, joka oli tehty osoittamaan, että kassaohjelmiston toteuttaminen oli mahdollista. Olemassaolevassa kassaohjelmistossa oli tehtynä yhteyden muodostaminen maksupäätteeseen. Kassaohjelmistoon lisättiin uusia ominaisuuksia sekä liitettiin se taustajärjestelmään.

Kassaohjelmisto toimii maksupäätteen kanssa ja vaatii internetyhteyden toimiakseen. Kassaohjelmiston tekemisessä huomioitiin kuitenkin internetyhteyden katkeaminen ja pyrittiin varmistamaan kaikkien transaktioiden tallennus taustajärjestelmään, vaikka internet-yhteys katkeaisikin hetkeksi ostoksen jälkeen. Jos osto on kesken maksupäätteessä ja yhteys katkeaa kassaohjelmiston ja maksupäätteen välillä, tarkistetaan maksun tila yhteyden tullessa takaisin.

Kassaohjelmiston liittäminen AWS-pilvipalvelussa toimivaan taustajärjestelmään oli helppoa AWS:n tarjoamien kirjastojen avulla. AWS helpotti myös taustajärjestelmien toteuttamista, sillä esimerkiksi käyttäjien autentikointiin ei tarvinnut tehdä omia toteutuksia.

Kassaohjelmiston kehittäminen jatkuu edelleen insinöörityön jälkeen.

Lähteet

1 The history of Android OS. 2018. Verkkodokumentti. Android Authority. <https://www.androidauthority.com/history-android-os-name-789433/>. Luettu 25.1.2018.

2 The history of AWS. 2017. Verkkodokumentti. Computerworld. <https://www.computerworlduk.com/galleries/cloud-computing/aws-12-defining-moments-for-the-cloud-giant-3636947/>. Luettu 25.1.2018.

3 AWS Revenue. 2017. Verkkodokumentti. GeekWire. < https://www.geekwire.com/2017/another-strong-quarter-aws-42-percent-jump-revenue-4-1b/>. Luettu 28.2.2018.

4 Serveless Computing. 2017. Verkkodokumentti. Boolean World. <https://www.booleanworld.com/serverless-computing-explained/>. Luettu 25.1.2018.

5 iZettle Go. 2018. Verkkodokumentti. Google Play Kauppa. <https://play.google.com/store/apps/details?id=com.izettle.android&hl=fi>. Luettu 5.2.2018.

6 Application Fundamentals. 2018. Verkkodokumentti. Android Developers. <https://developer.android.com/guide/components/fundamentals.html>. Luettu 12.1.2018.

7 Fragment. 2018. Verkkodokumentti. Android Developers. <https://developer.android.com/reference/android/app/Fragment.html>. Luettu 12.1.2018.

8 DialogFragment. 2018. Verkkodokumentti. Android Developers. <https://developer.android.com/reference/android/app/DialogFragment.html>. Luettu 12.1.2018.

9 Hyvästi Android-java – kotlin jyrää. 2017. Verkkodokumentti. Tivi. <https://www.tivi.fi/Kaikki_uutiset/hyvasti-android-java-kotlin-jyraa-6681995>. Luettu 12.1.2018.

10 Kotlin vs Java. 2017. Verkkodokumentti. Android edX Community. <http://androiddeveloper.galileo.edu/2017/10/16/kotlin-vs-java-what-is-the-difference/>. Luettu 12.1.2018.

11 Poplatek JSONPOS API. 2018. Verkkodokumentti. Poplatek. <https://www.poplatek.fi/jsonpos-api/maksupaaterajapinta>. Luettu 3.3.2018.

12 What is Git. 2018. Verkkodokumentti. Atlassian. <https://www.atlassian.com/git/tutorials/what-is-git>. Luettu 13.1.2018.

13 What Exactly Is Github Anyway. 2012. Verkkodokumentti. TechCrunch. <https://techcrunch.com/2012/07/14/what-exactly-is-github-anyway/>. Luettu 13.1.2018.

14 What is Continuous Integration. 2018. Verkkodokumentti. Visual Studio. <https://www.visualstudio.com/learn/what-is-continuous-integration/>. Luettu 6.2.2018.

15 Core Concepts for Beginners. 2018. Verkkodokumentti. Travis CI. <https://docs.travis-ci.com/user/for-beginners/>. Luettu 6.2.2018.

16 Amazon Cognito. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html>. Luettu 23.1.2018.

17 Amazon API Gateway. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/apigateway/latest/developerguide/welcome.html>. Luettu 23.1.2018.

18 AWS Lambda. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/lambda/latest/dg/welcome.html>. Luettu 23.1.2018.

19 Express/Node introduction. 2018. Verkkodokumentti. MDN. <https://developer.mozilla.org/en-US/docs/Learn/Server-side/Express_Nodejs/Introduction>. Luettu 23.1.2018.

20 Amazon RDS. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html>. Luettu 23.1.2018.

21 Amazon Simple Notification Service. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/sns/latest/dg/welcome.html>. Luettu 23.1.2018.

22 Amazon SES. 2018. Verkkodokumentti. AWS. <https://docs.aws.amazon.com/ses/latest/DeveloperGuide/Welcome.html>. Luettu 23.1.2018.