Opinnäytetyö (AMK)

Tietojenkäsittely

2018

Juha-Pekka Toivola

HENKILÖLASKENTA-JÄRJESTELMÄ

– toteutus Firebase-sovelluskehitysalustalle

TURKU AMK
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittely

2018 | 47

Juha-Pekka Toivola

HENKILÖLASKENTAJÄRJESTELMÄ

- toteutus Firebase-sovelluskehitysalustalle

Tämän opinnäytetyön tavoitteena oli luoda pilvipohjainen henkilölaskentajärjestelmä konenäköä käyttävälle liiketunnistimelle.

Opinnäytetyön teoriaosuudessa käsiteltiin henkilölaskentaa yleisesti ja työn soveltavassa osuudessa käydään läpi henkilölaskentajärjestelmän rakenne ja toteutus vaiheittain. Kehitettävä henkilölaskentajärjestelmä koostuu kolmesta osasta, liiketunnistimesta, pilvialustalla toimivasta tietokantasovelluksesta sekä mobiilisovelluksesta, jonka avulla voidaan näyttää laskettujen henkilöiden määrä.

Henkilölaskentajärjestelmän liiketunnistin käyttää kameralaitetta ja konenäkösovellusta (OpenCV) liikkeen tunnistamiseen ja laskee näin alueen ohi kulkevat henkilöt. Liiketunnistin tallentaa sen tekemät havainnot pilvialustalla toimivaan tietokantaan.

Henkilölaskentajärjestelmän pilvialustana toimii Googlen Firebase ja sen tietokantana Firestore, johon lasketut henkilömäärät tallennetaan. Henkilölaskentajärjestelmä sisältää myös käyttäjätilien hallinnan ja käyttäjien autentikoinnin. Lasketut henkilömäärät voidaan hakea tietokannasta ja esittää mobiililaitteelta oman mobiilisovelluksen avulla sekä reaaliajassa että eri aikajaksoilta. Mobiilisovellus toteutetaan Android-käyttöjärjestelmälle Android Studiolla. Mobiilisovellus voidaan myös asettaa hälyttämään käyttäjää aina kun liiketunnistinsovellus havaitsee liikettä.

Toimeksiantajayritys arvioi projektin onnistuneeksi ja projektin tuloksia voidaan käyttää sekä nykyisten että uusien tuotteiden kehittämiseen.

ASIASANAT:

Android, Cloud, Firebase, Firestore, Node.js, henkilölaskenta, liiketunnistus

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Business Information Technology

2018 | 47

Juha-Pekka Toivola

PEOPLE-COUNTING SYSTEM

- Google Firebase Application

The purpose of this thesis was to document the process of developing a cloud-based people-counting system that uses motion detection to track and collect information about the number of visitors within an area.

The people-counting system consists of three different parts, a motion detector, a cloud database, and a mobile application.

The cloud database stores the number of visitors and it was implemented using Google Firebase and its Firestore database. The people-counting system also uses Firebase Authentication for user management and to provide authentication for different users.

The motion detector is a camera device connected to a computer that uses computer vision software (OpenCV) to recognize moving objects. The cloud database is updated whenever a person enters or leaves the area. The connection from the motion detector to the cloud database was implemented with Node.js using Firebase SDK.

The people-counting system also includes a mobile application. The visitor numbers can be retrieved from the database and displayed remotely on a mobile device in real-time as well as from specified time periods. The mobile application was developed for Android using Android Studio which supports both phone and tablet devices. The mobile application can also be set to alert the user when motion is detected by the motion detector.

The project was considered a success by the client and the project results can be used to develop and improve new or existing products

KEYWORDS:

Android, Cloud, Firebase, Firestore, Node,js, Motion Detection, People Counter

SISÄLTÖ

1 JOHDANTO **8**

2 HENKILÖLASKENTA **10**
2.1 WLAN-pohjainen henkilölaskenta 10
2.2 Infrapunan käyttö henkilölaskennassa 11
2.3 Ultraäänen käyttö henkilölaskennassa 11
2.4 Henkilölaskenta konenäköalgoritmin avulla 12

3 TIETOKANTA **13**
3.1 Firebasen käyttöönotto 13
3.2 Firebase Console 14
3.3 Firebase-projektin käyttäjätilit 15
3.4 Firestore-tietokanta 17
3.5 Firebase Cloud Messaging 21

4 LIIKETUNNISTUS **23**
4.1 Node.js -toteutus 23
4.2 Liikehavaintojen päivittäminen Firestore-tietokantaan 23
4.3 Firebase Cloud Messaging -viestien lähetys 25

5 MOBIILISOVELLUS **26**
5.1 Android Studio 26
5.2 Android-sovelluksen rakenne 26
5.3 Helppokäyttöisyys ja lokalisointi 28
5.4 Activity ja Fragment 28
5.5 Android Manifest 31
5.6 Android-sovelluksen käyttöliittymä ja ulkoasu 32
5.7 Android-sovelluksen Java-luokat 34
5.7.1 Device-luokka 35
5.7.2 Timestamp-luokka 35
5.7.3 Utilities-luokka 36
5.7.4 LoginActivity-luokka 36
5.7.5 UserAuthStateListener-luokka 38
5.7.6 DeviceListActivity-luokka 38

5.7.7 DeviceDetailActivity-luokka 39
5.7.8 SettingsActivity-luokka ja Android-sovelluksen asetukset 40
5.7.9 Firebase Cloud Messaging -viestien vastaanotto Android-laitteella 41
5.8 Android-sovelluksen viimeistely julkaisua varten 42

6 YHTEENVETO **44**

LÄHTEET **46**

KOODI

Koodi 1. Liikehavainto JSON-muodossa. 24
Koodi 2. Android-sovelluksen riippuvuudet ohjelmistokirjastoihin. 27
Koodi 3. Sovelluksen luvat. 32
Koodi 4. Device-luokka. 35
Koodi 5. Timestamp-luokka. 36
Koodi 6. Utilities-luokka, verkkoyhteyden tilan tarkistava metodi. 36
Koodi 7. UserAuthStateActivity ja käyttäjän kirjautumistilan seuranta. 38
Koodi 8. AndroidManifest.xml Service-komponentit. 42

KUVAT

Kuva 1. Käyttäjän Firebase-projektit. 14
Kuva 2. Firebase Console. 15
Kuva 3. Firebasen käyttäjätilien hallinta. 15
Kuva 4. Firebasen käyttäjien kirjautumistapa. 16
Kuva 5. Firestore-tietokannan rakenne (Firebase 2018a). 17
Kuva 6. Projektin Firestore-tietokannan rakenne. 18
Kuva 7. Firestore-tietokannan säännöt (Rules). 20
Kuva 8. Firebase Cloud Messaging -viestin lähetys. 22
Kuva 9. Android-sovelluksen rakenne (Android Developers 2018i). 27
Kuva 10. Activityn elinkaari (Android Developers 2018m). 29
Kuva 11. Fragmentin elinkaari (Android Developers 2018e). 30
Kuva 12. Master/Detail-näkymä (Android Developers 2018e). 31
Kuva 13. View- ja ViewGroup-objektien hierarkia (Android Developers 2018f). 32
Kuva 14. Android-sovelluksen valikko. 33
Kuva 15. LoginActivity ja sen näkymä Android-laitteessa. 37
Kuva 16. LoginActivity ja virheilmoitus Android-laitteessa. 37
Kuva 17. DeviceListActivity ja sen näkymä Android-laitteessa. 39
Kuva 18. DeviceDetailActivity ja sen näkymä Android-laitteessa. 40
Kuva 19. SettingsActivity ja sen näkymä Android-laitteessa. 41
Kuva 20. Firebase Console ja google-services.json-tiedoston luonti. 43

KÄYTETTY SANASTO

Android Studio	Googlen kehittämä Android-käyttöjärjestelmälle tarkoitettu ohjelmointiympäristö.
Android	Googlen kehittämä mobiilikäyttöjärjestelmä.
API Key	Application Programming Interface Key. Merkkijono, josta ohjelmistorajapintaan pyynnön tekevä asiakasohjelma voidaan tunnistaa.
API	Application Programming Interface. Ohjelmistorajapinta.
APK	Android Application Package. Android-sovelluksen koodi, resurssit ja data paketoidaan APK-tiedostoiksi, josta ne voidaan asentaa.
C++	Ohjelmointikieli.
Callback-metodi	Callback-metodilla tarkoitetaan metodia, joka annetaan argumenttina suoritettavaksi jollekin toiselle metodille.
FCM	Firebase Cloud Messaging. FCM-palvelu mahdollistaa viestien lähettämisen Firebase-alustaa käyttävien sovellusten välillä.
Firebase	Sovelluskehitysalusta.
Firestore	Firebase-alustalla toimiva NoSQL-tietokanta.
Gradle	Gradle-työkalu vastaa Java-ohjelmakoodin riippuvuuksista muihin kirjastoihin sekä ohjelmakoodin kääntämisestä.
IoT	Internet of Things.
Java	Ohjelmointikieli.
JavaScript	Ohjelmointikieli.
JSON	JavaScript Object Notation. Tiedostomuoto tiedon siirtämiseen. Tieto tallennetaan avain- ja arvopareina.
Kotlin	Ohjelmointikieli.
MAC	Media Access Control. Verkkolaitteen yksilöivä osoite.
Node.js	Tapahtumapohjainen JavaScriptin ajoympäristö.
NoSQL	Kuvaa tietokantoja, jotka eivät perustu relaatiomalliin.
NPM	Node Package Manager. NPM on Node.js:n moduulien hallintaan tarkoitettu työkalu.

OpenCV	Open Source Computer Vision Library. Avoimeen lähdekoodiin perustuva konenäköön ja kuvankäsittelyyn tarkoitettu ohjelmistokirjasto.
Python	Ohjelmointikieli.
REST	Representational State Transfer. Arkkitehtuurimalli ohjelmistorajapinnoille.
RPC	Remote Procedure Call. Protokolla, jonka avulla asiakasohjelma voi kommunikoida palvelimen kanssa.
Rules	Firestore-tietokannan käytön säännöt ja rajoitukset.
SDK	Software Development Kit. Työkalut ohjelmistokehitystä varten jollekin tietylle sovellusalustalle.
SSH	Secure Shell. Salatun yhteyden mahdollistava tietoliikenneprotokolla.
Topic	Firebase Cloud Messaging -palvelun kautta lähetettävän viestin aihe.
UI	User Interface. Sovelluksen käyttöliittymä.
WLAN	Wireless Local Area Network. Langaton lähiverkkotekniikka.
ZeroMQ	Järjestelmien ja sovellusten väliseen viestintään tarkoitettu ohjelmistokirjasto.

1 JOHDANTO

Tämän opinnäytetyön tavoite on luoda pilvipohjainen henkilölaskentajärjestelmä konenäköpohjaiselle liiketunnistimelle. Tämä henkilölaskentajärjestelmä koostuu liiketunnistimesta, pilvialustalla toimivasta tietokannasta sekä mobiilisovelluksesta, jonka avulla voidaan näyttää laskettujen henkilöiden määrä.

Opinnäytetyön toimeksiantaja on Fidera Oy. Fidera Oy on vuonna 2013 perustettu yritys, joka tarjoaa automatisoidun kulunvalvonnan ja seurannan järjestelmiä sekä IoT-analytiikkaan perustuvia ratkaisuja eri alojen yrityksille (Fidera 2018). Opinnäytetyön tuloksia ja sen aikana kehitettävää henkilölaskentajärjestelmää tullaan käyttämään pohjana uusien palveluiden ja tuotteiden kehittämiseen.

Opinnäytetyön teoriaosuudessa kerrotaan, mitä henkilölaskenta on ja mihin sitä voidaan käyttää. Opinnäytetyön soveltavassa osuudessa luodaan henkilölaskentajärjestelmä. Henkilölaskentajärjestelmä koostuu kolmesta osasta, liiketunnistimesta, pilvialustalla toimivasta tietokannasta sekä mobiilisovelluksesta. Soveltavassa osuudessa käydään läpi vaiheittain henkilölaskentajärjestelmän sovellusratkaisun rakenne ja toteutus.

Henkilölaskentajärjestelmän pilvialustana toimii Googlen Firebase ja sen tietokantana Firestore, joka vastaa tehtyjen liikehavaintojen tallentamisesta. Tietokanta toimii pilvialustalla siksi, että liikehavainnot ovat näin kaikkien havaintotietoja tarvitsevien käyttäjien ja laitteiden saatavilla verkon kautta paikasta riippumatta. Firebase mahdollistaa myös henkilölaskentajärjestelmän käyttäjätilien hallinnan ja käyttäjien autentikoinnin.

Liiketunnistus perustuu konenäköön, ja siihen tarkoitettu sovellus on jo toteutettu toimeksiantajan puolesta. Liiketunnistin koostuu kamerasta, konenäkösovelluksesta sekä yksinkertaisesta JavaScript-sovelluksesta. Konenäkösovellus toimii tietokoneella, josta on yhteys kameraan. Konenäkösovellus vastaa kameralaitteesta saatavan videokuvan käsittelystä ja liiketunnistamisesta videokuvasta. JavaScript-sovellus vastaa liiketunnistimien tekemien liikehavaintojen seurannasta ja ottaa yhteyden Firebase-alustalla toimivaan Firestore-tietokantaan tallentaakseen liikehavainnot. Tämä JavaScript-sovellus toteutetaan Node.js:n avulla. Tässä opinnäytetyössä ei käsitellä varsinaista konenäkösovellusta ja liiketunnistamista kuvasta, vaan keskitytään tehtyjen liikehavaintojen lukemiseen ja käsittelyyn.

Henkilölaskentasovelluksen laskemat henkilömäärät voidaan esittää mobiililaitteelta sille luotavan oman mobiilisovelluksen avulla. Mobiilisovellus toteutetaan lähtökohtaisesti Android 6.0 -käyttöjärjestelmälle, ja se tukee sekä puhelin- että tablettilaitteita. Laskettuja henkilömääriä voidaan tarkastella mobiililaitteelta sekä reaaliajassa että eri aikajaksoilta. Mobiilisovellus voidaan myös asettaa hälyttämään käyttäjää aina kun liiketunnistin havaitsee liikettä. Mobiilisovellus kehitetään Android Studiolla.

2 HENKILÖLASKENTA

Henkilölaskennan tavoitteena on laskea jollakin ennalta määritellyllä alueella kulkevien henkilöiden määrä. Henkilönlaskenta toteutetaan jonkin liikettä tunnistavan laitteen avulla. Henkilölaskenta toimii siten, että aina kun henkilö ylittää jonkin tietyn rajan, niin liiketunnistin havaitsee tämän liikkeen ja kirjaa sen järjestelmään laskurin samalla kasvaen yhdellä. (Kajala ym. 2009.) Henkilölaskentaratkaisut perustuvat usein Infrapunaan, WLAN-käyttäjämäärien seurantaan tai konenäköön perustuvaan ratkaisuun. Henkilölaskuri voi myös olla jokin yksinkertainen laite, painike tai nimilista, jolla henkilö kirjaa itsensä järjestelmään (Kajala ym. 2009).

Henkilölaskentaa voidaan käyttää monissa erilaisissa kohteissa eri tarkoituksiin. Käyttökohteita voivat olla esimerkiksi kaupat, kauppakeskukset, museot sekä mitkä tahansa vastaavat ihmisten käytössä olevat kokoontumistilat. (Cetinkayaa & Akcayb 2015.)

Henkilaskennan avulla voidaan seurata tilassa olevien henkilöiden lukumäärää eri ajanjaksoina. Henkilölaskennasta saatavia tiedoista voi olla hyötyä esimerkiksi eri tilojen ja ympäristöjen turvallisuusjärjestelyissä. Henkilölaskentaa voidaan myös käyttää yrityksen liiketoiminnan kehittämiseen arvioimalla asiakkaiden lukumäärä eri vuorokauden aikoina, tämä auttaa erityisesti yritysten työntekijöiden työvuorojen suunnittelussa. (Fathi 2015.)

Henkilölaskentaan liittyviä haasteita ovat muun muassa liikkeentunnistukseen käytettävien laitteiden tarkkuus ja niiden sijoittaminen ympäristöön sopivalle paikalle, niin että se kattaa mitattavan alueen. Ongelmia saattaa myös aiheuttaa mahdollinen ympäristön vaihtuvuus, kuten muutokset tilan valaistuksessa tai lämpötiloissa. Ulkotiloissa ongelmia aiheuttaa myös mahdollinen ilkivalta, jolloin liiketunnistin tulee pystyä asentamaan huomaamattomaan paikkaan tai pystyä naamioimaan se ympäristöön. (Kajala ym. 2009.)

2.1 WLAN-pohjainen henkilölaskenta

WLAN-pohjainen henkilölaskenta perustuu tilassa olevien vierailijoiden käyttämien mobiililaitteiden käyttämään WLAN-yhteyteen. Organisaatio voi tarjota tilassa oleville henkilöille WLAN-yhteyden ja sen perusteella voidaan laskea tätä WLAN-yhteyttä käyttävien

laitteiden määrä. Mobiililaitteen hakiessa mahdollisia WLAN-verkkoja ja yhdistäessä siihen, saadaan selville yhdistävän mobiililaitteen yksilöivä MAC-osoite. Tilassa olevien henkilöiden määrä voidaan arvioida laskemalla WLAN-verkkoon yhteydessä olevien MAC-osoitteiden perusteella. WLAN-pohjaisen henkilölaskennan avulla saadaan myös selville henkilöiden keskimääräinen vierailuaika ja kuinka usein samat henkilöt vierailevat kyseisessä tilassa. (Information Commissioner's Office 2016.)

2.2 Infrapunan käyttö henkilölaskennassa

Infrapunaa voidaan käyttää henkilölaskennassa ulko- ja sisätiloissa. Infrapunatunnistimien käytön suurin etu on niiden edullisuus ja helppokäyttöisyys. Infrapunatunnistimet voidaan jakaa kahteen tyyppiin, aktiiviseen ja passiiviseen. (Kajala ym. 2009.)

Aktiivinen infrapunatunnistin koostuu lähettäjästä ja vastaanottimesta. Aktiivisella infrapunatunnistimella henkilölaskenta voidaan toteuttaa siten, että tilan sisäänkäynnille asetetaan lähetin toiselle puolelle ja sen vastaanotin vastakkaiselle puolelle, niin että niiden lähettämä ja heijastama säde ylittää sisäänkäynnin. Kun henkilö ylittää sisäänkäynnin, niin infrapunasäde katkeaa ja laskuri kasvaa yhdellä. (Kajala ym. 2009.)

Passiivinen infrapunatunnistin eroaa aktiivisesta siten, että se ei tarvitse vastaanotinta ja on tämän ansiosta helppo asentaa. Passiivisen infrapunatunnistimen lähettämä säde heijastuu takaisin ohi kulkevasta henkilöstä vastaanottimen sijaan. Säteen heijastuessa takaisin ohi kulkevasta henkilöstä, laskuri kasvaa yhdellä. Passiivisen infrapunatunnistimen suurin heikkous on se, että tietyn tyyppiset vaatteet tai henkilön kantamat tavarat voivat heijastaa säteen takaisin väärään suuntaa, jolloin ohi kulkeva henkilö jää havaitsematta. Ulkotiloissa myös muutokset säätilassa voivat aiheuttaa epätarkkuutta. Passiivinen infrapunatunnistin voi aktivoitua vahingossa esimerkiksi lumi- ja vesisateen aikana. (Kajala ym. 2009.)

2.3 Ultraäänen käyttö henkilölaskennassa

Henkilölaskennassa voidaan käyttää myös ultraääntä. Ultraäänitunnistimet toimivat osittain samalla periaatteella kuin infrapunatunnistimet ja ne voidaan jakaa samalla tavalla aktiivisiin ja passiivisiin. Ultraäänitunnistin toimii siten, että sen lähettämä ääniaalto heijastuu takaisin joko ohi kulkevasta henkilöstä tai vastakkaiselle puolelle asennettavasta

vastaanottimesta. Kun ääni heijastuu takaisin henkilöstä, laskuri kasvaa yhdellä. Ultraäänen käytön suurin ongelma on muutokset ilman lämpötilassa ja ne toimivat yleensä huonosti kylmissä lämpötiloissa. (Kajala ym. 2009.)

2.4 Henkilölaskenta konenäköalgoritmin avulla

Henkilömääriä voidaan myös laskea käyttämällä konenäköalgoritmia. Konenäön käyttö perustuu liikkuvien kohteiden tunnistamiseen videokuvasta. Kameralaite tallentaa videota ja konenäkösovellus etsii kuvista liikkuvia kohteita. Liikkuvat kohteet havaitaan vertaamalla kameran tuottamaa sen hetkistä kuvaa aiempaan kuvaan tai alkuperäiseen referenssikuvaan kuvattavasta ympäristöstä. Kun kuvassa tapahtuu jotain muutoksia, tämä voidaan olettaa jonkin kohteen liikkeeksi. OpenCV-ohjelmistokirjasto sisältää tarvittavat metodit liikkeentunnistamiseen kameran videokuvasta. (OpenCV 2017.)

Henkilölaskentaan voidaan käyttää myös kasvontunnistusta. Kasvontunnistus voidaan toteuttaa OpenCV-ohjelmistokirjaston avulla. OpenCV-ohjelmistokirjaston tarjoamat kasvojentunnistusmetodit ovat tarkkoja ja ne pystyvät erottamaan eri henkilöiden kasvot toisistaan. Kasvontunnistusta käyttäessä virheitä voi aiheuttaa henkilön kääntyminen tai liikkuminen, jolloin sama henkilö voi tulla laskettua kahdesti. Tämä voidaan yrittää estää asettamalla kameralaite sopivaan paikkaan siten, että se saadaan kuvaamaan tilan henkilöiden kasvojen etuosaa. (Cetinkayaa & Akcayb 2015.)

Konenäköalgoritmin käytön yleisiä ongelmia ovat erityisesti ulkotiloissa kuvattaessa ympäristössä tapahtuvat pienet muutokset ja liikkeet, kuten varjot, puiden ja kasvien liike tuulessa, sekä ympäristössä liikkuvat pienet eläimet.

3 TIETOKANTA

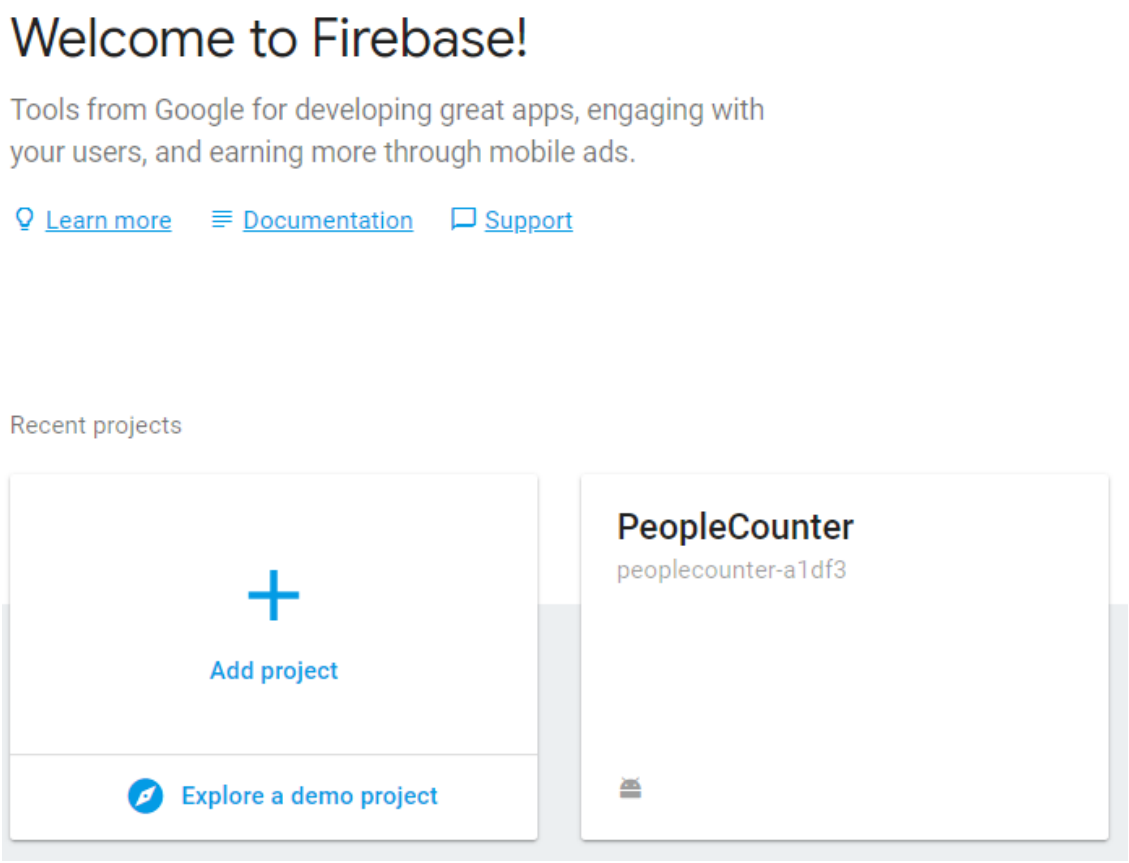
Tässä luvussa käsitellään henkilölaskentajärjestelmän pilvialustalla toimivaa tietokantaa, johon liiketunnistimen tekemät havainnot kirjataan. Pilvialustana toimii Googlen Firebase ja sen tietokantana Firestore.

Firebase on Googlen omistama ja ylläpitämä pilvessä toimiva sovelluskehitysalusta. Firebase mahdollistaa verkko- ja mobiilisovellusten nopean kehittämisen ja sisältää tätä varten useita eri työkaluja ja ominaisuuksia. Firebase sisältää muun muassa tietokannan ja tietovaraston kuva- ja videotiedostoille, sovellusten käyttäjätilien hallinnan sekä viestien lähetyksen eri sovellusten välillä. Firebase sisältää myös työkalut sovellusten testaukseen, suorituskyvyn mittaukseen ja analysointiin. (Firebase 2018d.)

Firebase tarjoaa tällä hetkellä SDK:n Pythonia, Unitya, iOS- ja Android-käyttöjärjestelmiä sekä Node.JS-sovelluksia varten (Firebase 2018c). Firestore-tietokantaa voidaan myös käsitellä SDK:n lisäksi REST- tai RPC-rajapinnan kautta (Firestore 2018e.)

3.1 Firebasen käyttöönotto

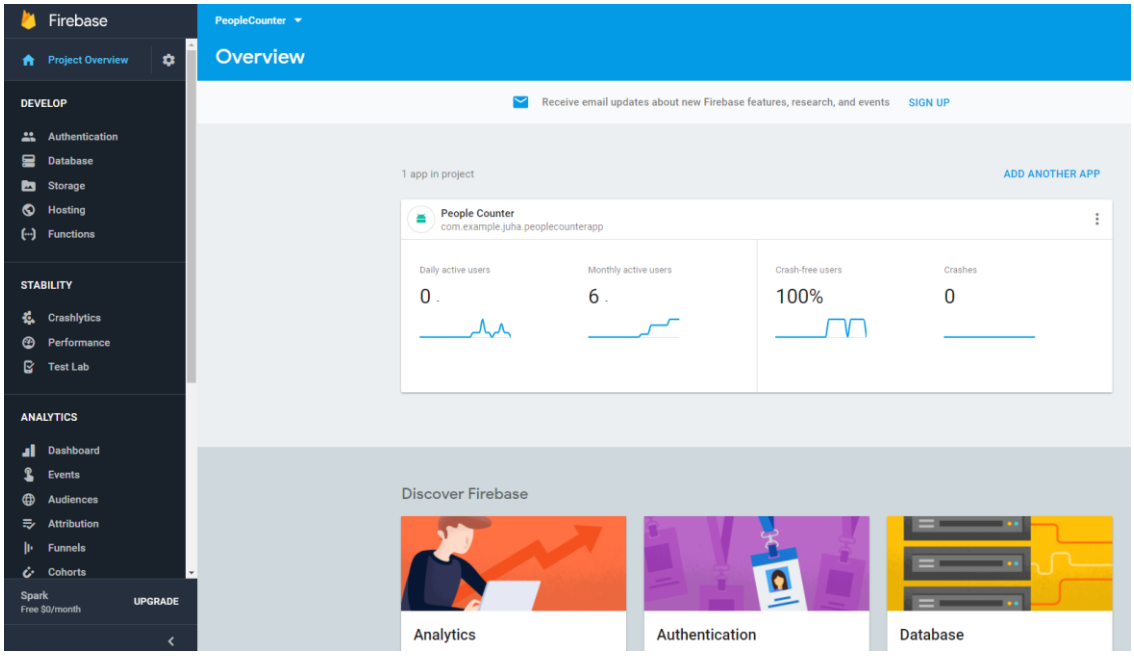
Firebase käyttöönotto on helppoa. Firebase voidaan ottaa käyttöön Firebasen omilta verkkosivuilta ja se vaatii ainoastaan Googlen käyttäjätunnuksen. Google-käyttäjätunnus voidaan luoda myös helposti Firebasen käyttöönoton yhteydessä Firebasen verkkosivuilta. Käyttäjän kirjauduttua Firebaseen, luodaan vain uusi Firebase-projekti, jolle annetaan jokin käyttäjän valitsema nimi. Käyttäjä voi myös luoda useita eri projekteja. Tässä opinnäytetyössä luotavan Firebase-projektin nimeksi annetaan People Counter (Kuva 1).



Kuva 1. Käyttäjän Firebase-projektit.

3.2 Firebase Console

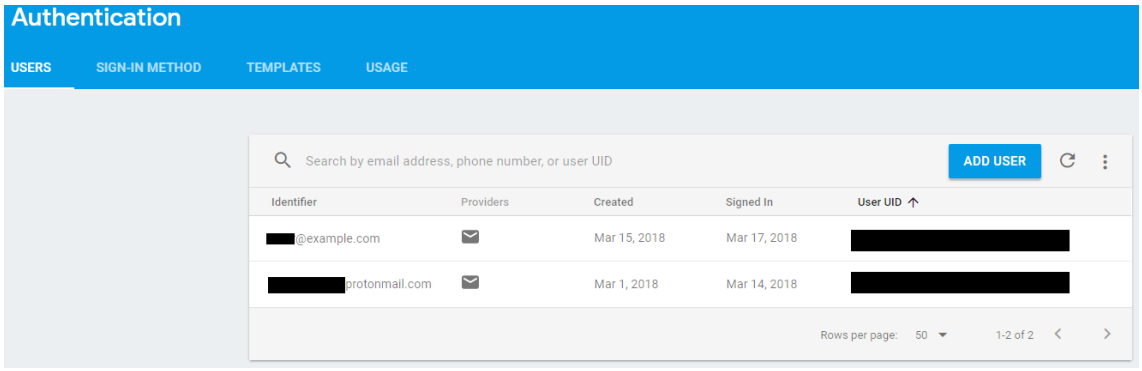
Firebase Console on Firebase-projekteille tarkoitettu verkkokäyttöliittymä (Kuva 2). Firebase Consolesta voidaan hallita koko projektin ominaisuuksia ja tarkastella sen käyttöön liittyviä tilastoja, kuten projektin sovellusten käyttäjiä- ja käyttäjämääriä. Firebase Consoleen kirjaudutaan Firebasen verkkosivulta omalla Googlen käyttäjätunnuksella. Tässä projektissa tullaan käyttämään Firebasen tarjoamista ominaisuuksista ainoastaan Firestore-tietokantaa, käyttäjätilien hallintaa sekä viestien lähettämistä (Notifications).



Kuva 2. Firebase Console.

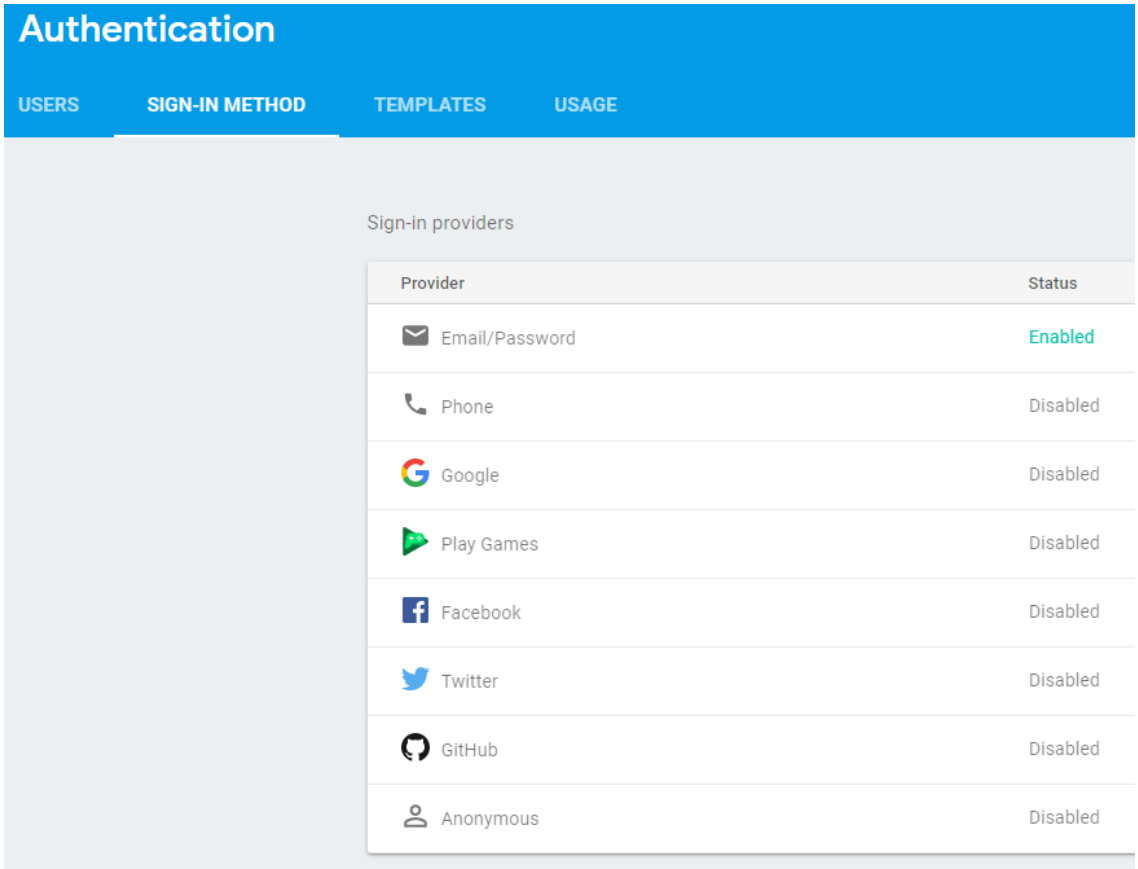
3.3 Firebase-projektin käyttäjätilit

Tässä projektissa luotavien sovellusten käyttäjiltä edellytetään aina kirjatumista omilla käyttäjätunnuksilla. Projektin sovelluksen käyttäjätilejä ja kirjautumiseen liittyviä ominaisuuksia voidaan lisätä, muokata ja poistaa Firebase Consolesta. Jokaisella Firebase-projektiin lisätyllä käyttäjällä on käyttäjätunnuksen ja salasanan lisäksi yksilöllinen UID-merkkijono, jolla käyttäjät voidaan erottaa toisistaan (Kuva 3).



Kuva 3. Firebasen käyttäjätilien hallinta.

Kirjautuminen Firebasea käyttäviin sovelluksiin voi tapahtua sähköpostin ja salasanan yhdistelmän tai pelkän puhelinnumeron kanssa. Firebase tukee myös eri sosiaalisen median palveluiden tarjoamia käyttäjätunnuksia kuten, Google, Facebook, Twitter sekä Github (Kuva 4). Käyttäjien luontiin voidaan myös asettaa rajoituksia sähköpostin ja IP-osoitteen suhteen. Tämän tarkoituksena on parantaa tietoturvaa estämällä useiden käyttäjätilien luomisen samasta IP-osoitteesta tai käyttämällä samaa sähköpostiosoitetta eri käyttäjille. Firebase sisältää myös tarvittavat työkalut käyttäjätunnusten hallintaan ja rekisteröintiin tarkoitettujen lomakkeiden luontiin verkkosivuja tai sovelluksia varten. (Firebase 2018b.)

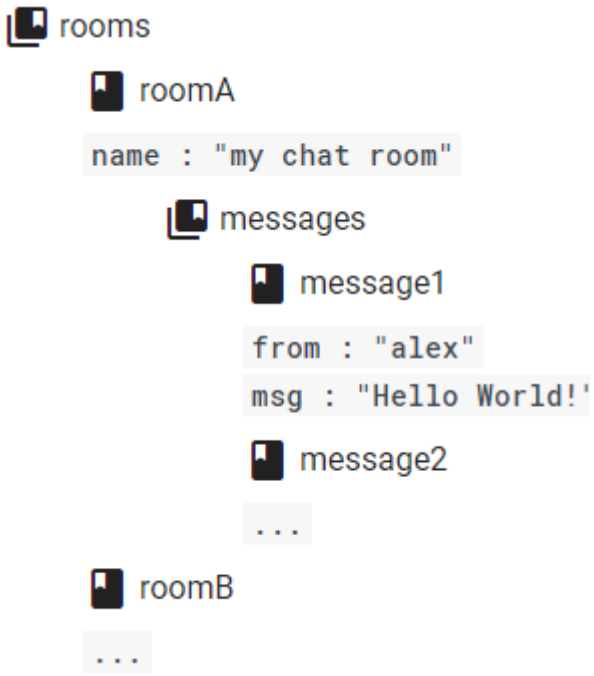


Kuva 4. Firebasen käyttäjien kirjautumistapa.

Tässä projektissa luotavien sovellusten käyttäjien kirjautuminen on mahdollista ainoastaan sähköposti- ja salasanayhdistelmällä. Projektin sovellusten käyttäjien luonti tapahtuu myös ainoastaan Firebase Consolen kautta.

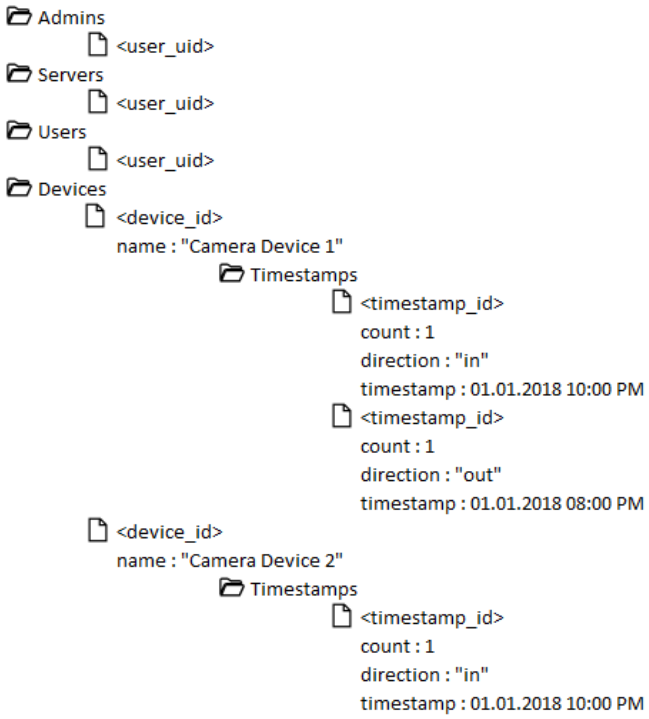
3.4 Firestore-tietokanta

Tietokantana toimii Firebasen sisältämä Firestore-tietokanta. Firestore on Firebase-alustalla toimiva NoSQL-tietokanta ja sen rakenne perustuu dokumentteihin (Document) ja hakemistoihin (Collection). Dokumenteilla on yksilöllinen Id-tunnusluku sekä niille voidaan määritellä eri tietotyyppisiä muuttujia. Dokumentti kuuluu aina johonkin hakemistoon ja dokumentit voivat sisältää alihakemistoja. Hakemistoille määritellään aina niiden yksilöivä nimi. Dokumentin Id-arvo voidaan määritellä itse tai se voi olla automattisesti generoitu satunnainen merkkijono. Kuvassa 5 on esimerkki Firestore-tietokannan rakenteesta. Firestore-tietokannan tuettuja muuttujatyyppejä ovat boolean, merkkijonot, kokonaisluvut, päivämäärät, paikkatiedot sekä taulukot. (Firebase 2018a.) Firestore on tällä hetkellä vielä beta-testausvaiheessa, mutta se sisältää tarvittavat ominaisuudet tätä projektia varten.



Kuva 5. Firestore-tietokannan rakenne (Firebase 2018a).

Tässä projektissa luotavan sovelluksen Firestore-tietokantaan on määritelty neljä päähakemistoa. Nämä neljä hakemistoa ovat *Devices*, *Admins*, *Users* ja *Servers*. Kuvassa 6 on esimerkki projektissa luotavan tietokannan rakenteesta.



Kuva 6. Projektin Firestore-tietokannan rakenne.

Devices-hakemiston sisältämät dokumentit kuvaavat järjestelmän liiketunnistinlaitteita, jotka tekevät havaintoja henkilömääristä. *Devices*-hakemiston jokaisella dokumentilla on yksi merkkijonotyyppinen muuttuja *name* ja yksi hakemisto nimeltä *timestamps*. Muuttuja *name* kuvaa laitteelle annettua nimeä, josta käyttäjä voi tunnistaa yksittäisen laitteen nimen perustella. *Timestamps*-hakemisto sisältää liiketunnistinlaitteen tekemät liikehavainnot. *Timestamps*-hakemiston dokumenteilla on muuttujat *count*, *direction* ja *timestamp*. Muuttuja *count* kertoo havaittujen henkilöiden lukumäärän havaintohetkellä ja muuttuja *direction* kertoo henkilöiden kulkusuunnan. Muuttuja *timestamp* on päivämäärätyyppinen muuttuja, joka kertoo liiketunnistuslaitteen tekemän havainnon kellonajan ja päivämäärän.

Hakemistot *Admins*, *Servers* ja *Users* sisältämät dokumentit kuvaavat henkilölaskentajärjestelmän eri käyttäjiä. Käyttäjät on jaettu kolmeen eri käyttäjätyyppiin tai rooliin, joilla on tarkoitus kuvata eri käyttäjien oikeuksia tietokannan käsittelyyn. Näiden kolmen hakemiston dokumentit käyttävät niiden *Id*-arvona käyttäjäkohtaista *UID*-tunnuslukua, dokumentit eivät sisällä *Id*-arvon lisäksi muita muuttujia.

Admins-hakemisto kuvaa tietokannan kehittäjiä ja ylläpitäjiä, joilla täydet oikeudet muokata, luoda ja poistaa kaikkia tietokannan hakemistoja sekä dokumentteja. Tämän tyypin

käyttäjiä ei varsinaisesti tarvita tämän projektin aikana, mutta niitä tarvitaan sovelluksen jatkokehitysvaiheessa. Henkilölaskentajärjestelmää voitaisiin laajentaa tulevaisuudessa, esimerkiksi luomalla järjestelmän ylläpitäjille erillinen verkkokäyttöliittymä järjestelmäkokonaisuuden hallintaa varten.

Servers-hakemisto kuvaa liiketunnistus- ja kameralaitteita. Näillä laitteilla on ainoastaan pääsy ja oikeudet päivittää henkilölaskentaan liittyviä havaintotietoja.

Users-hakemisto kuvaa mobiilisovelluksen käyttäjiä. Mobiilisovelluksen käyttäjillä on oikeudet ainoastaan lukea liiketunnistimista saatua havaintotietoa, mutta heillä ei ole oikeutta muokata tai poistaa niitä.

Tietokannan käytölle voidaan asettaa sääntöjä (Rules), joilla voidaan rajoittaa sen käyttöä. Sääntöjen tarkoituksena on parantaa sovelluksen tietoturvaa ja estää tietyt toiminnot. Sääntöjen avulla voidaan esimerkiksi estää dokumenttien ja hakemistojen lisääminen, katselu, muokkaaminen tai poistaminen tietyiltä käyttäjiltä. Tietokanta voidaan myös asettaa edellyttämään aina käyttäjän kirjautumista ennen sen käsittelyä tai lukemista.

Firestore-tietokannan säännöt (Rules) määritellään sen omalla kielellä. Tietokannan säännöt määritellään hakemisto- ja dokumenttikohtaisesti. Säännöissä voidaan myös määritellä yksinkertaisia funktioita. (Firebase 2018h.) Kuvassa 7 on tämän projektin tietokannan säännöt.

```
service cloud.firestore {
  match /databases/{database}/documents {

    function isSignedIn() {
      return request.auth.uid != null;
    }

    function isAdmin() {
      return exists(/databases/$(database)/documents/admins/$(request.auth.uid));
    }

    function isServer() {
      return exists(/databases/$(database)/documents/servers/$(request.auth.uid))
    }

    function isUser() {
      return exists(/databases/$(database)/documents/users/$(request.auth.uid));
    }

    match /devices/{document=**} {
      allow read: if isSignedIn() && isUser();
      allow read, create: if isSignedIn() && isServer();
      allow read, write, create, delete: if isSignedIn() && isAdmin();
    }

    match /timestamps/{document=**} {
      allow read, write, create, delete: if isSignedIn() && isAdmin();
      allow read, write, create, delete: if isSignedIn() && isServer();
      allow read: if isSignedIn() && isUser();
    }

    match /admins/{document=**} {
      allow read, write, create, delete: if isSignedIn() && isAdmin();
    }

    match /servers/{document=**} {
      allow read, write, create, delete: if isSignedIn() && isAdmin();
    }

    match /users/{document=**} {
      allow read, write, create, delete: if isSignedIn() && isAdmin();
    }

  }

}
```

Kuva 7. Firestore-tietokannan säännöt (Rules).

Tietokannan säännöissä (Rules) on määritelty, että kaikkiin tietokannan toimintoihin edellytetään aina käyttäjän kirjautumista. Myös tietokannan *devices*- ja *timestamps*-hakemistojen dokumenttien katseluun edellytetään, että käyttäjän UID-merkkijonolla ni-

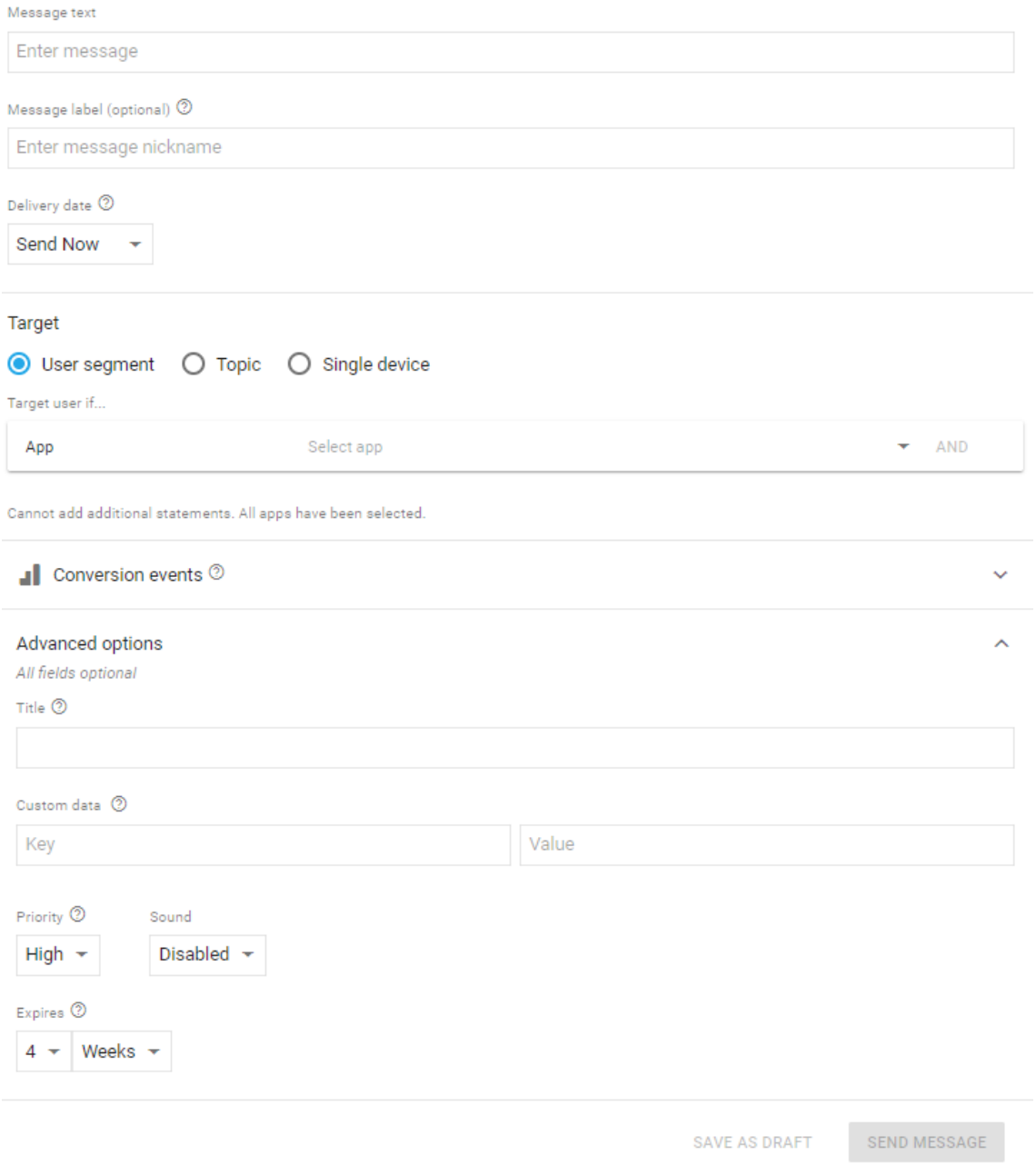
metty dokumentti on lisätty tietokannan *users*-hakemistoon. Tietokannan kaikkien muiden hakemistojen dokumenttien katselu, lisääminen, muokkaaminen ja poistaminen taas edellyttävät aina, että käyttäjä on lisätty joko *servers*- tai *admins*-hakemistoon.

3.5 Firebase Cloud Messaging

Firebase Cloud Messaging (FCM) tarkoituksena on mahdollistaa viestien (Notifications) lähetyksen ja vastaanoton eri sovellusten välillä reaaliajassa. Viestejä voi lähettää Firebase Consolesta sekä Firebase-projektin sovelluksista. Sovellukset voidaan asettaa kuuntelemaan viestejä, jolloin ne voidaan käsitellä heti niiden saapuessa. Viestejä voidaan käyttää esimerkiksi ilmoitusten tai datan lähettämiseen sovellusten välillä. (Firebase 2018c.)

Viestit sisältävät muun muassa otsikon, tekstin ja se voi sisältää dataa. Data lähetetään avain- ja arvopareina. Viestit voidaan lähettää heti tai ne voidaan asettaa lähetettäväksi tiettynä päivänä ja kellonaikana. Viestit voidaan kohdistaa tietyille vastaanottajille Firebase-projektin sovellusten tai aiheen (Topic) perusteella. Aiheen (Topic) avulla sovellus voidaan rajata vastaanottamaan vain tiettyjen aiheiden viestejä. (Firebase 2018g.) Kuvassa 8 on viestin lähetykseen tarkoitettu lomake Firebase Consolessa.

Firebase Cloud Messaging -palvelua tullaan käyttämään projektissa liiketunnistimen sekä mobiililaitteen väliseen yhteyteen Firebasen kautta. Tämä viestin lähetys ja vastaanottaminen kuvataan tarkemmin luvuissa 4.3 sekä 5.7.9.



Kuva 8. Firebase Cloud Messaging -viestin lähetys.

4 LIIKETUNNISTUS

Liiketunnistimena toimii kameralaite sekä siihen yhteydessä oleva tietokone. Kameralaite asennetaan tilassa sellaiseen paikka, mistä se voi kuvata kuvaa aluetta, jolta henkilömäärät halutaan laskea. Kameralaitteesta on yhteys tietokoneeseen, jossa kameralaitteen tuottama videokuva käsitellään ja siinä tapahtuva liike tunnistetaan sen omalla sovelluksella. Liiketunnistaminen kuvasta perustuu konenäköön ja sitä varten kehitetty sovellus on jo valmiiksi toimeksiantajan puolesta. Tämä sovellus on luotu Pythonilla ja käyttää OpenCV-ohjelmistokirjaston tarjoamia kuvankäsittely metodeja.

Tässä opinnäytetyössä keskitytään varsinaisen konenäkösovelluksen sijaan liiketunnistinlaitteiden tekemien liikehavaintojen lukemiseen ja niiden lähettämiseen eteenpäin edellisessä luvussa kuvattuun Firestore-tietokantaan. Liikehavaintotietojen lukemista ja tietokannan päivittämistä varten luodaan oma JavaScript-sovellus Node.js:n avulla, tämä sovellus kuvataan tässä luvussa.

4.1 Node.js -toteutus

Node.js on avoimeen lähdekoodiin perustuva tapahtumapohjainen JavaScriptin ajoympäristö, joka mahdollistaa palvelinsovellusten kehittämisen JavaScriptillä (Node.js 2018).

Node.js-sovellusten rakenne perustuu erilaisten moduulien käyttöön. Näiden moduulien voidaan ajatella olevan omia JavaScript-kirjastoja, jotka sisältävät metodeja erilaisiin tarkoituksiin. Node.js sisältää myös muutamia sisäänrakennettuja moduuleja valmiina sekä kehittäjät voivat myös luoda niitä itse. (W3Schools.com 2018a.) Node.js sisältää NPM-työkalun (Node Package Manager), jonka avulla voidaan hallita Node.js-sovellusten sisältämiä moduuleja ja niiden riippuvuuksia. Node.js-moduuleja voi hakea verkosta ja asentaa paikallisesti NPM -työkalun avulla. (W3Schools.com 2018b.)

4.2 Liikehavaintojen päivittäminen Firestore-tietokantaan

Tässä projektissa luodaan Node.js-ajoympäristössä suoritettava yksinkertainen komentorivikäyttöinen JavaScript-sovellus, jonka avulla voidaan kuunnella liiketunnistimien te-

kemiä havaintoja ja lähettää ne eteenpäin Firestore-tietokantaan. Kehitettävä JavaScript-sovellus voi kuunnella samanaikaisesti useita eri liiketunnistimia. Jokaisesta liikehavainnosta lähetään myös erillinen viesti järjestelmän mobiililaitteisiin luvussa 3.5 kuvatun Firebase Cloud Messaging -palvelun avulla.

JavaScript-sovellus koostuu kahdesta JavaScript-tiedostosta, *app.js* ja *config.js* sekä niiden tarvitsemista Node.js-moduuleista Firebase, Prompt ja ZeroMQ. Tiedosto *app.js* sisältää sovelluksen ohjelmakoodin ja *config.js* sisältää sovelluksen käyttämät asetukset, kuten verkko-osoitteet sekä sovelluksen yksilöivän API key -arvon, joiden avulla voidaan muodostaa yhteys Firebaseen.

Node.js-moduulit Firebase, Prompt ja ZeroMQ, ovat valmiiksi toteutettuja ja ne voidaan hakea verkosta NPM-työkalun avulla. Firebase-moduulia tarvitaan Node.js-sovelluksiin, joissa halutaan käsitellä Firebase-projektia ja sen Firestore-tietokantaa. Prompt-moduuli sisältää tarvittavat ominaisuudet komentorivikäyttöliittymän toteutukseen ja ZeroMQ-moduulin avulla voidaan kuunnella liiketunnistimien tekemiä havaintoja.

Liikehavaintojen kuuntelu toteutetaan ZeroMQ-moduulin avulla. ZeroMQ on sovellusten viestintään tarkoitettu ohjelmistokirjasto (ZeroMQ 2014). Liiketunnistimen havaitessa liikettä, se lähettää havaintotiedot SSH-tunnelin kautta kehitettävälle JavaScript-sovellus. Liikehavainnot lähetään liiketunnistimelta viesteinä JSON-muodossa avain- ja arvoparеina (Koodi 1). JavaScript-sovellus toimii näiden viestien vastaanottimena ZeroMQ-kirjaston avulla. Vastaanotettu havaintotieto puretaan JSON:ista ja lähetetään edelleen Firestore-tietokantaan.

```
{
  "id": "<unique-id-of-cameracounter>",
  "count": "<amount-of-people-passing-line>",
  "direction": "<in | out>",
  "timestamp": "<epoch timestamp in seconds>"
}
```

Koodi 1. Liikehavainto JSON-muodossa.

JavaScript-sovelluksen käyttö liikehavaintojen kuunteluun edellyttää aina käyttäjän kirjatumista. Käyttäjän kirjautumiseen käytetään luvussa 3.4 kuvattua *server*-tyypin käyttäjätunnusta. Käyttäjätunnusten syöttö tämän projektin JavaScript-sovelluksessa toteutetaan käyttämällä Prompt-moduulia, jonka avulla sovellus asetetaan kysymään käyttäjätunnusta ja salasanaa sen käynnistyessä. Käyttäjän syötettyä käyttäjätunnuksen ja salasanan. JavaScript-sovelluksen käyttämä Firebase-moduuli vastaa yhteydestä tietokantaan sekä käyttäjän autentikoinnista. JavaScript-sovellus myös sisältää kuuntelijametodin, jonka avulla voidaan seurata, onko käyttäjä kirjautuneena. Käyttäjän kirjautuessa ulos, JavaScript-sovellus pyytää käyttäjätunnukset uudelleen sekä katkaisee samalla myös yhteyden Firebaseen ja SSH-tunneliin.

4.3 Firebase Cloud Messaging -viestien lähetys

Kun JavaScript-sovellus vastaanottaa liiketunnistimen tekemän liikehavainnon, siitä lähetään myös erillinen Notification-viesti Firebase Cloud Messaging -palvelun avulla. Tämä viesti voidaan vastaanottaa luvussa 5 kuvattavassa mobiilisovelluksessa. Lähetettävä viesti sisältää muun muassa viestin aiheen (Topic), liikehavainnon tehneen liiketunnistinlaitteen Id-arvon, nimen sekä havaittujen henkilöiden lukumäärän. Viestin aihetta (Topic) voidaan käyttää rajaamaan viestien vastaanotto vain tietyille käyttäjille. Tässä projektissa kehitettävissä sovelluksissa viestin aiheella ei ole merkitystä, mutta sitä tarvitaan mahdollisesti sovelluksen jatkokehityksessä. Viestit lähetetään JavaScript-sovelluksesta Firebaseen HTTP-pyyntönä POST-metodilla.

5 MOBIILISOVELLUS

Mobiilisovellus toteutetaan Android-käyttöjärjestelmälle Android Studion avulla. Mobiilisovellus tukee Android 6.0 tai sitä uudempien käyttöjärjestelmäversioiden sisältämiä puhelin- ja tablettilaitteita.

5.1 Android Studio

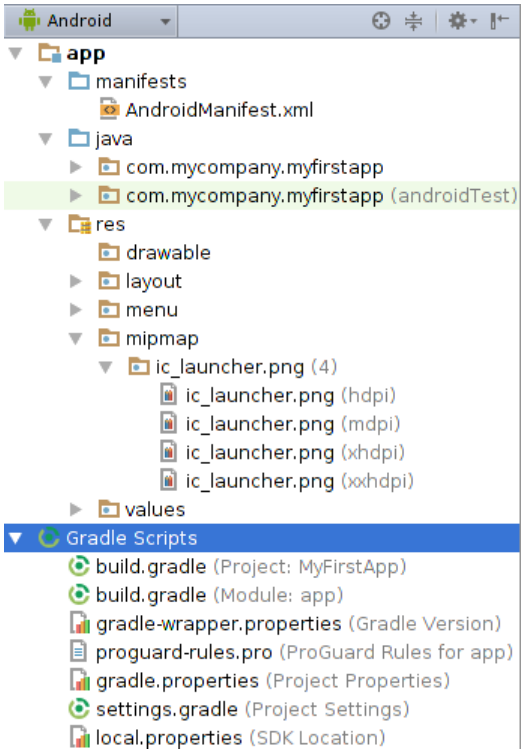
Android Studio on Googlen kehittämä Android-käyttöjärjestelmälle tarkoitettu ohjelmointiympäristö. Android Studio mahdollista Android-sovellusten kehittämisen Android-laitteille C++, Java tai Kotlin -ohjelmointikielellä. Android Studio sisältää myös emulaattorin, jonka avulla kehitettäviä mobiilisovelluksia voidaan testata tietokoneella oikean mobiililaitteen lisäksi. (Android Developers 2018d.)

5.2 Android-sovelluksen rakenne

Android-sovelluksen koodi koostuu komponenteista, jotka voidaan jakaa neljään eri tyyppiin. Nämä komponentit ovat Activities, Services, Broadcast receivers ja Content providers. (Android Developers 2018d.)

Android-sovellus jaetaan sen ohjelmakoodiin ja resursseihin. Resurssit ovat muun muassa Android-sovellukseen liittyviä kuva- teksti, ja käyttöliittymän ulkoasuun liittyviä tiedostoja. Nämä tiedostot kuvataan XML-kielellä. Sovelluksen käyttämät resurssit on koottu *Resources*-hakemiston alihakemistoihin. (Android Developers 2018d.)

Tässä projektissa kehitettävä sovellus kehitetään Java-kielellä ja tullaan käyttämään ainoastaan Activity- ja Service-tyypin komponentteja. Sovelluksen ohjelmakoodi kootaan yhteen Java-pakkauksiin ja sen sisältämiin alipakkauksiin. Kuvassa 9 on esimerkki Android-sovelluksen rakenteesta.



Kuva 9. Android-sovelluksen rakenne (Android Developers 2018i).

Android Studio käyttää Gradle-työkalua, joka vastaa ohjelmakoodin riippuvuuksista muihin kirjastoihin, ohjelmakoodin kääntämisestä sekä Android-sovelluksen tiedostojen paketoinnista APK-tiedostoiksi. Kun sovellus on pakattu APK-tiedostoksi, se voidaan asentaa Android-laitteeseen. (Android Developers 2018d.)

Tämän projektin aikana rakennettava Android-sovellus sisältää riippuvuudet muun muassa Joda-Time, Firebase Cloud Messaging, Firestore-tietokannan ja Firebasen autentikointikirjastoihin (Koodi 2). Firebasen Android-kirjastot mahdollistavat Firebasen käytön Android-sovelluksessa. Joda-Time-kirjasto sisältää päivämäärien ja kellonaikojen käsittelyyn tarkoitettuja metodeja, joilla voidaan esittää kävijämääriä eri ajanjaksoilta.

```
dependencies {
    implementation fileTree(include: ['*.jar'], dir: 'libs')
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support:design:26.1.0'
    implementation 'com.android.support:support-v4:26.1.0'
    implementation 'com.android.support:recyclerview-v7:26.1.0'
    implementation 'com.google.firebase:firebase-firestore:11.8.0'
    implementation 'com.google.firebase:firebase-auth:11.8.0'
    implementation 'com.google.firebase:firebase-messaging:11.8.0'
    implementation 'joda-time:joda-time:2.3'
}
```

Koodi 2. Android-sovelluksen riippuvuudet ohjelmistokirjastoihin.

5.3 Helppokäyttöisyys ja lokalisointi

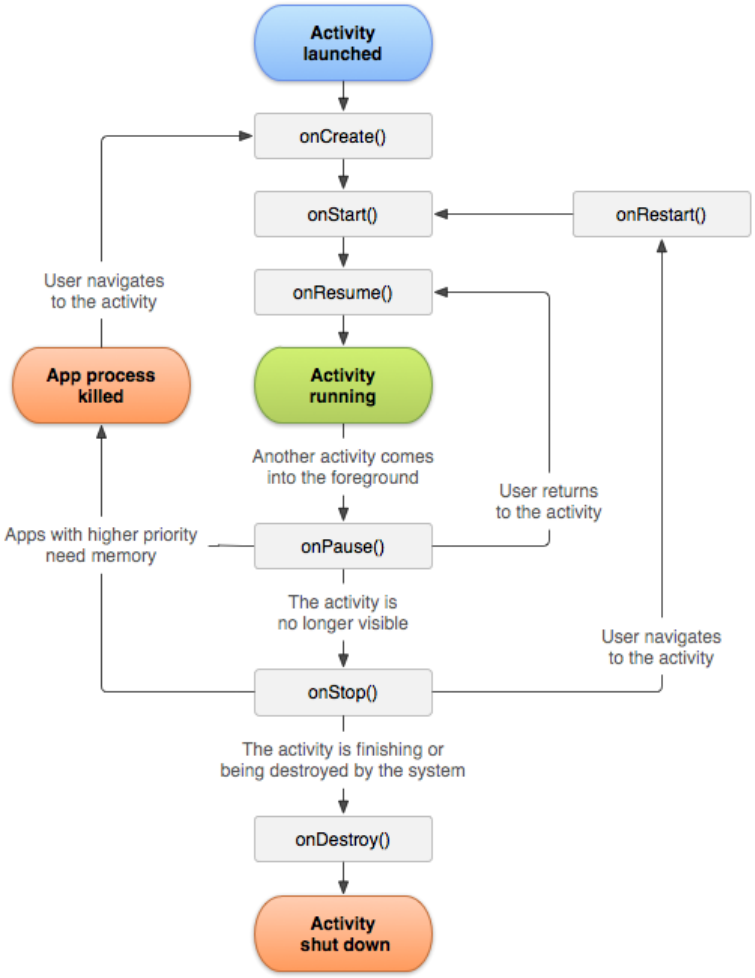
Helppokäyttöisyyden tarkoituksena on se, että mobiilisovellus on kaikkien käytettävissä toimintarajoitteista riippumatta. Esimerkiksi näkörajoitteisille voidaan kuvan tai tekstin sijaan käyttää ääntä. Android-sovelluksen ulkoasun värien valinta on myös hyvä huomioida jo sovelluskehityksen alkuvaiheessa. Android-käyttöjärjestelmän helppokäyttöisyysasetuksista voidaan Android-laite asettaa toistamaan ääni tekstin perusteella muuttamalla automaattisesti tekstin puheeksi. (Android Developers 2018a.)

Lokalisoinnin tarkoituksena on mahdollistaa mobiilisovelluksen käyttö kaikilla Android-käyttöjärjestelmän tukemilla kielillä (Android Developers 2018l). Android-sovelluksen käyttöliittymässä olevista teksteistä tai resursseista voidaan tehdä eri kielille omat versiot. (Android Developers 2018g.)

Tämän projektin aikana kehitettävä Android-sovelluksessa luodaan pääasiassa englannin- ja suomenkielisille käyttäjille, mutta sovelluksen lokalisointi muille kielille on hyvä huomioida sovelluksen mahdollista jatkokehitystä varten.

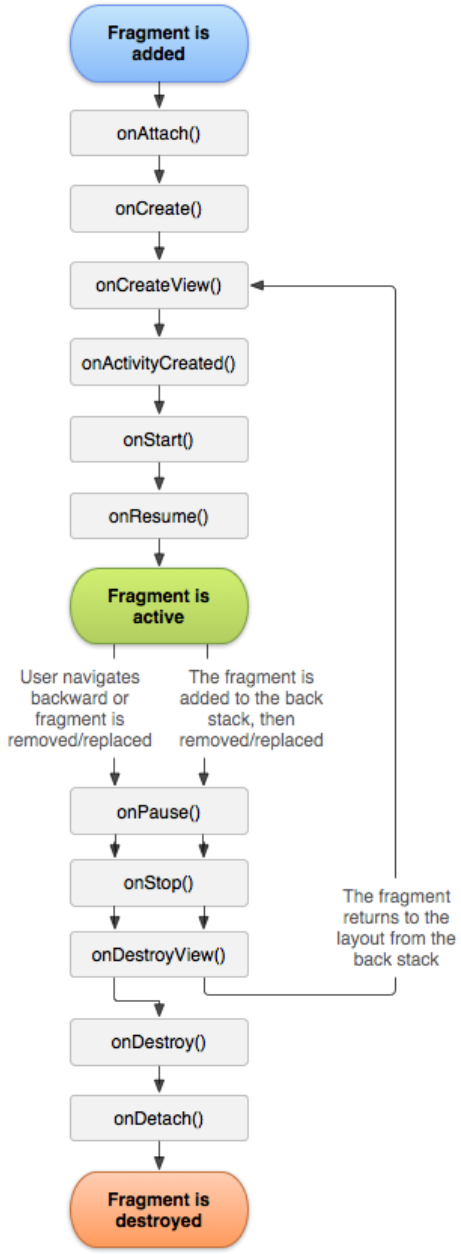
5.4 Activity ja Fragment

Activity vastaa yksittäisen näkymän luomisesta Android-laitteen näytölle. Android-sovellus koostuu yleensä useasta Activity-komponenttien luomasta näkymästä ja sovelluksen käyttö perustuu yleensä navigointiin eri näkymien välillä. Activityllä on aina tietty elinkaari. Activity sisältää callback-metodeja, joilla voidaan määrittään Activityn toiminta sen eri tiloissa ja sen elinkaaren eri vaihessa. Activity sisältää kuusi callback-metodia, *onCreate(), onStart()*, *onResume()*, *onPause()*, *onStop()* ja *onDestroy()*. Nämä suoritetaan aina Activityn tilan muuttuessa, kun käyttäjä esimerkiksi avaa näkymän poistuu näkymästä tai käynnistää sovelluksen uudelleen (Kuva 10). (Android Developers 2018b.)



Kuva 10. Activityn elinkaari (Android Developers 2018m).

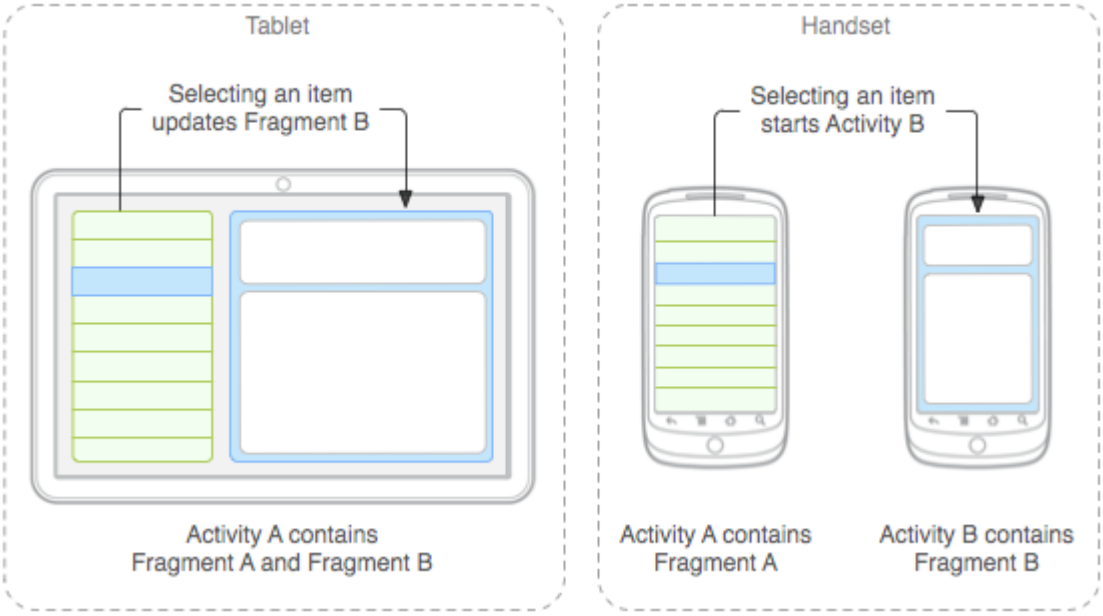
Fragmentin voidaan ajatella olevan erillinen modulaarinen näkymä, joka voidaan liittää osaksi jotain toista näkymää. Fragmentien avulla jokin tietty näkymä voidaan koota näistä eri moduuleista. Fragmentin tarkoituksena on voida toimia itsenäisenä osana näkymää. Fragmentilla on oma elinkaari ja se sisältää callback-metodeja, jotka suoritetaan sen tilan muuttuessa samalla tavalla kuin Activityllä (Kuva 11). (Android Developers 2018e.)



Kuva 11. Fragmentin elinkaari (Android Developers 2018e).

Fragment sijoitetaan tai luodaan aina Activityyn ja Activity voi sisältää useamman Fragmentin. Samaa Fragmentia voidaan myös käyttää useassa eri Activityssä. Fragmenttien avulla Android-laitteen näytöllä voidaan esittää useita eri näkymiä saman aikaisesti riippuen laitteen näytön koosta. Esimerkiksi tabletti- ja puhelinlaitteilla on erikokoinen näyttö ja Fragmenttien avulla voidaan esittää kaksi eri näkymää saman aikaisesti, tätä kutsutaan Master/Detail-näkymäksi.

Master/Detail-näkymässä käyttäjä valitsee listasta jonkin aiheen ja tabletti näyttää tästä tarkemmat tiedot viereisessä näkymässä. Molemmat kuvat voidaan näyttää laitteen näytöllä samanaikaisesti (Kuva 12). (Android Developers 2018f.)



Kuva 12. Master/Detail-näkymä (Android Developers 2018e).

Projektin aikana kehitettävässä Android-sovelluksessa tullaan käyttämään Fragmentia näkymän luontiin sovelluksen asetusnäkymässä, joka kuvataan luvussa 5.7.8.

5.5 Android Manifest

Jokainen Android-sovellus sisältää *AndroidManifest.xml*-tiedoston. *AndroidManifest.xml*-tiedostossa määritellään sovelluksen pakettinimen, eri komponentit, sovelluksen yksilöivän ID-arvon, sovelluksen käyttämät eri palvelut (Service) sekä käynnistys-Activityn, jonka näkymä näytetään käyttäjälle ensimmäisenä sovelluksen käynnistyessä. (Android Developers 2018c.)

Android-käyttöjärjestelmä vaatii Android-laitteen tiettyjen ominaisuuksien käyttämiseen aina käyttäjän luvan. Sovelluksen tarvitsemat ominaisuudet, joihin tarvitaan käyttäjän lupa pitää määritellä *AndroidManifest.xml*-tiedostossa. Sovellus tarvitsee käyttäjän luvan muun muassa verkkoyhteyden käyttöön tai pääsyyn Android-käyttöjärjestelmän sisältä-

miin eri tiedostoihin ja hakemistoihin. Android-käyttöjärjestelmä vastaa aina luvan pyytämisestä käyttäjältä ennen lupaa edellyttävän toiminnon suorittamista. (Android Developers 2018c.)

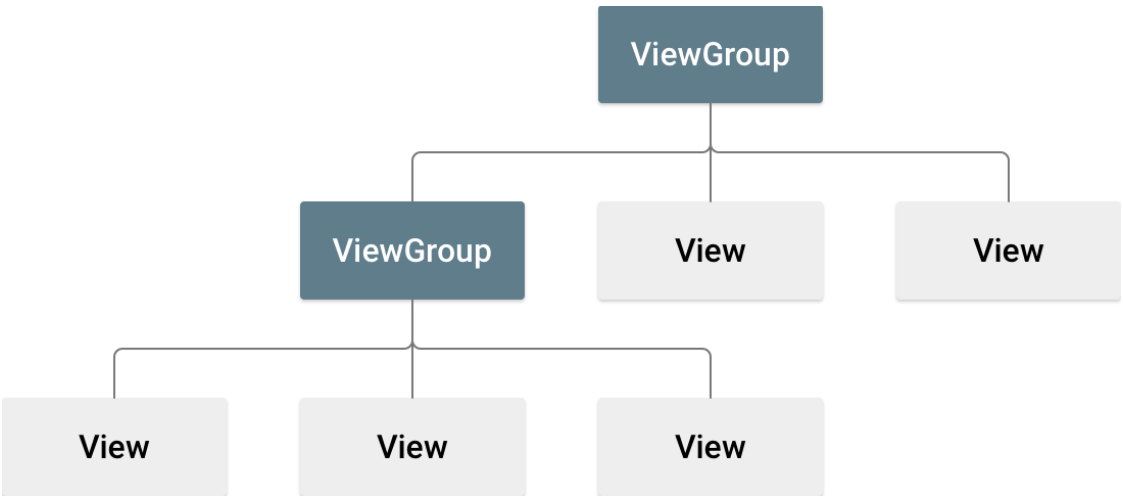
Tässä projektissa kehitettävä Android-sovellus tarvitsee käyttäjältä luvat mobiililaitteen verkkoyhteyden käyttöön, verkkoyhteyden tilan tarkasteluun sekä värinähälytyksen käyttöön (Koodi 3).

```
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.VIBRATE" />
```

Koodi 3. Sovelluksen luvat.

5.6 Android-sovelluksen käyttöliittymä ja ulkoasu

Android-sovelluksen näkymien ulkoasu koostuu erilaisista ennalta määritellyistä UI-komponenteista, kuten tekstikentistä tai näppäimistä. Näkymien ulkoasu perustuu *View*- ja *ViewGroup*-objektien hierarkiaan. Jokainen sovelluksen UI-komponentti perii *View*-luokan, jonkin *View*-luokan aliluokan tai *ViewGroup*-luokan. *ViewGroup*-luokan objektit perivät Layout-luokan ja niiden tarkoituksena on koota *View*-objektit yhdeksi kokonaisuudeksi. *ViewGroup* määrittää näkymän rakenteen ja *View*-objektien sijainnin sovelluksen käyttöliittymässä (Kuva 13). (Android Developers 2018f.)



Kuva 13. View- ja ViewGroup-objektien hierarkia (Android Developers 2018f).

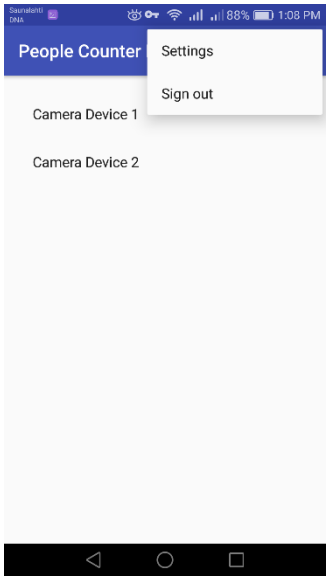
Näkymien ulkoasun rakenne kuvataan XML-kielellä ja jokaista Androidin UI-komponenttia varten on olemassa ennalta oma XML-elementti. XML-elementeillä on tietyt attribuutit,

jotka määrittävät sitä vastaavan UI-komponentin ulkoasun, toiminnan sekä UI-komponentin yksilöivän Id-arvon. UI-komponentteihin voidaan viitata ohjelmakoodista niiden Id-arvon avulla sekä niiden toimintaa tai ulkoasua voidaan myös muokata XML-kielen lisäksi ohjelmakoodista. Android-sovelluksen ulkoasuun liittyvät XML-tiedostot sijaitsevat *resources*-hakemiston *drawable*-, *layout*-, *menu*- ja *xml*-alihakemistoissa. (Android Developers 2018j.)

Kaikki Android-sovelluksen käyttämät animaatio- ja kuvatiedostot säilytetään *drawable*-hakemistossa. Tässä projektissa ei käytetä Android-sovelluksen oletuspikakuvakkeiden lisäksi muita kuvatiedostoja.

Android-sovelluksen näkymien ulkoasun rakenteeseen liittyvät tiedostot ovat *layout*-hakemistossa. Samasta näkymästä voi olla myös olemassa eri kokoisille näytöille, kuten puhelimelle ja tabletille omat versiot. Tämän tarkoituksena on saada näkymät sopimaan laitteiden eri kokoisiin näyttöihin. (Android Developers 2018f.)

Android-sovelluksen sisältämät valikot kuvataan *menu.xml* -tiedostossa (Android Developers 2018j). Valikoista voi valita eri toimintoja tai siirtyä eri näkymiin. Tässä projektissa kehitettävän Android-sovelluksen oikeassa yläkulmassa on valikko, josta käyttäjän on mahdollista kirjautua ulos tai siirtyä sovelluksen asetuksiin (Kuva 14).



Kuva 14. Android-sovelluksen valikko.

Android-sovelluksen asetusnäkymään liittyvät tiedostot sijaitsevat *xml*-hakemistossa. Asetusnäkymä kuvataan luvussa 5.7.8.

Android-sovelluksen käyttämät avain- ja arvopariresurssit, kuten merkkijonot, taulukot, sekä UI-komponenttien tyylit ja värit määritellään myös XML-kielellä. Nämä avain- ja arvoparit sisältävät tiedostot sijaitsevat *Resources*-hakemiston *values*-alihakemistossa tai sen alihakemistoissa. Projektissa kehitettävä Android-sovellus tarvitsee XML-tiedostot *arrays*, *colors*, *dimens*, *strings*, *styles* ja *strings*,

Arrays.xml-tiedostossa määritellään taulukkotyyppiset muuttujat. Sovelluksen käyttämät taulukot voidaan näin erottaa ohjelmakoodista. Tämä helpottaa ylläpitoa. Merkkijonotyyppisistä taulukoista voidaan myös tehdä eri kielille omat versiot sovelluksen lokalisointia varten tekemällä *arrays*.xml-tiedostosta kopiot eri kielille.

Colors.xml-tiedostossa määritellään Android-sovelluksen ulkoasussa käytettävien värikoodien muuttujat (Android Developers 2018j). Android-sovelluksen värit kuvataan HEX-värikoodilla.

Dimens.*xml*-tiedostossa voidaan määritellä Android-sovelluksen käyttämien UI-komponenttien mittasuhteet. Mittasuhteet voidaan määritellä esimerkiksi pikselimäärissä tai niiden fyysisessä koossa laitteen Android-laitteen näytöllä. (Android Developers 2018j.)

Strings.*xml*-tiedostoon voidaan tallentaa merkkijonotyyppisten muuttujien arvot. Tällä tavoin sovelluksen käyttöliittymässä käytettävien merkkijonojen arvoihin voidaan viitata ohjelmakoodista. Tämä helpottaa ohjelmakoodin ylläpitoa ja kaikki sovelluksen merkkijonot ovat yhdessä paikassa erossa varsinaisesta ohjelmakoodista. Tämä myös mahdollistaa Android-sovelluksen lokalisoinnin. Lokalisointia varten *strings*.xml -tiedostosta voidaan luoda omat versiot eri kielille ja ne sijoitetaan kielten omiin hakemistoihin. (Android Developers 2018g.)

Styles.*xml*-tiedostossa voidaan määritellä Android-sovelluksen UI-komponenttien käyttämä formaatti ja tyyli. Tyyli voidaan määritellä XML-koodissa yksittäiselle *View*-elementille tai kokonaiselle *Layoutille*. (Android Developers 2018j.)

5.7 Android-sovelluksen Java-luokat

Projektin Android-sovellus sisältää seuraavat Java-luokat: *Device*, *Timestamp*, *UserAuthStateListenerActivity*, *LoginActivity*, *DeviceListActivity*, *DeviceDetailActivity*, *DeviceDetailFragment*, *SettingsActivity* sekä *Utilities*. Luokat *Device* ja *Timestamp* on

sijoitettu omaan Java-pakettiin *document*, jotta niiden nimet eivät sekoitu Javan jo olemassa olevien luokkien nimien kanssa.

5.7.1 Device-luokka

Device-luokassa määritellään *Device*-objekti (Koodi 4). Tämä *Device*-objekti vastaa aiemmin kuvatun Firestore-tietokannan *devices*-hakemiston yksittäistä dokumenttia eli liiketunnistinlaitetta, joka lähettää tekemänsä havainnot Firestore-tietokantaan. *Device*-luokka sisältää vain muuttujat *deviceId* ja *name*, nämä kuvaavat yksittäisen liiketunnistimen yksilöllistä Id-tunnistelukua ja laitteen nimeä.

```
public class Device {

    private final String id;

    private final String name;

    public Device(String id, String name) {
        this.id = id;
        this.name = name;
    }

    public String getId() {
        return id;
    }

    public String getName() {
        return name;
    }

}
```

Koodi 4. Device-luokka.

5.7.2 Timestamp-luokka

Timestamp-luokassa määritellään *Timestamp*-objekti (Koodi 5). Tämä objekti vastaa *devices*-hakemiston dokumenttien *timestamps*-alihakemiston dokumentteja eli yksittäisiä liiketunnistimen tekemiä liikehavaintoja. *Timestamp*-hakemisto sisältää muuttujat *count*, *direction* ja *timestamp*. Nämä muuttujat kuvaavat laskettujen henkilöiden lukumäärää, liikesuuntaa, kellonaikaa ja päivämäärää.

```
public class Timestamp {

    private final Date timestamp;

    private final int count;

    private final String direction;
```

```
    public Timestamp(Date timestamp, int count, String direction) {
        this.timestamp = timestamp;
        this.count = count;
        this.direction = direction;
    }

    public Date getTimestamp() {
        return timestamp;
    }

    public int getCount() {
        return count;
    }

    public String getDirection() {
        return direction;
    }

}
```

Koodi 5. Timestamp-luokka.

5.7.3 Utilities-luokka

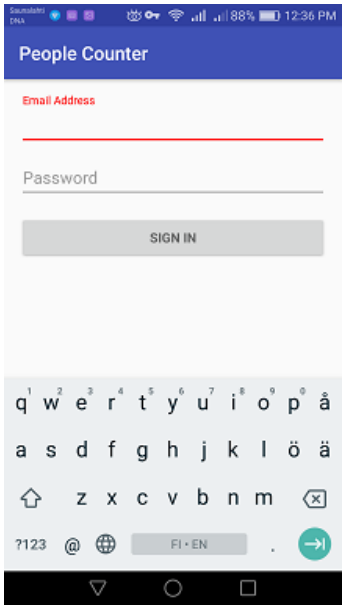
Utilities-luokka sisältää eri apumetodeja. Metodit ovat julkisia ja staattisia ja on tarkoitus voida käyttää useammassa eri luokassa. Esimerkiksi metodin *isConnection* (Koodi 6) avulla voidaan tarkistaa verkkoyhteyden tila ja tätä samaa metodia voidaan käyttää kaikissa sovelluksen eri luokissa.

```
public static boolean isConnection(Context context) {
    ConnectivityManager connMgr = (ConnectivityManager)
            context.getSystemService(Context.CONNECTIVITY_SERVICE);
    NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
    return (networkInfo != null && networkInfo.isConnected());
}
```

Koodi 6. Utilities-luokka, verkkoyhteyden tilan tarkistava metodi.

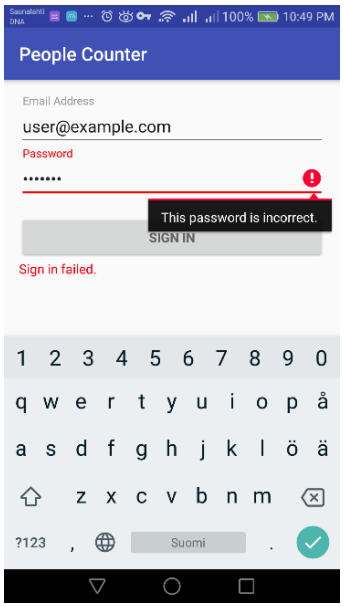
5.7.4 LoginActivity-luokka

Projektissa kehitettävän Android-sovelluksen käyttö edellyttää aina käyttäjän kirjautumista. *LoginActivityn* näkymä on Android-sovelluksen käynnistyessä sen ensimmäinen näkymä. *LoginActivity* vastaa käyttäjän kirjautumisesta sovellukseen. *LoginActivityn* luomalla näkymällä on kirjautumislomake, johon käyttäjä syöttää käyttäjätunnuksen ja salasanan (Kuva 15).



Kuva 15. LoginActivity ja sen näkymä Android-laitteessa.

Käyttäjätunnuksen tai salasanan ollessa virheellinen näytetään virheteksti, jossa kerrotaan virheen syy (Kuva 16). Kirjautumisen onnistuttua, siirtyy sovellus seuraavaan näkymään. Sovellus tallentaa myös käyttäjän kirjautumistiedot, jotta sovellukseen ei tarvitse kirjautua jokaisella käyttökerralla, kun sovellus käynnistetään. Käyttäjän ollessa jo kirjautunut sovellus siirtyy automattisesti seuraavaan näkymään näyttämättä kirjautumislomaketta.



Kuva 16. LoginActivity ja virheilmoitus Android-laitteessa.

5.7.5 UserAuthStateListener-luokka

UserAuthStateListenerActivity toimii *DeviceListActivity*-, *DeviceDetailActivity*- ja *SettingsActivity*-luokkien yläluokkana. *UserAuthStateListenerActivity*-luokka määrittelee kuuntelijametodin, joka periytyy sen aliluokille. Tämän kuuntelijametodi seuraa sovellukseen kirjautuneen käyttäjän kirjautumistilaa (Koodi 7).

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mFirebaseAuth = FirebaseAuth.getInstance();
    mFirebaseAuth.addAuthStateListener(new FirebaseAuth.AuthStateListener() {
        @Override
        public void onAuthStateChanged(@NonNull FirebaseAuth firebaseAuth) {
            if (firebaseAuth.getCurrentUser() == null) {
                goBackToLoginActivity();
            }
        }
    });
}

protected void goBackToLoginActivity() {
    FragmentManager fm = getSupportFragmentManager();
    for(int i = 0; i < fm.getBackStackEntryCount(); ++i) {
        fm.popBackStack();
    }
    Intent intent = new Intent(getApplicationContext(), LoginActivity.class);
    intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIV-
ITY_CLEAR_TASK | Intent.FLAG_ACTIVITY_NEW_TASK);
    startActivity(intent);
    finish();
}
```

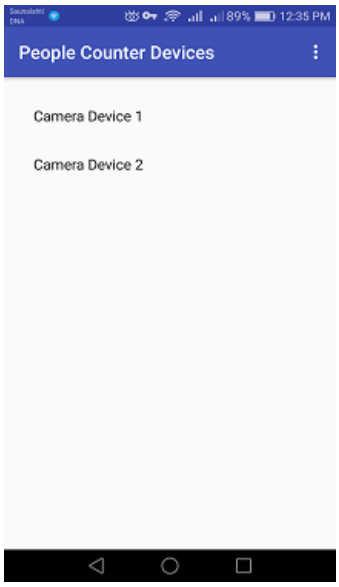
Koodi 7. UserAuthStateActivity ja käyttäjän kirjautumistilan seuranta.

Tämän kuuntelijametodin tarkoituksena on varmistaa, että sovelluksen tiettyjä näkymiä ei näytetä käyttäjälle ilman käyttäjän kirjautumista oikeilla käyttäjätunnuksilla. Mikäli luokan kuuntelijametodi havaitsee, että käyttäjä ei ole enää kirjautunut sovellukseen, siirtyy sovellus takaisin *LoginActivity*-luokan näkymään.

5.7.6 DeviceListActivity-luokka

DeviceListActivity-luokan tarkoituksena on luoda näkymä, joka näyttää listan henkilölaskentajärjestelmän kaikista liiketunnistimista. *DeviceListActivity* hakee Firestore-tietokannasta *devices*-hakemiston dokumentit ja ne järjestetään aakkosjärjestykseen niiden nimen mukaan (Kuva 17). *DeviceListActivity* seuraa Firestore-tietokannassa tapahtuvia muutoksia reaaliajassa ja päivittää automaattisesti Firestore-tietokannan *devices*-hakemistossa tapahtuvat muutokset listaan. Käyttäjä valitsee listasta liiketunnistimen, jonka

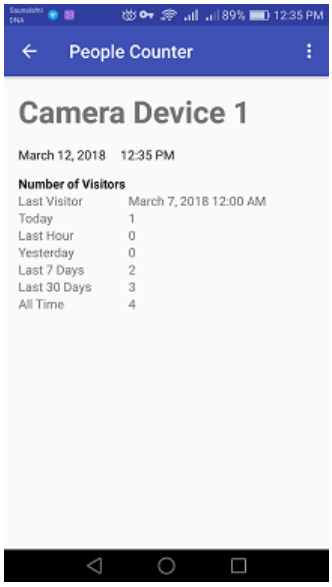
tarkempia tietoja käyttäjä haluaa tarkastella. Tällöin sovellus näyttää kyseisen liiketunnistimen yksityiskohtaiset tiedot ja sen tekemät kävijähavainnot eri ajanjaksoina. Kameralaitteen yksityiskohtaisemmat tiedot esitetään *DeviceDetailsActivity*-luokan näkymässä, joka kuvataan luvussa 5.7.7.



Kuva 17. DeviceListActivity ja sen näkymä Android-laitteessa.

5.7.7 DeviceDetailActivity-luokka

Käyttäjän valitessa liiketunnistimen *DeviceListActivityn* näkymästä, sovellus siirtyy *DeviceDetailActivityn* luomaan näkymään. *DeviceDetaiActivityn* näkymässä esitetään liiketunnistimen nimi sekä viime aikojen lasketut kävijämäärät eri ajanjaksoina (Kuva 18).

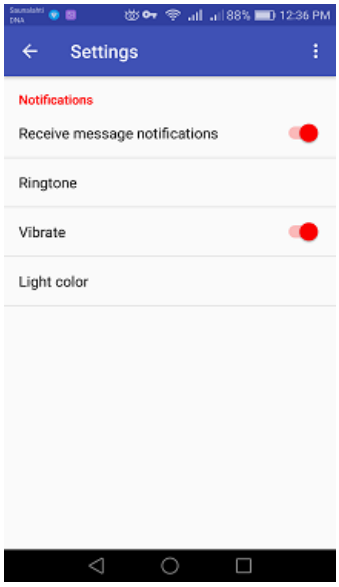


Kuva 18. DeviceDetailActivity ja sen näkymä Android-laitteessa.

DeviceDetailActivity hakee liiketunnistinta vastaavan dokumentin Firestore *devices*-hakemistosta sekä sen sisältämän *timestamps*-alihakemiston dokumentit. *DeviceDetailActivity* myös kuuntelee liiketunnistinta vastaavan dokumentin muutoksia ja päivittää muutokset sen tilassa ja lasketuissa kävijämäärissä automaattisesti näkymään. Tällä tavoin kävijämääriä voidaan seurata reaaliajassa Android-sovelluksella.

5.7.8 SettingsActivity-luokka ja Android-sovelluksen asetukset

SettingsActivity-luokan tarkoituksena vastata Android-sovelluksen asetusnäkymästä. *SettingsActivity* luo näkymän, josta käyttäjä voi muokata sovellukseen liittyviä asetuksia. Sovelluksen asetuksista Android-sovellus voidaan asettaa hälyttämään käyttäjää aina kun jokin järjestelmän liiketunnistin havaitsee liikettä. Ilmoitukset käyttävät Androidin Push Nofification -ominaisuutta. Käyttäjä voi myös asettaa ilmoitukset käyttämään värinää, merkkiääntä tai merkkivaloa (Kuva 19). Push Notification -ominaisuus kuvataan tarkemmin luvussa 5.7.9.



Kuva 19. SettingsActivity ja sen näkymä Android-laitteessa.

SettingsActivity-luokka sisältää sisäisen luokan *SettingsFragment*. *SettingsFragment* perii *PreferenceFragment*-luokan ja se luo näkymän, josta Android-sovelluksen asetuksia voidaan muokata.

PreferenceFragment-luokan perivät aliluokat poikkeavat muista Activityistä ja Fragmenteista siten, että *PreferenceFragment*-luokka tarkoitettu erityisesti Android-sovelluksen asetusten ja asetusnäkymää varten. Android-sovelluksen asetuksia varten on Androidissa olemassa omat *View*-objektit ja *PreferenceFragment*-luokka sisältää näille tarkoitettuja metodeja. Sovelluksen asetukset voidaan lukea ja tallentaa *SharedPreferences*-objektin avulla. *SharedPrefences*-objekti on tarkoitettu primitiivityppisten arvojen ja merkkijonojen avain- ja arvoparien pysyvämpään tallentamiseen ja sitä käytetään myös Android-sovelluksen asetuksia varten. (Android Developers 2018k.)

5.7.9 Firebase Cloud Messaging -viestien vastaanotto Android-laitteella

Android-sovellus voidaan asettaa hälyttämään käyttäjää aina kun jokin liiketunnistin havaitsee liikettä. Tämä voidaan toteuttaa Firebase Cloud Messaging -palvelun avulla. Liiketunnistimeen yhteydessä oleva luvussa 4 kuvattu JavaScript-sovellus voi lähettää Firebase Cloud Messaging -palvelun avulla viestin kaikkiin järjestelmän Android-laitteisiin.

Viestien vastaanottamiseen määritellään kaksi luokkaa, *PeopleCounterInstanceIDService* ja *PeopleCounterMessagingService*. Nämä luokat ja lisätään *AndroidManifest.xml*-tiedostoon Service-komponentteina, jotta viestien vastaanotto voisi toimia (Koodi 9) (Firebase 2018f).

```
<service android:name=".PeopleCounterMessagingService">
    <intent-filter>
        <action android:name="com.google.firebase.MESSAGING_EVENT" />
    </intent-filter>
</service>
<service android:name=".PeopleCounterInstanceIDService">
    <intent-filter>
        <action android:name="com.google.firebase.INSTANCE_ID_EVENT" />
    </intent-filter>
</service>
```

Koodi 8. AndroidManifest.xml Service-komponentit.

PeopleCounterInstanceIDService- ja *PeopleCounterMessagingService*-luokat mahdollistavat viestin vastaanoton ja käsittelyn. *PeopleCounterMessagingService*-luokassa viesti puretaan ja käsitellään. Vastaanotettua viestin, *PeopleCounterMessagingService*-luokka luo ja näyttää ilmoituksen käyttämällä Androidin Push Notification -ominaisuutta. Push Notification voi sisältää myös merkkiäänen, värinän tai merkkivalon, jos käyttäjä on ottanut ne käyttöön edellisessä luvussa kuvatussa sovelluksen asetusnäkymässä.

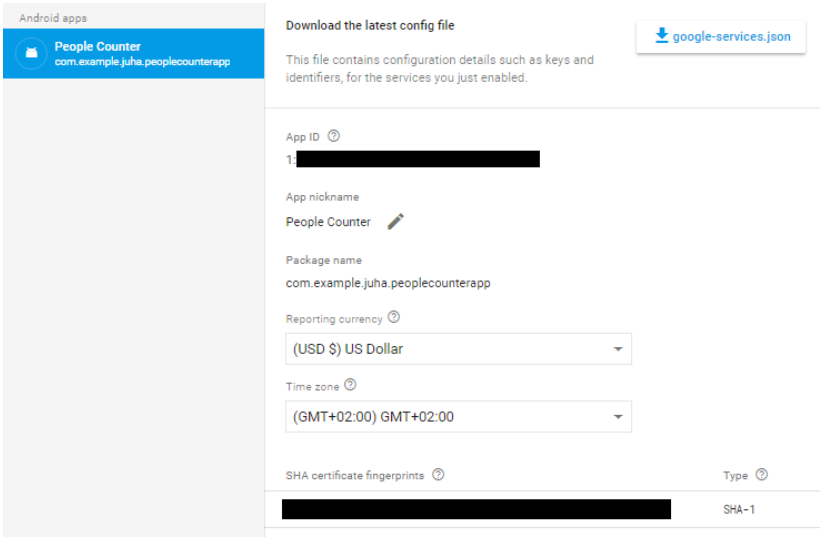
5.8 Android-sovelluksen viimeistely julkaisua varten

Android-sovelluksen julkaisun valmistelussa on tarkoitus saada sovellus sellaiseen tilaan, että se voidaan ottaa laajempaan käyttöön. Sovellus testataan huolellisesti ja varmistetaan että sovellus ja sen käyttämät ulkopuoliset verkkopalvelut tai sovellukset ovat käyttäjälle tietoturvallisia. Android-projektin tiedostot ja paketit nimetään oikein ja Java-kielen käytäntöjen mukaisesti. Android-projektin turhat tiedostot ja hakemistot poistetaan ja varmistetaan, että projektin rakenne noudattaa Android-sovellusten käytäntöjä. Android-sovelluksen versionumero tulee myös olla oikein. Android-sovelluksen versionumero määritellään sovelluksen *build.gradle*-tiedostossa. (Android Developers 2018h.)

Tässä projektissa sovellusta ei valmisteta kokonaan julkaisuvaiheeseen vaan se jää vielä toimeksiantajayrityksen kehitettäväksi tulevia asiakasprojekteja varten.

Firebasea käyttävien sovellusten julkaisua varten on myös ladattava Firebase Consolen Settings-verkkosivulta Firebase-projektille luotava yksilöllinen *google-services.json*-tie-

dosto (Kuva 20). Tämä tiedosto liitetään projektin juurihakemistoon ja se sisältää projektin Id-luvun, nimen, API-keyn, asetukset sekä kaikki muut tarvittavat tunnusluvut, joita sovellus tarvitsee toimiakseen ja voidakseen ottaa yhteyden projektin Firebaseen ja sen Firestore-tietokantaan. (Firebase 2018c.)



Kuva 20. Firebase Console ja google-services.json-tiedoston luonti.

6 YHTEENVETO

Tämän opinnäytetyön tavoitteena oli luoda pilvipohjainen henkilölaskentajärjestelmä konenäköpohjaiselle liiketunnistimelle. Opinnäytetyön aikana luotu henkilölaskentajärjestelmä koostuu liiketunnistimesta, pilvialustalla toimivasta tietokannasta sekä mobiilisovelluksesta, jonka avulla voidaan näyttää laskettujen henkilöiden määrä.

Opinnäytetyö toimeksiantajayrityksen puolesta onnistuneeksi. Opinnäytetyö oli kokonaisuudessaan varsin laaja mutta, sovellusten lähes kaikki ominaisuudet toimivat oikein odotusten mukaisesti. Ainoastaan Android-sovelluksen Push Notification -ominaisuutta ei saatu täysin toimimaan. Sovellus vastaanottaa Firebase Cloud Messaging -palvelun kautta lähetetyn viestin ja näyttää sen Android-laitteen Notifications-listassa, mutta merkkiääntä, värinää ja merkkivaloa ei saatu toimimaan testilaitteella. Opinnäytetyön aikataulun puitteissa ei saatu selville, johtuiko ongelma testilaitteen asetuksista vai sovelluksen ohjelmakoodista, joten tämän ominaisuus kehittäminen päätettiin toimeksiantajan kanssa jättää pois.

Toinen Android-sovelluksessa kehittämistä vaativa ominaisuus liittyy ulkoasuun. Tällä hetkellä sovellus ei tue Master/Detail-layoutia, kuten luvussa 5.4 on kuvattu. Liiketunnistimien lista ja liiketunnistimen laskentatietojen näkymät näytetään omissa Activityissä, eivätkä Fragmenteissa. Käyttämällä Fragmentteja molemmat näkymät voitaisiin näyttää tablettilaitteella samanaikaisesti. Tämä ominaisuus oli tarkoitus kehittää, mutta jätettiin pois Fragmenttien aiheuttamien ongelmien takia. Käyttäjän kirjautuessa sovelluksesta ulos Activityn sisäiset Fragmentit kaatoivat sovelluksen. Syytä tähän ei saatu selville.

Firestore-tietokannan käytön sääntöjä voisi myös kehittää tulevaisuudessa. Tämä johtuu siitä, että Firestore-tietokanta on tällä hetkellä vielä beta-testausvaiheessa eikä sisällä vielä kaikkia ominaisuuksia, muun muassa dokumenttien taulukkotyyppisten muuttujien käsittely on tällä hetkellä puutteellista. Tällä hetkellä järjestelmän liiketunnistimet ja niiden havainnot näkyvät kaikille järjestelmän Android-sovellusten luvussa 3.4 kuvatun *users*-hakemistoon lisätyille käyttäjille. Tulevaisuudessa tätä voitaisiin muuttaa siten, että kameralaitteiden ja niiden tekemien havaintojen näkyminen voitaisiin rajata vain tietyille käyttäjille.

Opinnäytetyön aikana valmistunut henkilölaskentajärjestelmä toimii hyvänä pohjana jatkokehitykselle ja sitä voidaan käyttää myös jatkossa vastaavanlaisten sovellusratkaisujen kehittämiseen.

LÄHTEET

Android Developers 2018a. Accessibility. Viitattu 23.2.2018. https://developer.android.com/guide/topics/ui/accessibility/index.html

Android Developers 2018b. Activity. Viitattu 25.2.2018. https://developer.android.com/reference/android/app/Activity.html

Android Developers 2018c. App Manifest Overview. Viitattu 26.2.2018. https://developer.android.com/guide/topics/manifest/manifest-intro.html

Android Developers 2018d. Application Fundamentals. Viitattu 25.2.2018. https://developer.android.com/guide/components/fundamentals.html

Android Developers 2018e. Fragments. Viitattu 25.2.2018. https://developer.android.com/guide/components/fragments.html

Android Developers 2018f. Layout. Viitattu 25.2.2018. https://developer.android.com/guide/topics/ui/declaring-layout.html

Android Developers 2018g. Localizing with Resources. Viitattu 23.2.2018. https://developer.android.com/guide/topics/resources/localization.html#managing-strings-for-localization

Android Developers 2018h. Preparing for Release. Viitattu 9.3.2018. https://developer.android.com/studio/publish/preparing.html

Android Developers 2018i. Projects Overview. Viitattu 16.3.2018. https://developer.android.com/studio/projects/index.html

Android Developers 2018j. Resource Types. Viitattu 13.3.2018. https://developer.android.com/guide/topics/resources/available-resources.html

Android Developers 2018k. Save Key-Value Data with SharedPreferences. Viitattu 10.3.2018. https://developer.android.com/training/data-storage/shared-preferences.html

Android Developers 2018l. Supporting Different Languages and Cultures. Viitattu 23.2.2018. https://developer.android.com/training/basics/supporting-devices/languages.html

Android Developers 2018m. The Activity Lifecycle. Viitattu 9.3.2018. https://developer.android.com/guide/components/activities/activity-lifecycle.html

Fidera 2018. Fidera. Viitattu 29.3.2018. http://fidera.fi

Firebase 2018a. Cloud Firestore Data Model. Viitattu 16.3.2018. https://firebase.google.com/docs/firestore/data-model

Firebase 2018b. Firebase Authentication. Viitattu 16.3.2018. https://firebase.google.com/docs/auth/

Firebase 2018c. Firebase Cloud Messaging. Viitattu 9.3.2018. https://firebase.google.com/docs/cloud-messaging/

Firebase 2018d. Firebase. Viitattu 2.3.2018. https://firebase.google.com/

Firebase 2018e. Firestore. Viitattu 2.3.2018. https://firebase.google.com/docs/firestore/

Firebase 2018f. Receive Messages in an Android App. Viitattu 9.3.2018. https://firebase.google.com/docs/cloud-messaging/android/receive

Firebase 2018g. Set Up a Firebase Cloud Messaging Client App on Android. Viitattu 9.3.2018. https://firebase.google.com/docs/cloud-messaging/android/client

Firebase 2018h. Structuring Security Rules. Viitattu 16.3.2018. https://firebase.google.com/docs/firestore/security/rules-structure

Firebase 2018i. Topic Messaging on Web/JavaScript. Viitattu 15.3.2018. https://firebase.google.com/docs/cloud-messaging/js/topic-messaging

Cetinkayaa H & Akcayb M 2015. People Counting at Campuses. Viitattu 18.3.2018. https://www.sciencedirect.com/science/article/pii/S1877042815030967

Information Commissioner's Office 2016. Wi-Fi location analytics. Viitattu 9.3.2018. https://ico.org.uk/media/1560691/wi-fi-location-analytics-guidance.pdf

Kajala, L.; Almik, A.; Dahl, R.; Diksaito, L.; Erkkonen, J.; Fredman, P.; Jensen, F.; Sondergaard, K,; Sievaner, T. 2007. Visitor Monitoring in Nature Area - a manual based on experiences from the Nordic and Baltic Countries. https://books.google.com/books?id=o2hbbfd2S8cC&printsec=frontcover#v=onepage&q&f=false

Nader Fathi 2015. How Shopper Analytics Ensure the Mall Thrives. Viitattu 18.3.2018. https://www.chainstoreage.com/article/how-shopper-analytics-ensure-mall-thrives/

Node.js 2018. About. Viitattu 15.3.2018. https://nodejs.org/en/about/

OpenCV 2017. Background Subtraction. Viitattu 22.2.2018 https://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html

W3Schools.com 2018a. Node.js Modules. Viitattu 15.3.2018. https://www.w3schools.com/nodejs/nodejs_modules.asp

W3Schools.com 2018b. Node.js NPM. Viitattu 15.3.2018. https://www.w3schools.com/nodejs/nodejs_npm.asp

ZeroMQ 2014. ZeroMQ. Viitattu 15.3.2018. http://zeromq.org/