TAMPEREEN
AMMATTIKORKEAKOULU

TRELAB SCANNER

Marko Lovén

Opinnäytetyö
Marraskuu 2015
Tietotekniikka
Ohjelmistotekniikka

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikan koulutusohjelma
Ohjelmistotekniikka

MARKO LOVÉN:
TreLab Scanner

Opinnäytetyö 44 sivua, joista liitteitä 0 sivua
Marraskuu 2015

Työn aiheena on Android-käyttöjärjestelmälle toteutettu ohjelma nimeltä TreLab Scanner. Työ toteutettiin TreLab Oy:lle, joka on Tampereella toimiva teknologiayritys. Yrityksen kehittämä TreLab Smart Data Mill –mittaratkaisu rakentuu langattomista, älykkäistä mittalaitteista, SmartTageista, joilta oli tarve lukea ja käsitellä tietoa helposti ja nopeasti ihmiselle ymmärrettävään muotoon. Tähän tarkoitukseen ratkaisuksi kehitettiin puhelimella käytettävä sovellus, jonka kehityksestä tämä opinnäytetyö kertoo.

Ensimmäisessä luvussa tutustutaan tarkemmin ohjelman vaatimusten taustoihin ja ohjelman toimeksiannon antaneeseen yritykseen. Toisessa ja kolmannessa luvussa tutustutaan ohjelman käyttämiin taustatekniikoihin, eli Bluetoothiin ja Android-käyttöjärjestelmään. Neljäs luku kertoo syvällisemmin itse ohjelman suunnitteluratkaisuista ja haluttujen ominaisuuksien toteutuksesta. Viimeisessä luvussa pohditaan, missä onnistuttiin ja epäonnistuttiin, ja tutkitaan mahdollisia jatkokehityssuuntia.

Valmis ohjelma täyttää sille esitetyt vaatimukset ja se on otettu yrityksessä käyttöön. Vastaanotto ohjelmalle on ollut positiivinen, ja sen käyttäjät ovat esittäneet myös jatkokehitysehdotuksia ohjelman toiminnallisuuteen.

Asiasanat: mobiilisovellukset, android, bluetooth, trelab oy

ABSTRACT

Tampereen ammattikorkeakoulu
Tampere University of Applied Sciences
Degree programme in ICT Engineering
Software Engineering

MARKO LOVÉN:
TreLab Scanner

Bachelor's thesis 44 pages, appendices 0 pages
November 2015

The subject of this thesis is a mobile application developed for the Android operating system, called TreLab Scanner. The application was developed for TreLab Ltd, a company based in Tampere. TreLab Ltd is a high-tech company which has developed a wireless and intelligent measurement solution, TreLab Smart Data Mill, which consists of smart sensors called SmartTags. The company had a need for a fast and easy way to read and handle data collected by these sensors. For that need they felt like a mobile application would be a solution and this thesis describes the development of that application.

The first chapter contains a more accurate description of TreLab Ltd and gives more insight on the backgrounds of the application. The second and third chapters describe the technologies used in the development of the application, Bluetooth and Android. The fourth chapter describes the planning and development of the actual application. The last chapter contains a summary of the successes and failures of the development and discusses the further development of the application.

The application that was produced during this thesis meets the requirements placed for it and it has seen use in the company. The reception for the application has been positive and I have received some suggestions on how to develop it further.

Key words: mobile applications, android, bluetooth, trelab ltd

SISÄLLYS

LYHENTEET JA TERMIT

APK	Android application package, tiedostomuoto, johon Android-ohjelmat paketoidaan.
CSV	Tiedostomuoto, jonka lyhenne tulee sanoista comma separated values. CSV-tiedostoissa tieto on tallennettu yksinkertaiseen taulukkorakenteeseen, jonka kentät on eroteltu pilkuilla ja rivinvaihdoilla.
GATT	General attribute profile, protokolla, johon Bluetooth Smartin tiedonsiirto perustuu.
GNU	GNU's not unix, ohjelmistoprojekti, jonka tavoitteena on rakentaa täysin vapaasta lähdekoodista koostuva käyttöjärjestelmä.
IDE	Integrated development environment on ohjelmakokonaisuus, jolla voidaan toteuttaa ja suunnitella ohjelmistoja.
JSON	JavaScript object notation, erityisesti tiedonsiirrossa käytetty yksinkertainen tiedostomuoto.
QR-koodi	QR-koodi, jonka lyhenne tulee sanoista quick response, on kaksiulotteinen viivakoodi.
SDK	Software development kit on ohjelmakokonaisuus, joka sisältää kaiken tarvittavan ohjelmistojen luomiseen ja kääntämiseen tietylle alustalle tai ohjelmistokehykselle.
SQLite	Androidin käyttämä relaatiotietokanta.
XML	Extensible markup language, merkintäkieli, jota käytetään sekä tiedonsiirtoon että dokumenttien määrittämiseen ja tallentamiseen. Androidin käyttöliittymä määritellään XML-tiedostojen avulla.

1 JOHDANTO

Tämä opinnäytetyö on kirjoitettu TreLab Oy:lle kehittämästäni Android-sovelluksesta, jolla skannataan ympäristöstä yrityksen kehittämiä älykkäitä mittalaitteita ja luetaan niiltä tietoa.

TreLab Oy on vuonna 2011 perustettu tamperelainen yritys, joka kehittää langattomia mittausjärjestelmiä. Yrityksen kehittämä Smart Data Mill -järjestelmä koostuu kolmesta pääelementistä: pienikokoisesta, älykkäästä mittalaitteesta (Smart Tag), langattomasta tukiasemasta ja pilvipalvelusta (kuva 1).

KUVA 1. Smart Data Mill –järjestelmän rakenteen kuvaus. (TreLab Oy, Smart Data Mill)

Tämän opinnäytetyön kannalta järjestelmän tärkein osa on mittalaite (kuva 2). Mittalaite kiinnitetään suoraan mittapisteen pinnalle ja siihen on integroitu useita mittaustoiminnallisuuksia, joita voidaan muuttaa ja optimoida tapauskohtaisesti. Mittalaitteella mitattavia suureita ovat muun muassa:

- kiihtyvyys, pyöriminen, kallistus, liikkeen suunta, aktiivisuus
- liike/liikkumattomuus, putoaminen, törmäys
- ympäristötiedot: lämpötila, suhteellinen kosteus, ilmanpaine, valoisuus
- läsnäolo, suhteellinen paikkatieto. (Trelab Oy, Smart Data Mill)

KUVA 2. Trelab Oy:n Smart Tag -mittalaite. (Trelab Oy, Smart Data Mill)

TreLabilla oli jo olemassa mobiilisovellukset iOS, Android ja Windows Phone –alustoille. Ohjelmat kuitenkin vaativat yhteyden yrityksen pilvipalveluun, jotta niillä pystyi lukemaan mittalaitteen tuottamaa tietoa. Myös mittalaitteen pitää olla yhteydessä tukiasemaan, jotta sen tuottama tieto päätyy pilvipalveluun. Yrityksellä oli tarve mobiilisovellukselle, joka pystyisi lukemaan ja esittämään mittalaitteelta tietoa ilman, että mittalaite tai sovellus on yhteydessä muihin järjestelmän osiin. Pääasiallinen tarve sovellukselle oli aluksi sisäiseen testauskäyttöön, sekä tulevaisuudessa mahdollisiin asiakastarpeisiin. Sovellus päätettiin toteuttaa Android-alustalle ja sen työnimeksi tuli Trelab Scanner.

Ohjelman kehitys aloitettiin elokuussa 2014, jolloin ohjelman perusrakenne saatiin valmiiksi. Tämän jälkeen kehitys laitettiin hetkeksi tauolle ja sitä jatkettiin saman vuoden joulukuusta vuoden 2015 maaliskuuhun, jolloin ohjelmalta vaaditut ominaisuudet rajattiin tätä opinnäytetyötä varten ja rajaukseen kuuluneet ominaisuudet toteutettiin loppuun saakka.

Opinnäytetyössä käydään läpi ensin ohjelmaa tehdessä käytetyt teknologiat ja työkalut. Tämän jälkeen tutustutaan sovelluksen rakenteeseen sekä siltä vaadittujen toiminnallisuuksien toteutukseen. Lopuksi pohditaan työn onnistumista ja mahdollisia jatkosuunnitelmia ohjelman suhteen. Salassapitosopimuksen vuoksi kaikesta ohjelman sisällöstä ei voida kertoa tämän työn puitteissa.

2 BLUETOOTH SMART

Bluetooth on avoin standardi langattomaan tiedonsiirtoon lähietäisyyksillä. Sen kehittämisen aloitti Ericsson vuonna 1994 ja vuodesta 1998 sitä on hallinnut Bluetooth Special Interest Group, joka koostuu telekommunikaatio- ja tietotekniikka-alan yrityksistä. (Nordic Semi: A short history of bluetooth 2014)

Bluetooth toimii 2400 – 2483,5 MHz taajuusalueella, joka on jaettu 79 kanavaan. Tiedonsiirto perustuu pakettikytkentään, eli siirrettävä tieto jaotellaan paketeiksi tiedonsiirtoa varten. Bluetooth mahdollistaa kahdeksan eri laitteen liittämisen samaan verkkoon master-slave -tyyppisesti. Bluetoothin käyttöetäisyydelle ei ole asetettu maksimirajaa, ja se on pitkälti kiinni laitteen lähetystehosta. Bluetoothia käyttävien laitteiden lähetystehot on kuitenkin luokiteltu kolmeen luokkaan, joista ensimmäinen mahdollistaa yhteydet metrin ja kolmas jopa sadan metrin säteellä. (Bluetooth SIG: Bluetooth Basics)

Bluetooth Smart, jota kutsutaan myös Bluetooth Low Energyksi, on Bluetoothiin perustuva tekniikka, jonka päätavoitteena on tarjota huomattavasti pienempää virrankulutusta kuitenkin säilyttäen muut Bluetoothin tarjoamat ominaisuudet. (Bluetooth SIG: Bluetooth Smart)

Bluetooth Smartin toiminnallisuus rakentuu GATT-protokollan päälle. GATT-protokollassa toinen laite toimii palvelimena ja toinen asiakkaana. Kaikki tiedonsiirrot aloittaa asiakas, jonka pyyntöihin palvelin vastaa. Tiedonsiirto perustuu profiileihin, jotka sisältävät yhden tai useamman palvelun. Palveluiden sisällä taas on ominaisuuksia, joista voidaan lukea tai joihin voidaan kirjoittaa tietoa, sekä viittauksia muihin palveluihin. Ominaisuuksiin voidaan myös rekisteröidä niin sanottuja ilmoituksia, jolloin Bluetooth-laite lähettää ilmoitukset rekisteröinneelle laitteelle ilmoituksen, kun kyseinen ominaisuus muuttuu. Näin ominaisuutta ei tarvitse käydä erikseen lukemassa, vaan laite saa tiedon siitä aina, kun se muuttuu. (Adafruit: Introduction to Bluetooth Low Energy 2014)

Tämän opinnäytetyön sekä Trelabin mittajärjestelmän tapauksessa Bluetooth Smartia käytetään mittalaitteen ja siihen yhteydessä olevien laitteiden väliseen tiedonsiirtoon.

3 ANDROID

3.1 Android yleisesti

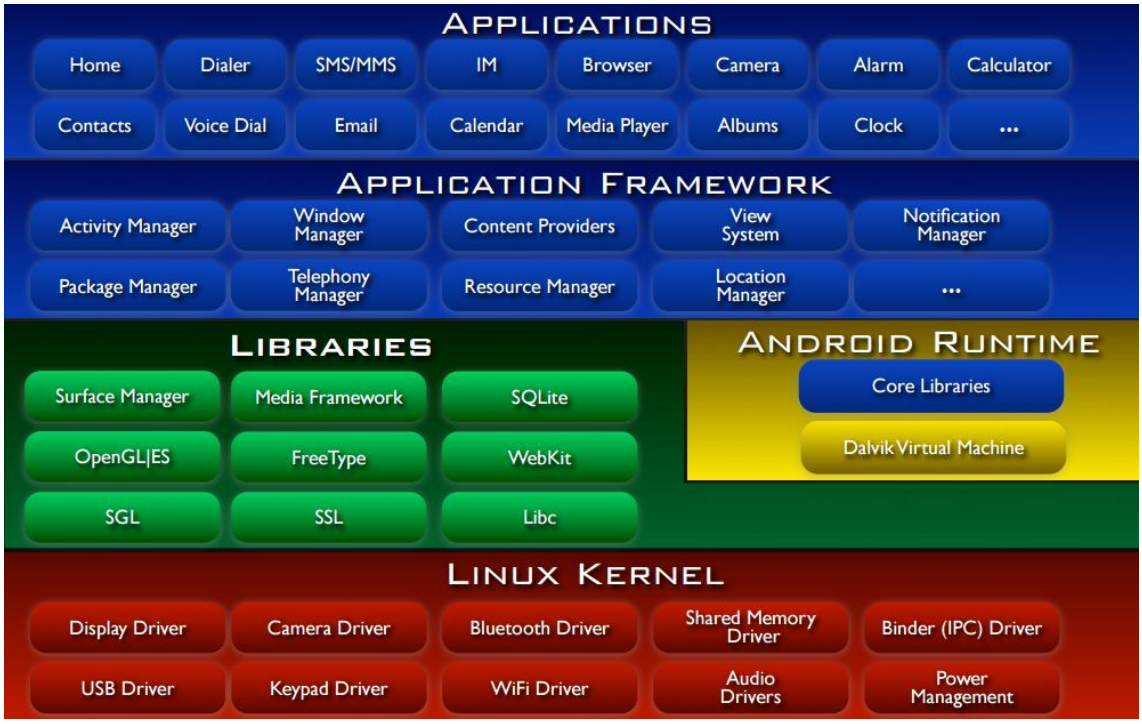
Android on Linux-ytimen päälle rakennettu avoimen lähdekoodin mobiilikäyttöjärjestelmä. Androidia on kehitetty vuodesta 2003 asti, alunperin Android Inc. -nimisen yrityksen toimesta. Vuonna 2005 Google osti alkuperäisen kehittäjän ja on toiminut Androidin pääasiallisena kehittäjänä siitä lähtien. Androidia kehittää myös sen ympärille perustettu Open Handset Alliance –konsortio, joka koostuu teleoperaattoreista ja elektroniikka-alan yrityksistä. (Steve Brachmann, A Brief History of Google's Android Operating System 2014)

Ensimmäinen Androidia käyttävä älypuhelin julkaistiin vuonna 2008. Alunperin älypuhelinvalmistajat eivät nähneet Androidia kovinkaan varteenotettavana kilpailijana, ja ensimmäisen vuoden aikana käyttöjärjestelmää ei nähty vielä kovinkaan monessa laitteessa. Sittemmin Androidista on tullut maailman käytetyin mobiilikäyttöjärjestelmä, ja se on vain jatkanut markkinaosuutensa kasvattamista. Androidin markkinaosuus vuoden 2014 kolmannella neljänneksellä oli 83,1 % kaikista maailmassa myydyistä älypuhelimista (Gartner 2014).

Android on laajentunut älypuhelinten ja tablettien lisäksi muun muassa televisioihin, autoihin, pelikonsoleihin, digitaalikameroihin ja kannettaviin tietokoneisiin.

3.2 Androidin ohjelmistopino

Androidin ohjelmistopino koostuu neljästä pääasiallisesta komponentista, jotka voidaan jakaa neljään eri kerrokseen (kuva 3).



KUVA 3. Androidin ohjelmistopino (Android Team 2009, Android Anatomy and Physiology).

Näistä kerroksista ensimmäinen käsittää Linux-ytimen, jonka päälle koko käyttöjärjestelmä on rakennettu. Tämä ei kuitenkaan tarkoita, että Android olisi Linux-jakelu, sillä Androidista puuttuu tiettyjä Linux-jakeluihin liitettyjä komponentteja, kuten GNU:n C-kirjasto. Linux-ytimeen perustuvat käyttöjärjestelmän muistin- ja prosessienhallinta, lupiin perustuva turvallisuusmalli, virranhallinta sekä useat laiteajurit. (Android Team 2009, Android Anatomy and Physiology)

Toiseen kerrokseen kuuluvat käyttöjärjestelmän käyttämät ohjelmistokirjastot sekä ajoympäristö. Ohjelmat Androidille kirjoitetaan Java-kielellä ja syötetään ajoympäristöille tavukoodiksi käännettyinä APK-paketteina, jonka ajoympäristö sitten kääntää laitteen ymmärtämäksi konekielisiksi käskyiksi. Androidin ajoympäristönä toimi ennen versiota 5.0 Dalvik Virtual Machine, ja sen jälkeen Android Runtime (ART). Molemmat ajoympäristöistä kehitettiin alunperin Androidia varten. Kumpikin

ajaa ohjelmat virtuaalikoneessa, jolloin ohjelmien ajo saadaan helposti eristettyä muista ohjelmista ja käyttöjärjestelmästä. Suurin ero näiden kahden ajoympäristön välillä on tapa, jolla tavukoodi käännetään. Dalvik käyttää Just-In-Time (JIT) –tapaa, jossa ohjelma käännetään käytännössä sillä hetkellä, kun se käynnistetään. Android Runtime taas käyttää Ahead-Of-Time (AOT) –tapaa, jolloin ohjelma käännetään, ennen kuin sitä käynnistetään. Käytännössä tämä tarkoittaa sitä, että ohjelmat käännetään, kun ne asennetaan laitteeseen, jolloin ohjelmat käynnistyvät nopeammin ja vaativat vähemmän prosessoritehoa käynnistyessään, mutta vievät enemmän sisäistä tallennustilaa. (Android Open Source Project: ART and Dalvik; Aatif Khan, What Is ART & How Is It Different From Dalvik Virtual Machine On Android? 2013)

Androidin käyttöjärjestelmätason ohjelmistokirjastot voidaan jakaa kolmeen kategoriaan. Java-yhteentoimivuuteen liittyvät kirjastot, eli käytännössä Dalvikille tai Android Runtimelle sovitettuja Javan perustoiminnallisuuksiin liittyviä kirjastoja, kuten merkkijonojen, verkkoyhteyksien ja tiedostojen käsittely. Android-kirjastot, eli erityisesti Androidille kehitetyt Java-kirjastot, jotka toimivat rajapintana Androidille rakennettaville sovelluksille. C/C++ -kirjastot, jotka tekevät itse asiassa suurimman osan Android-kirjastojen mahdollistamista toiminnoista. Koska Android rakentuu Linux-ytimen päälle, Android-kirjastot toimivat usein vain Javan kautta toteutettuina rajapintoina, tai ns. wrappereina, joita käytetään vain kutsumaan C/C++ -kirjastojen toimintoja. (Techotopia: An Overview of the Android Architecture 2014)

Ohjelmistopinon kolmannelta kerrokselta löytyy Androidin ohjelmistokehys, joka on kokoelma palveluita, jotka muodostavat ympäristön, jossa Androidille kehitettyjä ohjelmia voidaan ajaa. Taulukossa 1 on listattu osa Androidin ohjelmistokehyksen sisältämistä palveluista ja niiden käyttötarkoituksista.

TAULUKKO 1. Android-ohjelmistokehyksen palveluita (Techotopia: An Overview of the Android Architecture 2014).

Palvelu	Palvelun kuvaus
Activity Manager	Hallitsee ohjelmien elinkaarta ja näkymiä.
Resource Manager	Mahdollistaa ohjelmakoodin ulkopuolisten resurssien hakemisen ja käytön.
Notifications Manager	Hallitsee käyttöjärjestelmän ja ohjelmien ilmoituksia.
Package Manager	Hallitsee ja jakaa tietoa laitteelle asennetuista ohjelmista ja paketeista.
Telephony Manager	Hallitsee ja jakaa tietoa puhelintoiminnoista.
Location Manager	Hallisee ja jakaa tietoa puhelimen paikannustoiminnoista.

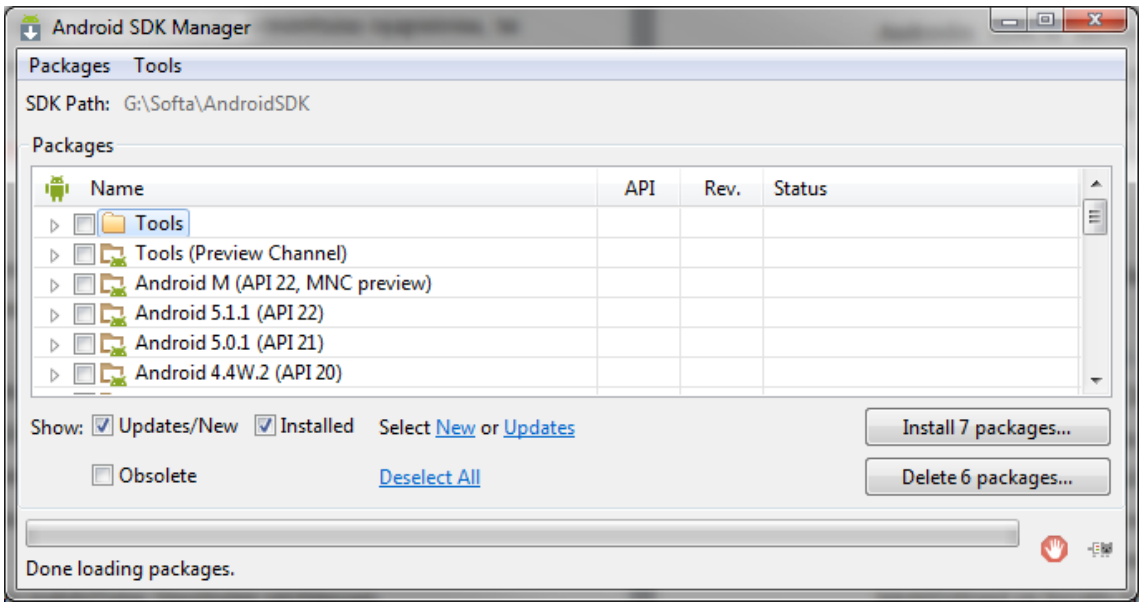
Neljännellä kerroksella ovat itse ohjelmat. Näihin lukeutuvat niin käyttäjän asentamat ohjelmat, käyttöjärjestelmän vakio-ohjelmat, kuin laitteen valmistajan siihen valmiiksi asentamat ohjelmat. (Techotopia: An Overview of the Android Architecture 2014)

3.3 Androidin kehitysympäristö

Androidin SDK:ta hallitaan Android SDK Manager –nimisellä työkalulla. SDK:n käyttämät komponentit on jaoteltu SDK Managerissa Androidin versioiden mukaan, jotta tarvittaessa ohjelma voidaan kääntää ja ajaa myös vanhemmilla Androidin versioilla. Android SDK Manager on esitetty kuvassa 4.

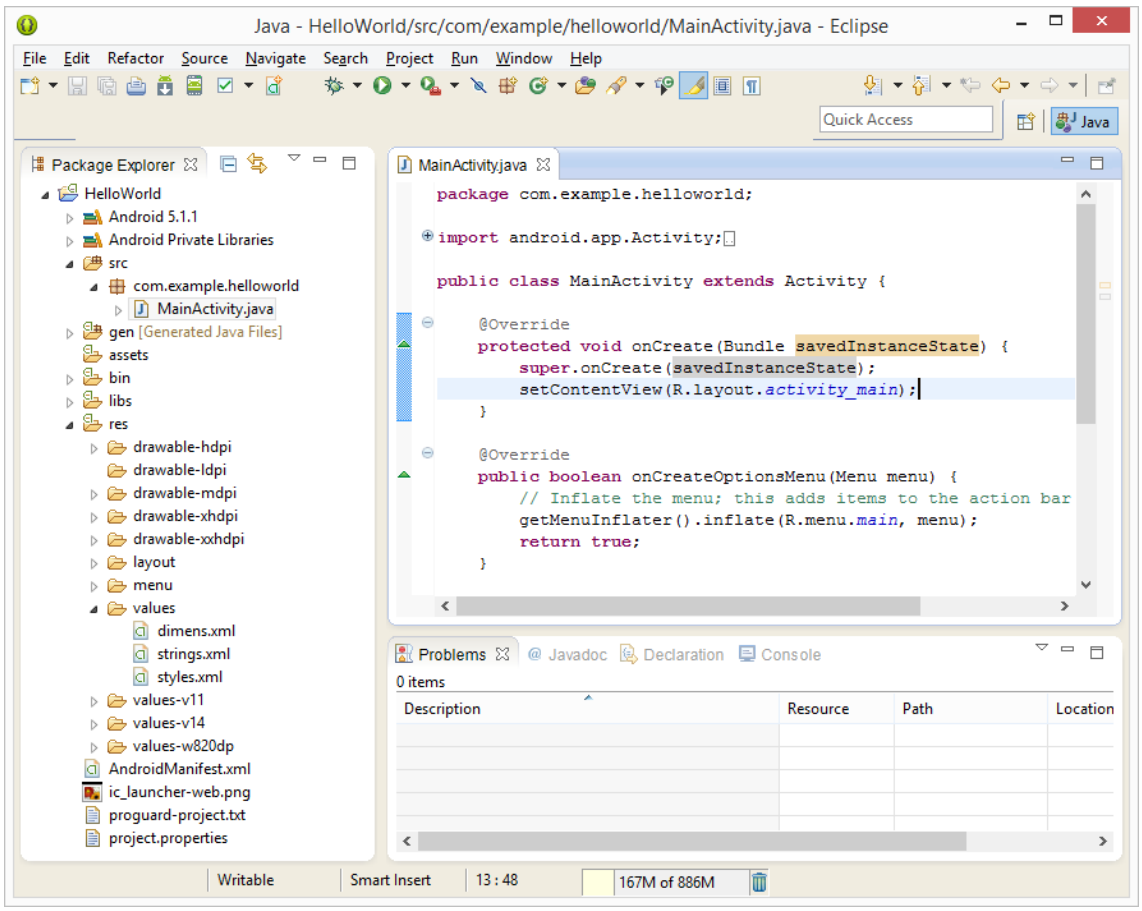
Androidin SDK sisältää seuraavat asiat:

- ohjelmointiympäristö
- debuggaustyökalut, joiden avulla voidaan etsiä ohjelmointivirheitä Android-ohjelman koodista ajamalla sitä
- kirjastot ja työkalut, joilla Android-ohjelmia voidaan kääntää
- emulaattori, jotta Android-ohjelmia voi ajaa ilman Android-laitetta
- dokumentaatiota SDK:n käyttämistä komponenteista sekä esimerkkiohjelmia.



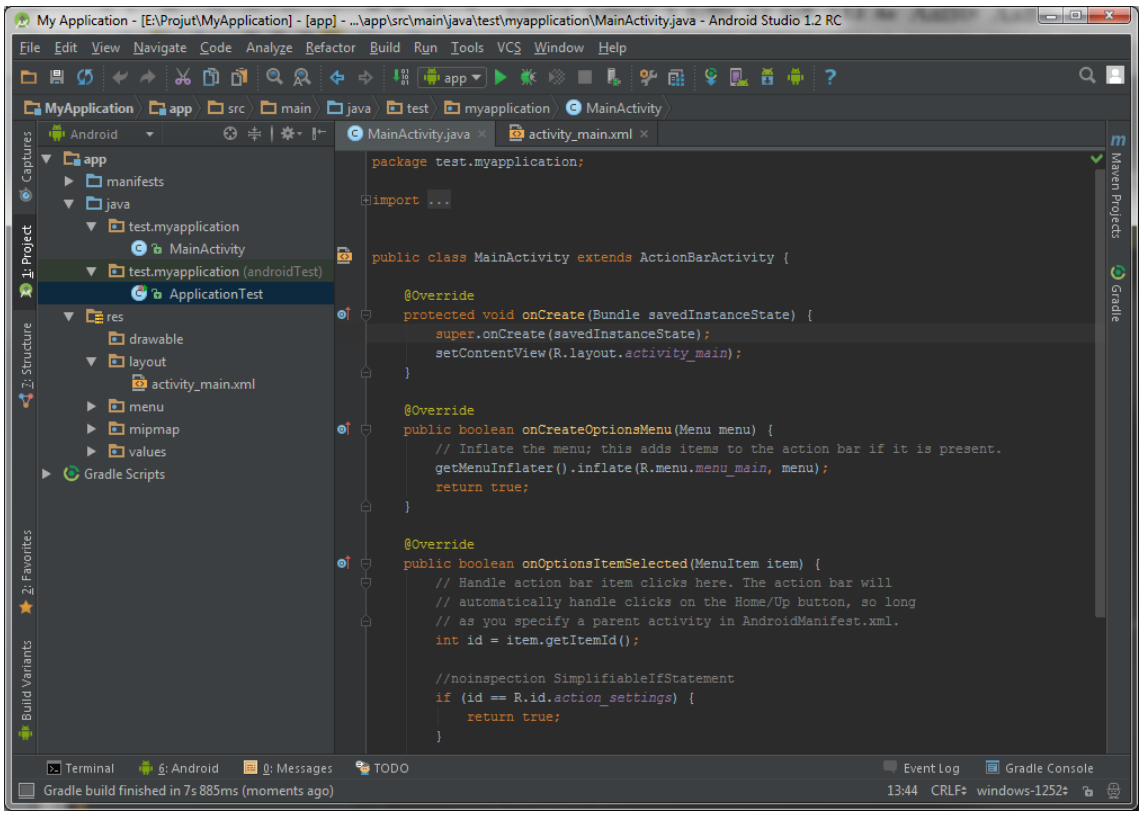
KUVA 4. Android SDK Manager

Vuoden 2014 loppuun asti Androidin virallisena ohjelmointiympäristönä toimi Eclipse ja sille kehitetty Android Development Tools –lisäosa (kuva 5). Eclipse on avoimen lähdekoodin ohjelmointiympäristö, joka perustuu helppoon laajennettavuuteen lisäosien kautta, ja suurin osa sen toiminnallisuudesta onkin toteutettu lisäosilla. Eclipse oli pitkään suosituimpia ohjelmointiympäristöjä Javan ja siihen liittyvien kielien kehittämiseen, mutta sitä on kritisoitu etenkin suorituskyvystä ja käyttöliittymän vaikeudesta (Eclipse wiki, IDE criticism 2014).



KUVA 5. Eclipse

Vuoden 2015 alusta alkaen ohjelmointiympäristö vaihtui Googlen kehittämään Android Studio –ohjelmaan (kuva 6), joka pohjautuu IntelliJ IDEA –ohjelmointiympäristöön. Se on myös laajennettavissa lisäosien kautta, mutta on huomattavasti Eclipseä suorituskykyisempi ja käyttäjäystävällisempi. Suurin muutos ohjelmointiympäristön vaihdon lisäksi Android Studiossa on Android-ohjelmien koontityökalun vaihtaminen Apache Antista Gradleen. Tämä vaihdos mahdollistaa Maven-pohjaisen riippuvuuksien hallinnan, joka helpottaa ohjelman lisäosien hallintaa huomattavasti, koska niitä ei enää tarvitse hallita käsin, vaan hallinta on automatisoitu.



KUVA 6. Android Studio

Tätä opinnäytetyötä aloitettaessa Android Studion kehitys oli vielä beta-vaiheessa, joten kehitysympäristönä toimi Eclipse, eikä siitä siirtymistä Android Studioon nähty tarpeellisena työtä tehdessä.

3.4 Google Play

Google Play, aiemmin Android Market, on Googlen ylläpitämä virallinen jakelukanava Android-sovelluksille. Google Playssa jaellaan myös mm. musiikkia, elokuvia, e-kirjoja ja digitaalisia lehtiä. Joillain laitevalmistajilla, kuten Samsungilla, on omia sovelluskauppoja, joista voi ladata ohjelmia. Ohjelmia voi myös asentaa suoraan APK-paketeista.

Jotta Google Playssa voi julkaista ohjelmia, pitää rekisteröidä kehittäjä-tili ja maksaa 25 $ rekisteröintimaksu (Android Developers: Get Started with Publishing). Maksullisista ohjelmista kehittäjä saa 70 % myyntihinnasta ja Google 30 % (Google Play: Transaction fees).

Google Play on tämän työn kirjoitushetkellä suurin jakelukanava mobiiliohjelmille. Maaliskuussa 2015 se sisälsi n. 1,5 miljoonaa ohjelmaa (Appbrain: Number of Android Applications 2015). Esimerkiksi Applen App Storessa, joka toimii jakelukanavana iOS-ohjelmille, oli tammikuussa 2015 n. 1,4 miljoonaa ohjelmaa (Apple Inc: App Store rings in 2015 with new records 2015).

Opinnäytetyön ohjelmaa ei laitettu Google Playhin, koska se tehtiin pääasiallisesti yrityksen sisäiseen käytöön.

3.5 Android-ohjelmien rakenne

Jotta opinnäytetyön toteutuksesta voidaan kertoa tarkemmin, pitää ensin tutustua siihen, kuinka Android-ohjelmat rakentuvat käytännön tasolla, ja kuinka Androidin Bluetooth Smart -rajapintaa käytetään.

3.5.1 Tiedostorakenne

Android-ohjelmaprojektit sisältävät käytännössä kahdenlaisia tiedostoja. Ohjelman lähdekoodi, joka sisältää kaiken ohjelman toiminnallisuuden, kirjoitetaan Java-tiedostoihin. Java-tiedostojen lisäksi käytetään XML-tiedostoja, joilla voidaan määrittää ulkoasua, ja ne voivat sisältää ohjelman käyttämiä vakioarvoja. Android-ohjelman kansiorakenteen tärkeimmät osa-alueet on esitetty taulukossa 2.

TAULUKKO 2. Android-ohjelman kansiorakenne

Kansio	Selite
src (Eclipse) java (Android Studio)	Sisältää ohjelman lähdekooditiedostot.
gen	Sisältää kehitysympäristön automaattisesti luoman lähdekoodin, joka vaaditaan ohjelman ajamiseen. Kansion sisältämät tiedostot kirjoitetaan uudestaan joka kerta, kun ohjelma kootaan.
libs	Sisältää ohjelman käyttämät kolmannen osapuolen kirjastot. Kirjastot on koottu Java-arkistoiksi, eli .jar-tiedostopäätteisiksi tiedostoiksi.
res	Sisältää ohjelman käyttämät niin ,sanotut resurssi-tiedostot. Resurssitiedostot sisältävät käyttöliittymää tai vakio-arvoja määritteleviä XML-tiedostoja ja esimerkiksi kuva- tai ääni-tiedostoja joita ohjelma käyttää.

Näiden kansioiden lisäksi jokainen Android-ohjelma sisältää AndroidManifest-nimisen XML-tiedoston, joka sisältää ohjelman toiminnan kannalta olennaisia määreitä. Näitä ovat mm. ohjelman Java-paketin nimi, ohjelman versiokoodi ja numero, Androidin SDK:n minimi ja kohdeversio, ohjelman käyttämät käyttöjärjestelmätason luvat sekä ohjelman aktiviteetit. Kuvassa 7 on esitetty opinnäytetyön ohjelman AndroidManifest-tiedosto.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="fi.trelab.trelabscanner"
    android:versionCode="8"
    android:versionName="0.2.3">
    <uses-sdk
        android:minSdkVersion="18"
        android:targetSdkVersion="21" />
    <!-- For using Bluetooth -->
    <uses-permission android:name="android.permission.BLUETOOTH"/>
    <!-- For discovering Bluetooth-devices -->
    <uses-permission android:name="android.permission.BLUETOOTH_ADMIN"/>
    <!-- For supporting BLE devices only -->
    <uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
    <!-- For sending emails -->
    <uses-permission android:name="android.permission.INTERNET" />
    <!-- For checking network availability -->
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
    <!-- For writing files to disk -->
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".SplashActivity"
            android:label="@string/app_name"
            android:theme="@android:style/Theme.Light.NoTitleBar.Fullscreen"
            android:configChanges="orientation"
            android:screenOrientation="portrait" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
            android:name=".TagListActivity"
            android:label="@string/app_name"
             >
        </activity>
    </application>
</manifest>
```

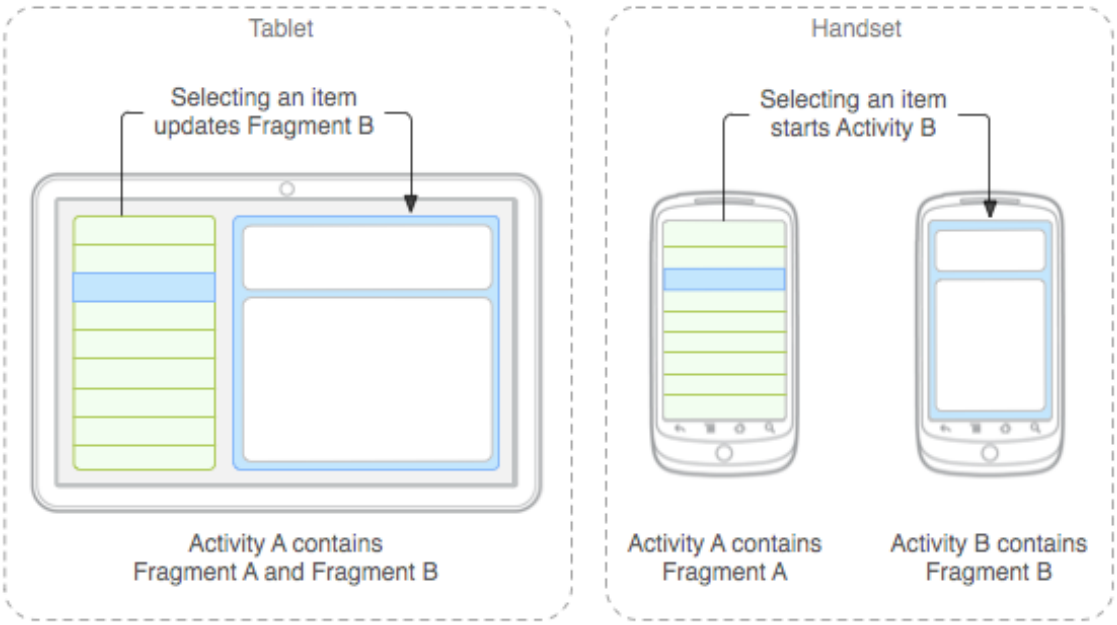
KUVA 7. TrelabScannerin AndroidManifest.xml

3.5.2 Ohjelman pääasialliset ohjelmakomponentit

Android-ohjelmalla on neljä pääasiallista ohjelmakomponenttityyppiä; Activity, Service, Content Provider ja Broadcast Receiver. Jokainen mainituista komponenteista on Java-luokka, joita käyttämällä ohjelman perustoiminnallisuus rakennetaan. Komponentteja kutsutaan tästä eteenpäin aktiviteeteiksi, palveluiksi, sisällöntarjoajiksi ja lähetysvastaanottajiksi. (Android Developer: Application Fundamentals)

Aktiviteetit esittävät yhtä näkymää ohjelman käyttöliittymällä. Jokaisella ohjelmalla on oltava ainakin yksi aktiviteetti. Vaikka monta aktiviteettia voikin muodostaa yhdessä ohjelman käyttöliittymäkokonaisuuden, pystyy jokainen niistä toimimaan itsenäisesti. Esimerkiksi puhelimen kamera-ohjelmasta voidaan kutsua sähköposti-ohjelman lähetys-aktiviteettia, eikä sinne erikseen tarvitse kulkea muiden näkymien kautta. (Android Developer: Application Fundamentals)

Aktiviteettien sisältö voidaan myös jakaa useampaan osaan käyttämällä Fragment-komponentteja, joita kutsutaan tästä eteenpäin palasiksi. Palanen esittää osaa näkymästä, ja niitä voidaan vaihdella aktiviteetin sisällä ja aktiviteettien välillä vapaasti. Palasista voidaan muodostaa helposti erilaisia näkymiä, jos esimerkiksi halutaan erilainen käyttöliittymä puhelimelle ja taulutietokoneelle (kuva 8).



KUVA 8. Taulutietokoneen ja puhelimen käyttöliittymä (Android Developer: Fragments).

Kuvassa vasemmalla puolella on taulutietokoneen käyttöliittymä, jossa käyttöliittymä on rakennettu yhdestä aktiviteetista ja kahdesta palasesta. Kun palasessa A valitaan listalta kohde, päivitettään palasen B sisältöä. Oikealla puolella taas käyttöliittymä on rakennettu kahdesta aktiviteetista, jotka molemmat sisältävät yhden palasen. Kun aktiviteetissä A valitaan palasen A listan sisältämä kohde, päivitetään aktiviteetissa B palasen B sisältöä. (Android Developer: Fragments)

Palvelut ovat ohjelman taustalla ajettavia ohjelmakomponentteja. Kun palvelu on käynnistetty, jatkaa se suoritusta, vaikka ohjelmasta poistuttaisiin. Palvelu voi esimerkiksi soittaa musiikkia silloin, kun käyttäjä käyttää jotain muuta ohjelmaa, tai hakea verkosta käyttöliittymän vaatimaa dataa ilman, että käyttöliittymän toiminta estyy. (Android Developer: Application Fundamentals)

Sisällöntarjoajat hakevat, muokkaavat ja tallettavat tietoa sovellusta varten. Tietoa käsitellään yleisimmin SQLite-tietokannasta, internetistä tai tiedostoista puhelimen muistista. Esimerkiksi puhelimen käyttäjän yhteystiedot voidaan hakea käyttöjärjestelmän ContactsContract-sisällöntarjoajalla. (Android Developer: Application Fundamentals)

Lähetysvastaanottajat kuuntelevat ja vastaavat järjestelmästä saapuviin ilmoituksiin. Useimmat näistä tulevat käyttöjärjestelmältä, esimerkiksi, jos puhelimen näyttö sammutetaan tai puhelimen akku on loppumaisillaan. Käyttöjärjestelmän ilmoitusten lisäksi muut sovellukset voivat lähettää ilmoituksia. (Android Developer: Application Fundamentals)

3.5.3 Android ja Bluetooth Smart

Android on sisältänyt tuen klassiselle Bluetoothille ensimmäisestä versiosta lähtien. Tuki Bluetooth Smartille lisättiin versiossa 4.3, ja se rakentuu pitkälti klassisen Bluetoothin rajapintojen päälle.

Androidin Bluetooth-rajapinnassa tämän työn kannalta pääasialliset komponentit ovat BluetoothAdapter- ja BluetoothDevice –luokat. BluetoothAdapter kuvaa puhelimen Bluetooth-lähetintä ja toimii yhteyspisteenä kaikelle Bluetooth-kommunikaatiolle. Androidin Bluetooth Smart –tuki laajentaa myös BluetoothAdapter-luokkaa Bluetooth Smartiin liittyvillä funktiolla ja rajapinnoilla. BluetoothDevice taas kuvaa Bluetooth-laitteita, joihin ollaan ottamassa yhteyttä.

Näiden päälle Bluetooth Smart –rajapinta lisää BluetoothGatt, BluetoothGattService, BluetoothGattCharacteristic-, BluetoothGattDescriptor- ja BluetoothManager –luokat, sekä BluetoothGattCallback-rajapinnan. Näistä kolme ensimmäistä kuvaavat toisessa

kappaleessa esitettyjä GATT-protokollan komponentteja, eli GATT-profiilia, palvelua ja ominaisuutta. BluetoothGattDescriptor kuvaa Bluetooth Smart -laitteen ominaisuuksiin kirjoitettavaa informaatiota ja asetuksia. BluetoothManager on luokka, jolla rakennetaan Bluetoothin käyttöön vaaditun BluetoothAdapter-luokan olioita. BluetoothGattCallback kuvaa GATT-ominaisuuksiin rekisteröitäviä ilmoituksia, eli kun Bluetooth Smart –laite lähettää ilmoituksen puhelimelle, ottaa BluetoothGattCallback-rajapinnan toteutus sen vastaan.

Jotta Androidin Bluetooth-ominaisuuksia voidaan käyttää, pitää ensin luoda BluetoothAdapter-luokan olio BluetoothManagerilla. Tämän jälkeen pitää tarkistaa, onko laitteen Bluetooth-lähetin päällä ja jos ei ole, pyytää käyttäjää laittamaan se päälle.

Tämän jälkeen halutaan todennäköisesti etsiä Bluetooth Smart -laitteita ympäristöstä. BluetoothAdapterissa on tätä varten funktiot startLeScan ja stopLeScan, jotka nimiensä mukaisesti aloittavat ja lopettavat laitteiden etsinnän. Joka kerta, kun BluetoothAdapter löytää Bluetooth Smart -laitteen, se kutsuu LeScanCallback-rajapinnan onLeScan-funktiota, josta Android-ohjelmalla pitää olla oma toteutus.

Yhteydenotto Bluetooth Smart –laitteisiin tapahtuu kutsumalla onLeScan-funktiolta saadun tai itse luodun BluetoothDevice-luokan olion connectGatt-funktiota, jolle pitää antaa parametrina BluetoothGattCallback-rajapinnan toteutus. Kyseisellä rajapinnalla on erinäisiä funktioita, joista tärkeimmät ovat onConnectionStateChange, onServicesDiscovered ja onCharacteristicChanged. Näiden lisäksi sillä on mm. GATT-ominaisuuksien lukemiselle ja kirjoittamiselle omat funktionsa. Näistä onConnectionStateChangen avulla hallitaan pitkälti yhteyttä Bluetooth Smart -laitteeseen. Kyseistä funktiota kutsutaan joka kerta, kun yhteys laitteiden välillä muuttuu, eli esimerkiksi silloin, kun yhteyttä muodostetaan, kun yhteys on muodostettu ja kun yhteys katkaistaan.

Kun yhteys Bluetooth Smart -laitteeseen on saatu muodostettua, halutaan usein tietää, mitä se sisältää. Tällöin kutsutaan onConnectionStateChangen mukana saatua BluetoothGatt-luokan olion discoverServices-funktiota. Tämä käynnistää palveluiden etsinnän Bluetooth Smart –laitteella, ja kun palvelut on saatu etsittyä, lähetetään siitä ilmoitus puhelimelle, joka puolestaan kutsuu BluetoothGattCallback-rajapinnan onServicesDiscovered-funktiota, jonka parametrina ovat löydetyt palvelut.

Kun palvelut on löydetty, pystytään BluetoothGatt-oliosta hakemaan kaikki löydetyt palvelut ja niiden sisältämät ominaisuudet. Ominaisuuksilta pystytään lukemaan niiden sisältämät arvot, ja niihin pystytään haluttaessa kirjoittamaan uusia arvoja.

Jos löydettyjen palveluiden ominaisuuksilta halutaan ilmoituksia, pitää Bluetooth Smart -laitteelle ensin kirjoittaa BluetoothGattDescriptorin avulla asetus, joka mahdollistaa ilmoitukset. Tämän jälkeen Bluetooth Smart -laite lähettää ilmoituksia Android-puhelimelle. Ilmoitukset otetaan vastaan BluetoothGattCallback-rajapinnan onCharacteristicChanged-funktion toteutuksen avulla.

Lopuksi Bluetooth-yhteys pitää muistaa sulkea. Sulkeminen tapahtuu kutsumalla BluetoothGatt-olion close-funktiota, jotta käyttöjärjestelmä pystyy vapauttamaan sille rajatut resurssit.

4 TRELAB SCANNER

4.1 Suunnittelu

Ohjelmaa suunniteltaessa pääasiallisena tarkoituksena oli saada älypuhelimilla toimiva sovellus, joka korvaisi osittain kannettavan tietokoneen mittalaitteiden tarkastelu- ja konfigurointityökaluna. Älypuhelimen eduiksi kannettavaan tietokoneeseen verrattuna voidaan lukea huomattavasti kevyempi paino ja käyttöliittymän yksinkertaisuus, jolloin sitä olisi helpompi ja nopeampi käyttää esimerkiksi asiakaskäynneillä.

Opinnäytetyötä varten rajattiin toteutettavaksi seuraavat ominaisuudet:

- mittalaitteiden etsiminen ympäristöstä
- mittausdatan lukeminen ja parsiminen
- lokien lukeminen, parsiminen ja lähettäminen
- parametrien kirjoitus mittalaitteen sovelluksiin.

Osa halutuista ominaisuuksista päätettiin toteuttaa kevyesti, koska tarkoituksena oli lähinnä todentaa niiden toteutuksen mahdollisuus Androidin puitteissa.

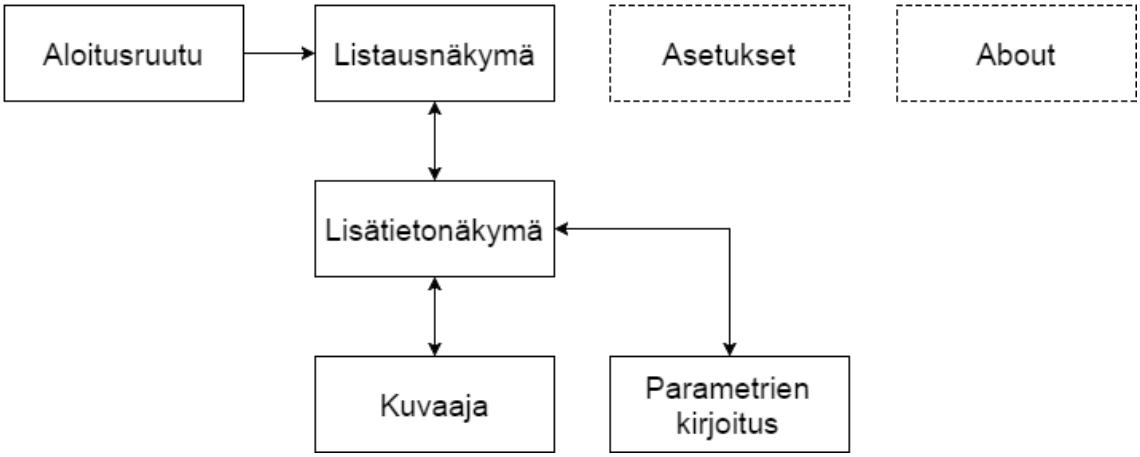
4.2 Ohjelman näkymät

Ohjelman käyttöliittymä rakentuu seitsemästä näkymästä. Ohjelman käynnistyessä näytetään käyttäjälle aloitusruutu, josta siirrytään pienen viiveen jälkeen ohjelman päänäkymään.

Päänäkymä on lista ohjelman löytämistä mittalaitteista. Valitsemalla löydetyn mittalaitteen listalta käyttäjä pääsee sen tarkempiin tietoihin. Tässä näkymässä on listattu mittalaitteelta parsittu mittaustieto sekä mahdollisuus ladata lokeja tai kirjoittaa sille parametreja. Mittalaitteelta ladatut lokitiedostot tulevat myös näkyviin tähän näkymään, ja niitä valitsemalla pääsee kuvaaja-näkymään, jossa valitun lokin sisältö on piirrettynä kuvaajalle.

Näiden näkymien lisäksi ohjelmassa on sekä Asetukset- että About-näkymä, joihin pääsee siirtymään ohjelman jokaisesta muusta näkymästä. Asetukset-näkymästä pystyy hallitsemaan ohjelman asetuksia, ja About-näkymästä löytyvät yrityksen yhteystiedot ja tekijänoikeudelliset tiedot.

Ohjelman näkymät ja niiden väliset suhteet on kuvattu kuvassa 9. Näkymät kuvataan tarkemmin ohjelman ominaisuuksia käsittelevissä kappaleissa.



KUVA 9. Ohjelman näkymät

4.3 Toteutus

Ohjelman käyttöliittymän toteutus jaettiin kahteen aktiviteettiin, aloitusruutuun ja pääaktiviteettiin. Pääaktiviteetti jaettiin eri palasiin, joita vaihdetaan, kun käyttäjä haluaa liikkua näkymästä toiseen. Jälkikäteen ajatellen ohjelman Asetukset- ja About-näkymät olisi kannattanut jakaa vielä omiin näkymiinsä käyttöliittymän toteutuksen selkeyttämiseksi.

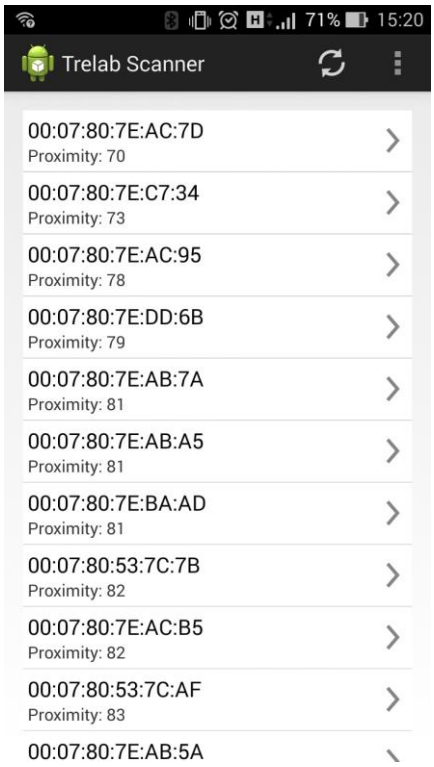
Ohjelmalta haluttujen ominaisuuksien toteutuksen kannalta ohjelman tärkeimmät osat ovat BtManager- ja TagManager –luokat.

BtManager hallitsee ohjelman Bluetooth-toimintoja ja pitää huolta Bluetooth-yhteyden tilasta. Koska Android-älypuhelin on Bluetoothin ja ohjelman kontekstissa asiakas, voi se olla yhteydessä vain yhteen mittalaitteeseen kerrallaan.

TagManager ylläpitää listaa löydetyistä mittalaitteista, jotta niiden tiedot saadaan helposti luettua. Mittalaitteiden tiedot tallennetaan Tag-nimisen luokan olioihin, jotka kuvaavat yksittäisiä mittalaitteita.

4.3.1 Mittalaitteiden etsintä

Jotta ohjelman muita ominaisuuksia pystytään toteuttamaan ja käyttämään, pitää sen osata etsiä mittalaitteita ympäristöstä. Etsintä käynnistyy automaattisesti, kun ohjelma käynnistetään. Käyttäjä pystyy myös itse aloittamaan etsinnän uudestaan painamalla oikeasta yläreunasta löytyvää etsintä-painiketta. Kun ohjelma on saanut etsinnän suoritettua, listataan löydetyt mittalaitteet listausnäkymälle (kuva 10).



KUVA 10. Listausnäkymä

Mittalaitteiden etsintä on toteutettu BtManager-luokan scanLeDevice-funktiossa. Funktio käyttää skannauksen aloittamiseen Androidin Bluetooth Smart -rajapinnan BluetoothAdapter-luokan startLeScan-funktiota. Ohjelma skannaa mittalaitteita ympäristöstä määritetyn ajanjakson ajan, jonka jälkeen se pysäyttää skannauksen kutsumalla BluetoothAdapterin stopLeScan-funktiota ja ilmoittaa käyttöliittymälle, että skannaus on suoritettu (kuva 11).

```
public void scanLeDevice() {
    // Initialize list for already found devices on this round of scanning
    foundDevices = new ArrayList<String>();

    if (!mScanning) {
        for ( BtScanCallback callback : mCallbacks ) callback.BtScanStarted();

        // Stops scanning after a pre-defined scan period.
        mHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                    mScanning = false;
                    mBluetoothAdapter.stopLeScan(mLeScanCallback);

                    for ( BtScanCallback callback : mCallbacks ) callback.BtScanFinished();
            }
        }, mScanPeriod);

        mScanning = true;
        mTagManager.clearTagList();
        mBluetoothAdapter.startLeScan(mLeScanCallback);
    } else {
        Log.e(CLASS, "Already scanning, not starting another scan");
    }
}
```

KUVA 11. Bluetooth Smart -skannauksen aloitus

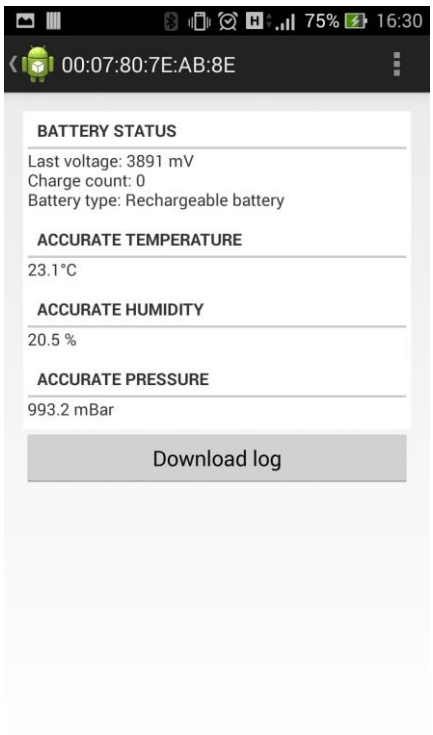
Kun Bluetooth Smart -laite löydetään, ohjelma saa siitä viestin ja kutsuu ohjelman toteutusta onLeScan-funktiosta (kuva 12). Tällöin tarkistetaan, onko löydetty laite TreLabin lukemalla sen palauttamasta skannausdatasta valmistajakohtainen tunniste. Jos laite on Trelabin mittalaite ja sitä ei ole ennen löydetty, se lisätään TagManagerin ylläpitämään mittalaitelistaan. Skannauksessa saadun datan perusteella arvioidaan myös etäisyys mittalaitteesta, jota käytetään mittalaitteiden listaamiseen sen mukaan, kuinka lähellä mittalaitetta puhelin on skannaushetkellä. Kun skannaus on saatu suoritettua loppuun, päivittää ohjelma näkymänsä ja listaa löydetyt mittalaitteet.

```
private BluetoothAdapter.LeScanCallback mLeScanCallback = new BluetoothAdapter.LeScanCallback() {
    @Override
    public void onLeScan(final BluetoothDevice device, int rssi,
        byte[] scanRecord) {
        try {
            if ( scanRecord[8] == TRELAB_ID_FIRST_BYTE && scanRecord[9] == TRELAB_ID_SECOND_BYTE ) {
                if ( foundDevices.contains(device.getAddress()) ) {
                    Log.i(CLASS, "Device already found once on this scanning round, not adding again");
                } else if ( !device.getAddress().equals("") ) {
                    int txPowerLevel = scanRecord[5];
                    int proximity = txPowerLevel - rssi;

                    Tag newTag = new Tag(mContext, device, proximity);
                    foundDevices.add(device.getAddress());
                    mTagManager.addTag(newTag);
                }
            } else {
                Log.i(CLASS, "Not our tag, not adding");
            }
        } catch ( NullPointerException npe ) {
            // Can't get name out of BluetoothDevice
            Log.e(CLASS, "Can't get name of bluetooth device!", npe);
        }
    }
};
```

KUVA 12. onLeScan-rajapinnan toteutus Trelab Scannerissa

4.3.2 Mittausdatan lukeminen ja parsiminen

Kun mittalaitteet on löydetty ja saatu listattua, olisi tarve lukea myös niiden mittaamia arvoja ja parsia nämä arvot ihmisen luettavaan muotoon käyttöliittymälle. Mittausdata esitetään löydettyjen laitteiden lisätietonäkymässä, johon päästään painamalla listalla olevaa laitetta (kuva 13).



KUVA 13. Mittalaitteen lisätietonäkymä

Tätä ominaisuutta varten päätettiin tehdä parseriluokka, joka kehitettiin erillisenä itse työn ohjelmasta, ja lisättiin siihen valmiiksi koottuna kirjastona. Parseri toteutettiin myös Java-kielellä. Se päätettiin erottaa tämän työn ohjelmasta sen takia, että Java-pohjaiselle parserille voisi olla käyttöä tulevaisuudessa muissakin yrityksen projekteissa. Parseriluokan toimintaa ei tässä työssä käydä läpi.

Kun skannaustoiminto on saatu suoritettua loppuun, otetaan löydettyihin mittalaitteisiin yhteys, ja niiltä luetaan niiden sisältämä mittausdata, joka välitetään eteenpäin parserille. Parseri jaottelee siitä eri suureet, muuttaa arvot ihmisen luettavaan muotoon ja palauttaa ne listana. Tämä lista tallennetaan Tag-luokan olioon, jolta käyttöliittymä voi helposti hakea sen sisältämän mittausdatan.

4.3.3 Lokien lataaminen ja parsiminen

Ympäristösuureiden mittaamisen lisäksi mittalaitteet pystyvät kirjoittamaan mittaamaansa dataa lokiin. Yksi ohjelman halutuista ominaisuuksista oli ladata laitteilta niiden kirjoittamat lokit ja parsia niiden sisältämä mittausdata.

Toimintoa varten ohjelmaan lisättiin mittalaitteen Lisätieto-näkymään painike lokien latausta varten (kuva 13). Kun käyttäjä klikkaa painiketta, ottaa ohjelma yhteyden mittalaitteeseen.

Kun yhteyden tila Bluetooth Smart -laitteeseen muuttuu, kutsuu Androidin Bluetooth-rajapinta ohjelman toteutusta BluetoothGattCallback-luokan onConnectionStateChanged-funktiosta (kuva 14). Jos yhteydenotto on onnistunut, alkaa ohjelma etsimään laitteelta löytyviä palveluita. Jos yhteydenotto ei onnistunut, palautetaan BtManager yhteydettömään tilaan.

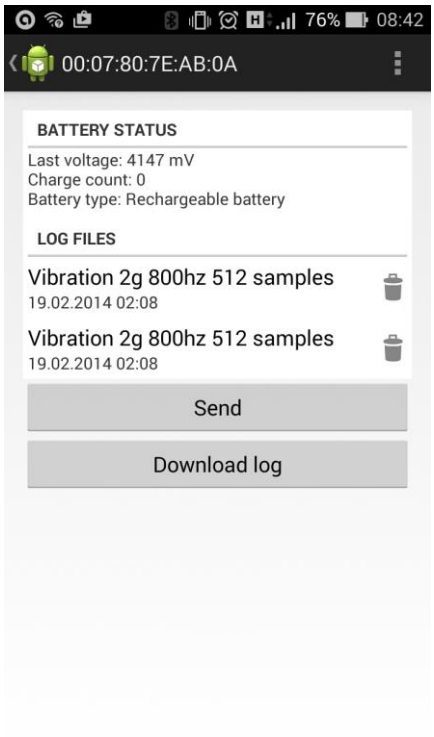
```
@Override
public void onConnectionStateChange(BluetoothGatt gatt, int status, int newState) {
    if (status == BluetoothGatt.GATT_SUCCESS
            && newState == BluetoothProfile.STATE_CONNECTED) {
        mConnectionState = ConnectionState.CONNECTED;

        //Discover services
        gatt.discoverServices();

        Log.i(CLASS, "GATT Connection connected");
    } else if (status == BluetoothGatt.GATT_SUCCESS
            && newState == BluetoothProfile.STATE_DISCONNECTED) {
        mConnectionState = ConnectionState.DISCONNECTED;

        //Handle a disconnect event
        Log.i(CLASS, "GATT Connection disconnected, closing it");
        gatt.close();
        mBluetoothGatt = null;
    } else if (status == BluetoothGatt.GATT_FAILURE) {
        mConnectionState = ConnectionState.DISCONNECTED;
        Log.e(CLASS, "GATT Connection failed");
    } else {
        mConnectionState = ConnectionState.DISCONNECTED;
        Log.e(CLASS, "GATT returned " + status + ", closing connection");

        gatt.close();
        mBluetoothGatt = null;
    }

    // Inform UI of connection state change
    mBtGattCallback.OnConnectionStatusChange(mConnectionState);
}
```

KUVA 14. Ohjelman onConnectionStateChanged-funktion toteutus

Palveluiden löytymisen jälkeen lokien lataukseen liittyviin GATT-ominaisuuksiin kirjoitetaan, että niiltä halutaan saada ilmoituksia. Tämä tehdään siksi, ettei dataa pystytä lukemaan yhdessä paketissa lokien suuren koon vuoksi, vaan se pitää tehdä osissa. Kun mittalaite saa yhden osan lokia valmisteltua lähetystä varten, lähetetään se

puhelimelle Bluetoothin ilmoitusten avulla. Kun mittalaitteelle on asetettu ilmoitukset, lähetetään sille komento, että lokien lataaminen voidaan aloittaa.

Kun kaikki lokin osat on saatu ladattua, ne koostetaan yhteen ja annetaan eteenpäin lokiparserille. Parseri erottelee taulukosta kokonaiset lokit ja parsii ne helpommin käsiteltävään muotoon. Nämä tallennetaan CSV-tiedostoina puhelimen ulkoiseen muistiin, josta ne on helppo ladata tietokoneelle tai lähettää eteenpäin sähköpostitse, sekä JSON-muodossa sisäiseen muistiin, josta ne on helpompi lukea datan visualisointia varten. Tämän jälkeen tallennetut lokit listataan esille mittalaitteen lisätietonäkymään (kuva 15). Ladattujen lokitiedostojen oikeassa reunassa olevasta roskakori-ikonista pystytään myös poistamaan ladattuja lokeja.

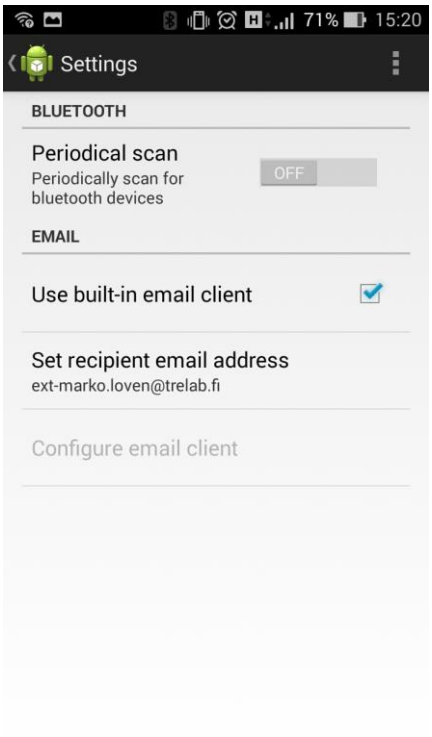


KUVA 15. Ladatut lokitiedostot

4.3.4 Lokien lähetys

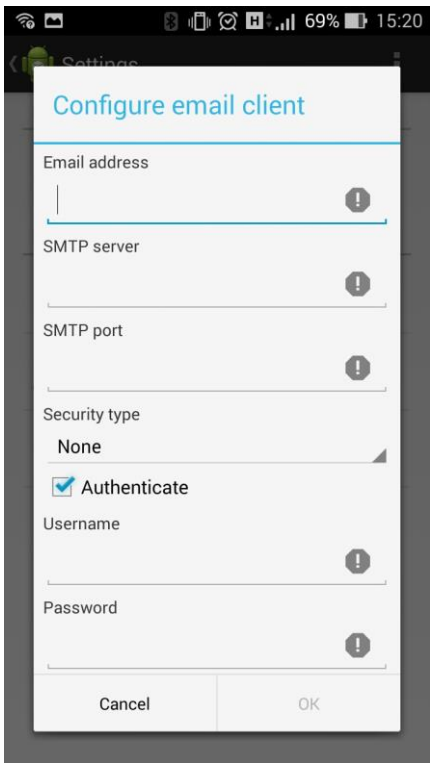
Lokien lukemisen ja parsimisen lisäksi ohjelmaan haluttiin valmius lähettää lokeja sähköpostitse eteenpäin. Sähköpostin lähettäjän ja vastaanottajan tulisi olla käyttäjän määriteltävissä. Ominaisuuden toteutuksessä käytettiin apuna JavaMail-kirjastoa, joka on Oraclen kehittämä Java-pohjainen avoimen lähdekoodin sähköpostirajapinta.

Ominaisuutta varten Asetukset-näkymään lisättiin osio sähköpostiasetuksille. Tässä osiossa käyttäjä voi määritellä, haluaako hän käyttää jo puhelimelle asennettuja sähköposti-ohjelmia, vai haluaako hän määrittää itse sähköpostiasetukset (kuva 16).



KUVA 16. Ohjelman asetukset-näkymä

Jos käyttäjä ei halua käyttää puhelimelle asennettuja sähköposti-ohjelmia, hän voi myös käyttää JavaMail-kirjastolla toteutettua sähköpostia, jolle pitää asettaa sähköpostin lähetystä varten asetukset painamalla asetusten määrityspainiketta. Tällöin aukeaa asetusten määritysnäkymä (kuva 17).



KUVA 17. Sähköpostin asetusten määritysnäkymä

Sähköpostiin liittyvät asetukset tallennetaan puhelimen sisäiseen, ohjelmalle varattuun muistiin. Tämä ratkaisu ei käyttäjätunnuksen ja salasanan osalta ole täysin tietoturvallinen, mutta sen monimutkaistamiseksi ei nähty tarvetta tässä vaiheessa. Jos puhelin on niin sanotusti rootattu, eli puhelimen käyttäjällä on pääkäyttäjän oikeudet puhelimeen, voi sovelluksille varattuun muistiin päästä käsiksi. Turvallisempaa olisi joko vaatia jokaiseen lähetykseen käyttäjätunnusta ja salasanaa, joka heikentäisi käyttäjäkokemusta huomattavasti, tai jos sähköpostipalvelin tukee sitä, vaatia autentikoitumista palvelimelle kerran ja tallentaa kyseisestä autentikoinista saatu sessiotunniste puhelimen muistiin. Tätä tunnistetta voitaisiin käyttää autentikointumiseen, kun halutaan lähettää sähköpostia.

Asetusten lisäksi mittalaitteen lisätietonäkymään lisättiin painike sähköpostin lähettämistä varten (kuva 15). Painikkeelle on lisätty kuuntelija, joka painiketta painettaessa laukaisee sille tehdyn toteutuksen Androidin onClick-funktiosta (kuva 18).

```
mSendEmailButton.setOnClickListener(new OnClickListener() {
    @Override
    public void onClick(View v) {
        String to;
        String subject;
        String body;

        // Get address from sharedpreferences
        SharedPreferences prefs = PreferenceManager.getDefaultSharedPreferences(getActivity());
        to = prefs.getString(Constants.PREFERENCE_KEY_TO_ADDRESS, "");

        // Add subject and body
        subject = getString(R.string.details_view_send_header) + mTag.getAddress();
        body = "";

        // Construct a mail object and send it
        Mail mailToSend = new Mail(getActivity(), to, subject, body, mTag.getAddress());
        mailToSend.sendMail();
    }
});
```

KUVA 18. Sähköpostin lähetyspainikkeen toteutus

Ohjelma hakee muistista tarvitsemansa asetukset ja luo niiden avulla Mail-luokan olion. Mail-luokka käsittää sähköpostin lähettämisen vaatimat toiminnot, ja sen oliot esittävät yksittäisiä sähköpostiviestejä. Luokalla on vain yksi ulospäin näkyvä funktio, sendMail, joka nimensä mukaisesti lähettää sähköpostin (kuva 19).

```
public void sendMail() {
    // Check whether network is available, if not show toast and return
    if ( !isNetworkAvailable() ) {
        Toast.makeText(mContext, R.string.details_view_send_no_connectivity, Toast.LENGTH_SHORT).show();
        return;
    }

    // If no address is set, show toast and return
    if ( mTo.equals("") ) {
        Toast.makeText(mContext, R.string.details_view_send_no_address, Toast.LENGTH_SHORT).show();
        return;
    }

    // Get any possible configurations from shared preferences
    SharedPreferences sharedPreferences = PreferenceManager.getDefaultSharedPreferences(mContext);
    boolean useEmailClient = sharedPreferences.getBoolean(Constants.PREFERENCE_KEY_USE_INTENT, true);
    mFrom = sharedPreferences.getString(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_ADDRESS, "");
    mUsername = sharedPreferences.getString(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_USERNAME, "");
    mPassword = sharedPreferences.getString(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_PASSWORD, "");
    mHost = sharedPreferences.getString(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_HOST, "");
    mPort = sharedPreferences.getString(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_PORT, "");
    mSecurityType = sharedPreferences.getInt(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_SECURITY_TYPE, 0);
    mAuthenticate = sharedPreferences.getBoolean(Constants.MAIL_SETTINGS_KEY + Constants.MAIL_SUFFIX_AUTH, true);

    // Send email through an installed app or smtp client
    if ( useEmailClient ) sendThroughInstalledApp();
    else sendWithSmtpClient();
}
```

KUVA 19. sendMail-funktio

Ensimmäiseksi sendMail-funktio tarkistaa, onko puhelimella verkkoyhteyttä. Jos ei, se antaa käyttäjälle virheilmoituksen. Sen jälkeen tarkistetaan, onko oliolle asetettu sähköpostin vastaanottajan osoite. Tämäkin antaa virheilmoituksen, jos näin ei ole.

Tämän jälkeen funktio hakee puhelimen muistiin tallennetuista asetuksista sähköpostiin liittyvät asetukset, kuten kumpaa sähköpostin lähetystapaa käytetään, sähköpostipalvelimen osoitteen ja portin sekä palvelimelle autentikoitumiseen liittyvät asetukset. Lopuksi sähköposti lähetetään määritetyllä tavalla.

Jos käyttäjä haluaa käyttää jotain puhelimelle asennettua sähköpostiohjelmaa, kutsutaan sendThroughInstalledApp-funktiota (kuva 20). Funktio luo sähköpostiohjelman avaamista varten Androidin Intent-luokan olion. Tämän jälkeen kutsutaan ohjelman kontekstin, joka toimii ohjelman rajapintana Androidin käyttöjärjestelmätoimintoihin, startActivity-funktiota luodulla Intent-oliolla. Tällöin käyttäjälle avautuu valitsin, jossa näkyvät kaikki saatavilla olevat sähköpostiohjelmat, tai ilmoitus, ettei yhtään sähköpostiohjelmaa ole asennettu. Käyttäjän valitessa jonkun sähköpostiohjelman siirrytään valitun ohjelman puolelle.

```
private void sendThroughInstalledApp() {
    // Create intent for using installed email client
    Intent i = new Intent(Intent.ACTION_SEND_MULTIPLE);
    i.putExtra(Intent.EXTRA_EMAIL, new String[] {mTo}); // Set recipient
    i.putExtra(Intent.EXTRA_SUBJECT, mSubject); // Set subject and body

    // Add log files as attachment if any
    ArrayList<Uri> uris = new ArrayList<Uri>();
    File filePaths[] = FileUtility.getLogFilePath(mTagAddress).listFiles();
    for (File file : filePaths) {
        Uri u = Uri.fromFile(file);
        uris.add(u);
    }

    if (uris.size() > 0)
        i.putParcelableArrayListExtra(Intent.EXTRA_STREAM, uris);


    // To filter out non-email clients
    final Intent restrictIntent = new Intent(Intent.ACTION_SENDTO);
    Uri data = Uri.parse("mailto:?to=" + mTo);
    restrictIntent.setData(data);
    i.setSelector(restrictIntent);

    // Start email app selector or give user a toast if there are none email apps installed
    try {
        mContext.startActivity(Intent.createChooser(i,
                mContext.getResources().getString(R.string.details_view_send_selection_text).toString()));
    } catch (android.content.ActivityNotFoundException ex) {
        Toast.makeText(mContext, mContext.getResources().getString(R.string.details_view_send_no_clients),
                Toast.LENGTH_SHORT).show();
    }
}
```

KUVA 20. sendThroughInstalledApp-funktio

Jos käyttäjä haluaa lähettää lokit eteenpäin ohjelman omalla sähköpostitoiminnolla, kutsutaan sendWithSmtpClient-funktiota (kuva 21). Jos käyttäjä ei olekkaan asettanut sähköpostin vaatimia asetuksia, annetaan käyttäjälle virheilmoitus. Muuten funktio luo

ja konfiguroi JavaMail-kirjaston vaatimat Session- ja Message-luokan oliot ja yrittää lähettää viestin eteenpäin kutsumalla SendMailTask-olion execute-funktiota.

```
private void sendWithSmtpClient() {
    if ( mFrom.equals("") || mUsername.equals("") || mPassword.equals("") ||
            mHost.equals("") || mPort.equals("") ) {
        Toast.makeText(mContext,
                mContext.getResources().getString(R.string.details_view_send_config_error),
                Toast.LENGTH_SHORT).show();
    } else {
        Session session = createSessionObject();

        try {
            Message message = createMessage(mTo, mSubject, mBody, session);
            new SendMailTask().execute(message);
        } catch (MessagingException e) {
            handleCreateMessageException(e);
        } catch (UnsupportedEncodingException e) {
            handleCreateMessageException(e);
        }
    }
}
```

KUVA 21. sendWithSmtpClient-funktio

SendMailTask on Androidin AsyncTask-luokan laajennus, joka käytännössä luo asynkronisen tehtävän, joka suoritetaan kutsumalla sen execute-metodia (kuva 22). AsyncTaskin avulla voidaan suorittaa raskaampia tehtäviä ilman, että ohjelman käyttöliittymä hidastuu, ja sen laajennokset voivat toteutettaa neljä metodia, onPreExecute, onPostExecute, onProgressUpdate ja doInBackground. Näistä onPreExecute ja onPostExecute kertovat, mitä tehdään, ennen kuin tehtävää aletaan suorittamaan, ja kun tehtävä on saatu suoritetuksi. DoInBackground kertoo mitä tehdään tehtävän asynkronisessa osiossa. DoInBackgroundista voidaan myös kutsua publishProgress-metodia, joka kutsuu luokan onProgressUpdate-metodia halutuilla parametreilla. Näin käyttöliittymää voidaan päivittää kesken tehtävän suorituksen, esim. jos halutaan näyttää käyttäjälle tehtävän tila tai edistymispalkki.

```
private class SendMailTask extends AsyncTask<Message, Void, MessagingException> {
    private ProgressDialog progressDialog;

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // Show a progress dialog with a spinning progress bar while sending
        progressDialog = ProgressDialog.show(mContext,
                mContext.getResources().getString(R.string.details_view_sending_header),
                mContext.getResources().getString(R.string.details_view_sending_content),
                true, false);
    }

    @Override
    protected void onPostExecute(MessagingException e) {
        super.onPostExecute(e);

        progressDialog.dismiss();

        String toastText = "";

        // If an error happened during sending or sending was successful, inform user through
        // a toast message
        if (e != null) {
            toastText =
                    mContext.getResources().getString(R.string.details_view_sending_failed)
                    + ": " + e.getLocalizedMessage();
            Log.e(CLASS, "Error when sending mail!", e);
        } else toastText =
                mContext.getResources().getString(R.string.details_view_sending_successful);
        Toast.makeText(mContext, toastText, Toast.LENGTH_LONG).show();
    }

    @Override
    protected MessagingException doInBackground(Message... messages) {
        try {
            Transport.send(messages[0]);
        } catch (MessagingException e) {
            return e;
        }
        return null;
    }
}
```

KUVA 22. SendMailTask-luokka

Ennen kuin tehtävää aletaan suorittamaan, avataan käyttöliittymälle dialoginäkymä, josta nähdään lähetyksen tila. Tämän jälkeen yritetään lähettää viesti JavaMailin Transport-luokan send-funktion avulla. Jos lähetyksessä tapahtuu virhe, lähetetään se eteenpäin onPostExecutelle, joka tarkistaa siitä, onko lähetys onnistunut vai ei. Kun tehtävä on saatu suoritettua, eli sähköpostiviesti on lähetetty tai lähetyksen on keskeyttänyt virhe, lähetysdialogi suljetaan ja käyttäjälle ilmoitetaan, onnistuiko lähetys.

4.3.5 Lokien visualisointi

Jotta mittalaitteen keräämiä lokeja pääsisi katselemaan nopeammin ja helpommin, ohjelmalta haluttiin valmius piirtää ladatuista lokitiedostoista kuvaajia.

Kun lokitiedostoa klikataan, avautuu mittalaitteen kuvaaja-näkymä, jossa klikatun lokitiedoston sisältämä data piirretään kuvaajalle. Ladattujen lokien visualisoinneista toteutettiin opinnäytetyötä varten vain kiihtyvyysdatan visualisointi. Visualisointi kuitenkin pyrittiin toteuttamaan siten, että uusien suureiden lisääminen tapahtuu helposti.

Ladatusta kiihtyvyysdatasta haluttiin näyttää kaikki kolme akselia sekä kiihtyvyydestä laskettu siirtymä, joka kuvaa mitattavan kohteen paikan muutosta. Kuvaaja-näkymän yläreunassa olevasta painikkeesta käyttäjä voi valita, näytetäänkö kuvaajalla kiihtyvyys vai siirtymä, sekä valita, mitkä akselit ovat näkyvillä.

Kiihtyvyysdataa varten ohjelmaan lisättiin luokka VibrationData, jonka oliot esittävät yksittäisiä kiihtyvyyden lokitiedostoja. Ne pitävät sisällään lokitiedostosta parsitun kiihtyvyysdatan, ja niiden kautta saadaan haettua helposti kunkin akselin data. Ne hoitavat myös siirtymän laskemisen.

Siirtymä laskettiin kaavaa (1) käyttäen, jossa s on siirtymä, a on kiihtyvyys ja f on taajuus (Connection Technology Center Inc. 2013, 54).

$$s = \frac{a}{(2\pi f)^2} \tag{1}$$

Kuvassa 24 on esitetty ohjelmaan toteutettu siirtymän laskentametodi.

```
private float calculateDisplacement(int acceleration, float frequency) {
    if (frequency == 0) return 0;
    return (acceleration * 9807) / Math.pow(2 * Math.PI * frequency, 2);
}
```

KUVA 24. Siirtymän laskenta

Koska mittalaite ilmoittaa kiihtyvyyden g-voimina, se pitää ensin muuntaa SI-järjestelmän mukaiseksi yksiköksi, eli mm/s^2. Yksi g vastaa 9,81 m/s^2. Tällöin saatu g-muotoinen kiihtyvyys saadaan muunnettua mm/s^2-muotoon kertomalla kiihtyvyysarvo g-voiman mm/s^2 –arvolla 9807. Tällöin funktio palauttaa siirtymän millimetreinä. (Connection Technology Center Inc. 2013, 56)

Visualisointia varten ohjelmaan lisättiin AChartEngine, joka on erityisesti Androidille kuvaajien piirtoa varten kehitetty ilmainen Java-kirjasto. Kyseinen kirjasto valittiin, koska siitä oli jo aiemmin kokemusta, ja se keskittyy enemmänkin datan tarkkaan kuvaamiseen kuin ulkoasuun.

AchartEnginellä piirretyt kuvaajat rakentuvat kolmesta pääkomponenttityypistä, jotka ovat datasarja, näkymä ja renderoija.

Datasarja kertoo kirjastolle, mitä tietoa sen pitää näyttää. Kirjastossa sarjaa kuvaava luokka on XYSeries, joka sisältää listan x- ja y-akselin arvoja. Jos x-akselilla on unix-muotoisia aikaleimoja, voidaan käyttää XYSeriesistä laajennettua TimeSeries-luokkaa, jolloin kuvaajan x-akseli indeksoidaan aikaväleillä. Useat datasarjat kootaan yhteen XYMultipleSeriesDataSet-luokalla.

Näkymälle piirretään itse kuvaaja. Näkymä määrittää myös, minkä tyylinen kuvaaja on, eli halutaanko esim. viiva- vai pylväskuvaaja. Kirjastossa näkymää kuvaa AbstractChart-luokka, josta on laajennettu mm. BarChart-, PieChart- ja LineChart-luokat. Näkymiä luodaan kirjaston sisältämillä näkymätehtailla.

Renderoija määrittää, miten kuvaajan sisältö piirretään. Kirjasto käyttää kahden tyyppistä renderoijaa. Ensimmäistä, MultipleXYSeriesRendereriä, käytetään kuvaajan akselien ja tekstien piirron määrittämiseen. Toista, XYSeriesRendereriä, käytetään määrittämään datasarjan piirtoa. MultipleXYSeriesRenderer sisältää XYSeriesRendererin jokaiselle datasarjalle, joka halutaan piirtää kuvaajalle.

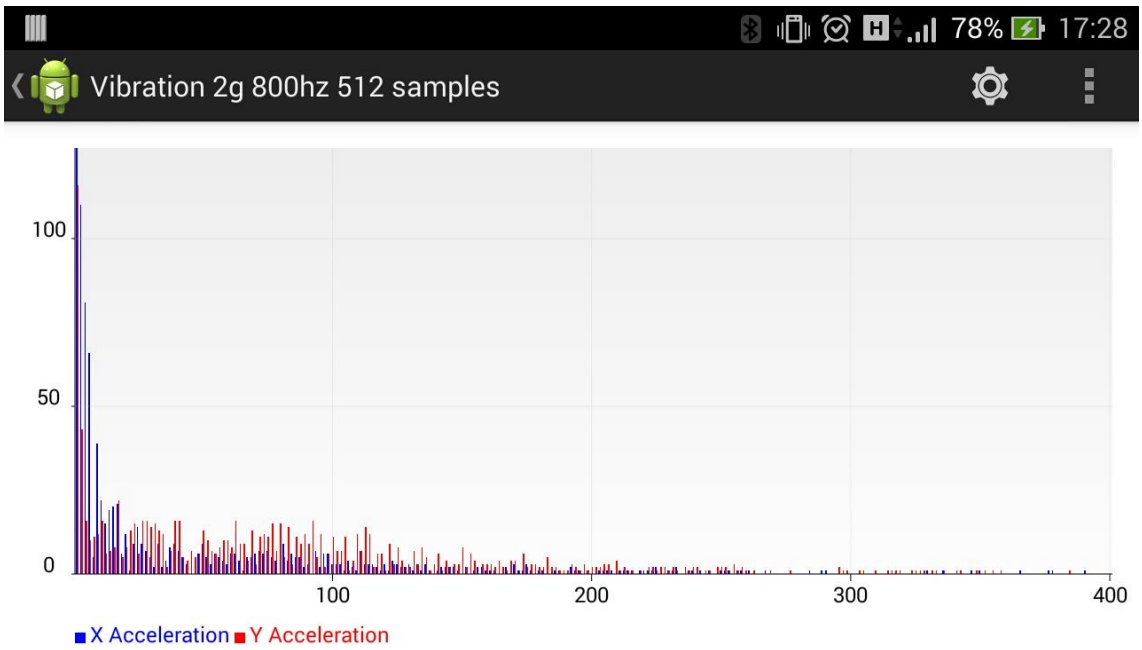
Kiihtyvyysdataa varten valittiin pylväskuvaaja. Kun kuvaajanäkymä avataan, haetaan ensin puhelimen muistista haluttu VibrationData-olio. Tämän jälkeen luodaan XYSeries-oliot jokaiselle kiihtyvyyden ja siirtymän akselille ja haetaan VibrationData-

oliolta data niitä varten. Datasarjat lisätään XYMultipleSeriesDataSet-luokan oliolle, joka annetaan lopussa itse kuvaajan luovan näkymätehtaan parametriksi. Tätä ennen luodaan vielä kuvaajan renderoija ja lisätään siihen renderoijat jokaiselle datasarjalle. Kuvassa 25 on esimerkki siitä, kuinka yhden akselin data lisätään kuvaajalle.

```
// Create series and fetch data
XYSeries xAccelerationSeries = new XYSeries(AxisType.X.name() +
        " " + R.string.chart_data_acceleration);
List<? extends Number> xAccelerationList =
        vibrationData.getSampleList(SampleType.X_ACCELERATION);

// Add data to series
int dataLength = vibrationData.getDataLength();
for (int i = 1; i < dataLength; i++) {
    xAccelerationSeries.add(vibrationData.getFrequency(i),
            AccelerationList.get(i).doubleValue());
}

// Add series to dataset
XYMultipleSeriesDataset mDataset = new XYMultipleSeriesDataset();
mDataset.addSeries(xAccelerationSeries);

// Add renderers for each dataset
XYMultipleSeriesRenderer mRenderer = new XYMultipleSeriesRenderer();
datasetSize = mDataset.getSeriesCount();
for (int i = 0; i < datasetSize; i++)
    mRenderer.addSeriesRenderer(new XYSeriesRenderer());

// Create chart
mChart = ChartFactory.getBarChartView(getActivity(), mDataset, mRenderer, Type.DEFAULT);
```

KUVA 25. AChartEngine-kirjaston esimerkki

Tämän lisäksi kuvaajan ulkoasua muokataan muuttamalla kuvaajan tai datasarjojen renderoijien parametreja. Kuvassa 26 on esitetty ohjelman kuvaajanäkymä.



KUVA 26. Kuvaajanäkymä

4.3.6 Parametrien kirjoitus mittalaitteen sovelluksiin

Viimeisenä haluttuna ominaisuutena oli parametrien kirjoitus mittalaitteen sovelluksiin. Ominaisuus haluttiin ohjelman kehityksen loppuvaiheessa. Se toteutettiin vain kiihtyvyyden mittaukselle, ja sillekkin hyvin alkeellisesti. Ominaisuuden toteutuksessa oli enemmänkin kyse ominaisuuden toiminnallisuuden todistamisesta ja pohjatyön tekemisestä, jotta muidenkin sovellusten parametreihin kirjoittaminen voidaan toteuttaa helposti.

Jos ohjelma lukee mittalaitteen tunnettuihin sovelluksiin liittyvää dataa, laitteen lisätiedoissa näytetään painike, josta pääsee parametrien kirjoitusnäkymään. Tämän näkymän pitää olla jokaiselle mittalaitteen sovellukselle räätälöity, koska parametrit vaihtelevat sovellusten välillä. Koska ominaisuus on toteutettu vain kiihtyvyydelle, näkyy painike vain kiihtyvyyttä mittaavien laitteiden yhteydessä.

Kun parametrien kirjoitusnäkymä aukeaa, mittalaitteelta luetaan sovelluksen nykyiset arvot, jotka listataan käyttöliittymälle. Kun käyttäjä on tehnyt haluamansa muutokset arvoihin, hän voi kirjoittaa uudet arvot mittalaitteen sovellukseen painamalla näkymän pohjalla olevaa kirjoituspainiketta.

5 YHTEENVETO

Vaikka olinkin työskennellyt Android-ohjelmien parissa jo aiemmin niin koulussa kuin työpaikallakin, tämä oli ensimmäinen ohjelma, jonka suunnittelin ja kirjoitin itse alusta asti kyseiselle alustalle. Tämän takia tiettyjä osa-alueita ohjelmasta piti käydä läpi uudelleen muutamaankin kertaan, koska osaaminen alustan suhteen kasvoi työtä tehdessä. Näen, että sain täysin uutta ohjelmaa tehdessäni enemmän kokemusta alustasta, kuin vanhojen ohjelmien muokkaamisesta ja korjaamisesta.

Bluetooth, ja erityisesti Androidin Bluetooth –ohjelmistopino, tulivat työn yhteydessä tutuiksi. Androidin Bluetooth-pino aiheutti opinnäytetyötä tehdessäni ehkä eniten ongelmia ja päänvaivaa. Toisaalta tästä syystä sain erittäin hyvän kuvan siitä, kuinka Android ja kyseinen tekniikka toimivat.

Työ laajensi myös ymmärrystäni TreLab Oy:n pääasiallisesta tuotteesta, Smart Tagista, koska jouduin tutustumaan entistä syvällisemmin myös sen toimintaperiaatteisiin. Tästä on ollut hyötyä opinnäytetyön jälkeisissä tehtävissäni, ja todennäköisesti on tulevaisuudessakin.

Ohjelmaa ei ole opinnäytetyön valmistumisen jälkeen aktiivisesti kehitetty, mutta se on saanut käyttäjiltä muutamia jatkokehitysehdotuksia toiminnallisuuteen liittyen. Ehdotuksia ovat olleet jatkuva skannaus, eli laitteita etsittäisiin jatkuvasti sen sijaan, että se pitää erikseen aloittaa, lokien parsiminen suoraan CSV-tiedostoista, sekä QR-koodien lukumahdollisuus puhelimen kameraa käyttäen. QR-koodi voitaisiin painaa mittalaitteen ulkopintaan. Se sisältäisi mittalaitteen Bluetooth-osoitteen, jolloin siihen voitaisiin yhdistää lukemalla koodi, eikä laitteita tarvitsisi erikseen etsiä ennen yhdistämistä.

Osa ohjelmalta halutuista ominaisuuksista jäi myös hieman keskeneräisiksi osittain senkin takia, ettei niitä oltu määritelty aivan loppuun asti tai haluttiin vain kokeilla, voidaanko kyseinen ominaisuus ensinnäkään toteuttaa alustalla.

Jatkokehitysehdotusten ja ohjelman ominaisuuksien täydentämisen lisäksi ohjelman joitain osa-alueita pitäisi parantaa. Ohjelman näkymärakennetta voitaisiin jakaa

loogisempiin osiin, ja tiedostojen tallennus pitäisi suunnitella uusiksi. Kahden eri tiedoston tallentaminen ja ylläpitäminen tekee tiedostojen tallennuksesta liian monimutkaista ja vaikeasti ylläpidettävää. Tiedostorakenteen parannusten jälkeen puhelimeen tuotujen tiedostojen lukeminen olisi myös mahdollista, kun ohjelma käyttäisi vain yhtä tiedostomuotoa. Ohjelman kehityksessä pitäisi myös siirtyä uudempaan kehitysympäristöön, Android Studioon, koska se toimii nykyään virallisena kehitysympäristönä, eikä Eclipse-pohjaisen kehitysympäristön tuki jatku ikuisesti.

Työn tavoitteena oli toteuttaa TreLab Oy:lle Android-mobiilikäyttöjärjestelmälle ohjelma, jolla voisi etsiä yrityksen tuottamia älykkäitä sensoreita ympäristöstä ja näyttää niiltä tietoa nopeasti ja helposti. Näkisin onnistuneeni tässä tavoitteessa. Ohjelma on ollut käytössä yrityksen sisällä muutamalla henkilöllä, ja vastaanotto on ollut positiivinen. Työ tarjosi sopivasti haastetta ja uutta opittavaa, sekä vanhan jo opitun soveltamista käytännössä. Pidin työn käytännönläheisyydestä ja siitä, että pääsin näkemään työni tulokset, kun ohjelman perustoiminnallisuus saatiin valmiiksi ja se otettiin käyttöön.

LÄHTEET

Aatif Khan. 2013. What Is ART & How Is It Different From Dalvik Virtual Machine On Android? Luettu 23.5.2015. http://www.addictivetips.com/android/art-vs-dalvik-android-runtime-environments-explained-compared/

Adafruit. 2014. Introduction to Bluetooth Low Energy. Luettu 4.3.2015. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt

Android Developer. Application Fundamentals. Luettu 5.9.2015. http://developer.android.com/guide/components/fundamentals.html

Android Developer. Bluetooth Low Energy. Luettu 15.10.2015. https://developer.android.com/guide/topics/connectivity/bluetooth-le.html

Android Developer. Fragments. Luettu 5.9.2015. http://developer.android.com/guide/components/fragments.html

Android Developer. Get started with publishing. Luettu 23.5.2015. http://developer.android.com/distribute/googleplay/start.html

Android Open Source Project. ART and Dalvik. Luettu 23.5.2015. https://source.android.com/devices/tech/dalvik/

Android Team. 2009. Android Anatomy and Physiology. Luettu 23.5.2015. http://androidteam.googlecode.com/files/Anatomy-Physiology-of-an-Android.pdf

Appbrain. 2015. Number of Android applications. Luettu 23.5.2015. http://www.appbrain.com/stats/number-of-android-apps

Apple Inc. 2015. App Store Rings in 2015 with New Records. Luettu 23.5.2015. http://www.apple.com/pr/library/2015/01/08App-Store-Rings-in-2015-with-New-Records.html

Bluetooth SIG. Bluetooth Smart. Luettu 4.3.2015.
http://www.bluetooth.com/Pages/low-energy-tech-info.aspx

Bluetooth SIG. Bluetooth Basics. Luettu 4.3.2015.
http://www.bluetooth.com/Pages/Basics.aspx

Connection Technology Center, Inc. 2013. Beginning Vibration Analysis.
http://www.ctconline.com/pdf/pubTechPapers/01-Beginning%20Vibration%20Analysis.pdf

Eclipse Wiki. 2014. IDE Criticism. Luettu 13.9.2015.
https://wiki.eclipse.org/IDE/Criticism

Gartner. 2014. Gartner Says Sales of Smartphones Grew 20 Percent in Third Quarter of 2014. Luettu 25.3.2015.
http://www.gartner.com/newsroom/id/2944819

Google Play. 2015. Transaction Fees. Luettu 23.5.2015.
https://support.google.com/googleplay/android-developer/answer/112622

Nordic Semi. 2014. A short history of Bluetooth. Luettu 4.3.2015.
https://www.nordicsemi.com/eng/News/ULP-Wireless-Update/A-short-history-of-Bluetooth

Steve Brachmann. 2014. A Brief History of Google's Android Operating System. Luettu 25.3.2015.
http://www.ipwatchdog.com/2014/11/26/a-brief-history-of-googles-android-operating-system/

Techotopia. 2014. An Overview of the Android Architecture. Luettu 23.5.2015.
http://www.techotopia.com/index.php/An_Overview_of_the_Android_Architecture

TreLab Oy. Smart Data Mill. Luettu 17.6.2015
http://www.trelab.fi/eng/smart-data-mill/