Joni Lehtinen

# Android-paikannussovellus ja -palvelin

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tietotekniikan koulutusohjelma

Insinöörityö

26.11.2016

Metropolia

| | |
|---|---|
| Tekijä(t)<br>Otsikko<br><br>Sivumäärä<br>Aika | Joni Lehtinen<br>Android-paikannussovellus ja -palvelin<br><br>46 sivua<br>26.11.2016 |
| Tutkinto | Insinööri (AMK) |
| Koulutusohjelma | Tietotekniikan koulutusohjelma |
| Suuntautumisvaihtoehto | Ohjelmistotekniikka |
| Ohjaaja(t) | Lehtori Juha Kämäri |


Insinöörityö syntyi halusta tehdä sijaintitietoihin perustuva mobiiliohjelma, jonka avulla ystävät ja perheenjäsenet voivat seurata toistensa sijainteja. Yksi sovelluksen mahdollisista käyttötapauksista on auttaa vanhempia näkemään lastensa reaaliaikaiset sijaintitiedot, mikä tarjoaa turvallisuutta ja mielenrauhaa.

Insinöörityön tavoitteena oli kehittää sijaintitietoihin perustuva asiakasohjelma sekä palvelin datan hallinnointiin. Asiakasohjelmasta tavoitteena oli tehdä vähävirtainen ja helppokäyttöinen Android-paikannussovellus, jossa käyttäjä voi hallita ryhmiä ja nähdä ryhmän jäsenten paikannustiedot kartalla. Palvelimesta oli tavoitteena saada skaalautuva ja tietoturvallinen ja sen pitäisi kommunikoida salattua yhteyttä käyttäen asiakasohjelmien kanssa.

Palvelin toteutettiin Javalla puhtaalta pöydältä. Samalla käytettiin sen tarjoamia yhteys- ja salausmenetelmiä. Toteutuksena syntyi ei-odottava, monisäikeinen palvelin, jossa pieni ryhmä säikeitä suorittaa työjonoa aina, kun tämä voidaan tehdä ilman odotusta. Palvelimen ja asiakasohjelman väliseen kommunikointiin luotiin oma helppokäyttöinen protokolla, joka on mahdollista vaihtaa helposti. Kyseinen protokolla rakennettiin Javan TLS-toteutuksen päälle.

Asiakasohjelmasta syntyi tyylikäs kokonaisuus, jossa käyttäjä hallitsee ryhmiä ja näkee niiden jäsenten sijaintitiedot reaaliajassa Google-karttoja käyttäen. Sovelluksesta saatiin vähävirtainen käyttämällä Google Play -palvelun paikannusrajapintaa. Siitä tuli myös helppokäyttöinen ja visuaalisesti tyylikäs sen yksinkertaisten grafiikoiden ansiosta.

Insinöörityön tavoitteet saavutettiin ja lopputuote on toimiva kokonaisuus. Se vaatii kuitenkin jatkokehitystä, jotta siitä saataisiin kaupallinen tuote.



| | |
|---|---|
| Avainsanat | Android, Java, palvelin, GPS, salaus |


Metropolia

| Author(s)<br>Title<br><br>Number of Pages<br>Date | Joni Lehtinen<br>Location Based Android Application and Server<br><br>46 pages<br>26 November 2016 |
|---|---|
| Degree | Bachelor of Engineering |
| Degree Programme | Information Technology |
| Specialisation option | Software Programming |
| Instructor(s) | Juha Kämäri, Senior Lecturer |

The objectives were to develop a location based android application backed by a server. The Android application was to be built with low-power consumption and easy-to-use interface in mind. Its function is to provide users the means of tracking friends that have joined their circle and the management of those circles. Communication with the server was to be encrypted and the server made scalable and secure.

The server was built with the Java programming language using its connection and encryption APIs. The end product was a non-blocking, multi-threaded server where a small group of threads executes a work queue every time it can be done without blocking. Server-client communication was realized with a self-made easy-to-use communication protocol that could easily be replaced. The communication protocol was built on top of Java's TLS implementation.

Client was created with a stylish design, where a user controls groups and sees their members' location data in real-time with the help of Google maps. The use of Google Play service's location API helped to lower the application power consumption. Its easiness to use and visual styles were achieved with the use of a simplistic look and feel.

Objectives for the thesis were all met and the end product is functional. It still requires further development for it to be a commercial product.

| Keywords | Android, Java, Server, GPS, encryption |
|---|---|

## Sisällys

Lyhenteet

1 Johdanto 1

2 Määrittely ja tavoitteet 1

2.1 Palvelinohjelma 2
2.2 Asiakasohjelma 2
2.3 Tietokanta 3
2.4 Yhteydet 3

3 Teknologiat 5

3.1 Android 5
3.1.1 Historia 5
3.1.2 Avoimen lähdekoodin Android 6
3.1.3 Versiot ja niiden valinta 6
3.1.4 Ohjelman rakenne 7
3.2 SSL ja TLS 16
3.2.1 TLS Record -kerros 17
3.2.2 TLS Handshake -kerros 18
3.2.3 Asymmetrinen ja symmetrinen algoritmi 20
3.2.4 Sertifikaatti 20
3.3 JDBC 21
3.4 JCA 21
3.5 JSSE 22
3.5.1 SSLContext 23
3.5.2 SSLSocket ja SSLServerSocket 23
3.5.3 SSLEngine 24
3.6 SQL 25
3.6.1 SQLite 25
3.6.2 PostgreSQL 26
3.7 Apache Commons DBCP 2 26

4 Toteutus 27

4.1 Java-palvelinohjelma 28

4.1.1 Säiemalli ja siirtokerros 28
4.1.2 Tietokanta 32
4.1.3 Salaus ja yhteyden suojaus 32
4.2 Android-asiakasohjelma 34
4.2.1 Näkymät 35
4.2.2 Service ja säikeet 38
4.2.3 Paikannus 40
4.2.4 Oikeudet ja asetukset 41
4.3 Yhteysprotokolla 42
5 Yhteenveto 42
Lähteet 44

## Lyhenteet

| | |
|---|---|
| GPS | Global Positioning System. Maailmanlaajuinen satelliittinavigaatiosysteemi, joka tarjoaa sijainti- sekä aikatiedot joka puolelle maata. |
| OHA | Open Handset Allience. Yrityksistä koostuva yhteisö, jonka tavoite on kehittää avoimia standardeja mobiililaitteille. |
| AOSP | Android Open Source Project. Nimitys Googlen Android-projektille. |
| API | Application Programming Interface. Ohjelmointirajapinta. |
| XML | Extensible Markup Language. Rakenteellinen merkkauskieli. |
| SSL | Secure Socket Layer. Salaustekniikka kahden yhteyden välillä. |
| TLS | Transport Layer Security. Salaustekniikka kahden yhteyden välillä. |
| HMAC | Keyed-Hashing for Message Authentication Code. Kryptografinen hash-funktio. |
| TCP | Transmission Control Protocol. Siirtokerrosprotokolla, joka tarjoaa luotettavan tiedon siirron virheentarkistuksella. |
| JDBC | Java Database Connectivity. Java-rajapinta tietokantayhteyden luomiseen. |
| CLI | Call Level Interface. Ohjelmistostandardi tietokannan kanssa kommunikointiin SQL-kielellä. |
| JCA | Java Cryptography Architecture. Sovelluskehys kryptografisten palveluiden kehittämiseen |
| ACID | Atomicity, Consistency, Isolation, Durability. Tietokanta-transaction-määritelmä. |
| PBKDF2 | Password-Based Key Derivation Function 2. Algoritmi/Funktio salasanan muuttamiseksi johdetuksi avaimeksi. |

# 1 Johdanto

Nykyään jokaisen omistaessa mobiililaitteen on kommunikointi helpottunut valtavasti. Toisinaan tulee kuitenkin tilanteita, jolloin haluaisi löytää toisen helposti, mutta pelkkä viesti ei siihen riitä. Tarvitaan siis graafinen kuvaus ihmisen sijainnista kyseisellä ajan hetkellä. Tästä syntyi idea tehdä ohjelma, joka auttaa ihmisiä löytämään toisensa jokaisena ajan hetkenä. Ohjelma tarjoaa käyttäjälle kartan, josta nopeasti ja helposti on nähtävissä muiden käyttäjien sijainnit, osoitetiedot sekä aika, jolloin data on rekisteröity. Toimiakseen asiakasohjelmat tarvitsevat palvelimen, jolta ne pystyvät hakemaan muiden käyttäjien sijaintitiedot.

Tavoitteena insinöörityössä on tehdä GPS-paikannusta käyttävä Android-asiakasohjelma, joka näyttää ja välittää paikannusdataa, sekä Java-palvelin, jonka kanssa asiakasohjelma kommunikoi. Asiakasohjelmasta on tavoitteena saada vähän virtaa kuluttava helppokäyttöinen sovellus, joka lähettää dataa aina tietyn ajan välein puhelimen ollessa suljettunakin. Palvelinohjelman tavoitteena on saada skaalautuva tämänhetkistä tietoturvatasoa vastaava ohjelma, joka salaa verkkoliikenteen sekä salasanatiedot murtautumattomalla algoritmilla.

Insinöörityö määrittelee lukijalle ensin tehtävän ohjelmiston sekä sen tavoitteen. Tämän jälkeen perehdytetään lukija käytettyihin teknologioihin, mikä antaa lukijalle hyvän tietotaidon toteutuksen ymmärtämiseen. Lopputuotteena lukijalle jää selkeä kuva käytetyistä teknologioista sekä ohjelmistojen toteutuksista.

# 2 Määrittely ja tavoitteet

Tässä luvussa tarkastelemme syvemmin ohjelmistojen määrittelyä sekä niiden tavoitteita. Komponentit on jaettu omiin ala-lukuihinsa, jotta jokaisella komponentilla olisi selkeä määrittely ja tavoite.

## 2.1 Palvelinohjelma

Palvelin yhdistää käyttäjät toisiinsa ja tarjoaa asiakasohjelmille kanavan tiedon jakamiseen. Ominaisuuksiltaan palvelinohjelman tulee olla skaalautuva ja tietoturvallinen. Yhteys asiakasohjelmiin tulee toteuttaa suojatusti ja niiden tulee käyttää yhteistä protokollaa viestien välitykseen. Tarkempi määrittely tästä löytyy luvusta 2.4. Palvelin yhdistää asiakasohjelman ja tietokannan tarjoten rajapinnan tiedonhakua ja muokkausta varten (kuva 1). Tämän rajapinnan tulee suojata tietokantaa hyökkäyksiltä tarkistamalla asiakasohjelmalta saadut pyynnöt hyökkäysten varalta.

## 2.2 Asiakasohjelma

Asiakasohjelma toteutetaan Android-alustalle. Toteutuksessa pyritään tekemään helppokäyttöinen, mutta visuaalisesti tyylikäs lopputuote. Ohjelmaan mahtuisi monia ominaisuuksia, mutta otamme tähän insinöörityöhön mukaan vain ne ominaisuudet, jotka antavat käyttäjälle mahdollisuuden perusasioiden toteuttamiseen. Luvun 5 yhteenvedossa käydään tarkemmin läpi sovelluksen jatkokehitystä ja sen lisäominaisuuksia.

Sovelluksen käyttöliittymään sisältyy seuraavat näkymät

- kirjautuminen ja rekisteröinti
- ryhmien luonti, poisto sekä jäsenten hallinnointi
- ryhmäpyynnön hyväksyminen ja hylkääminen
- karttanäkymä ryhmän jäsenten sijainnista sekä näkymä osoitteen, tarkkuuden ja ajan sisältävästä sijaintitiedosta.

Asiakasohjelma perustuu ryhmän jäsenten jakamiin sijaintitietoihin ja vaatii toimiakseen käyttäjän rekisteröitymisen/kirjautumisen. Tämän jälkeen käyttäjä on vapaa luomaan ryhmiä sekä liittymään niihin. Ryhmäpyynnöt tulee voida hyväksyä tai hylätä ennen kuin käyttäjä lisätään kyseiseen ryhmään. Itse karttanäkymä on sovelluksen sydän. Sen pitää pystyä antamaan käyttäjälle sijaintitiedot ryhmän jäsenistä sekä tarkempi data, milloin ja millä tarkkuudella kyseinen sijainti on hankittu.

Sovelluksen toiminnallisuuteen sisältyy seuraavat ominaisuudet

- suojatun yhteyden muodostaminen palvelimeen
- kommunikointi palvelimen kanssa yhteistä protokollaa käyttäen
- sijaititiedon lähetys palvelimelle tietyn ajan välein käyttäjän ollessa kirjautuneena sisään, riippumatta siitä, onko ohjelma päällä
- sijaintitietojen hakeminen palvelimelta tietyn ajan välein, kun käyttäjä on kirjautuneena sisään ja sovellus on päällä.

Sovellus perustuu paikannustietoihin, joita käyttäjä pitää omana henkilökohtaisena tietona. Tätä varten on muodostettava suojattu yhteys palvelimeen, jotta käyttäjän sijaintitiedot eivät leviäisi kolmansille osapuolille. Käyttäjälle on tärkeää, että ystävien sijaintitiedot eivät ole vanhoja. Näin vähennetään käyttäjien tarvetta arvuutella ystävien sijaintia. On siis tärkeää lähettää käyttäjän sijaintitiedot palvelimelle tihein väliajoin. Yhtä tärkeää on myös hakea ryhmän jäsenten sijaintitiedot palvelimelta tietyn ajan välein ohjelman ollessa päällä. Sovelluksessa tulee myös olla mahdollista päivittää karttanäkymä käyttäjän pyynnöstä.

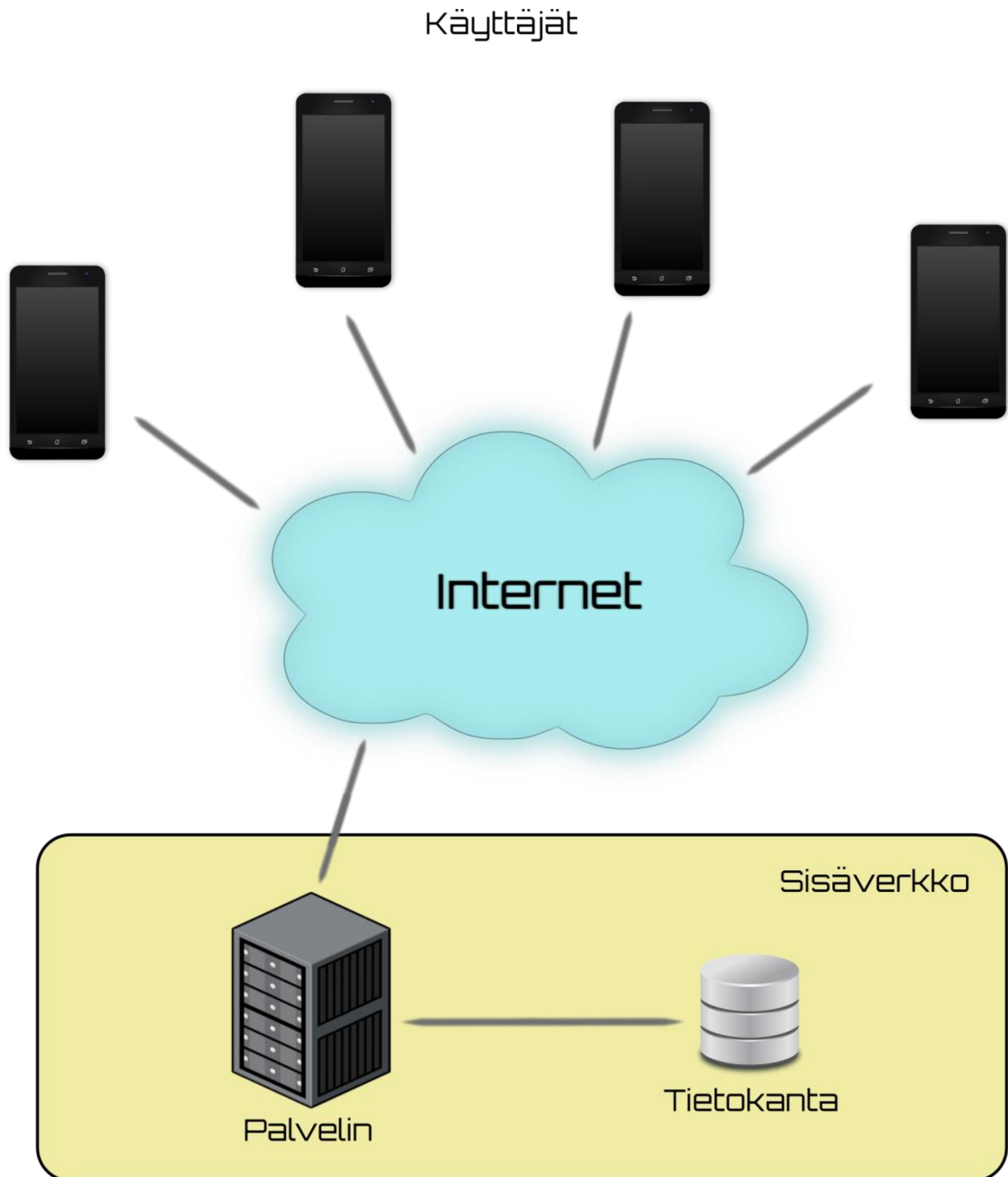
## 2.3 Tietokanta

Tietokannaksi tulee valita relaatiotietokanta, joka tukee SQL-kieltä. Sen tulee vastata palvelimen pyyntöihin tarpeeksi nopeasti. Tietokannan optimointi on kuitenkin tämän insinöörityön ulkopuolella.

## 2.4 Yhteydet

Asiakasohjelma ja palvelin kommunikoivat internetin välityksellä (kuva 1). Asiakasohjelman lähettämien pakettien tulee selvitä perille kokonaisina ja virheettöminä salatussa muodossa. Salaus tulee hoitaa nykyaikaisella menetelmällä, jota pidetään turvallisena

vaihtoehtona internetissä liikkuvan datan salaamiseen. Jotta palvelin ymmärtää asiakasohjelman pyyntöjä, täytyy kyseisille komponenteille määritellä yhteinen protokolla datan lähetykseen ja vastaanottamiseen.




Kuva 1. Ohjelmistojen yhteysdiagrammi.

Toisin kuin asiakasohjelmat, tietokanta sijaitsee sisäverkossa palvelimen kanssa (kuva 1). Tällöin vain palvelin on yhteydessä tietokantaan, poistaen tarpeen yhteyden suojaamiselle.

# 3 Teknologiat

Tässä luvussa käydään läpi insinöörityössä käytetyt tekniikat havainnollistaen niiden käyttöä kuvilla ja ohjelmakoodin pätkillä. Luvun tarkoitus on antaa lukijalle kattava kuva eri teknologioiden toimintaperiaatteista ja auttaa ymmärtämään luvun 4 toteutus.

## 3.1 Android

Android on Googlen kehittämä, avoimeen lähdekoodiin perustuva mobiilikäyttöjärjestelmä. Viimeisimpien laskelmien mukaan Androidin markkinaosuus oli vuoden 2016 toisella kvartaalilla 87,6 %. [1.] Android on selkeästi maailman käytetyin mobiilikäyttöjärjestelmä. Sen suosio perustuu avoimeen lähdekoodiin ja siihen, että puhelinvalmistajat voivat käyttää sitä omiin puhelimiinsa ilman lisenssimaksuja samalla muokaten siitä oman näköisensä. Ymmärtääkseen paremmin, mitä on Android, täytyy meidän siirtyä ajassa hieman taaksepäin. [3.]

### 3.1.1 Historia

Android sai alkunsa vuonna 2003 Andy Rubinin perustaessa Android Inc:n. Yrityksen tavoite oli luoda seuraavan sukupolven mobiilikäyttöjärjestelmä, minkä päälle kehittäjien olisi helppo rakentaa omia ratkaisujaan. Parin vuoden kehityksen jälkeen Android Inc:n lähtiessä etsimään rahoittajia oli Google myös samaan aikaan etsimässä itselleen mobiilikäyttöjärjestelmää. He halusivat tuoda oman hakukoneensa puhelimiin, ja Android tarjosi tähän hyvät mahdollisuudet. Vuoden 2005 lopussa Google osti Android Inc:n itselleen, ja Andy Rubin sekä hänen tiiminsä vaihtoivat toimistoa. Pari vuotta myöhemmin vuonna 2007 OHA (Open Handset Allience) perustettiin ja AOSP (Android Open Source Project) syntyi. Open Handset Allience on yrityksistä koostuva yhteisö, jonka tarkoitus on kehittää avoimia standardeja mobiilialustoille. Yhteisön ensimmäinen projekti oli Android Open Source Project, joka nimensä mukaan tarkoittaa avoimenlähdekoodin Android-projektia. Vuoden ja muutaman prototyypin jälkeen ensimmäinen Android-puhelin olikin jo markkinoilla ja Android oli kaikkien saatavilla. [2; 3.]

### 3.1.2 Avoimen lähdekoodin Android

Android eli Android Open Source Project perustuu avoimeen lähdekoodiin. Näin ollen kuka tahansa pystyy tekemään Androidista oman version ja jopa kilpailemaan sillä Androidia vastaan. Android ei kuitenkaan ole pelkkä käyttöjärjestelmä vaan siitä on syntynyt ekosysteemi. Tämän ekosysteemin keskellä ovat Googlen palvelut kuten Gmail, Maps, Youtube ja Play Store, jotka tekevät Androidista sen, mitä se nykyään on. Saadakseen palvelut omaan Android-versioon yrityksen on saatava lisenssi niihin. Lisäksi yrityksen on otettava kaikki Google Apps -paketin ohjelmat käyttöönsä omassa Android-toteutuksessaan. Lisenssin saaminen onnistuu ilman OHA:aan liittymistä, mutta liittyminen tekee siitä paljon helpompaa. Saadakseen lisenssin on yrityksen lupauduttava tekemään Android-versio, joka ei kilpaile Googlen Android-version kanssa. Tämä siis tarkoittaa, että yritys lupautuu rakentamaan yhteistä Android-ekosysteemiä. Toinen, mikä estää yrityksiä tekemästä Androidin kanssa kilpailevaa versiota, on Google API -palvelut, jotka vaativat toimiakseen Google Apps -paketin sisällyttämisen Android-versioon. [3.]

Kehittäjät haluavat tehdä ohjelmansa nopeasti ja tehokkaasti. Google tarjoaa tähän kehittäjille oman ohjelmointirajapinnan eli API:n (Application Programming Interface), jolla kehittäjät voivat nopeasti luoda toimivia ohjelmistokokonaisuuksia. Google API -palvelun tärkeimpiä ohjelmistorajapintoja ovat Maps API, Location API sekä Play Games API. Vaikka itse Android on avoimen lähdekoodin projekti, on Google viemässä sitä enemmän suljetun lähdekoodin suuntaan. Hyvä esimerkki tästä on vuonna 2013 tullut paikannusohjelmistorajapinta eli Google Location API. Sijainnin voi nykyään hankkia, joko avoimen lähdekoodin virtaa syövällä tavalla tai suljetun lähdekoodin vähävirtaisella tavalla. Android on siis avoimen lähdekoodin käyttöjärjestelmä, jota Google tarkkaan kontrolloi omilla suljetun lähdekoodin ohjelmilla sekä rajapinnoilla. [3.]

### 3.1.3 Versiot ja niiden valinta

Google pyrkii parantamaan Android-käyttöjärjestelmäänsä jatkuvasti tuoden sinne uusia ominaisuuksia käyttäjän sekä kehittäjän hyödyksi. Kehittäessä ohjelmaa Android-alustalle on versioiden tunteminen erittäin tärkeää, etenkin se, mitä uutta ne tuovat mukanaan. Taulukko 1 esittää tämän hetken versiotilannetta. Kyseinen taulukko on erinomainen lähde pienintä versiota valittaessa. Yleensä Androidille ohjelmaa kehitettäessä tähdätään moneen eri käyttöjärjestelmäversioon. Projektin alussa ohjelman tekijä päättää

pienimmän tuetun käyttöjärjestelmäversion, ja kehittää ohjelmaa Android-käyttöjärjestelmille, jotka ovat suurempia tai yhtä suuria kuin tämä versio. Pienintä versiota valittaessa on tärkeää tietää, että puhelinvalmistajat päivittävät puhelimissaan olevia Android-versioita ja näin ollen versiojakauma muuttuu ajan kanssa. Uusimmista puhelimista löytyy myös yleensä uusin Android-versio, joten kaikkein uudempien Android-alustojen käyttäjä määrä kasvaa myös tätä kautta. Kehittämisen alussa onkin syytä miettiä, kauanko kehittäminen kestää, ja valita pienin versio niin, että se käsittää suurimman osan käyttäjistä, kun ohjelma julkaistaan. On myös tärkeää tietää, mitä tekniikoita ohjelman tekemiseen tarvitsee käyttää. Jos tuki tekniikkaan tuli tietyssä versiossa, on silloin paras valita se pienimmäksi tuetuksi versioksi. Versiota valittaessa on myös hyvä ottaa huomioon se, että mitä enemmän Android-alustoja on tuettava, sitä enemmän on ohjelmaa testattava ja näin ollen ohjelman kustannukset kasvavat. [5.]

Taulukko 1. Android-versiojakauma 7.11.2016. [4.]

| Version | Codename | API | Distribution |
|---|---|---|---|
| 4.0.3 - 4.0.4 | Ice Cream Sandwich | 15 | 1.3% |
| 4.1.x | Jelly Bean | 16 | 4.9% |
| 4.2.x | | 17 | 6.8% |
| 4.3 | | 18 | 2.0% |
| 4.4 | KitKat | 19 | 25.2% |
| 5.0 | Lollipop | 21 | 11.3% |
| 5.1 | | 22 | 22.8% |
| 6.0 | Marshmallow | 23 | 24.0% |
| 7.0 | Nougat | 24 | 0.3% |

### 3.1.4 Ohjelman rakenne

#### Android Manifest

Android Manifest on xml-tiedosto, jonka jokainen ohjelma tarvitsee toimiakseen. XML (Extensible Markup Language) on merkkauskieli, jossa asiat ilmaistaan puun tapaisessa rakenteessa. XML koostuu elementeistä ja attribuuteista, joiden nimet järjestelmän tekijä

voi itse päättää. Kuvassa 2 on supistettu versio insinöörityössä tehdystä Android-ohjelman manifestista. Siinä elementit ovat sinisellä värillä ja attribuutit vihreällä. Se pitää sisällään ohjelman pohjapiirustuksen määritellen, minkälaisia komponentteja ohjelmasta löytyy. Itse kuva on suuntaa antava, jossa on esitelty pääpiirteittäin, minkälaisista komponenteista ohjelma koostuu. Toimiakseen ohjelman manifestin täytyy sisältää manifest- ja application-elementit yhden kerran esiteltynä, sekä ainakin yhden muun elementin application-elementin sisällä. [6; 7.]

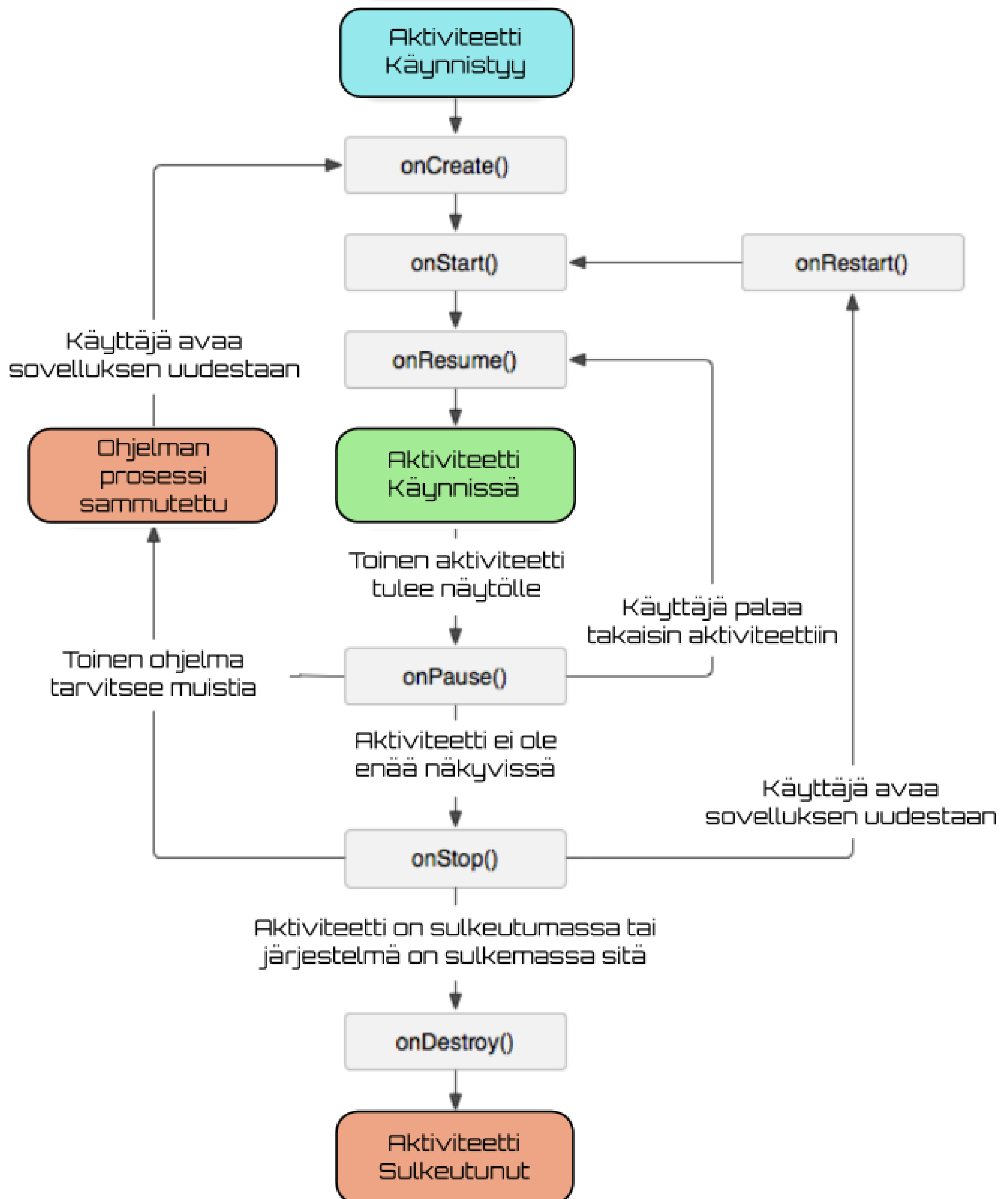
```
<manifest . . . >
    <user-permission android:name="android.permission.ACCESS_FINE_LOCATION" />
    . . .
    <application . . . >
        <activity android:name=".MainActivity"
                . . . >
            . . .
        </activity>
        <receiver android:name=".OnBootReceiver" />
        <service android:name=".ServerConnectionService" />
        <provider android:name=".DatabaseProvider"
                  android:authorities="fi.joni.lehtinen.friendfinder.provider" />
    </application>
</manifest>
```

Kuva 2. AndroidManifest.xml-tiedoston sisältö.

### Activity

Activity eli suomeksi aktiviteetti on Android-ohjelman komponentti, joka tarjoaa käyttäjälle näkymän, minkä kanssa käyttäjä voi tehdä toimenpiteitä. Näkymä voi olla koko ruudun kokoinen tai viedä osan ruudusta leijuen edellisen aktiviteetin päällä. Yleensä Android-ohjelma koostuu monista aktiviteeteistä, jotka ovat jotenkin sidoksissa toisiinsa. Ne tarjoavat käyttäjälle näkymän, jonka kanssa käyttäjä voi tehdä aktiviteetin tarjoamia palveluita. Jokainen aktiviteetti voi käynnistää oman ohjelman aktiviteetin sekä myös toisen ohjelman aktiviteetin, jos kyseinen ohjelma ei ole kieltänyt sitä omassa manifest-tiedostossaan. Aktiviteetin käynnistyksen muilta ohjelmilta voi kieltää lisäämällä manifest-tiedostoon kyseisen aktiviteetin elementtiin android:exported-attribuutin ja antamalla sille arvon false. Tällöin vain oman ohjelman aktiviteetit voivat käynnistää sen. Uuden aktiviteetin käynnistyessä pistetään se pinon päällimmäiseksi ja aukaistaan se näytölle.

Käyttäjän ollessa valmis uuden aktiviteetin kanssa tai painaessa takaisin painiketta, tuhotaan kyseinen aktiviteetti pinon päältä ja avataan pinossa seuraavaksi päällimmäisenä oleva edellinen aktiviteetti näytölle. [8.]




Kuva 3. Aktiviteetin elämänkaari [8.]

Käyttäjä yleensä huomaa vain kuin aktiviteetti käynnistyy ja sulkeutuu, mutta aktiviteetilla on kolme tilaa sen ollessa päällä. Kuvassa 3 esitellään aktiviteetin elämänkaari ja miten käyttöjärjestelmän tai käyttäjän tekemät asiat muuttavat sitä. Asioiden tekemiseksi tietyssä ohjelman tilassa voi kehittäjä ylikirjoittaa kuvassa 3 olevia metodeja, joita käyttöjärjestelmä kutsuu ajon aikana. [8.]

Aktiviteetin ollessa päällä voi se olla seuraavissa tiloissa.

- Käynnissä tila – Aktiviteetti on näytöllä päällimmäisenä ja sillä on käyttäjän huomio.

- Pysäytetty tila – Aktiviteetti näkyy vielä, mutta toinen aktiviteetti on sen päällä ja sillä on käyttäjän huomio. Aktiviteetti on päällä se säilyttää tilansa ja on vielä kiinnitettynä ikkunamanageriin.

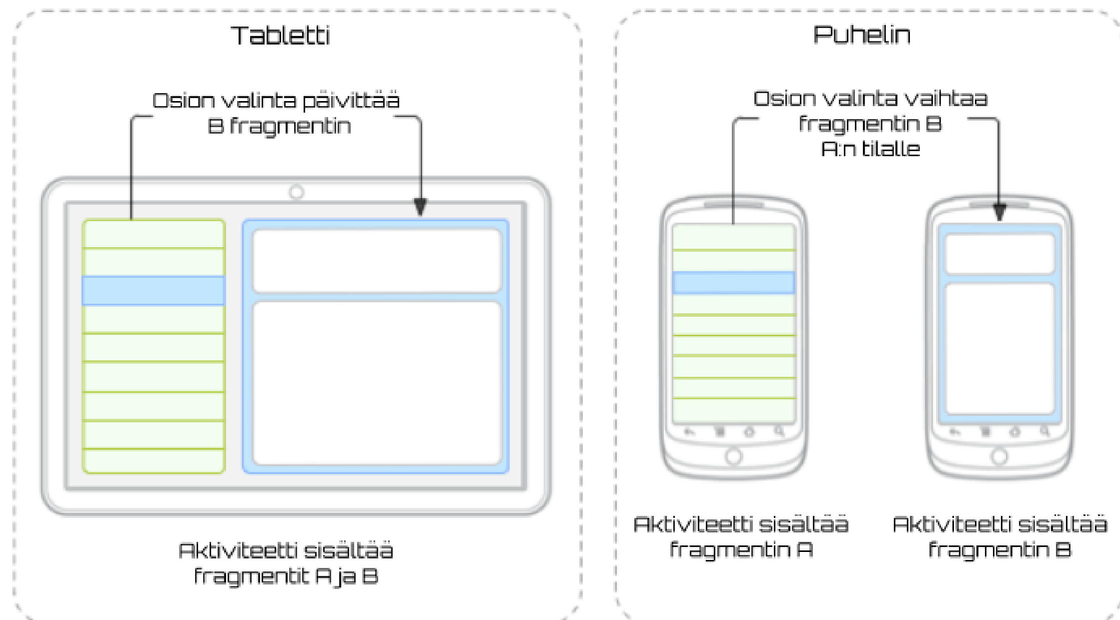
- Lopetettu tila – Aktiviteetti ei näy käyttäjälle. Se säilyttää tilansa samalla tavalla kuin pysäytetty tila, mutta se on poistettu ikkunamanagerista. [8.]

Kuvassa 3 esittelimme, että käyttöjärjestelmä voi tuhota aktiviteetin, jos se on lopetetussa tilassa ja käyttöjärjestelmä tarvitsee resursseja. Yleinen syy aktiviteetin tuhoamiseen on käyttöjärjestelmän asetuksien muutokset, joita ovat esimerkiksi näytön orientaation sekä kielen muutos. Näytön orientaation muutoksen tapahtuessa voi olla tarve uuden layout-tiedoston lataamiseen, jonka takia aina aktiviteetti tuhotaan ja käynnistetään uudelleen. [8.]

Fragment

Aktiviteetit tarjosivat ennen käyttäjälle näkymän, jota aktiviteetit itse hallitsivat. Tämä kuitenkin tuotti ongelmia tablettien yleistyessä, koska aktiviteetit eivät voi sisältää toisia aktiviteetteja. Se taas esti niiden uudelleen käytön sekä vaikeutti niiden muuttamista ajon aikana. Nykyään aktiviteetit eivät enää tarjoa käyttäjälle omaa näkymää, vaan aktiviteettien näkymä koostuu paikoista, mihin fragmentit voivat kiinnittää oman näkymänsä. Fragmentin näkymä näytetään käyttäjälle aktiviteetin määrittelemässä kohdassa. Kyseistä näkymää hallitsee ainoastaan fragmentti, mutta aktiviteetti voi myös vaikuttaa fragmentissa tapahtuviin toimintoihin, jos fragmenttiin on luotu rajapinta kommunikaatiota varten.

Aktiviteetin tarjoama näkymä voi sisältää yhden tai monta fragmenttia. Fragmentit ovatkin hyviä rakennuspaloja, kun aktiviteetti haluaa tarjota erilaisen näkymän erikokoisille laitteille. Kuva 4 antaa tästä hyvän esimerkin. Siinä kummatkin fragmentit ovat vierekkäin näkyvillä tabletilla, mutta puhelimella, jossa tilaa on vähemmän, ovat fragmentit omina näkyminään. [9.]



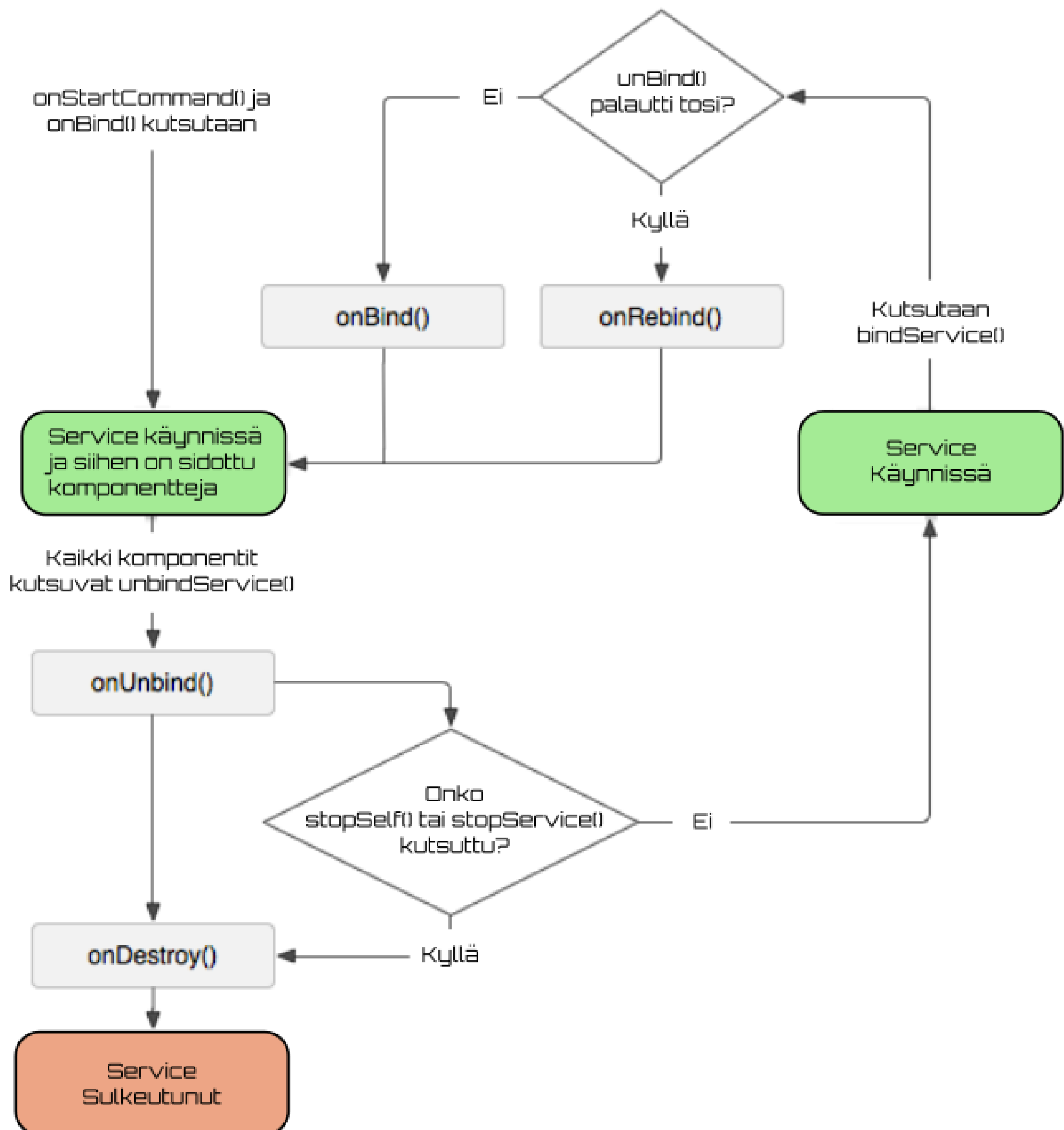

Kuva 4. Aktiviteetin näkymä eri laitteilla käyttäen samoja Fragment-toteutuksia. [9.]

Elämänkaari fragmentilla on samanlainen aktiviteetin kanssa. Fragmentin ollessa tietyssä tilassa on aktiviteetin oltava samassa tilassa. Ainoa ero aktiviteettiin on lopetettu tila, jossa fragmentti poistetaan aktiviteetista ja se lisätään fragmenttimanagerin pinoon. Kyseinen fragmenttimanageri hallitsee aktiviteetin fragmentteja. Sen tehtävä on lisätä ja poistaa fragmentit aktiviteetista sekä hallita fragmenttipinoa. Kyseinen pino toimii samalla periaatteella kuin aktiviteetissa, ja sen navigointi tapahtuu puhelimen peruutusnäppäimellä. [9.]

## Service

Service on ohjelmakomponentti, joka ei sisällä käyttöliittymää. Se on tarkoitettu tekemään pitkään kestäviä toimintoja ja se voikin toimia taustalla loputtomasti. Toimiakseen service ei tarvitse aktiviteettia, mutta se tarvitsee jonkun komponentin itsensä käynnistämiseen. Servicen voi käynnistää ilman sitomista kutsumalla metodia Context.startService() tai sitomalla aktiviteettiin kutsumalla metodia Context.bindService(). Kuvan 5 diagrammissa esitetään servicen elämänkaari, kun siihen sidotaan komponentteja. Kuvasta

ilmenee, että myös sitomaton service voidaan sitoa. Tällöin service ei pysähdy, vaikka kaikki komponentit kutsuvat Context.unbindService(). Sen pysäyttäminen täytyy hoitaa, joko komponentin tai käynnissä olevan servicen toimesta. Edellä mainittu Context-luokka on yliluokka kaikille Android-pääkomponenteille. Vaikka service onkin tarkoitettu pitkää kestäviin toimintoihin, ei se itse toteuta erillistä säiettä vaan pyörii ohjelman käyttöliittymäsäikeessä. On tärkeää, että kehittäjä itse luo säikeen tai käyttää IntentService-luokkaa, jossa tämä on tehty valmiiksi. [10.]




Kuva 5. Sidotun servicen elämänkaari. [11.]

Kommunikaatio servicen ja sen käynnistäneen komponentin kanssa tapahtuu IBinder-luokan kautta. Service toteuttaa kyseisen luokan ja käynnistävän komponentin sitoessa

itsensä serviceen palauttaa se IBinder-toteutuksen. Kun komponentti on sidottu serviceen, kutsuu käyttöjärjestelmä komponentin ServiceConnection.onServiceConnected()-metodia antaen tälle IBinder-ilmentymän parametrina. Tämän jälkeen komponentti on valmis kutsumaan servicen palveluja. Toisensuuntainen kommunikaatio toteutetaan ResultReceiver-luokalla. Servicen käynnistäneen komponentin on toteutettava ja välitettävä tämä servicelle. Luokan toteutus perustuu samaan IBinder-rajapintaan. [10; 12.]

Content Provider

Content Provider tarjoaa yhtenäisen rajapinnan datan hallinnointiin (kuva 7). Kaikille ohjelmille tarjotaan mahdollisuus kyseisen datan hankkimiseen, ellei sitä kielletä manifestissa android:exported=false-attribuutilla. Kuvassa 6 on esimerkki insinöörityössä olevan Content Providerin manifest-esittelystä, jossa kyseinen asia on kiellettynä. [13.]

```
<provider
    android:name=".DatabaseProvider"
    android:authorities="fi.joni.lehtinen.friendfinder.provider"
    android:enabled="true"
    android:exported="false" />
```

Kuva 6. Insinöörityössä olevan Content Providerin esittely manifestissa.

Content Provideria käytetään ContentResolver-luokan kautta, jolloin kutsutaan kuvan 7 metodeita. Se saadaan kutsumalla metodia Context.getContentResolver(). Parametrina kuvan 7 metodeille annetaan Content Uri, joka sisältää kaksi osaa. Authority-osa, joka on nähtävissä kuvasta 6, kertoo mitä Content Provideria ContentResolver kutsuu, sekä polun, mikä kertoo kutsuttavalle Content Providerille pyydettävän datan sijainnin. Content Urit ovat yleensä staattisia julkisia muuttujia Content Provider -toteutuksen sisällä, jotta niitä olisi helppo käyttää mistä tahansa ohjelmasta. Content Providerille voi myös antaa oikeuksia, jotka sitä käyttävän ohjelman täytyy pyytää manifestissa. [13.]

```
public abstract Cursor query(Uri, String[], String, String[], String);
public abstract Uri insert(Uri, ContentValues);
public abstract int update(Uri, ContentValues, String, String[]);
public abstract int delete(Uri, String, String[]);
```

Kuva 7. Content Provider -luokan metodit, jotka kehittäjä ylikirjoittaa.

### Broadcast Receiver

BroadcastReceiver-luokka vastaanottaa lähetyksiä, jotka on lähetetty sendBroadcast-metodilla jonkun komponentin toimesta. Niiden lähettäjä ja vastaanottaja eivät tarvitse sijaita samassa prosessissa, vaan luokka on tehty prosessien väliseen kommunikaatioon. Kuvissa 8 ja 9 on yksinkertainen BroadcastReceiver-toteutus insinöörityössä olevasta ohjelmasta. Siinä BroadcastReceiver odottaa BOOT_COMPLETED-signaalia, joka lähetetään, kun käyttäjärjestelmä on käynnistynyt. Tämän jälkeen BroadcastReceiver käynnistää Service-komponentin ja lopettaa toimintansa. [14.]

```
<uses-permission android:name="android.permission.RECEIVE_BOOT_COMPLETED"/>
. . . .
<receiver
    android:name=".OnBootReceiver"
    android:enabled="true"
    android:exported="false">
    <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED"/>
    </intent-filter>
</receiver>
```

Kuva 8. BroadcastReceiverin rekisteröinti AndroidManifest.xml-tiedostossa.

Toimiakseen Broadcast Receiver täytyy rekisteröidä, joko kutsumalla metodia Context.registerReceiver() tai määrittelemällä se manifest-tiedostossa. Manifest-tiedostossa määriteltyä Broadcast Receiveriä kutsutaan aina, kun taas dynaamisesti rekisteröityä kutsutaan vain sen aikaa, kun se on rekisteröitynä. Yleensä dynaaminen rekisteröinti tehdään Activity.onResume()-metodissa ja poistetaan Activity.onPause()-metodissa. Rekisteröinnin yhteydessä on myös määriteltävä intent filter, joka määrittelee, mitä lähetyksiä toteutus Broadcast Receiveristä ottaa vastaan. Toteutuksessa on myös otettava huomioon, että Broadcast Receiver on globaali, ja kaikki ohjelmat voivat lähettää sille viestejä. Näistä joku voi myös yrittäen väärinkäyttää sitä. Turvallisuutta voi kohentaa oikeuksilla tai manifest-attribuutilla android:exported=false, jolloin muut ohjelmat eivät voi lähettää viestejä sille. [14.]

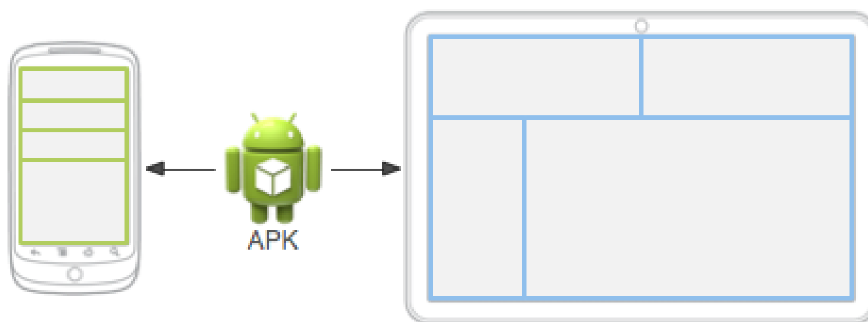
Broadcast Receiverin elämänkaari käsittää vain onReceive-metodin kutsun ja tämän loppuessa käyttöjärjestelmä näkee komponentin päättyneenä sulkien sen. Tämä rajoittaa asioita, joita kehittäjä voi tehdä metodin sisällä. Kehittäjä ei siis voi tehdä asynkronisia asioita, eli näyttää dialogeja tai sitoa Serviceen. Servicen voi kuitenkin käynnistää kuvan 9 mukaisesti ja käyttäjälle voi näyttää viestejä käyttäen Notification APIa. [14.]

```
public class OnBootReceiver extends BroadcastReceiver {

    @Override
    public void onReceive( Context context, Intent intent ) {
        Intent myIntent = new Intent(context, ServerConnectionService.class);
        context.startService(myIntent);
    }
}
```

Kuva 9. Servicen käynnistys kun Broadcast Receiver saa signaalin BOOT_COMPLETED.

Resurssitiedostot

Androidissa staattinen data on sijoitettu resurssitiedostoihin, jotka suurimmaksi osaksi ovat xml-tiedostoja. Staattinen data sisältää kuvia, tekstiä, layout-tiedostoja sekä muita samankaltaisia asioita. Resurssitiedostot tekevät helpoksi kuvan 10 tapauksen, missä koon tai orientaation mukaan näytöille ladataan erilaiset tyylipohjat. Toinen hyvä esimerkki resurssitiedostoista on Localization eli käyttäjän sijaintiin perustuva staattinen data. Kehittäjä voi tarjota ohjelmasta eri kielisiä versioita kääntämällä res/values/strings.xml-tiedoston kyseiselle kielelle ja tallentamalla se res/values-<maakoodi>/strings.xml-tiedostoon, missä <maakoodi>-kohta korvataan kyseisen maan koodilla. Esimerkiksi ranskan maakoodi on fr. Localizatiossa voi mennä myös niin pitkälle, että tarjoaa kyseiselle sijainnille omat resurssitiedostot sijoittamalla ne res/<maakoodi>/-kansioon. [15.]




Kuva 10. Ajon aikana näytön koon tai orientaation mukaan valittu layout-tiedosto. [15.]

Säikeet, Loaderit ja verkko

Android-ohjelman käynnistyessä käyttöjärjestelmä luo sille oman prosessin, jos kyseistä ei ole jo aikaisemmin luotu. Tämän jälkeen se luo ohjelmalle pääsäikeen, joka on vastuussa tapahtumien lähettämisestä, piirtämisestä näytölle sekä kaikista muista ohjelman

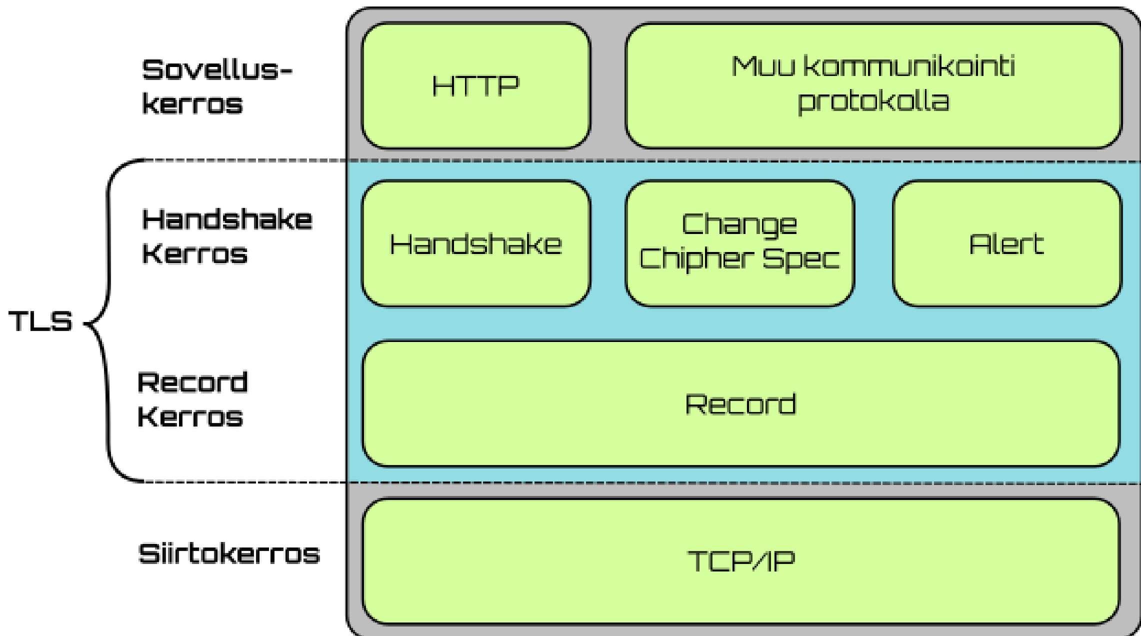
komponenttien tapahtumista. Komponentit luodaan pääsäikeessä ja ne käyttävät säiettä koko elinkaaren ajan, ellei kehittäjä luo niille omia säikeitä. Tämän takia on tärkeää, ettei pääsäikeessä ajeta mitään pitkäkestoisia toimintoja. Pääsäikeen jäädessä odottamaan jotain toimintoa eivät muut komponentit voi käyttää kyseistä säiettä. Ohjelma siis jäätyy ja viiden sekunnin kuluttua käyttöjärjestelmä näyttää jo käyttäjälle dialogin josta ohjelman voi sulkea. [16.]

Pitkäkestoisia toimintoja ovat yleensä verkkosiirräntä tai tietokantakyselyt. Tietokantakyselyitä varten on Androidissa Loaderit. Loaderit ovat kaikkien aktiviteettien ja fragmenttien käytössä. Ne tarjoavat asynkronisen tiedon latauksen taustasäikeessä, eivätkä häiritse pääsäikeen suoritusta. Tiedon latauksen jälkeen ne jäävät monitoroimaan tiedonlähdettä ja sen muuttuessa palauttavat uuden tuloksen. Verkkosiirräntää varten kehittäjän on luotava oma säie tai käytettävä AsyncTask-luokkaa. Yksinkertaisen toiminnon suorittamiseen AsyncTask on erinomainen valinta. Se tarjoaa kehittäjälle taustasäikeen sekä tuloksien suorittamisen pääsäikeessä. Pääsäiettä tarvitaan käyttöliittymä-elementtien muuttamiseen, koska ne ovat tehty säieturvalliseksi rajoittamalla niiden kutsumiset pääsäikeeseen. Kokonaan oman säikeen toteutus on taas hyvä idea, jos käyttöliittymäelementtejä ei tarvitse päivittää tai halutaan enemmän kontrollia säikeen toteutukseen. [16; 17.]

## 3.2 SSL ja TLS

SSL (Secure Socket Layer) ja TLS (Transport Layer Security) ovat salaustekniikoita, jotka tarjoavat salatun ja luotettavan yhteyden kahden kommunikoivan laitteen kanssa. Yhteyttä pitkin voidaan lähettää käyttäjälle tärkeää dataa turvallisesti ilman, että verkkorikolliset pystyvät sitä urkkimaan. Kummatkin salaustekniikat tarjoavat palvelin- ja asiakasohjelmatodennuksen, kulkevan tiedon salauksen sekä tiedon koskemattomuuden. TLS on SSL-protokollan seuraaja. TLS-versioon 1.2 asti pystyttiin salausmenetelmä laskemaan alaspäin SSL 3.0 -tasolle yhteyttä luodessa, mutta uusimmassa 1.2-versiossa tämä ei ole enää mahdollista. Siinä on pyritty poistamaan heikkoja ja vanhoja salausmenetelmiä lopettaen takaisin päin menevä tuki SSL-salaustekniikalle, jonka kolmas ja viimeinen versio esiteltiin vuonna 1996. Useimmiten puhuttaessa SSL- ja TLS-protokol-

lista, niihin viitataan sanalla SSL. SSL vanhentuneena teknologiana jätetään tämän insinöörityön ulkopuolelle ja keskitymme jatkossa ainoastaan TLS-salaustekniikkaan. Myöhemmät viittaukset SSL-protokollaan ovat yleisviittauksia SSL- ja TLS-tekniikkoihin. [18.]




Kuva 11. TLS-protokollakerrokset. [18.]

### 3.2.1 TLS Record -kerros

TLS sisältää Record- ja Handshake-kerrokset kuvan 11 mukaisesti. TLS-protokollan etuna on sen riippumattomuus sovelluskerroksesta. Kehittäjä voi siis rakentaa TLS-toteutuksen päälle kerroksia vaikuttamatta itse TLS-toteutukseen. TLS-toteutuksen alimmalta tasolta löytyvä TLS Record -protokolla tarjoaa suojatun yhteyden, jolla on seuraavat kaksi ominaisuutta. [18.]

- Yksityinen yhteys, jossa data suojataan symmetrisellä algoritmilla. Salausavaimet symmetriseen algoritmiin luodaan uudelleen jokaisen yhteyden luonnin aikana ja ne pohjustuvat toisen protokollan käymään neuvotteluun. Tällainen on esimerkiksi TLS Handshake -protokolla. Yhteyden voi myös luoda ilman suojausta.

- Luotettava data, jossa data yhtenäisyys tarkistetaan joka lähetyksen yhteydessä käyttäen HMAC (Keyed-Hashing for Message Authentication Code) -funktiota.

TLS Record -protokolla voi toimia myös ilman tätä, mutta se tapahtuu vain, kun toinen protokolla neuvottelee salausparametreja TLS Record -protokollaa käyttäen. [18.]

### 3.2.2 TLS Handshake -kerros

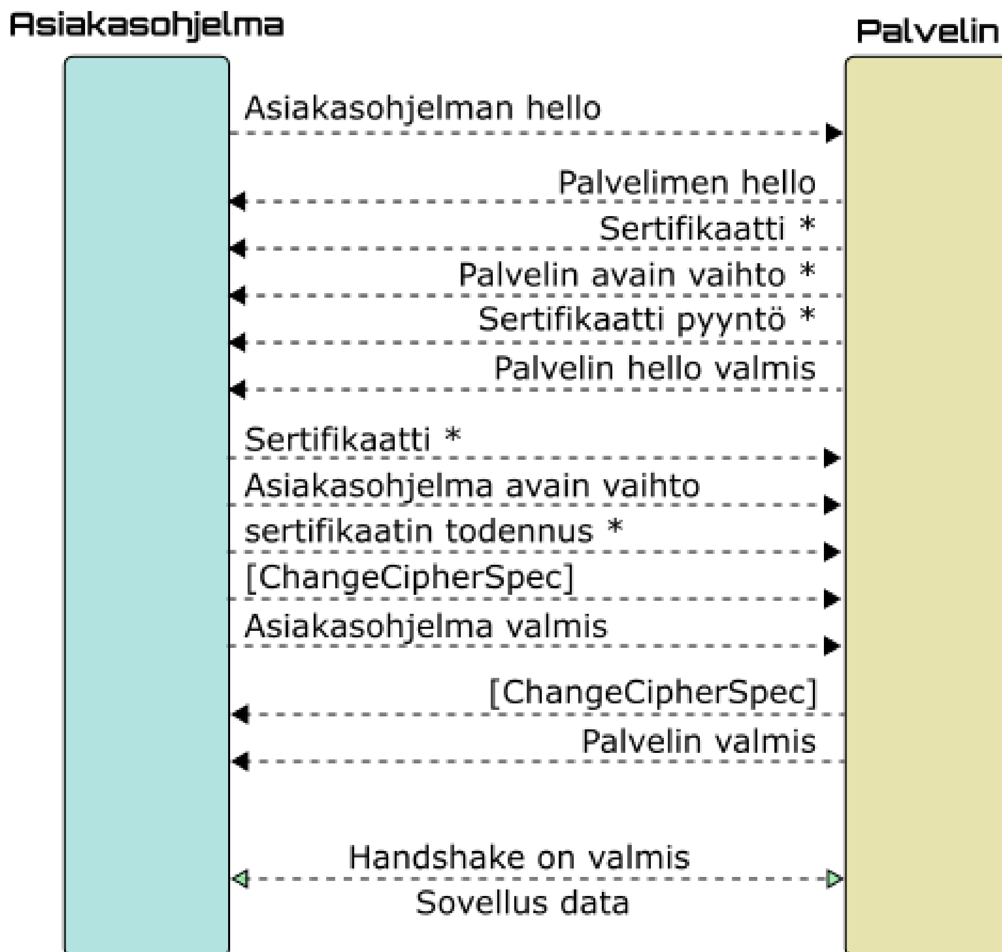
TLS Record -protokolla toimii rajapintana Handshake-tason protokolille. TLS Handshake -protokolla on yksi näistä. Se mahdollistaa palvelimen ja asiakasohjelman todentamisen, neuvottelee salausalgoritmin sekä salausavaimet ennen kuin todellista dataa ehditään lähettämään. TLS Handshake -protokolla tarjoaa suojatun yhteyden, jolla on seuraavat ominaisuudet. [18.]

- Osapuolten identiteetti voidaan todentaa käyttäen asymmetristä tai julkisen avaimen kryptografiaa. Todennus ei ole pakollinen, mutta usein ainakin palvelin todentaa itsensä.

- Jaetun salausavaimen suojattu neuvottelu. Salausavain neuvottelu hoidetaan niin, ettei sitä pystytä saamaan käsiksi, vaikka hyökkääjä pääsisi yhteyden väliin.

- Neuvottelun luotettavuus. Hyökkääjä ei pysty muokkaamaan neuvotteludataa ilman, että osapuolet huomaisivat sitä. [18.]

Toimiakseen TLS-protokolla tarvitsee luotettavan siirtokerrosprotokollan, esimerkiksi TCP (Transmission Control Protocol). TCP tarjoaa luotettavan yhteyden, jossa data kulkee järjestyksessä. Perille päässyt data tarkistetaan virheiden varalta ja virheiden sattuessa pyydetään pakettia uudestaan. Yhteyden muodostus tapahtuu aina suojaamattomana ja onkin kehittäjän tehtävä liittää TLS-protokolla siirtokerroksen päälle. Yhteyden suojaamiseksi käytetään TLS Handshake -protokollaa, joka neuvottelee osapuolten kanssa salaisen avaimen kuvan 12 mukaisesti. Kuvassa 12 kaikki tähdellä merkityt kohdat ovat tilanteesta riippuvia eikä niitä aina lähetetä. [18.]




Kuva 12. TLS Handshake -protokollan viestien kulku. [18.]

Neuvottelu alkaa ”hello”-viesteillä, jossa asiakasohjelma ja palvelin kertovat toisillensa omasta TLS-toteutuksestaan. Tämän jälkeen lähetetään sertifikaatti sekä pyydetään asiakasohjelmalta sertifikaattia. Palvelimen avaimen vaihtopyyntö tapahtuu yleensä, jos palvelimella ei ole sertifikaattia. Lopuksi lähetetään ”hello valmis” -viesti. Asiakasohjelma lähettää heti perään oman sertifikaatin, jos tätä pyydettiin sekä tarkistaa palvelinsertifikaatin oikeellisuuden. Tämän jälkeen asiakasohjelma lähettää viestin salausprotokollan vaihtumisesta TLS ChangeCipherSpec -protokolalla, jota seuraa ”valmis”-viesti, mikä on salattu neuvotellulla algoritmilla ja salausavaimilla. Palvelin vastaa tähän samoilla viesteillä, ja kun palvelin on lähettänyt ”valmis”-viestin, on salattu yhteys muodostettu ja dataa voi lähettää. [18.]

### 3.2.3 Asymmetrinen ja symmetrinen algoritmi

Asymmetrinen algoritmi käyttää julkista ja yksityistä avainta salauksen toteuttamiseen. Julkisen avaimen voi nimensä mukaan jakaa muille ilman haittavaikutuksia. Yleensä julkinen avain lähetetään sertifikaatin yhteydessä. Avaimen saaja voi sitten lähettää viestin, joka on salattu kyseisellä julkisella avaimella, jonka vain avaimen omistaja pystyy avaamaan yksityisellä avaimellaan. Salaus toimii myös toiseen suuntaan, eli palvelin voi lähettää yksityisellä avaimella salatun viestin asiakasohjelmalle, joka avaa sen palvelimelta saadulla julkisella avaimella. Asiakasohjelma pystyy olemaan varma, että viesti tuli palvelimelta, koska vain palvelin tietää kyseisen yksityisen avaimen. Asymmetriset algoritmit ovat kuitenkin reilusti hitaampia kuin symmetriset, minkä takia niitä käytetään yleensä vain istuntoavainten luontiin. [19, s. 2, 33.]

Symmetrinen algoritmi käyttää vain yhtä avainta tiedon salaukseen ja sen purkuun. Kyseistä avainta ei siis voi lähettää verkon välityksellä ilman salausta. Avain yleensä syntyy TLS-yhteyden neuvottelun tuloksena, joka on suojattu asymmetrisellä algoritmilla. [19, s. 11.]

### 3.2.4 Sertifikaatti

Sertifikaatti on identiteettitiedosto julkisen avaimen turvalliseen jakamiseen. Se sisältää julkisen avaimen lisäksi tiedot hakijasta ja myöntäjästä, voimassaoloajan, käytetyn kryptografisen algoritmin sekä paljon muuta. Yleensä sertifikaatit ovat sertifikaattitoimijan myöntämiä, mutta ne voivat olla myös itse allekirjoitettuja. Sertifikaattitoimijat ovat tahoja, joihin luotamme. He tekevät itse allekirjoitetun sertifikaatin, jonka he julkistavat kaikkien saataville. Tämän jälkeen he allekirjoittavat hakijan sertifikaatin. Allekirjoitus tapahtuu ottamalla hash-arvo sertifikaatista ja salaamalla se yksityisellä avaimella, joka on toimijan sertifikaatin julkisen avaimen pari. Näin ollen on helppo tarkistaa saatu sertifikaatti purkamalla allekirjoitus käyttäen toimijan sertifikaatin julkista avainta sekä kryptografista algoritmia ja verrata hash-arvoa sertifikaatin hash-arvoon. Sertifikaattitiedosto sisältää myös sormenjäljen, joka on sertifikaatin hash-arvo. Tällä voidaan helposti tarkistaa sertifikaattitiedoston aitous. Edellä mainittujen asioiden avulla ehkäistään hyökkäykset, jossa kolmas taho tulee yhteyden väliin seuraamaan yhteyttä. Kummankin osapuolen saadessa toistensa julkiset avaimet ei kolmas taho pysty enää näkemään niillä salattuja viestejä. [19, s. 62, 89; 20.]

## 3.3 JDBC

JDBC eli Java Database Connectivity API on rajapinta, joka mahdollistaa yhteydenoton tietokantaan tai mihin tahansa taulukolliseen datan lähteeseen. Se tarjoaa CLI API:n tietokannan kanssa kommunikointiin SQL-kielellä. CLI (Call Level Interface) on ohjelmistostandardi, joka määrittelee, kuinka ohjelman tulisi lähettää SQL-viestejä tietokantaan ja miten sieltä tuleva tulosjoukko tulisi hallita. JDBC sisältää kaksi pääohjelmistorajapintaa. Ylemmässä kerroksessa toimivan ohjelman kehittäjille tarkoitetun JDBC API:n sekä alemman tason tietokanta-ajureiden tekoon tarkoitetun JDBC driver API:n. Ajurit on yleensä tehty tietokannan kehittäjän puolesta, eikä ohjelman kehittäjän tarvitse huolehtia kuin SQL-kyselyiden tekemisestä. [21; 22.]

## 3.4 JCA

JCA (Java Cryptography Architecture) on Java Standard Editionin mukana tuleva sovelluskehys kryptografisten palveluiden kehittämiseen. Nämä palvelut toteutetaan Provider-arkkitehtuurin mukaisesti. Tämän lisäksi mukana tulee valmiita Provider-toteutuksia. JCA sisältää Provider-arkkitehtuurin lisäksi sertifikaatin, salauksen, avaimen luonnin sekä muita kryptograafisia rajapintoja. Provider-arkkitehtuuri on suunniteltu seuraavia periaatteita ajatellen. [23.]

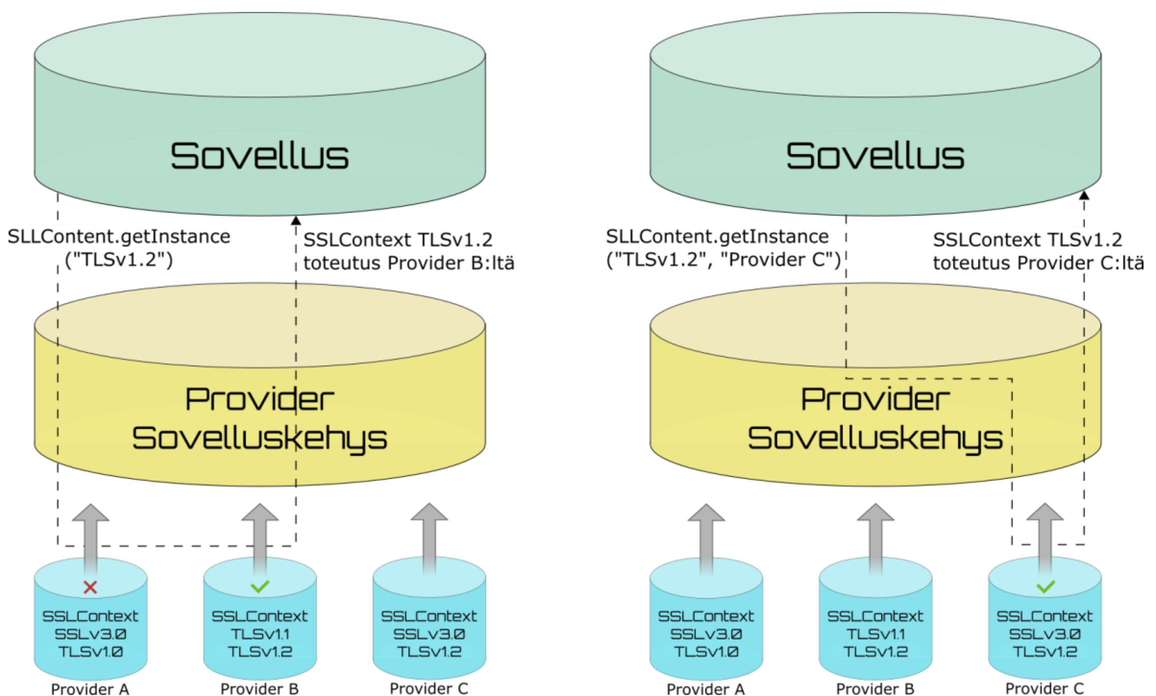
- Itsenäinen toteutus – Sovellukset eivät itse toteuta kryptografisia algoritmeja vaan pyytävät palveluita Java-järjestelmältä.

- Toteutusten yhteistoimivuus – Sovellus ei ole sidottu tiettyyn Provider-palveluun, eikä palvelu sovellukseen.

- Algoritmien laajennus – Kehittäjä voi luoda oman Provider-toteutuksen, joka toteuttaa kryptografisia palveluita ja asentaa sen Java-alustalle. [23.]

Providers

Provider eli kryptografisten palveluiden tarjoaja on komponentti, joka sisältää joukon engine-luokille tehtyjä algoritmeja. Provider ei näy ollenkaan kehittäjälle, joka käyttää sen palveluita. Kuvassa 13 on kaksi käyttötapausta, joissa kummassakin kehittäjä pyytää Providerilta "TLSv1.2"-toteutusta SSLContext-luokasta kutsulla SLLContext.getInstance("TLSv1.2"). Vasemmanpuolisessa ei ole annettu parametrina Provider-toteutuksen nimeä, joten Provider-sovelluskehys etsii ensimmäisen "TLSv1.2"-toteuttavan Provider-komponentin omasta tietokannastaan ja palauttaa SSLContext-olion tällä toteutuksella. Oikeanpuolisessa tapauksessa etsitään Provider-komponentti suoraan nimen perusteella ja haetaan kyseinen toteutus. Provider-arkkitehtuuri ei siis sido ohjelmaa toteutukseen vaan joka kerralla ajon aikana se etsii jonkun Providerin, joka tarjoaa toteutuksen. [23.]




Kuva 13. Palvelun haku Provider-sovelluskehyksessä. [23.]

## 3.5 JSSE

JSSE (Java Secure Socket Extension) on Java Standard Editionin mukana tuleva turvallisuuskomponentti, jonka tarjoama rajapinta mahdollistaa suojatun viestinnän. JSSE pohjautuu samoihin suunnitteluperiaatteisiin kuin JCA-sovelluskehys. Alkujaan JSSE oli

Javan valinnainen lisäosa, mutta nykyään se tulee Java Standard Editionin mukana. JSSE tarjoaa sovelluskehyksen sekä toteutuksen SSL- sekä TLS-protokollista, jotka sisältävät tiedon salauksen, molemminpuolisen todentamisen ja viestin eheyden tarkistuksen. Tämänhetkinen uusin Java-versio 8 tukee SSL 3 sekä TLS 1.0-1.2 -versioita, joista TLS 1.2 -versiota käytetään vakiona. JSSE sisältää kolme luokkaa suojatun yhteyden toteuttamiseen. Ne ovat SSLSocket, SSLServerSocket sekä SSLEngine. [24.]

### 3.5.1 SSLContext

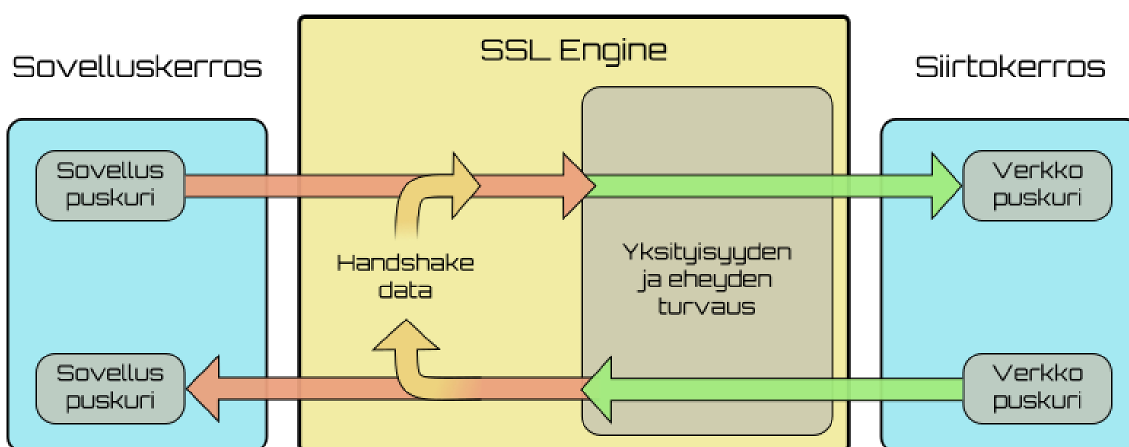
SSLContext on JCA engine -luokka suojatun socket-protokollan toteutukselle, ja se toimii tehdasluokkana SSL socket -tehtaille sekä SSLEnginelle. Se pitää sisällään kaikkien sen luomien olioiden jaetun tilatiedon. SSLContext luodaan kuvan 13 mukaisesti, jonka jälkeen se alustetaan init-metodissa key- ja truststore-komponenteilla. Keystore pitää sisällään omat avainparit ja sertifikaatit, joita käytetään handshake-prosessin aikana. Saapuvat sertifikaatit tarkistetaan käyttäen truststore-ilmentymää, ja vertaamalla sertifikaatteja toisiinsa. Alustamisen jälkeen SSLContext on valmis luomaan tehtaita sekä SSLEngineitä. [24.]

### 3.5.2 SSLSocket ja SSLServerSocket

SSLSocket- ja SSLServerSocket-luokat ovat Socket- ja ServerSocket-luokkien aliluokat. Ne tarjoavat kehittäjälle nopean ja helpon tavan suojatun yhteyden toteuttamiseen. SSLSocket-luokka luodaan joko käyttäen SSLSocketFactory-luokan createSocket-metodia tai SSLServerSocket-luokan accept-metodia. Kummatkin tavat palauttavat käyttövalmiin SSLSocket-olion, joka on yhdistetty. Handshake voidaan aloittaa käyttäjän toimesta, mutta myös datan lukeminen ja kirjoittaminen aloittavat sen. SSLSocket perustuu virtapohjaiseen siirräntämenetelmään ja on näin luonteeltaan blokkaava. Säie odottaa, että virtaan tulee luettavaa dataa tai viestin lähetystä kokonaisuudessaan. Tämän takia SSLSocket ei ole hyvä etenkään palvelinympäristössä, jossa tarvitaan skaalautuvuutta. Skaalautuvuus saadaan kuitenkin toteutettua SSLEngine-luokalla. [24; 25.]

### 3.5.3 SSLEngine

SSL-yhteyksiä tarvitaan nykypäivänä moniin asioihin, ja yleensä tämä halutaan toteuttaa skaalautuvasti ja suorituskykyistä mallia käyttäen. SSLEngine on Javan vastaus tähän. Sovellus- ja siirtokerros ovat ulkona sen toteutuksesta ja jääkin kehittäjän valittavaksi, mitä säiemallia ja siirrantämenetelmää käyttää (kuva 14). [24.]




Kuva 14. Tiedonkulku SSLEnginen läpi. [24.]

SSLEnginessä tieto kulkee sen läpi puskureita pitkin kuvan 14 mukaisesti. Salaus ja sen purkaminen tapahtuvat wrap- ja unwrap-metodeilla, jotka ottavat parametreikseen sekä sovellus- että siirtokerroksen puskurin. Nämä puskurit toimivat pareina. Kummassakin kerroksessa on sisään ja ulos menevät puskurit, ja vain sisään menevät puskurit vaihtavat dataa toistensa kanssa unwrap-metodilla. Sama logiikka toimii ulospäin menevien kanssa wrap-metodilla. Kummatkin sovelluspuskurit sisältävät salaamatonta dataa, jonka ne saavat sovellukselta tai SSLEngineltä. Verkkopuskurit sisältävät salattua dataa. Tämä data voi olla asiakasohjelman ja palvelimen keskustelua tai handshake-dataa. Sen ollessa viimeistä ei se koskaan tule sovelluskerrokseen asti vaan SSLEngine tuottaa ja lukee sitä. Toisin kuin SSLSocket-luokan kanssa kehittäjä joutuu itse toteuttamaan handshake-prosessin. Tätä varten SSLEnginestä löytyy tilamuuttujat omalle sekä handshake-tilalle. Tilat ohjaavat handshake-prosessin läpi SSLEnginen hoitaessa kommunikaation. Kehittäjän huoleksi jää datan siirto siirtokerroksen ja SSLEnginen välillä sekä SLLEngine-tilan vaatimien tehtävien tekeminen. Edellä mainittujen kahden tilan lisäksi SSLEngine on elämänsä aikana seuraavissa vaiheissa. [24; 25.]

- Luonti – SSLEngine on juuri luotu, mutta sitä ei ole vielä käytetty. Asetuksia voi muuttaa tässä tilassa.

- Ensimmäinen handshake – Handshake-prosessi päällä. Sovellusdataa ei voi vielä lähettää eikä asetuksia muuttaa. Tehdyt muutokset tulevat voimaan, jos handshake tehdään uudelleen.

- Sovellusdata – SSLEngine on valmis sovellusdatan lähettämiseen ja vastaanottamiseen.

- Uusi handshake – Molemmat osapuolet voivat pyytää uutta handshake-prosessia. Asetuksia voi vaihtaa ennen sen aloittamista.

- Sulkeminen – Yhteyttä ei enää tarvita. SSLEngine tulee sulkea. [25.]

## 3.6 SQL

SQL (Structured Query Language) on ohjelmointikieli, joka on tarkoitettu relaatiotietokantajärjestelmien datan hallintaan. SQL sisältää datan määrittely-, manipulointi- ja kontrollointikielet. SQL-määrittelykieli sisältää komennot CREATE, DROP, ALTER ja muita määrittelyyn liittyviä komentoja. Manipulointikomennot ovat UPDATE, INSERT, DELETE ja SELECT. Viimeseksi mainittu kontrollointi osuus SQL-kielestä sisältää komentoja kuten GRANT ja REVOKE. Kieli itsessään pohjautuu relaatioalgebraan. [26.]

### 3.6.1 SQLite

SQLite on avoimen lähdekoodin relaatiotietokantatoteutus sulautetuille järjestelmille kaikilla SQL-kielen ominaisuuksilla varustettuna. Se ei tarvitse toimiakseen erillistä palvelinta vaan toimii kirjastona kehittäjän ohjelman sisällä ilman edes tarvetta sen asetuksien säätöön. Tietokanta luodaan laitteen massamuistiin, ja sen koko toteutus kirjoitetaan yhden tiedoston sisään. Tietokantatiedosto on myös siirrettävissä laitteelta toiselle, eikä ole riippuvainen laitteen arkkitehtuurista. Se, mikä tekee SQLitestä erityisen hyvän sulautetuille järjestelmille, on kirjaston pieni koko, joka voi olla kaikkien ominaisuuksien ollessa päällä alle 500 KiB.  Se voidaan myös ajaa niin, että sen ajonaikainen muistin kulutus on erittäin pieni. Kirjasto on myös hyvin testattu ja näin ollen erittäin luotettava. Transaktiot

siinä ovat ACID (Atomicity, Consistency, Isolation, Durability), ja se selviytyy siirräntävirheistä kunnialla. [27.]

ACID-termi määrittelee tietokannan transaktion seuraavat ominaisuudet.

- Atomicity – Transaktio menee tietokantaan kokonaisena tai ei ollenkaan.
- Consistency – Transaktio vie tietokannan yhtenäisestä tilasta toiseen.
- Isolation – Transaktioiden rinnakkaissuoritus päätyy samaan tilaan kuin jos transaktiot olisi ajettu peräkkäin.
- Durability – Takaa tietokantaan lisätyn transaktion pysyvyyden kaikissa tilanteissa. Jopa virran katketessa tai ohjelman kaatuessa. [28, s. 9-15.]

### 3.6.2 PostgreSQL

PostgreSQL on avoimen lähdekoodin relaatiotietokantajärjestelmä. Sitä on kehitetty yli 15 vuoden ajan, ja se on saanut hyvän maineen luotettavuudessa sekä datan yhtenäisyydessä. Nykyään PostgreSQL ohjelma on saatavissa kaikille tärkeimmille käyttöjärjestelmille. PostgreSQL:n toteutus seuraa uskollisesti ISO/IEC 9075:2008 -standardia, ja se sisältää suurimman osan siinä määritellyistä asioista. Se tarjoaakin monipuoliset haut, tuen vieraille avaimille, triggerit, muutettavat näkymät, ja sen transaktiot noudattavat ACID- määritystä. Tämän lisäksi kehittäjä voi itse lisätä ja muuttaa PostgreSQL-toteutusta. Se on lisensoitu PostgreSQL-lisenssin alla ja antaa kehittäjälle oikeudet muuttaa ja käyttää koodia yksityiseen sekä kaupalliseen tuotantoon eikä anna rajoitteita tuotetun koodin lähdekoodin saatavuudesta. Se myös sisältää ohjelmointirajapinnan suosituimpiin ohjelmointikieliin. [29; 30.]

## 3.7 Apache Commons DBCP 2

Apache Commons DBCP 2 (Database Connection Pool) on Java 7:lle tehty uudelleen käytettävien JDBC-tietokantayhteyksien joukkoa hallitseva komponentti. Uuden yhteyden luonti tietokantaan on aikaa vievä prosessi, joka kestää yleensä pidempään kuin tietokantakysely. Yhteyttä ei siis kannata sulkea, vaan se kannattaa uusiokäyttää. DBCP

ylläpitää JDBC-tietokantayhteysjoukkoa, ja se poistaa kuolleet tai jumittuneet yhteydet ja lisää yhteyksiä tarpeen vaatiessa kehittäjän määrittelemien asetusten mukaan. [31.]

```
public class Database {

    public static final Database instance = new Database();
    private final BasicDataSource mBasicDataSource;

    private Database(){
        mBasicDataSource = new BasicDataSource();
        mBasicDataSource.setUsername( "username" );
        mBasicDataSource.setPassword( "password" );
        mBasicDataSource.setUrl( "jdbc:postgresql://localhost:5432/FriendFinder" );
        mBasicDataSource.setDriverClassName( "org.postgresql.Driver" );
        mBasicDataSource.setInitialSize( 8 );
        mBasicDataSource.setMaxTotal( 8 );

    public Connection getConnection() throws SQLException{
        return mBasicDataSource.getConnection();
    }
}
```

Kuva 15. Apache Commons DBCP -toteutus yksinkertaisimmillaan.

Kuvassa 15 on karsittu versio insinöörityössä olevasta toteutuksesta, jossa Apache Commons DBCP otetaan käyttöön. Singleton-suunnittelumallilla tehty luokka luo ohjelman käynnistyksen aikana BasicDataSource-luokan, joka hallitsee tietokantayhteysjoukkoa. Yhteyden saa kyseiseltä luokalta hankittua kutsumalla Database.instance.getConnection()-metodia. Käytön jälkeen yhteyden hallitsija sulkee yhteyden, mikä palauttaa sen takaisin tietokantayhteysjoukkoon. Yhteyden muodostusta varten BasicDataSource-luokka tarvitsee käyttäjätiedot tietokantaan, tietokannan osoitteen sekä tietokanta-ajurin nimen.

# 4 Toteutus

Insinöörityön toteutus alkoi Java-palvelin toteutuksiin tutustuen. Java-ohjelmointikielen valinta palvelin toteutukseksi perustui omaan mielenkiintoon Javaa kohtaan sekä sen käyttöön Android-ohjelmointikielenä. Palvelimen ominaisuuksiksi määriteltiin luvussa 2 skaalautuvuus ja tietoturvallisuus. Nämä ongelmat oli ensin ratkaistava. Tämän jälkeen tarvittiin tietokantatoteutus palvelimen datan säilyttämiseen. Siinä oli kaksi varteen otettavaa vaihtoehtoa MySQL sekä PostgreSQL, mutta oman mielenkiinnon takia valittiin PostgreSQL-tietokantaratkaisuksi. Android-ohjelman rakenne piti ratkaista seuraavaksi sekä tietokantoihin tallennettavan datan muoto. Itse Android oli tuttu alusta eikä näin

vaatinut opettelua. Teknologioiden valinnan ja niiden haltuunoton jälkeen olikin aika kehitysympäristön pystyttämisen. Siinä syntyi kolme projektia, jotka olivat palvelin, asiakasohjelma sekä niiden väliseen keskusteluun käytetty protokolla. Kommunikointiprotokolla on kirjastoprojekti, joka sisältyi sekä palvelin että asiakasohjelmaprojektiin. Projektienhallintaan käytettiin Gradle-rakennustyökalua ja versionhallinnassa työkaluna oli Git. Itse ohjelmointi suoritettiin IntelliJ:llä sekä Android-studiolla. Android-studio on IntelliJ:n päälle kehitetty Android-kehitysympäristö.

## 4.1 Java-palvelinohjelma

Määrittelyn mukaan palvelimen täytyi olla skaalautuva sekä tietoturvallinen. Tietoturvallisuuden saamiseksi tarvitaan suojattu yhteys. Tähän Java tarjosi meille kaksi tapaa, jotka kävimme läpi luvussa 3.5. Näistä vain SSLEnginen toteutus tarjosi skaalautuvuutta palvelimeen. Se ei kuitenkaan sitonut meitä tiettyihin säiemalleihin tai siirtokerroksen toteutuksiin. Seuraavassa käymme läpi ensin säiemallin ja siirtokerroksen toteutuksen, jonka jälkeen katsotaan SSLEngine-toteutusta sekä salausta.

### 4.1.1 Säiemalli ja siirtokerros

Ensimmäisenä aloitamme valitsemalla siirtokerroksen toteutuksen, jonka jälkeen on helpompaa miettiä säiemallin toteutusta. Sitä valittaessa vaihtoehtoina olivat Multiplexed sekä asynkroninen siirräntä, jotka kummatkin ovat skaalautuvia. Näistä Multiplexed toimi nopeammin Linux-pohjaisissa käyttöjärjestelmissä, kun taas asynkroninen siirräntä toimii nopeammin Windows käyttöjärjestelmissä. Multiplexed valittiin oletuksen pohjalta, että palvelinohjelmisto tulee pyörimään Linux-käyttöjärjestelmässä.

Multiplex-siirräntämenetelmä ei rajoita säiemallin valintaa. Se tosin pakoittaa meidät työnjakaja/työntekijä suunnittelumalliin. Multiplex-toteutus perustuu rekisteröityjen yhteyksien tarkkailuun. Javassa tämä on toteutettu Selector-luokalla, joka ei kuitenkaan itse tarkkaile yhteyksiä, vaan käyttöjärjestelmä tekee sen tämän puolesta. Selector pyytää käyttöjärjestelmältä tätä palvelua metodilla select jääden odottamaan, että ainakin yksi yhteys palautetaan. Se myös sisältää ei-blokkaavan toteutuksen selectNow. Selectorin saadessa listan valmiista yhteyksistä aloittaa säie työn jakamisen. Töiden palveleminen voidaan hoitaa samassa säikeessä tai antaa toisen säikeen hoidettavaksi. Selectorille voi myös antaa yhteyksien hyväksymisen, joka toimii samalla tavalla kuin itse

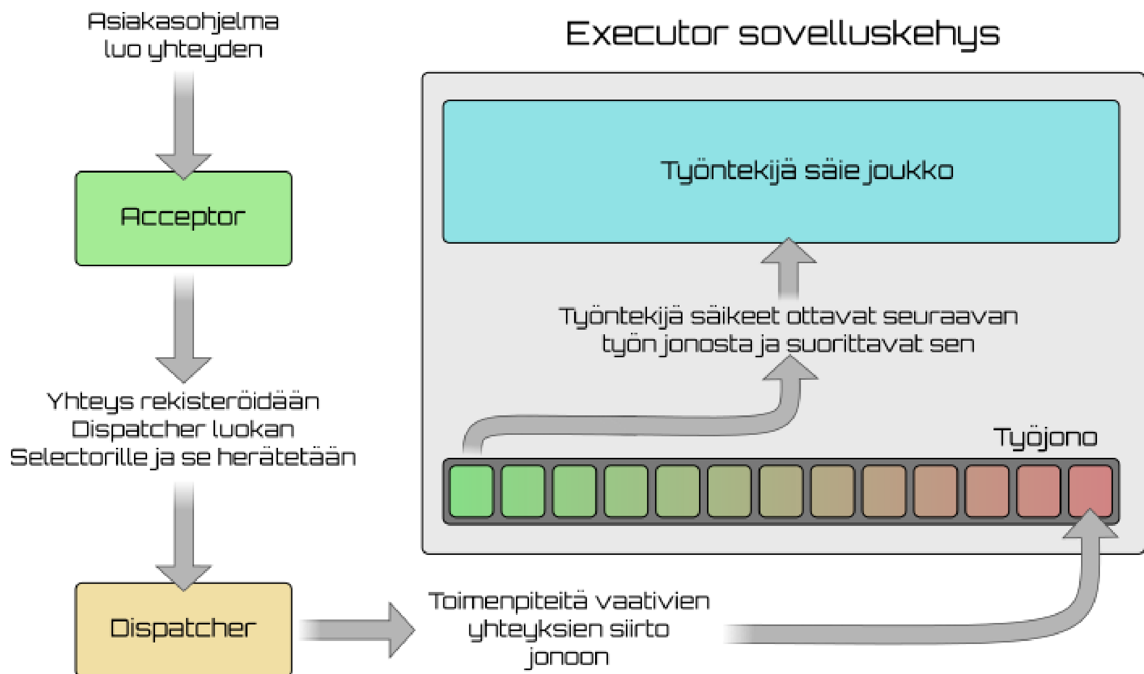
siirräntä. Multiplex siis tarvitsee yhden säikeen Selectorille, mutta se ei muuten ota kantaa säiemalliin.

```
public Dispatcher() throws IOException {
    mSelector = Selector.open();

    int threads = getWorkerCount();

    mThreadPool = new ThreadPoolExecutor(threads, threads,
        0L, TimeUnit.MILLISECONDS,
        new LinkedBlockingQueue<>(QUEUE_CAPACITY),
        new ThreadPoolExecutor.CallerRunsPolicy());
}

public void run() {
    for (;;) {
        try {
            mSelector.select();
            for ( Iterator i = mSelector.selectedKeys().iterator(); i.hasNext(); ) {
                SelectionKey selectionKey = (SelectionKey)i.next();
                i.remove();

                // Remove interestOps so that multiple threads don't try to handle same client
                selectionKey.interestOps(selectionKey.interestOps() & ~selectionKey.readyOps());

                // if queue is full task will execute in this thread
                mThreadPool.execute( (Runnable)selectionKey.attachment() );
            }

            // Gives time for SelectableChannel.register to acquire selector's inner lock.
            synchronized (mLock) { }
        } catch (IOException x) {
            x.printStackTrace();
        }
    }
}

public void register( SelectableChannel selectableChannel, int ops, SessionHandler handler) throws IOException {
    synchronized (mLock) {
        // Important! This is needed for selector to release the lock that register needs.
        mSelector.wakeup();

        // register synchronizes with same object that Selector.select does and provides visibility for Dispatcher
        SelectionKey selectionKey = selectableChannel.register(mSelector, ops, handler);
        handler.setSelectionKey( selectionKey );
    }
}
```

Kuva 16. Otos työnjakajasäikeestä.

Kuvassa 16 on otos insinöörityössä olevasta työnjakaja eli Dispatcher-säieluokasta, jossa Multiplex-siirräntä on toteutettu. Siinä yhteydet rekisteröidään ensin Selectorille toisesta säikeestä käsin, samalla määritellen yhteyden tarkkailtavat operaatiot. Tarkkailtavat operaatiot voivat olla lähetys, vastaanotto, yhdistäminen tai yhteyden hyväksyminen. Insinöörityössä tehdyssä ohjelmassa tarkkaillaan vain lähetystä ja vastaanottoa. Rekisteröinti luo valinta-avaimen, joka toimii tositteena rekisteröinnistä. Valinta-avain sisältää myös SessionHandler-olion, joka sille annetaan rekisteröitäessä. Kyseinen olio palvelee yhteyttä ja sisältää kaikki yhteyden tiedot. Dispatcher-säie ei näe rekisteröityjä muutoksia sen odottaessa select-metodikutsussa. Tämän takia rekisteröinti sisältää säikeen herätyksen. Itse säie ei sisällä muuta kuin yhteyksien tarkkailun ja niiden työn jakamisen Executor-sovelluskehykselle. Ennen yhteyden siirtoa jonoon poistetaan valinta-avaimesta operaatiot, joita tarkkailtiin, koska työsäie tulee hoitamaan kyseisen operaation.

Työsäikeet on toteutettu käyttäen Executor-sovelluskehystä (kuva 17). Executor-rajapinnan toteuttava luokka määrittelee säiemallin sekä jonon, jossa työt odottavat. Tämä jono on BlockingQueue-rajanpinnan toteuttava luokka. Kyseinen jono tarjoaa näkyvyyden Dispatcher-säikeen ja työsäikeiden välillä. Työsäie näkee siis työjonossa olevan olion siinä tilassa kuin Dispatcher sen jätti siihen. Executor ajaa sille annetut Runnable-rajapinnan toteuttavat oliot omissa säikeissään. Java sisältää monia eri Executor-toteutuksia. Palvelimen Executor-toteutukseksi valittiin ThreadPoolExecutor, jonka määrittely on nähtävissä kuvassa 16. Se tarjoaa rajatun määrän säikeitä, jotka yhtäaikaisesti tyhjentävät työjonoa. Työjonoksi valittiin LinkedBlockingQueue sen paremman rinnakkaissuorituskyvyn takia. Sille myös annettiin suurin mahdollinen koko, ettei se vain jatkaisi täyttymistään palvelimen kuormittuessa. Viimeiseksi kuvan 16 mukaisesti sille määriteltiin, mitä tehdä työjonon täyttyessä. Tähän valittiin, että Dispatcher palvelee kyseisen yhteyden laittaessaan työtä jonoon sen ollessa täynnä. Tämä antaa muille säikeille aikaa työjonon tyhjennykseen ilman, että Dispatcher jatkaa töiden lisäämistä.




Kuva 17. Yhteyksien rekisteröinti sekä niiden prosessointi.

Palvelinohjelman käyttäessä monia säikeitä elämänsä aikana, on näkyvyys otettava huomioon ohjelmaa tehdessä. Edellisessä luvussa raotettiin hieman säikeiden välistä näkyvyyttä. Javan muistimalli määrittelee, miten säikeen A tekemä muutos olioon näkyy säikeessä B. Ei ole itsestään selvää, että säie B näkee säikeen A tekemän muutoksen olioon. Huonoimmassa tapauksessa säie B voi nähdä vain osan A:n tekemistä muutoksista. Helpoin tapa näkyvyyden saantiin on lukkojen käyttäminen. Säikeen B hankkiessa lukon X, jonka A vapautti enne tätä takaa sen, että kaikki A:n näkemät arvot lukon vapauttaessa näkyvät säikeelle B sen hankkiessa lukon. Volatile muuttujan kirjoitus säikeestä A ja sitä seuraava luku säikeestä B takaavat myös saman edellä mainitun näkyvyyden. Java-muistimalli on iso kokonaisuutensa. Näkyvyyden käsittelyn tarkoitus on auttaa lukiaa ymmärtämään palvelinohjelman sekä 4.2 luvussa olevan asiakasohjelman suhteen tehtyjä ratkaisuja.

```
public void run() {
    for (;;) {
        try {
            SocketChannel socketChannel = mServerSocketChannel.accept();
            SecureChannel secureChannel = new SecureChannel( socketChannel, mSSLContext );

            SessionHandler sessionHandler = new SessionHandler(secureChannel);

            // Add Dispatcher's selector to secure channel so that Dispatcher
            // can be woken up when selectionkey interestops are changed
            sessionHandler.setSelector( mDispatcher );

            mDispatcher.register(socketChannel, SelectionKey.OP_READ, sessionHandler);

        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Kuva 18. Acceptor-säikeen toteutus.

Aikaisemmin kävimme läpi työnjakaja- sekä työntekijäsäikeet. Tämän lisäksi palvelin käsittää Acceptorin eli yhteyden luontisäikeen, joka hyväksyy uudet yhteydet ja rekisteröi ne Selectorille (kuva 17 ja 18). Acceptor-luokan toteutus on nähtävissä kuvasta 18. Siinä yhteys muodostetaan ServerSocketChannel-luokalla, eikä se ole salattu vielä tässä vaiheessa. Salaus ja Handshake tapahtuvat SecureChannel-luokan sisällä, joka kapsuloi yhteyden eli SocketChannelin. SessionHandler, josta puhuimme yhteyden rekisteröinnin yhteydessä, luodaan yhteyden luonnin jälkeen, ja jokainen yhteys sisältää oman SessionHandler-olionsa. Tämä luokka vastaa kommunikaatiosta asiakasohjelman kanssa.

Toteutuksen säiemalli sisältää Acceptor- ja Dispatcher-säikeet sekä työsäikeet. Palvelinta on ajettu 4 ytimisellä prosessorilla, jossa on 8 säiettä, jolloin työsäikeiden määrä on säädetty kuuteen. Optimaalisen säiemäärän sekä ThreadPoolExecutor-luokan optimaalisten asetuksien saaminen vaatii palvelimen testausta sen maksimaalisella kävijämäärällä. Tämä voi kasvattaa palvelimen yhtäaikaisten kävijöiden maksimaalista määrää, mutta nämä arvot ovat kone kohtaiset eivätkä vaikuta skaalautuvuuteen, joten kyseinen asia jätettiin tämän insinöörityön ulkopuolelle.

### 4.1.2 Tietokanta

Palvelimenohjelman tietokannaksi valittiin PostgreSQL, jota insinöörityössä ajettiin sen kanssa samalla palvelimella. Yhteys siihen luodaan Javan JDBC-rajapinnalla ja niiden uusiokäyttö sekä hallinta on toteutettu Apache Commons DBCP -komponentilla. Ohjelman DBCP-toteutus esiteltiin suppeasti jo luvussa 3.7. Itse toteutus sisältää kuvan 15 lisäksi kyseisten arvojen haun konfiguraatiotiedostosta. Yhteyksien määräksi DBCP:ssä valittiin seitsemän, joka käsittää työsäikeet sekä Dispatcher-säikeen. Dispatcher-säie tarvitsee tietokantayhteyttä työsäiejonon ollessa täynnä, jolloin se suorittaa itse työt niitä tarjotessaan jonolle.

### 4.1.3 Salaus ja yhteyden suojaus

Palvelin tehtiin tietoturvalliseksi salaamalla yhteys sekä tallentamalla käyttäjän salasanasta vain siitä johdettu avain tietokantaan. Kuvassa 19 on toteutus salasanan muuttamisesta johdetuksi avaimeksi käyttäen PBKDF2 (Password-Based Key Derivation Function 2) funktiota. Johdettu avain muodostuu siinä käyttäen HMAC:ia (keyed-hash message authentication code), jonka hash-funktiona toimii SHA256 (Secure Hash Algorithm 2). Menemättä liikaa teknisiin yksityiskohtiin, PFKDF2 ajaa HMAC-funktion kehittäjän määrittelemän iteraatiomäärän verran antaen sille avaimeksi salasanan sekä viestiksi suolan ensimmäisellä iteraatiolla. Seuraavilla iteraatioilla sille annetaan salasana sekä edellisen hash-arvon. Tuloksena syntyy johdettu avain, joka on kohtalaisella iteraatio määrällä ja tarpeeksi isolla suola-arvolla turvallista tallentaa tietokantaa. Oletus on että tietokannasta salasanat tulevat aina vuotamaan, joten ne tulee suojata. Suola-arvolla tarkoitetaan n-bittistä satunnaislukuarvoa.

```
private static final int PASSWORD_BIT_LENGTH = 256;

public static byte[] hash( String password, int iterations, byte[] salt ) {

    PBEKeySpec spec = new PBEKeySpec( password.toCharArray(), salt, iterations, PASSWORD_BIT_LENGTH );
    try {
        SecretKeyFactory skf = SecretKeyFactory.getInstance( "PBKDF2WithHmacSHA256" );
        return skf.generateSecret( spec ).getEncoded();
    } catch( NoSuchAlgorithmException | InvalidKeySpecException e ) {
        e.printStackTrace();
    } finally {
        spec.clearPassword();
    }

    return null;
}
```

Kuva 19. Salasanan suojaus.

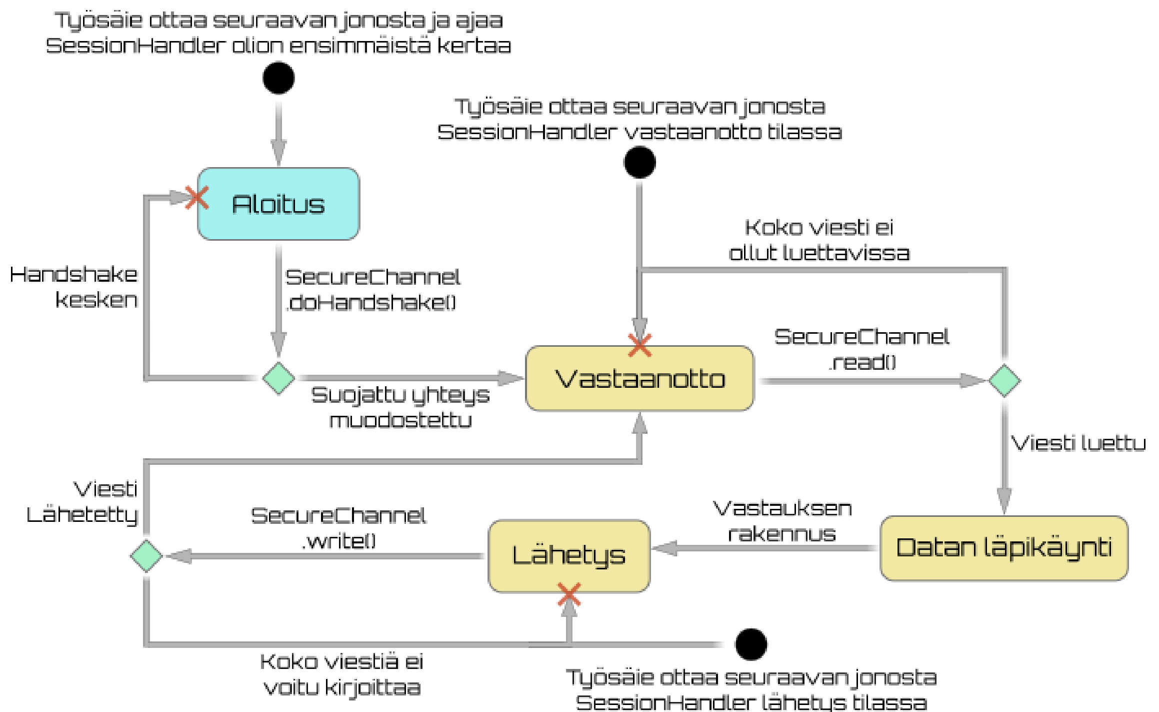
Yhteyden suojaus tehtiin SSLEngine-luokkaa käyttäen, jonka peruskäsitteet käytiin läpi luvussa 3.5.3. SSLEngine-luokka kapsulointiin SecureChannel-luokan sisään SocketChannel-luokan kanssa. Kuvassa 20 on lista SecureChannel-luokan tärkeimmistä metodeista ja muuttujista. Siinä on määriteltynä sisään ja ulospäin menevät puskurit, joita käytetään read, write ja doHandshake metodien sisällä. Sovelluspuolen puskuri joko annetaan write-metodille tai pyydetään getReadBuffer-metodilla, kun taas siirtokerrospuskurit ovat yksityisiä. SecureChannelin lopuksi vielä ajetaan alas shutdown-metodilla, jonka jälkee se voidaan sulkea close-metodilla. Shutdown-metodi sulkee SSLEnginen ja lähettää ”close_notify”-varoitus viestin, jonka jälkeen close-metodin kutsulla voi sulkea SocketChannel-yhteyden.

```
private SocketChannel mSocketChannel;
private SSLEngine mSSLEngine = null;

private ByteBuffer mInAppBB;
private ByteBuffer mInNetBB;
private ByteBuffer mOutNetBB;

boolean doHandshake(SelectionKey selectionKey) throws IOException
int read() throws IOException
ByteBuffer getReadBuffer()
int write(ByteBuffer outAppBB) throws IOException
boolean flush() throws IOException
boolean shutdown() throws IOException
void close() throws IOException
```

Kuva 20. SecureChannel-luokan tärkeimmät muuttujat ja metodit.

SessionHandler-luokka hallitsee yhteyttä kapsuloimalla edellä mainitun SecureChannel-luokan sisäänsä ja käyttäen sen palveluita. SessionHandler-luokalla on vastaanotto, datan läpikäynti ja lähetystilat, jonka lisäksi sillä on SecureChannel-luokan sisällä handshake-tila. SecureChannel ei aloita handshake-prosessia itse vaan heittää

poikkeuksen, jos read- tai write-metodeja kutsutaan ilman, että handshake on valmis. Kuvassa 21 on tilakaavio SessionHandler-luokasta. Siinä säikeen suoritus alkaa mustista pisteistä ja loppuu punasiin rukseihin, kun niihin saavutaan vihreän valintakohdan kautta. SessionHandler alkaa aloitustilasta. Tässä tilassa se kutsuu doHandshake-metodia, joka suorittaa kuvassa 12 olevan handshake-prosessin yhden tai useamman saman suuntaisen nuolen. Se ei koskaan lue ja kirjoita samassa säikeessä, vaan tässä välissä se vaihtaa kiinnostuksenkohdetta ja selector huomaa tämän ja lisää sen työjonoon. Kun suojattu yhteys on luotu, toimii SessionHandler-luokka vastaanotto- tai lähetystiloissa. Yleensä kuitenkin ensimmäinen säie lukee ja käy datan läpi seuraavan säikeen kirjoittaessa viestin takaisin asiakasohjelmalle. Yhteys on tehty pysymään päällä ja näin ollen aloitustilaan tullaan vain siinä tilanteessa kun yhteys on kaatunut.




Kuva 21. SessionHandler-tilakaavio.

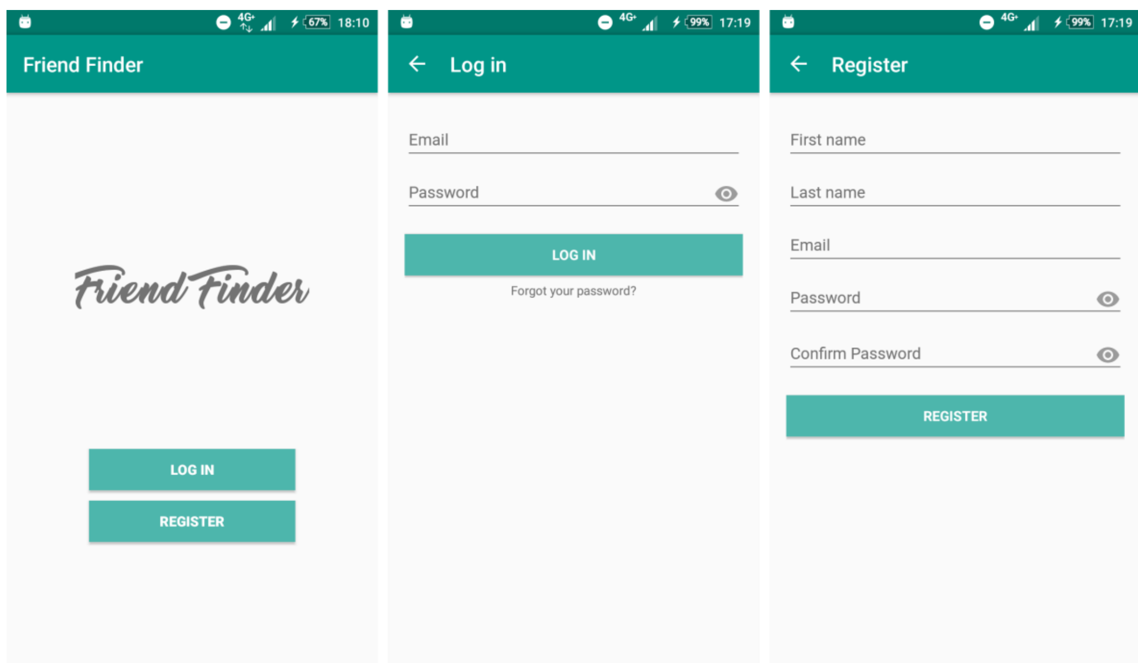
## 4.2 Android-asiakasohjelma

Android-toteutus alkoi luvussa 2.2 esitettyjen näkymien määrittelyn ja ohjelman rakenteen hahmottelulla. Näkymäkerroksen jälkeen täytyi ratkaista, miten puhelin jatkaa sijaintipäivitysten lähettämistä palvelimelle sovelluksen ollessa suljettuna. Tämä myös täy-

tyi hoitaa virtaa säästäen, ettei käyttäjä poista sovellusta sen kovan virrankulutuksen takia. Oli myös tärkeää saada sijaintitietojen lähetys toimimaan heti puhelimen käynnistyessä, jotta käyttäjän ystävät voivat nähdä tämänhetkisen tiedon. Viimeinen haaste oli saada Android-kuuden mukana tulema oikeuksien kieltäminen toimimaan niin, että palvelu kertoo siitä käyttäjälle ja lakkaa toimimasta ilman niitä.

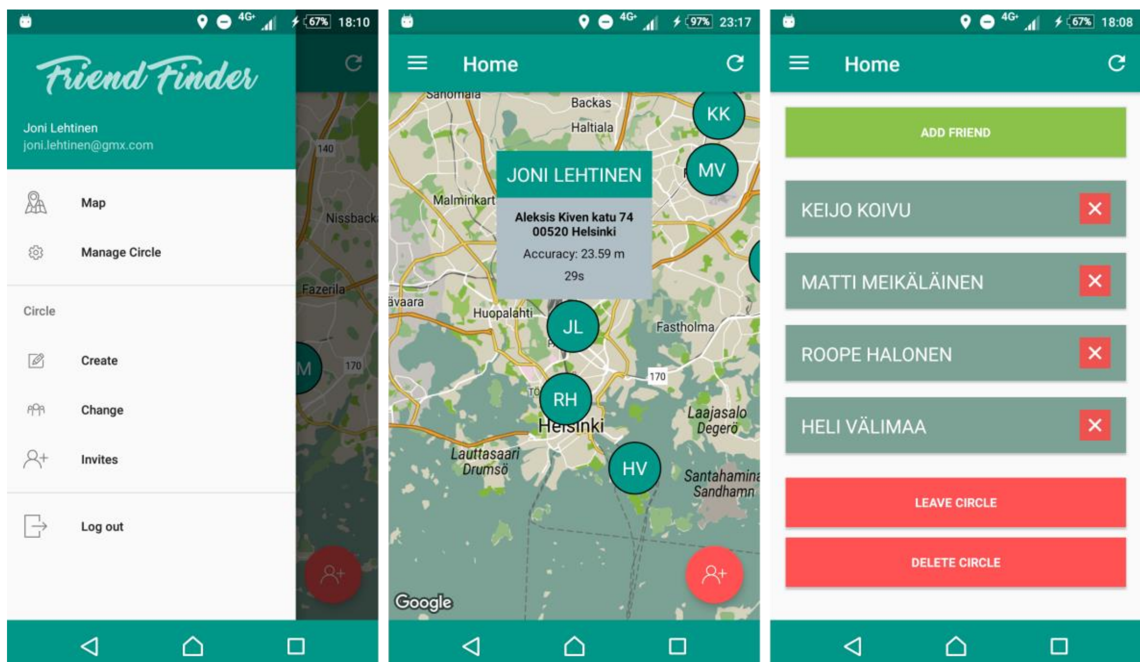
### 4.2.1 Näkymät

Näkymäkerroksen toteutus aloitettiin jakamalla se todennus- sekä päänäkymäosaan. Todentaminen hoidetaan omassa aktiviteetissaan, jossa on kolme fragmenttia kuvan 22 mukaisesti. Itse todennusaktiviteetti ei sisällä näkymää. Sovellukseen kirjautuminen tapahtuu kolmella eri tavalla. Näistä sisäänkirjautuminen tai rekisteröinti tapahtuu käyttäjän toimesta täyttämällä kuvan 22 fragmenteissa olevat lomakkeet, jonka jälkeen aktiviteetti vaihtuu pääaktiviteettiin. Sovellus pysyy kirjautuneena niin kauan ennen kuin käyttäjä kirjaa itsensä ulos pääaktiviteetissa olevan sivuvalikon kautta. Kolmas kirjautumistapa on automaattinen kirjautuminen ohjelman käynnistyessä. Todennusaktiviteetti tarkistaa onko käyttäjän kirjautumistiedot tietokannassa, jos ne löytyvät kirjaudutaan sisälle. Todennusaktiviteetti toimii myös oikeuksien tarkastajana. Pyytäen tarvittavat oikeudet käyttäjältä niiden puuttuessa samalla kertoen, että oikeudet ovat pakollisia ohjelman toimivuuden kannalta. Oikeuksien puuttuessa ohjelma sulkeutuu.




Kuva 22. Todennusaktiviteetti, jossa on rekisteröinti- ja kirjautumisnäkymät.

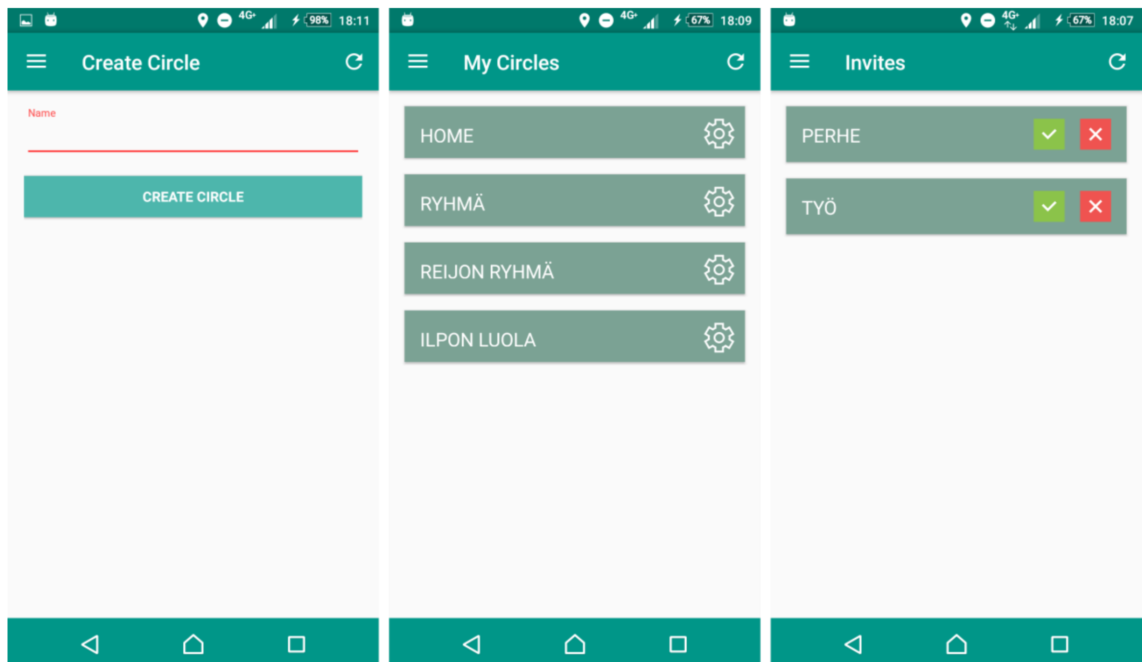
Käyttäjä viettää suurimman osan ajastaan pääaktiviteetissa katsellen ryhmänjäsenten sijainteja karttanäkymästä tai halliten ryhmiä niiden omasta näkymästään (kuva 23 ja 24). Pääaktiviteetti ei todennusaktiviteetin kaltaisesti sisällä näkymää, mutta karttanäkymän logiikka on toteutettu sen sisällä. Pääaktiviteetti alkaa kuvassa 23 olevasta karttanäkymästä, jossa näkyy viimeiseksi avoinna ollut ryhmä tai uuden käyttäjän kohdalla vain oma sijainti. Ryhmän ollessa valittuna näkyy työkalupalkissa kyseisen ryhmän nimi sekä punainen lisää uusi käyttäjä painike alhaalla. Ryhmän jäsenen tunnistaa karttanäkymässä hänen nimikirjaimistaan. Painaessa nimikirjaimilla koristettua palloa tulee tämän päälle ikkuna sisältäen tarkemmat tiedot käyttäjän sijainnista. Tämä sisältää tarkan osoitteen, jos kyseiselle paikalle sitä löytyy sekä sijainnin tarkkuuden ja ajan, kuinka kauan sitten kyseiset arvot on rekisteröity. Pääaktiviteetti sisältää myös työkalupalkista löytyvän päivityspainikkeen, joka hakee palvelimelta käyttäjän ryhmätiedot ja tallentaa ne ohjelman tietokantaan, mikä saa taas aikaan karttanäkymän päivityksen.




Kuva 23. Pääaktiviteetin kartta- ja ryhmänhallintanäkymä sekä navigointivalikko.

Sovelluksen navigointi tapahtuu kuvassa 23 olevan sivuvalikon kautta, jonka saa auki painamalla työkalupalkin vasemmassa reunassa olevaa viiva kuvaketta tai vetäisemällä sormella ruudun vasemmasta laidasta oikeaan päin. Sivupalkki sisältää neljä osiota, jossa ylimpänä on käyttäjän tiedot. Tämän jälkeen tulee tämänhetkisen ryhmän navigointi, joka sisältää painikkeet kartta- sekä ryhmänhallintanäkymän avaamiseksi. Kolmas osio sivupalkista sisältää asioita, jotka eivät liity tämänhetkiseen ryhmään. Näitä

ovat ryhmän luonti ja vaihto sekä kutsujen hyväksyminen, jotka avaat niitä vastaavan kuvan 24 näkymän. Viimeinen osio sivupalkista sisältää uloskirjautumispainikkeen, joka avaa todennusaktiviteetin. Tämä on tärkeä painike kahdesta syystä. Se on ainoa tapa käyttäjän uloskirjautumiseen, sekä dataa palvelimelle lähettävän palvelun pysäyttämiseen. Palvelu lopettaa datan lähettämisen heti uloskirjautumisen jälkeen, mutta sammuu vasta, kun käyttäjä navigoi pois ohjelmasta.

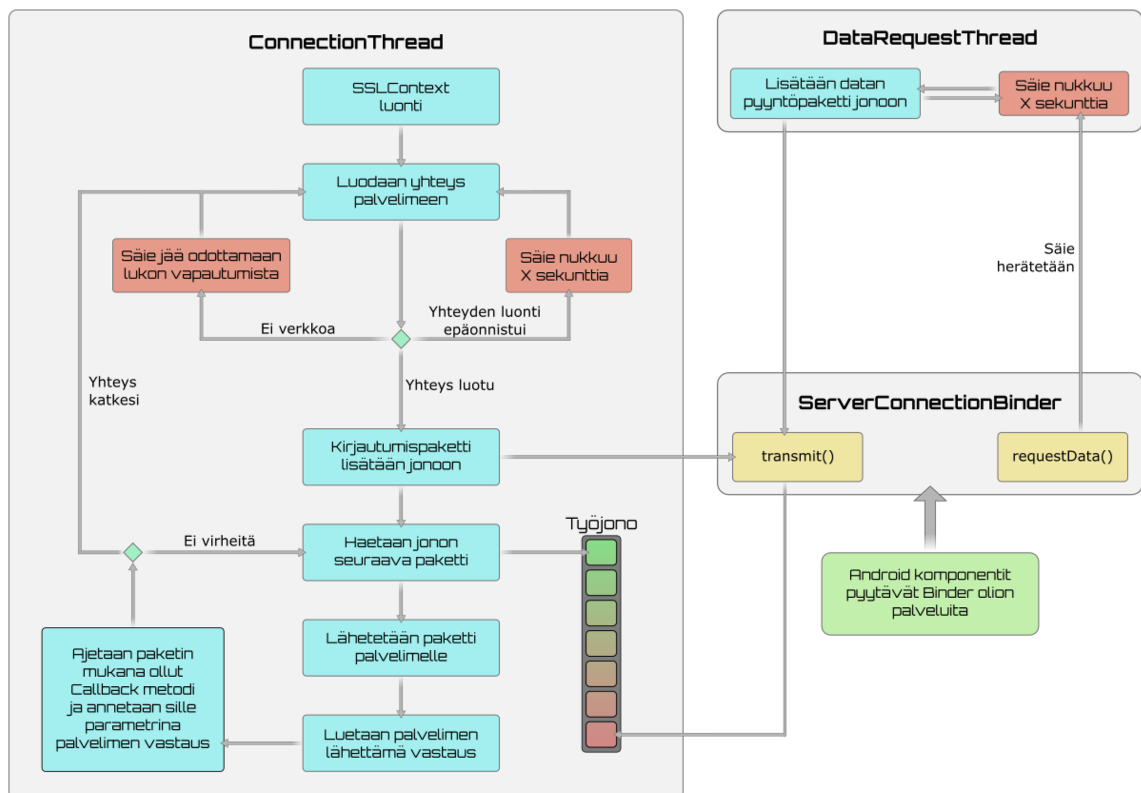



Kuva 24. Pääaktiviteetin ryhmän luonti, vaihto ja kutsun hyväksyntänäkymät.

Kuvassa 23 olevassa ryhmänhallintanäkymässä voi poistaa ryhmän jäseniä sekä itse ryhmän. Kaikki painikkeet, jotka poistavat asioita avaavat ikkunan, jossa käyttäjä vahvistaa poiston. Ylin jäsenen lisäys ryhmään -painike avaa saman näkymän, jonka karttanäkymän punainen painike avaa. Siinä näkymässä käyttäjän tulee syöttää ystävän sähköpostiosoite, jota käytetään käyttäjien tunnistukseen sovelluksessa. Lisäyksen jälkeen ystävä näkee omassa kuvan 24 mukaisessa ryhmäkutsun hyväksymisnäkymässä kutsun kyseiseen ryhmään. Ryhmänluontinäkymä luo käyttäjälle ryhmän ja avaa karttanäkymän, jossa kyseinen ryhmä on valittuna. Kuvan 24 keskimmäinen näkymä on ryhmänvaihto- ja asetusnäkymä. Siinä käyttäjä voi vaihtaa tämänhetkistä ryhmää tai avata jonkun ryhmän hallintanäkymän. Hallintanäkymä on samanlainen kuin sivupalkista aukeava, mutta se ei ole sidoksissa tämänhetkiseen ryhmään, ellei kyseisen ryhmän hallintanäkymää avattu.

### 4.2.2 Service ja säikeet

Android lopettaa prosessit tarvitessaan muistia. Tuleeko kyseinen prosessi lopetetuksi riippuu siinä olevista komponenteista. Tämän takia Android-sovellus ei voi ajaa pelkästään säiettä, vaan se tarvitsee komponentin, jossa säie toimii. Tähän Service on mainio komponentti etenkin, jos näkymäkerrosta ei tarvita. Service itse toimii prosessin pääsäikeessä sovelluksen kanssa. Tämän takia yleensä aina Service on toteutettu luomalla sille oma säie. Sovelluksen Service-toteutus sisältää ConnectionThread- sekä DataRequestThread-säikeet.




Kuva 25. Servicen sisältämät säikeet sekä niiden vuorovaikutus.

ConnectionThread toimii pääsäikeenä Servicelle muodostaen TLS-yhteyden palvelimen kanssa ja hoitamalla kaiken kommunikaation asiakasohjelman ja palvelimen välillä (kuva 25). Kyseisellä säikeellä on monta tilaa elämänsä aikana, mutta vain kolme vaihetta. Ensimmäisessä vaiheessa se alustaa itsensä luoden SSLContextin, jonka jälkeen siirrytään yhteydenotto vaiheeseen. Kyseisen vaiheen jälkeen säie omaa toimivan TLS-yhteyden. Viimeinen vaihe on pakettien lähetys palvelimelle ja vastauksien vastaanottaminen. Yhteyden katketessa palataan luomaan yhteyttä uudelleen, mutta vaiheeseen yksi ei enää mennä.

ConnectionThread toimii työsäikeenä tyhjentäen työjonoa. Työsäikeen kanssa kommunikointi tapahtuu Service-komponentin ServerConnectionBinder-luokan kautta, joka saadaan sitomalla Serviceen. Kyseinen IBinder-toteutus sisältää transmit-metodin, joka tarjoaa ainoan tavan pakettien lisäämiseksi työjonoon. Työjono on toteutettu BlockingQueue-rajapintaa vasten tarjoten näin näkyvyyden säikeiden välillä. Työsäie myös käyttää Binder-oliota kirjautumispaketin lisäämiseen. Yhteydenoton jälkeen työsäie tarkistaa, onko asiakasohjelman tietokannassa kirjautumistiedot. Niiden löytyessä lisätään kirjautumispaketti työjonoon, joka on aina ensimmäinen paketti. Näin ollen kirjautuminen tapahtuu aina ensimmäisenä työsäikeen päästessä vaiheeseen kolme. Tämä taataan tyhjentämällä työjono yhteyden katketessa. Työsäie lähettää palvelimen vastauksen asynkronisesti, jotta paketin antajan ei tarvitse blokata säiettä, mistä se lähetetään. Tätä varten transmit-metodille voi antaa parametrina callback-metodin, jota kutsutaan vastauksen saapuessa palvelimelta. Sille annetaan parametrina palvelimen vastaus ja se ajetaan sovelluksen pääsäikeessä, jotta callback-metodit voivat kutsua käyttöliittymän metodeita.

Työsäikeen lisäksi Service sisältää DataRequestThread-säikeen. Sen tehtävä on lisätä tietyn ajan välein datapyyntö paketteja työpuskuriin. Datapyyntöpaketti pyytää palvelimelta käyttäjän ryhmätiedot, joita tarvitaan asiakasohjelman ryhmädatan päivitykseen. Käyttäjä voi myös pyytää datan päivitystä kuvien 23 ja 24 oikeasta ylänurkasta löytyvällä päivityspainikkeella. Tämä saa säikeen pyytämään pakettia heti, jonka jälkeen se jatkaa normaali suoritustaan. DataRequestThread-säie on toiminnassa vain sovelluksen ollessa päällä ja näkyvänä näytöllä. Sovelluksen sulkeutuessa sammutetaan säie Service-komponentin Binder-oliota käyttäen.

Service-komponentin käyttö takaa säikeiden päällä pysymisen. Palauttamalla START_STICKY-arvon Servicen onStartCommand-metodista varmistamme, että Service käynnistetään uudelleen järjestelmän tuhotessa sen. Käyttäjän kirjautumisen jälkeen Service pysyy päällä niin kauan ennen kuin käyttäjä kirjautuu ulos painamalla kuvan 23 kirjaudu ulos painiketta. Edes laitteen uudelleen käynnistäminen ei lopeta sitä, vaan se käynnistyy laitteen käynnistyessä uudelleen OnBootReceiver-luokan toimesta. Kyseinen luokka on rekisteröitynyt saamaan käyttöjärjestelmältä signaalin, kun käynnistyminen on valmis ja sen saatuaan käynnistää Servicen (kuva 8 ja 9).

### 4.2.3 Paikannus

Service-komponentti sisältää säikeiden lisäksi myös paikannuksen. Paikannus tapahtuu sovelluksessa käyttäen Google play -palvelun paikannusrajapintaa. Tämä on Androidin uusi tapa sijaintitietojen hakemiseen ja sen käyttöä suositellaan vanhan android.location-paikannusrajapinnan sijaan. Google play -paikannusrajapinnan etuna on sen virtaa säästävä tapa hakea paikannustiedot.

```
@Override
public void onCreate() {
    if (mGoogleApiClient == null) {
        mGoogleApiClient = new GoogleApiClient.Builder(this)
                .addConnectionCallbacks(this)
                .addOnConnectionFailedListener(this)
                .addApi( LocationServices.API )
                .build();
    }
}

@Override
public void onConnected(@Nullable Bundle bundle) {

    mLocationRequest = new LocationRequest()
        .setInterval(LOCATION_UPDATE_INTERVAL)
        .setFastestInterval(LOCATION_UPDATE_INTERVAL)
        .setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

    LocationSettingsRequest.Builder builder = new LocationSettingsRequest.Builder()
        .addLocationRequest(mLocationRequest);

    PendingResult<LocationSettingsResult> result = LocationServices.SettingsApi
        .checkLocationSettings(mGoogleApiClient, builder.build());

    result.setResultCallback(new ResultCallback<LocationSettingsResult>() {
        @Override
        public void onResult(LocationSettingsResult result) {
            switch (result.getStatus().getStatusCode()) {
                case LocationSettingsStatusCodes.SUCCESS:
                    mLocationSettingsValid = true;

                    if(mIsLoggedIn)
                        startLocationUpdates();

                    break;
                default:
                    mGoogleApiClient.disconnect();
                    permissionError();
            }
        }
    });

}

private void startLocationUpdates() {
    if ( ContextCompat.checkSelfPermission(this, android.Manifest.permission.ACCESS_FINE_LOCATION) ==
         PackageManager.PERMISSION_GRANTED) {
        mLocationUpdateStarted = true;
        LocationServices.FusedLocationApi.requestLocationUpdates(mGoogleApiClient, mLocationRequest, this);
    } else {
        permissionError();
    }
}

@Override
public void onLocationChanged( Location location ) {

    Sendable mSendable = new fi.joni.lehtinen.friendfinder.connectionprotocol.dto.Location(
            mUserID, location.getLatitude(), location.getLongitude(), location.getAccuracy(), location.getTime() );

    /* Sovelluksen päivitys koodi poistettu */

    mBinder.transmit( ConnectionProtocol.Protocols.LOCATION, mSendable, null );
}
```

Kuva 26. Google Play -paikannusrajapinnan osittainen toteutus Service-komponentissa.

Kuvassa 26 on Service-komponentissa oleva paikannus käyttäen Google play -palvelua. Siinä rakennetaan ensin GoogleApiClient-luokka, joka määrittelee, mitä Googlen palveluita tullaan käyttämään. Sen yhdistäminen ja yhteyden katkaisu tapahtuvat onStartCommand- ja onDestroy-metodeissa, jotka on jätetty kuvasta pois. Yhteyden saatua kutsuu käyttöjärjestelmä onConnected-metodia, jossa tarkistetaan laitteen asetukset. Sijaintipalvelun ollessa päällä ja omistaessamme tarvittavat oikeudet aloitetaan sijaintipäivitykset. Järjestelmä kutsuu onLocationChanged-metodia LocationRequest-luokassa määritellyn ajan välein, mutta se voi myös kutsua sitä aikaisemmin. Tämän takia on tärkeää määritellä nopein aikaväli sijaintitietojen päivitykselle. Paikannuksen tapahtuessa päivitetään tietokantaa ja lisätään paikannuspaketti työsäikeelle käyttäen Binder-luokkaa, jonka työsäie lähettää edelleen palvelimelle. Omistaessamme puutteelliset oikeudet tai sijaintipalvelun ollessa pois päältä ei Service itse voi pyytää käyttäjältä niiden muutosta. Tämän takia Service joko sulkeutuu tai lähettää pääaktiviteetille viestin tarvittavien oikeuksien puutteesta, jolloin pääaktiviteetti kirjaa käyttäjän ulos ja avaa todennusaktiviteetin.

### 4.2.4 Oikeudet ja asetukset

Android-kuuden mukana tuli käyttäjän mahdollisuus yksittäisten oikeuksien kieltämiseen. Tämän takia täytyy oikeudet aina tarkistaa ennen niitä vaativia toimintoja. Toimiakseen sovellus tarvitsee oikeuden tarkkaan sijaintitietoon sekä sijaintipalvelun päällä olemisen. Niiden tarkastus on toteutettu todennusaktiviteetissa. Aktiviteetin käynnistyessä tarkistetaan tarvittavat oikeudet ja sijaintipalvelun päällä oleminen, joiden puuttuessa annetaan käyttäjälle mahdollisuus oikeuksien muuttamiseen. Käyttäjän päättäessä kieltää kyseiset oikeudet ilmoittaa sovellus tarvitsevansa niitä toimiakseen, jonka jälkeen se lopettaa toimintansa. Kuvassa 26 nähtiin oikeuksien tarkastus, joka tapahtui Service-komponentissa eikä tämän takia oikeuksien muuttamisikkunoita voida avata. Oikeudet tarkistetaan sovelluksessa todennusaktiviteetin alussa sekä Servicen käynnistyessä. Tämän lisäksi Service sisältää BroadcastReceiver-toteutuksen, joka kuuntelee sijaintipalvelun muutoksia. Käyttäjän poistaessa sijaintipalvelun Service sulkeutuu heti tai lähettää viestin tästä pääaktiviteetille. Oikeuksien muutoksista ei lähde käyttöjärjestelmältä signaalia, joten ne tarkastetaan Servicen pääsäikeessä aina jokaisen paketin lähetyksen yhteydessä.

### 4.3 Yhteysprotokolla

Palvelin ja asiakasohjelma tarvitsevat yhteisen protokollan kommunikoidakseen toistensa kanssa. TLS-yhteys ei itse rajoita kommunikointiprotokollan valintaa, vaan se jää kokonaisuudessaan kehittäjän päätettäväksi. Protokollaksi kävisi hyvin http, mutta yhtä hyvin voimme laatia omia tarpeita vastaavan oman protokollan.

Kommunikointiprotokollaksi valittiin oma protokolla. Tätä varten luotiin kirjastoprojekti, joka lisättiin palvelin- sekä asiakasohjelmaan. Protokolla perustuu olioiden lähettämiseen ja vastaanottamiseen. Projekti sisältää lähetettävät oliot joka toteuttavat Sendable-rajapinnan sekä luokan, joka muuttaa kyseiset oliot tavujonoiksi ja takaisin. Protokolla on yksinkertainen ja tarvittaessa helposti vaihdettavissa toiseen.

## 5 Yhteenveto

Insinöörityö alkoi asioihin perehtymällä. Täytyi suunnitella sekä paikannussovelluksen sekä sitä tukevan palvelinohjelman rakenne. Alku sisälsi paljon opeteltavaa monilta eri osa-alueilta. Samalla täytyi tehdä päätökset ohjelmissa käytetyistä tekniikoista. Asiakasohjelman tekeminen Androidilla sekä palvelinohjelman tekeminen Javalla olivat omia kiinnostuksen kohteita, joten lähdin niistä liikkeelle. Kummatkin näistä ovat varteenotettavia tekniikoita nykypäivänä omilla osa-alueillaan.

Palvelinohjelman kehitys toi mukanaan monia haasteita, ja vaikka Java on kielenä erittäin tuttu, opittiin matkan varrella paljon uutta. Haastavimmiksi asioiksi palvelinta tehdessä osoittautui säiemallin tekeminen sekä salatun yhteyden aikaan saaminen, kun kummatkin täytyi tehdä skaalautuvasti. Valinta palvelimen tekemiseksi itse oli kuitenkin mainio, ja se opetti erittäin paljon salaustekniikoista, yhteyksistä ja siitä, miten data kulkee laitteiden välillä.

Android-asiakasohjelman kehityksessä oli myös monta ongelmaa ratkaistavaksi. Täytyi toteuttaa Service, joka aina välittää palvelimelle paikannusdataa silti käyttämättä liikaa virtaa sen tekemiseen. Tämä onnistuttiin ratkaisemaan erittäin elegantilla tavalla ja itse Service-komponentista tuli hieno kokonaisuus. Vähävirtaisuus saatiin toteutettua Androidin uudella paikannusrajapinnalla sekä lähettämällä dataa palvelimelle vain tietyin aika-

välein. Vaikka Android oli alustana erittäin tuttu, toi se mukanaan uusia asioita opittavaksi. Näistä Android-kuuden mukana tullut yksittäisten oikeuksien kieltäminen toi lisää työtä projektiin sekä Googlen karttapalveluiden käyttö oli myös mielenkiintoinen kokemus.

Insinöörityön tuloksena syntyi paikannussovellus Android-alustalle sekä skaalautuva Java-palvelin. Näistä kummastakin saatiin tehtyä toimiva kokonaisuus, joka täytti niille annetut määrittelyt. Lopputuotteeseen voi olla erittäin tyytyväinen sen sulavan toiminnallisuuden sekä ulkonäön puolesta.

Insinöörityössä kehitetty sovellus on kaikin puolin toimiva ja määrittelyn mukainen, mutta se ei pysty kilpailemaan toiminnallisuudellaan Google Play -kaupassa olevien vastaaviin sovelluksien kanssa vaan tarvitsee jatkokehitystä. Suurimpana tarpeena on nähtävissä paikannushistorian tarjoaminen käyttäjille. Tämä lisää palvelimen kuormitusta sekä tietokannan datan määrä, joka on otettava huomioon jatkokehityksessä. Oikeanlainen rahastusmalli on myös kehitettävä ennen sovelluksen julkistamista. Tämän tulee kattaa ainakin palvelimen kulut, jotta tuote olisi omavarainen. Palvelin tarvitsee myös jatkokehitystä ja testausta isolla kävijämäärällä ennen sovelluksen julkaisemista.

## Lähteet


1 IDC. Smartphone OS Market Share, 2016 Q2. Verkkodokumentti. <http://www.idc.com/prodserv/smartphone-os-market-share.jsp> Luettu 3.11.2016.

2 Android Central. Android Pre-History. Verkkodokumentti. <http://www.androidcentral.com/android-pre-history> Luettu 3.11.2016.

3 Ars Technica. Google’s iron grip on Android: Controlling open source by any means necessary. Verkkodokumentti. <http://arstechnica.com/gadgets/2013/10/googles-iron-grip-on-android-controlling-open-source-by-any-means-necessary/> Luettu 3.11.2016.

4 Android Developer. Dashboards. Verkkodokumentti. <https://developer.android.com/about/dashboards/index.html> Luettu 14.11.2016.

5 Android Developer. <uses-sdk>. Verkkodokumentti. <https://developer.android.com/guide/topics/manifest/uses-sdk-element.html> Luettu 3.11.2016.

6 Android Developer. App Manifest. Verkkodokumentti. <https://developer.android.com/guide/topics/manifest/manifest-intro.html> Luettu 3.11.2016.

7 xml.com. A Technical Introduction to XML. Verkkodokumentti. <http://www.xml.com/pub/a/98/10/guide0.html> Luettu 5.11.2016.

8 Android Developer. Activitíes. Verkkodokumentti. <https://developer.android.com/guide/components/activities.html> Luettu 5.11.2016.

9 Android Developer. Fragments. Verkkodokumentti. <https://developer.android.com/guide/components/fragments.html> Luettu 5.11.2016.

10 Android Developer. Services. Verkkodokumentti. <https://developer.android.com/guide/components/services.html> Luettu 5.11.2016.

11 Android Developer. Bound Services. Verkkodokumentti. <https://developer.android.com/guide/components/bound-services.html> Luettu 5.11.2016.

12 Android Developer. Displaying a Location Address. Verkkodokumentti. <https://developer.android.com/training/location/display-address.html> Luettu 5.11.2016.

13 Android Developer. Content Provider Basics. Verkkodokumentti. <https://developer.android.com/guide/topics/providers/content-provider-basics.html> Luettu 6.11.2016.

14 Android Developer. BroadcastReceiver. Verkkodokumentti. <https://developer. android.com/reference/android/content/BroadcastReceiver.html> Luettu 6.11.2016.

15 Android Developer. Resources Overview. Verkkodokumentti. <https://developer.android.com/guide/topics/resources/overview.html> Luettu 6.11.2016.

16 Android Developer. Processes and Threads. Verkkodokumentti. <https://developer.android.com/guide/components/processes-and-threads.html> Luettu 6.11.2016.

17 Android Developer. Loaders. Verkkodokumentti. <https://developer.android.com/guide/components/loaders.html> Luettu 6.11.2016.

18 T. Dierks. 2008. TLS 1.2. Verkkodokumentti <https://tools.ietf.org/html/rfc5246> Luettu 7.11.2016.

19 Hans Delfs & Helmut Knebl. 2007. Introduction to Cryptography. Berlin: Springer.

20 What is SSL and what are Certificates. Verkkodokumentti. <http://www.tldp.org/HOWTO/SSL-Certificates-HOWTO/x64.html> Luettu 8.11.2016.

21 JDBC Overview. Verkkodokumentti. <http://www.oracle.com/technetwork/java/overview-141217.html> Luettu 9.11.2016.

22 Call Level Interface. Verkkodokumentti. <https://en.wikipedia.org/wiki/Call_Level_Interface> Luettu 9.11.2016.

23 Java Cryptography Architecture (JCA) Reference Guide. Verkkodokumentti. <http://docs.oracle.com/javase/8/docs/technotes/guides/security/crypto/CryptoSpec.html> Luettu 9.11.2016.

24 Java Secure Socket Extension (JSSE) Reference Guide. Verkkodokumentti. <https://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html> Luettu 9.11.2016.

25 SSLEngine. Verkkodokumentti. <https://docs.oracle.com/javase/7/docs /api/javax/net/ssl/SSLEngine.html> Luettu 9.11.2016.

26 SQL. Verkkodokumentti. <https://en.wikipedia.org/wiki/SQL> Luettu 8.11.2016.

27 SQLite. Verkkodokumentti. <https://sqlite.org/about.html> Luettu 8.11.2016.

28 Philip A. Bernstein & Eric Newcomer. 2009. Principles of transaction processing 2nd 2ed. Burlington: Morgan Kaufmann.

29 PostgreSQL about. Verkkodokumentti. <https://www.postgresql.org/about/> Luettu 8.11.2016.

30 PostgreSQL documentation. Verkkodokumentti. <https://www.postgresql.org/files/documentation/pdf/9.6/postgresql-9.6-US.pdf> Luettu 8.11.2016.

31 Apache Commons DBCP. Verkkodokumentti. <http://commons.apache.org/proper/commons-dbcp/> Luettu 8.11.2016.