OULUN AMMATTIKORKEAKOULU

Jari Tolonen

MITTAUSDATAN VÄLITYS GOOGLE CLOUD MESSAGING FOR ANDROID -PALVELUN AVULLA

MITTAUSDATAN VÄLITYS GOOGLE CLOUD MESSAGING FOR ANDROID -PALVELUN AVULLA

Jari Tolonen
Opinnäytetyö
Kevät 2014
Tietotekniikan koulutusohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, sulautetut järjestelmät

Tekijä: Jari Tolonen
Opinnäytetyön nimi: Mittausdatan välitys Google Cloud Messaging for Android -palvelun avulla.
Työn ohjaaja: Teemu Korpela
Työn valmistumislukukausi ja -vuosi: Kevät 2014
Sivumäärä:44 + 5 liitettä

Työssä suunniteltiin järjestelmä, jossa siirretään useilta fyysisesti eri paikoissa sijaitsevilta mittauspaikoilta automaattisesti päivittyvää mittausdataa palvelimelle sekä jaetaan datapalvelimelta Android-sovellukselle käyttäen Google Cloud Messaging for Android -palvelua.

Työ jakautui kahteen vaiheeseen. Ensimmäisessä osassa suunniteltiin siirtomekanismi mittauspaikalta palvelimelle. Jälkimmäisessä osassa otettiin käyttöön Google Cloud Messaging -palvelu ja suunniteltiin palvelinohjelmisto sekä Android-sovelluksen tiedonsiirto.

Työn tuloksena saatiin luotettavasti toimiva tiedonsiirtomekanismi mittauspaikalta palvelimelle sekä GCM-palvelun kautta toimiva mittausdatan päivitys Android-sovellukselle.

Asiasanat: tiedonsiirto, palvelimet, tietokannat, pilvipalvelut, Android, Linux

ABSTRACT

Oulu University of Applied Sciences
Degree programme in Information Technology, embedded systems

Author: Jari Tolonen
Title of thesis: Sensor data distribution via Google Cloud Messaging for Android service.
Supervisor: Teemu Korpela
Term and year when the thesis was submitted: Spring 2014
Pages: 44 + 5 appendices

The objective of this Bachelor's thesis was to design data transmission system for automatically measured sensor data. Data is collected from multiple measurement installation locations to data server and distributed to Android devices via Google Cloud Messaging service.

Design project was divided for two individual main phases. The first part consists of design data transmission system from measurement location to server side. During the second phase, Google Cloud Messaging for Android service along with server- and Android-application was implemented to system.

As a result of this thesis work, a reliable data transferring system from measurement locations to server was created. Also data transmission from server to android devices via Google Cloud Messaging service was implemented to system.

Keywords: data transmission, servers, databases, cloud services, Android, Linux

SISÄLLYS

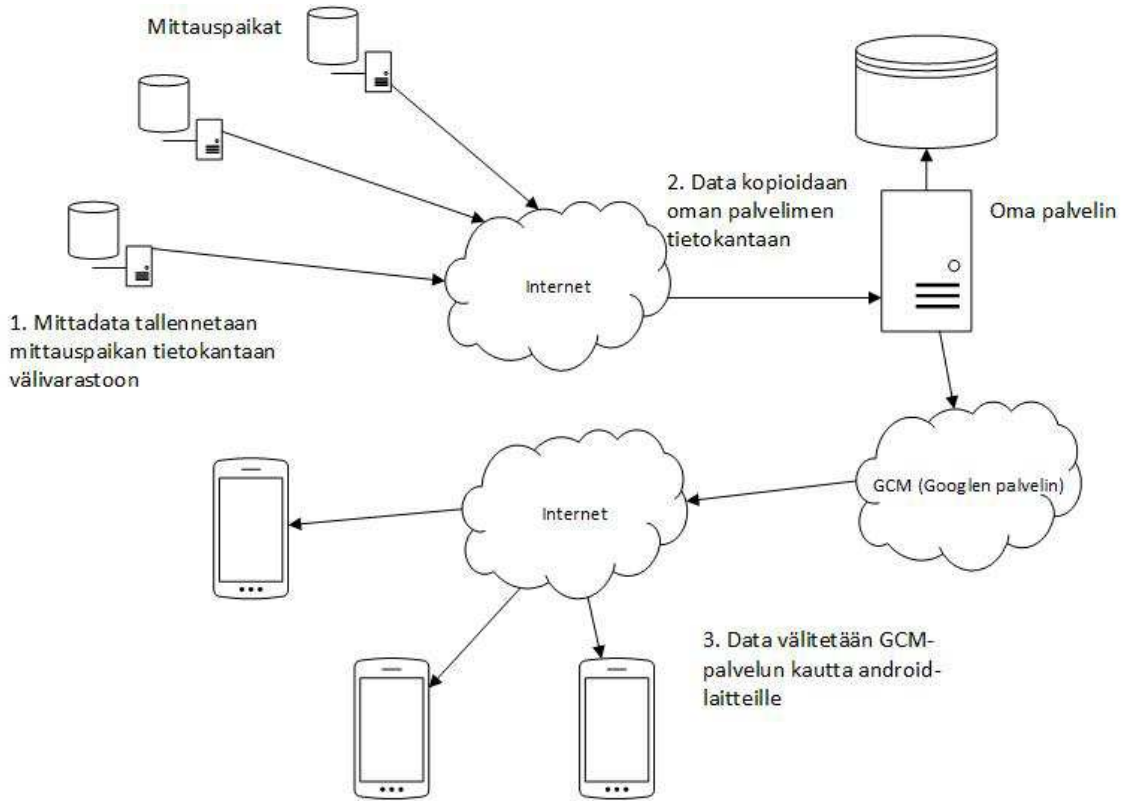
1 JOHDANTO

Työn tarkoituksena oli suunnitella mittausdatan tiedonsiirtomekanismi eri paikkoihin hajautetuilta mittauspaikoilta palvelimelle sekä mittausdatan jakelu push-tyyppisesti Google Cloud Messaging for Android -palvelun avulla Android-laitteille. Lopullinen tavoite on suunnitella Android-sovellus, jossa voidaan esittää mittausdatoista erilaisia graafisia kuvaajia sekä ennusteita.

Mittauspaikat koostuvat erilaisia laitekokoonpanoja sisältävistä tietokoneista sekä Linux-käyttöjärjestelmän päälle rakennetusta mittausohjelmistosta. Ohjelmisto lukee tietokoneeseen kytkettyyn Onewire-väylään liitettyjen antureiden lukemia ja tallentaa lukemat välivarastoon mittauspaikan MySQL-tietokantaan.

Alkuperäinen ajatus mittauspaikkojen rakentamiseen lähti omasta mielenkiinnosta seurata oman asunnon sähkönkulutusta sekä huoneiston lämpötiloja. Ensimmäinen versio mittauspaikasta suunniteltiin opintoihin kuuluneen harjoittelu-projekti-opintojakson aikana ja lopulliseen muotoonsa mittauspaikan ohjelmisto rakentui opintojakson päättymisen jälkeen harrastusprojektina.

Tässä työssä oli tarkoitus perehtyä Node.js-palvelinohjelmiston käyttöön sekä Google Cloud messaging -palvelun käyttöönottoon. Google Cloud Messaging -palvelu valittiin tiedonjakelun siirtokanavaksi, koska lähes jokainen Android-laite käyttää entuudestaan Googlen push-palvelua. Tällä ratkaisulla haluttiin minimoida laitteen tehonkulutuksen lisäys sovellusta käytettäessä. Node.js valittiin palvelinohjelmistoksi sen vuoksi, että sen mainostetaan olevan niukasti palvelimen resursseja käyttävä palvelinalusta ja siksi hyvin soveltuva kevytrakenteisten palvelimien palvelinohjelmistoksi. Kaavio järjestelmän toimintaperiaatteesta on esitetty kuvassa 1.



KUVA 1. Kaavio koko järjestelmän toimintaperiaatteesta

2 TIETOKANNAN SIIRTO PALVELIMELLE

Työn ensimmäinen kokonaisuus oli toteuttaa mittaustulosten siirto mittauspaikalta palvelimelle. Mittausdata tallennetaan ensin välivarastoon mittauspaikan tietokantaan, josta se siirretään palvelimen tietokantaan. Molemmat tietokannat ovat rakenteeltaan identtisiä, joten mitään erillistä muunnosta ei tarvitse tehdä niiden välillä. Siirron täytyy toimia siten, että vaikka tiedonsiirrossa olisi jokin satunnainen häiriö, ohjelmisto osaa siirtää myös aikaisemmin siirtämättä jääneet mittaustulokset palvelimelle.

2.1 Asiakas

Mittauspaikka koostuu pienitehoisesta tietokoneesta, johon on asennettu Linux-käyttöjärjestelmä sekä tietokantaohjelmisto. Tietokoneen perään on rakennettu Onewire-verkko, johon mittausanturit on kytketty.

Järjestelmä mittaa kahta eri suuretta, kWh-mittarin ledin pulssien lukumäärää/aikayksikkö sekä lämpötilaa celsius-asteina. Mittaustulokset luetaan antureilta ja tallennetaan tietokantaan käyttöjärjestelmän sisäisen cron-ajastuksen ajastamina. Ajastus suoritetaan kerran minuutissa tai kahdessa minuutissa.

2.2 Palvelin

Palvelimen palvelinohjelmistona toimii Node.js-sovellusalustalle tehty verkkosovellus. Sovellus kuuntelee ennalta määrättyä porttia ja käsittelee mittauspaikan JSON-viestit sekä tietokantahaut.

Node.js

Node.js perustana toimii Googlen V8 JavaScript -moottori. Node.js-sovellukset kirjoitetaan JavaScript-kielellä. Node.js perustuu tapahtumapohjaiseen ohjelmointimalliin ja käyttää ei-blokkaavia asynkronisia tapahtumia. Tässä mallissa ohjelmisto luo tapahtumasilmukan, joka vastaanottaa kutsuja verkosta ja ohjelmakoodilta sekä ohjaa tapahtumat välittömästi eteenpäin omalle asynkroniselle funktiolle.

Node.js-ohjelmoinnin tukena käytettiin Mozilla-säätiön JavaScript-dokumentaatiota (1) sekä Node.js kehittäjän virallista dokumentaatiota (2, linkki Docs).

Seuraavassa on esitetty esimerkki yksinkertaisesta HTTP-palvelimesta.

```
var http = require('http');

http.createServer(function (req, res) {

res.writeHead(200, {'Content-Type': 'text/plain'});

res.end('Hello World\n');

}).listen(80);
```

Esimerkissä luodaan HTTP-pyyntöjä kuunteleva palvelin ja säie laitetaan nukkumaan. Kun asiakas lähettää porttiin HTTP-pyynnön, suoritetaan callback-tyyppisen createServer-funktion koodi. Esimerkin funktiossa lähetetään asiakkaalle HTTP-sivu, joka sisältää sivun header-osion sekä tekstin ’Hello World’.

Tarvittavat Node.js-moduulit

Node.js sisältää joukon sisäänrakennettuja moduuleja, joihin mm. palvelimen luomiseen tarvittava HTTP-moduuli sisältyy. Moduuli otetaan käyttöön ohjelmakoodin alkuun sijoitettavalla require-määrittelyllä. Esimerkiksi, ’*var http = require(’http’);’.*

Node.js:n käyttämät lisämoduulit ladataan sisäänrakennetun latausjärjestelmän avulla. Moduulit asennetaan sovelluskohtaisesti, joten jokaiselle sovellukselle ladataan vain tarvittavat kirjastot.

Työssä tarvittiin lisäkirjastoa MySQL-tietokantahakujen suorittamiseen. Moduuli asennetaan konsoli-ikkunassa siirtymällä sovelluksen sisältävään kansioon ja antamalla komento ’*sudo npm install mysql’*. Tämän jälkeen moduuli voidaan ottaa käyttöön edellä kuvatulla tavalla: ’*var mysql = require(’mysql’);’.*

Omien moduulien tekeminen Node.js-sovellukseen tapahtuu luomalla uusi .js-päätteinen tiedosto sekä kirjoittamalla sen sisään haluttu funktio. Tiedoston loppuun kirjoitetaan exports-rivi, joka mahdollistaa funktion käytön tiedoston ulko-

puolella. Moduuli otetaan käyttöön kutsuvassa ohjelmakoodissa lisäämällä require-rivi kuten edellä. Esimerkiksi luodaan moduuli tietokantaHakuModuuli.js, kirjoitetaan sille funktio haeviimeinen() ja otetaan se käyttöön kutsuvassa ohjelmassa.

Tiedoston tietokantaHakuModuuli.js sisältö:

```
var mysql = require('mysql');

function haeViimeinen(){
// Suorita tässä osiossa jotain
}
exports.haeViimeinen = haeViimeinen;
```

Kutsuvassa funktiossa esitellään moduuli sekä kutsutaan funktiota normaalisti:

```
var tietokantahaut = require('./tietokantaHakuModuuli.js');

tietokantahaut.haeviimeinen();
```

2.3 Tietokanta

Tietokantaohjelmistona järjestelmässä toimii MySQL-tietokantaohjelmisto. Tietokannan sekä tietokantahakujen suunnittelussa käytettiin hyväksi Oraclen tuottamaa, verkossa julkaistua dokumentaatiota (3, linkki MySQL 5.5 Reference Manual) sekä OAMK:n verkkosivuilla olevaa MySQL-itseopiskelumateriaalia (4).

Molemmille mittaussuureille tehtiin tietokantaan oma taulu, johon mittaustulokset tallennetaan. Taulun rakenteet ovat kuvien 2 ja 3 mukaiset.

Kentän nimi	Tietotyyppi	Attribuutit	Muuta
mittaus	int	UNSIGNED	
sn	tinyint	UNSIGNED	
pulssilukema	mediumint		
aikaleima	datetime		

KUVA 2. KWhmittaus MySQL-taulun rakenne

Kentän nimi	Tietotyyppi	Attribuutit	Muuta
mittaus	int	UNSIGNED	
sn	tinyint	UNSIGNED	
astemaara	float(5,1)		
aikaleima	datetime		

KUVA 3. Lämpötilamittaus MySQL-taulun rakenne

Palvelimelle luotiin oma tietokanta jokaista mittauspaikkaa kohden. Palvelimen tietokannan taulut ovat identtiset mittauspaikan taulujen kanssa sillä erotuksella, että mittaus-kentän auto-increment-attribuutti jätettiin pois.

2.4 Tiedonsiirtotapa

MySQL-tietokannan kopiointi koneelta toiselle on mahdollista tehdä useilla tavoilla, mm. käyttämällä MySQL-Replication- tai MySQLdump-toimintoja tai luomalla oma siirtomekanismi.

Tässä työssä oli tarkoituksenmukaista luoda oma Web Services -tyyppinen siirtomekanismi, koska tarkoituksena on myös myöhemmin siirtää mittaustuloksia suoraan mittausantureilta tietokantaan ilman välivarastointia mittauspaikan tietokantaan. Tiedonsiirtomuodoksi työssä valittiin JSON-tiedonsiirtomuoto, koska haluttiin tutustua sen käyttöön ja ominaisuuksiin tiedonsiirrossa erilaisten järjestelmien kesken.

JSON

JSON (JavaScript Object Notation) on yksinkertainen tiedonsiirtomuoto, joka soveltuu hyvin erilaisten ohjelmistoalustojen väliseen tiedonsiirtoon. JSON-tiedonsiirtomuodon kuvaus on luettavissa www-osoitteessa http://www.json.org (5).

JSON-syntaksi

JSON-paketti perustuu nimi-arvo-pareihin. Seuraavassa esimerkissä kenttä ”kwhlukema” saa arvokseen numeroarvon 155.

```
”kwhlukema” : 155
```

JSON-arvo-kentän tyyppi voi olla jokin seuraavista:

- numero (int, float)
- merkkijono
- totuusarvo (true, false)
- taulukko tai lista (hakasulkeiden sisällä)
- objekti (aaltosulkeiden sisällä).

JSON-objekti kirjoitetaan aaltosulkeiden sisälle. Objekti voi sisältää useita nimi-arvo-pareja, jotka erotellaan toisistaan puolipisteellä.

```
{”kwhlukema” : 155; ”kwhlukema” : 305}
```

JSON-taulukko kirjoitetaan hakasulkeiden sisälle ja taulukko voi sisältää useita JSON-objekteja.

```
esimerkki_array: {
”tietokanta” : ”mittauspaikka1”;
”taulu” : ”kwhlukema”,
”mittaustulokset” : [{”kwhlukema” : 155; ”aikaleima” : ”2014-02-26 20:52:01”}
{”kwhlukema” : 103 ; ”aikaleima” : ”2014-02-26 20:53:02”}
{”kwhlukema” : 355 ; ”aikaleima” : ”2014-02-26 20:54:02”}
]
```

Esimerkissä *esimerkki_array:n* objekteja ovat ”tietokanta”, ”taulu” sekä ”mittaus-tulokset”.

2.5 Lähetys

Mittaustulosten lähetystoiminto toteutettiin PHP-kielellä. Lähetys on ajastettu tapahtumaan heti sen jälkeen, kun mittausanturit on luettu. PHP-ohjelmoinnissa tukeuduttiin PHP-kehitysryhmän julkaisemaan verkkodokumentaatioon (6).

Lähetystapahtuman tapahtumakaavio on seuraavanlainen, esimerkki kwhpulssi-taulun tiedonsiirtotapahtumasta.

1. Lähetetään palvelimelle JSON-viesti

```
{”metodi” : ”kysely_viimeinen” ; ”tietokanta” : ”kwhpulssit” ; ”taulu” : ”kwhpulssit” }
```

Ilmoitetaan palvelimelle, että halutaan suorittaa tietokantaan viimeisen mittausindeksin kysely, sekä ilmoitetaan tietokannan ja taulun nimi, josta kysely halutaan tehdä.

2. Vastaanotetaan palvelimelta JSON-muotoinen viesti, esimerkiksi

```
{”mittaus” : 65819}
```

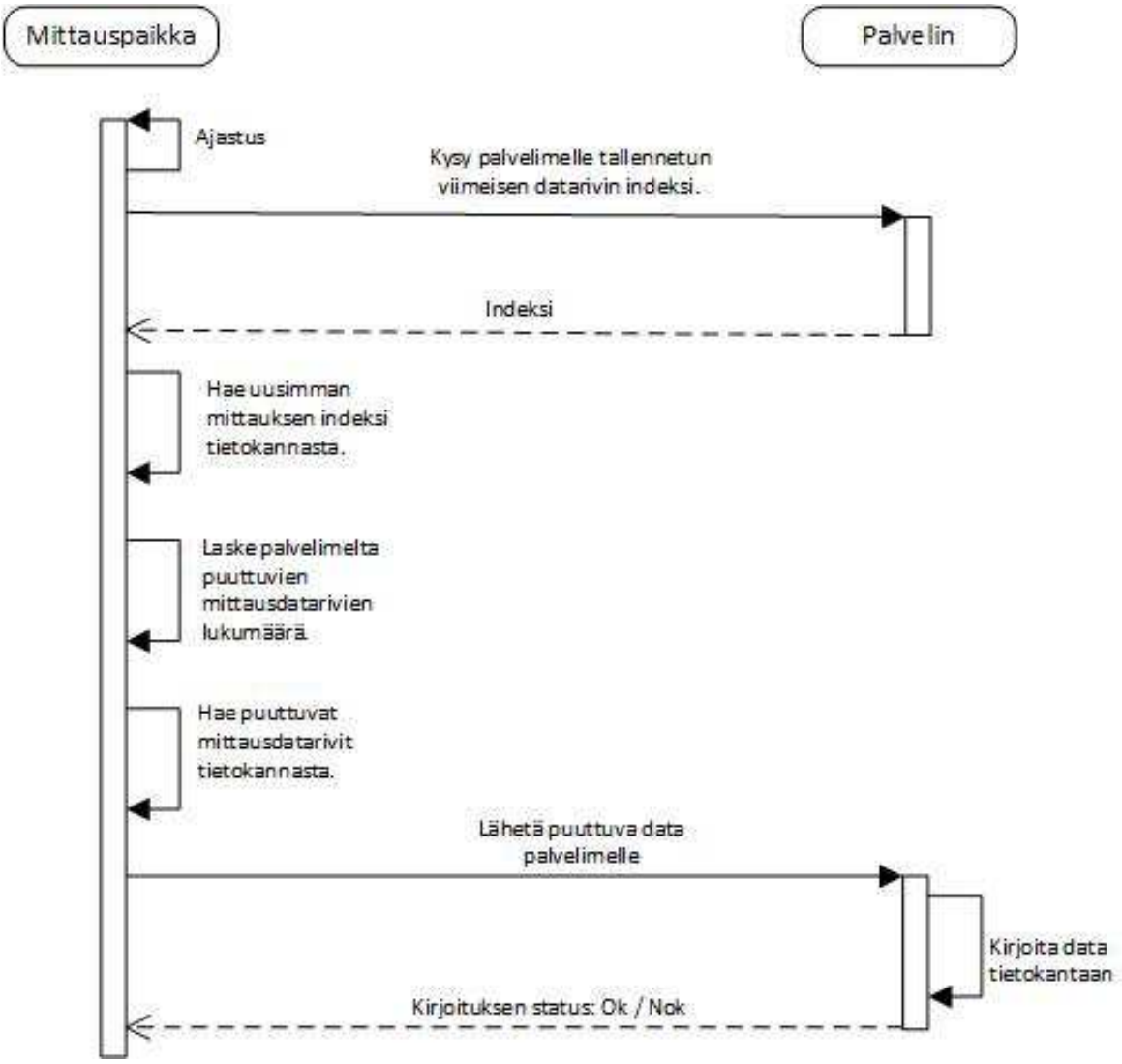
3. Haetaan omasta (mittauspaikan) tietokannasta viimeisen mittauksen indeksi.
4. Lasketaan, kuinka monta mittausta palvelimen tietokannasta puuttuu.
5. Jos puuttuvia mittauksia on enemmän kuin 0, aloitetaan tiedonsiirto.

Tiedonsiirto suoritetaan hakemalla *n* kappaletta rivejä tietokannasta. Rivit luetaan taulukkoon, joka lisätään JSON-viestiin objektiksi. Muita objekteja ovat metodi, joka ilmoittaa, että tulossa on tallennettavaa mittausdataa, sekä tietokannan ja taulun nimet, johon data tallennetaan.

JSON-viesti kokonaisuudessaan:

```
esimerkki_array: {
”metodi” : ”datalahetys” ;
”tietokanta” : ”mittauspaikka1” ;
”taulu” : ”kwhlukema”;
”mittausrivit” :
[ {”mittaus”: 65821” ; sn”: 1 ; ”kwhlukema” : 155 ; ”aikaleima” : ”2014-02-2620:52:01”}
{”mittaus” : 65820” ; sn”: 1 kwhlukema” : 103 ; ”aikaleima” : ”2014-02-26 20:53:02”} ]
}
```

Lähetystapahtumassa lähetetään korkeintaan *n* kappaletta mittaustuloksia yhdellä lähetyskerralla, koska palvelimella käytetty JSON-parseri (Node.js - parse-json) suorittaa virhetoiminnon, mikäli parsittava JSON-paketti on liian suuri. Koska itse lähetystapahtuman aikana oman tietokannan viimeistä mittausindeksiä ei tarkasteta, lähetystapahtuma on rajoitettu ajallisesti 40 sekuntiin, jotta tietokantaan ei ehdi tulla uusia mittaustuloksia siirron aikana. Lähetystapahtuman sekvenssikaavio on esitetty kuvassa 4.



KUVA 4. Mittausdatan lähetys palvelimelle

2.6 Vastaanotto

Palvelin toteutettiin Node.js:n HTTP-moduulia hyväksikäyttäen. Moduulin avulla luodaan palvelin, joka kuuntelee TCP-porttia 61280. Portti kuuluu Internet Assigned Numbers Authorityn (IANA) mukaan alueeseen, jota ei voi rekisteröidä pysyvään käyttöön, vaan on tarkoitettu mm. yksityiseen sekä väliaikaiseen käyttöön (7).

Palvelinohjelmistoon tehtiin jokaista tietokantaoperaatiota varten omat funktiot, jotka asiakas pyytää metodi-objektissa suorittamaan. Funktioihin kirjoitettiin valmiiksi koodin sisälle tietokannan käyttäjätunnukset sekä MySQL-hakua varten kyselyn runko. Tämä toteutettiin, jotta pienennettäisiin riskiä joutua SQL-

injektiohyökkäyksen uhriksi. Funktioiden toiminta on esitelty jäljempänä luvussa Tietokantahaut.

Palvelin

Aluksi luodaan HTTP-palvelin, joka laitetaan lepotilaan odottelemaan HTTP-kyselyjä. Kun asiakkaalta tulee HTTP-kysely, luetaan kyselyn otsikkotiedoista, onko kyseessä POST-tyyppinen kysely. Mikäli kyseessä on POST-tyyppinen kysely, aloitetaan viestin käsittely. Mikäli kysely on GET-tyyppinen, ohjataan kysely toiseen osoitteeseen jossa on selkokielinen sivu.

POST-viestin käsittely:

1. Otetaan vastaan tulevan viestin dataosio muuttujaan. Data sisältää JSON-merkkijonon.
2. Parsitaan JSON-merkkijono json.parse-metodin avulla.
3. Luetaan ’metodi’-objektin sisältö ja kutsutaan sen perusteella määrättyä MySQL-haun tekevää funktiota.
    A. Mikäli kyselyssä halutaan hakea viimeinen mittausindeksi, kutsutaan funktiota tietokantahaut.haeViimeinen(). Funktiolle annetaan parametriksi JSON-paketissa tulleet tietokannan ja taulun nimet. Funktio palauttaa arvonaan taulun viimeisen mittausrivin numeron.
    B. Mikäli kyselyssä objektin ’metodi’ arvona on ’datalähetys’, kutsutaan funktiota tietokantahaut.kirjoitaUudet(). Funktiolle välitetään JSON-paketti kokonaisuudessaan. Funktio palauttaa arvonaan tiedon tietokantaoperaation onnistumisesta.
4. Kun funktiot ovat suorittaneet tehtävänsä, kirjoitetaan porttiin tietoa. A-kohdassa kirjoitetaan MySQL-kyselyltä tullut mittausindeksi. B-kohdassa kirjoitetaan HTTP-statuskoodia vastaava numero. Kirjoituksen onnistuessa lähetetään asiakkaalle viesti ”200 – OK”, tai virheen sattuessa ”500 – Internal Server Error”.
5. Vastaukset muutetaan ennen lähetystä JSON-merkkijonomuotoiseksi.

MySQL-kyselyjä suorittavat funktiot ovat asynkronisia, joten ne eivät estä pääsäikeen suoritusta. Säie on valmiina kuuntelemaan uusia HTTP-kutsuja kyselyn aikana.

Palvelimen JavaScript-kielinen ohjelmalistaus on esitetty liitteessä 1 ja asiakaslaitteen PHP-ohjelmalistaukset liitteessä 2.

Tietokantahaut

Tietokantahakuja varten luotiin kaksi erillistä funktiota. Toinen hakee tietokannasta viimeisen mittausindeksin ja toinen kirjoittaa tietokantaan uusia rivejä. Liitteessä 1 on esimerkkinä tietokantahaun suorittava funktio kokonaisuudessaan.

haeViimeinen(kyselyParametrit, viimMittaus)

Funktio hakee tietokannasta viimeisen mittausrivin numeron. Funktiolle välitetään parametrina merkkijono, joka sisältää tietokannan sekä taulun nimet. Paluuarvona palautetaan mittausrivin numero.

Funktion toiminta on seuraavanlainen:

1. Katkaistaan parametrina saatu merkkijono katkaisumerkin kohdalta. Tästä saadaan tietokannan sekä taulun nimi omaksi merkkijonokseen (parametrit[0] ja parametrit[1]).
2. Luodaan tietokantayhteys mysql.createConnection-metodin avulla. Metodille määritellään seuraavat parametrit:

   host : 'localhost',
   port : 3306,
   database: parametrit[1],
   user : ’MySQLkäyttäjätunnus’,
   password : salasana.
3. Suoritetaan tietokantakysely. MySQL-kysely on seuraavanlainen:
   SELECT mittaus from '+ parametrit[0] +' ORDER BY mittaus DESC LIMIT
   parametrit[0] sisältää taulun nimen, johon haku suoritetaan.
4. Mikäli kyselyssä ei tapahtunut virhettä, palautetaan kyselystä tullut arvo. Virheen sattuessa asetetaan palautusarvoksi numero 0.
5. Lopuksi puretaan luotu tietokantayhteys.

kirjoitaUudet(jsonObjekti, status)

Funktio kirjoittaa tietokantaan mittaustuloksia sisältäviä rivejä. Funktiolle välitetään parametrina JSON-paketti. Funktio palauttaa arvonaan tiedon, siitä onnistuiko tietokantaan kirjoittaminen.

1. Luodaan tietokantayhteys *mysql.createConnection*-metodin avulla. Metodille määritellään seuraavat parametrit:
   host : 'localhost',
   port : 3306,
   database: jsonObjekti.tietokanta,
   user : ’MySQLKäyttäjätunnus’,
   password : salasana.
2. Käydään mittausrivit sisältävä taulukko läpi for-silmukan avulla. Jokaisella silmukan kierroksella luetaan taulukon sisältä rivin tiedot ja koostetaan niistä MySQL-kyselyrivi.
3. Mikäli kysely onnistui, asetetaan palautusarvon ”status” sisällöksi ”ok”. Virheen sattuessa palautusarvoksi asetetaan ”nok”.
4. Lopuksi katkaistaan tietokantayhteys.

Esimerkki tietokantakyselystä on esitetty liitteessä 2.

2.7 Virhetilanteiden käsittely

Tässä luvussa on kuvattu todennäköisimpien virhetilanteiden havaitsemista ja virhetilanteiden käsittelyä.

Virhetilanteiden käsittely rakennettiin siten, että mikäli asiakaslaitteessa tapahtuu virhetilanne, ohjelman suoritus lopetetaan hallitusti. Palvelimelle ei tarvitse välittää mitään tietoa ongelmasta ohjelmiston rakenteen vuoksi.

Mikäli palvelimella tapahtuu virhetilanne, palvelin lähettää HTTP-statuskoodeja hyväksikäyttäen ilmoituksen virheestä asiakkaalle. Eri status-koodien merkitykset on selitetty W3Schoolsin verkkosivuilla (8).

HTTP-statuskoodi otetaan vastaan asiakaslaitteessa curl.getinfo(CURLINFO_HTTP_CODE)-metodissa. Mikäli koodin numero on

jokin muu kuin ”200 – OK”, lopetetaan PHP-skriptin suoritus die()-metodin avulla.

Tietokannan lukeminen asiakaslaitteelta (PHP)

Tutkitaan, palauttiko kysely tyhjän rivin if-else-rakenteen avulla.

```
if ($query_result){
 //Jos haku palautti tuloksen, käsitellään ja palautetaan tulos.
}
else{
 //Jos tulos on tyhjä ,lopetetaan koodin suoritus die() metodin avulla.
}
```

Yhteysvirheet asiakaslaitteella (PHP)

Käytetään hyväksi curl_getinfo()-metodia. Mikäli metodi palauttaa virheilmoituksen, ohjelman suoritus lopetetaan die()-metodin avulla.

JSON-paketin purkaminen palvelimella (JavaScript)

JSON-paketti puretaan palvelimella try-catch-rakenteessa json.parse-metodia käyttäen. Mikäli paketin purkamisessa ilmenee virhe, palautetaan asiakkaalle virhekoodi ”500 – Internal Server Error”.

Palvelimen tietokantahaut (JavaScript)

Mikäli mysql.createConnection()-metodi palauttaa virheilmoituksen, tietokantakyselyjä suorittava funktio palauttaa palvelinta ylläpitävälle säikeelle tiedon kirjoituksen onnistumisesta.

Virheen tapahtuessa asiakkaalle palautetaan koodi ”500 – Internal Server Error”. Tästä poikkeuksena on viimeisen mittausindeksin hakeva funktio, joka palauttaa asiakkaalle asti arvon 0.

Tiedonsiirron onnistuminen

Tiedonsiirron onnistuminen tarkastetaan JSON-paketin purkamisen onnistumisen avulla. Mikäli JSON-paketin purkavassa funktiossa tapahtuu virhe, funktio

palauttaa pääohjelmalle arvon ”null”. Virhetilanteessa palvelin palauttaa asiakkaalle merkkijonon ”jsonparser error”.

3 GOOGLE CLOUD MESSAGING FOR ANDROID

Google Cloud Messaging for Android on Googlen tarjoama palvelu, joka mahdollistaa viestien lähettämisen omalta palvelimelta Googlen pilvipalvelun kautta Android-laitteille. Palvelu mahdollistaa myös viestien lähettämisen asiakaslaitteelta palvelimen suuntaan. Palvelun kuvaus sekä ohjelmointirajapinnan dokumentaatio, johon tämä työ perustuu, löytyvät Android-kehitysyhteisön www-sivuilta (9).

Tässä työssä ei käsitelty lainkaan asiakaslaitteelta GCM-palvelimen kautta omalle palvelimelle lähetettäviä viestejä, joten selostuksessa on keskitytty ainoastaan palvelimelta asiakaslaitteelle lähetettävien push-tyyppisten viestien lähetykseen.

3.1 Järjestelmän pakolliset komponentit

Järjestelmän kuuluu vähintään kolme eri osapuolta.

- Käyttäjän Android-laitteeseen asennettu asiakassovellus.
    - Vähintään Androidin versio 2.2. Mikäli Androidin versio on vanhempi kuin 4.0.4, täytyy laitteen olla kirjautunut johonkin Googlen palveluun.
- Googlen ylläpitämä GCM-palvelin.
    - Ottaa vastaan viestejä omalta kolmannen osapuolen palvelimelta ja välittää ne asiakkaalle.
- Kolmannen osapuolen palvelin.
    - Itse tehty palvelin, joka lähettää viestejä asiakassovellukselle GCM-palvelinten kautta.

Palvelun käytössä tarvittavat tunnukset ja tunnistautumisnumerosarjat

- Sender ID
    - Projektin numero. GCM-palvelu tunnistaa palvelimen, jolla on oikeus lähettää viestejä palvelun kautta.

- Application ID
    - Asiakassovellus tunnistautuu tämän avulla palvelun käyttäjäksi.
- Registration ID
    - GCM-palvelin antaa tämän tunnuksen asiakaslaitteelle. Tunnus on jokaiselle asiakaslaitteelle yksilöllinen. Palvelin lähettää tunnuksen avulla viestit määrätylle asiakkaalle.
- Google User Account
    - Mikäli Android-ohjelmistoversio on vanhempi kuin 4.0.4, täytyy laitteen olla kirjautunut johonkin Googlen palveluun.
- Sender Auth Token
    - Palvelin tunnistautuu Googlen palvelun käyttäjäksi tämän tunnuksen avulla.

3.1.1 GCM-palvelin

GCM-palvelua voidaan käyttää kahdella eri tavalla: yksisuuntaisesti lähettämällä palvelimelta push-viestejä Android-laitteelle GCM-palvelun kautta tai kaksisuuntaisesti, jolloin Android-laite voi lähettää viestin palvelun kautta takaisin omalle palvelimelle.

Google tarjoaa käytettäväksi kaksi erityyppistä palvelinratkaisua: HTTP- ja CCS-palvelimen, joista ainoastaan CCS-palvelimella voidaan käyttää kaksisuuntaista viestien lähetystapaa.

Palvelimet eroavat toisistaan seuraavasti.

- Viestien lähetyssuunta
    - GCM HTTP – Yksisuuntainen
    - CCS – Molempiin suuntiin
- Asynkroniset viestit
    - GCM HTTP – Ei. Viestit lähetetään HTTP-post-pyyntöinä, jolloin lähettäjä odottaa vastauksen, ennen kuin voidaan lähettää uusi viesti.
    - CCS – Kyllä. CCS-palvelin käyttää yhteytenä pysyvää XMPP-yhteyttä, joka käyttää asynkronista lähetystapaa

- JSON
    - GCM HTTP – JSON-viestit kulkevat HTTP-post-pyynnön mukana.
    - CSS – JSON-viestit on pakattu XMPP-viestien sisälle.

3.1.2 Android-sovellus

Android-sovelluksessa käytetään GoogleCloudMessaging-ohjelmointirajapintaa, jonka avulla suoritetaan tiedonsiirto GCM-palvelimen ja Android-laitteen välillä.

Sovelluksen tekemistä varten täytyy projekti asettaa käyttämään Google Play Services SDK:ta. Google Play Services on ohjelmakirjasto, jonka avulla Android-ympäristössä voidaan ottaa käyttöön Googlen tarjoamia palveluita.

Ohjelmointirajapinnan dokumentaatio sekä ohje projektin luomiseksi on esitetty virallisella Android-kehitykseen keskittyneellä developer.android.com-sivustolla (9).

Android-manifest
Sovellukselle täytyy määritellä seuraavat luvat sovelluksen manifest-tiedostossa:

- com.google.android.c2dm.permission.RECEIVE
- android.permission.INTERNET
- android.permission.GET_ACCOUNTS
- android.permission.WAKE_LOCK
- applicationPackage + ".permission.C2D_MESSAGE"
- com.google.android.c2dm.intent.RECEIVE
- com.google.android.c2dm.SEND.

Sovelluksen manifest-tiedostoon täytyy lisäksi määritellä BroadcastReceiver. Android-sovelluksen BroadcastReceiver-luokka kuuntelee järjestelmän sisällä sovellusten välillä lähetettyjä viestejä. Yleisimmin GCM-sovelluksessa käytetään WakefulBroadcastReceiver-luokkaa, joka vastaanottaa laitteen herätystapahtuman sekä estää laitteen prosessorin menemisen lepotilaan sillä aikaa, kun IntentService suorittaa itse viestin käsittelyn. Järjestelmä käyttää Android-

luokan PowerManager.WakeLock-mekanismia pitääkseen järjestelmän toiminnassa. Mikäli sovellus käyttää WakeLock-mekanismia, täytyy manifest-tiedostoon lisätä lupa sen käyttöön.

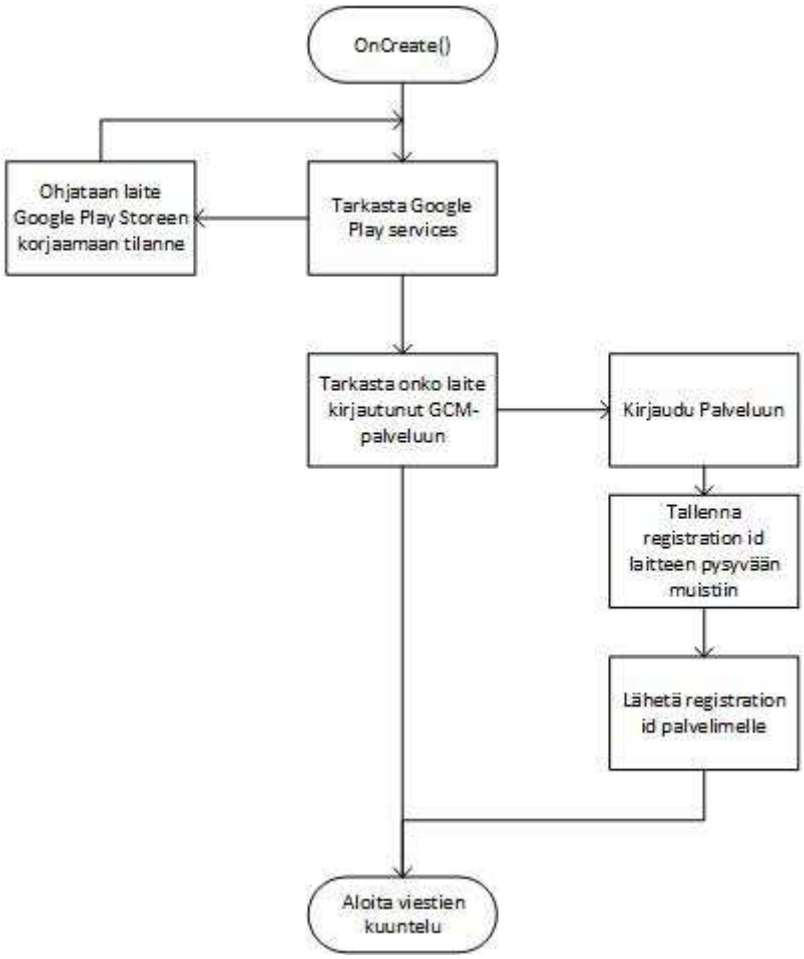
Lisäksi manifest-tiedostossa täytyy määritellä Android-ohjelmiston viimeiseksi sallituksi versioksi (minSdkVersion) vähintään 8, koska GCM-palvelu ei toimi sitä vanhemmissa versioissa.

Sovelluksen toiminta

Sovelluksen laadinnassa tärkeimmät kirjoitettavat toiminnot ovat seuraavat:

- Tarkastetaan, täyttääkö laite Google Play Servicen vaatimukset.
- Tarkastetaan, onko laite ennestään kirjautunut GCM-palvelun käyttäjäksi.
    - Laitteen pysyvään muistiin täytyy tallentaa tieto onnistuneesta rekisteröinnistä palveluun. Tässä vaiheessa on hyvä tallentaa registration_Id laitteeseen.
- Kirjaudutaan GCM-palveluun käyttäjäksi.
    - GoogleCloudMessaging register()-metodi palauttaa registration_Id:n GCM-palvelimelta.
- Lähetetään registration_Id omalle palvelimelle.
    - Mikäli käytössä on ainoastaan yksisuuntaiset viestit, joudutaan käyttämään jotain muuta kuin GCM-palvelun tarjoamaa reittiä tiedon välitykseen. Tavallisimmin suoritetaan suora HTTP-kysely palvelimelle.
- Odotetaan viestejä palvelimelta ja käsitellään ne.
    - Mikäli laite on idle-tilassa sekä delay_while_idle-parametri lähetyksessä on asetettu pois päältä, laite herätetään käsittelemään viestin sisältö.

Sovelluksen rekisteröinti GCM-palveluun sekä omalle palvelimelle on tarkoituksenmukaista suorittaa mainActivityn onCreate()-metodissa, mikäli verkkopalvelu on sovelluksen toiminnan kannalta kriittinen osa. Laitteen rekisteröinnin tapahtumakaavio on esitetty kuvassa 5.



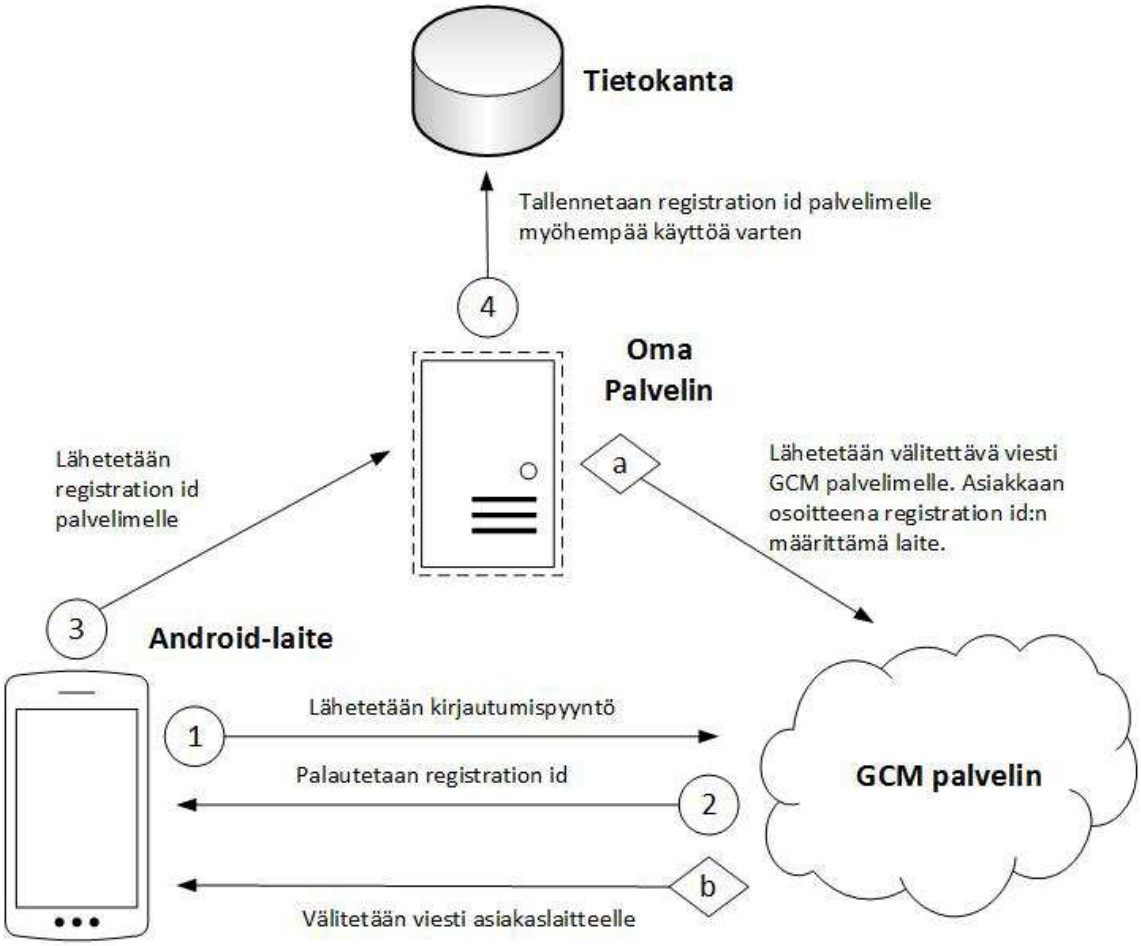
KUVA 5. Asiakaslaitteen rekisteröinti GCM-palveluun Android-sovelluksen onCreate()-metodissa.

3.1.3 Oma palvelin

Oman palvelimen tehtävä on säilöä laitteiden registration_Id-numerot, lähettää viestejä niiden perusteella asiakkaille sekä kuunnella verkosta tulevia HTTP-kyselyitä. Mikäli käytössä on kaksissuuntainen GCM-palvelu, täytyy palvelimen myös pystyä vastaanottamaan viestejä, jotka käyttävät XMPP-protokollaa.

3.2 Järjestelmän toiminta

Kuvassa 6 on kuvattu järjestelmän toiminnan periaate.



KUVA 6. GCM-palvelun toimintaperiaate.

3.2.1 Kirjautuminen käyttäjäksi

Asiakassovelluksen kirjautuminen palvelun käyttäjäksi tapahtuu seuraavasti. Numeroidut kohdat viittaavat kuvaan 6.

1. Asiakas lähettää Sender ID -tunnuksen GCM-palvelimelle kirjautumista varten.
2. Onnistuneen kirjautumisen jälkeen GCM-palvelu lähettää registration_Id-tunnuksen asiakaslaitteelle.
3. Asiakas lähettää registration_Id-tunnuksen palvelimelle.
4. Palvelin tallentaa asiakkaan lähettämän tunnuksen palvelimelle myöhempää käyttöä varten.

3.2.2 Viestien lähetys

Push-viestien lähetys palvelimelta asiakkaalle tapahtuu seuraavasti. Kohdat a. ja b. viittaavat kuvaan 6.

a. Palvelin lähettää viestin GCM-palvelimelle. Asiakaslaitteen osoite määrittyy registration_Id -tunnuksen perusteella.
b. GCM-palvelin tallentaa saapuneen viestin ja yrittää lähettää sitä asiakaslaitteelle. Mikäli viestiä ei jostain syystä voida välittää, lähetystä yritetään uudelleen niin kauan kuin viestin parametreissa on määritelty tai viesti saadaan onnistuneesti välitetyksi.

3.3 Lähetettävien viestien tyypit

Palvelussa voidaan välittää tietosisältöä asiakkaalle kahdella eri tavalla. Joko lähetetään ilmoitustyyppinen synkronointiviesti palvelimelta asiakkaalle tai viestin mukaan voidaan upottaa välitettävä tietosisältö, mikäli tietomäärä on vähäinen.

3.3.1 Synkronointiviesti (Collapsible messages)

Synkronointiviestiä käytettäessä lähetetään asiakaslaitteelle ilmoitus siitä, että palvelimella on saatavilla uutta dataa. Tämän jälkeen asiakaslaite ottaa suoran yhteyden palvelimelle ja lataa tietosisällön laitteeseen.

Nämä viestit ovat collapsible-tyyppisiä, mikä tarkoittaa sitä, että edellinen viesti poistetaan uuden viestin saapuessa GCM-palvelimelle, mikäli viestiä ei jostain syystä pystytty toimittamaan asiakkaalle määritellyn aikarajan sisällä. Ainoastaan viimeinen viesti välitetään asiakaslaitteelle.

Tätä tapaa käytetään silloin, kun siirrettävä tietomäärä on suuri, tai silloin, kun uusi tieto korvaa aikaisemmin lähetetyn tiedon. Esimerkiksi mikäli palvelimella on ladattavissa suurikokoinen tiedosto (kuva, video jne.), viestin kokorajoituksen vuoksi tiedostoa ei voida lähettää GCM-viestin mukana, vaan lähetetään ainoastaan ilmoitus siitä, että palvelimella on saatavissa uutta sisältöä. Toinen esimerkkitapaus voisi olla sellainen, että palvelin ilmoittaa saatavilla olevasta uudesta sähköpostista. Tällöin riittää yksi ilmoitus (uusin) siitä, että uutta sähkö-

postia on saatavilla, jotta asiakas osaa ladata kerralla kaikki saapuneet sähköpostit.

3.3.2 Viesti jossa tietosisältö mukana (Non-collapsible messages)

Mikäli välitettävä tietomäärä on vähäinen, tietosisältö voidaan liittää osaksi lähetettävää GCM-viestiä. Välitettävän tiedon enimmäismäärä on rajoitettu 4 kb:n kokoiseksi. Tämän tyyppiset viestit ovat non-collapsible-tyyppisiä, joka tarkoittaa sitä, että kaikki palvelimelta lähetetyt viestit välitetään asiakkaalle.

GCM-palvelu tallentaa enimmillään 100 kappaletta asiakaslaitteelle toimittamattomia viestejä. Kun viestipuskuri on täynnä palvelu hylkää uudet viestit ja generoi viestin, jossa asiakkaalle kerrotaan kuinka monta viestiä on hylätty. Asiakassovelluksen ja palvelimen välille tulee rakentaa tällaista tilannetta varten synkronointimekanismi puuttuvan tietosisällön ajan tasalle saamiseksi.

3.3.3 HTTP-viestien rakenne

GCM-viesti lähetetään HTTP-post pyyntönä Googlen palvelimelle, josta se välitetään asiakaslaitteille GCM-palvelun kautta. Palvelimen osoite jonne viesti lähetetään: https://android.googleapis.com/gcm/send.

Viesti koostuu kahdesta osasta: HTTP header- sekä body-osasta.

Header-osiossa täytyy määritellä vähintään seuraavat osat:

- Authorization: key=YOUR_API_KEY
    - API _KEY on palvelua käyttöön otettaessa talteen otettava SENDER ID-tunnus
- Content-type
    - Käytettäessä JSON viestiä
        - Content-Type: application/json
    - Tavallista tekstiä käytettäessä
        - application/x-www-form-urlencoded;charset=UTF-8.

Esimerkki JSON-viestin headerista.

```
Content-Type:application/json
Authorization:key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA

{
  "registration_ids" : ["APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx..."],
  "data" : {
    ...
  },
}
```

HTTP-body-osiossa määritetään viestin sisältö. Seuraavassa on lueteltu mahdolliset datakentät käytettäessä JSON-viestejä. Plain-text-viestiä käytettäessä voidaan käyttää kaikkia muita datakenttiä paitsi notification_key-kenttää.

registration_ids

- Luettelo vastaanottajista. Sama viesti voidaan lähettää kerrallaan 1–1000 asiakkaalle.
- Pakollinen.

notification_key

- Kokoelma registration_Id-tunnuksista, jonka avulla voidaan sitoa yksi rekisteröitynyt käyttäjä useampaan rekisteröityneeseen laitteeseen.
- Käytetään, mikäli käyttäjä on rekisteröitynyt useammalla eri laitteella palveluun ja halutaan lähettää sama viesti jokaiselle laitteelle yhtä aikaa.
- Enimmillään 10 laitetta yhteen kokoelmaan.
- Ei pakollinen.

collapse_key

- Mielivaltainen merkkijono, joka ryhmittelee samasta aiheesta lähetetyt viestit.
- Mikäli tämä parametri asetetaan päälle, asiakkaalle välitettävistä epäonnistuneista samaan ryhmään kuuluvista synkronointiviesteistä lähetetään ainoastaan viimeinen sitten, kun lähetys taas onnistuu.

- Kerrallaan palvelussa voidaan käyttää korkeintaan neljää eri collapse_key-parametria.
- Ei pakollinen.

delay_while_idle

- Mikäli asetettu päälle, kentän totuusarvo ilmoittaa palvelulle, että viestiä ei tarvitse lähettää heti, jos asiakaslaite on lepotilassa (idle). Ainoastaan viimeinen viesti jokaisesta collapse_key-ryhmästä välitetään asiakkaalle.
- Ei pakollinen.

data

- JSON-objekti, jonka nimi-arvo-kentät edustavat viestin datasisältöä.
- Datakenttien lukumäärälle ei ole rajoitusta, mutta viestin kokonaiskoko ei voi olla suurempi kuin 4 kb.
- Value voi olla minkä tahansa tyyppinen JSON-objekti, mutta on suositeltavaa käyttää ainoastaan merkkijonoa, koska GCM-palvelu muuttaa objektin joka tapauksessa merkkijonoksi ennen välittämistä asiakkaalle. Mikäli tahdotaan välittää jonkin muun tyyppistä tietoa, tulee muunnos suorittaa itse.
- Ei pakollinen.

time_to_live

- Määrittää sekunneissa, kuinka pitkän ajan GCM-palvelu tallentaa viestin, mikäli sitä ei pystytty toimittamaan.
- Ei pakollinen, oletusarvo: 4 viikkoa.

restricted_package_name

- Merkkijono, joka sisältää Android-sovelluksen nimen (package name).
- Mikäli asetettu, viestit välitetään ainoastaan niille laitteille, jonka registration_Id tunnukseen on yhdistetty kyseinen nimi.
- Ei pakollinen.

dry_run

- Kehityskäyttöön, mikäli tämä lippu on asetettu päälle, sovelluskehittäjä voi testata ohjelman toimintaa ilman, että lähettää viestiä tosiasiassa palvelimelle.
- Ei pakollinen, oletusarvo: false

4 OMA SOVELLUS

Työn aikana aloitettiin oman sovelluksen suunnittelu. Lopullisena tavoitteena on Android-sovellus, joka esittää sekä numeraalisessa että graafisessa muodossa automaattisesti päivittyvää mittausdataa.

Työn aikana saatiin valmiiksi tiedonsiirron kannalta tarvittavat osat:

- laitteen rekisteröinti GCM-palveluun sekä rekisteröintitietojen tallennus asiakaslaitteeseen
- laitteen rekisteröinti omalle palvelimelle sekä rekisteröintitietojen tallennus palvelimelle
- palvelimen tietokantaan päivittyvien antureiden haku tietokannasta Android-sovelluksen näytölle
- sovelluksen päivitettäväksi halutun anturin tallennus yhdessä käyttäjätietojen kanssa palvelimelle.
- anturin mittaustulosten haku tietokannasta sekä niiden lähetys ja vastaanotto Android-laitteelle GCM-palvelun kautta.

Lisäksi Android-sovellukseen lisättiin ohjelman toiminnan esittelyä varten laitteen näytölle pudotettava widgetti, johon tulostetaan palvelimelta saapunut viesti.

Sovelluksen tekeminen aloitettiin kopioimalla hyvin yksinkertainen esimerkki palvelinohjelmasta omalle palvelimelle. Tässä vaiheessa palvelin ainoastaan lähetti yksinkertaisia viestejä GCM-palvelun kautta asiakkaalle sekä tulosti konsoliin tekstiä.

Android-sovelluksen tekeminen aloitettiin tutustumalla Googlen tarjoamaan valmiiseen esimerkkiratkaisuun sovelluksesta. Tämän avulla saatiin todennettua palvelinohjelman rungon ja tiedonsiirron toimivuus palvelimelta asiakkaalle toimivaksi. Tämän jälkeen ohjelmistoa alettiin rakentaa aina yksi asiakas-palvelin-toiminto kerrallaan, sekä palvelimen että Android-sovelluksen osa aina yhdellä kertaa valmiiksi.

4.1 Palvelin

Palvelinpäässä lisättiin työn aikaisemmassa vaiheessa tehtyyn Node.js-palvelimeen kuuntelijat Android-sovelluksen rekisteröintiä ja päivitettävän anturin valintaa varten. Lisäksi palvelimelle tehtiin ajastimella toimiva, GCM-palvelun kautta viestejä asiakkaalle lähettävä toiminto.

Palvelimen ohjelmalistaus kokonaisuudessaan on esitetty liitteessä 1.

4.1.1 Käyttäjätietojen rekisteröinti

Käyttäjätietoja varten luotiin oma tietokanta, johon luodaan jokaista käyttäjää varten oma taulu. Tauluun tallennetaan käyttäjänimi, Android-laitteen registration_Id sekä tiedot päivitettäviksi valituista antureista.

Käyttäjätietotaulun nimeksi lasketaan registration_Id:stä ja käyttäjänimestä yhdistetystä merkkijonoista laskettu SHA-1-tiiviste. Tämä tehdään sen vuoksi, että jokaisen taulun nimen täytyy vastata yhtä uniikkia laitetta. Pelkkä registration_Id-taulun nimenä olisi ollut muuten riittävä ratkaisu, mutta MySQL-taulun nimen pituus on rajoitettu 64 merkin pituiseksi ja registration_Id:n pituus voi Googlen mukaan olla jopa 4 kt.

Palvelimen toiminnot toimivat samalla periaatteella kuin luvussa 2, tietokannan siirto palvelimelle, on kuvattu.

- Käyttäjän rekisteröinti
  - Luo tietokantaan taulun, jolle annetaan nimeksi parametrina saatu SHA-1-tiiviste. Tallentaa tauluun käyttäjänimen sekä registration_Id:n.
- Käyttäjänimen vaihtaminen
  - Vaihtaa taulun nimen parametrina saadun uuden sha-tiivisteen mukaiseksi sekä tauluun tallennetun käyttäjänimen vastaamaan uutta nimeä. Muiden kenttien arvot pysyvät muuttumattomina.
- Anturilistan palautus asiakkaalle
  - Hakee palvelimelta kaikkien mittaustietokantojen kaikkien taulujen kaikkien antureiden tiedot sekä lähettää ne vastauksena asiakaslaitteelle.

- Käyttäjän tilauksen tallennus
    - Tallentaa käyttäjätietokantaan käyttäjän päivitettäväksi valitsemien antureiden tiedot.

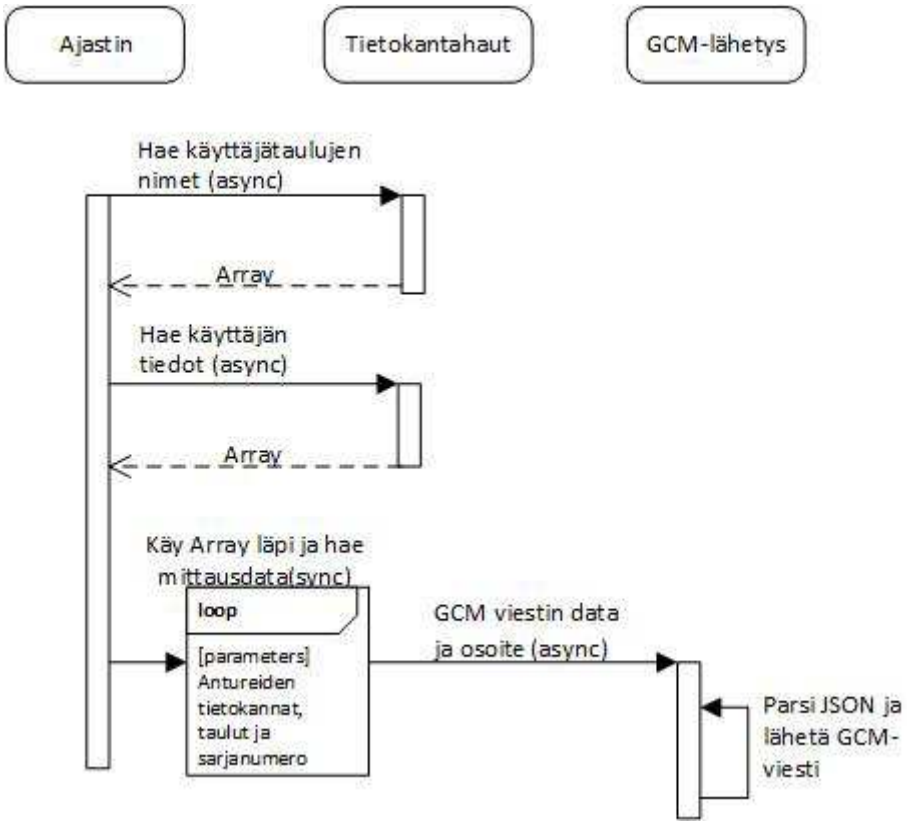
4.1.2 GCM-viestien lähetys

GCM-viestien lähetystä varten luotiin ajastin, joka suorittaa ennalta määrätyin väliajoin tietokannasta käyttäjätietotaulujen lukemisen, taulujen perusteella anturidatan hakemisen tietokannoista sekä tietojen lähetyksen GCM-palvelun kautta asiakaslaitteille.

Lähetystapahtuman toiminta on seuraavanlainen.

1. Haetaan käyttäjätietokannasta kaikkien taulujen nimet.
2. Haetaan edellisessä vaiheessa saaduista tauluista käyttäjätiedot eli tilatun anturin tietokanta, taulu ja sarjanumero sekä asiakaslaitteen registration_Id.
3. Käydään läpi edellisen tietokantahaun palauttama taulukko.
    a. Haetaan jokaisen anturin mittausdata. Tämä haku on asynkroninen.
    b. Jokaisen tietokantahaun callback-funktiossa parsitaan asiakkaalle lähetettävä JSON-viesti sekä lähetetään se GCM-palveluun.

On huomattava, että tässä luodaan jokaista käyttäjää kohden oma säie tiedon lähetystä varten. Anturidatan lähetyksen sekvenssikaavio on esitetty kuvassa 7.



KUVA 7. Anturidatan lähetys asiakkaalle

Viestin lähetyksessä käytetyt ohjelmalistaukset on esitetty liitteessä 4. Timer.js sisältää pääohjelman, joka huolehtii tehtävän ajoituksesta sekä tietokantahakujen tekemisestä. Gcmserveri.js suorittaa itse lähetystapahtuman.

4.2 Android-sovellus

Työn aikana laadittu Android-sovellus jäi keskeneräiseksi. Valmiiksi saatiin ainoastaan toiminnot rekisteröintiä sekä GCM-viestien vastaanottamista varten. Suurimmaksi puutteeksi jäi vastaanotetun tiedon käsittely sekä esittäminen laitteen näytöllä.

Sovelluksen tärkeimmät toiminnot, jotka saatiin valmiiksi, ovat

- GCM-palveluun kirjautuminen
- omalle palvelimelle kirjautuminen
- luokka, joka hoitaa tallennuksen laitteen pysyvään muistiin, tallennettavia tietoja ovat esimerkiksi käyttäjätiedot

- palvelimen anturilistan haku ja esitys näytöllä vastaanotettavan datan valintaa varten
- valitun anturin tietojen lähetys palvelimelle
- luokka, joka lähettää asynkronisesti viestejä omalle palvelimelle käyttäen HTTP-yhteyttä.

Eclipsen generoima luokkakaavio sovelluksesta on esitetty liitteessä 5.

Projekti tehtiin ADT Bundle -kehitysympäristössä. ADT Bundle on Android-sovelluskehitystä varten paketoitu ohjelmointityökalukokoelma. ADT Bundle -pakettiin on sisällytetty mm. seuraavia työkaluja:

- Eclipse
- Android SDK
- Android-kehitystä varten hyödyllisiä apuohjelmia, mm. emulaattori, sisältävä paketti.

Projektia varten ladattiin Google Play Services SDK käyttäen työkalun SDK Manageria. Projektia luotaessa se asetettiin käyttämään Google Play Services -kirjastoa kopioimalla google-play-services_lib-tiedosto projektin kansioon sekä viittaamalla siihen projektin asetuksissa.

4.2.1 Rekisteröinti

Ennen kuin laite voi vastaanottaa GCM-viestejä palvelimelta, täytyy laite rekisteröidä sekä GCM-palveluun että omalle palvelimelle. Tätä vaihetta varten luotiin seuraavat toiminnot.

GCM-rekisteröinti
GCM-palveluun rekisteröinti tapahtuu mainActivityn onCreate()-metodissa.

- Tarkastetaan, onko laitteeseen asennettu Google Play Services
  - Mikäli palvelua ei ole asennettu, kehotetaan käyttäjää asentamaan se.
- Tarkastetaan, onko laitteen pysyvään muistiin kirjoitettu registration_Id -tunnusta.

    - Mikäli tunnus löytyy laitteesta, voidaan rekisteröintiprosessi lopettaa.
- Mikäli tunnusta ei löydy, kutsutaan asynkronista taustalla ajettavassa metodissa toimintoa, joka hakee uuden registration_Id -tunnuksen GCM-palvelimelta.

Omalle palvelimelle rekisteröinti

Laitetta omalle palvelimelle rekisteröidessä palvelimelle lähetetään registration_id-tunnus, käyttäjätunnus sekä näistä laskettu tiiviste. Tämän vuoksi rekisteröinti voidaan suorittaa vasta sen jälkeen, kun sovellus on kirjautunut GCM-palveluun ja käyttäjä on syöttänyt sovellukselle käyttäjänimensä.

Ohjelma kehottaa käyttäjää antamaan nimensä editTextBoxiin, josta napin painallus lukee syötetyn nimen. Tämän jälkeen kutsutaan metodia, joka lukee registration_Id -tunnuksen laitteen pysyvästä muistista. Merkkijonot yhdistetään ja uusi syntynyt merkkijono välitetään metodille, joka palauttaa siitä lasketun SHA-1-tiivisteen. Käyttäjänimi sekä tiiviste tallennetaan samaan paikkaan, jossa registration_Id on jo ollut aiemmin tallennettuna.

Lopuksi kutsutaan itse tehdyn luokan metodia, joka lähettää tiedot palvelimelle. Luokka on esitelty jäljempänä luvussa 4.2.2.

Tallennus

Laitteen pysyvään muistiin tallennusta varten luotiin oma luokka. Luokkaan kirjoitettiin julkiset metodit, jotka tallentavat käyttäjätiedot laitteeseen, lukevat tiedot laitteesta sekä poistavat ne laitteesta. Lisäksi luokkaan luotiin metodi, joka laskee SHA-1-tiivisteen parametrina annetusta merkkijonosta.

4.2.2 HTTP-viestien lähetys palvelimelle

Sovelluksen rekisteröimistä omalle palvelimelle sekä päivitettäväksi tilattavan anturin valintaa varten luotiin oma AsyncTask-pohjainen luokka, AsyncJsonPost. Luotaessa luokasta instanssi annetaan sille parametrina merkkijono, jossa kerrotaan, minkälainen viesti palvelimelle halutaan lähettää. Palautusarvona saadaan palvelimen palauttama viesti. Palvelimen palauttama vastaus käsitellään lopuksi luokan interface-metodissa.

JSON-objektin rakennus

Luokkaan rakennettiin yksityinen metodi, joka käy else-if-rakenteessa läpi sille välitetyn merkkijonon. Siinä kerrotaan, minkä tyyppinen viesti palvelimelle lähetetään. Lauseissa rakennetaan jokaiseen tilanteeseen sopiva JSON-objekti.

Mahdolliset lähetettävät viestityypit ovat seuraavia:

- rekisteröidään käyttäjätunnus
- muutetaan käyttäjänimeä
- kysytään palvelimelta kaikkien olemassa olevien antureiden tiedot
- lähetetään palvelimelle niiden antureiden tiedot, joiden tilaa halutaan seurata.

Objekti välitetään taustaprosessina tapahtuvalle yksityiselle metodille postData(), joka suorittaa itse lähetystapahtuman. Palvelimelle luotiin vastaavasti näihin viesteihin vastaavat funktiot.

4.2.3 Anturilistan haku palvelimelta

Sovellukseen päätettiin luoda jo heti alkuvaiheessa toiminto, jolla voi valita anturit, joiden mittaustulokset päivittyvät laitteeseen GCM-palvelun kautta. Tätä varten sovellukseen luotiin lähetystapahtuma, joka kysyy palvelimelta antureiden tiedot. Antureiden tietojen esitystä varten luotiin oma CustomListView-widgetti, jossa alkioiden sisältönä ovat yhden anturin tiedot sekä CheckBox-widgetti. CheckBox-widgetin tilan (tosi / epätosi) perusteella lähetetään palvelimelle tiedot valituista antureista.

Anturin valitsemista varten luotiin uusi activity-luokan instanssi ja näytölle lisättiin kaksi nappia. Toinen käynnistää anturilistan haun palvelimelta ja toinen valinnan lähetyksen palvelimelle. Lisäksi näytölle lisättiin listView-widgetti antureiden näyttöä ja valintaa varten.

Kun käyttäjä painaa anturilistan haun aloittavaa nappia, luodaan instanssi AsyncJsonPostista, jossa luodaan JSON-objekti. Objektin sisältönä on ainoastaan pyyntö saatavilla olevista antureista. JSON-objekti muutetaan merkki-

jonoksi lähettämistä varten ja lähetetään palvelimelle käyttäen HTTP-post-pyyntöä.

Vastauksena saatu JSON-objekti parsitaan sitä varten luodussa metodissa. Metodissa puretaan JSON-paketti ja luodaan jokaisesta anturista uusi sensor-olio. Sensor-olio sisältää tiedot palvelimella olevista antureista. Oliot sijoitetaan antureita varten luodun SensorModel-luokan ArrayListin sisällöksi. Kun JSON-paketti on käyty kokonaan läpi, päivitetään näytölle listViewin alkioiden sisällöksi antureiden tiedot.

Kun käyttäjä valitsee listViewin listalta CheckBoxin tilan valituksi, muutetaan SensorModel-luokan taulukon sisältämän olion kohdalle vastaava tila.

4.2.4 Anturin valinta ja lähetys palvelimelle

Valintaruudun lähetysnappia painettaessa luodaan jälleen instanssi AsyncJsonPostista. Parametriksi annetaan pyyntö lähettää valittujen antureiden tiedot palvelimelle.

JSON-objekti luodaan hakemalla SensorModelin listasta niiden antureiden tiedot, joissa updateState-muuttuja on asetettu true-arvoon, ja lisäämällä antureiden tiedot lähetettävään objektiin. Objektiin lisätään myös lähettäjän käyttäjätiedot, jotta palvelin osaa vaihtaa tiedot oikeaa käyttäjää vastaaviksi.

4.2.5 GCM kuuntelija

GCM-viestejä kuunnellaan sovellukseen implementoidun GcmBroadcastReceiverin avulla. Luokka käyttää WakefulBroadcastReceiveriä, joka huolehtii vastaanottotapahtuman aikana siitä, että prosessori pysyy koko tapahtuman ajan hereillä.

Kun GcmBroadcastReceiver-luokan onReceive-metodia kutsutaan, se kutsuu metodia onHandleIntent, jossa käsitellään saapunut datapaketti. Kun viesti on käsitelty loppuun, ohjelma vapauttaa wakelockin ja laite päästetään takaisin lepotilaan.

Widgetti

Tämän työn puitteissa saapuvan viestin tarkempaa käsittelyä ei tehty. Saapunut viesti ainoastaan tulostetaan sellaisenaan demonstraatiota varten tehdyn widgetin tekstikenttään. Widgetti koostuu Androidin LinearLayout-pohjasta sekä sen päälle asetetusta textView-kentästä.

Saapunut viesti kopioidaan onHandleIntentin sisälle kirjoitetussa updateWidget-metodissa. TextViewin kirjoitukseen käytetään android.widget.RemoteViews.setTextViewText-metodia.

4.3 Sovelluksen teon aikana esiintyneet ongelmat

Työn aikana esiintyi muutamia ongelmia. Seuraavassa on esitetty eniten aikaa vienyt ongelma sekä ongelman ratkaisu.

Palvelimella joudutaan tekemään sellaisia tietokantahakuja, joissa tietokannasta haetaan ensin useampi tulos, ja sen jälkeen uusien kyselyiden parametreiksi annetaan edellisen kyselyn palauttamat tulokset. Nämä tulokset tulee yhdistää samaan taulukon taulukkoon.

Ongelmaksi muodostui se, että käytettäessä koko ajan asynkronisia callback-funktioita tietokantahakujen suorittamiseksi saadaan aikaan tilanne, jossa yhtäaikaisesti laukaistujen tietokantahakusäikeiden sekä sisäkkäisten callback-kutsujen määrä kasvaa hallitsemattomaksi. Tämän seurauksena ohjelmakoodin hallinta ja lukeminen menee mahdottomaksi. Tavallisen silmukan sisälläkään jälkimmäistä hakua ei voida tehdä, koska ei voida millään tietää, missä järjestyksessä haut valmistuvat. Silloin 1. ja 2. hakujen tulosten yhdistäminen voi mennä väärin.

Tämän ongelman kiertämiseksi jouduin kehittämään ratkaisun, jossa ensimmäisen asynkronisen haun tuottama lista käydään läpi synkronisen funktion sisällä olevassa silmukassa. Funktiossa lista käydään läpi siten, että listan ensimmäinen objekti välitetään parametriksi asynkroniselle tietokantahaku-funktiolle sekä samalla välitetty objekti poistetaan listan sisällöstä. Haun tulos sekä parametrina annettu hakuparametri lisätään uuteen listaan, jonka jälkeen palataan takai-

sin synkroniseen funktioon. Alkuperäinen lista käydään läpi samalla tavalla loppuun saakka, kunnes lopulta on valmiina uusi lista, jossa on sekä ensimmäisen että toisien kyselyiden tulokset yhdistettyinä.

5 YHTEENVETO

Työn tarkoituksena oli kehittää järjestelmä, jossa siirretään eri mittauspaikoilta kerättyä mittausdataa palvelimelle sekä välitetään data Google Gloud Messaging for Android -palvelun avulla asiakaslaitteille asiakkaan toivomista mittauspisteistä. Työn aikana tehtäviin toimintoihin kuului oman palvelimen ohjelmiston laatiminen, GCM-palvelun käyttöönotto sekä Android-sovelluksen suunnittelu. Työn tuloksena saatiin tehtyä toimiva mittausdatan kopiointijärjestelmä sekä mittausdatan välitys GCM-palvelun kautta asiakaslaitteelle.

Koska tässä työssä käsitellyssä datassa esiintyy datan luonteen vuoksi pitkiä luonnollisia aikaviiveitä (lämpötila, pulssien keskiarvoistus), datan välitön saatavuuskaan ei ole välttämätöntä. Tästä seikasta johtuen GCM-palvelun käyttö tämän typpisessä sovelluksessa on hieman vääränlainen ratkaisu. Tässä sovelluksessa järjestelmä, jossa asiakaslaite itse aktiivisena ollessaan pyytäisi haluamansa datan palvelimelta, olisi ehkä toimivampi kuin palvelimen lähettämä data. Koska dataa ei ole tarkoitus päivittää kuin enimmillään n. 1/2–1 tunnin välein, sovellukseen joudutaan joka tapauksessa rakentamaan mekanismi, jolla käyttäjä saa halutessaan päivitettyä senhetkisen datan tilan manuaalisesti palvelimelta.

GCM-palvelun käyttöä puoltava seikka on asiakaslaitteen pienempi akun kulutus. Koska GCM-viestejä kuunteleva palvelu on joka tapauksessa koko ajan aktiivisena, akun kulutuksen ei pitäisi lisääntyä juuri ollenkaan verrattuna tilanteeseen, jossa asiakas itse pyytää säännöllisin väliajoin palvelimelta uutta dataa.

GCM-palvelun käyttö sopii hyvin tilanteisiin, joissa datan välitys heti asiakkaan saataville on perusteltua, esimerkkeinä tästä mm. tekstiviesti- tai chat-tyyppiset sovellukset, pallopelien tulos- tai pörssikurssienseurantasovellus, hälytystapahtumat yms. käyttäjän välitöntä huomiota haluavat sovellukset.

Työn aikana vastaantulleita haasteita, jotka tässä tapauksessa voidaan lukea positiivisiksi oppimiskokemuksiksi, olivat uusien tekniikoiden opiskelu. JavaScriptistä tai Node.js-palvelinohjelmistosta minulla ei aikaisemmin ollut minkäänlaista aikaisempaa kokemusta, joten näissä riitti opiskeltavia asioita huomattavan paljon. Kolmas suurempi uusi kokonaisuus oli GCM-palvelu. Työn aikana sain hyvän käsityksen, mihin palvelua voi käyttää, ja työn aikana tapahtuneen perehtymisen perusteella kykenen laatimaan huomattavasti helpommin myös uuden, toisentyyppisen GCM-palvelua käyttävän sovelluksen.

Projektin tekeminen tämän työn jälkeen jatkuu viimeistelemällä Android-sovelluksen datan käsittely sekä käyttöliittymä. Kun sovellus on ensimmäisen kerran valmis, tehtävälistalla on kokeilla myös viestien lähetystä asiakaslaitteelta pilven kautta palvelimelle.

LÄHTEET

1. JavaScript. Mozilla Developer Network 2014. Saatavissa: https://developer.mozilla.org/en-US/docs/Web/JavaScript. Hakupäivä 26.4.2014.

2. Node.js. Joyent,Inc. Saatavissa: http://nodejs.org/ Hakupäivä 26.4.2010.

3. MySQL Reference Manuals. Oracle 2014. Saatavissa: http://dev.mysql.com/doc/. Hakupäivä 26.4.2014.

4. MySQL materiaali. OAMK. Saatavissa: http://www.ratol.fi/opensource/mysql/. Hakupäivä 26.4.2014.

5. Introducing JSON. JSON Community Group 2014. Saatavissa: http://www.json.org/. Hakupäivä 26.4.2014.

6. PHP Manual. PHP Documentation Group 2014. Saatavissa: http://www.php.net/manual/en/. Hakupäivä 26.4.2014.

7. Service Name and Transport Protocol Port Number Registry. Internet Assigned Numbers Authority 2014. Saatavissa: http://www.iana.org/assignments/service-names-port-numbers/service-names-port-numbers.xhtml. Hakupäivä 26.4.2014.

8. HTTP Status Messages. Refsnes Data 2014. Saatavissa: http://www.w3schools.com/tags/ref_httpmessages.asp. Hakupäivä 24.4.2014.

9. Google Cloud Messaging for Android. Android Development Community 2014. Saatavissa: http://developer.android.com/google/gcm/index.html Hakupäivä 26.4.2014.

LIITTEET

Liite 1 Palvelimen ohjelmalistaus

Liite 2 Esimerkki palvelimen Node.js tietokantakyselystä.

Liite 3 Mittauspaikan php-ohjelmalistaukset

Liite 4 GCM-viestien lähetyksen ohjelmalistaukset

Liite 5 Android-sovelluksen luokkakaavio

```
var http = require('http');
var tietokantahaut = require('./tkhaut.js');
var haeSn = require('./testihaku_sn.js');

var tietokanta='';
var metodi='';
var taulu='';
var taulut2=new Array;
var viimeisin=999;

var gcmfunktiot = require('./rmservu.js');
var gcmServer = require('./rmservu.js');

var server = http.createServer(function(request, response){

if( request.method ==='POST'){
        // Luetaan viestin data
    request.addListener("data",function(chunk){
    request.content='';
    request.content += chunk;
    });

    request.addListener("end",function(){
    var yhteyskerta =0;


    var obj = puraJSON(request.content); //Kutsutaan JSON paketin purkavaa funktiota

      //Mikäli meillä on validi JSON-stringi käsitellään se, muutoin palautetaan virhekoodi 500
    if(obj){
        metodi=obj.metodi;
        tietokanta=obj.tietokanta;
        taulu=obj.taulu;
        var kyselyParametrit = taulu+';'+tietokanta;

        if(metodi =='kysely'){
          // Kutsutaan funktiota, joka hakee tietokannasta viimesimmän "mittaus"-rivin numeron
            tietokantahaut.haeViimeinen(kyselyParametrit,function(viimeisin){
              var vastaus ={'mittaus':viimeisin};
              var vastausString = JSON.stringify(vastaus);
              response.writeHead(200,{'Content-Type':'application/json'});
              response.write(JSON.stringify(vastaus));
              response.end();
            });
        }
        elseif(metodi =='datalahetys'){ // Kirjoitetaan tietokantaan uutta dataa
            tietokantahaut.kirjoitaUudet(obj,function(kirjoituksenStatus){
            if(kirjoituksenStatus =='ok'){
                response.writeHead(200,{'Content-Type':'application/json'});
                response.write(JSON.stringify(kirjoituksenStatus));
                response.end()
            }
            else{
                response.writeHead(500,{'Content-Type':'application/json'});
                response.write(JSON.stringify(kirjoituksenStatus));
                response.end()
                }
            });
        }
```

```
elseif(metodi =='register'){ // Rekisteröidään käyttäjä
        tietokantahaut.regIdTallennus(obj,function(kirjoituksenStatus){
        if(kirjoituksenStatus =='ok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
            }
        elseif( kirjoituksenStatus =='nok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
            }
        });
}

elseif(metodi =='changeUserTableName'){ // Muutetaan käyttäjänimeä
        tietokantahaut.vaihdaKayttajaNimi(obj,function(kirjoituksenStatus){
        if(kirjoituksenStatus =='ok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
        }
        elseif( kirjoituksenStatus =='nok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
            }
        });
}

elseif(metodi =='tallennaTaulut'){ // Kirjoitetaan käyttäjätietoihin dataa
tietokantahaut.muutaTilatutAnturit(obj,function(kirjoituksenStatus){
        if(kirjoituksenStatus =='ok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
            }
        elseif( kirjoituksenStatus =='nok'){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(kirjoituksenStatus));
            response.end()
            }
        });
}

elseif(metodi =='haeTaulut'){
//Haetaan mysql:stä anturidataa sisältävät tietokannat ja niiden taulut
tietokantahaut.haeTaulut("getTables",obj,function(taulut){
haeSn.haeSn(taulut,function(err,palautusJson){
        if(err){
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify("Tietokantavirhe"));
            response.end();
        }
        else{
            response.writeHead(200,{'Content-Type':'application/json'});
            response.write(JSON.stringify(palautusJson));
            response.end();
            }
```

```
                    });
                });
            }
    }

    else{
            response.writeHead(500,{'Content-Type':'application/json'});
            response.write(JSON.stringify('jsonparser error'));
            response.end()
            }
    });
}

else{
       response.writeHead(301,{Location:'http://www.google.com'});
       response.write(request.connection.remoteAddress);
       response.end();
       }
});


server.listen(61280);

// Funktio purkaa JSON-paketin
function puraJSON(paketti){
try{
     var json = JSON.parse(paketti);
    return json;
    }catch(e){
       return null;
    }
}
```

```
var mysql = require('mysql');

// Funktio hakee tietokannasta viimeisen indeksin, tietokanta ja taulu
//välitetään funktiolle arrayn kyselyParametrit sisällä
// funktio palauttaa indeksin numeron viimMittaus parametrissä
function haeViimeinen(kyselyParametrit, viimMittaus){

            var parametrit = kyselyParametrit.split(';');

            // Luodaan tietokantayhteys
            var connection = mysql.createConnection({
                        host : 'localhost',
                        port : 3306,
                        database: parametrit[1],
                        user : 'user',
                        password : 'password'
            });

            //Suoritetaan tietokantakysely
            connection.query('SELECT mittaus from '+ parametrit[0]
             +' ORDER BY mittaus DESC LIMIT 1', function(err, rows){
                        if(err != null) {
                                    viimMittaus(0);
                        }
                        else {
                                    viimeisin=rows[0].mittaus;
                                    viimMittaus(viimeisin);
                        }
            connection.end();
            });
}
```

PÄÄOHJELMA.PHP

```
<?php
$starttime = microtime(true);

require_once( "/var/www/skriptit/tkkopiointi_lt/haerivit.php");
require_once( "/var/www/skriptit/tkkopiointi_lt/kysyservulta.php");

$rpi_tietokanta="anturit";
$serveri_tietokanta="tietokannan nimi";
$puuttuvat_mittaukset=0;
$taulu = "taulun nimi";
$servun_viimeinen_mittaus=0;

//Kysytään serveriltä (servu.js:ltä) viimeinen mittaus
$servun_viimeinen_mittaus = kysyViimeisinServulta($taulu,$serveri_tietokanta);

//Haetaan omasta tietokannasta uusin mittaus
$uusin_mittaus_rpi = hae_uusin_mittaus($taulu,$rpi_tietokanta);

//Lasketaan kuinka monta mittausriviä pitää hakea lähetettäväksi
$puuttuvat_mittaukset = $uusin_mittaus_rpi - $servun_viimeinen_mittaus;
echo "puuttuvat mittaukset: ".$puuttuvat_mittaukset."\r\n";

$i=1;
$siirtolkm=12; // Montako mittariviä halutaan siirtää kerrallaan
$eka_haettava=$servun_viimeinen_mittaus+1;
$servun_palautus = 200;
$kokonaisaika=0;
$ihan_eka = $eka_haettava;

// Jos serveriltä puuttuu mittauksia, siirretään niitä tietty määrä kerrallaan
rpi:ltä serverille.
// Määrää joudutaan rajoittamaan, koska node.js:n 'json.parse'-funktio lukee
koko json paketin ensin kerralla muistiin,
// ja vasta sen jälkeen tekee parseroinnin sille.
// Tämä rajoitus tietenkin hidastaa tietokannan kopiointia serverille, mutta
kyhäelmä toimii.

if ($puuttuvat_mittaukset !=0){
do {
                        $aloitusaika = microtime(true);

                        if ($puuttuvat_mittaukset < $siirtolkm){
                                                $siirtolkm = $puuttuvat_mittaukset;
                        }

                        $palautettava_json = haeTietokannastaPuuttu-
vat($taulu,$rpi_tietokanta,$siirtolkm, $eka_haettava);
                        $servun_palautus = lähe-
tä_serverille_uudet($taulu,$serveri_tietokanta,$palautettava_json);
                        $eka_haettava = $eka_haettava+$siirtolkm;
                        $puuttuvat_mittaukset = $puuttuvat_mittaukset - $siir-
tolkm;
                        $siirretyt = $eka_haettava+$siirtolkm-1-$ihan_eka;

                        $endtime = microtime(true);
                        $duration = $endtime - $aloitusaika;
                        $kokonaisaika = $kokonaisaika + $duration;
                        echo "\r\n Aikaa mennyt : ". $kokonaisaika."\r\n\r\n ";
```

```
}
while (($servun_palautus == 200) && ($puuttuvat_mittaukset != 0 ) && ($kokonais-
aika < 30));
}
            $endtime = microtime(true);
            $duration = $endtime - $starttime;
            echo "\r\n \r\n Koko hommaan meni aikaa: ". $duration."\n";
?>
```

TIETOKANTAHAUT.PHP

```
<?php
/*******************************************************************************
            Funktio haeTietokannastaPuuttuvat() hakee tietokannasta rivejä.
            Parametreinä, tietokanta, taulu, sekä rivien lukumäärä
            Funktio palauttaa arrayn joka sisältää tietokannan
*******************************************************************************/
function haeTietokannastaPuuttuvat($taulu, $tietokanta, $lukuMaara, $ekarivi) {
$starttime = microtime(true);
ini_set('memory_limit', '256M');

            //Yhteysparametrit
            $link = mysql_connect("localhost","user","salasana") or
            die("Yhteysvika");

            //Valitaan tietokanta
            mysql_select_db($tietokanta);

            $mittaukset = $lukuMaara; //Montako mittausta halutaan hakea
            $vikarivi = $ekarivi + $lukuMaara - 1; //Viimeinen haettava rivi

            //Koska mittaus-suure on eri tauluissa eri,
            //joudutaan muokkaamaan mysql-hakua varten haettava parametri.
            if($taulu=="ltanturi"){
                        $haettavaSuure="astemaara";
                        }
            else if ($taulu=="kwhpulssit"){
                        $haettavaSuure="pulssilukema";
                        }

            //Tehdään kysely mysql kantaan.
            $query_result = mysql_query("SELECT mittaus, sn, $haettavaSuure,
            DATE_FORMAT(aikaLeima,'%Y-%c-%d %H:%i:%s') as aikaleima FROM $taulu
            WHERE mittaus BETWEEN $ekarivi AND $vikarivi ORDER BY mittaus",
            $link);

            if ($query_result){
            //Jos haku palautti tuloksen, käsitellään ja palautetaan tulos.

                        //Tallennetaan tähän muuttujaan mysql haun tulokset
                        $haetut_lista = array();

                        // lisätään listaan queryn tulokset
                        while(($row = mysql_fetch_assoc($query_result)))
                        {
                                    array_push($haetut_lista, $row);
                        }

                        // Suljetaan yhteys tietokantaan
```

```
                        mysql_close($link);

                        //Lasketaan kulunut aika
                        $endtime = microtime(true);
                        $duration = $endtime - $starttime;

                        //Palautetaan lista
                        return $haetut_lista;
                        }
            else {
                        die('Invalid query: ' . mysql_error());
            }
}

/*****************************************************************
funktio hae_uusin_mittaus() hakee rpi:n tietokannasta uusimman mittausrivin in-
deksin
parametreina annetaan tietokannan, sekä taulun nimi.
Funktio palauttaa luvun, joka on taulun 'mittaus'-indeksi
*****************************************************************/
function hae_uusin_mittaus($taulu, $tietokanta)
{
$starttime = microtime(true);

            //Yhteysparametrit
            $link = mysql_connect("localhost","user","salasana") or
            die("Yhteysvika");

            //Valitaan tietokanta
            mysql_select_db($tietokanta);

            $query_result = mysql_query("SELECT mittaus FROM $taulu ORDER BY mit-
            taus DESC LIMIT 1", $link);
            if ($query_result){
            //Jos haku palautti tuloksen, käsitellään ja palautetaan tulos.
                        $rivi=mysql_fetch_assoc($query_result);
                        mysql_close($link);
                        $viimeisin = $rivi['mittaus'];

                        $endtime = microtime(true);
                        $duration = $endtime - $starttime;
                        return $viimeisin;
            }
            else {
                        die('Invalid query: ' . mysql_error());
            }
}
?>
```

HTTP_KYSELYT.PHP

```php
<?php
/*************************************************************
KysyServultaViimeisin() funktio kysyy serveriltä viimeisen
mittaus-indeksin numeron halutusta tietokannasta sekä taulusta.
Funktio palauttaa numeroarvon, joka on serverin tietokannan viimeinen
indeksinumero.
Kyselyyn annetaan parametriksi tietokannan sekä taulun nimet.
Palautusavona kyselystä tulee json muotoinen stringi,
joka sisältää taulun suurimman indeksin. esim. {"mittaus":1}
*************************************************************/
function kysyViimeisinServulta($taulu, $tietokanta) {

            $starttime = microtime(true);

            //Kysytään viimeinen mittaus sarakkeen arvo serveriltä.
            //Parametrinä tietokanta ja taulu

            // Array sisältää serverillle välitettävän kyselyn parametrit
            $kysy_viimeiset= array(
                        'metodi' =>'kysely',
                        'taulu' => '',
                        'tietokanta' => '',
                        );
                        $kysy_viimeiset['taulu']=$taulu;
                        $kysy_viimeiset['tietokanta']=$tietokanta;

            //Muutetaan array JSON muotoon
            $json_kysyviimeinen = json_encode($kysy_viimeiset);

            $url = "Palvelimen ip-osoite:portti";
            $curl = curl_init($url);
            curl_setopt($curl, CURLOPT_HEADER, false);
            curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
            curl_setopt($curl, CURLOPT_HTTPHEADER,
                                    array(
                                    "Content-Type: application/json",
                                    "Content-Lenght: ".
                                    strlen($json_kysyviimeinen))
                                    );
            curl_setopt($curl, CURLOPT_POST, true);
            curl_setopt($curl, CURLOPT_POSTFIELDS, $json_kysyviimeinen);

            $json_response = curl_exec($curl);

            $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);

            if ( $status != 200 ) {
                        die("Error: call to URL $url failed with status $status,
                        response $json_response, curl_error ".
                        curl_error($curl) . ", curl_errno " . curl_errno($curl));
            }
            curl_close($curl);

            $ServunViimeMittaus = json_decode($json_response, true);
            //Puretaan json-paketti

            if (isset($ServunViimeMittaus["mittaus"])){
            }
            //Lasketaan aika
```

```
            $endtime = microtime(true);
            $duration = $endtime - $starttime;
            echo $duration ."\r\n" ;
            return $ServunViimeMittaus["mittaus"];
}
/*
Funktio lähettää serverille json paketin, joka sisältää serveriltä puuttuvat
mittaustulokset.
Funktiolle annetaan parametreina: tietokanta, taulu sekä tietokannasta haettu
json paketti.
*/
function lähetä_serverille_uudet($taulu, $tietokanta, $lahetettava_array)
{
$starttime = microtime(true);

            $lahetettava_array_kokonainen = array(
                        'metodi' => 'datalahetys',
                        'tietokanta' => $tietokanta,
                        'taulu' => $taulu,
                        'mittausrivit' => $lahetettava_array,
            );
            $json_paketti = json_encode($lahetettava_array_kokonainen);

            $url = "Palvelimen ip-osoite:portti";
            $curl = curl_init($url);
            curl_setopt($curl, CURLOPT_HEADER, false);
            curl_setopt($curl, CURLOPT_RETURNTRANSFER, true);
            curl_setopt($curl, CURLOPT_HTTPHEADER,
                    array(
                                "Content-Type: application/json",
                                "Content-Lenght: ". strlen($json_paketti),
                                "Expect: ")
                                );
            curl_setopt($curl, CURLOPT_POST, true);
            curl_setopt($curl, CURLOPT_POSTFIELDS, $json_paketti);

            $json_response = curl_exec($curl);
            $status = curl_getinfo($curl, CURLINFO_HTTP_CODE);

            if ( $status != 200 ) {
                                return $status;
                                curl_close($curl);
            }
            else {
                        curl_close($curl);
                        return $status;
                        $endtime = microtime(true);
                        $duration = $endtime - $starttime;
            }
}
?>
```

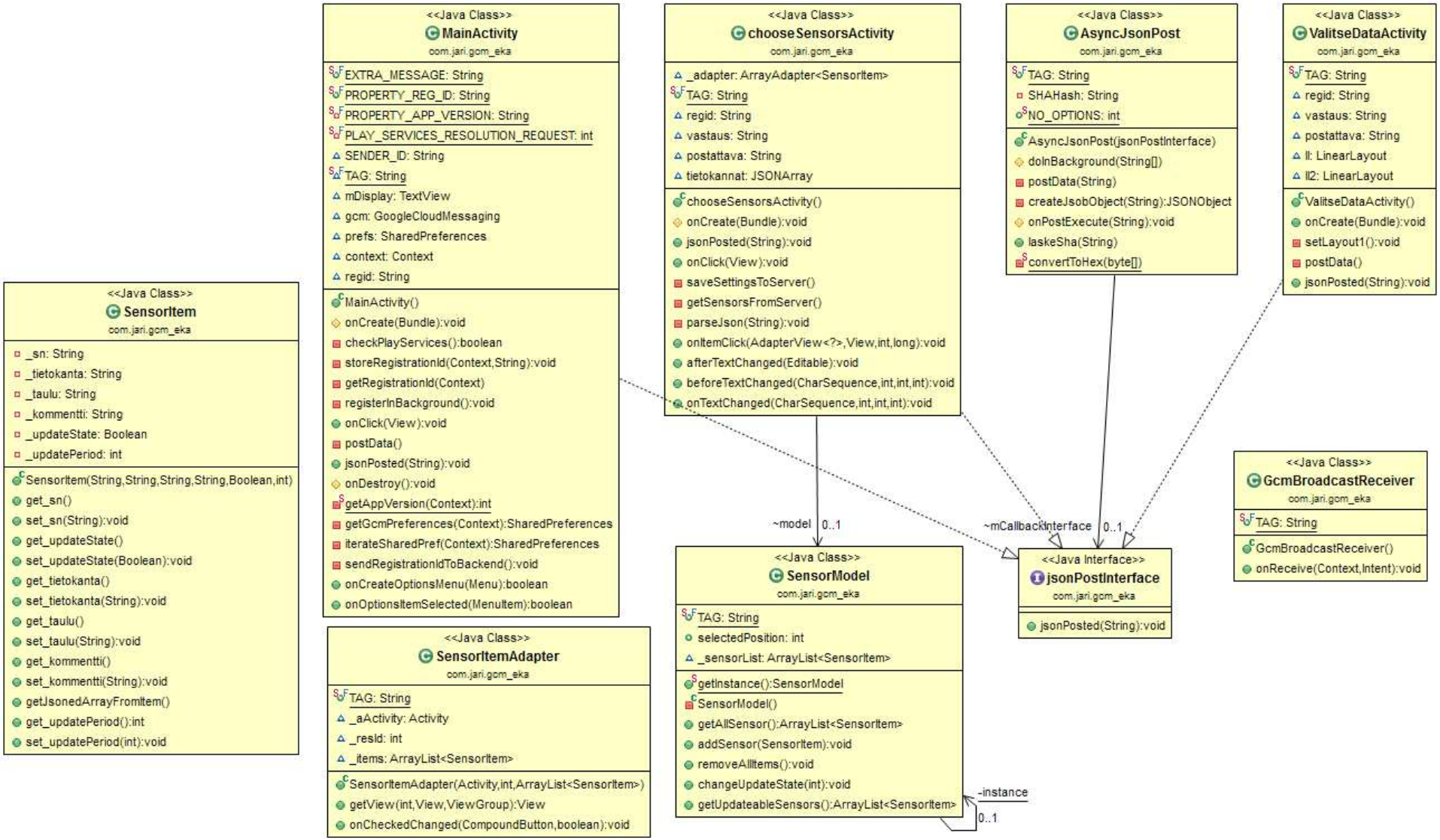
TIMER.JS

```
var mysql = require('mysql');
var gcmServer = require('./gcmsender.js');
var haeSn = require('./testihaku_sn.js');
var tietokantahaut = require('./tkhaut.js');

var counter = 0;
var interval = setInterval( function() {

var taulut = new Array;
var rivit = new Array;

//Haetaan gcm_regid_rekisteri tietokannasta käyttäjätiedot sisältävät taulun
nimet (esim. gcm_regid_rekisteri.Jg1Q7WXJomqLlMh9sltxdTAzink)

haeTaulutQuery = "SELECT table_schema, table_name FROM information_schema.tables
WHERE TABLE_SCHEMA = 'gcm_regid_rekisteri'";
tietokantahaut.haeTaulut("timer", haeTaulutQuery, function (taulut) {

// Haetaan käyttäjätietotauluista käyttäjien regideet sekä käyttäjiän tilaamien
antureiden tiedot (esim. text, jari_at.ltanturi, 1)
haeSn.haeSn(taulut, function (err,returnArray) {
   if(err){
      console.log(err);
   }
   else {
// Käydään array läpi, ja haetaan viimeinen mittaustulos
// jokaiselle tilatulle  anturille
           for (i = 0 ; i < returnArray.length; i++) {
           haeViimeinen(returnArray[i], function(viimeisin){
           });
         }
       }
    });
  });
}, 60*1000);



// Haetaan tietokannasta gcm_regid_rekisteri kaikki taulut
function haeTaulut(err, palautusArray){

           var connection = mysql.createConnection({
           host : 'localhost',
           port : 3306,
           //database: parametrit[1],
           user : 'root',
           password : 'jari',
           })

           mysqlKysely = "SELECT table_name FROM information_schema.tables
           WHERE TABLE_SCHEMA = 'gcm_regid_rekisteri'";
           connection.query(mysqlKysely, function(err, rows){
                     var taulut = new Array;
                     if(err != null) {
                        console.log("Query error:" + err);
                     }
                     else {
```

```
                        for( i=0; i < rows.length+0 ; i++){
                           taulut[i] = rows[i].table_name;
                    }
                    palautusArray(taulut);
                    }
            connection.end();
            });
}

/**
            Haetaan tietokannasta halutun anturin viimeinen lukema,
            ja lähetetään se GCM vastaanottajalle.
            Parametriksi annetaan array, jossa on kerrottu anturin tietokanta,
            taulu ja sarjanumero, sekä regid, johon data lähetetään.
            Kun kysely on saatu suoritettua onnistuneesti,
            kutsutaan GCM:lle dataa lähettävää funktiota,
            Annetaan funktiolle parametreiksi mittaus#, mittausdata sekä regid
*/
function haeViimeinen(param, viimMittaus){

            taulu =  param.tietokanta+'.'+param.taulu;
            var connection = mysql.createConnection({
                    host : 'localhost',
                    port : 3306,
                    user : 'root',
                    password : 'jari'
            });

            if (param.taulu == 'kwhpulssit') {
               queryString = 'SELECT aikaleima, mittaus, pulssilukema
                              AS mittausdata from '+taulu+'
                              WHERE sn='+param.anturisn+'
                              ORDER BY mittaus DESC LIMIT 1';
            }
            else if (param.taulu == 'ltanturi') {
               queryString = 'SELECT aikaleima, mittaus, astemaara
                              AS mittausdata from '+taulu+'
                              WHERE sn='+param.anturisn+'
                              ORDER BY mittaus DESC LIMIT 1';
            }
            connection.query (queryString, function (err, rows){
                    var palautusRivit = new Array;
                    if(err != null) {
                    }
                    else {
                                palautusRivit[0]=rows[0].mittaus;
                                palautusRivit[1]=rows[0].mittausdata;
                                palautusRivit[2]=param.regid;
                                palautusRivit[3]=param.taulu;
                                palautusRivit[4]=param.tietokanta;
                                palautusRivit[5]=rows[0].aikaleima;
                                gcmServer.sendToGcm(palautusRivit);
                    }
            connection.end();
            });
}
```

GCMSENDER.PHP

```
var gcm = require('node-gcm');

function sendToGcm(param){

            var message = new gcm.Message();

            //Luetaan lähetettävä data parametrina saadusta arraysta
            if (param[0] != null || param[1] != null || param[5] != null) {
                        mittaus = param[0].toString();
                        mittausdata = param[1].toString();
                        aikaleima = param[5].toString();
            }
            else {
                        mittaus = "n/a";
                        mittausdata = "n/a";
                        aikaleima = "n/a";
            }
            regid = param[2].toString();
            taulu = param[3].toString();
            tietokanta = param[4].toString();

            // Kootaan lähetettävä viesti
            var message = new gcm.Message({
                        collapseKey: 'demo',
                        delayWhileIdle: true,
                        timeToLive: 60,
                        data: {
                                    Mittaus : mittaus ,
                                    Tietokanta: tietokanta,
                                    Taulu: taulu,
                                    Lukema: mittausdata,
                                    Aikaleima: aikaleima
                        }
            });

            var sender = new gcm.Sender('Tähän ApiKey'); //ApiKey
            var registrationIds = []; //Vastaanottajien 'osoitteet'

            registrationIds.push(regid);
            // Lähetetään viesti
            sender.send(message, registrationIds, 4, function (err, result) {
            // Tässä voidaan ottaa vastaan lähetyksen status
            });
}


exports.sendToGcm = sendToGcm;
```


<<Java Class>>
MainActivity
com.jari.gcm_eka
EXTRA_MESSAGE: String
PROPERTY_REG_ID: String
PROPERTY_APP_VERSION: String
PLAY_SERVICES_RESOLUTION_REQUEST: int
SENDER_ID: String
TAG: String
mDisplay: TextView
gcm: GoogleCloudMessaging
prefs: SharedPreferences
context: Context
regid: String
MainActivity()
onCreate(Bundle):void
checkPlayServices():boolean
storeRegistrationId(Context,String):void
getRegistrationId(Context)
registerInBackground():void
onClick(View):void
postData()
jsonPosted(String):void
onDestroy():void
getAppVersion(Context):int
getGcmPreferences(Context):SharedPreferences
iterateSharedPref(Context):SharedPreferences
sendRegistrationIdToBackend():void
onCreateOptionsMenu(Menu):boolean
onOptionsItemSelected(MenuItem):boolean
<<Java Class>>
chooseSensorsActivity
com.jari.gcm_eka
_adapter: ArrayAdapter<SensorItem>
TAG: String
regid: String
vastaus: String
postattava: String
tietokannat: JSONArray
chooseSensorsActivity()
onCreate(Bundle):void
jsonPosted(String):void
onClick(View):void
saveSettingsToServer()
getSensorsFromServer()
parseJson(String):void
onItemClick(AdapterView<?>,View,int,long):void
afterTextChanged(Editable):void
beforeTextChanged(CharSequence,int,int,int):void
onTextChanged(CharSequence,int,int,int):void
<<Java Class>>
AsyncJsonPost
com.jari.gcm_eka
TAG: String
SHAHash: String
NO_OPTIONS: int
AsyncJsonPost(jsonPostInterface)
doInBackground(String[])
postData(String)
createJsobObject(String):JSONObject
onPostExecute(String):void
laskeSha(String)
convertToHex(byte[])
<<Java Class>>
ValitseDataActivity
com.jari.gcm_eka
TAG: String
regid: String
vastaus: String
postattava: String
ll: LinearLayout
ll2: LinearLayout
ValitseDataActivity()
onCreate(Bundle):void
setLayout1():void
postData()
jsonPosted(String):void
<<Java Class>>
SensorItem
com.jari.gcm_eka
_sn: String
_tietokanta: String
_taulu: String
_kommentti: String
_updateState: Boolean
_updatePeriod: int
SensorItem(String,String,String,String,Boolean,int)
get_sn()
set_sn(String):void
get_updateState()
set_updateState(Boolean):void
get_tietokanta()
set_tietokanta(String):void
get_taulu()
set_taulu(String):void
get_kommentti()
set_kommentti(String):void
getJsonedArrayFromItem()
get_updatePeriod():int
set_updatePeriod(int):void
<<Java Class>>
SensorItemAdapter
com.jari.gcm_eka
TAG: String
_aActivity: Activity
_resId: int
_items: ArrayList<SensorItem>
SensorItemAdapter(Activity,int,ArrayList<SensorItem>)
getView(int,View,ViewGroup):View
onCheckedChanged(CompoundButton,boolean):void
<<Java Class>>
SensorModel
com.jari.gcm_eka
TAG: String
selectedPosition: int
_sensorList: ArrayList<SensorItem>
getInstance():SensorModel
SensorModel()
getAllSensor():ArrayList<SensorItem>
addSensor(SensorItem):void
removeAllItems():void
changeUpdateState(int):void
getUpdateableSensors():ArrayList<SensorItem>
<<Java Interface>>
jsonPostInterface
com.jari.gcm_eka
jsonPosted(String):void
<<Java Class>>
GcmBroadcastReceiver
com.jari.gcm_eka
TAG: String
GcmBroadcastReceiver()
onReceive(Context,Intent):void
~model
0..1
~mCallbackInterface
0..1
-instance
0..1