Hälytyspalvelu Java-palvelimessa

TreLab Oy

Minna Isokorpi

Opinnäytetyö
Marraskuu 2014
Tietotekniikka
Ohjelmistotekniikka

TAMPEREEN AMMATTIKORKEAKOULU
Tampere University of Applied Sciences

TIIVISTELMÄ

Tampereen ammattikorkeakoulu
Tietotekniikka
Ohjelmistotekniikka

MINNA ISOKORPI:
Hälytyspalvelu Java-palvelimessa
TreLab Oy

Opinnäytetyö 51 sivua, joista liitteitä 0 sivua
Marraskuu 2014

Tämän opinnäytetyön toimeksiantaja on langattomia mittalaitejärjestelmiä valmistava TreLab Oy, joka tarvitsi toisen sukupolven taustajärjestelmäänsä hälytyspalvelun. Hälytyspalvelun tehtävänä oli vastaanottaa muusta taustajärjestelmästä tulevat hälytykset ja välittää ne eteenpäin eri tavoin, esimerkiksi push-viestillä tai tekstiviestillä. Opinnäytetyössä toteutettiin vain yksi hälytystapa: push-viestit Android-mobiililaitteisiin, mutta suunnittelussa ja toteutuksessa tuli ottaa huomioon myös muiden hälytystapojen lisääminen palveluun myöhemmin. Mobiilisovellus push-viestien vastaanottoa varten oli jo valmis.

Hälytyspalvelu toteutettiin Javalla käyttäen Spring-sovelluskehystä. Ulkoiset rajapinnat push-viestejä välittävään Google Cloud Message –palvelimeen ja muihin taustajärjestelmän palveluihin toteutettiin REST-arkkitehtuurimallin mukaisesti. Push-viestien lähettämisessä käytettiin avoimen lähdekoodin gcm-server –kirjastoa. Valmis hälytyspalvelu tuli paketoida Mavenilla tuotantopalvelimelle siirrettäväksi war-paketiksi.

Luvussa yksi kerrotaan enemmän opinnäytetyön aiheesta ja vaatimuksista sekä TreLab Oy:stä. Toinen luku käsittelee hälytyspalvelun taustalla olevaan teoriaa ja valittuja tekniikoita. Opinnäytetyön kolmannessa luvussa kerrotaan työn käytännön toteutuksesta: rajapinnoista, tietorakenteista, kontrollerista, push-viestin lähettämisestä, käytetyistä annotaatioista sekä testauksesta.

Valmis hälytyspalvelu on toimiva ja täyttää sille asetetut vaatimukset. Jatkokehitys on jo aloitettu, ja uusien hälytystapojen lisääminen hälytyspalveluun onnistuu helposti suunnitelmien mukaan.

Asiasanat: web-palvelin, Spring, push-viesti, REST, Maven

ABSTRACT

Tampere University of Applied Sciences
Computer Science
Software engineering

MINNA ISOKORPI:
External Alerts Service on a Java Server
TreLab Oy

Bachelor's thesis 51 pages, appendices 0 pages
November 2014

The assignment for my Bachelor's thesis was received from TreLab Oy. TreLab needed alerting service for their wireless measurement system's new server. The function of the external alert service is to send alerts coming from system's server forward to the clients in different ways, for example by using push notifications or text messages. In this project only one way was implemented: push notification for Android mobile devices, but the external alert service was to be planned and built so that other ways would be easy to add afterwards. The Android mobile application by TreLab was already made and ready to receive push notifications.

The external alert service was to be done in Java and built as a Maven war-package to be run on the TreLab's server. The external alert service uses Spring framework and is RESTful: the interfaces to the TreLab server and Google Cloud Message service for push notifications were done by using REST architectural style. The sending of the push messages was done by using open-source gcm-server library.

Chapter one describes the title and requirements more accurately. The first chapter also tells more about TreLab. Chapter two discusses the theory and the techniques behind external alert service. The third chapter of the Bachelor's thesis is about the practical implementation of the service: the interfaces, the controller, sending a push notification, annotations used and testing.

The ready-made external alerts service works and meets the set requirements. The further development has already begun and adding the missing alert sending methods is going as smoothly as planned.

Key words: web-server, Spring framework, push notification, REST, Maven

SISÄLLYS

ERITYISSANASTO

Annotaatio	Javassa @-alkuinen merkintä, joka kertoo muuttujan, metodin, parametrin tai luokan toiminnasta sovelluskehykselle.
APNs	Apple Push Notification service, eli Applen push-viestejä iOS-laitteille välittävä palvelin.
cURL	Komentorivityökalu tiedon siirtoon käyttäen URL-syntaksia. Käytettiin opinnäytetyön testaamisessa.
EAS	External Alert Service, eli hälytyspalvelu. Opinnäytetyössä toteutetun palvelun nimi.
Exponential backoff	Eksponentiaalinen toipumismenettely. Algoritmi, jossa odotettu aika lisääntyy eksponentiaalisesti jokaisen epäonnistumisen jälkeen ennen seuraavaa yritystä.
GCM	Google Cloud Messaging, eli Googlen push-viestejä Android-laitteille välittävä palvelin.
Git	Opinnäytetyössä käytetty versionhallintatyökalu, käytetään yleensä komentoriviltä.
HTTP	HyperText Transfer Protocol, eli internet-protokolla. Käytetään selaimen ja palvelimen väliseen tiedonsiirtoon.
JSON	JavaScript Object Notation, eli yksinkertainen, tekstipohjainen ja ohjelmistokielestä riippumaton tiedonsiirtoformaatti, jota käytetään jäsennetyn tiedon välittämisessä ohjelmien välillä.
Maven	Opinnäytetyössä käytetty, avoimen lähdekoodin projektinhallintatyökalu.
MVC	Model View Controller, eli malli, näkymä, käsittelijä. Arkkitehtuurityyli, joka jakaa sovelluksen kolmeen osaan. Jaon tarkoituksena on erottaa sovelluslogiikka käyttöliittymästä.
Push-viesti	Viesti, joka lähetettään automaattisesti riippumatta vastaanottajan tilasta. Push-viestin vastakohta on ”Pull”-viesti, jonka vastaanottaja pyytää aina oma-aloitteisesti.
Repository	Versionhallintaohjelman tietovarasto.

REST	Representational State Transfer, eli HTTP-protokollaan perustuva malli, joka määrittelee miten HTTP-protokollan pyyntömetodeja ja osoitteita tulee käyttää ohjelmointirajapintojen toteuttamiseen.
STS	Spring Tool Suite on opinnäytetyössä käytetty kehitysympäristö Spring-sovelluskehystä varten.
WNS	Windows Push Notification Service, eli Microsoftin Windows-laitteille push-viestejä välittävä palvelin.
XML	Extensible Markup Language, eli metakieli, jolla määritellään rakenteellisia merkkauskieliä. XML-kieliä voi käyttää minkä tahansa tiedon kuvailemiseen.

1 JOHDANTO

Syksyllä 2013 TreLab Oy oli kehittämässä langattoman mittalaitejärjestelmän seuraavan sukupolven taustajärjestelmää. Taustajärjestelmän yhtenä osana tarvittiin palvelu, joka lähettää asiakkaalle hälytyksen, esimerkiksi mobiilisovellukseen tai tekstiviestillä puhelimeen. Palvelu sopi hyvin opinnäytetyöksi kooltaan ja vaikeusasteeltaan. Palvelun nimeksi tuli External Alert Service, EAS, eli suomeksi hälytyspalvelu. Opinnäytetyö rajattiin välittämään hälytyksiä vain Andoid-mobiilisovelluksille, mutta palvelun suunnittelussa tuli ottaa huomioon myös muiden hälytystapojen lisääminen lähitulevaisuudessa. Valmis sovellus on välikappale TreLabin muun taustajärjestelmän ja kolmannen osapuolen palvelimien välillä.

Tämän opinnäytetyön ensimmäinen kappale käsittelee hälytyspalvelun vaatimuksia ja aiheen rajausta. Toisessa kappaleessa käydään läpi opinnäytetyöhön liittyvää teoriaa yleisesti palvelimen perustoiminnallisuudesta ja rajapinnoista, Spring-sovelluskehyksestä sekä push-viesteistä painottuen Androidiin. Kolmas kappale käsittelee käytännön suunnittelua ja toteutusta sekä esittelee käytettyjä työkaluja ja tekniikoita.

1.1 TreLab Oy

TreLab Oy on vuonna 2011 perustettu tamperelainen noin kymmenen hengen yritys, joka kehittää langattomia mittausratkaisuja. Langattomat mittalaitteet ovat osa suurempaa mittalaitejärjestelmää, jonka avulla voidaan valvoa sekä kerätä ja analysoida tietoa esimerkiksi rakennuksista, koneista ja laitteista, ympäristön olosuhteista sekä ihmisistä. (TreLab Oy 2013, TreLab)

TreLab-mittajärjestelmä koostuu pienestä pyöreästä mittalaitteesta, yhdestä tai useammasta tukiasemasta sekä pilvipalvelusta. Mittalaite painaa noin 17 grammaa ja on halkaisijaltaan 4 cm leveä ja 1,2 cm paksu. Yrityksen internet-sivuilla (http://www.trelab.fi/mittajarjestelma/) mittalaitteen vakio-ominaisuuksiksi listataan

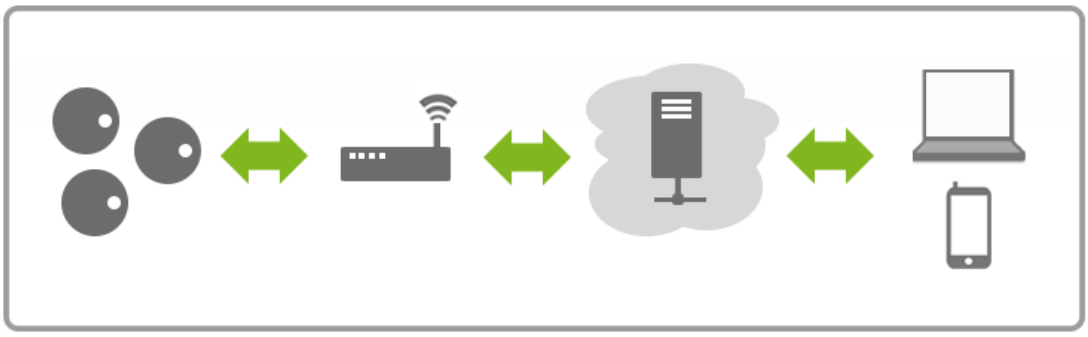
- kiihtyvyys, pyöriminen, kallistus, liikkeen suunta, aktiivisuus
- liike/liikkumattomuus, putoaminen, törmäys

- ympäristötiedot: ilmanpaine, lämpötila, suhteellinen kosteus, valoisuus
- läsnäolo, suhteellinen paikkatieto

Mittalaite on pieni ja huomaamaton ja mahdollistaa järjestelmän soveltamisen moneen käyttötarkoitukseen. Kuvassa 1 on mittalaite sekä avain kokovertailua varten. (TreLab Oy 2013, mittajärjestelmä)

KUVA 1. Langaton mittalaite sekä avain (TreLab Oy 2013, mittajärjestelmä)

Tukiasemien avulla mittalaitteen tiedot kerätään pilvipalveluun (kuva 2). Kerättyjä tietoja voidaan seurata esimerkiksi www-selaimella tai TreLabin oman mobiilisovelluksen avulla. Näin luodaan reaaliaikaista tilannekuvaa esimerkiksi tehtaasta, asunnosta, ihmisistä tai ympäristöstä. Mittaustiedoista voidaan koostaa myös pidemmän ajan raportteja helpottamaan tiedon analysointia.



KUVA 2. TreLab-mittajärjestelmä (TreLab Oy 2013, Ihmiset)

1.2 Vaatimusluettelo ja aiheen rajaus

EAS-hälytyspalvelu sijoittuu TreLabin taustajärjestelmän ja Googlen Cloud Messaging –palvelimen välille. Molemmat rajapinnat asettavat sovellukselle erilaisia vaatimuksia.

Google Cloud Message –palvelimen asettamista vaatimuksista sovelluspalvelimelle on kerrottu luvussa 2.5.2.

TreLab oli laatinut opinnäytetyötä varten projektin alussa vaatimusluettelon. EAS:in tuli olla Java 7 –sovellus, joka koostetaan Mavenilla war-paketiksi ja joka voidaan sijoittaa Tomcat-palvelimelle. Sovelluksen tuli kommunikoida muun taustajärjestelmän kanssa, esimerkiksi vastaanottaa hälytyskäskyjä, kysellä vastaanottajien tietoja ja ilmoittaa vanhentuneista puhelinnumeroista tai epäonnistuneista push-viesteistä.

Vaatimuksiin kuului luonnollisesti myös oikean hälytyksen lähettäminen oikealle henkilölle määritellyllä tavalla. Hälytyksen lähettäminen palvelimelta pitäisi tapahtua mahdollisimman nopeasti. Hälytyspalvelun piti myös reagoida viestin lähettämisen mahdolliseen epäonnistumiseen, esimerkiksi GCM tokenin vanhentumiseen tai mobiilisovelluksen poistamiseen laitteesta, ja näiden tietojen välittämiseen eteenpäin muulle taustajärjestelmälle.

Erilliseen properties-tiedostoon tuli kerätä kaikki “vakiomuutujat”. Muuttujien kerääminen yhteen properties-tiedostoon helpottaa niiden ylläpitoa, kuin jos ne olisi ripoteltu ympäri koodia. properties-tiedosto muodostaa tavallaan sovelluksen asetukset. Vakiomuuttujia ovat esimerkiksi seuraavan laiset tiedot:

- tietoa käytetään useissa kohdissa koodia mutta jotka eivät muutu usein
- tiedot halutaan pitää erillään muusta ohjelmakoodista ja määritellä vain kerran, jotta ne on helppo löytää ja ylläpitää
- tiedot, jotka eivät muutu ajon aikana.

Esimerkiksi lokiin kirjoittamisen taso (info, debug, fatal, error, warn), käyttäjätunnukset ja salasanat, lokalisointi tai muiden palvelimien osoitteet ovat properties-tiedostoon kerättäviä tietoja.

EAS:in tehtävänä on lähettää hälytyksiä asiakkaalle monilla eri tavoilla. Koska opinnäytetyö on vain 15 opintopistettä (vastaa noin 400 tuntia työtä) ja työmäärän arvioitiin olevan paljon enemmän, päätettiin tässä projektissa toteuttaa vain yksi hälytystapa: push-viestit TreLabin omalle Android-sovellukselle. EAS:in arkkitehtuuri- ja rajapintasuunnittelussa tuli kuitenkin ottaa huomioon myös hälytysten lähetys muilla tavoin.

Opinnäytetyön suunnitteleminen alkoi jouluna 2013 ja sen aihe oli suoraa jatkoa kesällä 2013 aloitettuun push-viestien lähettämiskokeiluun TreLabille. Tavoitteena oli saada opinnäytetyö valmiiksi ja valmistua kevään 2014 aikana. Suunnitteluun ja aiheen opiskeluun arvioitiin menevän noin kuukausi, varsinaiseen toteuttamiseen kaksi kuukautta ja viimeistelyyn sekä integrointiin kuukausi. Opinnäytetyön raportointi suunniteltiin tehtäväksi toteuttamisen ohella työn loppuvaiheilla.

2 TEORIA

Toisessa luvussa käsitellään hälytyspalvelun pohjana olevaa teoriaa. Luvun ensimmäiset kolme kappaletta käsittelevät web-palvelimen toimintaa ja niissä käytettyä tekniikkaa yleisesti. Neljäs kappale kertoo tarkemmin opinnäytetyössä hyödynnetystä Spring-sovelluskehyksestä. Luvun viimeisessä kappaleessa kerrotaan push-viesteistä ja niiden lähettämisestä sovelluksen palvelimelta käyttöjärjestelmän palvelimelle, ja lopuksi syvennytään tarkemmin push-viestien lähettämiseen Andoid-mobiilisovellukselle.

2.1 Web-palvelin

Internet muodostuu monista tehokkaista palvelintietokoneista, joiden tarjoamia palveluita asiakasohjelmistot käyttävät. Tätä kutsutaan asiakas-palvelin –malliksi. Palvelimet ja asiakkaat sijaitsevat fyysisesti eri tietokoneilla, jotka on kytketty toisiinsa tietoliikenneyhteyksillä ja joiden avulla ne kommunikoivat. Tämän ansiosta loppukäyttäjä voi sijaita toisella puolella maapalloa palvelimeen nähden. Asiakasohjelma pyytää palvelimelta tietoja ja palvelin vastaa pyyntöön lähettämällä pyydetyt tiedot. (Vihavainen 2014, 2.2; Suomen Internetopas: Internetin toimintaperiaate.)

HTTP (HyperText Transfer Protocol) on TCP/IP -protokolla, jota käytetään web-palvelimien ja asiakkaiden väliseen kommunikointiin. HTTP-asiakasohjelma, esimerkiksi selain, lähettää pyynnön HTTP-viestinä HTTP-palvelimelle, joka palauttaa vastauksen HTTP-muodossa. Jokaista pyyntöä kohden lähetetään yksi vastaus. (Vihavainen 2014, 2.1.)

Pyyntöjen ja vastausten lähettäminen ei onnistu ilman kohdetietoa, IP-osoitetta. Tietokonekohtainen IP-osoite varmistaa, että tieto kulkee oikealle vastaanottajalle. Osoitteiden muistamisen helpottamiseksi osoite näkyy käyttäjälle osoitekentässä tekstimuotoisena (URI tai URL). Selain muuttaa tekstimuotoisen osoitteen numeeriseksi tekemällä kyselyn DNS-palvelimelle (Domian Name System), joka palauttaa oikean IP-osoitteen. (Vihavainen 2014, 2.1; Suomen Internetopas: Internetin osoitekäytäntö.)

HTTP-viestit ovat tekstimuotoisia. Palvelimelle lähetettävässä pyynnössä on ensimmäisellä rivillä pyynnön tyyppi (esimerkiksi POST tai GET), pyydetyn resurssin polku ja HTTP-protokollan versionumero. Seuraavat rivit on varattu viestin otsakkeille sekä rungolle, jotka eivät ole pakollisia. Viesti päättyy kahteen peräkkäiseen rivinvaihtoon. LISTAUS 1 on esimerkki HTTP-pyynnöstä. Koska yhteys palvelimeen on jo muodostettu, ei osoitetta tarvitse mainita viestissä erikseen, vain resurssin polku riittää. (Vihavainen 2014, 2.2.)

LISTAUS 1. HTTP-pyyntöesimerkki (Vihavainen 2014, 2.2)

```
GET /index.html HTTP/1.0
```

Palvelin vastaa aina pyyntöön, jos yhteys saadaan muodostettua. Selain ilmoittaa käyttäjälle, jos palvelinta ei löydy. Palvelimen vastauksen ensimmäisellä rivillä kerrotaan HTTP-protokollan versionumero, viestiin liittyvä statuskoodi sekä statuskoodin selitys. Seuraavaksi tulee otsakkeita ja otsakkeiden jälkeen tyhjä rivi ja varsinainen vastaus. Kun asiakasohjelma saa vastauksen, se tarkistaa statuskoodin ja päättelee sen jälkeen mitä vastaukselle pitäisi tehdä. Vastaus voi olla esimerkiksi internetsivu, jonka selain näyttää statuskoodin tarkastettuaan. Tavallisesti vastaus sisältää vain pyydetyn tiedon, sillä liikkuvan tiedon määrä pyritään pitämään vähäisenä. Listaus LISTAUS 2 on esimerkki HTTP-vastauksesta. (Vihavainen 2014, 2.2.)

LISTAUS 2. HTTP-vastausesimerkki (Vihavainen 2014, 2.2)

```
HTTP/1.1 200 OK
Date: Mon, 01 Sep 2014 03:12:45 GMT
Server: Apache/2.2.14 (Ubuntu)
Vary: Accept-Encoding
Content-Length: 973
Connection: close
Content-Type: text/html;charset=UTF-8

.. vastaus runko ..
```

Palvelimen vastauksessa lähetetty statuskoodi on kolmenumeroinen luku, joka kuvaa pyynnön seurauksena palvelimella tapahtunutta toimintaa. Esimerkiksi yleisin, 200 ”OK”, kertoo että kaikki meni oikein. Toinen hyvin yleinen statuskoodi on 400 ”Not

Found”, eli yhteys palvelimeen muodostettiin, mutta palvelin ei löytänyt pyydettyä resurssia. (Vihavainen 2014, 2.2.)

Kun asiakas haluaa vain hakea tietoa, on HTTP-viestin tyyppi GET. Jos taas halutaan hakemisen sijaan muokata palvelimen hallussa olevaa dataa, käytetään HTTP-viestin otsikossa tyyppiä POST, PUT tai DELETE. Kun halutaan lisätä uutta tietoa, käytetään POST-metodia. Lisättävä tieto annetaan yleensä viestin rungossa jossakin standardoidussa muodossa kuten JSON tai XML. Kun tietoa halutaan poistaa, käytetään DELETE-metodia. PUT-metodilla muutetaan tai päivitetään jo olemassa olevaa tietoa. (Vihavainen 2014, 2.2, 5.)

Asiakas ei tarkoita vain selainta ja sitä käyttävää ihmistä; palvelimen kannalta asiakas voi olla myös toinen palvelin tai ohjelma. Koska HTTP-pyynnöt ja vastaukset noudattavat aina samaa kaavaa ja muotoa, voidaan niitä käyttää myös ohjelmien tai palvelimien väliseen kommunikointiin. Tekniikkaa käytetään hyödyksi esimerkiksi mobiilisovelluksissa, jotka tarvitsevat tietoa, mutta joiden oma kapasiteetti ei yksin riitä tiedon hallintaan. (Williams 2014, 474.)

Asiakasmäärän kasvaminen voi johtaa palvelimen ylikuormittumiseen. Ylikuormittuminen näkyy pitkinä vastausaikoina tai sivuston kaatumisena. Kuormittumiseen vaikuttavat muun muassa verkon nopeus ja laatu, palvelimen fyysinen kapasiteetti sekä palvelun tyyppi; esimerkiksi tiedon tallentaminen vie enemmän palvelimen resursseja kuin pelkän staattisen tiedon hakeminen. Ylikuormittumista voidaan ehkäistä lisäämällä palvelimen kapasiteettia tai optimoimalla sen toimintaa. (Vihavainen 2014, 2.2.)

2.2 REST-rajapinta

Hälytyspalvelussa sovellettu REST (Representational State Transfer) on arkkitehtuurimalli ohjelmarajapintojen toteuttamiseen (Vihavainen 2014, 5). Termin määritteli Roy Fielding tohtorinväitöskirjassa vuonna 2000. Fieldingin mukaan REST ei oikeastaan ole arkkitehtuuri, vaan kokoelma ohjeita, joita noudattamalla päädytään tiettyyn arkkitehtuurityyliin. REST ei ota kantaa käytettäviin tekniikoihin tai protokolliin, vaan määrittelee miten tieto liikkuu komponenttien välillä ja mitä hyötyä siitä on. Sitä voidaan sovel-

taa minkä tahansa verkon suunnittelussa. REST-arkkitehtuurityylin tunnusmerkkejä ovat:

- Palvelin-asiakas-malli
- Tilattomuus (ei sessioita; jokainen pyyntö on riippumaton muista)
- Välimuistin tukeminen
- Resurssien yksikäsitteinen osoitteistus ja saavutettavuus
- Kerrosrakenteinen, skaalautuva (Sandoval 2009, 7-8).

REST:istä ja sen määrittelystä löytyy monta tulkintaa. Esimerkiksi Vihavainen määrittelee oppimateriaalissaan (2014, 5) seuraavasti: ”REST on HTTP-protokollaan perustuva malli, joka määrittelee miten HTTP-protokollan pyyntömetodeja ja osoitteita tulee käyttää ohjelmointirajapintojen toteuttamiseen.” Markhamin (2014, 226) kirjasta Java programming interviews exposed löytyvä määritelmä on lähes sama:”Representational State Transfer, or REST for short, is a style of using HTTP to provide remote API calls between systems.” Sandovalin (2009, 7-8) mielestä REST ei tarkoita yksittäistä tekniikkaa, ja pohtii esimerkiksi onko internet REST-arkkitehtuurin mukainen:

> “On the one hand, the static web is RESTful, because static websites follow Fielding's definition of a RESTful architecture. For instance, the existing web infrastructure provides caching systems, stateless connection, and unique hyperlinks to resources, where resources are all of the documents available on every website and the representations of these documents are already set by files being browser readable (HTML files, for example). Therefore, the static web is a system built on the REST-like architectural style.
>
> On the other hand, traditional dynamic web applications haven't always been RESTful, because they typically break some of the outlined constraints. For instance, most dynamic applications are not stateless, as servers require tracking users through container sessions or client-side cookie schemes. Therefore, we conclude that the dynamic web is not normally built on the REST-like architectural style.”

Määritelmien välillä on kulunut muutama vuosi, joten voidaan päätellä että HTTP-prokollan käyttö REST-rajapintojen yhteydessä on yleistynyt ja muodostunut normiksi. REST:issä on yksinkertaisimmillaan kyse ohjelmistojen vuorovaikutuksesta ja resurssien muokkauksesta verkon välityksellä (Sandoval 2009, 12).

REST-arkkitehtuurityyliä voidaan soveltaa myös web-palvelinohjelmoinnin rajapinnoissa, kuten myös hälytyspalvelussa tehtiin. Käytännössä se toteutettiin käyttäen

HTTP-protokollaa ja ohjelmistorajapintojen toteuttamista sen pyyntömetodien ja osoitteiden avulla. HTTP-protokolla perustuu asiakas-palvelin-malliin ja jokaisella resurssilla on yksilöllinen tunnus eli osoite ja pyyntötyypillä (GET, PUT, POST ja DELETE) kuvataan resurssiin kohdistuva toiminto. (Vihavainen 2014, 5.) Pyydetty resurssi voi olla missä muodossa tahansa: HTML, XML, JSON, tekstiä tai ohjelmakoodia, kunhan sekä lähettäjä että vastaanottaja ymmärtävät sitä (Williams 2014, 476).

Hyvin tunnettu, yksinkertainen ja standardoitu HTTP-protokolla ja pyyntötyypit tekevät REST-rajapinnasta yksinkertaisen ja helpon oppia. REST-rajapinnan etuina on myös sen monipuolisuus: HTTP–protokolla mahdollistaan esimerkiksi tekstin, äänen ja videon jakamisen asiakkaan ja palvelimen välillä. Myös asiakkaan ja palvelimen roolit ovat selkeät, jolloin niitä voidaan kehittää erillään. REST on myös hyödyllinen ja paljon käytetty malli esimerkiksi vanhojen palveluiden kapselointiin sekä uusien rajapintojen tarjoamiseen (Vihavainen 2014, 5).

2.3 JSON-syntaksi

JSON (JavaScript Object Notation) on yksinkertainen, tekstipohjainen ja ohjelmistokielestä riippumaton tiedonsiirtoformaatti, jota käytetään jäsennetyn tiedon välittämisessä ohjelmien välillä. Se esiteltiin ensimmäisen kerran vuonna 2001 JSON.org-sivustolla. (Ecma International 2013, ii, 2.) JSON-formaattia käytetään varsinkin palvelin- ja selainohjelmistojen välisessä tiedonsiirrossa yhä enemmän sen yksinkertaisuuden takia (Vihavainen 2013, 6.2). Opinnäytetyössä JSON-formaattia hyödynnettiin ulkoisissa rajapinnoissa GCM-palvelimeen ja TreLabin muuhun taustajärjestelmään.

JSON syntaksi koostuu hakasulkeista, aaltosulkeista, kaksoispisteistä ja pilkuista. JSON tarjoaa helpon tavan nimi/arvo –parien merkitsemiseen kuten LISTAUS 3 on esitetty (parit ”id”:2 ja ”name”:”Harry Potter and the Chamber of Secrets”). Melkein jokaisesta ohjelmointikielestä löytyy tietorakenne sitä varten, esimerkiksi `record`, `struct`, `dict`, `map`, `hash`, tai `object`. (Ecma International 2013, ii.)

LISTAUS 3. JSON-esimerkki (Vihavainen 2013, 6.3.1)

```
{
  "id":2,
  "name":"Harry Potter and the Chamber of Secrets"
}
```

Jokaisesta ohjelmointikielestä löytyy tietorakenne myös erilaisten listojen ja vektoreiden säilömiseen, esimerkiksi `array`, `vector` tai `list`. Myös JSON tukee niiden käyttöä. Koska oliot ja listat voidaan kapseloida, on myös monimutkaisempien tietorakenteiden, esimerkiksi `tree`, käyttäminen JSON:in avulla mahdollista. (Ecma International 2013, ii.)

LISTAUS 4 on esimerkki listasta kirjoja JSON-muodossa, huomaa erilaiset sulkeet.

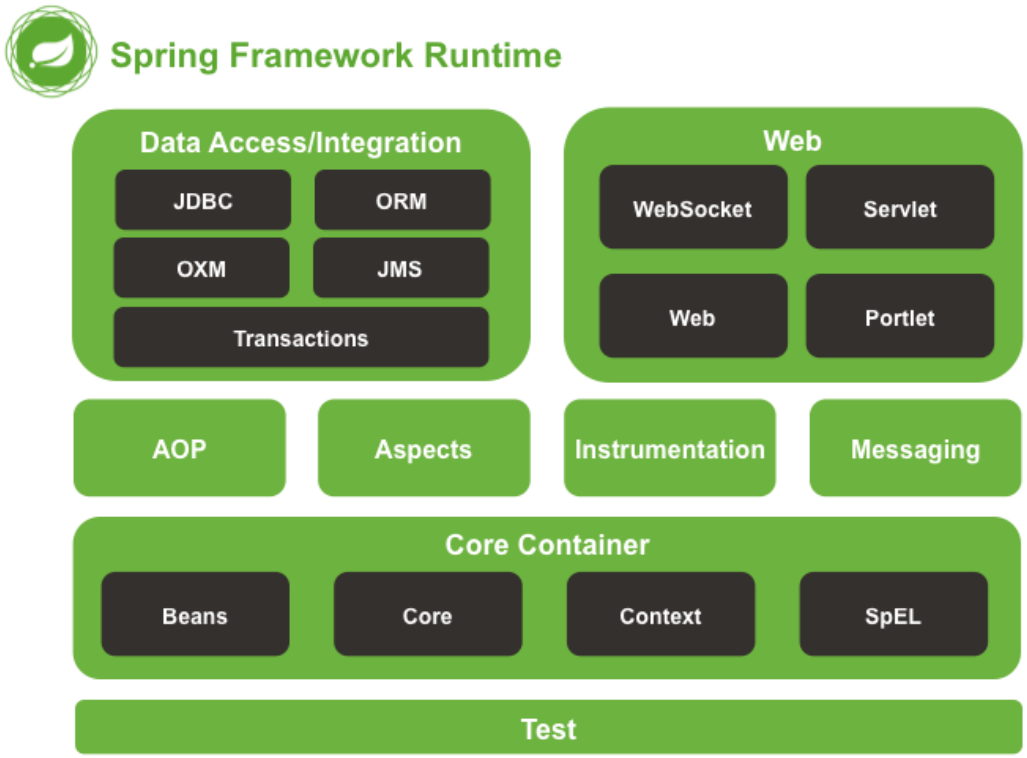
LISTAUS 4. JSON-lista

```
[
  {
    "id":1,
    "name":"Harry Potter and the Sorcerer's Stone"
  },
  {
    "id":6,
    "name":"Harry Potter and the Half-Blood Prince"
  },
  {
    "id":2,
    "name":"Harry Potter and the Chamber of Secrets"
  },
  {
    "id":7,
    "name":"Harry Potter and the Deathly Hallows"
  }
]
```

JSON:in yksinkertaisen formaatin ansiosta eri kielillä toteutetut ohjelmat voivat jakaa monimutkaisiakin tietorakenteita keskenään. Yksinkertaisuuden vuoksi myös formaatin muuttuminen on erittäin epätodennäköistä. (Ecma International 2013, ii.)

2.4 Spring-sovelluskehys

Spring-sovelluskehys on avoimen lähdekoodin sovelluskehys Java-sovelluksille ja se valittiin opinnäytetyön rungoksi. Sprign-sovelluskehys koostuu erilaisista komponenteista, ”moduuleista”. Koska komponentteja voi yhdistellä vapaasti tarpeen mukaan, on ohjelmien kehittäminen Springillä joustavaa. Komponentit, joita on noin 20 kappaletta, voidaan karkeasti jakaa seitsemään eri kerrokseen niiden käyttötarkoituksen mukaan (kuva 3). (Vohra 2012, 341.) Opinnäytetyössä käytettiin pääasiassa Web-, Context- ja Test-moduuleita, mutta sovelluskehys tarjoaa työkaluja paljon muuhunkin, kuten tietokantoihin ja rinnakkaisuuteen.



KUVA 3. Spring-sovelluskehyksen moduulit (Spring Framework Reference Documentation 2010-2014, 2.2 Modules)

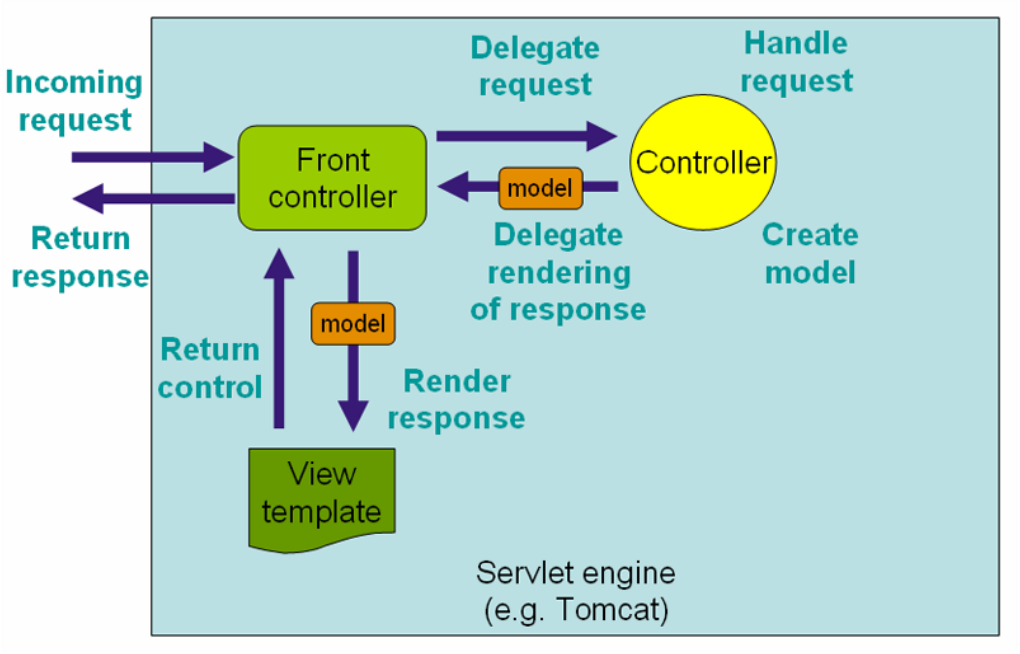
Spring-sovelluskehystä alettiin kehittää vuonna 2002 Java EE:n korvaajaksi, joka koettiin erittäin puutteelliseksi ja vaikeaksi käyttää. Ensimmäinen versio julkaistiin vuonna 2004 ja viimeisin versio Spring Framework 4.0 vuonna 2013. (Williams 2014, 359.)

Vaikka Spring sai alkunsa Java EE:stä, se ei rajoitu vain EE-ympäristöön. Spring-sovellus toimii millä tahansa Java EE Application Server:illä (esimerkiksi Oracle WebLogic Server), Java EE Web Container:issa (esimerkiksi Apache Tomcat, Jetty), pilvipalvelimessa (esimerkiksi Amazon EC2) tai erillisenä Java SE-sovelluksena. (Vohra 2012, 341; Williams 2014, 359.)

Spring-sovelluskehys tukee monia suosittuja web-tekniikoita kuten AJAX ja REST ja tietokantatekniikoita kuten JDBC, JPA, Hibernate ja JDO (Vohra 2012, 341). Vuosien aikana Spring Framework on laajentunut sovelluskehyksestä yhteisöksi ja poikinut monia sivuprojekteja kuten Spring Security, Spring Data, Spring Integration, Spring Batch, Spring Mobile, Spring for Android, Spring Social, Spring Boot, ja Spring.NET (Williams 2014, 359).

MVC (Model, View, Controller eli “malli”, ”näkymä”, ”käsittelijä”) on ohjelmistoarkkitehtuurityyli, joka jakaa sovelluksen kolmeen osaan. Jaon tarkoituksena on erottaa sovelluslogiikka käyttöliittymästä. (Vihavainen 2013, 4.7.1.) Palvelinohjelmissa käsittelijää vastaa luokka, jonka metodeille palvelin ohjaa pyynnöt (Vihavainen 2014, 3.1). Malli on komponenttien välillä kulkeva data ja näkymä vastaa käyttöliittymän ulkoasua. (Vihavainen 2013, 4.7.1).

Spring Web MVC on yksi Spring-sovelluskehyksen komponenteista (Moduuli Web kuvassa 3), joka tarjoaa tarvittavat apukirjastot MVC- ja REST-arkkitehtuuria noudattavan web-sovelluksen kehittämiseen. Spring MVC noudattaa niin sanottua ”Front Controller”-suunnittelumallia (kuva 4), jossa kaikki sovellukselle tulevat pyynnöt ohjataan ensin ”esikäsittelijälle”. (Spring Framework Reference Documentation 2014, 17.1-17.2.) Front Controller –suunnittelumallissa kaikille sovellukselle tuleville pyynnöille tarjotaan keskitetty esikäsittely. Springissä esikäsittelijää vastaa oletuksena ”DispatcherServlet”, joka kuuntelee sovellukselle tulevia pyyntöjä ja siirtää ne eteenpäin kehittäjän määrittämälle varsinaiselle käsittelijälle. Käsittelijä käsittelee pyynnön ja antaa vastauksen takaisin esikäsittelijälle, joka pyytää vastaukseen liittyvän näkymän ja palauttaa koko vastauksen. Keskitetty esikäsittely tuleville pyynnöille on hyödyllistä esimerkiksi käyttäjän oikeuksien varmistamisessa, jolloin varmistus tulee tehdä kaikille sovellukselle tuleville pyynnöille.



KUVA 4. Front Controller –suunnittelumalli (Spring Framework Reference Documentation 2014, 17.2)

Springin avulla voidaan komponenttien määrittelemisessä käyttää annotaatioita. Annotaatiot ovat lyhyitä, @-merkillä alkavia Java-kielisiä sanoja luokan, metodin tai parametrin edessä, jotka ohjaavat vahvasti sovelluksen toimintaa. Suurin osa hälytyspalvelussa käytetyistä annotaatioista kuuluu Spring-sovelluskehyksen Web-moduuliin ja muut, yleisemmät annotaatiot Context-moduuliin (kuva 3). Annotaatioiden avulla Spring tekee automaattisia riippuvuuksien latauksia ja konfigurointeja. Annotaatiot tarjoavat metatietoa esimerkiksi luokista, metodeista ja muuttujista. Esimerkiksi listauksessa 5 annotaatio @Controller luokan määrittelyn edellä kertoo, että luokka on MVC-mallin käsittelijä, joka kuuntelee HTTP-pyyntöjä. @RequestMapping(…) –annotaatio käsittelijäluokan metodin edessä määrittelee tarkemmin kuunneltavan osoitteen (”hello”) ja pyynnön tyypin (GET) . Käsittelijä on siten myös REST-arkkitehtuurin mukainen. (Vihavainen 2013, 5.2, 6.3; Kumar 2014, Java Annotations Tutorial with Custom Annotation Example and Parsing using Reflection.)

LISTAUS 5. Käsittelijän annotaatiot, esimerkki (Vihavainen 2013, 5.2)

```
@Controller
public class HelloWorldController {
  @RequestMapping(value="hello", method=RequestMethod.GET)
  public void helloWorld(…) {
    …
  }
}
```

Springin kotisivulta (http://spring.io/) löytyy runsaasti itseopiskelumateriaalia ja -oppaita, dokumentit, sekä linkit StackOverFlow:n Springiä koskeviin keskusteluihin ja kysymyksiin. Se toimii myös kehitystyökalujen ja projektien latauspaikkana.

2.5 Push-viesti mobiilisovellukselle

Hälytysten lähettämisessä TreLab-mobiilisovelluksen käytettiin push-viestejä. Push-viestillä tarkoitetaan viestiä, jonka palvelin lähettää asiakkaalle automaattisesti, joko reaaliaikaisesti tai säännöllisin väliajoin. Push-viestin vastakohta on ”Pull”-viesti, jolloin asiakas pyytää aina tiedon oma-aloitteisesti. Esimerkiksi kännykkään tulevat WhatsApp -viestit tai automaattisesti päivittyvät säätiedot ovat normaalisti push-viestejä. (PCMag 2014, Definition of push technology.) Push-viestejä voi verrata myös perinteisiin tekstiviesteihin sillä erotuksella että tekstiviesti ei vaadi erillistä sovellusta viestien vastaanottamiseen (Mobiilikehitys.fi 2013, Push-viestit ja push-palvelin – mitä ne ovat?). Pull-viesti on esimerkiksi internet-sivun avaaminen (PCMag 2014, Definition of push technology).

Päivitettyjen tietojen hakeminen palvelimelta mobiililaitteeseen on mahdollista toteuttaa tekemällä kyselyitä palvelimelle tasaisin väliajoin, pull-viestein. Tämä kuitenkin kuluttaa sekä akkua että mobiilidataa, ja kyselyt voivat olla turhia, jos palvelimella ei ole tarjota uudempaa tietoa. Käyttämällä push-viestejä uuden tiedon lähettämiseen laitteelle, minimoidaan käyttäjälle kalliin mobiilidatan käyttö ja säästetään akkua.

Push-viestien käyttö on yleistynyt älypuhelimien suosion myötä. Käyttäjä lataa mobiilisovelluksen ja hyväksyy push-viestien lähetyksen. Useissa sovelluksissa on mahdolli-

suus myös estää viestien vastaanotto. Push-viestillä voidaan lähettää erilaista sovellukseen liittyvää tietoa: kehotus päivittää sovellus uuteen, hälytys uudesta viestistä, tweetistä, kommentista, sähköpostista, mainos esimerkiksi erikoistarjouksesta tai tapahtumasta. Push-viesti voi siis toimia myös herätteenä hakea tietoa varsinaiselta palvelimelta. Viestit näkyvät mobiililaitteessa reaaliaikaisesti, vaikka sovellus ei olisi sillä hetkellä käytössä ja käyttäjän voidaan olettaa saaneen viestin heti. (Mobiilikehitys.fi 2013, Push-viestit ja push-palvelin – mitä ne ovat?).

Ensimmäisiä push-viestejä käytettiin ilmoittamaan käyttäjälle uudesta lukemattomasta sähköpostista. Ominaisuus toteutettiin ensimmäisenä BlackBerry-puhelimiin. Push-viestiominaisuus tuli Microsoftin Windows Mobile 5.0 –laitteisiin vuonna 2007, Applen iOS 3.0 –laitteisiin vuonna 2009 ja Googlen Android 2.2 –laitteisiin 2010. (kellabyte.com 2011, Push: The history, experience, cost and future.)

Applen iOS-laitteissa push-viesti, tai ”notifikaatio”, näkyy käyttäjälle pop-up-viestinä tai hieman pienempänä bannerina näytön yläreunassa. Jos käyttäjä ei reagoi viestiin, kerääntyvät ne luetteloksi näppäinlukon avaussivulle. Notifikaatiolla voidaan myös päivittää työpöydällä olevan kuvakkeen yläkulmassa näkyvää lukua. Erillisessä Notification Center:issä käyttäjä voi selata viestejä ja muokata push-viestien asetuksia. Notifikaation sisältö, painikkeiden määrä, äänimerkki ja viestiin liittyvä toiminto on kehittäjän muokattavissa. (Ritchie 2014, Interactive notifications in iOS 8: Explained.)

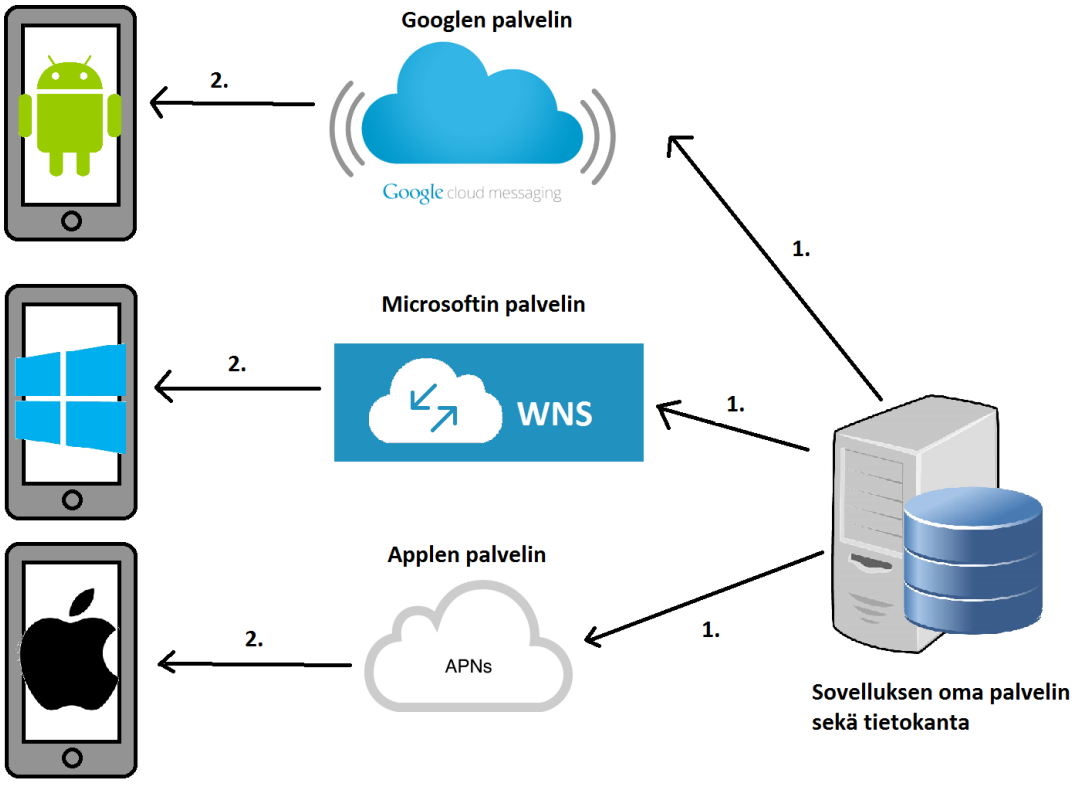
Windows Phone -laitteissa push-viestin tyypit ovat tile, toast ja raw. Äänimerkki ja puhelimen värinä viestin saapuessa on käyttäjän itse säädettävissä. Tile-tyyppinen viesti päivittää etusivulla olevan sovelluksen kuvakkeen, ”laatan”, tietoja. Laatassa on kaksi puolta ja se voi pyörähtää akselinsa ympäri säännöllisesti. Laatan kummallakin puolella on otsikko sekä taustakuva. Etupuolella vasemmassa yläkulmassa on luku ja takapuolella lyhyt lisäteksti. Nämä kaikki ovat päivitettävissä tile-viestillä. Toast on perinteinen, näytön yläreunaan ilmestyvä banneri, jossa on sovelluksen logo, otsikko ja teksti. (Windows Dev Center: Push notifications for Windows Phone 8 2014.) Raw-tyyppistä viestiä voi käyttää vapaammin. Sillä voi esimerkiksi käynnistää taustatoimintoja tai näyttää käyttäjälle suurempi pop-up –ikkuna, jossa on enemmän tekstiä kuin toastissa. Taustatoimintona voidaan esimerkiksi hakea lisää tietoa palvelimelta. (Windows Dev Center: Raw notification overview 2014.)

Push-viestin lähettäminen ja saapuminen Googlen Android-mobiililaitteisiin on kuvattu tarkemmin omassa luvussaan 2.5.2.

2.5.1 Push-viestin lähettäminen

Push-viestin lähettäminen mobiililaitteeseen vaatii laitteeseen asennettavan sovelluksen sekä sovellukseen liittyvän palvelimen kehittämisen. Sovelluksen palvelimen tehtävänä on huolehtia mobiilisovellukselle lähetettävästä tiedosta sekä sen lähettämisestä käyttöjärjestelmäkohtaiselle palvelimelle sen määrittelemällä tavalla. Jokaisella käyttöjärjestelmällä on hieman erilaisia tapoja näyttää saapunut push-viesti, mutta sen lähetystapa on periaatteessa sama (kuva 5) (Mobiilikehitys.fi 2013, Push-viestit ja push-palvelin – mitä ne ovat?):

1. push-viesti luodaan ja lähetetään sovelluksen palvelimelta käyttöjärjestelmän palvelimelle
2. käyttöjärjestelmän palvelin välittää viestin oikealle laitteelle
3. push-viesti saapuu sovellukselle.



KUVA 5. Push-viesti palvelimelta sovellukselle

Push-viestin lähetys Androidille tapahtuu Google Cloud Message (GCM) –palvelimen, iOS-sovellukselle Apple Push Notification Service (APNs) –palvelimen ja Windows-sovellukselle Windows Push Notification Service (WNS) –palvelimen kautta. Rajapinta sovelluksen oman palvelimen ja käyttöjärjestelmän palvelimen välillä (kuvassa 5 merkitty numerolla 1.) vaihtelee riippuen käyttöjärjestelmästä eikä niille ole yhteistä standardia. Kehittäjän onneksi jokaiselle käyttöjärjestelmälle löytyy valmiita avoimen lähdekoodin apukirjastoja ja esimerkkejä push-viestin lähettämiseen. Myös rajapinnat käyttöjärjestelmän palvelimen ja sovelluksen välillä vaihtelevat riippuen käyttöjärjestelmästä (kuvassa 5 merkitty numerolla 2.). (Android developers: Implementing GCM Server, 2014; iOS Developer Library: Provider Communication with Apple Push Notification Service, 2014; Windows Dev Center: Windows Push Notification Services (WNS) overview, 2014.)

Jokaisella käyttöjärjestelmällä on erilaisia tunnuksia ja sertifikaatteja, joiden avulla varmistetaan push-viestin kulkeminen oikealle laitteelle ja viestin lähettäjän aitous. Tunnuksien ylläpito on tärkeä osa palvelimen toimintaa ja niitä täytyy voida lisätä, poistaa ja päivittää kun uusia sovelluksia ladataan ja poistetaan. Jos palvelin yrittää lähettää paljon push-viestejä useille poistetuille sovelluksille, voi käyttöjärjestelmän palvelin lakata välittämästä viestejä. Sovelluspalvelimen täytyy usein myös yksilöidä lähetettävä viestinsä. (Android developers: Overview, 2014; iOS Developer Library: Apple Push Notification service, 2014; Windows Dev Center: Windows Push Notification Services (WNS) overview, 2014.)

Käytännössä push-viestien saapumisessa on usein viivettä. Käyttäjän mobiilidataverkko voi puuttua kokonaan tai katkeilla, jolloin viestin vastaanotto ei onnistu. Yksikään käyttöjärjestelmien palvelimista ei takaa viestin perillemenoa tai että lähetetyt viestit saapuvat perille oikeassa järjestyksessä. Kriittisen tai salaisen tiedon lähettämistä push-viestillä ei suositella. (Android developers: Advanced topics, 2014; iOS Developer Library: Apple Push Notification service, 2014; Windows Dev Center: Windows Push Notification Services (WNS) overview, 2014.)

2.5.2 Push-viestit Android-mobiililaitteeseen

Uusimmissa Android 4.4:ssa push-viestit näkyvät käyttäjälle ruudun ylälaitaan ilmestyvänä bannerina (kuva 6B). Normaalisti se sisältää kuvan (esimerkiksi sovelluksen logo tai viestin lähettäjän kuva), otsikon, lyhyen lisätekstin sekä kellonajan. Bannerin ulkoasun voi tehdä myös kokonaan itse ja siihen voi lisätä ominaisuuksia: bannerin voi esimerkiksi avata suuremmaksi, jolloin esiin tulee enemmän tekstiä, kuva tai painikkeita. Painikkeita voi lisätä enintään kolme. Push-viestille voi asettaa myös prioriteetin. Alimman prioriteetin viestejä ei näytetä ja niitä käytetään taustatoimintojen tukemiseksi. Suurimman prioriteetin viestit keskeyttävät minkä tahansa sovelluksen ilmestymällä päälimmäiseksi. Bannerissa voidaan näyttää myös tilapalkki, joka kuvaa esimerkiksi lataamista tai lähettämistä. Kehittäjä voi myös itse määritellä push-viestien ulkoasun, kuten kuvassa 6A, jossa on vastaanotettu yksi viesti Facebookin Messengerissä. Lähettäjän profiilikuva sekä viestien lukumäärä näkyvät oikeassa reunassa kaikkien sovellusten päällä, kunnes kuvaketta painetaan ja viesti avataan. (Android Developers: Design/Patterns/Notifications & Develop, API Guides, User Interface, Notifications).

KUVA 6. Push-viesti Androidiin. Kuva A: ilmoitus Facebookissa vastaanotetusta viestistä. Kuva B: ilmoitus WhatsApp –sovelluksen kautta vastaanotetusta viestistä.

Saapuneet mutta lukemattomat push-viestit kerätään näppäinlukon avaus –näkymään sekä yläreunasta alas vedettävään valikkoon. Saapunut viesti voidaan myös ilmaista vilkuttamalla laitteessa olevaan pientä lediä laitteen näytön ollessa pois päältä. Värinä- ja äänihälytys ovat käyttäjän hallittavissa. (Android Developers: Design/Patterns/Notifications & Develop/API Guides/User Interface/Notifications).

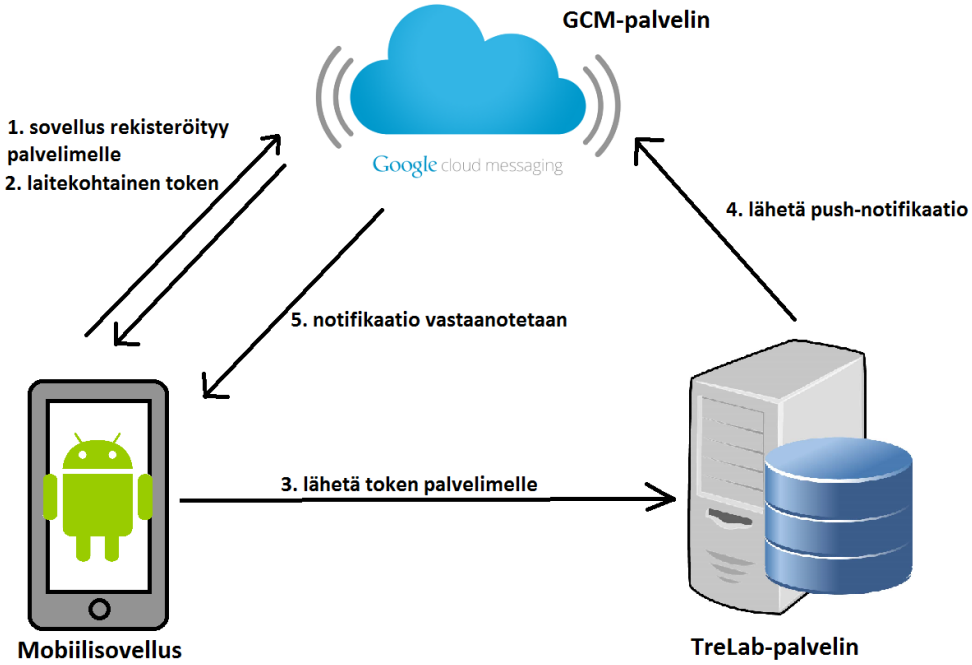
Google asettaa sovelluksen palvelimelle kriteerit, joita noudattamalla viestin lähettäminen onnistuu:

- mobiilisovelluksen ja sovelluksen palvelimen täytyy pystyä kommunikoimaan (puhelimen täytyy lähettää palvelimelle esimerkiksi sovelluksen ja laitteen yksilöivä tunnus, eli token)
- palvelimen täytyy pystyä lähettämään push-viestit määrittelyn mukaisesti GCM-palvelimelle
- palvelimen tulee reagoida mahdollisiin virheilmoituksiin lähetyksen epäonnistuessa
- palvelimen täytyy pystyä lähettämään push-viesti uudelleen tietyn ajan kuluttua (”exponential backoff”), jos lähetys epäonnistuu ensimmäisellä kerralla
- palvelimen täytyy pystyä varastoimaan mobiilisovelluksen tunnus eli ”API key” sekä jokaisen ladattavan sovelluksen yksilöivät tunnukset eli ”tokenit”
- jokainen viesti tulee yksilöidä.

Jotta kriteerit täyttävän sovelluspalvelimen tekeminen olisi helpompaa, Google tarjoaa paljon esimerkkejä ja apukirjastoja eri kielillä (Java, Python). (Android Developers, Implementing GCM Server.)

Palvelimen kehittäminen alkaa rekisteröitymällä ja luomalla tarvittavat tunnukset Googlen kehittäjille suunnatussa palvelussa. Rekisteröitymisen yhteydessä saatu ”API key” tallennetaan sovelluksen palvelimelle. Kun käyttäjä avaa mobiilisovelluksen ensimmäistä kertaa, rekisteröidään sovellus ja laite palvelimelle, ja palvelin palauttaa laitekohtaisen tunnuksen, ”token” (kuva 7, vaiheet 1. ja 2.). Token lähetetään sovelluksen omalle palvelimelle (kuva 7, vaihe 3), jonne se tallennetaan. Push-viestin lähetyksessä API key:n ja token lisätään viestiin (kuva 7, vaihe 4.). API key:n avulla GCM-palvelin

tunnistaa viestin lähettävän palvelimen ja token kertoo laiteen ja sovelluksen, jolle viesti tulee välittää.



KUVA 7. Mobiilisovelluksen rekisteröistyminen ja push-viestin lähettäminen. Vaiheet 4. ja 5. löytyvät myös kuvasta 5.

Push-viestin lähettämiseen Google tarjoaa kaksi palvelinta, ”GCM HTTP Connection server” tai ”GCM Cloud Connection server” eli CCS, joiden rajapinnat ja toiminta ovat hieman erilaisia. HTTP–palvelin on hieman yksinkertaisempi: se ei tue kaksisuuntaista viestintää sovelluksen ja palvelimen välillä ja on synkroninen (viestin lähettämisen jälkeen jäädään odottamaan vastausta eikä toista viestiä voida lähettää ennen). Push-viestit lähetetään GCM HTTP –palvelimelle käyttäen HTTP POST-pyyntöä. CCS käyttää XMPP-protokollaa ja mahdollistaa nopean kaksisuuntaisen viestinnän sovelluksen ja palvelimen välillä. Sovellus voi esimerkiksi vastata viestiin heti käyttäen samaa yhteyttä ja palvelin saa tiedon push-viestin onnistuneesta tai epäonnistuneesta lähetyksestä asynkronisesti. Molemmat Googlen palvelimet käyttävät JSON-muotoista sisältöä push-viestin lähettämiseen. (Android Developers: Implementing GCM Server; GCM HTTP Connection Server; GCM Cloud Connection Server (XMPP).)

3 HÄLYTYSPALVELU JAVA-PALVELIMEEN

Kolmannessa luvussa käydään läpi EAS:in käytännön toteutusta ja esitellään valittuja tekniikoita sekä järjestelmän suunnittelu- ja toteutusratkaisuja. TreLabin sekä GCM-palvelimen asettamat vaatimukset määrittelivät hyvin pitkälle käytetyt työkalut sekä millainen hälytyspalvelusta tulisi.

3.1 Työkalut

Luvussa 3.1 on kerrottu opinnäytetyön keskeisimmistä työkaluista. Spring Tool Suite – kehitysympäristö sekä Git-versionhallintatyökalu olivat tuttuja jo aikaisemmista pienemmistä ohjelmistoprojekteista. Luvun pääpaino on siten Maven-projektinhallintatyökalussa, sillä sen käyttö ei ollut ennestään tuttua ja vaati eniten perehtymistä.

3.1.1 Kehitysympäristö Spring Tool Suite

Spring Tool Suite (usein lyhennetty STS) on avoimen lähdekoodin kehitysympäristö kaikenlaisten Java-sovellusten kehittämistä varten. Sen runkona on Java-ohjelmoinnissa paljon käytetty Eclipse, jota on muokattu sopimaan paremmin Spring-sovelluskehyksen käyttämiseen. Spring Tool Suite tarjoaa valmiin ympäristön toteutukseen, testaukseen, ajamiseen ja käyttöönottoon ja siihen on valmiiksi integroitu esimerkiksi Pivotal tc Server, Pivotal Cloud Foundry, Git, Maven ja AspecJ. Kuten Eclipseenkin, Spring Tool Suiteen on ladattavissa paljon lisäosia ja työkaluja.

Spring Tool Suite valittiin opinnäytetyön kehitysympäristöksi, koska muutkin TreLabin kehittäjät käyttivät sitä. Heiltä pystyi kysymään neuvoa ongelmatilanteissa eikä kaikkea tarvinnut opiskella itse. Opinnäytetyön varsinainen tekeminen alkoi Spring Tool Suiten lataamisella ja Spring-sovelluskehykseen tutustumisella tutoriaalien kautta. Koska Eclipsen käyttö oli tuttua koulun projektikursseilta, ei Spring Tool Suiten käyttämisen opettelu ollut vaikeaa.

3.1.2 Maven-projektinhallintatyökalu

Maven on Apache Software Foundationin kehittämä ohjelmistoprojektin hallintatyökalu. Mavenia voidaan käyttää missä tahansa Java-projektissa ja se perustuu avoimeen lähdekoodiin. (What is Maven? 2014.)

Aluksi Maven kehitettiin "Jakarta Turbin"-projektia varten. Haluttiin työkalu, joka helpottaisi käännöstyötä ja antaisi selkeä määritelmän projektin riippuvuuksista ja jonka avulla projektin perustietojen julkaisu ja JAR-tiedostojen jako eri projektien välillä olisi helpompaa. Projektin kotisivuilla (http://maven.apache.org/what-is-maven.html) on Mavenin tavoitteiksi listattu:

* käännösprosessin helpottaminen
* tarjota yhdenmukaisen menetelmän kääntämiseen ja koostamiseen
* tarjota laadukasta tietoa projektista
* tarjota ohjeita kehitystyöhön
* mahdollistaa selkeän ja avoimen tavan tehdä muutoksia ja lisäyksiä projektiin. (What is Maven? 2014).

Maven perustuu ajatukselle, että jokaisella ohjelmistoprojektilla on samanlainen elinkaari: jokaisen projektin kehitystyö sisältää samat vaiheet. Mavenissa jokainen vaihe on oikeastaan oma liitännäisensä. Liitännäisiä voidaan lisätä ja poistaa riippuen projektista ja sen tarpeista. Esimerkiksi oletus-elinkaari koostuu seuraavista vaiheista:

1. `validate`: Tarkistaa, että kaikki projektin tiedot ovat oikein ja saatavilla
2. `compile`: Kääntää projektin lähdekoodit
3. `test`: Ajaa testit
4. `package`: Pakkaa käännetyn koodin haluttuun muotoon
5. `integration-test`: Suorittaa integrointitestit asetetussa ympäristössä
6. `verify`: Tarkistaa, että paketointi onnistui ja täyttää laatuvaatimukset
7. `install`: Asentaa paketin paikallisesti
8. `deploy`: Kopioi paketin ulkoiseen repositoryyn

Jokainen vaihe sisältää myös edelliset vaiheet, esimerkiksi suorittamalla vaiheen 7. install, suoritetaan myös kaikki edelliset vaiheet listasta (validate, compile, test, package, integration test sekä verify). (Introduction to the Build Lifecycle 2014.)

Mavenin avulla voi myös luoda projektin tiedostorakenteen rungon käyttämällä valmiita standardoituja malleja nimeltään ”archetypes” (suomeksi ”arkkityyppi” tai ”malliesimerkki”). Myös oman arkkityypin luominen on mahdollista. Kun kaikki projektit käyttävät samaa valmista tiedostorakennetta, on projektista toiseen siirtyminen ja uuden projektin ymmärtäminen ja omaksuminen helpompaa ja nopeampaa. (Introduction to Archetypes 2014.)

Perinteisesti Mavenia käytetään komentoriviltä. Eclipseen ja Spring Tool Suite –kehitysympäristöihin on kuitenkin ladattavissa Maven-lisäosa. Opinnäytetyön aloitettiin lataamalla Maven-lisäosa ja käyttämällä Mavenin ”webapp”-arkkityyppiä luotiin listauksen 6 mukainen tiedostorakenne EAS:in rungoksi. Webapp-arkkityyppi valittiin, koska sitä käytettiin web-sovellus –tutoriaaleissa, joista otettiin aluksi mallia.

Opinnäytetyön edetessä päädyttiin kuitenkin noudattamaan Mavenin quickstart-arkkityypin kaltaista tiedostorakennetta (listaus 7), kun jsp-sivujen ja xml-tiedostojen käytöstä voitiin luopua ja koska muut taustajärjestelmän projektit noudattivat quickstart-arkkityypin mukaista tiedostorakennetta. Tiedostorakennetta voi siis vapaasti muuttaa koska tahansa. Mavenin arkkityypit on tarkoitettu vain nopeuttamaan ja helpottamaan projektin aloittamista ja pitämään tiedostorakenteen eri projektien välillä samanlaisina.

LISTAUS 6. Maven webapp tiedostorakenne

```
project
|-- pom.xml
`-- src
    `-- main
        `-- webapp
            |-- WEB-INF
            |   `-- web.xml
            `-- index.jsp
```

LISTAUS 7. Maven quickstart tiedostorakenne

```
project
|-- pom.xml
`-- src
    |-- main
    |   `-- java
    |       `-- App.java
    `-- test
        `-- java
            `-- AppTest.java
```

Jokaiseen Maven-projektiin liittyy konfiguraatiotiedosto nimeltään pom.xml. Lyhenne POM tulee sanoista Project Object Model. Tiedosto on XML-muotoinen ja siinä määritellään mm. projektin nimi ja tunnus, versio, tyyppi, asetukset, käytetyt repositoryt, riippuvuudet, resurssit ja liitännäiset. Kaikki kääntämisen ja ajamisen kannalta oleellinen tieto projektista on yhdessä tiedostossa ja esimerkiksi uuteen projektiin tutustuessa pom.xml -tiedoston avulla voi saada kattavan yleiskuvan projektista ja sen riippuvuuksista. (Turatti & Pillitu 2013, 8-17.) pom.xml-tiedostoon on siis koottu sovelluksen ulkoiset riippuvuudet ja ominaisuudet, se ei kerro mitään sovelluksen sisäisistä riippuvuuksista tai toiminnasta. Listauksessa 8 on typistetty esimerkkinä opinnäytetyön pom.xml-tiedoston sisältö ja rakenne.

LISTAUS 8. pom.xml

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/maven-v4_0_0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>fi.trelab</groupId>
  <artifactId>eas</artifactId>
  <packaging>war</packaging>
  <version>0.1.7</version>
  <name>eas</name>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <spring.version>3.2.2.RELEASE</spring.version>
    <springsecurity.version>3.1.4.RELEASE</springsecurity.version>
    <GCMserver.version>1.0.2</GCMserver.version>
    <jackson.version>2.2.3</jackson.version>
  </properties>

  <repositories>
    <repository>
      <id>gcm-server-repository</id>
      <url>
        https://github.com/slorber/gcm-server-repository/raw/master/
      </url>
    </repository>
  </repositories>

  <dependencies>

    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-webmvc</artifactId>
      <version>${spring.version}</version>
    </dependency>

    <dependency>
      <groupId>com.google.android.gcm</groupId>
      <artifactId>gcm-server</artifactId>
      <version>${GCMserver.version}</version>
    </dependency>
```

```
    <dependency>
      <groupId>com.fasterxml.jackson.core</groupId>
      <artifactId>jackson-databind</artifactId>
      <version>${jackson.version}</version>
    </dependency>

    <dependency>
      <groupId>javax.servlet</groupId>
      <artifactId>javax.servlet-api</artifactId>
      <version>3.0.1</version>
      <scope>provided</scope>
    </dependency>

    <dependency>
      <groupId>com.google.collections</groupId>
      <artifactId>google-collections</artifactId>
      <version>1.0-rc2</version>
    </dependency>

    <!-- logging -->
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
    </dependency>

    <!-- security -->
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-web</artifactId>
      <version>3.0.0.RELEASE</version>
      </dependency>
    <dependency>
      <groupId>org.springframework.security</groupId>
      <artifactId>spring-security-config</artifactId>
      <version>3.2.1.RELEASE</version>
    </dependency>

    <!-- testing dependecies -->
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-test</artifactId>
      <version>${spring.version}</version>
      <scope>test</scope>
    </dependency>
  </dependencies>

  <build>
    <finalName>${artifactId}</finalName>
    <defaultGoal>package</defaultGoal>

    <resources>
      <resource>
        <directory>src/main/resources</directory>
        <filtering>true</filtering>
      </resource>
```

```
    </resources>

    <plugins>
      <plugin>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.7</source>
          <target>1.7</target>
        </configuration>
      </plugin>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-war-plugin</artifactId>
        <version>2.3</version>
        <configuration>
          <failOnMissingWebXml>false</failOnMissingWebXml>
        </configuration>
      </plugin>
    </plugins>
  </build>
</project>
```

pom.xml-tiedostossa on oma osionsa sovelluksen ulkopuolisille riippuvuuksille, ”<dependencies>”, jonka sisälle on listattu yksittäisiä kirjastoja, ” <dependency>”. Kirjaston versionumero on merkitty <versio>-osiossa joko suoraan (esimerkiksi <version>1.0-rc2</version>) tai muuttujan kautta (esimerkiksi <version>${spring.version}</version>). Riippuvuuden ”<groupId>” kertoo riippuvuuden sijainnin ja ”<artifactId>” varsinaisen nimen. Tietojen perusteella voidaan siten etsiä oikea kirjasto.

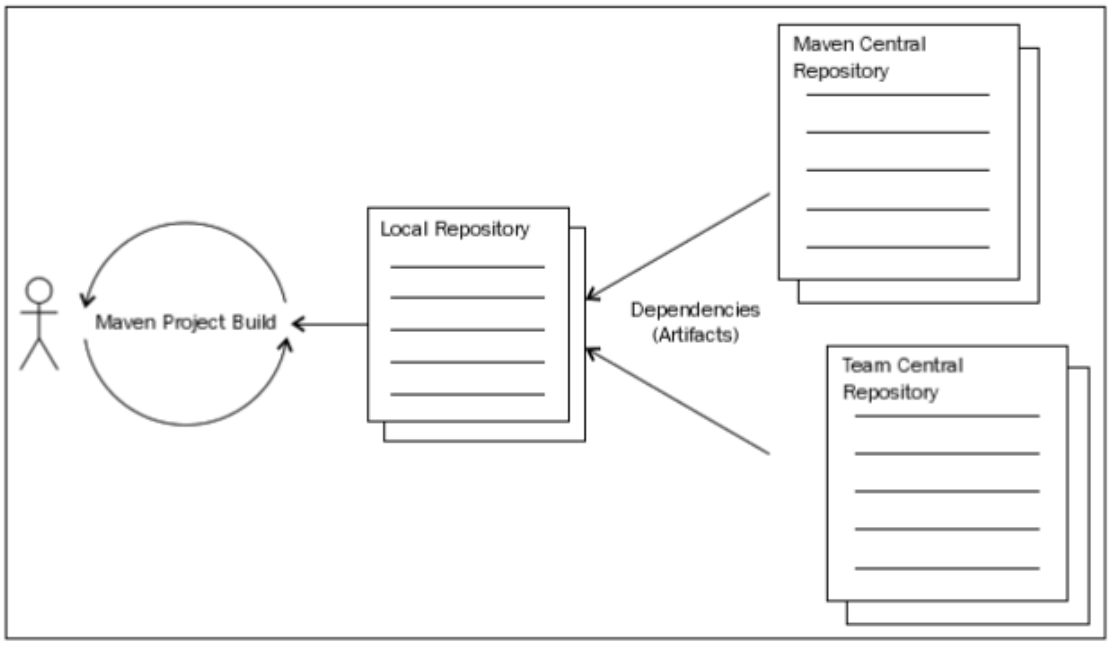
Kirjastoja voi projektista riippuen olla jopa satoja ja Mavenin tarkoituksena on auttaa hallinnoimaan niitä. Esimerkiksi jos ladattava kirjasto riippuu muista kirjastosta, lataa Maven ne automaatisesti. (Iyer 2011, 31.) Riippuvuuksien hallinnoinnin idea on, että käytetyt kirjastot haetaan automaattisesti keskitetyistä lähteistä, repositoryistä, eikä niitä ladata erillisiksi paikallisiksi kopioiksi. Näin voidaan esimerkiksi varmistaa, että kaikki kehittäjät tai projektit käyttävät samaa versiota kirjastosta. (Iyer 2011, 48.)

Mavenilla on muutama oletus-repository, josta kirjastoja haetaan, esimerkiksi Maven Repository (http://mvnrepository.com/) ja The Central Repository (http://search.maven.org/), mutta niitä voi myös lisätä itse (kuva 8). (Iyer 2011, 48.) Esimerkiksi EAS:n toteutuksessa käytetiin googlen valmista push-viestien lähettämiseen tarkoitettua kirjastoa gcm-server, mutta kirjastoa ei löytynyt oletus-repositoryistä vaan GitHubista (https://github.com/slorber/gcm-server-repository). Repository piti lisätä itse pom.xml-tiedostoon (listaus 9) kohtiin repositories ja dependencies.

LISTAUS 9. GCM-server pom.xml-riippuvuus

```
<repository>
    <id>gcm-server-repository</id>
    <url>
        https://github.com/slorber/gcm-server-repository/raw/master/
    </url>
</repository>
```

```
<dependency>
    <groupId>com.google.android.gcm</groupId>
    <artifactId>gcm-server</artifactId>
    <version>${GCMserver.version}</version>
</dependency>
```



KUVA 8. Riippuvuuksien lataaminen (Iyer 2011, 48)

pom.xml-tiedoston properties-kohdassa on koottu yhteen ladattavien riippuvuuksien versionumeroita. Versionumeroihin on dependencies-kohdassa viitattu muutujina. Esimerkiksi

```
<jackson.version>2.2.3</jackson.version>
```

ja dependencies-osiossa

```
<version>${jackson.version}</version>.
```

Tämä ei ole pakollista, mutta kun versionumerot kokoaan yhteen heti pom.xml-tiedoston alkuun, ei koko tiedostoa tarvitse selata läpi kun versiota halutaan muuttaa.

Riippuvuuksien hallinnan, projektirungon luomisen ja automatisoidun käännösprosessin lisäksi Mavenilla on muitakin ominaisuuksia, esimerkiksi raportointia ja staattista koodianalyysiä varten löytyvät omat liitännäiset. (Available Plugins 2014.)

3.1.3 Git-versionhallinta

Git on avoimen lähdekoodin versionhallintatyökalu. Perinteisesti Gitiä käytetään komentoriviltä, mutta koska TortoiseSNV oli aikaisemmista kouluprojekteista jo tuttu, käytin Gitiä TortoiseGitin avulla. TortoiseGit tarjoaa graafisen käyttöliittymän versionhallintaan ja projektin tiedostojen tila on nähtävissä Windowsin tiedostoselaimessa kuvakkeen päällä olevista merkeistä. Opinnäytetyölle luotiin jo projektin alussa oma repository.

3.2 Suunnittelu ja rajapinnat

Opinnäytetyön alussa TreLab pyysi alustavaa suunnitelmaa, johon kuului käyttötapauskuvaukset ja luokkakaavio, ja jonka pohjalta EAS:ia voisi lähteä toteuttamaan. Rajapintoja palvelimen muihin osiin ei ollut vielä tässä vaiheessa tarkasti määritelty.

Käyttötapauskuvaus oli listaus erilaisista tilanteista ja miten hälytyspalvelun tulisi reagoida niihin. Esimerkiksi tapaus ”Uusi hälytys”:

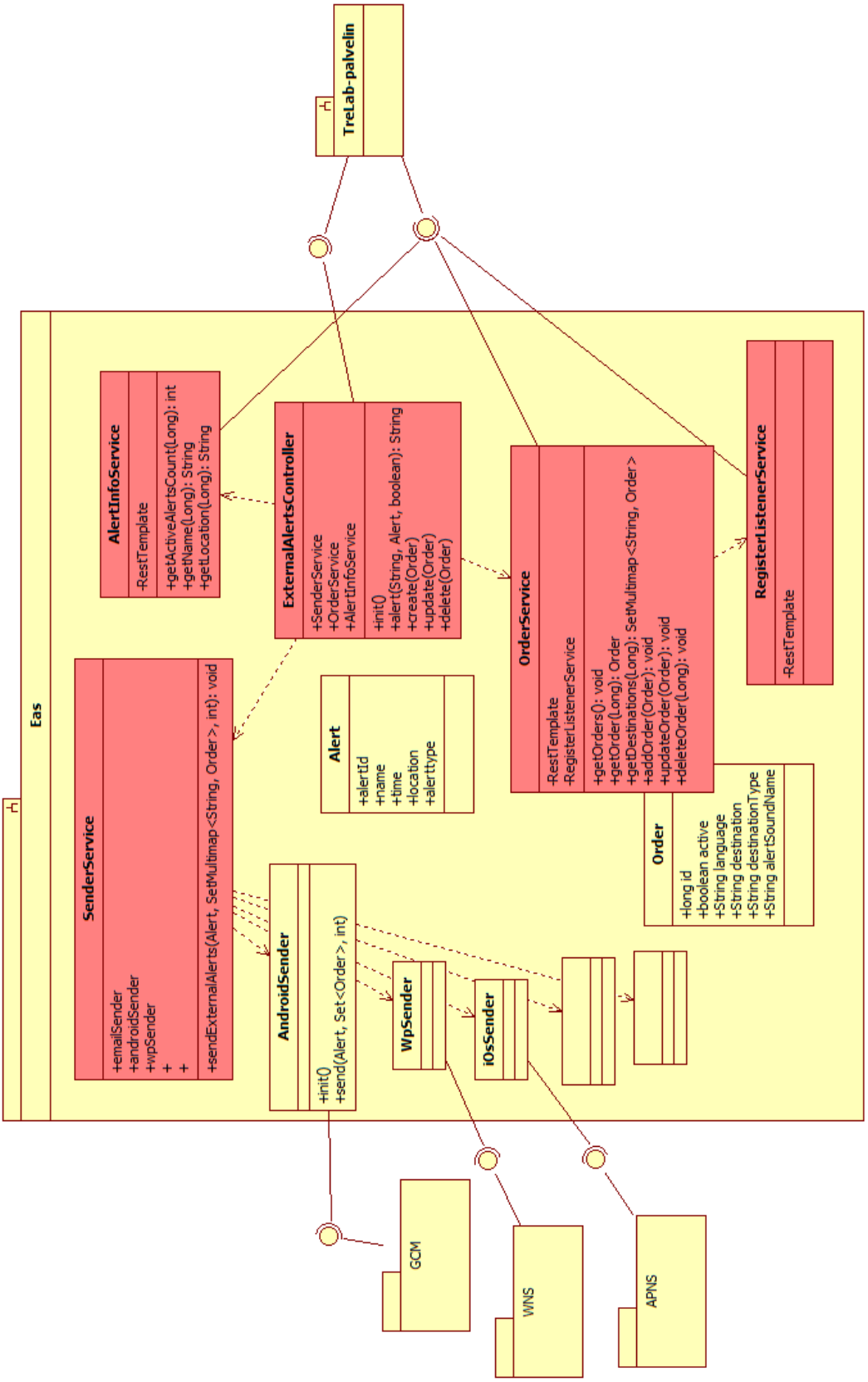
1. vastaanotetaan tiedot hälytyksestä
2. pyydetään yhteystiedot kenelle pitää hälyttää
3. lähetetään push-viesti, päivitetään myös kuittaamattomien hälytysten lukumäärä
4. lähetys onnistui, kirjataan lokiin tieto

Käyttötapauskuvauksen voisi tehdä myös piirtämällä kaavioita, mutta yksinkertainen lista tekstitiedostossa oli nopea ja helppo toteuttaa ja ylläpitää.

Luokkakaavio hälytyspalvelun sisäisestä toiminnasta oli aluksi hyvin suurpiirteinen ja sitä korjattiin useaan kertaan kun opittiin uutta ja rajapinnat tarkentuivat. Lopullinen luokkakaavio on kuvassa 9. Luokkakaavioon on koottu vain toiminnan kannalta oleellisimmat luokat kuvan selkeyttämiseksi. AndroidSender-luokan alapuolella olevat luokat kuvaavat vaihtoehtoisia hälytystapoja, joita ei kuitenkaan vielä toteutettu. Taulukkoon 1 on kerätty luokkakaaviossa esitettyjen luokkien vastuualueet ja tehtävät lyhyesti. Luokkakaavion lisäksi opinnäytetyön esittelyä sekä dokumentointia varten piirrettiin sekvenssikaavioita hälytyspalvelun keskeisimmistä toiminnoista.

TAULUKKO 1. Luokat ja niiden vastuualueet

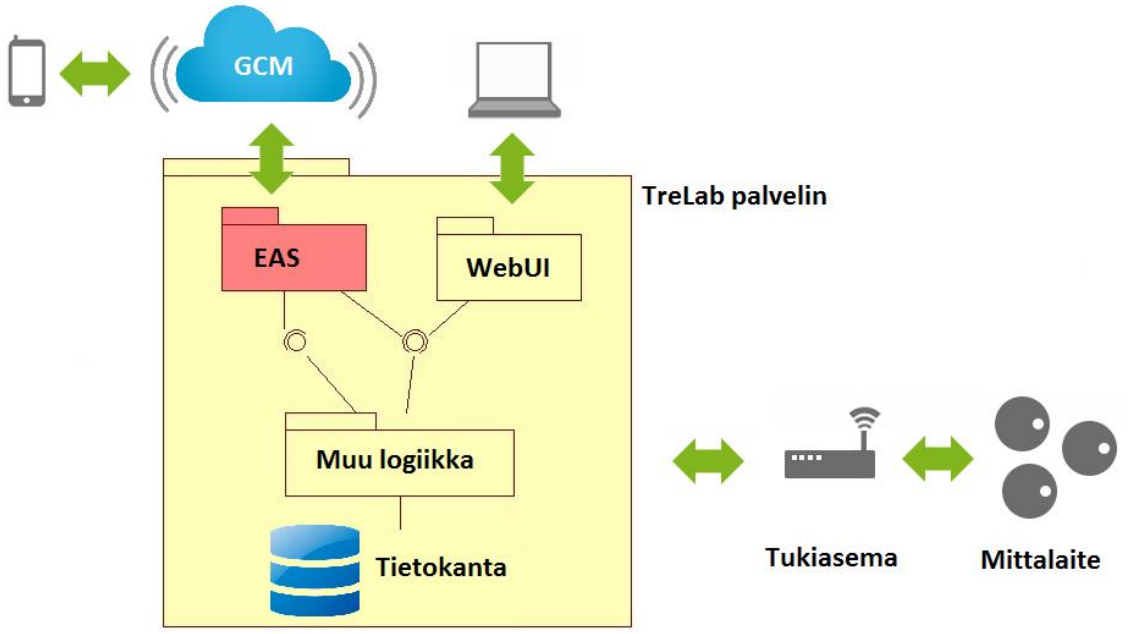
ExternalAlertsController	Hälytyspalvelun käsittelijä, ”kontrolleri”
Order	”Malli”, eli luokka oliolle, joka sisältää tiedot yhdestä asiakkaasta. Nimitys ”Order” viittaa yhteen hälytystilaukseen.
OrderService	Palvelu, joka huolehtii hälytystilauksista ja niiden ylläpidosta.
SenderService	Palvelu, joka huolehtii hälytyksen ja yhteystietojen ohjaamisesta oikean käyttöjärjestelmän palvelimelle.
Alert	”Malli”, eli luokka joka sisältää tiedot yhdestä hälytyskäskystä.
AlertInfoService	Palvelu, joka pyytää hälytyksestä puuttuvat lisätiedot muualta taustajärjestelmästä.
RegisterListenerService	Palvelu, joka luo tarvittavat hälytyskuuntelijat jokaiselle Order-oliolle taustajärjestelmään.
AndroidSender	Palvelu huolehtii varsinaisen hälytyksen lähettämisestä kaikille parametreissä annetuille Android-käyttöjärjestelmää käyttäville asiakkaille GCM-palvelimen kautta.



KUVA 9. Hälytyspalvelun luokkakaavio

Kuvassa 10 näkyy hälytyspalvelun sijoittuminen suhteessa muihin palveluihin ja sen rajapinnat sekä koko ketju mittalaitteelta mobiilisovellukselle. Yhteydet muuhun taustajärjestelmään ja mobiililaitteiden käyttöjärjestelmien palvelimille toteutettiin REST-rajapintoina, eli HTTP:n POST, PUT, GET ja DELETE –pyyntötyypeillä ja JSON-muotoisella viestirungolla. Mobiilisovellukset olivat jo valmiiksi toteutettuja ja valmiita vastaanottamaan push-viestejä, vain yhteys eri käyttöjärjestelmien palvelimille puuttui.

EAS:in vastuualueisiin ei kuulu tietokannat tai graafinen käyttöliittymä, käyttäjälle hälytyspalvelu näkyy vain saapuvana viestinä. Kaikki tarvittavat tiedot push-viestin lähettämiseen saatiin taustajärjestelmän muista palveluista eikä suoria tietokantahakuja tehty.



KUVA 10. Hälytyspalvelu EAS ja rajapinnat

Opinnäytetyön alussa rajapinta muuhun taustajärjestelmään ei ollut kokonaan määritelty. Projektin edetessä sitä tarkennettiin ja muutettiin yhteistyössä muiden taustajärjestelmän kehittäjien kanssa. Rajapinnassa hyödynnettiin jonkin verran jo valmiina olevan muun taustajärjestelmän ja web-käyttöliittymän välistä rajapintaa (kuva 10).

3.3 Kontrolleri, palvelut ja kuuntelijat

EAS:in toiminta on jaettu käsittelijään ja palveluihin. Niiden lisäksi on luokka jokaista hälytystyyppiä varten. Käsittelijä kontrolloi toimintaa käyttämällä erilaisia palveluita apunaan ja ohjaa lopuksi hälytyksen lähetettäväksi tarvittaville luokille.

EAS:in alustus alkaa hälytettävien asiakkaiden yhteystietojen hakemisella taustajärjestelmästä, jonka suorittaa yhteystiedoista huolehtiva OrderService-luokka HTTP-pyynnöllä GET. Vastauksena tulleesta JSON-muotoisesta datasta parsitaan yhteystiedot, jonka jälkeen jokaiselle yhteystiedolle luodaan hälytyskuuntelija, eli taustajärjestelmälle ilmoitetaan kaikki niiden asiakkaiden tunnukset, joihin liittyvät hälytykset tilataan hälytyspalvelulle. Kuuntelijoiden luomisesta ja poistamisesta huolehtii luokka RegisterListenerService.

Alustuksen jälkeen hälytyspalvelu jää odottamaan käsittelijälle tulevia hälytyksiä. Se ei siis aktiivisesti tee itse mitään ja toimii vasta kun muusta taustajärjestelmästä tulee käsittelijälle hälytys tai päivitys asiakkaan yhteystietoihin.

EAS:in tärkein luokka on käsittelijä, ExternalAlertsController. Käsittelijä vastaanottaa tulevat hälytykset, päivityksen yhteystietoihin ja ohjaa koko palvelun toimintaa. Käsittelijä sisältää metodeja, joihin Springin esikäsittelijä eli DispatchServlet-palvelin ohjaa tulevat HTTP–pyynnöt. Jotta palvelin tietää, mille metodille pyyntö ohjataan, on jokaisella metodilla oma polku sekä HTTP-pyyntötyyppinsä. Esimerkiksi listaus 10 on ExternalAlertsControllerin metodi ”create”, joka on tarkoitettu uuden hälytystilauksen luomista varten. Annotaatiolla @RequestMapping kerrotaan metodin polku (value=”alertrequest/{id}”) ja HTTP–pyynnön tyyppi (method=RequestMethod.POST). Jos EAS:in osoite olisi esimerkiksi http://192.168.2.2:8080/eas/, niin HTTP POST–pyyntö osoitteeseen http://192.168.2.2:8080/eas/alertrequest/87 ohjautuisi käsittelijän create-metodille.

LISTAUS 10. käsittelijän create-metodi

```
@RequestMapping(value="alertrequest/{id}", method=RequestMethod.POST)
@ResponseBody
public ResponseEntity<String> create( @PathVariable Long id, @Valid
@RequestBody Order newOrder)
{
    logger.debug("New order " + id + " received");

    Order o = orderService.getOrder(id);
    if( o != null )
        return new ResponseEntity<String>(HttpStatus.CONFLICT);

    orderService.addOrder(newOrder);
    logger.debug(orderService.allRecipientsAsString());

    return new ResponseEntity<String>(HttpStatus.CREATED);
}
```

Käsittelijä vastaanottaa pyyntöjä muilta, mutta EAS:in pyynnöt muulle taustajärjestelmälle hoidetaan palveluissa. HTTP-pyyntöjen tekemiseen muille käytetään Springin RestTemplate-luokkaa. Listauksessa 11 on esimerkki AlertInfoService:n eräästä palvelusta, joka pyytää kuittaamattomien hälytysten lukumäärän taustajärjestelmästä. RestTemplate-luokan getForObject-metodi tekee HTTP GET-pyynnön parametrina annettuun osoitteeseen (url) ja muuntaa vastauksen parametrina annettuna oliona (Integer) (Spring Framework 4.1.1. Release API, RestTemplate).

LISTAUS 11. AlertInfoService-palvelun getActiveAlertsCount-metodi

```
public int getActiveAlertsCount(long id) {

    String url = hlUrl + "alertcount?id=" + id;
    Integer count = resttemplate.getForObject( url, Integer.class );

    logger.debug("Id " + id + ", active alerts: " + count);
    return count;
}
```

Jotta muuhun taustajärjestelmään tehtyjen HTTP GET-pyyntöjen määrä olisi mahdollisimman pieni, käytettiin usein samanlaisena toistuvien kyselyjen tietojen tallentamiseen välimuistia. Pyynnön tekevän metodin edessä käytettiin tarkoitukseen sopivaa annotaatiota @Cacheable. Metodin palautusarvo asetetaan välimuistiin ja jos metodia kutsutaan samoilla parametreilla uudestaan, ei metodia suoriteta vaan palautettava arvo noudetaan välimuistista. Välimuisteja voi olla useita ja ne voidaan myös tyhjentää käyttämällä annotaatiota @CacheEvict. (Vihavainen 2014, 12.4.)

3.4 Käytetyt annotaatiot

Taulukossa 2 on listattu hälytyspalvelussa käytetyt annotaatiot ja lyhyt selitys niiden käyttötarkoituksesta. Lähteinä on käytetty Vihavaisen oppimateriaalia (2013, 2014) sekä Spring Framework 4.1.1. Release API –dokumentaatiota. Suurin osa taulukossa listatuista annotaatioista löytyy Spring Frameworkin Web- ja Context-moduuleista, mutta esimerkiksi @Autowired ja @Value löytyvät Beans-moduulista (kuva 1). Annotaatioiden käyttäminen osoittautui erittäin tehokkaaksi työkaluksi pienin opettelun jälkeen.

TAULUKKO 2. Käytetyt annotaatiot (Vihavainen 2013, 2014; Spring Framework 4.1.1. Release API)

@Controller	Annotaation avulla Spring tunnistaa HTTP–pyyntöjä käsittelevät luokat ja lataa ne muistiin automaattisesti. Luokan täytyy pystyä huolehtimaan useista palvelupyynnöistä samanaikaisesti.
@Service	Yleiskäyttöön tarkoitettu annotaatio kuten **@Component**, kehittäjä voi valita paremmin luokan toimintaa kuvaavan. Annotaatiolla merkitty luokka ladataan muistiin automaattisesti.
@Bean	Kertoo Springille että metodi palauttaa olion, ”beanin”. Annotaatiota käytetään yleensä konfiguroinnin yhteydessä ja se luo olion muistiin automaattisesti (katso @Configuration).
@Autowired	Annotaatiolla merkitään muuttujia sekä metodeja, joihin automaattisesti muistiin ladatut luokat yritetään sijoittaa (katso @Service ja @Bean).
@Value	Annotaatiota käytetään yleensä asettamaan muuttujaan properties-tiedostosta haettu arvo.
@Async	Annotaatio metodin edessä kertoo, että metodi on asynkroninen.
@RequestMapping	Annotaatio kontrolleriluokan metodin edessä kertoo polun ja HTPP-pyyntötyypin, joita metodi kuuntelee.
@ResponseBody	Annotaatio kontrolleriluokan metodin edessä kertoo, että metodin palauttamat arvot lähetetään suoraan takaisin pyynnön tekijälle.

@PathVariable	Annotaatio kontrolleriluokan metodin parametreissä erottaa parametrin pyyntöpolusta.
@RequestBody	Annotaatio kontrolleriluokan metodin parametreissä muuttaa pyynnössä tulevan JSON-muotoisen datan annotaation jälkeen tulevan luokan olioksi.
@RequestParam	Annotaatiolla kontrolleriluokan metodin parametreissä voidaan antaa pyynnössä tulevan parametrin nimi, jonka arvo asetetaan annotaation jälkeen tulevaan muuttujaan.
@Cacheable	Annotaatio metodin edessä kertoo että metodille annettavat parametrit laitetaan välimuistiin ja jos metodia kutsutaan uudestaan samoilla parametreillä, noudetaan palautettava arvo välimuistista.
@CacheEvict	Annotaation avulla voidaan tyhjentää yksi välimuisti.
@Configuration	Konfiguroinnit on mahdollista tehdä Java-luokissa XML-tiedostojen sijaan käyttäen luokan alussa @Configuration-annotaatiota. Konfigurointiluokka sisältää usein beaneja ja @Configuration-annotaation lisäksi käytetään usein myös @Enable-alkuisia muita annotaatioita.
@EnableCaching	Annotaatiota käytetään yhdessä @Configuration-annotaation kanssa. Ottaa käyttöön välimuisti-ominaisuuden ja vastaa XML-muotoisen konfiguraation <cache:*>-elementtiä.
@EnableAsync	Annotaatiota käytetään yhdessä @Configuration-annotaation kanssa. Mahdollistaa sovelluksen asynkronisuuden ja vastaa XML-muotoisen konfiguraation <task:*>-elementtiä.
@EnableWebMvc	Annotaatiota käytetään yhdessä @Configuration-annotaation kanssa. Mahdollistaa Spring MVC-toiminnallisuuden käytön ja vastaa XML-muotoisen konfiguraation <mvc:annotation-driven />-elementtiä.
@ComponentScan	Annotaatiota käytetään yhdessä @Configuration-annotaation kanssa. Kertoo hakemiston, josta komponentit ladataan ja vastaa XML-muotoisen konfiguraation <context:component-scan>-elementtiä.

@Profile	Annotaatio metodin tai luokan edessä kertoo, että komponentti on käytettävissä kun yksi tai useampi määrätty profiili on käytössä.

3.5 Tietorakenteet ja rinnakkaisuus

Web-palvelimet palvelevat monta asiakasta samaan aikaan, ja niin myös hälytyspalvelu voidaan pyytää lähettämään hälytys vaikka edellisten lähettäminen on vielä kesken. Koska hälytyspalvelu säilöö asiakkaiden yhteystiedot (Order-luokka), voi rinnakkaisuudesta muodostua ongelma, kun tietoja yhtä aikaa päivitetään ja haetaan. Tämän vuoksi tiedot piti tallentaa tietorakenteeseen, josta tiedon hakeminen olisi nopeaa, mutta jonka muokkaaminen estäisi muokattavien tietojen hakemisen muokkauksen ajaksi. Tähän tarkoitukseen sopiva tietorakenne on Javan ConcurrentHashMap (concurrent = rinnakkainen) (Java Platform Standard Edition 7 API Specification 2014, ConcurrentHashMap).

Rinnakkaisuuden huomioivan tietorakenteen lisäksi käytettiin lukkoja varmistamassa tietojen onnistunut päivittäminen. Hälytyspalvelun käyttöön sopiva lukko oli Javan ReentrantReadWriteLock. ReentrantReadWriteLock on tarkoitettu käytettäväksi, kun tietorakannetta lukevia säikeitä on enemmän kuin päivittäviä, kuten oletettiin olevan myös hälytyspalvelulla. Lukkoja luodaan siis kaksi, Read ja Write. Kun tietorakenteeseen lisätään, poistetaan tai muokataan tietoja, lukitaan ensin Write-lukko, suoritetaan päivitys ja vapautetaan Write-lukko. Kun halutaan hakea tietoa, yritetään hakumetodin alussa lukita Read-lukko. Lukitseminen ei onnistu, jos Write-lukko on lukittu ja säie pysähtyy odottamaan vuoroaan. Tietojen rinnakkainen hakeminen on mahdollista, sillä Read-lukko ei estä muita Read-lukkoa käyttäviä metodeja hakemasta tietoa. Tietojen hakeminen on siis nopeaa kun tietorakenteen lukitsevia päivityksiä tulee vähän. Esimerkki lukkojen käytöstä on listauksessa 12. (Java Platform Standard Edition 7 API Specification 2014, ReentrantReadWriteLock.)

LISTAUS 12. ReentrantReadWriteLock-esimerkki (Java Platform Standard Edition 7 API Specification 2014, ReentrantReadWriteLock)

```
class RWDictionary {
    private Map<String, Data> m = new TreeMap<String, Data>();
    private final ReentrantReadWriteLock rwl =
        new ReentrantReadWriteLock();
    private final Lock r = rwl.readLock();
    private final Lock w = rwl.writeLock();

    public Data get(String key) {
        r.lock();
        try { return m.get(key); }
        finally { r.unlock(); }
    }
    public String[] allKeys() {
        r.lock();
        try { return m.keySet().toArray(); }
        finally { r.unlock(); }
    }
    public Data put(String key, Data value) {
        w.lock();
        try { return m.put(key, value); }
        finally { w.unlock(); }
    }
    public void clear() {
        w.lock();
        try { m.clear(); }
        finally { w.unlock(); }
    }
}
```

3.6 Push-viestien lähettäminen GCM-palvelimelle

Push-viestejä lähettävän palvelimen kehitys alkaa Googlen palvelimen valinnalla. Aiemmin tehdyissä kokeiluissa push-viestien lähettämiseen käytettiin Googlen HTTP-palvelinta sekä avoimen lähdekoodin kirjastoa gcm-server. Siksi opinnäytetyön toteutuksessa päädyttiin tekemään samoin. Apukirjasto esimerkkeineen löytyy sivulta https://code.google.com/p/gcm/. Kirjaston lisääminen Maven-riippuvuuksiin on kuvattu tarkemmin luvussa 3.1.2. Palvelimen valintaan olisi ehkä kannattanut perehtyä enemmän, sillä CCS-palvelin mahdollistaa esimerkiksi kaksisuuntaisen viestinnän palvelimen ja sovelluksen välillä.

Hälytyspalvelussa varsinaisen push-viestin lähettämisen tekevän AndroidSender-luokan (katso kuva 9 luvussa 3.2) pohjaksi otetiin gcm-demo-server -esimerkin SendAllMessagesServlet-luokka, joka käyttää push-viestin lähettämiseen gcm-server-kirjastoa. Esi-

merkki on tehty ottaen huomioon kaikki Googlen vaatimukset (lueteltu luvussa 2.5.2) ja vain pienin muutoksin push-viestit oli helppo saada kulkemaan mobiilisovellukselle.

Virhetilanteisiin reagoimisen osalta AndroidSender-luokan toteuttaminen jäi hieman kesken. Googlen HTTP–palvelin ei anna viestin perillemenosta tietoa, mutta jos push-viestin lähettämisessä tapahtuu virhe, palautetaan virheilmoitus. Jos laitteen sovelluksella on jostain syystä kaksi tokenia tai token on muuttunut, täytyy sovelluspalvelimen päivittää tietoihinsa uusi token. Jos sovellus on jostain syystä poistettu laitteesta tai token on vanhentunut, täytyy sovelluspalvelimen poistaa token ja lopettaa push-viestien lähettäminen kyseiselle laitteelle. (Android Developers, Advanced Topics.) Näiden virhetilanteiden hoitaminen ei projektin alussa ollut mahdollista eikä niitä myöhemmin ehditty toteuttaa lopputyöhön.

Mobiililaitteet ja -sovellukset kehittyvät nopeasti ja opinnäytetyön kirjallista osuutta tehtäessä kävi ilmi, että käyttäjälle saapuneen push-viestin esittämiseen on tullut uusia ominaisuuksia käyttöjärjestelmän päivityksen myötä (4.4 KitKat). Tämä ei kuitenkaan vaikuta sovelluksen oman palvelimen ja Googlen palvelimen rajapintaan ja push-viestin lähettäminen tapahtuu edelleen samalla tavalla. Jatkokehityksen kannalta olisi kuitenkin hyvä tutustua tarkemmin mobiilisovelluksen uusiin ominaisuuksiin sekä CCS-palvelimen käytön tuomiin mahdollisuuksiin.

3.7 Debuggaus ja testaus

Hälytyspalvelun tekeminen aloitettiin yhden ”Hello World” push-viestin lähetyksestä mobiilisovellukselle. Lähetyksen testauksessa käytettiin testipuhelinta, johon sovellus oli asennettu ja jonka tunnukset saatiin vanhasta taustajärjestelmästä. Viestin sisällöllä ei vielä ollut merkitystä. Hälytyksen lähettäminen todettiin toimivaksi, kun hälytyspalvelu käynnistyi ja testiviesti saapui testipuhelimeen.

Kun hälytys toimi, siirryttiin toteuttamaan oikean hälytyksen vastaanottamista palvelimelta. Uudet hälytykset tulivat ensin käsittelijälle, jonka jälkeen haettiin paljon lisätietoa hälytyksestä ja vastaanottajista ja vasta sitten valmis viesti lähetettiin mobiililaitteille. Lisätietojen ja vastaanottajien hakeminen jätettiin tässä kohdassa välistä ja push-

viestillä lähetettiin vain testidataa yhdelle testipuhelimelle keskittyen itse käsittelijän toimintaan. Käsittelijän (ExternalAlertController-luokka) toimintaa testattiin ensin cURL-komentorivityökalulla (http://curl.haxx.se/). Kun rajapinta todettiin toimivaksi ja push-viestit saapuivat puhelimeen, siirryttiin testaamaan hälytyspalvelua yhdessä muun taustajärjestelmän kanssa. Hälytyksiä vastaanotettiin kehityskäytössä olevalta taustajärjestelmältä ja testihälytykset aiheutettiin oikeilla mittalaitteilla. Testi todettiin onnistuneeksi, kun mittalaitteen aiheuttama hälytys kulki push-viestillä testipuhelimeen.

Kun hälytys saatiin kulkemaan mittalaitteelta puhelimeen, siirryttiin hakemaan oikeaa hälytyksiin ja asiakkaisiin liittyvää tietoa muusta taustajärjestelmästä. Nyt puhelimeen saapuvat push-viestit sisälsivät oikeaa tietoa esimerkiksi hälytyksen aiheuttajasta, sijainnista ja kellonajasta, joten lähes valmiin hälytyspalvelun integroiminen osaksi muuta taustajärjestelmää voitiin aloittaa.

Opinnäytetyön loppuvaiheessa aloitettiin testien tekeminen JUnit:illa (http://junit.org/) hälytyspalveluun. Käsittelijän toimintaa voidaan testata Spring-sovelluskehyksen ”MockMvc”-oliolla, jonka avulla voi tehdä pyyntöjä käsittelijän eri osoitteisiin ja tarkastella pyyntöjen onnistumista ja vastauksena saatua tietoa (Vihavainen 2014, 8.1). Palveluita voidaan testata käyttämällä Spring-sovelluskehyksen MockRestServiceServer-oliota. Testattava palvelu konfiguroidaan käyttämään HTTP–pyynnöissään MockRestServiceServer-oliota tavallisen RestTemplaten sijaan. MockRestServiceServer-oliolle asetetaan oletettu pyyntötyyppi, parametrit ja tulos, jonka jälkeen suoritetaan pyyntö testattavalla palvelulla. Oletettuja ja saatuja tuloksia vertaamalla voidaan tarkistaa testin onnistuminen. (Spring Framework 4.1.1. Release API, MockMvc & MockRestServiceServer.)

JUnit-testien kirjoittaminen jäi kuitenkin kesken ja testit eivät ole kattavia. Tarvittava pohja, konfiguraatiot ja profiilit testien ajoa varten saatiin kuitenkin tehtyä ja projektin jatkajan tarvitsee vain kirjoittaa lisää testejä.

3.8 Dokumentointi

Hälytyspalvelun dokumentoinnissa käytettiin Javadoc-työkalua. Dokumentti luodaan kirjoittamalla Javadoc-tyyliset kommentit ennen jokaista luokkaa ja luokan metodia. Javadoc-kommenteissa kerrotaan ensin lyhyesti luokan tai metodin tarkoitus. Käytettävien parametrien tarkentamisessa käytetään alussa @param-annotaatiota sekä parametrin nimeä. @return-annotaatiolla kerrotaan metodin palautusarvosta ja @throws-annotaatiolla mahdollisesti heitettävistä poikkeuksista. Kommenteista voidaan koostaa erillinen dokumentti jälkikäteen. Javadoc-dokumentointi on samalla myös koodin kommentointia ja niitä on helppo kirjoittaa ja päivittää koodatessa.

Myös kirjallinen osuus opinnäytetyöstä on osa dokumentointia, kuten myös suunnittelussa käytetyt luokka- ja sekvenssikaaviot.

3.9 Integrointi

Hälytyspalvelun integrointi osaksi muuta TreLabin taustajärjestelmää jätettiin TreLabin tehtäväksi. Ennen integrointia hälytyspalvelun konfigurointitiedostot piti kuitenkin muuttaa xml-tiedostosta Java-luokiksi ja metodeiksi, jotta konfigurointi ja eri käännösprofiilien käyttäminen olisi mahdollisimman samanlainen kuin taustajärjestelmän muissa osissa. Xml-konfiguraatiotiedostot voidaan muuttaa Java-luokiksi käyttämällä annotaatioita luokan edessä (katso luvun 3.4 taulukon 2 kuutta viimeistä riviä). Myös konfiguraatiotiedoston eri bean-komponentit luotiin @Bean-annotaation avulla.

Eri profiilien avulla voidaan sovelluksen konfiguraatiota ja asetuksia vaihtaa muuttamatta koodia. Sovellus voidaan esimerkiksi asettaa käyttämään eri profiilia ympäristön vaihtuessa. (Vihavainen 2014, 8.3.) Integrointia varten opinnäytetyöhön luotiin kaksi profiilia, ”development” ja ”default”. ”development”-profiili on tarkoitettu käytettäväksi kehityksen aikana ja ”default”-profiilia varsinaisessa ajoympäristössä.

4 YHTEENVETO

Opinnäytetyön tavoitteena oli tehdä Android-käyttöjärjestelmään push-viestejä lähettävä Java-sovellus, joka olisi helposti täydennettävissä lähettämään hälytyksiä myös esimerkiksi muille käyttöjärjestelmille. Tavoitteet saavutettiin suurelta osin ja sovelluksesta tuli vaatimukset pääasiassa täyttävä ja toimiva. Taustajärjestelmän hälytykset lähetetään asiakkaalle oikealla sisällöllä ja nopeasti. Hälytyspalveluun on kesän ja syksyn aikana lisätty yksi hälytystapa lisää. Hälytystavan lisääminen onnistui helposti ja suunnitelmien mukaan, eikä se aiheuttanut suuria muutoksia alkuperäiseen hälytyspalveluun.

Opinnäytetyön yksi selkeä puute oli push-viestin lähetyksen epäonnistumisesta seuraavien toimenpiteiden puuttuminen. Myös hälytyspalvelun jatkokehittäminen mobiilisovellusten ja käyttöjärjestelmien kehittyessä eteenpäin voi tulevaisuudessa tuoda mukanaan odottamattomia haasteita. Lähdekoodin dokumentointi ja kommentoiminen jäi kesken ja olisin halunnut tehdä niistä kattavammat seuraavaa kehittäjää varten. Myös JUnit-testejä olisi voinut kirjoittaa paljon lisää, sillä nyt ne jäivät erittäin suppeiksi.

Opinnäytetyön alussa tehty aikataulusuunnitelma ei toteutunut. Alkuperäisen suunnitelman mukaan työn piti olla valmis jo keväällä, mutta kesätöiden alkaminen muualla katkaisi opinnäytetyön viimeistelyn toukokuussa ja kirjallinen osuus aloitettiin vasta lokakuussa. Myös rajapinnan määrittely muuhun taustajärjestelmään venyi ja hidasti etenemistä. Olen kuitenkin tyytyväinen, että se tehtiin hitaasti ja harkiten, sillä juuri rajapinta muuhun taustajärjestelmään on mielestäni erittäin onnistunut ja toimiva.

Aiempaa kokemusta palvelinohjelmoinnista ja Spring-sovelluskehyksestä ei yhtä tutoriaalia enempää ollut, mutta Vihavaisen hyvät luentomateriaalit auttoivat alkuun pääsemisessä todella paljon. Opinnäytetyön tekemistä helpotti suuresti myös se, että TreLabin mittalaitejärjestelmä ja eri käyttöjärjestelmien vaatimukset olivat tulleet hyvin tutuiksi aiemmissa kouluprojekteissa yrityksen kanssa. Uutena asiana tuli palvelinohjelmoinnin lisäksi myös Maven-projektihallintatyökalu sekä testaustyökalut kuten JUnit ja cURL. Spring Tool Suiten käyttäminen oli tuttua, koska se muistutti paljon Eclipseä, jota käytettiin paljon kouluprojekteissa. Opinnäytetyö opetti myös paljon projekti- ja tiimityöskentelystä, koska se oli paljon laajempi kuin aiemmat pienemmät kouluprojektit.

Opinnäytetyö oli ensimmäinen kokonaan itse tehty suurempi sovellus, jossa sovellettiin ohjelmistotekniikan kaikkia osa-alueita aina määrittelystä ja suunnittelusta testaukseen ja integrointiin asti. Työ oli erittäin mielenkiintoinen ja sopivan haastava.

LÄHTEET

Android Developers, Develop, Google Services, Google Cloud Messaging: Implementing GCM Server. Luettu 30.10.2014.
https://developer.android.com/google/gcm/server.html

Android Developers, Develop, Google Services, Google Cloud Messaging: Overview. Luettu 11.10.2014. https://developer.android.com/google/gcm/gcm.html

Android Developers, Develop, Google Services, Google Cloud Messaging: Advanced topics. Luettu 3.10.2014. http://developer.android.com/google/gcm/adv.html

Android Developers, Develop, Google Services, Google Cloud Messaging: GCM HTTP Connection Server. Luettu 30.10.2014.
https://developer.android.com/google/gcm/http.html

Android Developers, Develop, Google Services, Google Cloud Messaging: GCM Cloud Connection Server (XMPP). Luettu 30.10.2014.
https://developer.android.com/google/gcm/ccs.html

Android Developers, Design, Patterns: Notifications. Luettu 9.10.2014.
http://developer.android.com/design/patterns/notifications.html

Apple Inc. 2014. iOS Developer Library, Local and Remote Notification Programming guide: Apple Push Notification service. Luettu 3.10.2014.
https://developer.apple.com/library/IOs/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

Apple Inc. 2014. iOS Developer Library, Local and Remote Notification Programming guide: Provider Communication with Apple Push Notification Service. Luettu 11.10.2014.
https://developer.apple.com/library/IOs/documentation/NetworkingInternet/Conceptual/RemoteNotifica-
tionsPG/Chapters/CommunicatingWIthAPS.html#//apple_ref/doc/uid/TP40008194-CH101-SW1

Ecma International. 2013. The JSON Data Interchange Format. Luettu 1.10.2014.
http://www.ecma-international.org/publications/files/ECMA-ST/ECMA-404.pdf

Iyer, Srirangan. 2011. Apache Maven 3 Cookbook. Yhdistynyt kuningaskunta: Packt Publishing.

Java Platform Standard Edition 7 API Specification. 2014. ConcurrentHashMap. Luettu 25.10.2014.
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/ConcurrentHashMap.html.

Java Platform Standard Edition 7 API Specification. 2014. ReentrantReadWriteLock. Luettu 25.10.2014.
http://docs.oracle.com/javase/7/docs/api/java/util/concurrent/locks/ReentrantReadWriteLock.html

kellabyte.com. 13.2.2011. Push: The history, experience, cost and future. Luettu 15.11.2014.
http://kellabyte.com/2011/02/13/push-the-history-experience-cost-and-future/

Kumar Pankaj. 18.11.2012. JournalDev, Java Annotations Tutorial with Custom Annotation Example and Parsing using Reflection. Luettu 19.10. 2014.
http://www.journaldev.com/721/java-annotations-tutorial-with-custom-annotation-example-and-parsing-using-reflection

Microsoft. 2014. Windows Dev Center: Raw notification overview (Windows Runtime apps). Luettu 2.10.2014. http://msdn.microsoft.com/en-us/library/windows/apps/jj676791.aspx

Microsoft. 2014. Windows Dev Center: Push notifications for Windows Phone 8. Luettu 2.10.2014. http://msdn.microsoft.com/en-us/library/windows/apps/ff402558(v=vs.105).aspx

Microsoft. 2014. Windows Dev Center: Windows Push Notification Services (WNS) overview (Windows Runtime apps). Luettu 3.10.2014. http://msdn.microsoft.com/en-us/library/windows/apps/hh913756.aspx

Opasmedia Oy. Suomen Internetopas. Luettu 28.9.2014. http://www.internetopas.com/

Pivotal Software. 2014. Spring Tool Suite. Luettu 27.9.2014. http://spring.io/tools/sts

Rene Ritchie. 2014. Mobile Nations, iMore: Interactive notifications in iOS 8: Explained. Luettu 2.10.2014. http://www.imore.com/interactive-notifications-ios-8-explained.

Sandoval, Jose. 2009. RESTful Java Web Services : Master Core REST Concepts and Create RESTful Web Services in Java. Yhdistynyt kuningaskunta: Packt Publishing.

Spring Framework Reference Documentation 4.1.0.RELEASE. 2010-2014. Luettu 27.9.2014. http://docs.spring.io/spring-framework/docs/current/spring-framework-reference/html/index.html

Spring Framework 4.1.1. Release API. Luettu 19.10.2014.
http://docs.spring.io/spring/docs/current/javadoc-api/overview-summary.html

The Apache Software Foundation. 2014. Available Plugins. 17.9.2014. Luettu 22.09.2014. http://maven.apache.org/plugins/index.html

The Apache Software Foundation. 2010. Maven Quickstart Archetype. 1.11.2010. Luettu 22.09.2014. http://maven.apache.org/archetype/maven-archetype-bundles/maven-archetype-quickstart/

The Apache Software Foundation. 2014. Introduction to the Build Lifecycle. 21.9.2014. Luettu 22.09.2014. http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html#Lifecycle_Reference

The Apache Software Foundation. 2014. Introduction to Archetypes. 17.9.2014. Luettu 22.09.2014. http://maven.apache.org/guides/introduction/introduction-to-archetypes.html

The Apache Software Foundation. 2014. What is Maven? 17.9.2014. Luettu 22.09.2014. http://maven.apache.org/what-is-maven.html

TreLab Oy. 2013. TreLab. Luettu 18.10.2014. http://www.trelab.fi/

TreLab Oy. 2013. Mittajärjestelmä. Luettu 18.10.2014. http://www.trelab.fi/mittajarjestelma/

TreLab Oy. 2013. Ihmiset. Luettu 18.10.2014. http://www.trelab.fi/ihmiset/

Turatti, M. & Pillitu, M. 2013. Instant Apache Maven Starter. Yhdistynyt kuningaskunta: Packt Publishing.

Qviki 2013. Mobiilikehitys.fi, Push-viestit ja push-palvelin – mitä ne ovat? Luettu 1.10.2014. http://mobiilikehitys.fi/push-viestit-ja-push-palvelin/

Vihavainen, Arto. 2013. Web-palvelinohjelmointi. Luettu 28.9.2014. http://www.cs.helsinki.fi/group/java/wepa/

Vihavainen, Arto. 2014. Web-palvelinohjelmointi. Luettu 28.9.2014. http://wepa-2014.herokuapp.com/material/public_html/index.html

Vohra, Deepak. 2012. Java EE Development with Eclipse. Yhdistynyt kuningaskunta: Packt Publishing.

Williams, Nicholas. 2014. Professional Java for web Applications. Yhdysvallat: John Wiley & Sons, Incorporated.

Ziff Davis. 2014. PCMag: Definition of:push technology. Luettu 1.10.2014. http://www.pcmag.com/encyclopedia/term/49977/push-technology