Simo Ala-Kotila

NATIIVIN ANDROID-SOVELLUKSEN JA REST-RAJAPINNAN TOTEUTUS

Case: Tiedotepalvelu

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma

Marraskuu 2014

MAMK
University of Applied Sciences

KUVAILULEHTI

MAMK University of Applied Sciences	**Opinnäytetyön päivämäärä** **13.11.2014**
Tekijä(t) Simo Ala-Kotila	**Koulutusohjelma ja suuntautuminen** Tietojenkäsittelyn koulutusohjelma

Nimeke

Natiivin Android-sovelluksen ja REST-rajapinnan toteutus - Case: Tiedotepalvelu

Tiivistelmä

Opinnäytetyön tavoitteena oli suunnitella ja toteuttaa natiivi Android-sovellus ja REST-rajapinta. Opinnäytetyö toteutettiin helsinkiläiselle palveluyritykselle Ch5 Finland Oy:lle. Yrityksellä on luotuna aikaisempi tiedotepalvelu, joka on sidonnainen yksittäiseen julkaisujärjestelmään. Opinnäytetyön tilaaja halusi sovelluksesta geneerisen, jonka tulevaisuudessa kuka vain yksittäinen henkilö voisi hankkia itselleen. Opinnäytetyölle asetettiin minimitavoitteet, jotka pidettiin matalana. Sen ansiosta sovellus voitiin saada opinnäytetyön aikataulun puitteissa valmiiksi.

Teoriaosuudessa kuvataan opinnäytetyössä käytetyt tekniikat. Opinnäytetyössä käydään läpi Android ja REST-rajapinta. Androidista kerrotaan mikä on sen arkkitehtuuri ja esitellään Google I/O:ssa vuonna 2010 julkaistu suunnittelumalli, jonka avulla voidaan liittää se oikeaoppisesti REST-rajapintaan. Toinen esiteltävistä tekniikoista on REST-rajapinta ja kuinka se kommunikoi Internetin välityksellä eri laitteiden kanssa. REST-rajapinta on toteutettu opinnäytetyössä Jersey-frameworkilla. REST-rajapintaan liittyy myös olennaisesti tietokanta. Tietokantaa käytetään Hibernaten avulla. Opinnäytetyön teoriaosuudessa kuvatut asiat ohjelmoidaan käytäntöön toteutus-osuudessa. Opinnäytetyön tuloksena luotiin Tiedotepalvelu-sovellus, joka toteuttaa opinnäytetyössä asetetut minitavoitteet.

Opinnäytetyössä saavutetun lopputuloksen myötä voidaan jatkokehittää sovellus tuotteelliseksi kaikkien saataville. Opinnäytetyössä toteutettua Tiedotepalvelu-sovellusta on mahdollista jatkokehittää opinnäytetyön jälkeen. Sovelluksen tuotteistaminen jää pois opinnäytetyöstä tiukan aikataulun johdosta.

Asiasanat (avainsanat)

Android, Ohjelmistokehitys

Sivumäärä	**Kieli**	**URN**
32	Suomi	

Huomautus (huomautukset liitteistä)

Ohjaavan opettajan nimi	**Opinnäytetyön toimeksiantaja**
Jukka Selin	Ch5 Finland Oy

DESCRIPTION

MAMK University of Applied Sciences	**Date of the bachelor's thesis** 13 November 2014
Author(s) Simo Ala-Kotila	**Degree programme and option** Business Information Technology

Name of the bachelor's thesis

The implementation of native Android software and the REST interface - Case: Tiedotepalvelu software

Abstract

The purpose of this bachelor's thesis was to design and create native Android software and a REST interface for Ch5 Finland Oy. The company is working at information technology services and was located in Helsinki. The company had earlier created software notification service. I developed a similar notification service application in this bachelor's thesis. The new developed software would be a generic version of the notification service. In the future anyone could have the new software. Minimum aims were set for this bachelor's thesis to make the software within schedule.

The theory section described the software technologies used in the bachelor's thesis. This section focused on Android and REST. The Android part gave information on its architecture and introduced implementation patterns. These patterns were presented in 2010 in Google I/O to give an idea of communicating with REST. The implementation had three different patterns and this bachelor's thesis focused on one of them. The REST theory section described how REST communicated over the Internet with different types of devices and how to implement Hibernate to REST. REST was implemented by the Jersey framework. The software technologies described in theory section were programmed in practice in this bachelor's thesis.

As a result of this bachelor's thesis the Tiedotepalvelu software was made. With that the minimum aims were accomplished. The results achieved in this bachelor's thesis will make it possible to develop the Tiedotepalvelu software further to a product that anyone can have. The productization of this software was not done because of the tight schedule of the bachelor's thesis.

Subject headings, (keywords)

Android, Software Development

Pages	**Language**	**URN**
32	Finnish	

Remarks, notes on appendices

Tutor	**Bachelor's thesis assigned by**
Jukka Selin	Ch5 Finland Oy

SISÄLTÖ

1 JOHDANTO

Opinnäytetyöni aiheena on suunnitella ja toteuttaa palvelu, joka välittää tiedotteita Internet-sivujen käyttäjille. Tiedotteiden välityksessä hyödynnetään yleistä eri laitteiden mahdollistavaa kommunikointia hyödyntävää REST-rajapintaa. Rajapinnan lisäksi ohjelmoidaan natiivi Android-mobiilisovellus, jolla tuotetaan tiedotteita. Opinnäytetyössäni käsitellään natiiviin mobiilisovellukseen vaadittavat työkalut ja tekniikat. Lisäksi kerron, kuinka REST-rajapinta liitetään mobiilisovellukseen.

Opinnäytetyön tilaajana on Ch5Finland Oy. Yrityksen nykyinen tiedotesovellus on sidoksissa yksittäiseen julkaisujärjestelmään. Yritys halua luoda uuden tiedotesovelluksen, jossa päästäisiin eroon yksittäisen julkaisujärjestelmän rajoitteista. Opinnäytetyössä toteuttava sovellus täten toimisi täysin omassa palvelimessa. Palvelimelle luodaan pääsy toteuttamalla REST-rajapinta.

Opinnäytetyössä tutkitaan, kuinka luodaan täysin geneerinen sovellus, joka ei ole sidoksissa yksittäisiin julkaisujärjestelmiin. Sovellukseen halutaan toiminnallisuus, joka mahdollistaa tiedotteiden luonnin paikasta riippumatta. Tutkimusongelmana opinnäytetyössä on REST-rajapinnan toteuttaminen ja kuinka se saadaan kommunikoimaan eri päätelaitteiden kanssa. Toteutettavat päätelaitteet ovat Internet-selaimessa toimiva JavaScript sekä natiivi Android-sovellus.

Sain kipinän työlleni keväällä 2014 koulun järjestämästä Mobiili- ja tietoliikenneohjelmointi -kurssista. Kurssilla esiteltiin kaksi nykyhetkisistä mobiilikäyttöjärjestelmistä, Android ja Windows Phone. IOS:ia kurssilla ei käyty läpi resurssien puutteiden vuoksi. Kurssin myötä minua alkoi kiinnostaa yhä enemmän mobiiliohjelmointi. Androidin valitsemiseen vaikutti kohtuullisen hyvä Java-ohjelmointikielen osaaminen.

2 TEKNIIKOIDEN KUVAUS

Tässä luvussa käsitellään opinnäytetyön kaksi keskeisintä tekniikkaa. Tekniikat Android ja REST-rapinta. Android-luvussa kuvaillaan sen historia ja kuinka pitäisi ohjelmoida REST-rajapinnan hyödyntäminen sovelluksessa. Opinnäytetyön REST-rajapinnan osuudessa kerrotaan mitä rajapinta on ja sen toiminta periaatteista

käytännössä. Työssä on eroteltu oma ohjelmakoodi kursiivilla valmiista frameworkien tuomista kirjastoista.

2.1 Android

Android on avoimeen lähdekoodiin perustuva käyttöjärjestelmä (Android Suomi 2014). Androidin alkuperäinen kehitysyritys oli nimeltä Android Inc. Yritys perustettiin vuonna 2003, ja sen alkuperäinen tarkoitus ei ollut tehdä käyttöjärjestelmiä puhelimille vaan digitaalikameroille. Hyvin nopeasti he muuttivat strategiaansa älypuhelimille, koska he kokivat digitaalikameroiden markkinat kapeiksi. (Deleon 2013.)

KUVA 1. Android-käyttöjärjestelmän historia (**Sourabh 2014**)

Vuonna 2005 Google astui mukaan mobiilibisnekseen ostamalla Androidin oikeudet Android Inciltä. Googlen astuminen kuvioihin oli kuitenkin huhupuheita. Google ei suoranaisesti ilmoittanut kehittävänsä Androidia. (Elgin 2005.) Pari vuotta myöhemmin 2007 Google virallisesti ilmoitti kehittävänsä Android-käyttöjärjestelmää. Samalla he ilmoittivat kehittämisen olevan täysin ilmaista eli täysin vastakohta Applen IOS:lle. (Deleon 2013.) Kuvasta 1 näkee, kuinka Androidista on ilmestynyt vuoteen 2014 mennessä jo 8 eri käyttöjärjestelmäversioita. Uusin Android 5.0 eli Lollipop ilmestyi 15. lokakuuta 2014 (Morgan 2014).

2.1.1 Android-arkkitehtuuri

Android-sovellukset kirjoitetaan Java-ohjelmointikielellä. Sovelluksien kehittämiseen käytetään Android SDK:ta tai Android Studiota. Ohjelmointia voidaan tosin tehdä Android NDK:ta käyttäen. Android NDK:ssa käytetään C / C++ ohjelmointikieltä, tosin Android NDK:ta kannattaa käyttää vain ja ainoastaan, jos sovellus todella

tarvitsee sitä. Ei koskaan vain sen takia, että sovelluskehittäjä itse halua käyttää C / C++ ohjelmointikieltä. (Android Developers 2014a.)

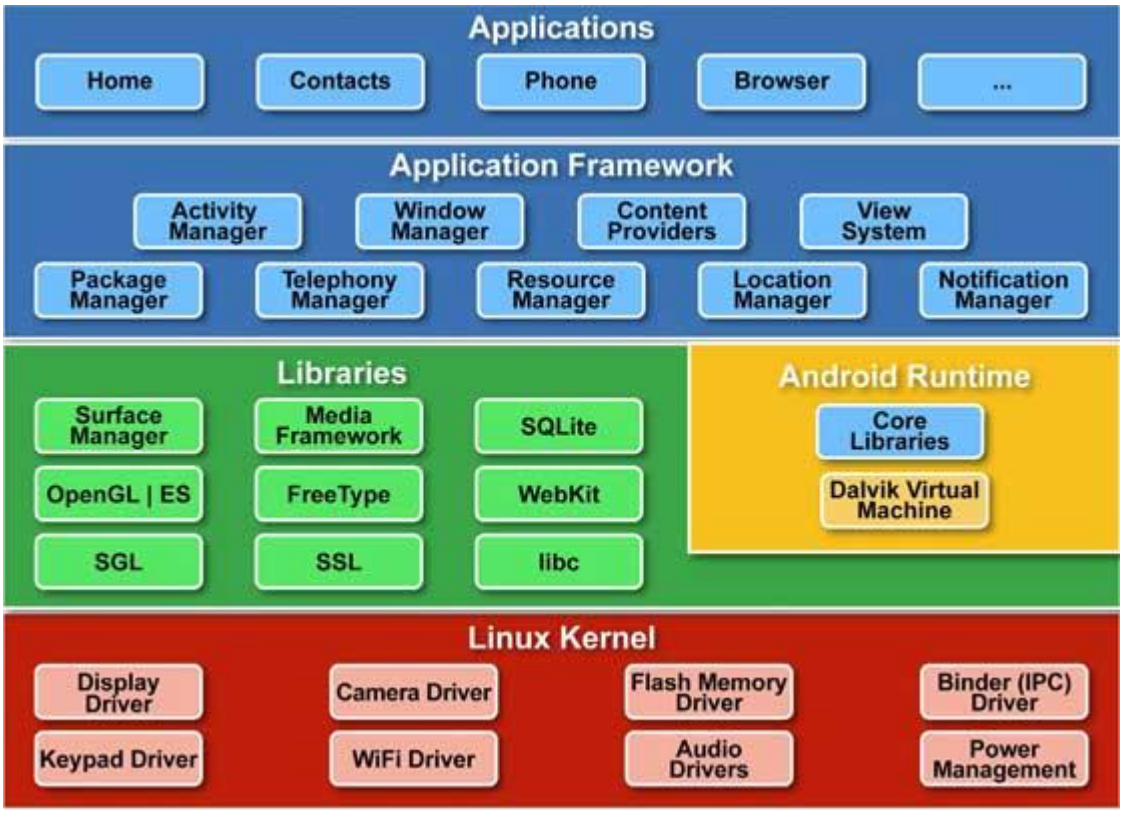
Kirjoitettu ohjelmakoodi käännetään kehittimellä .apk-tiedostoksi. Android-laite tunnistaa automaattisesti APK-tiedoston sovellukseksi. (Android Developers 2014a.) Sovelluksen ollessa valmis julkaistavaksi se voidaan laittaa yleistä jakamista varten Google Playhin. Se on Googlen tarjoama jakelukanava, jossa jaetaan esimerkiksi sovelluksia ja elokuvia. Google Playssa sovellus voidaan myös asettaa beta- tai alpha-testaukseen. Täten ohjelman löytää ainoastaan kutsutut henkilöt. (Google 2014.) Sovellus voidaan myös asentaa suoraan tiedostolta ilman, että sitä on laitettu yleiseen jakoon Google Playhin.

Android-arkkitehtuuri on monien ohjelmistokomponenttien summa. Se voidaan jakaa karkeasti neljään pääkerrokseen ja viiteen eri osastoon. (Tutorialspoint 2014.) Kerroksen alimpana kerroksena on Linux Kernel. Google perustelee syitä, miksi Linux on valittu Androidin ytimeksi, heidän mielestä se loistaa muistin ja prosessin hallinnassa. He kertovat myös Linuxin tarjoavan hyvin monimutkaisen tuen ajureilla eri laitteille, kuten kameralle tai näyttöruudulle. Se loistaa myös verkonhallinnassa ja tietoturvassa. Linux toimii myös kuvitteellisena tasona laitteiston ja ohjelmiston välillä. (Ayesha 2012.)

Androidissa Linux luo jokaisen uuden sovelluksen asennettaessa uuden käyttäjätilin. Käyttäjätili ei näy puhelimen käyttäjälle. Yksittäinen sovellus toimii omassa virtuaalikoneessa eli Dalvik Virtual Machinessa. (Android Developers 2014a.)

Dalvik Virtual Machine eli Dalvik-virtuaalikone on Googlen kehittämä virtuaaliajoympäristö suoritettavalle ohjelmakoodille. Kuvassa 2 Dalvik-virtuaalikone sijaitsee kolmannella tasolla, jossa esitellään myös sovelluskirjastot. Alkuperäinen kehittäjä Dalvik-virtuaalikoneelle oli Dan Bornstein. Hän työskenteli Googlella vuodesta 2005 vuoteen 2011 virtuaalikoneen parissa. Android 4.4 KitKat -käyttöjärjestelmässä esiteltiin Dalvik-virtuaalikoneelle vaihtoehto Android Runtine eli ART. ART korvaa Dalvik-virtuaalikoneen kokonaan Android 5.0 Lollipopissa. (Android Source 2014).

Androidin arkkitehtuurin kolmannella tasolla sijaitsevat sovelluskirjastot kuten WebKit, SQLite-tietokanta, SSL ja monia muita ohjelmointikirjastoja. WebKit on Internet-selaimien moottori. SQLite-tietokanta tallentaa sovelluksien tietoja tietokantaan. SSL-kirjasto lisää turvaa Internetin käyttämiseen. (Tutorialspoint 2014.)



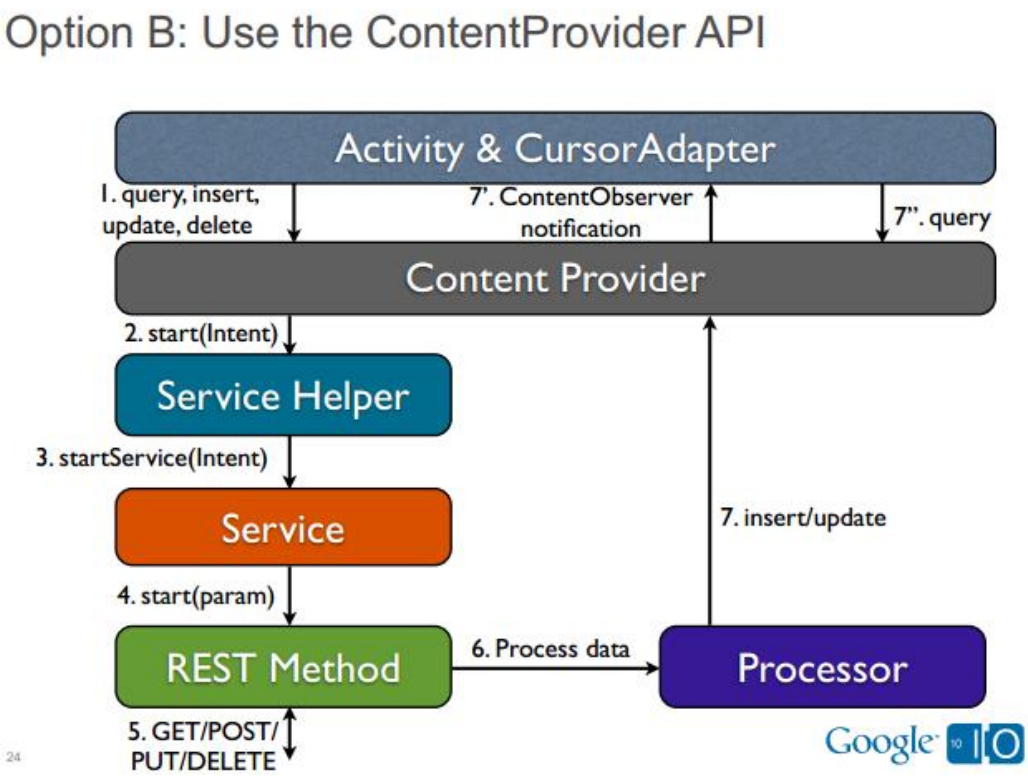
KUVA 2. Androidin arkkitehtuurin kaavio (Tutorialspoint 2014)

Arkkitehtuurin toisella tasolla kuvataan sovelluskehityksen frameworkeja. Frameworkit ovat vapaasti käytettävissä omien sovelluksien luontiin. Ensimmäisellä tasolla esitetään kaikki Androidiin asennetut sovellukset. (Tutorialspoint 2014.)

2.1.2 Android-sovellus hyödyntäen REST-rajapintaa

Opinnäytetyön yksistä tutkimusongelmista oli tutkia, kuinka luodaan natiivi Android-sovellus, joka kommunikoi REST-rajapinnan kanssa. Etsimällä tietoa tutkimusongelmaani löytyi Googlen esittelemä suunnittelumalli. Malli on esitelty vuonna 2010 Google I/O:ssa. Google I/O on Googlen kehittäjien vuotuinen konferenssi, joka yleensä pidetään San Franciscossa (Rouse 2014). Suunnittelumalli esittelee kolme eri lähestymistapaa A:n, B:n ja C:n.

A-mallissa Androidin Service-luokka kiinnitetään Activity-luokkaan, jonka avulla voidaan kommunikoida niiden välillä. Kiinnitys tapahtuu Binder-luokan avulla. B-malli on muuten samanlainen kuin A-malli, paitsi ettei Service-luokkaa kiinnitetä Activity-luokkaan. Kommunikointi hoidetaan muuta keinoa käyttäen esimerkiksi Broadcast Receiver -luokalla. C-malli on simppelein, mutta huonona puolena siinä on Service-luokan korvaaminen Sync Adapter -luokalla. Sync Adapterin käytöllä tässä tarkoituksessa ei olisi ollut hyötyä, koska sitä yleensä käytetään kun tarvitaan myös Account Manager -luokan käyttöä. Täten näistä kolmesta mallista päädyin käyttämään B-mallia. Kuvasta 3 nähdään havainnollisemmin B-malli.



KUVA 3. Google I/O 2010 REST-suunnittelumallista B-malli (Dobjanschi 2010)

B-mallia tarkemmin tutkiessa nähdään kolmantena *ServiceHelper*-apuluokka. Apuluokan kutsuminen tapahtuu vain ja ainoastaan Activity-luokasta. Luokka määrittää Service-luokalle sovelluksessa ajettavan toiminnon. Määritetty toiminta voisi olla esimerkiksi tiedotteiden hakeminen REST-rajapinnalta.

Service-luokka on valmiina sovelluskehittäjiä varten Androidissa. Se mahdollistaa suorittamaan pitkiä ja aikaa vieviä toimintoja. Voidaan täten luoda sovelluksia, jotka toimivat Androidissa taka-alalla. Voidaan suorittaa toimintoja, vaikka sovellus olisi

suljettu tai käyttöliittymää ei olisi näkyvissä. Service-luokan toiminta ei oletuksena tapahdu erillisessä säikeessä. (Meir 2012, 331.) Jotta toiminta saadaan ajettua erillisessä säikeessä, sille täytyy luoda käsin oma säe. Android tosin tarjoaa Intent Service -luokan, joka perii Service-luokan ja tarjoaa valmiiksi luodun säikeen. Jokainen Íntent Service -luokan kutsuminen menee säejonoon, ja se sammuu itsestään toiminnon saatua päätökseen, eli sitä voidaan kutsua käynnistä ja unohda -periaatteella. (Meir 2012, 348.)

B-mallissa kuvassa 3 Service-luokkaan pyydetään aloittamaan toiminto, joka valmistelee sen perusteella REST-pyynnön. REST-pyynnöt hyödyntävät HTTP-standardeja kuten GET- tai POST-metodeja. Tämän jälkeen prosessoidaan REST-rajapinnalta palautunut data *Processor*-luokassa.

Processor-luokka tutkii tarvitaanko luoda uusi tietue, päivitystä vaativia tietueita tai jopa kokonaan poistoa vaativia tietueita. Edellä mainitut CRUD-toiminnat välitetään Content Providerin avulla. Content Provider on Androidin tarjoama abstrakti kerros SQLiten ja sovelluslogiikan välillä. Content Provider voidaan toki käyttää myös tiedostojen tallentamiseen, datan jakamiseen eri sovelluksien välillä, sisäisen haun luontiin ja moneen muuhunkin.

Content Providerille määritetään polku, joka sallii pääsyn sovelluksen tietoihin. Tietojen jako voidaan toki asettaa pois päältä. Tällöin tiedot ovat ainoastaan siinä sovelluksessa, jossa se on määritetty. Tiedot haetaan hyödyntäen Content Resolveri -rajapintaa. (Meir 2012, 262.). Content Providerissa voidaan ohjelmoida, että jokaisen tietueen lisäys, päivitys tai poisto aiheuttaa huomautuksen Cursor Adapterissa.

Cursor Adapterin toimintana on vastata kuvitteellisena siltana käyttöliittymänäkymän ja adapteriin liitettyjen tietojen välillä. Adapteri kiinnitetään Androidin listanäkymään. Listanäkymä näyttää adapterissa olevat listaelementit. (Meir 2012, 156.) Lopuksi huomataan, että on saatu kuvattua kokonaisuudessaan B-malli.

Mitä hyötyä on lopulta toteuttaa B-mallia Android-sovelluksessa, jossa hyödynnetään REST-rajapintaa? Ensinnäkin olisi väärä tapa ohjelmoida REST-metodit ja prosessoida data pääsäikeessä eli Activity-luokassa, joka on jo itse asiassa varattu

käyttöliittymälle. Olisi myös huono tapa säilyttää prosessoitu data ainoastaan muistissa ja hakea se Cursor Adapateriin näytettäväksi käyttäjälle. (Dobjanschi 2010.)

Suunnittelumallin toteuttaminen mahdollistaa myös sovelluksen käytön ilman Internet-yhteyttä. Jokaisella käynnistyskerralla Android-sovellus hakee REST-rajapinnalta uusimmat tietueet, jonka jälkeen ne prosessoidaan SQLite-tietokantaan. Tietueiden tallentaminen sovelluksen omaan tietokantaan vähentää rasitusta REST-rajapinnalla.

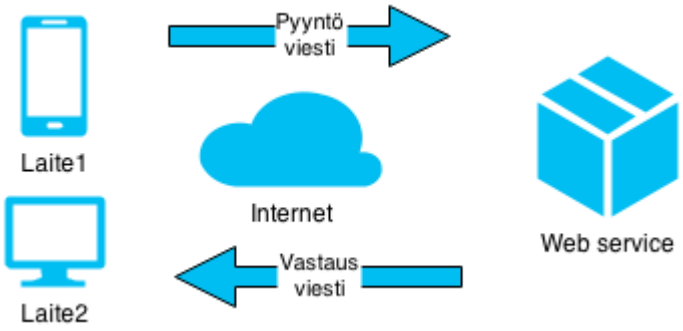
Rasitus vähenee, koska tietueet haetaan sovelluksen tietokannasta, eikä rajapinnasta uudelleen näytettäväksi, kunnes käyttäjä painaa esimerkiksi päivitä-nappia. B-malli toteutetaan käytännössä opinnäytetyön Android-osuudessa.

2.2 Web service

Web service on sovellusjärjestelmä, joka mahdollistaa eri laitteiden välisen kommunikoinnin Internet-verkon ylitse HTTP:lla tai jonkin muun Internetiä hyödyntävän protokollan avulla. Web service ei itsestään mahdollista laitteiden välistä keskustelua. Se, joka mahdollistaa tiedonsiirron, on rajapinta. (World Wide Web Consortium 2004.) Rajapintoja löytyy nykypäivänä lukemattomasti. Rajapintoja, jotka mahdollistavat laitteiden välisen keskustelun, kutsutaan esimerkiksi SOAP ja REST (Rozlog 2010).

2.2.1 REST-rajapinnan nousu

Pääajatuksena web service -sovellusjärjestelmässä on, että laitteella ei ole väliä, joka kommunikoi rajapinnan kanssa. Esimerkiksi kuvassa 4 laite yksi voisi olla Android tai iOS. Laite kaksi voisi olla Linux- tai Windows-käyttöjärjestelmällä varustettu tietokone. Eli web service -sovellusjärjestelmälle ei ole väliä, mikä laite siihen ottaa yhteyttä, ainoana vaatimuksena on Internet-yhteys. Tämäkään ei ole kuitenkaan absoluuttinen totuus, sillä Internetin puuttuminen voidaan ohittaa luomalla laitteelle keinotekoinen välimuisti. Kyseisestä välimuistista kerrotaan enemmän Androidin teoriaosuudessa.



KUVA 4. Web service -sovellusjärjestelmän karkea toimintakaavio

SOAP eli Simple Object Access Protocol on XML-merkkauskieleen perustuva rajapinta. SOAP:in toiminta perustuu siihen, että se vastaa laitteen lähettämiin pyyntöihin. Pyyntö koostuu kolmesta osasta head, body ja itse datasta. Tämän jälkeen SOAP tarkistaa löytyykö kyseistä pyyntöä. Jos pyyntö löytyy, palautetaan käyttäjälle vastaus XML-muodossa. (Barry 2014.) SOAP on määritelty World Wide Web Consortiumin standardiksi. World Wide Web Consortiumia voidaan kutsua myös W3C:ksi. (World Wide Web Consortium 2004.) SOAP:iin on myös kuulunut lisäksi WSDL ja UDDI. Edellä mainituista standardeista on kuitenkin luovuttu. Täten SOAP:ia voidaan käyttää ilman WSDL ja UDDI. (Barry 2014.)

KUVA 5. Käytetyimmät web service -rajapinnat vuonna 2014 (Hong 2014)

Web service -sovellusjärjestelmän luontia on helpotettu kehittämällä REST eli Representational State Transfer. Se on SOAP:ia helpompi käytettävyydeltään. REST:issa ohjelman komennot suoritetaan HTTP-protokollaa hyödyntäen. Komento

kirjoitetaan URL- eli Uniform Resource Identifier -muodossa. Komento lähetetään REST-rajapinnalle esimerkiksi HTTP POST -pyyntönä. POST-pyyntö käsitellään rajapinnassa ja tehdään sille tarvittavat toiminnot. Toiminnat voivat olla esimerkiksi tiedon kaivaminen tietokannasta, uuden tiedon lisääminen tietokantaan ja niin edespäin. Toiminnan perusteella palautetaan vastaus ohjelmoijan valitseman tietotyypin mukaan. Muoto voisi olla esimerkiksi JSON tai XML. W3C ei ole kuitenkaan määrittänyt REST:ia yleiseksi standardiksi. (Elkstein 2014.) Kuvasta 5 huomataan, että standarditomuudesta huolimatta REST on käytetyin rajapinta vuonna 2014. REST-rajapinta kiilasi SOAP:in ohitse vuonna 2009 Googlen etsityimpien hakusanojen joukosta (Google Trends 2014).

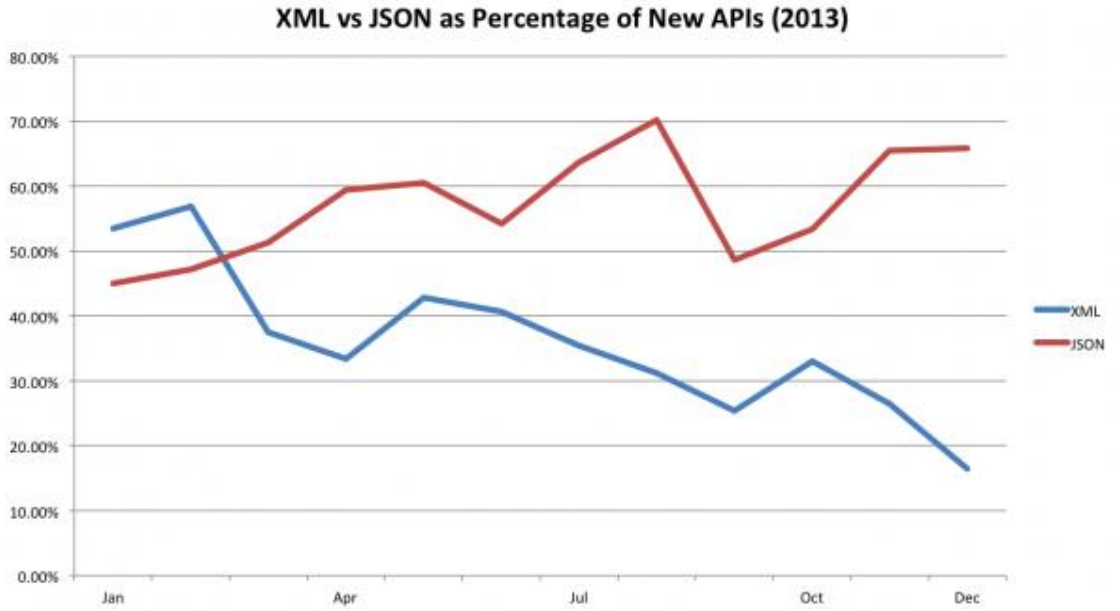
2.2.2 REST-rajapinta

Opinnäytetyön käytäntöosuudessa toteutetaan REST-rajapinta. REST-rajapinnan valintaa voidaan perustella monilla eri syillä.

REST-rajapinnassa voidaan palauttaa ihan mitä tahansa tiedostoformaattia. Tiedostoformaattien vapaus tuo mahdollisuuden käyttää simppeliä JSON-merkkauskieltä. Se tunnetaan myös nimellä JavaScript Object Notation. (Rozlog 2010.) JSON:ia selaimet käsittelevät todella paljon nopeammin XML-merkkauskieleen verrattuna. XML eli Extensible Markup Language sisältää rutkasti enemmän tiedonjärjestelyä verrattuna JSON:in yksinkertaiseen tiedonlajitteluun. JSON:issa tiedonjärjestely perustuu taulukoihin ja yhden tiedon näyttämiseen. Tätä voidaan pitää myös heikkoutena. Toisaalta XML:n tuomassa tiedonjärjestelyssä joudutaan kirjoittamaan elementtien nimiä ja niiden lopetuksia. Elementtienkuvaukset tuovat tiedostoon lisää kirjaimia. Tämä taas kasvattaa tiedostonkokoa. Tämän seurauksena JSON on tiedostokooltaan pienempi kokoinen. Pienempi kokoinen välittyy Internetissä nopeammin. JavaScriptin on myös helpompi käsitellä JSON:ia. (Zazueta 2014.) Kuvassa 6 nähdään XML:n ja JSON:in suosion ero uusissa rajapinnoissa vuonna 2013.

XML:n etuna voidaan pitää kompleksikkaiden tietorakenteiden luontia. Toinen etu on se, että tiedot tarkistetaan nimiavaruuksien avulla. (Metajack.im 2010.) Edellä mainittuja etuja voidaan kuitenkin pitää varauksella (Zazueta 2014). SOAP-rajapinnassa on pakotettu käyttämään XML:ää, jota voidaan pitää heikkoutena.

Molempien etuna voidaan pitää sitä, että tietokone ja ihminen voi lukea XML:ää sekä JSON:ia (JSON 2014).



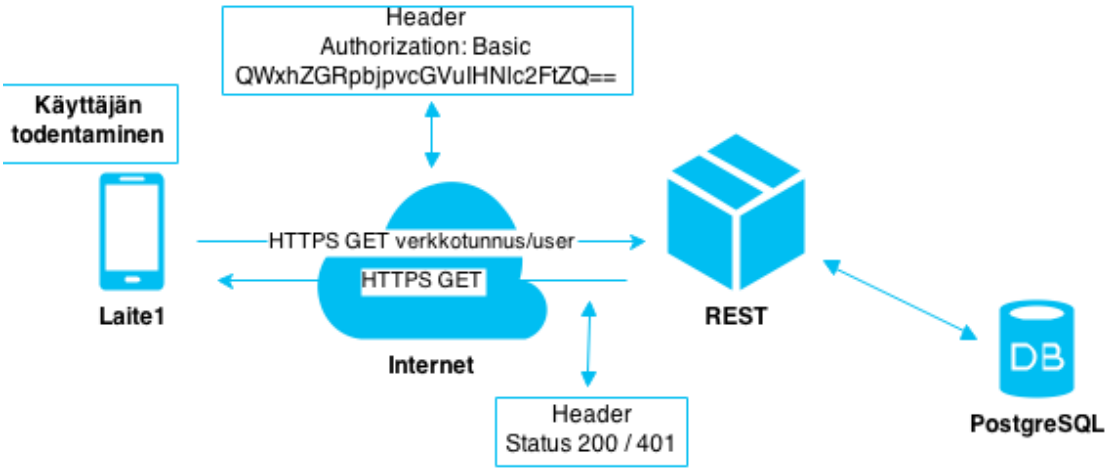
KUVA 6. XML vs. JSON (Hong 2014)

REST-rajapinnan etuna pidetään HTTP-protokollan hyödyntämistä. REST:issä komennot suoritetaan hyödyntäen URL:ää sekä HTTP-protokollaa. URL:n sekä HTTP- protokollan käyttäminen on paljon yksinkertaisempaa verrattuna vaikeasti lähestyttävään SOAP-rajapintaan.

HTTP-protokollan myötä saadaan käyttöön standardisoidut metodit kuten GET, POST, PUT ja niin edespäin. HTTP-protokolla mahdollistaa myös XMLHttpRequest-metodin käyttöönottamisen JavaScriptissa. (Rozlog 2010.) Metodille voidaan ohjelmoida GET-pyyntö, joka pyytää REST-rajapinnalta esimerkiksi tämän viikon uutiset.

Kuva 7 havainnollistaa REST-rajapinnan ja laitteen välistä kommunikointia. Esimerkissä laite yksi haluaa tarkistaa käyttäjän oikeellisuuden kirjautumisen yhteydessä. Laitteessa yksi on ohjelmoitu komento, joka olisi seuraava: Kirjoitetaan URL-muotoon *verkkotunnus/user*. Tämän jälkeen valmistellaan GET-pyyntö. Pyyntö sisältää edellisessä lauseessa määritetyn komennon. HTTP-protokolla mahdollistaa sen, että voidaan käyttää salattua eli HTTPS:ää tai ei salattua yhteyttä eli HTTP:tä. Protokollaan kuuluu myös header-osio. Header-osio mahdollistaa basic access authentication liittämisen GET-pyyntöön. Basic access authentication sisältää

käyttäjätunnuksen sekä salasanan, joka mahdollistaa käyttäjän tunnistamisen. Käyttäjätunnus ja salasana salataan käyttäen Base64. Lopuksi REST-rajapinnassa käsitellään GET-pyyntö. Rajapinta vastaa koodin 200 eli kaikki on ok. Jos käyttäjää ei löytynyt, palautetaan 401 unauthorized ja virheilmoitus käyttäjää ei löytynyt.



KUVA 7. Käyttäjän todentaminen kirjautumisen yhteydessä

REST-rajapinnassa on hyötynä, että kaikki toiminnot suoritetaan täysin tilattomassa tilassa (Rozlog 2010). Tilattomasta tilasta ei ole haittaa opinnäytetyön käytännön toteuttamisessa. REST-rajapinnalle suoritetaan HTTP-protokollan välityksellä simppeleitä CRUD-metodeja. CRUD eli create, read, update ja delete ovat pysyviä tietokantametodeja, joihin sopii varsin hyvin tilaton ajoympäristö.

3 TIEDOTEPALVELUN TOTEUTUS

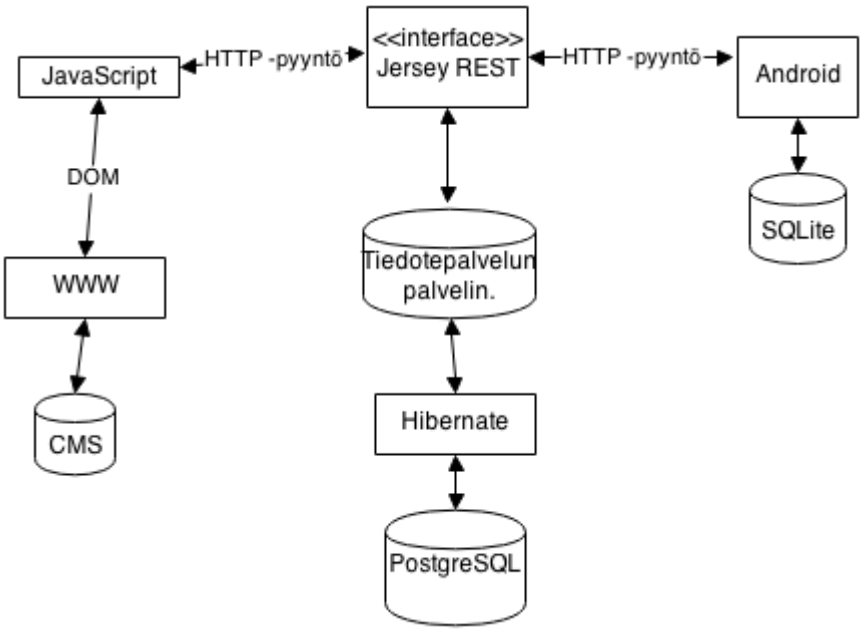
Opinnäytetyön käytännön toteutuksessa ohjelmoidaan Tiedotepalvelu. Palvelu välittää tiedotteita siihen sivulle, johon palvelu on liitetty. Esimerkiksi X-nimiselle Internet-sivulle ylläpitäjä voi lisätä JavaScriptin. Tämän jälkeen se pyytää tiedotteet REST-rajapinnasta. Pyyntö lähetään rajapinnalle jokaisella sivuston latauskerralla. Rajapinta lukee pyynnöstä URL-osoitteen ja palauttaa tiedotteet pyyntöpaikkaan takaisin. DOM-manipuloinnilla lisätään saadut tiedotteet sivustolla haluttuun HTML-elementti kohtaan. Tiedotepalvelua käyttäjä ei huomaa, mikäli tiedotettavaa ei ole. Tiedotteita voi luoda sivuston valtuutettu ylläpitäjä.

Tiedotepalvelulle ohjelmoidaan REST-rajapinta. Opinnäytetyössä toteutetaan ainoastaan puhelimessa toimiva sisällönhallintaohjelma. Ylläpitäjä syöttää tiedotteita Android-sovelluksen avulla. Kirjoitetut tiedotteet kulkevat REST-rajapinnalle.

3.1 Tavoitteet

Opinnäytetyö toteutetaan helsinkiläiselle Ch5Finland Oy:lle. Toteutus tapahtuu muuten itsenäisesti, paitsi että yritys antaa apua teknisissä ongelmissa. Työn etenemistä seurataan viikoittaisilla välikatsauksilla.

Jotta opinnäytetyö saadaan päätäntöön, sille asetetaan minimivaatimukset. Opinnäytetyön projektisuunnitelmassa määritettiin REST-rajapinnan minivaatimukset. Minimivaatimuksissa vaadittiin rajapinnan kestävän tietyn käyttäjäkuorman, pystyy käsittelemään simppeleitä CRUD-metodeja, sujuva kommunikointiväylä WWW-sivun ja mobiililaitteen välillä. Android-sovellukselta minimitoiminnallisuudet ovat tiedotteen lisäys, selaaminen ja tiedotteiden piilottaminen sekä offline-tila. Minitavoitteisiin kuuluvat myös Android-sovelluksen julkaisu Google Play -sovelluskaupassa. Upotettavalta JavaScriptilta vaaditaan sen olevan helposti lisättävissä Internet-sivustolle, tiedotteitten näyttäminen käyttäjälle, väärin käytön estäminen sekä se ei saa toimimattomana estää sivuston normaalia käyttöä. Kuvasta 8 nähdään toteuttavan opinnäytetyön kokonaiskuva.



KUVA 8. Kokonaiskuva Tiedotepalvelu-sovelluksesta

Tiedotepalvelua ei ole tarkoitus saada aivan valmiiksi välttämättä, vaan sitä olisi tarkoitus jatkokehittää myös opinnäytetyön jälkeen. Jatkokehitystoiminnallisuus voisi olla se, että hyödynnetään Google Playssa olevaa kuukausittain tapahtuvaa veloitusta. Uudet käyttäjät voisivat lisätä Tiedotepalvelun omalle Internet-sivulleen. Kirjautumisen ja veloituksen jälkeen Android-sovellus kysyisi, minne sivustolle Tiedotepalvelu halutaan lisätä. Tämän jälkeen sovellus antaisi linkin, josta JavaScript lisätään Internet-sivulle.

Voitaisiin myös kehittä toiminnallisuus jossa samaan organisaatiotaan kuuluvat saisi uusista tiedotteista ilmoituksia puhelimensa yläreunaan tai halutessaan voi lisätä widget-kuvakkeen työpöydälleen. Widget-kuvake on pienoisohjelma, joka sisältää yksinkertaisen toiminnallisuuden, joka tässä tapauksessa olisi tiedotteen näyttäminen.

Lisäksi yhtenä lisätoiminnallisuutena voisi olla, että offline-tilassa sallitaan uusien tiedotteiden luonti tai vanhojen muokkaus. Tiedotteiden talletus palvelimelle tapahtuisi automaattisesti heti Internet-yhteyden ollessa saatavilla. Voitaisiin toteuttaa myös ”vakavan” kriisin sijainnin näyttäminen mobiililaitteessa.

Toiveellinen ominaisuus voisi olla myös WWW-käyttöliittymän toteutus. Sivulla ylläpitäjälle sallitaan mobiilisovelluksen tavoin luoda uusi tiedote sekä tiedotehistorian selaaminen. WWW-käyttöliittymän sijaan Android-sovelluksen lisäksi voisi ohjelmoida iOS- ja Windows -natiivisovellukset.

Käyttäjätiedot tulisi sovelluksessa hallita jotakin keinoa käyttäen. Projektin alkuvaiheessa käyttäjätiedot syötetään tietokantaan käsin. Kyseiseen ongelmaan tulisi täten luoda WWW-käyttöliittymä, josta pystyttäisi hallita käyttäjätietoja.

Synkronisointialgoritmia voitaisiin opinnäytetyön jälkeen hioa nopeammaksi. Nopeutta saataisiin luomalla kyselyyn lisäparametri, joka lähettää synkronoinnin yhteydessä SQLite-tietokannasta kaikista tietueista viimeisimmän päivitetyn arvon. REST-rajapinta palauttaisi arvon perusteella tietueita, jotka olisivat tätä arvoa suurempia, jos suurempia tietueita löytyisi. Ne palautetaan Androidille synkronointia varten. Täten vältytään turhien tietueiden päivityksien tarkistamiselta.

Käyttöliittymää voitaisiin myös parantaa. Voitaisiin muun muassa lisätä monivalinta tiedotteiden valintaan, lisätä animaatioita näkymien vaihtamisen yhteydessä ja hioa käyttöliittymää näyttävämmäksi.

Androidin tietoturvaa pitäisi myös muokata paremmaksi. Yksi parannus voisi olla lisäämällä kirjautumisen yhteydessä ”lippu”, joka palautetaan REST-rajapinnasta Androidille. ”Lippu” vastaisi arvoa, joka vanhenee tietyn ajan kuluttua. Esimerkiksi ”lippu” vanhenisi salasanan vaihtamisen yhteydessä. Tällöin ei tarvitsisi tallentaa käyttäjän salasanaa ollenkaan Androidiin.

3.2 Kehitys- ja testausympäristö

REST-rajapinta toteutetaan hyödyntäen Jersey-frameworkia. Jersey laajentaa Javan omaa JAX-RS API:a. Apache Tomcatin versiota 6 käytetään REST-rajapinnan palvelimena. Ch5Finland tarjoaa kehityspalvelimen REST-rajapinnalle. Kehityspalvelimella on asennettuna Javan JRE versio 1.7. Syynä Java version 1.7 valintaan on Jersey frameworkin toimimattomuus vanhemmissa versioissa. Tietokantana käytetään PostgreSQL versiota 9.2. Kyseiseen tietokantaan päädyttiin, koska se on eniten käytetty Ch5Finlandilla. Tietokanta frameworkina käytetään hyvin suosittua Hibernatea. Hibernaten testaamisen käytettiin yleistä JUnit frameworkia.

Sovelluskehitykseen käytetään Android SDK versiota 22 sekä Eclipsen versiota Luna. Sovelluksien ohjelmakoodien versiohallintana käytetään Subversiota. Projektin riippuvuuksien hallintaan Mavenia. Projektin etenemistä seurataan Trellossa. Kaavioiden piirtämiseen hyödynsin pilvipalveluna toimivaa draw.iota.

Ohjelmakoodia kirjoitetaan pääasiassa Javalla. Javan JDK versiona on 1.7.51. JavaScript-ohjelmointikieltä käytetään Internet-sivustolle upotettavan koodin ohjelmointiin. Ohjelmakoodi kirjoitetaan englanniksi ja sitä on myös kommentoitu mahdollisimman paljon.

3.3 REST-toteutus

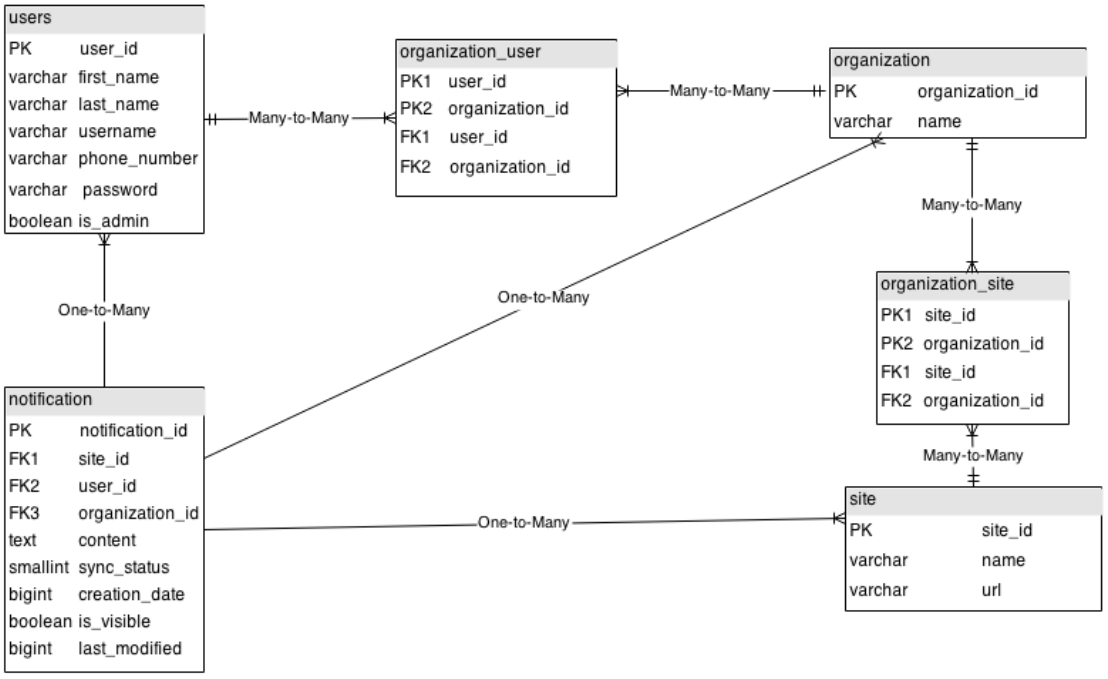
REST-rajapinta on laitteesta riippumaton sovellusjärjestelmä, jonka avulla saadaan Internet-sivu tai mobiililaite kommunikoimaan. Seuraavissa alaluvuissa kerrotaan kuinka ohjelmoidaan konkreettisesti REST-rajapinta ja natiivi Android-sovellus.

3.3.1 Tietokanta

REST-rajapinnan toteuttaminen aloitettiin suunnittelemalla tietokantakaavio. Tietokannaksi valittiin PostgreSQL 9.2 -versio. Kaavion hahmottelu oli ensimmäinen vaihe ennen ohjelmointiin ryhtymistä. Tietokantatauluja joutui muuttamaan työn edetessä. Syy muutoksiin tuli huomattuani, ettei tietokanta ollut tarpeeksi riittävä toiminnallisuuksia varten. Kuvasta 9 nähdään kokonaisuudessaan Tiedotepalvelun tietokantarakenne.

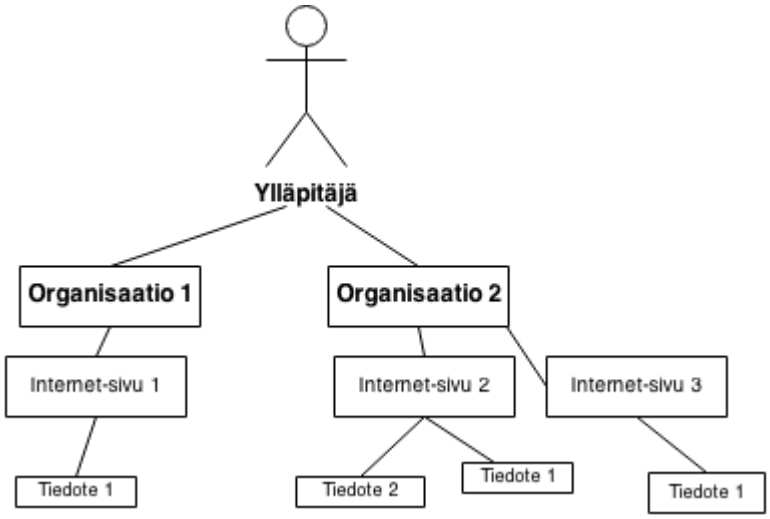
Tietokannassa säilytetään tietoja käyttäjistä, organisaatioista, sivustoista ja tiedotteista. Jokaisesta tiedotteesta on linkki tiedotteen tekijään, organisaation sekä sivustoon.
Tietokannasta pyrin tekemään skaalautuvan, jolloin ei ole väliä moneenko organisaatioon käyttäjä liittyy. Sama pätee siihen, moneenko sivustoon organisaatio voi kuulua. Edellä mainittujen suhteita kuvataan tietokantatauluissa *organization_user* ja *organization_site*. Kuva 10 havainnollistaa tietokannasta puurakenteen.

Tietokantaan voidaan myös lisätä miltei loputtomasti uusia sivustoja, käyttäjiä ja organisaatioita. Tämän ansioista upotettava JavaScript voidaan lisätä niin moneen sivustoon kuin on mahdollista.



KUVA 9. REST-rajapinnan tietokantakaavio

Users-tietokantataulun tietue sisältää perustiedot kustakin käyttäjästä. *Organization*-taulussa säilytetään kaikki palveluun liitetyt asiakkaat, joita kuitenkin selvyyden vuoksi on nimetty tietokantaan *organization*. *Site*-tietokantataulussa säilytetään tiedot sivustoista, joissa halutaan näyttää tiedotteita. *Notification*-tietueessa on nimensä mukaan kaikki ylläpitäjän lisäämät tiedotteet.



KUVA 10. Puurakenne REST-rajapinnan tietokannasta

Tietokantaan on myös ohjelmoitu trigger-funktio joka aktivoituu automaattisesti, jokaisella rivin päivityksen yhteydessä. Viimeksi päivitettyä arvoa säilytetään

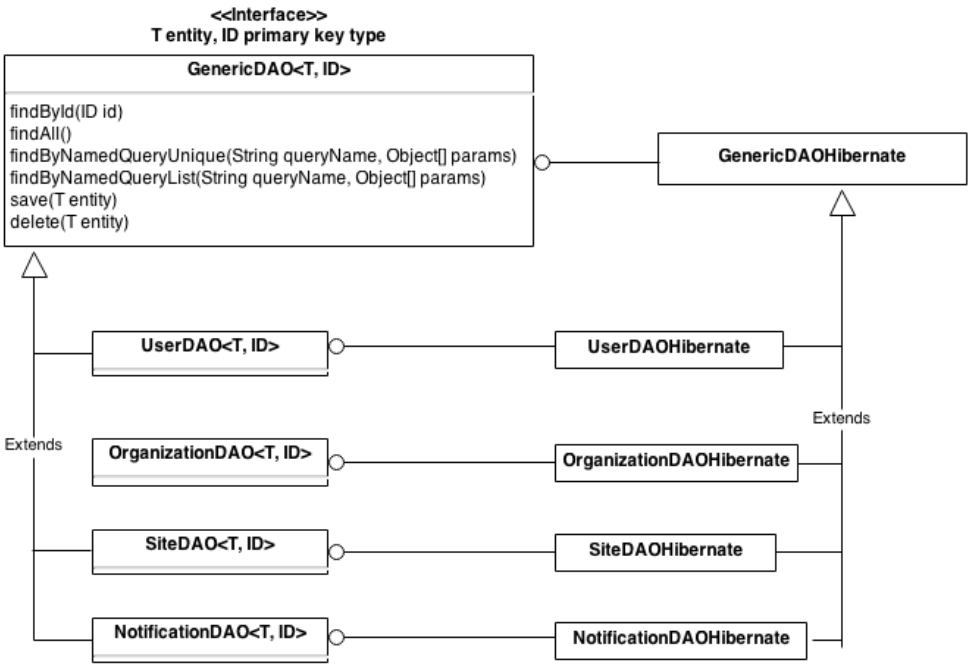
last_modified-sarakkeessa. Sarakkeen avulla pystytään tarkistamaan Android-sovelluksella tarvitaanko tehdä päivitystä puhelimen omaan SQLite-tietokannassa.

3.3.2 Hibernate

REST-rajapinnan toteuttaminen aloitettiin tietokantataulujen luomisen jälkeen. Tietokannan ja REST-rajapinnan välillä toimii Hibernate-framework. Hibernaten ottaminen mukaan opinnäytetyöhön nopeuttaa roimasti ohjelmointia, sillä se tuo mukaan valmiiksi ohjelmoidun olio-pohjaisen lähestymisen tietokantaan. Olioiden ohjelmoiminen alusta itse suoraan JDBC:llä olisi vienyt aivan liian paljon aikaa opinnäytetyön käytännön toteuttamisesta.

Hibernatelle luotiin neljä POJO- eli Plain Old Java Object -luokkaa, jotka vastaavat tietokannan *users*-, *organization*-, *site*- ja *notification*-tauluja. Kuvasta 9 nähdään kuinka *users*-, *organization*- ja *site*-tietokantatauluista on monen suhde moneen viittauksia. Hibernaten avulla voitiin kertoa suhteet *organization_user*- ja *organization_site*-tietokantatauluihin. Tämän jälkeen se luo automaattisesti niiden suhteet. Eli ei tarvitse luoda lisä POJO-luokkia kahteen edellä mainittuun tietokantatauluun. Sama pätee myös *notification*-tietokantataulusta yhden suhde moneen. Hibernatelle riittää se, että kertomalla *Notification*-luokassa *User*-, *Organization*- ja *Site*-luokkien olemassaolon. Tämän jälkeen se pitää huolen niiden suhteista toisiinsa. POJO-luokkien toteutuksen jälkeen ohjelmoitiin DAO-luokat eli Data Access Objectit.

DAO-luokkien toteuttamisessa käytin apuna Hibernatea käsittelevää ohjelmointikirjaa. Bauerin ja Kingin (2007) Manning Java Persistence With Hibernate -kirjassa opastetaan käyttämään geneeristä data access -kerrosta. Kuvassa 11 nähdään toteutetun geneerisen DAO-kerroksen malli. Kirjassa käytetyn kaavan on alun perin toteuttanut Sun Microsystems eli nykyinen Oracle, joka on kehittänyt Java-ohjelmointikielen. DAO-luokkien avulla saadaan erotettua REST-rajapinnan sovelluslogiikka irti tietokannan ohjelmointikoodista, jota voidaan pitää suurena hyötynä.



KUVA 11. Geneerinen toteutus Hibernaten data access objectista

Geneerisessä DAO-lähestymistavassa luodaan geneerinen interface eli rajapinta, jossa määritetään kaikki tavalliset tietokantojen CRUD-metodit. Tämän jälkeen luodaan Geneerinen luokka, joka toteuttaa edellä mainitun rajapinnan. Geneerisessä luokassa kirjoitetaan rajapinnan tuomiin metodeihin sovelluslogiikat.

Tämän jälkeen luodaan jokaista POJO-luokkaa varten itsenäinen DAO-rajapinta. DAO-rajapinnassa laajennetaan geneerisellä DAO-rajapinnalla. Yksittäisen POJO:n rajapintaa voidaan ohjelmoida tietokantataulukohtaisia metodeja. Tietokantataulukohtaiset metodit saavat sovelluslogiikkansa omassaan DAO-luokassa. Omassa DAO-luokassa POJO saa geneerisessä luokassa määritetyt CRUD-metodit. Esimerkiksi *User*-luokasta tehdään *UserDAO*-rajapinta, joka laajentaa geneerisen rajapinnan. Tämän jälkeen luodaan *UserDAOHibernate*-luokka, joka laajentaa geneerisen DAO-luokan. *UserDAOHibernate*-luokka lopuksi luokka toteuttaa *UserDAO*-rajapinnan. Rajapinnan toteuttamat metodit voitaisiin ylikuormittaa *UserDAOHibernate*-luokassa.

Koko tiedonkäsittelykerros voidaan paketoida tehdasluokkaan. Tehdasluokka toteuttaa oman rajapinnan, jossa on määritetty geneerisesti luodut DAO-rajapinnat. Tehdasluokka alustaa pyydetyn DAO:n metodin kutsun yhteydessä. Tehdasluokan luonti ei olisi ollut välttämätön, mutta toteutin sen siltin REST-rajapintaan varten.

Ennen kuin ryhdyin toteuttamaan REST-rajapintaa, luotiin JUnit-testausluokka, jotta voitiin todentaa toimivatko Hibernatelle luodut DAO:t käytännössä. Toimimisen todettua todeksi, siirryin toteuttamaan REST-rajapintaa.

3.3.3 Jersey REST-rajapinta

REST-rajapinnan toteutus tapahtui Jersey-frameworkia hyödyntämällä. Opinnäytetyössä rajapinta koostuu neljästä eri luokasta, *AuthResource*, *BannerResource*, *UserResource* ja *NotificationResource*. *AuthResource*-luokka on rajapinnan ”ydin”. Kyseisessä luokassa todennetaan käyttäjä ja Tiedotepalvelun oikeutettu käyttäminen.

AuthResource-luokassa käyttäjän todentamisessa hyödynnetään HTTP-protokollan tarjoamaa basic access authenticationia. Käyttäjätunnus salataan hyödyntäen base64:ää. Salattu käyttäjätunnus muutetaan palvelimen päässä takaisin luettavaan muotoon. Salausta lisätään käyttämällä suojattua HTTPS-yhteyttä. Mikäli käyttäjätunnus on oikea, palautetaan rajapinnalta HTTP:n mukainen 200-arvo. Mikäli käyttäjätunnus on väärä, palautuu 401-arvo. *AuthResource* sisältää myös todentamisen Tiedotepalvelun Internet-sivulle upotettavaa JavaScriptiä varten.

JavaScript todennetaan sinne saapuvasta GET-pyynnöstä. Pyynnöstä tarkistetaan URL-osoite, joka saadaan JavaScriptin lähettämästä referer-osasta. Jos URL-osoitetta ei löydy tietokannasta, REST-rajapinta lähettää pyyntöpaikkaan virheilmoituksen. *AuthResource*-luokasta päästään Jerseyssä mukana tulevan ominaisuuden avulla sub-resourceen eli alaresurssiin.

BannerResource, *UserResource* ja *NotificationResource* ovat kaikki alaresursseja. *BannerResource* kapseloi yhdellä Internet-sivulla näytettävät tiedotteet. Luokkaan on ohjelmoitu kaikki tietokantahaut tiedotteiden näyttämistä varten. *UserResource*-luokasta voidaan hakea käyttäjän tiedot ilman tiedotteita. *NotificationResource*-luokka sisältää kaikki käyttäjien luomat tiedotteet. Alaluokkiin on ohjelmoitu tarkisteita, mikäli valtuuttamaton ylläpitäjä yrittää tehdä toimintoja. Rajapinta estää toiminnan ja palauttaa virheilmoituksen.

REST-rajapinta palauttaa JSON-muodossa kaikki onnistuneet kyselyt. JSON luodaan hyödyntäen Jerseyn tarjoamaa JAXB-papua. Kaikille palautettaville tiedoille luodaan POJO-luokka. POJO-luokkaan haetaan tietokannasta rajapinnalta kysytyt vastaukset. Virhetilanteessa rajapinta palauttaa tekstin JSON:in sijasta.

3.3.4 WWW-sivulle upotettava JavaScript

Tiedotepalvelun yksi toiminnallisuus on näyttää sivuston käyttäjille tiedotteita. Tiedotteen näyttäminen onnistuu toteuttamalla JavaScriptin, joka tekee HTTP GET -pyynnön REST-rajapinnalle. REST-rajapinnasta palautuvaa JSON:ia on helppo käsitellä JavaScriptilla. JSON:ista poimitaan tiedotteet, jotka näytetään DOM-manipuloinnilla HTML-elementissä.

Ohjelmoituun JavaScriptiin lisättiin ominaisuus sitä varten, jos Tiedotepalvelun REST-rajapinta olisi kaatunut. Ilman tätä ominaisuutta JavaScript todennäköisesti estäisi normaalin sivuston käyttämisen. Ongelma saatiin ratkaistua käyttämällä niin sanottua lazy-loading -metodia. Ominaisuuden jälkeen JavaScriptin lataus tapahtuu epäsynkronisesti. Tämä lataus aikakatkaistaan 5 sekunnin kuluttua, mikäli lataus epäonnistuu.

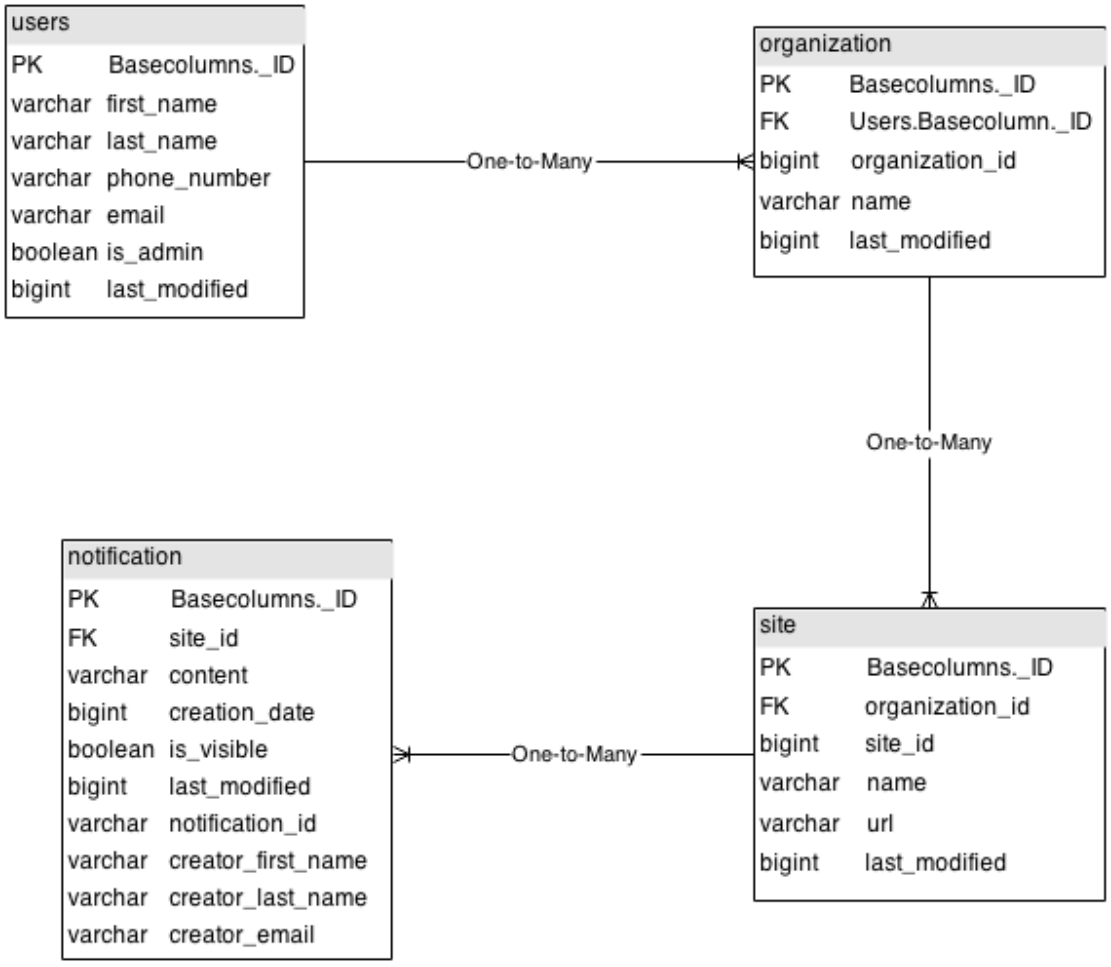
3.4 Androidin toteutus

Android API:ksi valitaan Jelly Bean eli versio 16. Valitsemalla sitä suuremman API-version saadaan Androidista käyttöön uusia ominaisuuksia sovelluksessa. Kyseinen API-versio kattaa Android-laitteista noin 75 prosenttia (Android Developers 2014b). Jättämällä noin 35 prosenttia Android-laitteista sovelluksen ulkopuolelle, on mielestäni häviävän pieni osa.

Androidin toteutus alkoi sisäistämällä teoriaosuudessa kuvatun suunnitelman. Suunnitelman avulla sain luotua itselleni kuvan toteutettavasta sovelluksen eri vaiheista. Ensimmäisenä vaiheena oli luoda projekti Eclipsen Android SDK:lla. Projektin varmuuskopiointi tapahtui pitämällä sitä SVN:ssä. Projektin luonnin jälkeen aloin toteuttamaan Android-sovelluksen SQLite-tietokantaa.

3.4.1 SQLite

SQLite-tietokanta on periaatteessa kopio REST-rajapinnan tietokannasta. Kuvassa 12 nähdään kaavio tietokannasta. Tietokanta sisältää *users*-, *organisation*-, *site*- ja *notification*-tietokantataulut. *Users*-tietokantatauluun tallennetaan sovelluksessa nykyisellä ajanhetkellä oleva kirjautunut käyttäjä. Sovellus mahdollistaa ainoastaan yhden käyttäjän olevan kirjautuneena. *Organisation*-tietokantataulussa on kirjautuneen käyttäjän organisaatiot. Se sisältää viittauksen kirjautuneeseen käyttäjään, jonka Android luo automaattisesti. *Site* sisältää käyttäjän organisaation liitetyt Internet-sivut. *Site*-tietokantataulusta saadaan viittaus organisaatioihin, joihin tämä kyseinen Internet-sivu kuuluu. Neljäs tietokantataulu eli *Notification*-tietokantatauluun tallennetaan kaikki tiedotteet. Tiedotteet saavat viittauksen *Site*-tietokantatauluun eli niihin mihin ne kuuluvat.



KUVA 12. SQLite-tietokantakaavio

Tietokanta luodaan Androidissa hyödyntäen SQLite Open Helper -apuluokkaa. Luokka peritään omaan Java-luokkaan. Toteutuksessa nimesin luokan *DatabaseHelper*-luokaksi. Apuluokkaa kutsutaan Content Providerissa.

Content Provider luo SQLite-tietokannan, kun sovellus käynnistetään. Kuitenkaan jokaisella kerralla ei alusteta tietokantaa tyhjäksi. Tietokanta tyhjennetään ainoastaan silloin, kun tietokanta päivitetään uudempaan versioon. Sovelluksen tietokantaversio on asetettu staattiseen muuttujaan. *DatabaseHelper*-luokka ottaa huomioon päivitystä vaativan tietokannan, jos huomataan sen olevan uudempi kuin aikaisempi.

3.4.2 Content Provider

Android-sovellukseen luodaan *TiedotepalveluContentProvider*-luokka, joka perii Content Provider -luokan. *TiedotepalveluContentProvider*-luokkaan määritetään polku, joka mahdollistaa pääsyn tietokantaan Content Provider -luokan avulla. Polku kirjoitetaan sovelluksen AndroidManifest.xml -tiedostoon. Polun lisäksi määritetään se sijainti, jossa luokka sijaitsee projektissa. Tehdään myös määritys, jolla estetään muiden Android-sovelluksien pääsyn Content Provideriin.

AndroidManifestin määrityksien lisäksi joudutaan määrittämään neljään eri tietokantatauluun omat tunnistepolut. Tunnistepolut kirjoitetaan *TiedotepalveluContentProvider*-luokkaan.

Yksittäinen tunnistepolku sisältää AndroidManifestissa määritetyn polun lisäksi tietoa kuvaavan nimen. Opinnäytetyön sovelluksen tunnistepolku Content Provideriin on *com.ch5finland.tiedotepalvelu*. Yksittäisen tiedon tunnistepolku saadaan lisäämällä tähän *content://com.ch5finland.tiedotepalvelu/user*. *User*-tunniste kuvastaisi tässä tapauksessa tietokantataulun nimeä eli *Users*:ia. Yksittäistä polkua tarvitaan, koska sen avulla saadaan luotua SQL-kyselyitä sovelluksessa. Yksittäisen polun lisäksi Content Providerille määritetään sisältöä vastaava tunniste.

Sisältöä vastaavan tunnisteen luonti tapahtuu luomalla yksittäiselle polulle kaikkiin tai yksittäiseen elementtiin tapahtuva pyyntö. Esimerkiksi *USER* vastaisi arvoa 10, *USER_ID* arvoa 11. Arvoilla määritetään polkuun yksittäistä vastaava elementti. Lopuksi arvo asetetaan staattisella Javan koodilohkolla seuraavasti hyödyntäen

UriMatcher-objektia eli uriMatcher.addURI*(”com.ch5finland.tiedotepalvelu”, ”user”, 10)*. Yksittäisen elementin määritys olisi muuten samanlainen, paitsi että sille lisättäisiin villikorttimerkki eli *user/#*. Lopuksi sisältötunnisteen avulla pystytään luomaan koodilohkoja Content Provider -luokasta tulleisiin CRUD-metodeihin.

CRUD-metodien sisälle luodaan switch-case -rakenne, joka etsii sisältöpyyntöä vastaavan lohkon. Tämän sisässä voidaan määrittää mistä tietokantataulusta halutaan hakea tietoa. Jokaisen CRUD-metodin sisälle on myös ohjelmoitu huomauttaja, joka aktivoituu tietokantaan tapahtuvista muutoksista. Täten saadaan pidettyä Cursor Adapter ajan tasalla tietokannassa tapahtuvista muutoksista. Yksittäisen polun määritys ei riitä toteuttamaan Content Provideria. Sille määritetään myös tyyppi jokaisesta palautettavasta elementistä eli MIME-tyyppi.

Palautettava tyyppi määritetään niin, että yksittäistä MIME:ä vastaa *vnd.android.cursor.item/vnd.com.ch5finland.user*, ja kaikkia vastaava MIME olisi *vnd.android.cursor.dir/vnd.com.ch5finland.user*. Edellä mainitut MIME:t kirjoitetaan Content Providerin tuomaan getType-metodiin. Switch-case -rakenteella, etsitään vastaavat sisältötunnisteet. Tunnisteen löydettyä se palauttaa tämän MIME:n eli tyypin.

Content Provider käyttöönottamisella voidaan hyödyntää Loader-luokkaa, joka huolehtii automaattisesti käyttöliittymän ja tietokannan kommunikoinnista. Tietokanta toimii Loader-luokan avulla omassa säikeessä, kuitenkin huolehtien käyttöliittymän olevan synkronissa (Meir 2012, 277). Loader-luokkaa hyödynnetään käyttöliittymä osiossa. Opinnäytetyön Android-sovellus olisi voitu toteuttaa ilmankin Content Provideria. Siitä on kuitenkin paljon enemmän hyötyä kuin haittaa.

3.4.3 Service

Android-sovelluksen ohjelmointi jatkui toteuttamalla kuvassa 3 B-mallissa näkyvät kolme eri vaihetta *ServiceHelper*, Service ja *Processor*. Kolmesta luokasta toteutettiin ensimmäisenä *ServiceHelper*-luokka.

ServiceHelper-luokkaan on ohjelmoitu ainoastaan yksi metodi, jota kutsutaan Activity-luokasta. Metodi käynnistää halutun *Processor*-luokan. *Processor*-luokka

suoritetaan Service-luokassa erillisessä säikeessä. B-mallista muutin Service-luokan Intent Service -luokaksi, koska se tarjoaa valmiin toteutuksen säehallinnasta. Tällöin ei tarvitse huolehtia säikeen lopettamisesta ja se on myös säeturvallinen. Säeturvallisuuden mahdollistaa sen toteutus, koska kaikki Intent Service -pyynnöt menevät säejonoon. Täten Intent Service -luokka ei voi olla ajossa päällekkäin samaan aikaan.

ServiceHelper-luokasta pyyntö saatiin vietyä Intent Service -luokalle hyödyntäen Androidissa olevaa putExtra-metodia. Metodin sisälle voidaan asettaa ylimääräisiä arvoja, jotka välitetään Intent-luokan avulla. Opinnäytetyössä sovellukseen ohjelmoitiin neljä erilaista komentoa, jotka olivat *SYNCHRONIZE_LOCAL_DATABASE*, *POST_NEW_NOTIFICATION*, *CHANGE_NOTIFICATION_VISIBILITY* ja *VALIDATE_USER*.

SYNCHRONIZE_LOCAL_DATABASE-pyynnöllä haetaan kaikki data REST-rajapinnalta. Rajapinnalta saadut tiedot tallennetaan Androidin SQLite-tietokantaan. Haettavat tiedot ovat käyttäjätiedot, organisaatiot, sivustot ja tiedotteet. *POST_NEW_NOTIFICATION*-pyyntö pyytää sovellusta valmistelemaan uuden tiedotteen lähettämistä REST-rajapinnalle. *CHANGE_NOTIFICATION_VISIBILITY*-pyynnöllä pystytään laittamaan piiloon valittu tiedote. *VALIDATE_USER*-pyyntö tarkistaa käyttäjän sähköpostiosoitteen ja salasanan REST-rajapinnalta.

Intent Service -luokassa tutkitaan *ServiceHelper*-luokasta saatu suoritettava komentopyyntö. Komentopyynnön löydettyä luodaan *Processor*-luokka. *Processor* on abstrakti luokka, joka tarjoaa periytettävälle luokalle yksinkertaiset metodit prosessointia varten. Luokka näkyy kuvassa 13. Prosessointilogiikka kirjoitetaan abstraktimetodin sisälle. Esimerkiksi voidaan luoda *AuthProcessor*-luokka, joka perii *Processor*-abstraktiluokan. Luokan suoritettava ohjelmakoodi kirjoitetaan *execute*-metodin sisälle, joka tässä tapauksessa on käyttäjätunnuksien todentamista.

AuthProcessor-luokasta pystytään täten luomaan käyttäjän todentamista tutkiva luokka, joka ei vaikuta muihin *Processor*-luokkiin, mutta omaa kuitenkin globaalit metodit yksittäisen luokan toteuttamiseen. Sovellukseen toteutin *AuthProcessor*-luokan lisäksi seitsemän eri *Processor*-luokkaa. Näistä seitsemästä luokasta yksi paketoi synkronisointia suorittavia luokkia SQLite-tietokannassa. Paketoivan luokan

nimeksi annoin *SynchronizeDatabaseProcessor*. Kyseisen luokan sisällä ovat *User*-, *Site*-, *Organization*- ja *NotificationProcessor*-luokat, jotka suorittavat synkronisointia.

SynchronizeDatabaseProcessor-luokka lukee REST-rajapinnalta palautetun JSON-merkkauskielellä kirjoitetun paluuviestin. Paluuviestistä luetaan ensimmäisenä yksittäisen tietueen tunniste, jolla voidaan tarkistaa tietokannasta kyseinen tietue. Tietue haetaan tietokannasta käyttäen Content Resolver -rajapintaa. Rajapinta haetaan metodilla getContentResolver. Metodi hakee AndroidManifestissa määritetyn tunnistepolun. Se sallii pääsyn Content Provideriin.

```java
/**
 * Abstract class which provide main processing methods.
 *
 * @author Simo Ala-Kotila
 * @see Connection
 */
public abstract class Processor extends Connection{

    /**
     * Keep JSON in memory.
     */
    private User mUser;

     * Provide ContentResolver for CRUD-methods.
    private ContentResolver mResolver;

     * Empty constructor.
    public Processor(){}

     * Constructor with only account details.
    public Processor(String username, String password) {

     * Allow to pass Content Resolver object to sub classes.
    public Processor(String username, String password, ContentResolver resolver) {

     * Handle entity deleting from SQLite-database.
    public void delete(){}

     * Delete all entity's which all related to given id.
    public void deleteAllBelong(Object id){}

     * Insert entity to SQLite-database.
    public void insert(){}

     * Update given entity.
    public void update(Cursor cursor){}

     * Update given entity.
    public void update(Cursor cursor, Object[] entity){}

     * @return DTO of User class.
    public User getUser(){

     * Set User to DTO.
    public void setUser(User user){

     * Provide ContentResolver for CRUD-methods.
    public ContentResolver getContentResolver() {

     * @param Set ContentResolver for class.
    public void setContentResolver(ContentResolver resolver) {

     * This method will query latest data from REST. Method will also
    public void transferJSONToUserDTO(){

     * This method execute written logic for processor.
    public abstract void execute();
```

KUVA 13. Kuvankaappaus Processor-luokasta

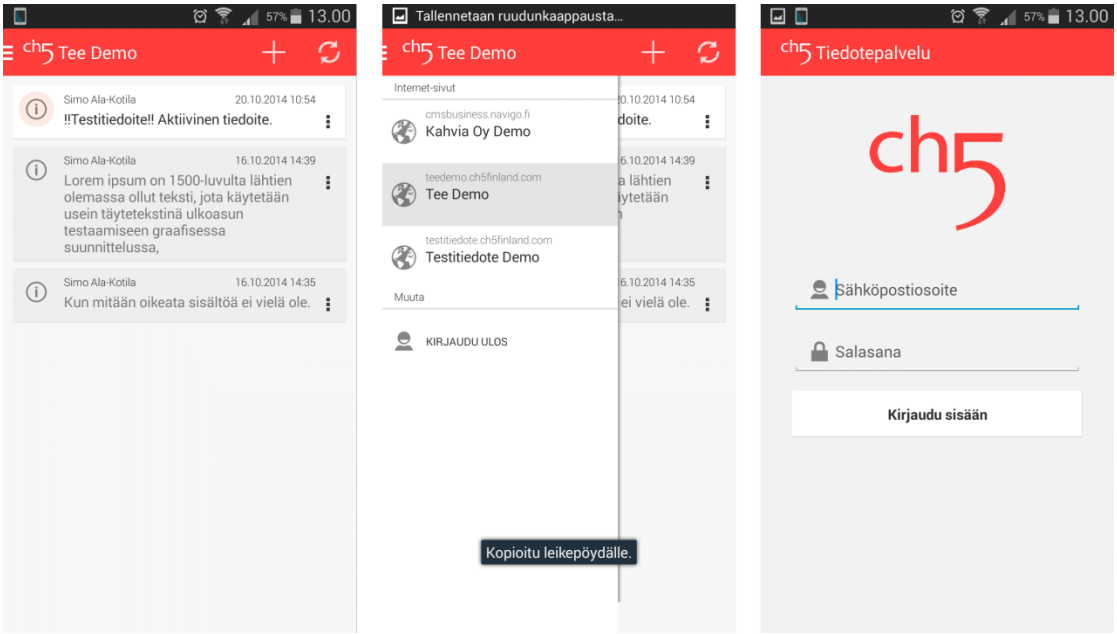
Yksittäisessä *Processor*-luokassa esimerkiksi *OrganizationProcessor*-luokassa ensimmäisenä tarkistetaan, onko tietue poistettu REST-rajapinnalta. Tarkistaminen toteutetaan hakemalla tietueesta tunniste, jota verrataan SQLite-tietokantaan. Mikäli tunnistetta ei löydy, tämän oletetaan poistuneen REST-rajapinnalta. Täten tietue

voidaan myös poistaa SQLite-tietokannasta. Tietueen poistamisen tarkistuksen jälkeen käsitellään uusien tietueiden lisääminen. Otetaan tunniste, jolla tarkistetaan onko kyseistä tietuetta tietokannassa. Mikäli ei ole, luodaan uusi tietue. Jos tietue löytyy tietokannasta, käydään tarkistamaan onko kyseistä tietuetta muokattu.

Päivitettävän tietueen tarkistaminen aloitetaan vertailemalla REST-rajapinnan ja SQL-tietokannan viimeisintä päivitettyä saraketta. Jos REST-rajapinnan tarjoama viimeiseksi päivitetty arvo on suurempi kuin SQLite:ssä oleva, voidaan olettaa tämän tietueen olevan päivitetty, jonka jälkeen päivitetty arvo päivitetään myös SQLite-tietokantaan. Lokaalin SQLite-tietokannan hyödyntäminen täten mahdollistaa sovelluksen käyttämisen Internet-yhteyden ei ollessa saatavilla. Sovelluksen offline-tilassa käyttäjä ei koskaan näe vain tyhjänä olevaa käyttöliittymänäkymää. Loput toteutetut *Processor*-luokat ovat *EditNotificationProcessor* ja *NewNotificationProcessor*. *EditNotificationProcessor*-luokka piilottaa näkyvistä valitun tiedotteen. *NewNotificationProcessor*-luokalla lähetetään uusi tiedote näytettäväksi.

3.4.4 Käyttöliittymä

Processor-luokkien toteuttamisen myötä voitiin alkaa toteuttamaan sovelluksen käyttöliittymää. Käyttöliittymään kuuluvat teoriaosuudessa vielä käsittelemättä olevat Activity- ja Cursor Adapter -luokat.



KUVA 14. Toteutetun Android-sovelluksen käyttöliittymä

Activity-luokka on sovelluksen ydin. Se toimii sovelluksen sisäänpääsynä. Luokka liimaa kaikki muut sovelluksessa olevat komponentit yhteen. Sovelluksessa voi olla ainoastaan yksi Activity-luokka kerrallaan ajossa. Kuvasta 14 nähdään toteuttavan sovelluksen Activityt. Vasemmalta katsottuna kaksi ensimmäistä käyttöliittymänäkymää vastaavat *HomeActivity*-luokkaa. Kolmas näkymä on *LoginActivity*-luokka.

Sovelluksien Activity-luokkia voidaan vaihtaa ohjelmakoodilla. Tiedotepalvelu-sovellus käynnistetään jokaisella käynnistyskerralla *HomeActivity*-luokkaan. Luokassa tarkistetaan onko käyttäjä kirjautunut sovellukseen, mikäli ei ole, sammutetaan *HomeActivity*-luokka ja avataan *LoginActivity*-luokka. Käyttäjätunnukset syötetään *LoginActivity*-luokassa, jotka tarkistetaan *AuthProseccor*-luokan avulla. Mikäli tunnukset ovat oikeat, ne tallennetaan Androidin Shared Preferences -luokkaan. Shared Preferences -luokkaan asetetaan arvo, joka sallii ainoastaan tämän sovelluksen käyttää asetettuja tietoja.

Shared Preferences -luokka tarjoaa sovelluksessa yksinkertaisten tietojen tallentamisen ilman tietokantaa. Tietoturvallisesti sähköpostiosoitteen ja salasanan tallentaminen käyttäen Shared Preferences -luokkaa ei ole kovinkaan turvallista, mutta opinnäytetyöhön riittävä. Tallennuksen jälkeen käyttäjän ei tarvitse kirjoittaa tunnuksia uudestaan jokaisella sovelluksen käynnistyskerralla, vaan ne pysyvät Androidissa muistissa. Tunnuksia pidetään tallessa siihen asti, kunnes käyttäjä kirjautuu ulos sovelluksesta.

Uloskirjautumisen yhteydessä SQLite-tietokanta alustetaan myös tyhjäksi. Activity-luokkaan voidaan lisätä Fragment-luokkia, jotka kapseloivat käyttöliittymänäkymän. Fragmentit lisäävät Activity-luokkaan koodin hallittavuutta, dynaamisuutta ja skaalautuvuutta monille eri näyttökooille (Meir 2012, 114).

Fragment- ja Activity-luokat kommunikoivat kahdella eri tavalla. Fragment-luokka tarvitsee rajapinnan, joka liitetään Activity-luokkaan *onAttach*-metodissa. Täten jokainen lähetetty objekti saadaan kaapattua Activity-luokassa Fragmentin rajapinnan avulla. Activity kommunikoi Fragment-luokan kanssa Bundle-luokan avulla. Bundle-

luokka lähetetään transactionin yhteydessä Fragment-luokalle. Kuvassa 14 näkyvä vasemmalta ensimmäinen listanäkymä on toteutettu Fragment-luokan avulla. Se sekä keskellä näkyvät tiedotteet sijaitsevat Fragment-luokassa. Sovelluksessa käytettiin myös muualla Fragment-luokkia. Kuvassa näkyvät listaelementit saadaan luotua Cursor Adapter -luokkaa hyödyntäen.

Cursor Adapter -luokkaan liitetään tietokannasta haettavat tietueet. Tietueet lisätään käyttäen Loader-luokkaa. Loader-luokasta toteutetaan rajapinta, joka tuo mukanaan metodit, joissa voidaan määrittää SQL-lause. Alustusmetodissa luodaan Cursor Loader -luokka, jolle määritetään tämä SQL-hakulause. SQL-lause hakee Content Providerin avulla SQLite-tietokannasta halutut tietueet näytettäväksi. Tietueet tuodaan sovelluksen käyttäjälle näytettäväksi ListView-näkymällä.

ListView-näkymä kustomoidaan luomalle sille kustomi Cursor Adapter -luokka. Cursor Adapter -luokalle luodaan periytetään luokka, kuten *NotificationAdapter*. Luokalle periytyy Cursor Adapter -luokan ominaisuudet.

NotificationAdapter-luokalla päästään käsittelemään yksittäistä listaelementtiä. Tällöin voidaan muokata listanäkymää halutunlaiseksi. Tiedotepalvelu-sovelluksessa kaksi eri listanäkymää luotiin omanlaisiksi, jotka näkyvät kuvassa 14. Sovelluksessa uuden tiedotteen luonti toteuttiin tapahtumaan kuvassa 14 näkyvästä plus-ikonista. Tiedotteen piilottaminen tapahtuu yksittäisestä action-ikonista.

Tiedotteet päivitetään automaattisesti ainoastaan sovelluksen käynnistäessä. Täten saadaan vähennettyä rasitusta REST-rajapinnalla. Kuvassa oikealla näkyvästä päivitä-ikonista käyttäjä voi päivittää halutessaan tiedotteet. Prosessia seurataan Broadcast Receiver -luokalla. Sillä voidaan välittää prosessin kulkua Activity-luokkaan prosessin aikana.

Broadcast Receiver -luokka kommunikoi Activity-luokan ja Intent Servicen kanssa. Luokkaa kuuntelee tapahtuuko toimintoja Intent Service -luokassa. Mikäli tapahtuu, käyttäjälle välitetään päivitetään-ikoni päivitys-ikonin sijaan. Toiminnan jälkeen palautetaan päivitys-ikoni takaisin käyttöön.

4 PÄÄTÄNTÖ

Opinnäytetyön onnistumiseen vaikutti minusta hyvin paljon projektisuunnitelman laatiminen. En ollut ennen opinnäytetyötäni ohjelmoinut näin suurta ohjelmistoprojektia. Olen yllättynyt, kuinka hyvin suunnittelemalla saadaan yhdistettyä kovin vaikeat yksittäiset asiat helpoksi kokonaisuudeksi. Projektisuunnitelmalla sain luottamusta itselleni ja selkeän kuvan tulevasta projektista. Suunnitelmaa katsomalla näin, mitä pitäisi seuraavaksi toteuttaa. Täten ei tarvinnut pähkäillä, mitä tekisin seuraavaksi. Projektisuunnitelman avulla saatiin myös rajattua projektin paisumista äärettömiin mittasuhteisiin.

Tutkimusongelmaani löysin ratkaisut lukemalla aiheeseen liittyvää kirjallisuutta. Kirjallisuudesta sain mallin, kuinka olisi hyvä ohjelmoida REST-rajapinta sekä Android-sovellus. Android-sovelluksen saaminen edes opinnäytetyössä saatuun valmiuteen tuskin olisi onnistunut, mikäli en olisi aloittanut tutustumista sen ohjelmointiin jo aiemmin opintojeni aikana.

Opinnäytetyössä minusta ongelmaksi muodostui suhteellisen kiire aikataulu. Työlle asetettiin 10 viikkoa aikaa saada se valmiiksi. REST-rajapinnan ohjelmointiin kului aikaa noin kolme viikkoa. Androidin ohjelmointiin meni saman verran aikaa. Lopulta kuitenkin sain toteutettua opinnäytetyölle asetetut minimitavoitteet, josta olen erittäin tyytyväinen.

Sain yritykseltä positiivista palautetta toteutetusta Tiedotepalvelu-sovelluksesta. He kommentoivat Android-sovelluksen noudattavan hyvin Googlen määrittelemiä ohjelinjoja käyttöliittymän toteutuksessa. Yrityksen mukaan sovellusta on nopea ja helppo käyttää.

Opinnäytetyön jälkeen aion jatkaa REST-rajapinnan ja Android-sovelluksen kehittämistä Ch5 Finland Oy:lla. Opinnäytetyöni aikana olen ohjelmoitaessa pitänyt mielessä ajatusta, että sovellusta tullaan jatkokehittämään opinnäytetyöni jälkeen. Ohjelmakoodi on pyritty pitämään mahdollisimman siistinä ja helposti laajennettavana. Olen kirjoittanut kommentteja runsaasti ja luokat kapseloivat aina tietyn kokonaisuuden. Yhdellä lauseella pystyy kuvastamaan yhden luokan toimintaperiaatteen Etenkin Android-sovellusta ohjelmoitaessa ohjelmakoodi alkaa

melko nopeasti muodostua erittäin vaikealukuiseksi ja hallitsemattomaksi. Ohjelmoidun koodin siisteys ja laajennettavuus luo edellytyksen jatkokehittämiseen.

Tiedotepalvelu-sovellukseen on tullut projektin aikana runsaasti uusia käyttötapoja ja laajennuksia. Sovellusta on tarkoitusta laajentaa ensin jo tunnetuille asiakkaille ja vasta sen jälkeen automatisoida se suuremmalle joukolle.

LÄHTEET

Android Developers 2014a. Application Fundamentals. WWW-dokumentti. http://developer.android.com/guide/components/fundamentals.html. Ei päivitystietoa. Luettu 20.10.2014.

Android Developers 2014b. Dashboards. WWW-dokumentti. https://developer.android.com/about/dashboards/index.html. Ei päivitystietoa. Luettu 20.10.2014.

Android Source 2014. ART and Dalvik. WWW-dokumentti. https://source.android.com/devices/tech/dalvik/index.html. Ei päivitystietoa. Luettu 21.10.2014.

Android Suomi 2014. Mikä on Android? WWW dokumentti. http://blog.androidsuomi.fi/mika-on-android/. Ei päivitystietoa. Luettu 23.7.2014.

Ayesha, .A 2012. Why is Android built on Linux Kernel? WWW-dokumentti. http://www.unixmen.com/why-is-android-built-on-linux-kernel/. Päivitetty 4.12.2012. Luettu 21.10.2014

Barry, Douglas K. 2014. Web Services Explained. WWW-dokumentti. http://www.service-architecture.com/articles/web-services/web_services_explained.html. Ei päivitystietoa. Luettu 22.9.2014.

Bauer, Christian & King, Gavin 2007. Hibernate in action, Java Persistence With Hibernate. Yhdysvallat: Manning Publications Co, 709-714.

Deleon, Walter 2013. A brief history of Android. WWW-dokumentti. http://technoblimp.com/2013/09/14/a-brief-history-of-android/. Päivitetty 14.9.2013. Luettu 23.7.2014.

Dobjanschi, Virgil 2010. Developing Android REST Client Applications. PDF-dokumentti. https://dl.google.com/googleio/2010/android-developing-RESTful-android-apps.pdf. Päivitetty 20.5.2010. Luettu 23.10.2014.

Elgin, Ben 2005. Google Buys Android for Its Mobile Arsenal. Businessweek. Verkkolehti. http://www.webcitation.org/5wk7sIvVb. Päivitetty 17.7.2005. Luettu 24.7.2014.

Elkstein, M. 2014. What is REST? Blogi. http://rest.elkstein.org/2008/02/what-is-rest.html. Ei päivitystietoa. Luettu 22.9.2014.

Google 2014. Alfa-/betatestauksen ja vaiheittaisen käyttöönoton käyttö. WWW-dokumentti. https://support.google.com/googleplay/android-developer/answer/3131213?hl=fi. Ei päivitystietoa. Luettu 20.10.2014.

Google Trends 2014. Explore. WWW-dokumentti. http://www.google.com/trends/explore?hl=en-US#q=soap+api,+rest+api&cmpt=q. Päivitetty 24.9.2014. Luettu 24.9.2014.

Hong, K. 2014. Web technologies - open APIS, SOAP, AND REST 2014. WWW-dokumentti. http://www.bogotobogo.com/WebTechnologies/OpenAPI_RESTful.php. Ei päivitystietoja. Luettu 24.9.2014.

JSON 2014. JSON: The Fat-Free Alternative to XML. http://www.json.org/xml.html. Ei päivitystietoa. Luettu 24.9.2014.

Meir, Reto 2012. Professional Android 4 Application Development. Kanada: John Wiley & Sons, Inc.

Metajack.im 2010. JSON versus XML: Not as Simple as You Think. Blogi. http://metajack.im/2010/02/01/json-versus-xml-not-as-simple-as-you-think/. Päivitetty 1.2.2010. Luettu 24.9.2014.

Morgan, Tom 2014. Android 5.0 Lollipop - release date, features & update plans, artikkeli. expert reviews. WWW-dokumentti. http://www.expertreviews.co.uk/phones-tablets/1401595/android-50-lollipop-release-date-features-update-plans. Päivitetty 19.10.2014. Luettu 20.10.2014.

Rouse, Margaret 2014. Google I/O. WWW-dokumentti. http://searchconsumerization.techtarget.com/definition/Google-I-O. Ei päivitystietoja. Luettu 2.10.2014.

Rozlog, Mike 2010. REST and SOAP: When Should I Use Each (or Both)? WWW-dokumentti. http://www.infoq.com/articles/rest-soap-when-to-use-each. Päivitetty 1.4.2010. Luettu 22.9.2014.

Sourabh 2014. Timeline History of Android: Smartphones That Were Launched With First Android OS Versions. WWW-dokumentti. http://sourcedigit.com/8219-timeline-history-android-smartphones-launched-first-android-os-versions/. Päivitetty 21.4.2014. Luettu 20.10.2014.

Tutorialspoint 2014. Android Architecture. WWW-dokumentti. http://www.tutorialspoint.com/android/android_architecture.htm. Ei päivitystietoa. Luettu 20.10.2014.

World Wide Web Consortium (W3C) 2004. Web Services Glossary. WWW-dokumentti. http://www.w3.org/TR/2004/NOTE-ws-gloss-20040211/. Päivitetty 11.2.2004. Luettu 22.9.2014.

Zazueta, Rob 2014. API Data Exchange: XML vs. JSON. WWW-dokumentti. http://www.mashery.com/blog/api-data-exchange-xml-vs-json. Päivitetty 23.1.2014. Luettu 24.9.2014.