Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Tietojärjestelmien kehitys

Joonas Ruotsalainen

Välimuistin ja hälytysten toteutus mobiilisovellukseen

Opinnäytetyö 2012

Tiivistelmä

Joonas Ruotsalainen
Välimuistin ja hälytysten toteutus mobiilisovellukseen, 62 sivua
Saimaan ammattikorkeakoulu
Tekniikka Lappeenranta
Tietotekniikka
Tietojärjestelmien kehitys
Opinnäytetyö 2012
Ohjaajat: Lehtori Mikko Huhtanen, Saimaan ammattikorkeakoulu,
toimitusjohtaja Mikko Mäkelä, Lean Development Oy

Tässä opinnäytetyössä on suunniteltu ja toteutettu välimuisti sekä hälytykset Sinser-mobiilisovellukseen. Sinser-palvelu koostuu websovelluksesta ja mobiilisovelluksesta. Se on kehitteillä oleva Lean Development Oy:n palvelu erilaisten asioiden, elämän, töiden ja muistioiden hallintaan.

Sinser-mobiilisovellus on toteutettu käyttäen Titanium Mobilea. Mobiilisovellus on yhteydessä Sinser-palvelimeen, jossa on käytetty tekniikoissa muun muassa MongoDB-tietokantaa ja JSON-tiedonsiirtomuotoa.

Välimuistin toteutuksen tavoitteena oli parantaa Sinser-mobiilisovelluksen käytettävyyttä ja vähentää tiedonsiirtoa. Välimuistin avulla mahdollistetaan myös mobiilisovelluksen käyttö, kun verkkoyhteyttä ei ole saatavilla. Opinnäytetyössä tutkittiin erilaisia vaihtoehtoja toteuttaa välimuisti. Tutkimusten pohjalta suunniteltiin toteutusvaihtoehdot, joista valittiin paras. Suunnitelman pohjalta toteutettiin yksinkertainen välimuisti, jota testattiin. Testien perusteella tehtiin päätelmät välimuistin tarpeellisuudesta mobiilisovellukselle.

Hälytysten tavoitteena oli saada käyttäjälle ilmoitus tärkeästä mobiilisovelluksen tapahtumasta mahdollisimman nopeasti. Tapahtumia ovat muun muassa muistioiden jakaminen tai viestit. Opinnäytetyössä tutkittiin ja vertailtiin eri vaihtoehtoja toteuttaa hälytykset. Tekniikaksi valittii push notification -teknologia, jonka avulla palvelin voi lähettää mobiililaitteelle lyhyitä viestejä. Push notification -teknologian palveluntarjoajaksi valittiin Google. Hälytykset toteutettiin mobiilisovellukseen käyttäen Github-palvelusta löytynyttä moduulia. Moduuli päivitettiin toimimaan Titanium mobilen uusimman version kanssa.

Opinnäytetyön tuloksena syntyi mobiilisovellukseen toimivat push notificationit, jotka testattiin ja todettiin onnistuneeksi. Välimuistia ei toteutettu ensimmäiseen julkaistavaan mobiilisovelluksen versioon, koska välimuisti ei merkittävästi lisännyt sovelluksen käytettävyyttä.

Asiasanat: Titanium Mobile, Android, JavaScript, push notification, välimuisti

Abstract

Joonas Ruotsalainen
Developing cache and alarms to mobile application, 62 pages
Saimaa University of Applied Sciences
Lappeenranta
Technology, Degree Programme In Information Technology
Information systems development
Bachelor´s Thesis 2012
Instructors: Lecturer Mikko Huhtanen, Saimaa University of Applied Sciences,
CEO Mikko Mäkelä, Lean Development Oy

The purpose of this thesis was to design and develop a cache and alarms to Sinser mobile application. Sinser is Lean Development Oy's product for managing tasks, task lists, work and life. The mobile application has been developed using Titanium Mobile. The server side of Sinser uses MongoDB and JSON format.

This thesis researched ways to develop a cache to the mobile application and how it could be developed with Titanium Mobile. The goal was to improve usability and minimize data transmission in the mobile application. The cache would also make offline use possible. A simple cache was developed for mobile application. It was tested and conclusions were made.

The goal in developing alarms was that the mobile application should alarm the user in some events as soon as possible. The thesis studied some technologies and push notifications were chosen to be the best option for Sinser mobile application. Push notification technology can be used to send short messages to the device. Push notifications are using Google's push notification service and were developed to the mobile application by using a module found from Github. The module was updated to work with the newest version of Titanium Mobile.

The result of this thesis was working push notifications in the mobile application. It was tested and stated successful. The cache was not developed to the first version of the mobile application because it did not improve significantly the usability of the application.

Keywords: Titanium Mobile, Android, JavaScript, push notifications, cache

Sisältö

Termit ja lyhenteet

Amazon EC2	Amazon Web Services LLC:n palvelu, joka tarjoaa pilvipalvelimia ja pilvilaskentaa.
Android market	Verkkopalvelu, jonne kehittäjät voivat lähettää omia sovelluksia, ja josta käyttäjät voivat niitä ladata.
Android-mobiilikäyttöjärjestelmä	Googlen kehittämä käyttöjärjestelmä älypuhelimille ja tablet-laitteille.
Ant	Ant on javapohjainen kääntötyökalu.
Appcelerator Inc	Yritys joka on tehnyt Titanium Studion ja Titanium Mobilen.
Attribuutti	Olioon tai objektiin kuuluva muuttuja.
Avoin lähdekoodi	Sovellusten kehitysmenetelmä, jossa sovelluksen lähdekoodi on kaikkien saatavissa.
BSON	MongoDB-tietokannan käyttämä objektimuoto.
Cache	Ks. välimuisti.
Callback-funktio	Funktio, joka annetaan parametrinä kutsuttavalle funktiolle ja jota kutsutaan kutsuttavasta funktiosta.
Drupal	Avoimen lähdekoodin sisällönhallintajärjestelmä.
Eclipse	Kehitysympäristö ohjelmien kehittämiseen.
Emulaattori	Ohjelma, joka mahdollistaa toiselle laitetyypille, kuten puhelimelle, tehtyjen ohjelmien ajamisen tietokoneessa.
Funktio	Ohjelmoinnissa käytettävä pienoisohjelma.
Github	Github Inc:n verkkopalvelu, joka tarjoaa lähdekoodinhallintaa ohjelmistokehitysprojekteille.
HTML	HyperText Markup Language on kuvauskieli internet sivujen sisällön esittämisessä.
HTML5	HTML-kuvauskielen viides versio.
HTTP-protokolla	HTTP eli Hypertext Transfer Protocol on protokolla tiedonsiirtoon web-palvelimen ja selaimen välillä.
ItemManager	Sinser-mobiilisovelluksessa oleva moduuli muistioiden hallintaa varten.

Java	Oliopohjainen ohjelmointikieli.
JavaScript	Prototyyppipohjainen ja dynaaminen scriptikieli.
JSON	JavaScript object notation eli JSON on tekstipohjainen tiedonsiirtomuoto.
Ketterät menetelmät	Ohjelmistokehitystapa.
LAMP	LAMP on sovelluskokoelma, joka koostuu Linux-käyttöjärjestelmästä, Apache-palvelimesta, MySQL-tietokannasta ja PHP-ohjelmointikielestä.
Lean Development Oy	Lappeenrantalainen yritys.
Lean software developement	Ks. ketterät menetelmät.
Mobiililaite	Helposti liikuteltava laite.
Mobiilisovellus	Mobiililaitteella, kuten älypuhelimella, käytettävä sovellus.
Moduuli	Sovelluksen tai järjestelmän toiminnallinen osa.
MongoDB	Avoimen lähdekoodin dokumenttipohjainen tietokanta.
Muistio	Sinser-verkkopalvelussa käytetty nimitys tehtävästä, kommentista, muistiinpanosta, ym.
Nimiavaruus	Nimiavaruus on abstrakti ympäristö, jolla jaetaan ohjelmakoodin osia, kuten luokkia, loogisiin joukkoihin.
Parametri	Sovellukselle tai funktiolle käynnistyksen yhteydessä välitettävä tieto.
PHP	PHP eli Hypertext Preprocessor on ohjelmointikieli erityisesti dynaamisten websivujen kehitykseen.
Pilvipalvelu	Verkossa toimiva palvelu, joka tarjoaa käyttäjälle jotakin resurssia, kuten laskentakapasiteettia.
Push notification	Palvelimen lähettämä lyhyt viesti mobiililaitteelle.
Relaatiotietokanta	Tietokanta, joka koostuu tauluista ja niiden yhteyksistä.
REST	HTTP-protokollaan perustuva arkkitehtuurimalli verkkosovelluksille.
Sekvenssikaavio	Olioiden välistä vuorovaikutusta kuvaava kaavio.
Selaimen lisäosa	Internet-selaimeen asennettava lisäosa, joka laajentaa

	tai muuttaa selaimen toimintaa tai tuo siihen lisäominaisuuksia.
Sinser	Lean Development Oy:n tuote muistioiden, tehtävien, ja asioiden hallintaan.
Skeema	Malli.
SQL	Kyselykieli, jolla relaatiotietokantaan voi tehdä hakuja, poistoja ja muutoksia.
SQLite	Kevyt ja pieni tietokanta, joka ei tarvitse erillistä palvelinta.
Tablet	Kosketusnäytön avulla käytettävä mobiili tietokone.
Tietokanta	Säiliö sovelluksen tai palvelun tiedoille.
Titanium Mobile	Appcelerator Inc:n kehittämä kokoelma kirjastoja ja rajapintoja mobiilisovelluskehitykseen.
URL	Uniform Resource Locator eli URL on merkkijono, jolla kerrotaan jonkin tiedon sijainti, kuten esimerkiksi www-sivun.
Verkkopalvelu	Verkossa sijaitseva websivusto tai palvelu.
Verkkoportaali	Ks. verkkopalvelu.
Välimuisti	Komponentti, joka varastoi usein haettua dataa mahdollistaen datan nopeamman saatavuuden.
Websovellus	Internet-selaimella käytettävä sovellus, joka ladataan verkon kautta.

1 Johdanto

Muutaman viime vuoden aikana älypuhelimet, mobiiliverkot sekä mobiilisovellukset ovat yleistyneet nopeaan tahtiin. Älypuhelimet ovat melko edullisia ja tarjontaa on runsaasti. Puhelimeen saa edullisesti myös lähes rajattoman internetliittymän. Mobiililiittymät ovat kehittyneet tarpeeksi nopeiksi mahdollistaen monipuoliset sovellukset, jotka ovat lähes jatkuvassa yhteydessä verkkoon ja pilvipalveluihin. Tämä on avannut monia uusia mahdollisuuksia yrityksille ja luonut lähes kokonaan uuden sovelluskehitysalueen. Tulevaisuudessa erilaiset mobiililaitteet ja sen myötä mobiilisovellukset tulevat olemaan tärkeitä. Tavallisten työpöytä- sekä websovellusten lisäksi halutaan myös mobiiliversiot sovelluksista.

Uskon mobiilisovellusten olevan yksi tärkeimmistä osa-alueista tietotekniikassa tulevaisuudessa. Tässä opinnäytetyössä olen ollut mukana Sinser-palvelun kehittämisessä ja erityisesti sen mobiilisovelluksen kehityksessä.

Opinnäytetyön alussa kerrotaan Lean Development Oy:stä sekä Sinser-palvelusta ja sen rakenteesta. Opinnäytetyössä on kerrottu muun muassa käytetyt tekniikat ja arkkitehtuuri. Välimuistista on tutkimusosio ja siinä kerrotaan muun muassa välimuistin mahdollistavista tekniikoista Titanium Mobilessa. Hälytyksissä on vertailtu eri tekniikoita sekä palveluntarjoajia. Lopussa kerrotaan lopputulos ja päätelmät.

2 Lean Development Oy

Tässä luvussa kerrotaan Lean Development Oy:sta sekä opinnäytetyön aiheen synnystä ja tarkoituksesta. Kerron myös, mitä hyötyä opinnäytetyön tekemisestä voi olla Sinser-mobiilisovellukselle.

2.1 Yrityksen toiminta

Lean Development Oy on lappeenrantalainen yritys, jonka tarkoituksena on tuottaa verkkoportaaleja ja -palveluita sekä sovelluskehitystä muille yrityksille. Lean Development Oy tarjoaa myös sovelluskehitystä mobiili- ja tablet-alustoille. Yrityksen strategiana on ymmärtää asiakkaan todelliset tarpeet ja kehittää

valmis tuote tai palvelu mahdollisimman nopeasti ilman turhia ylimääräisiä välivaiheita ja byrokratiaa, joita monissa nykypäivän projekteissa on. Yritys käyttää ohjelmistokehityksessään ketteriä menetelmiä ja palveluissaan moduuliperustaista kehitystä. Avoimen lähdekoodin käyttäminen asiakasprojekteissa ei sido asiakasta yhteen toimittajaan, ja tämä takaa asiakkaalle riskittömän yhteistyön Lean Development Oy:n kanssa. Avoin lähdekoodi parantaa myös kehityksen joustavuutta, koska komponentteja voi vapaasti muokata omien ja asiakkaan tarpeiden mukaan. Lean Development Oy tarjoaa asiakkailleen myös apuja ja neuvoja ideoiden toteuttamisessa sekä konsultointia internetperustaiselle liiketoiminnalle. (Lean Development Oy 2011)

Lean Development käyttää nimensä mukaisesti niin sanottua ”lean software development” -kehitystapaa projekteissaan. Perinteisen mallin mukaisissa ohjelmistokehitysprojekteissa käytetään paljon aikaa määrittelyyn ja suunnitteluun, jolloin mahdollistetaan selkeä kehitysprosessi valmiille tuotteelle. Tämä on kuitenkin hankala muutosten kannalta. Asiakkaalle tulee usein mieleen uusia ominaisuuksia tai kesken projektin havaitaan uusia tarpeita, joita tarvittaisiin valmiissa tuotteessa tai palvelussa. Muutokset ovat vaikea toteuttaa ja tulevat kalliiksi, jos projekti on ehtinyt jo pitkälle. ”Lean software development” -kehitystavassa nämä ongelmat on pyritty poistamaan ymmärtämällä asiakkaiden todelliset liiketoiminnalliset tarpeet ja kehittämällä nopeasti valmista tuotetta asiakkaan nähtäväksi. Kaikki ylimääräinen on pyritty karsimaan ja komponenttien uudelleenkäyttöä on pyritty suosimaan. Tämä mahdollistaa joustavan ohjelmistokehitystyön, jossa tärkeimmät ominaisuudet toteutetaan ensin ja uusia ominaisuuksia voi lisätä helposti ja joustavasti, ilman suuria lisäkuluja.

Lean Development Oy:n avainteknologioihin kuuluvat Drupal, LAMP, MongoDB, HTML5, selainlisäosat ja pilviteknologiat. Yrityksellä on vahva osaaminen esimerkiksi Amazon EC2 -pilvipalvelusta sekä LAMP-palvelimien ylläpidosta. Osaamista löytyy myös uudenlaisista teknologioista, kuten dokumenttipohjaisesta MongoDB-tietokannasta. Avoimen lähdekoodin sisällönhallintajärjestelmä Drupal toimii yrityksen web-palveluiden ja verkkoportaalien pääkehitysalustana. (Lean Development Oy 2011)

Lean Development Oy ja sen edeltäjä Tiksis Technologies Oy ovat olleet jo melkein vuoden työnantajiani. Olen työskennellyt näissä yrityksissä yhden ison

verkkoportaalin toteutuksessa ja muutamissa pienemmissä töissä. Kesällä 2011 menin mukaan Sinserin kehitystyöhön. Aluksi sain siitä opinnoissani vaadittavan projektityön ja nyt opinnäytetyöaiheen.

2.2 Aiheesta ja sen synnystä

Opintoihini kuului projektityö, johon sain aiheen Lean Development Oy:ltä. Projektityön aiheena oli tehdä mobiilisovellus yrityksen Sinser-verkkosovelluksesta. Sinseristä kerrotaan tarkemmin tässä opinnäytetyössä luvussa 3.

Mobiilisovellus ei tullut valmiiksi projektityön aikana. Jatkoin tämän jälkeen mobiilisovelluksen kehitystä muiden töiden ohella Lean Development Oy:ssä. Arvelin Sinserin mobiiliversion olevan sopiva opinnäytetyön aihe, koska siinä oli vielä tekemistä ja hyvän opinnäytetyöaiheen saisi rajaamalla sopivan osa-alueen mobiilisovelluksesta. Tiesin, että mobiilisovelluksessa kannattaisi olla välimuisti tiedonsiirron rinnalla, joten valitsin aiheeksi välimuistin suunnittelemisen ja toteuttamisen mobiilisovellukseen. Aihetta suunniteltiin ja pohdittiin, jonka jälkeen päätettiin jättää tutkimusta vähemmälle ja keskittyä itse suunnittelu- ja toteutustyöhön. Tarkoituksena on myös lisätä hälytyksien toteuttaminen sovellukseen. Tein aiheesta esitutkimuksen ja anomuksen. Aloittamaan pääsin joulukuun alussa.

2.3 Opinnäytetyön tarkoitus

Opinnäytetyön tarkoituksena on suunnitella ja toteuttaa välimuisti sekä hälytykset Sinser-mobiilisovellukseen. Välimuisti toteutetaan paremman käyttäjäkokemuksen saamiseksi sekä tiedonsiirron minimoimiseksi. Mobiiliverkot voivat olla välillä hitaita tai yhteyttä internetiin ei aina saada. Tällöin välimuistin avulla pystytään mahdollistamaan mobiilisovelluksen väliaikainen käyttö ilman internetyhteyttä. Käyttäjäkokemus myös paranee, kun käyttäjän ei tarvitse odottaa sovelluksen hakiessa palvelimelta tietoa. Kun käyttäjäkokemus on hyvä, haluavat ihmiset käyttää sovellusta työssään ja vapaa-ajalla. Hyvä käyttäjäkokemus parantaa myös tuotteen kilpailukykyä muihin samantyylisiin mobiilisovelluksiin verrattuna.

Hälytyksien suunnittelemisen ja toteuttamisen tarkoituksena on ratkaista ongelma, kuinka käyttäjää voidaan huomauttaa esimerkiksi viestin saapuessa. Olem-

me katsoneet hälytykset yhdeksi mobiilisovelluksen vaadituista ominaisuuksista.

3 Sinser

Tässä luvussa kerrotaan vielä kehitteillä olevasta Sinser-webpalvelusta ja -websovelluksesta sekä Sinser-mobiilisovelluksesta. Molemmat sovellukset tulevat muuttumaan kehityksen aikana. Luvussa kerrotaan sovellusten toiminnoista ennen opinnäytetyön toteutusvaihetta. Tätä kirjoittaessa sovellukset ovat menneet jo kehityksessä eteenpäin.

3.1 Tietoa tuotteesta

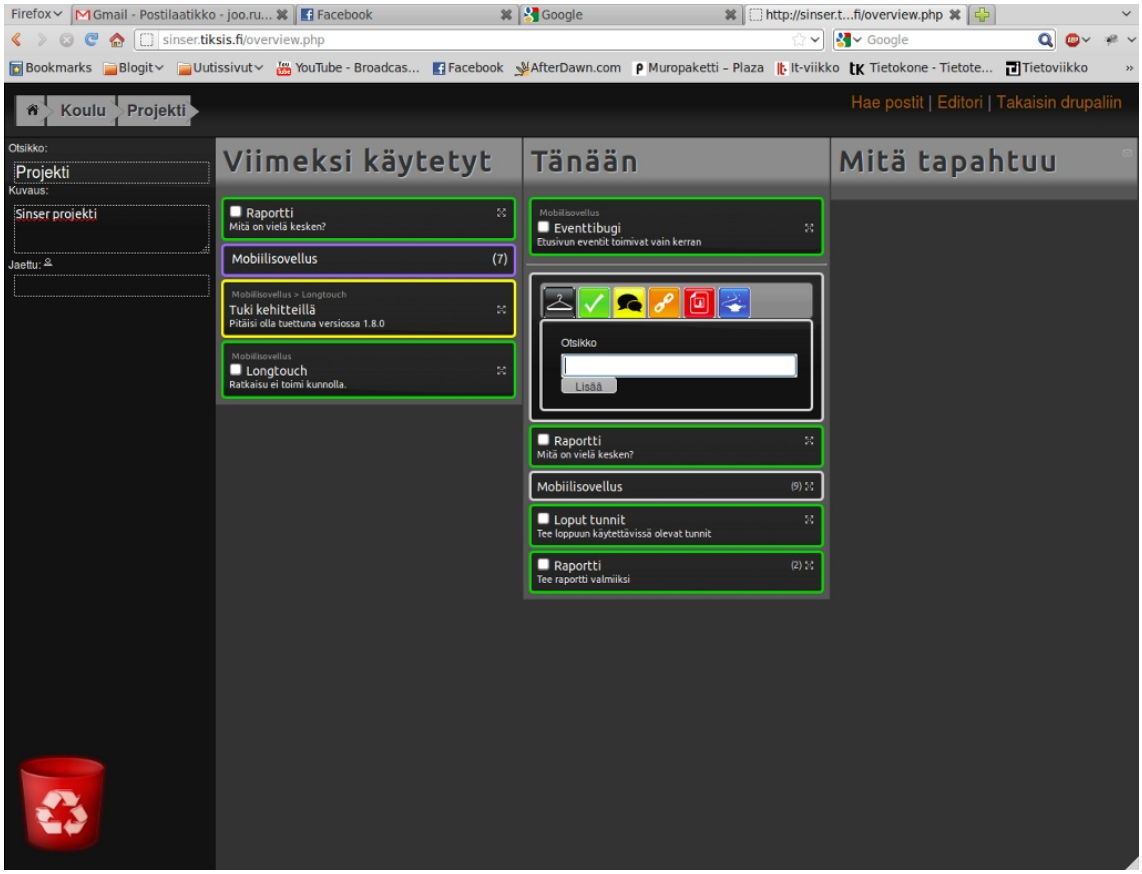
Sinser on webpalvelu ja -sovellus, jossa käyttäjät voivat tehdä itselleen muistioita. Muistiot voivat koostua tehtävistä, tekstistä, kuvista, tiedostoista ja kommenteista sekä muunlaisista elementeistä. Käyttäjä voi rakentaa muistion itselleen vapaasti ja järjestellä sekä organisoida niitä haluamallaan tavalla. Käyttäjä voi esimerkiksi tehdä muistion, johon kuuluu muutama tehtävä, ja tehtäviin liittyy pari kuvaa. Käyttäjä Matti voi luoda itselleen koulun ohjelmointitehtävästä muistion. Hän lisää siihen tehtävän ”Koodaa ominaisuus X valmiiksi”. Tehtävään hän liittää kaverin tekemän lähdekooditiedoston, jota Matti ajatteli käyttää apuna koodauksessa. Tiedostoon hän liittää pari kommenttia lähdekoodista. Lopuksi hän liittää vielä linkin eräälle sivustolle, josta hän löysi ratkaisun erääseen algoritmiin. Käyttäjä Matti on siis tehnyt itselleen muistion, johon kuuluu tehtävä, tiedosto, pari kommenttia sekä linkki.

Käyttäjä voi lisätä kavereitaan palveluun ja jakaa haluamiaan muistioita palvelussa olevien kavereidensa kanssa. Käyttäjät voivat myös lähettää palvelun välityksellä viestejä toisilleen. Palvelun viestit ovat integroituna Googlen gmail-sähköpostiin.

Sinser-verkkopalvelusta on kehitteillä webversio ja mobiiliversio. Kahdessa seuraavassa luvussa kerrotaan näistä.

3.2 Sinser-websovellus

Sinser-verkkopalvelun webversiossa (Kuva 1) käyttäjä voi hallita ja luoda omia muistioitaan. Muistioista voi muodostaa puumaisia rakenteita.



Kuva 1. Sinser-websovellus

Kuvassa 1 näkyy kuvankaappaus Sinser-websovelluksen kehitysversiosta. Kuvankaappauksessa näkyvät listat Viimeksi käytetyt, Tänään ja Mitä tapahtuu. Yllä näkyy polku Koti, Koulu, Projekti. Tämä tarkoittaa, että olemme katsomassa Projekti-muistiota, joka on Koulu-muistion alla. Koulu-muistio taas on kohdan Koti alla. Koti symboloi muistiohierarkian juurta ja ylimmäisiä muistiota eli muistioita, jotka eivät ole minkään muistion alla. Koulu-muistio on siis ylin muistio. Kuvasta saattaa myös nähdä, että Projekti-muistioon kuuluvat muistiot Raportti, Mobiilisovellus, Loput tunnit ja Raportti. Raportti-muistio on mainittu kahteen kertaan, koska Projekti-muistio sisältää kaksi samannimistä muistiota.

Muistiota voi vapaasti järjestellä haluamallaan tavalla tai siirtää Tänään-listaan. Tänään-lista auttaa käyttäjää hallitsemaan omia asioitaan. Käyttäjä voi tehdä esimerkiksi itselleen tehtäviä, jotka on tehtävä jonain tiettynä päivänä. Aamulla

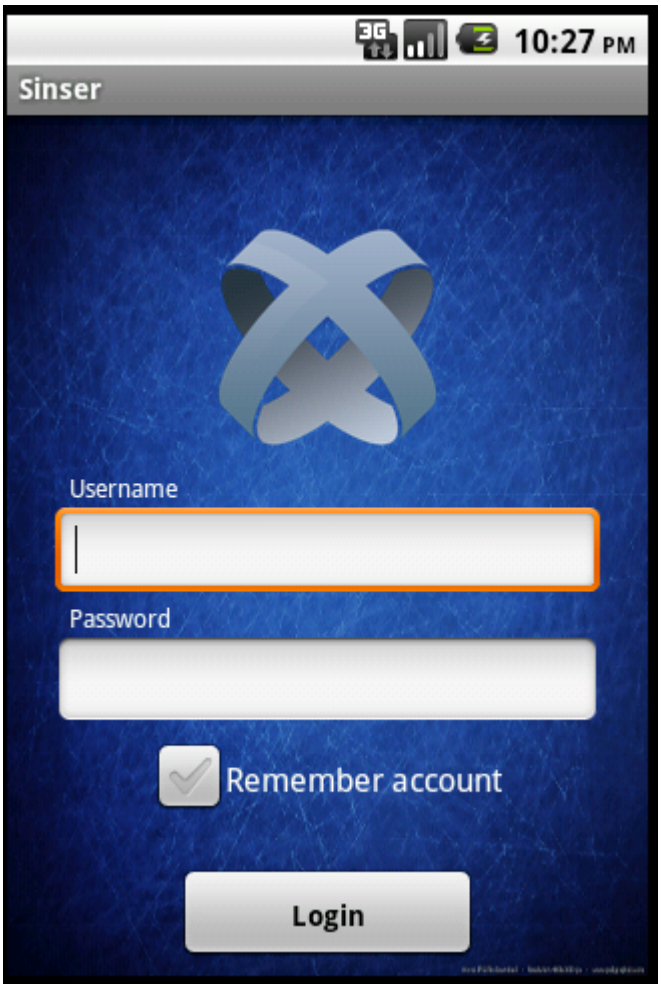
hän siirtää päivän tehtävät Tänään-listaan ja merkitsee tehtäviä tehdyksi päivän aikana. Listasta käyttäjä voi siis helposti nähdä, mitkä tehtävät ovat vielä jäljellä.

Muistioita voi luoda kuvassa näkyvästä laatikosta. Laatikosta pitää valita ensin muistion tyyppi, kuten tehtävä tai linkki. Tämän jälkeen täytetään muistion tyypin vaatimat kentät, kuten linkin osoite. Lopuksi tallennetaan muistio. Muistiota voi muokata ja jakaa kuvan vasemmassa laidassa olevalla alueella ja poistaa vetämällä muistion vasemman alareunan roskakoriin.

Websovelluksen toiminta ja ulkoasu voivat vielä muuttua paljon, koska monia asioita on vielä suunnittelematta ja toteuttamatta. Projektin aikana voi tulla uusia ominaisuuksia ja ideoita, jotka voivat aiheuttaa muutoksia sovellukseen.

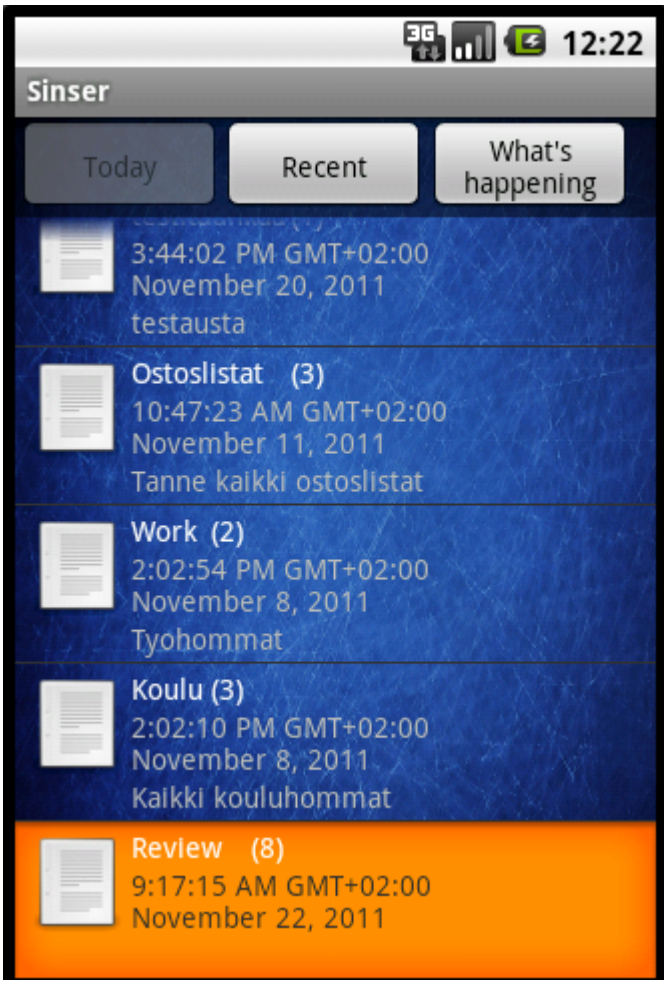
3.3 Sinser-mobiilisovellus

Sinser-mobiilisovellus (Kuvat 2 ja 3) on älypuhelimella käytettävä versio palvelusta. Mobiilisovellukseen on pyritty toteuttamaan samat ominaisuudet kuin webversioonkin, mutta mobiilia käyttöä ajatellen ja älypuhelimien rajoitteet huomioon ottaen. Sovellus on kehitteillä Android-käyttöjärjestelmälle.



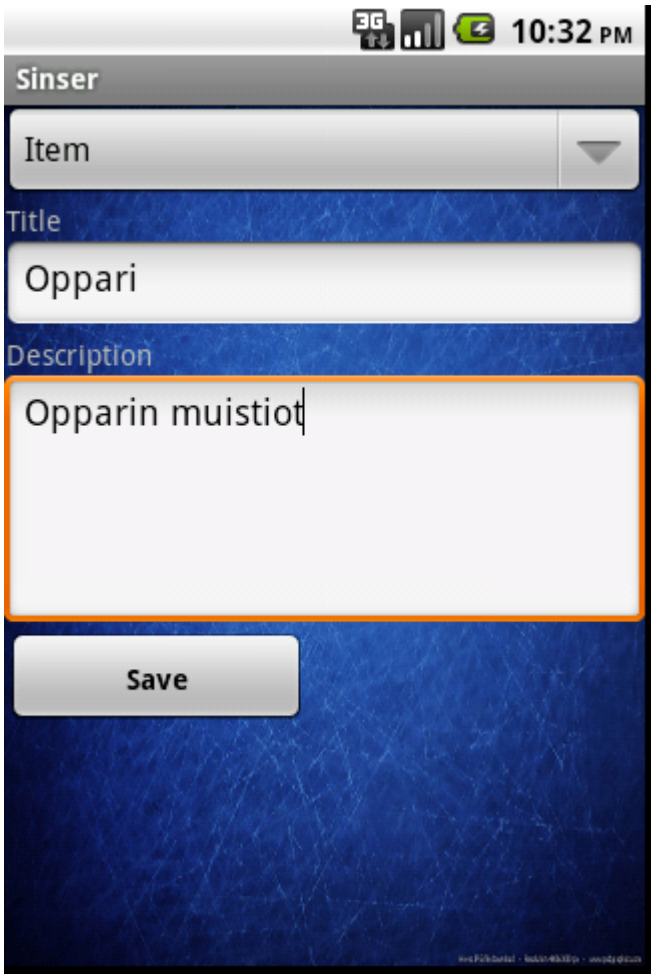
Kuva 2. Mobiilisovelluksen kirjautumisikkuna

Kuvassa 2 näkyy sovelluksen käynnistyttyä käyttäjälle näkyvä kirjautumisikkuna. Käyttäjä voi halutessaan tallentaa oman käyttäjätilinsä, jolloin seuraavalla käynnistyskerralla sovellus kirjautuu automaattisesti. Kirjautumisikkunan logo ei ole Sinser-mobiilisovelluksen logo, vaan Titanium-Studion oletuslogo projekteissa. Logoa ei ole vielä kehitysversiossa vaihdettu.



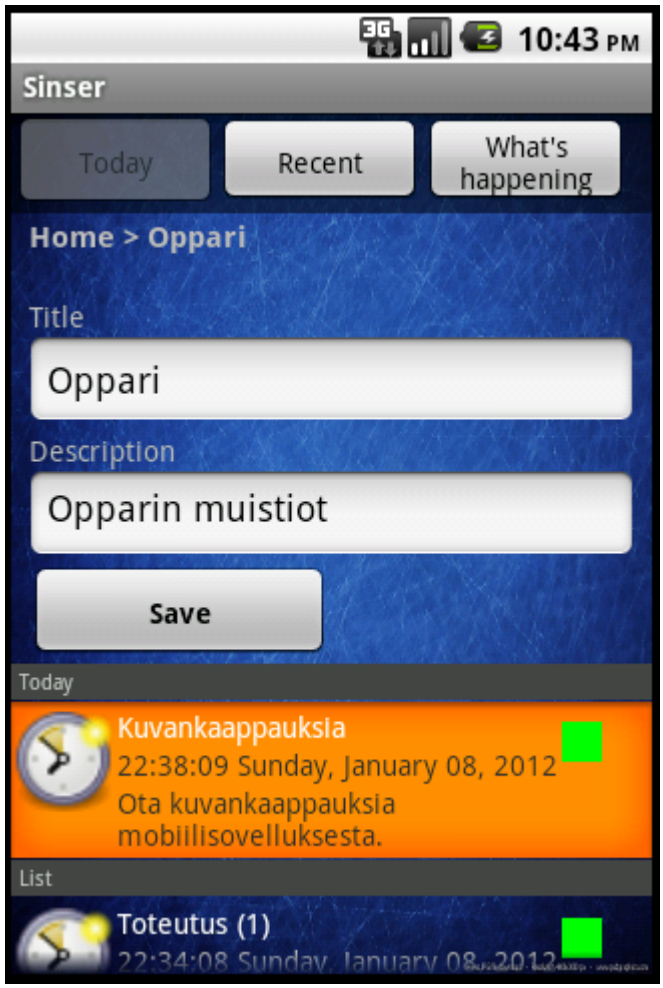
Kuva 3. Sinser-mobiilisovellus

Kuvassa 3 näkyy kirjautumisen jälkeen avautuva pääikkuna. Ikkunasta voi tässä kehitysversiossa valita välilehdet Tänään, Viimeisimmät ja Mitä tapahtuu. Tänään-välilehdellä näkyy Tänään-listan muistiot sekä muistiot, jotka kuuluvat ylimmäiselle tasolle. Ne ovat siis muistiohierarkian ylimmät muistiot. Puhelimen menupainikkeesta saa auki valikon, josta voi luoda uuden muistion (Kuva 4).



Kuva 4. Uuden muistion luominen

Kuvassa 4 näkyy uuden muistion lisäysnäkymä. Ulkoasu on vielä kesken kehitysversiossa ja muutoksia voi tulla. Lomakkeen ensimmäisestä kohdasta valitaan muistion tyyppi, kuten tehtävä tai linkki. Lomakkeen kentät muuttuvat valitun muistion tyypin mukaan. Muistion tallentamisen jälkeen sen voi avata listasta (Kuva 5).



Kuva 5. Avattu muistio

Kuvassa 5 näkyy avattu muistio. Muistioita voi avata valitsemalla listasta. Avattua muistiota voi muokata tekstikentistä. Tekstikenttiä näkyy muistion tyypin mukaan. Välilehtipainikkeiden alapuolella näkyy avatun muistion polku. Muistiosta näkyy kehitysversiossa nimi, kuvaus, päiväys ja tyypin kuvake. Neliö oikeassa laidassa tarkoittaa tehtävissä tehtävän tilaa. Tehtävä on tehty tai kesken. Tämän on kuitenkin kehitysversiossa väliaikainen ratkaisu eikä sitä tule valmiiseen julkaistavaan versioon. Kuvasta näkee myös listan nimet. Valittuna on Today-välilehti ja siinä näkyvät tässä kehitysversiossa listat Today ja List.

4 Sinserin rakenne ja tiedonsiirto ennen toteutuksia

Tässä luvussa kerrotaan Sinser-mobiilisovelluksen rakenne ennen välimuistin ja hälytysten toteuttamista. Mobiilisovelluksen tila vastaa tässä luvussa aikaa ennen ja hieman jälkeen opinnäytetyön aloittamisesta joulukuun 2011 alussa. Lu-

vussa käsitellään myös käytettyjä tekniikoita ja kehitystyökaluja sekä Sinser-palvelun arkkitehtuuri.

4.1 Käytetyt tekniikat

Tässä luvussa kerrotaan tärkeimmistä tekniikoista ja työkaluista, joita on käytetty Sinser-palvelun ja mobiilisovelluksen kehittämisessä. Luvussa 4.1.1 kerrotaan käytetystä työskentelymenetelmästä. Luvussa 4.1.2 kerrotaan mobiilisovelluksen kehityksessä käytetyistä työkaluista sekä tekniikoista. Luvuissa 4.1.3, 4.1.4 ja 4.1.5 kerrotaan Sinser-palvelun rajapinnan sekä palvelimen tekniikoista, jotka ovat tärkeässä osassa mobiilisovellusta kehitettäessä.

4.1.1 Ketterät menetelmät

Sinser-projektissa on käytetty ketteriä menetelmiä. Ketterissä menetelmissä pyritään poistamaan kehitystyöstä kaikki ylimääräinen, josta ei ole hyötyä asiakkaalle tai projektille. Mitään erillisiä määrittelydokumentteja ei ole, vaan projektia vietiin eteenpäin lyhyiden scrum-kehitystavan tapaisten suunnittelupalavereiden pohjalta. Minulla on ollut myös melko vapaat kädet mobiilisovelluksen toteuttamisessa. Olemme sopineet ominaisuuksista, joita toteutamme Sinseriin. Tämän jälkeen olen suunnitellut ja toteuttanut kyseiset ominaisuudet mobiilisovellukseen. Teemme työtä itsenäisesti ja välillä scrum-kehitystapaan kuuluvissa lyhyissä sprinteissä. Sprint kestää muutamasta päivästä kuukauteen ja sen aikana tehdään jokin ominaisuus valmiiksi (Sininen meteoriitti 2012).

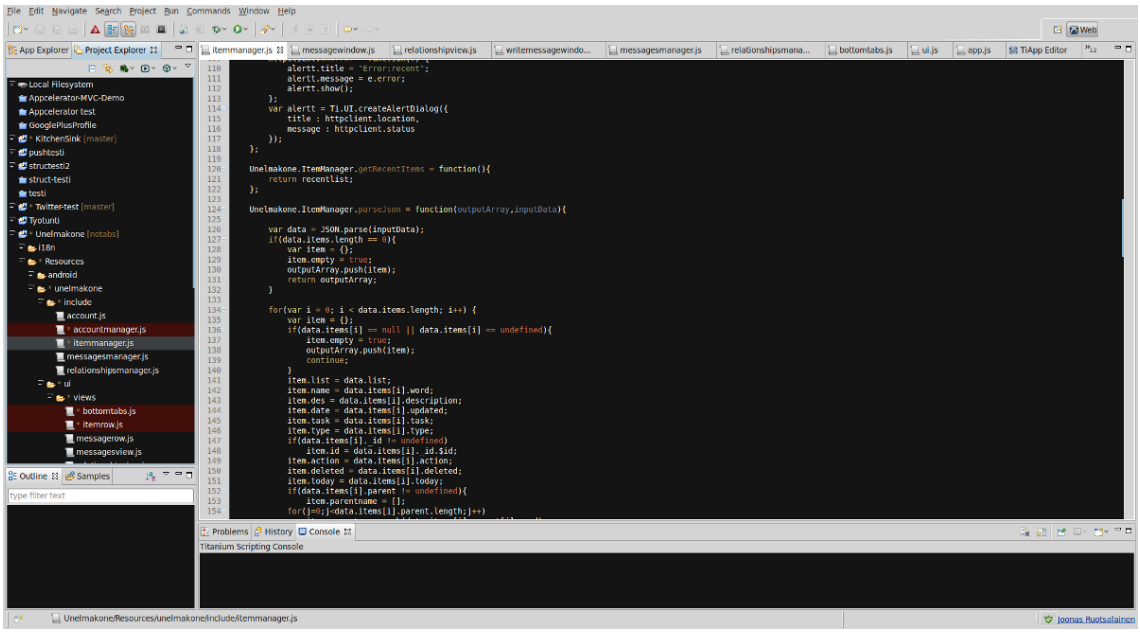
Ketterissä menetelmissä pääperiaatteena on käydä läpi kaikki ominaisuudet ja tavoitteet, priorisoida näistä tärkeimmät ja keskittyä niihin. Kaikki epävarmat asiat pyritään minimoimaan, jolloin päätökset voidaan tehdä mahdollisimman myöhään.

Ominaisuuksia toteuttaessani olen etsinyt tietoa internetistä sekä käyttänyt apuna Titanium Mobilen dokumentaatiota. Keskustelupalstoilta löytyneet keskustelut ovat auttaneet myös toteutuksessa. Muiden kehittäjien kokemukset erilaisista tekniikoista ovat vaikuttaneet päätöksentekooni valitsemissani mobiilisovelluksen teknisissä ratkaisuissa.

4.1.2 Titanium Mobile ja Titanium Studio

Titanium Studio on Appcelerator-nimisen yrityksen tekemä kehitystyökalu natiivien mobiilisovellusten ohjelmoimiseksi eri mobiilikäyttöjärjestelmille, kuten Androidille ja iOS:lle. Titanium Studiolla voi tehdä myös työpöytä- ja tablet-sovelluksia. Sovelluksia kehitetään websovelluskehityksessä tutuilla tekniikoilla ja ohjelmointikielillä kuten JavaScriptillä, HTML-kuvauskielellä, CSS-tyylimäärittelyillä sekä Python-, Ruby-, ja PHP-ohjelmointikielillä (Appcelerator 2011). Titanium Studion hyötynä on, että ohjelmakoodi tarvitsee tehdä vain kertaalleen, eikä erillistä koodia tarvitse tehdä Androidille ja iPhonelle. Pienillä ohjelmakoodin muokkauksilla sovelluksen saa toimimaan molemmilla eri alustoilla.

Titanium Studio (Kuva 6) on Eclipse-kehitysympäristö pohjainen. Kaikki Eclipsen ominaisuudet ovat mukana ja lisäksi kehitysympäristöön on integroitu muun muassa Git-versionhallinta. Kehitysympäristö sisältää työkalut mobiilikehitykseen. Älypuhelimen voi helposti yhdistää ja käyttää sovelluksien testaamiseen.



Kuva 6. Titanium Studio

Kuvassa 6 näkyy kuvankaappaus Titanium Studiosta. Kehitysympäristön asettelu on samantapainen kuin kaikissa yleisimmissä kehitysympäristöissä. Projektin tiedostot näkyvät vasemmalla ja koodi keskellä. Kehitysympäristö vaatii, että mobiilialustan omat työkalut ovat asennettuina tietokoneelle. Mobiilialustojen eri kehitystyökaluihin kuuluu muun muassa emulaattori (Kuva 7).

Kuva 7. Android-emulaattori

Kuvassa 7 näkyy Android-emulaattori, jota voi käyttää mobiilisovelluksien kehitykseen ja testaamiseen. Titanium Studion mukana tulee virtuaalikone, jota käytetään Androidin omalla emulaattorilla.

Titanium Mobile on kirjasto, joka muuntaa web-kehityksessä käytettävät kielet Objective-C- tai Java-ohjelmointikielille ja siitä Android- tai iOS-mobiilisovelluksiksi. Titanium Mobile sisältää yli 300 rajapintaa, joiden avulla mobiilikehitystyötä voi nopeuttaa (Appcelerator 2011). Titanium Mobilen avulla pääsee käyttämään melkein kaikkia ominaisuuksia, joita natiivilla kehitystyökalullakin pääsee käyttämään. Titanium Mobileen voi myös itse toteuttaa natiiveja moduuleita, jotka voi liittää omiin Titanium Mobile -projekteihin. TItanium Mobilella on myös helppo tehdä pilviteknologioita hyödyntäviä mobiilisovelluksia.

Titanium Studio sisältää siis työkalut mobiilisovellusten kehitykseen ja Titanium Mobile kirjastot sekä rajapinnat.

4.1.3 JSON

JSON eli JavaScript Object Notation on tekstipohjainen ja kevyt tiedonsiirtomuoto. Se on helposti ihmisen luettavissa ja lisäksi se ei ole riippuvainen ohjel-

mointikielestä. JSON määrittelee sarjan sääntöjä datan rakenteelliselle esittämiselle (RFC4627, 2011). JSON:n kehityksen tavoitteena on luoda minimaalinen, siirrettävä ja tekstipohjainen JavaScriptin osa (RFC4627, 2011).

JSON tukee neljää primitiivistä tietotyyppiä string, boolean, number ja null sekä kahta rakenteellista tyyppiä object ja array. Seuraavassa esimerkissä on määritelty JSON-muotoinen objekti.

```
{
   ”id”: 1,
   ”Nimi”: ”Joonas”,
   ”Pituus”: 187.5,
   ”Kotikaupunki”: ”Lappeenranta”,
   ”SisarusId”: [12, 43, 60],
   ”Kuva”: {
      ”Leveys”: 50,
      ”Korkeus”: 50
   },
   ”Opiskelijastatus”: true,
   ”Lemmikit”: null
}
```

Aaltosulut tarkoittavat esimerkissä objektia. Niiden jälkeen tulevat avaimet, joista jokaisella on yksi tieto. Esimerkiksi Nimi-avaimen tieto on Joonas ja Kotikaupunki-avaimen Lappeenranta. SisarusId-avaimen tieto on taulukko, jossa on arvot 12, 43 ja 60. Kuva-avaimen tieto on objekti, jolla on avaimet Leveys ja Korkeus. OpiskelijaStatus on boolean-tyyppinen tietotyyppi ja Lemmikit-avaimen arvo on taas null. Null tarkoittaa sitä, että arvoa ei ole.

Sinserin mobiilisovelluksessa kaikki tiedonvälitys palvelimen rajapinnan sekä mobiilisovelluksen välillä tapahtuu JSON-muodossa. Esimerkiksi haettaessa muistioita, palvelin palauttaa JSON-muotoisen taulukon, jossa muistiot ovat objekteja.

4.1.4 MongoDB

MongoDB on C++-ohjelmointikielellä tehty avoimen lähdekoodin tietokanta (MongoDB, 2011). Se on tehokas ja skaalautuva sekä perinteisistä tietokannoista poiketen dokumenttipohjainen ja skeematon. Skeematon tarkoittaa, että MongoDB-tietokannalle ei tarvitse etukäteen tehdä tauluja, vaan dataa voi tallentaa tietokantaan BSON-dokumenttimuodossa. BSON on JSON-pohjainen datan esitysmuoto, jossa JSON:a on laajennettu muun muassa uusilla ja tarkemmilla tietotyypeillä, kuten päivämäärä- tai byte-tietotyypeillä. MongoDB tietokanta ei

myöskään käytä relaatiotietokannoista tuttua SQL-kieltä, vaan haut tehdään JavaScriptillä. Seuraavassa esimerkissä tallennetaan ja haetaan tietoa MongoDB-tietokannasta.

```
> data = {”Nimi” : ”Joonas”};
{”Nimi” : ”Joonas”}
> db.henkilot.save(data);
> db.henkilot.find();
{ "_id" : ObjectId("4c2209f9f3924d31102bd84a"), ”Nimi” : ”Joonas”}
```

Esimerkissä suurempi kuin -merkillä alkavat rivit tarkoittavat komentoa ja muut rivit tulostusta. Ensimmäisellä rivillä data-muuttujaan tallennetaan objekti. Tallennus tulostaa objektin seuraavalle riville. Objekti tallennetaan tietokantaan data-muuttujan avulla kolmannella rivillä. Save-funktio tallentaa tietokantaan. Db on käytettävän tietokannan nimi ja henkilot on kokoelma, johon data-muuttujan objekti tallennetaan. Neljännellä rivillä kaikki db-tietokannan henkilo-kokoelman objektit haetaan. Find-funktion avulla voi hakea tietoja tietokannasta. Viimeisellä rivillä näkyy find-funktion tulostus. Tallennetulle objektille on muodostunut automaattisesti _id-avain, jonka avulla tallennettu objekti voidaan tunnistaa.

Sinser-verkkopalvelussa käytetään MongoDB-tietokantaa palvelimella. Muistiot ovat tallennettuina MongoDB-tietokantaan.

4.1.5 REST-arkkitehtuuri

REST-arkkitehtuuri on HTTP-protokollaan perustuva malli verkkosovelluksille. REST-arkkitehtuuria ei kuitenkaan ole sidottu pelkästään HTTP-protokollaan, mutta se on alunperin suunniteltu käyttämään sitä. REST-arkkitehtuuriin kuuluu palvelin- ja asiakassovellus. Asiakassovellus lähettää palvelimelle pyyntöjä ja palvelin vastaa niihin sopivalla paluuviestillä. Palvelinsovelluksen vastaus riippuu asiakassovelluksen käyttämästä HTTP-metodista sekä polusta eli URL-osoitteesta. URL-osoite kertoo palvelimen ja resurssin sijainnin internetissä. HTTP-metodit, joita REST-arkkitehtuurissa käytetään, ovat GET, POST, DELETE ja PUT. GET-metodilla haetaan tietoa, POST-metodilla luodaan uutta, DELETE-metodilla poistetaan ja PUT-metodilla päivitetään. Palvelin lähettää paluuviestin yleensä JSON- tai XML-muodossa. REST-arkkitehtuurissa polut suositellaan tehtäväksi hakemistopuun tapaiseksi (IBM 2011). Sinser-verkkopalvelun rajapinta on toteutettu REST-arkkitehtuurin mukaisesti. Seuraavassa esimerkissä kerrotaan kuinka asiakassovellus hakee palvelimelta muistioita.

Asiakassovellus on aluksi tunnistautunut käyttäjätunnuksella ja salasanalla. Palvelin on vastannut antamalla käyttäjätiedot ja käyttäjän istuntotunnuksen. Asiakassovellus tallentaa istuntotunnuksen ja asettaa sen evästeeksi seuraavia pyyntöjä varten. Asiakassovellus voi hakea muistiot palvelimelta seuraavanlaisella pyynnöllä:

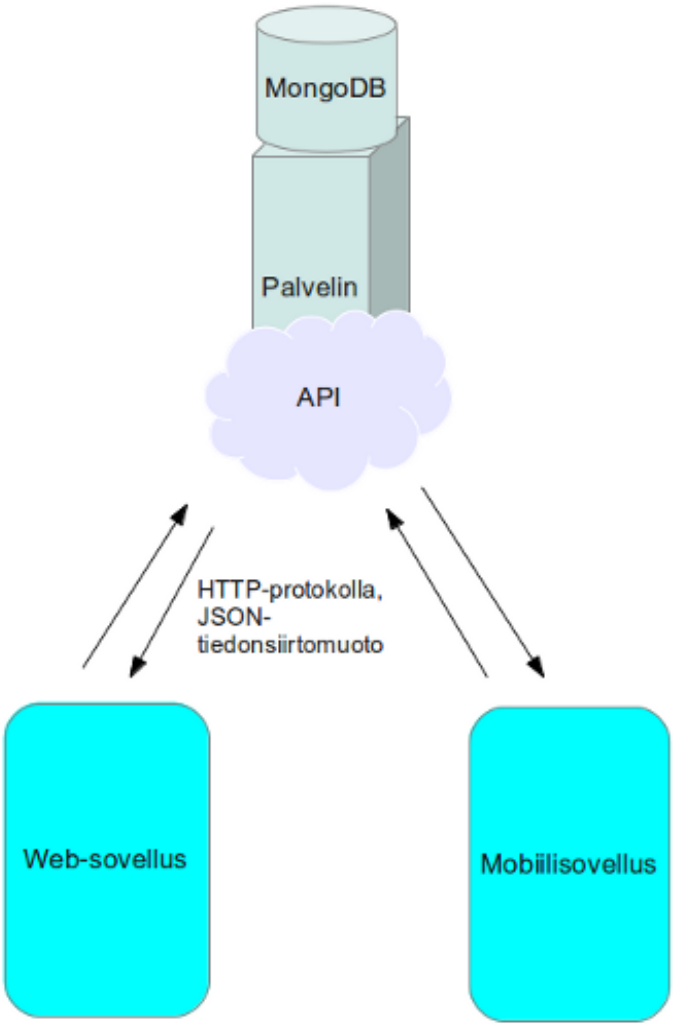
```
GET /sinser/api/list(/:id(/:group))
```

Muistiot on haettava GET-metodia käyttäen polusta /sinser/api/list. Polun perään voi laittaa id- ja group-parametrin, jos on tarvetta hakea muistioita id:n perusteella.

4.2 Arkkitehtuurin kuvaus

Sinser-verkkopalvelun arkkitehtuuri on yksinkertainen (Kuva 8). Palvelun data sijaitsee Amazon EC2 -pilvipalvelimella. Amazonin pilvipalvelut mahdollistavat helpon ja kustannustehokkaan ylläpidon. Esimerkiksi palvelimen laitteistosta ei tarvitse huolehtia ollenkaan, koska palvelin on virtualisoitu. Palvelimesta voidaan ottaa helposti varmuuskopioita ja lisätä tehoa nopeasti Amazonin omilla työkaluilla.

Sinser-verkkopalvelun data säilytetään MongoDB:ssä ja käyttäjätiedot Drupal-sivuston tietokannassa. Palvelimelle on toteutettu rajapinta PHP-ohjelmointikielellä ja se noudattaa REST-arkkitehtuuria.



Kuva 8. Sinser-verkkopalvelun arkkitehtuuri

Kuvassa 8 näkyy verkkopalvelun arkkitehtuuri. Palvelun asiakassovellukset kommunikoivat rajapinnan eli API:n kanssa HTTP-protokollalla. Rajapinnan palauttamat viestit ovat JSON-muodossa.

4.3 Sinser-mobiilisovelluksen rakenne

Sinser-mobiilisovellus on jaettu modulaarisiin osiin. Mobiilisovelluksen tehtäviä hoitavat eri komponentit. Muun muassa käyttöliittymälle sekä käyttäjätilin hallinnalle ovat omat komponenttinsa mobiilisovelluksessa. Tässä opinnäytetyössä käsitellään kuitenkin vain yksi komponentti ItemManager, joka käsittelee palvelimen ja mobiilisovelluksen välistä viestintää ja hallitsee muistioita. ItemManager-komponenttilla on siis suuri merkitys muun muassa välimuistin kehittämisessä.

4.3.1 ItemManager

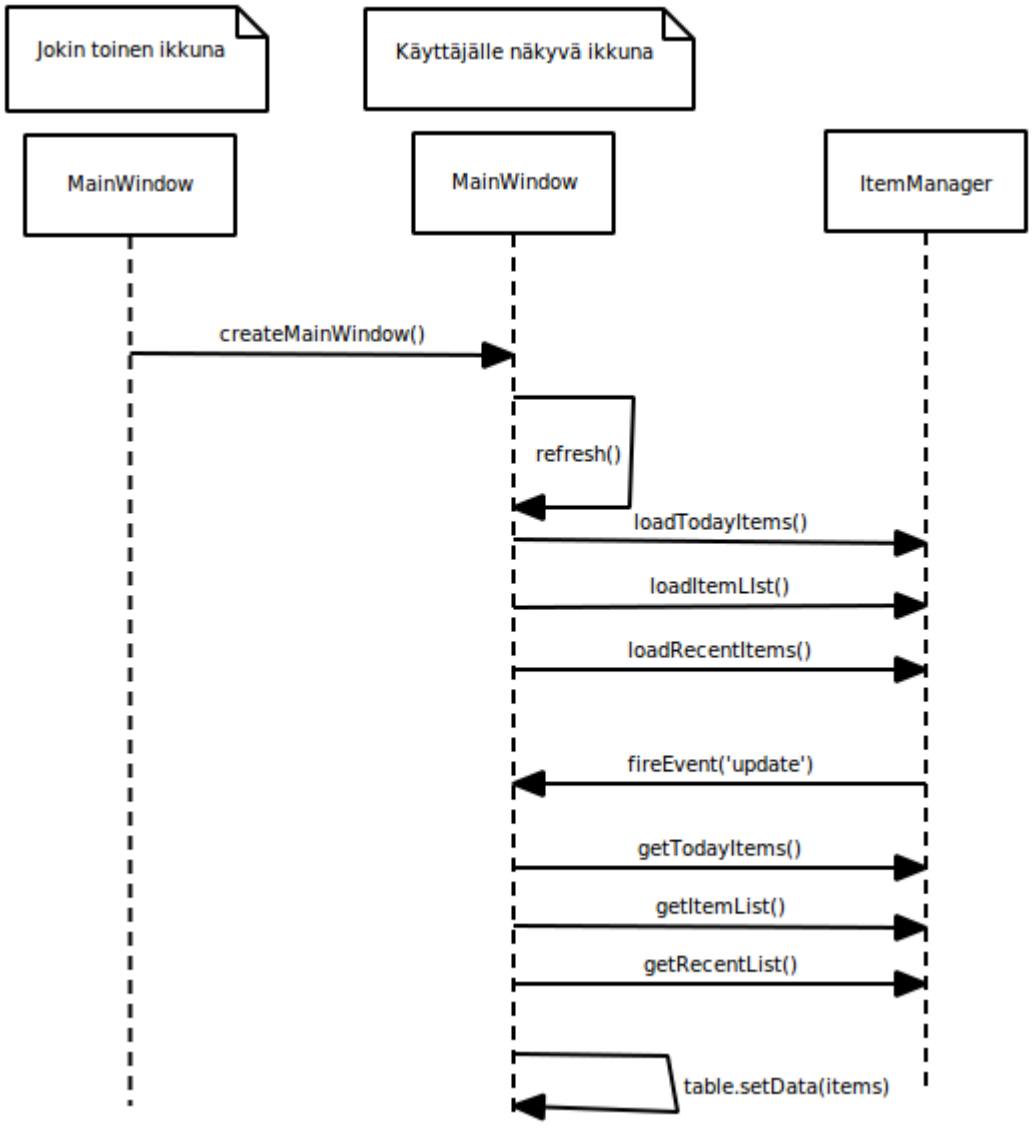
ItemManager-komponentti on yksi tärkeimmistä mobiilisovelluksen komponenteista. Se käsittelee muistioita, kommunikoi rajapinnan kanssa ja palauttaa muistiot käyttöliittymäkomponenteille. ItemManager-komponentin ansiosta käyttöliittymäkomponenttien ei tarvitse kommunikoida ollenkaan rajapinnan kanssa. ItemManager helpottaa myös ohjelmakoodin ylläpitoa. Jos rajapinta muuttuisi, ei käyttöliittymäkomponentteja tarvitsisi muuttaa. Seuraavassa listassa on lueteltuna ItemManagerin funktiot ja attribuutit.

- loadTodayItems()
- getTodayItems()
- loadItemList()
- getItemList()
- loadRecentItems()
- getRecentItems()
- parseJson()
- deleteItem()
- setToday()
- setComplete()
- createNewItem()
- updateItem()
- itemlist[]
- todaylist[]
- recentlist[]

Funktiot loadTodayItems, loadItemList ja loadRecentItems aloittavat muistioiden hakemisen palvelimelta. Funktio loadTodayItems hakee kaikki Tänään-listan muistiot, loadRecentItems hakee Viimeisimmät-listan muistiot ja loadItemList-funktio hakee muistiolistan. Palvelin palauttaa muistiot JSON-muodossa. JSON-muotoiset muistiot parsitaan JavaScript-objekteiksi parseJSON-funktion avulla, jonka jälkeen haetut muistiot tallennetaan väliaikaisesti itemlist-, todaylist- ja recentlist-taulukoihin. Kun haku on valmis, ItemManager laukaisee tapahtuman eli eventin hakuja kutsuneelle käyttöliittymäkomponentille. Tällöin käyttöliittymäkomponentti tietää haun olevan valmis. Käyttöliittymäkomponentti voi tämän jälkeen pyytää muistiot getItemList-, getTodayItems-, getRecentItems-funktioilla.

4.3.2 Muistioiden lataaminen

Tässä kappaleessa kerrotaan, kuinka muistiot ladataan palvelimelta. Esimerkkinä käytetään muistioiden hakemista, kun käyttäjä avaa uuden ikkunan mobiilisovelluksessa (Kuva 9).



Kuva 9. Muistioiden hakeminen uuden ikkunan avautuessa

Kuvassa 9 näkyy sekvenssikaavion avulla kuvattu muistioiden hakeminen. MainWindow-objekti avaa createMainWindow-funktiolla uuden MainWindow-objektin. MainWindow on käyttäjälle näkyvä ikkuna, jossa muistioita näytetään. MainWindow kutsuu omaa refresh-funktiota ikkunan avautuessa. Refresh-funk-

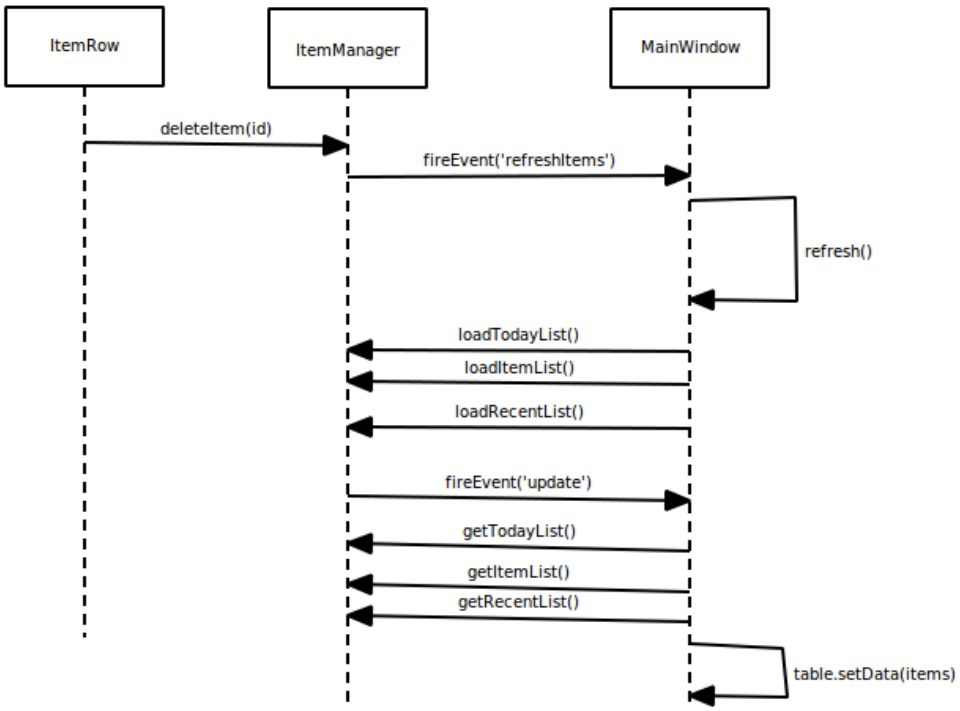
tio aloittaa muistioiden hakemisen. Aluksi MainWindow kutsuu ItemManager-objektin load-funktioita riippuen siitä, minkä listan muistioita halutaan hakea. Esimerkiksi loadTodayItems-funktio aloittaisi Tänään-listan muistioiden hakemisen. Tämän jälkeen MainWindow-objekti jää odottamaan.

ItemManager-komponentti tekee haun ja kommunikoinnin rajapinnan välillä. Kun haku on valmis, laukaisee ItemManager MainWindow-objektin update-tapahtuman. Tämä tehdään fireEvent-funktiolla, jolle annetaan parametriksi laukaistavan tapahtuman nimi.

Kun MainWindow saa update-tapahtuman, se kutsuu ItemManager-komponentin get-funktioita sen mukaan, mitä load-funktiota MainWindow on alunperin kutsunut. MainWindow tallentaa get-funktiosta paluuarvona saadut muistioit väliaikaiseen taulukkoon, muokkaa muistiot esitettävään muotoon ja kutsuu lopulta setData-funktiota. SetData-funktio näyttää muistiot Table-käyttöliittymäkomponentissa. Table-komponentti on käyttöliittymäkomponentti muiden käyttöliittymäkomponenttien esittämiseen allekkain mobiililaitteella. Tämän jälkeen käyttäjä näkee muistiot ja voi tehdä haluamiaan toimintoja.

4.3.3 Muistioiden muokkaaminen

Muistioiden muokkaamisessa tarvitaan myös muistioiden hakemista palvelimelta. Muokkaustoimintoja ovat muistion poistaminen, päivittäminen, tehtävän asettaminen tehdyksi ja muistion asettaminen Tänään-listaan. Toiminnot ovat toteutettu logiikaltaan samanlaisiksi, joten esimerkkinä tässä luvussa käsitellään vain muistion poistaminen (Kuva 10).



Kuva 10. Muistion poistamisen sekvenssikaavio

Kuvassa 10 näkyy muistion poistaminen. ItemRow on MainWindow-objektiin kuuluva objekti. ItemRow on tarkoitettu muistion esittämiseen käyttäjälle Table-käyttöliittymäkomponentissa. Kun käyttäjä valitsee jonkin muistion, avautuu hänelle valikko, josta valitsemalla hän voi tehdä haluamansa toiminnon. Jos käyttäjä valitsee poistotoiminnon, kutsuu ItemRow-objekti ItemManager-komponentin deleteItem-funktiota. ItemManager tekee tarvittavan kommunikoinnin rajapinnan kanssa, jonka jälkeen ItemManager laukaisee fireEvent-funktion avulla sen MainWindow-objektin refreshItems-tapahtuman, johon poistofunktiota kutsunut ItemRow kuuluu. RefreshItems-tapahtuma taas saa aikaan refresh-funktiokutsun MainWindow-objektissa. Tämän jälkeen muistiot päivittyvät MainWindow-objektin samalla tavalla kuin luvussa 4.3.2 on kerrottu.

5 Välimuistitekniikat

Tässä luvussa käsitellään välimuistin teoriaa ja tekniikoita. Luvussa käsitellään myös, kuinka välimuistissa olevan datan oikeellisuus voidaan tarkistaa. Esimerkkinä tarkistuksessa käytetään HTTP-protokollaa. Luvussa on myös tutkittu Titanium Mobilen rajapinnan keinoja toteuttaa välimuistikomponentti. Tätä tutki-

musta tarvitaan, jotta Sinser-mobiilisovellukseen pystytään suunnittelemaan välimuisti.

5.1 Yleisesti

Tietotekniikassa on yleisesti ollut ongelmana erilaisten sovelluksien hitaus. Dataa täytyy hakea esimerkiksi verkon kautta hitaita yhteyksiä pitkin, jolloin käyttäjä joutuu usein odottamaan, kunnes haku on tehty, data muokattu käyttäjälle esitettävään muotoon ja näytetty käyttäjälle. Datan lukeminen voi jäädä pullonkaulaksi monessa sovelluksessa. Esimerkiksi tietokoneen keskusmuistista tiedon lataaminen voi olla niin nopeaa, että käyttäjä ei huomaa hidastumista tiedonhakuvaiheessa. Kovalevyltä lukeminen on taas huomattavasti hitaampaa. Hitailla yhteyksillä verkosta datan hakeminen voi olla hidasta ja vaikuttaa merkittävästi jonkin sovelluksen käytettävyyteen.

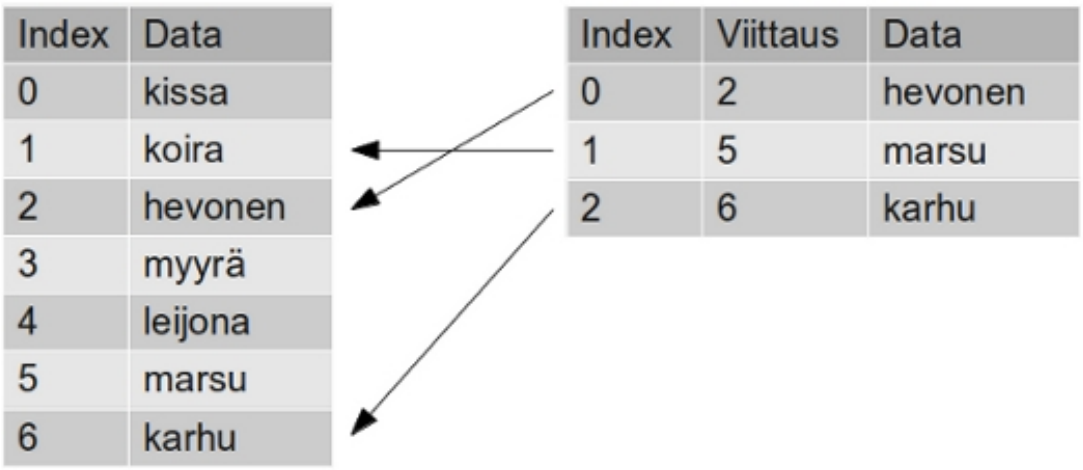
Kuvassa 11 näkyy Android-mobiilikäyttöjärjestelmään liittyvän Android marketin latausnäkymä. Tiedot haetaan verkosta, jolloin sovelluksen ensimmäisellä käynnistyskerralla lataamiseen voi mennä kymmeniä sekunteja. Seuraavilla kerroilla latausaika on huomattavasti lyhyempi. Tämä johtuu siitä, että tietoja on tallennettu välimuistiin.

Kuva 11. Android marketin lataaminen

Välimuisti eli cache (Kuva 12) on käyttäjälle näkymätön komponentti, joka varastoi dataa mahdollisten tulevien hakujen varalle (Wikipedia 2011). Android marketin tapauksessa on käytetty välimuistia, joka mahdollistaa datan nopeamman käsittelyn ja esittämisen käyttäjälle. Välimuistin tehtävänä on siis nopeuttaa sovelluksen toimintaa. Välimuistiin tallennettu data voi olla aikaisemmin ja valmiiksi käsiteltyä tietoa. Esimerkiksi tietokone voi suorittaa jotakin laskentaa ja tallentaa siitä saatua dataa välimuistiin valmiiksi. Kun dataa tarvitaan, se voidaan hakea välimuistista eikä prosessoria kuormittavaa laskentaa tarvitse suorittaa uudelleen. Välimuistiin tallennettu data voi olla myös kopio jostakin toisesta datasta. Esimerkiksi internetselain yleensä tallentaa lataamansa verkkosivut selaimen välimuistiin. Selain siis tallentaa kopion haetusta internetsivusta. Kun käyttäjä siirtyy myöhemmin uudelleen samalle sivulle, vertailee selain pyydettyä verkkosivun osoitetta välimuistista löytyviin. Jos sivu löytyy jo selaimen välimuistista, se ladataan käyttäjälle sieltä. Tällöin hitailla yhteyksillä verkkosivun

lataaminen on nopeampaa. Välimuistiin tallennetaan yleensä data, jota tarvitaan ja johon viitataan usein.

Välimuisti pyritään usein pitämään pienikokoisena. Pienestä välimuistista on nopeampi hakea dataa kuin isosta tietokannasta.



Kuva 12. Esimerkki välimuistista

Kuvassa 12 näkyy yksinkertainen esimerkki välimuistista. Vasemmanpuoleinen taulu on datan säilö, kuten kovalevy tai tietokannan taulu. Oikeanpuoleinen on taas välimuisti. Välimuistista on nopeampi hakea dataa kuin varsinaisesta datan säilöstä, koska välimuistista haettaessa täytyy käydä vähemmän dataa läpi, kunnes tarvittu data löytyy. Esimerkiksi haettaessa dataa ”marsu” välimuistista, tarvitaan käydä vain kaksi riviä läpi datan löytymiseksi, jos käytetään algoritmia, joka käy rivit yksi kerrallaan läpi ylhäältä alaspäin. Suoraan datavarastosta haettaessa läpikäytäviä rivejä olisi kuusi. Jos yhden rivin lukemiseksi kuluisi kuvitteellisesti aikaa kaksi sekuntia, kuluisi datavarastosta haettaessa aikaa 12 sekuntia. Välimuistista haettaessa aikaa kuluisi tässä tapauksessa vain 4 sekuntia.

Jos käyttäjä tekisi yllä olevassa esimerkissä muutoksia dataan, joka on haettu välimuistista, viittauksella saataisiin selville alkuperäinen data varastosta. Tällöin pystytään kirjoittamaan datan muutokset oikeaan indeksiin datavarastossa.

5.2 Välimuistin tarkistaminen

Kuinka voidaan tarkistaa, että välimuistista haettu tieto on oikeaa tietoa, eikä esimerkiksi vanhentunutta tai päivittämätöntä? On olemassa monia keinoja tarkistaa välimuistin datan oikeellisuus, mutta käsittelen niistä vain muutaman ja käytän esimerkkinä HTTP-protokollan tapaa.

HTTP on sovellustason protokolla, joka on tarkoitettu tiedonsiirtoon verkossa (RFC-2616, 2011). Muun muassa webpalvelimet ja internetselaimet käyttävät HTTP-protokollaa kommunikoinnissa keskenään. HTTP-protokolla määrittelee kolme tapaa välimuistin datan oikeellisuuteen (RFC 2616, 2011).

1. Asiakasohjelma tarkistaa palvelimelta ennen datan palauttamista, että se on validia.
2. Välimuistikomponentti tarkistaa, että palautettava data on tarpeeksi tuoretta.
3. Palvelin ilmoittaa, että haettava data ei ole muuttunut.

Kohdassa 1 datan validointi tarkoittaa sitä, että asiakasohjelma tarkistaa ensin palvelimelta, onko välimuistissa oleva tieto vielä käytettävissä. Palvelin tarkistaa oman datan. Jos data ei ole muuttunut, palvelin palauttaa sopivan statuskoodin asiakkaalle, jolloin asiakassovellus saa tiedon, että omassa välimuistissa oleva data ei ole muuttunut ja sitä voidaan käyttää. Jos data taas ei ole validia, niin palvelin palauttaa asiakkaalle pyydetyn datan.

Kohdassa 2 asiakassovellus tarkistaa datan tuoreuden ennen palvelimelta pyytämistä. Palvelin on lähettänyt aikaisemmin asiakkaalle aikamääreen, jonka perusteella asiakas tarkistaa datan tuoreuden.

Kohdassa 3 palvelin palauttaa suoraan asiakkaalle HTTP-protokollan mukaisen statuskoodin, joka kertoo asiakkaalle datan pysyneen ennallaan. Näitä statuskoodeja ovat muun muassa 3-alkuiset koodit, 4- ja 5-alkuiset virhekoodit.

5.3 Titanium Mobile

Tässä luvussa kerrotaan Titanium Mobilen mahdollisista tavoista toteuttaa välimuistiin tallentaminen. Aloitin tutkimalla Titanium Mobilen rajapintojen dokumentaatiota ja löysin tiedon tallentamiseen kolme moduulia, joiden avulla voidaan

toteuttaa välimuistikomponentti. Valmista välimuistikomponenttia Titanium Mobileen ei ole tehtynä.

5.3.1 Properties

Properties on Titanium Mobilen moduuli, jonka avulla voidaan tallentaa avaimen perusteella haettavaa dataa. Esimerkiksi avain voi olla taustakuva ja data taustakuvan polku tiedostojärjestelmässä. Alla on sama esimerkki ohjelmakoodina.

```
Titanium.App.Properties.setString(”taustakuva”,”/kuvat/taustakuva.png”);
```

Yllä olevassa ohjelmakoodissa Titanium.App tarkoittaa nimiavaruutta, josta Properties-moduuli löytyy. Tallentaminen tehdään setString-funktiolla. Tämä tallentaa yhden tekstimuotoisen tiedon ”/kuvat/taustakuva.png”, jonka voi hakea avaimella ”taustakuva”.

5.3.2 Database

Database on Titanium Mobilen tietokantamoduuli. Sen avulla voi luoda ja käsitellä tietokantoja mobiilisovelluksessa. Database-moduuli löytyy nimiavaruudesta Titanium.Database. Database-moduuli jakautuu DB- sekä ResultSet-objekteihin. DB-objektin avulla käsitellään tietokantaa ja ResultSet-objektilla käsitellään tietokantahakujen tuloksia. Seuraavassa ohjelmakoodissa on esimerkki tietokannan käsittelystä.

```
var db = Titanium.Database.open('asiakkaat.db');
var rows = db.execute('SELECT * FROM ASIAKKAAT WHERE asiakasID = 1234');
```

Esimerkin ensimmäisellä rivillä avataan tietokanta asiakkaat.db ja tallennetaan open-funktiosta saatu objekti db-muuttujaan. Toisella rivillä db-objektin execute-funktiolla ajetaan tietokantakysely, joka hakee kaikki tiedot ASIAKKAAT-taulusta ehtona, että asiakasID:n arvo on 1234. Tuloksena saadut rivit tallennetaan käsittelyä varten rows-muuttujaan. Funktio palauttaa ResultSet-objektin, jonka funktioilla voidaan käsitellä hakutulokset yksi kerrallaan.

Database käyttää SQLite-tietokantaa. SQLite on tietokanta, joka ei tarvitse erillistä palvelinta vaan se tallentaa tietoja suoraan muistiin. SQLite on suunniteltu mahdollisimman kevyeksi ja kuluttamaan todella vähän muistia. Se myös toimii nopeasti vähän muistia sisältävissä laitteissa (SQLite, 2012). Keveytensä vuoksi se soveltuu erinomaisesti älypuhelimiin ja muihin vastaaviin mobiililaitteisiin.

5.3.3 Filesystem

Filesystem-moduulia käytetään mobiililaitteen tiedostojen ja hakemistojen lukemiseen ja kirjoittamiseen. Filesystem-moduuli löytyy Titanium.Filesystem-nimiavaruudesta ja se jakautuu kahteen File- ja FileStream-objektiin. Seuraavassa ohjelmakoodiesimerkissä luetaan tiedosto ja tulostetaan tiedoston sisältö.

```
var tiedosto = Titanium.Filesystem.getFile('tiedosto.txt');
if(tiedosto.exists()){
   var sisalto = tiedosto.read();
   Titanium.API.info('tiedosto: ', sisalto.text);
}
```

Esimerkissä rivillä yksi ja kaksi avataan tiedosto getFile-funktiolla. Saatu File-objekti tallennetaan tiedosto-muuttujaan. Funktio ottaa parametrikseen käsiteltävän tiedoston polun. Tässä tapauksessa tiedosto on tiedosto.txt. Seuraavalla rivillä tarkistetaan, että tiedosto on olemassa. Tämän voi tehdä exists-funktiolla ja if-lauseella. Funktio palauttaa true- tai false-arvon riippuen tiedoston olemassaolosta. Jos tiedosto on olemassa, suoritetaan kaksi muuta riviä. Tiedosto luetaan read-funktiolla. Luettu sisältö tallennetaan sisalto-muuttujaan. Seuraavalla rivillä sisältö tulostetaan konsoliin. Sisalto-muuttuja on JavaScript-objekti. Luettu sisältö saadaa objektin ominaisuudesta sisalto.text. Konsoliin tulostaminen onnistuu info-funktiolla. Funktio löytyy Titanium.API-nimiavaruudesta.

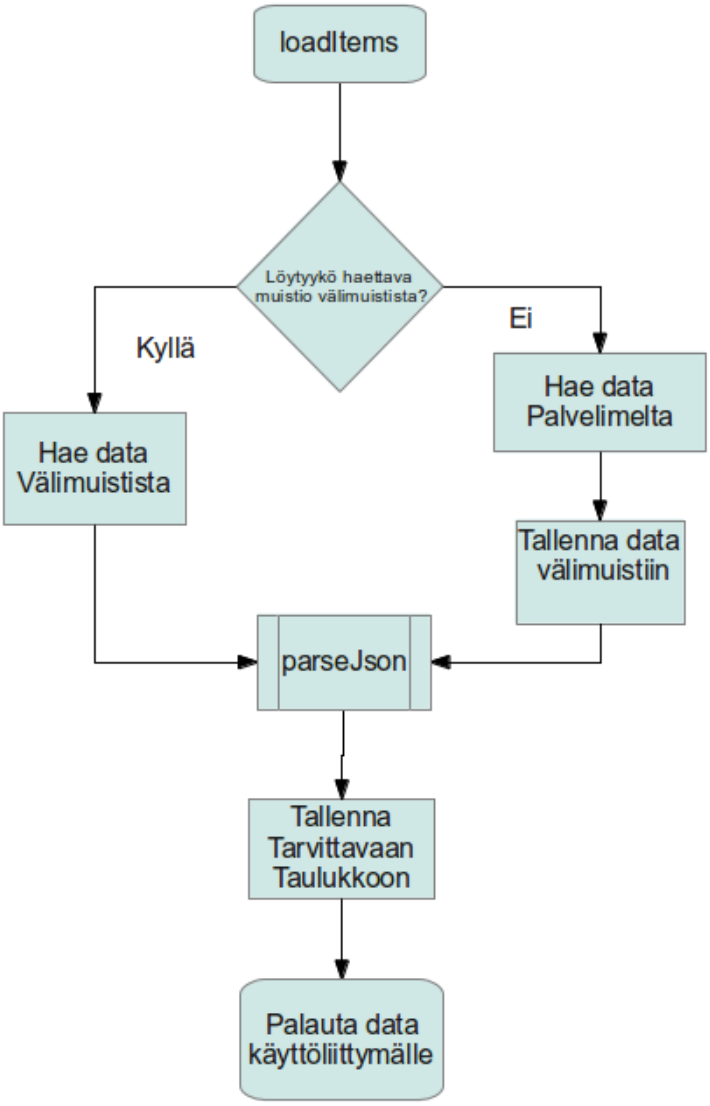
6 Välimuistin toteuttaminen Sinseriin

Tässä luvussa kerrotaan, kuinka välimuisti on suunniteltu ja toteutettu Sinser-mobiilisovellukseen. Luvussa 6.1 kerrotaan suunnitelmasta ja käydään läpi eri toteutusvaihtoehtoja välimuistille. Luvussa 6.2 käsitellään toteutuksen kulku ja lopussa syntynyt päätelmä ja päätös välimuistista. Luvussa 6.3 kerrotaan lopputulokset.

6.1 Suunnitelma

Tutustuttuani eri tapoihin tallentaa tietoa puhelimeen Titanium Mobilessa, aloitin suunnittelemaan välimuistia mobiilisovellukseen. Etsin internetistä tietoa eri toteutustavoille. Suunnittelussa käytin hyväksi luvussa 4 tehtyjä kuvauksia ja kuvioita mobiilisovelluksen rakenteesta. Suunnittelin aluksi, kuinka välimuisti toimii

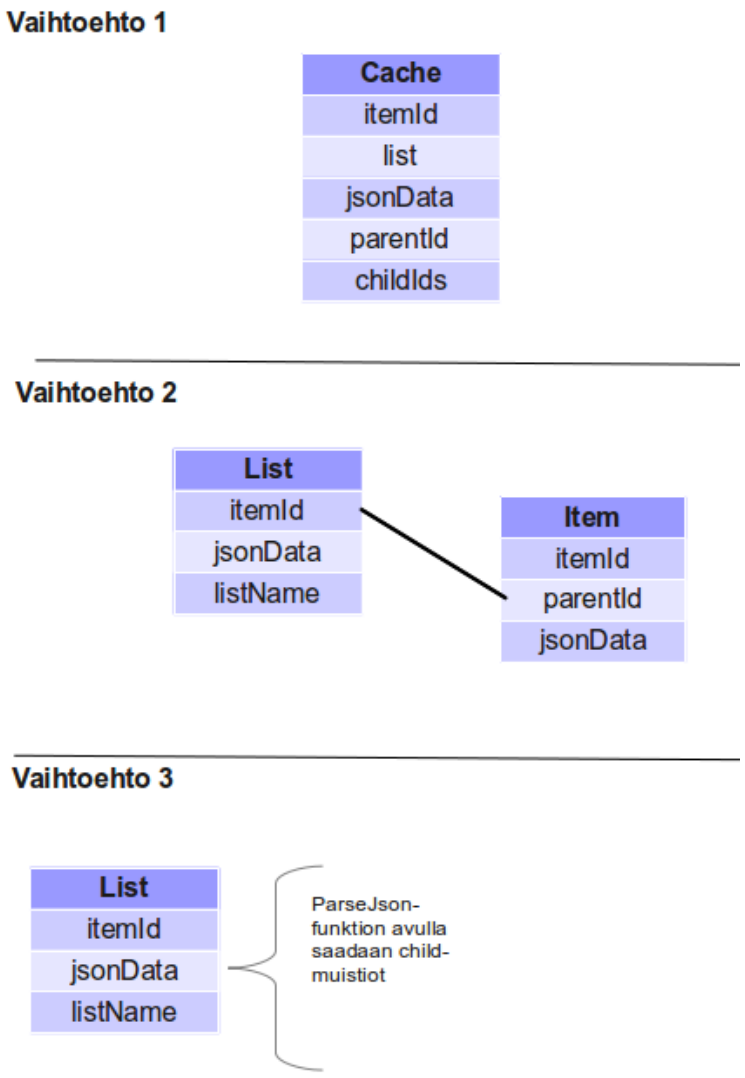
yleisellä tasolla. Tämän jälkeen tarkensin suunnitelmaani yksityiskohtaisemmaksi ja eri toiminnoille, kuten esimerkiksi muistioiden hakemiselle (Kuva 13).



Kuva 13. Välimuistista hakeminen

Kuvassa 13 näkyy kaavio, jossa välimuisti on liitetty muistioiden hakemiseen. Välimuistista hakeminen on liitetty loadItems-funktioihin. Kun jotain muistiolistaa, kuten Tänään-listaa haetaan, tarkistetaan ensin, löytyykö haettava muistio tai lista välimuistista. Jos ei löydy, niin se haetaan palvelimelta ja tallennetaan välimuistiin. Data muutetaan JavaScript-objekteiksi parseJson-funktion avulla, tallennetaan taulukkoon muistiin ja palautetaan käyttöliittymälle tai muistioita tarvitsevalle komponentille. Seuraavalla kerralla samaa listaa tai muistiota haettaessa se löytyy välimuistista. Tällöin verkkoyhteyttä ei tarvita.

Tämän jälkeen päätin, mitä tekniikkaa käytän välimuistissa ja kuinka toteutan suunnitelman. Vertailin eri tekniikoita Titanium Mobilessa, joista olen kertonut luvussa 5. Päätin valita tekniikaksi mobiililaitteen SQLite-tietokantaa käyttävän Ti.Database-moduulin, koska sen avulla pystyy hallitsemaan dataa paremmin tietokantataulujen avulla kuin esimerkiksi kirjoittamalla datan tiedostoihin laitteen muistiin. SQLite on myös nopea ja SQL-kieltä käyttäen datan hakeminen pitäisi olla helppoa. Suunnittelin taulujen rakenteen välimuistille (Kuva 14).



Kuva 14. Välimuistin taulut

Kuvassa 14 näkyvät tekemäni suunnitelmat välimuistin rakenteelle. Suunnittelin kolme eri vaihtoehtoa välimuistin taulujen rakenteeksi. Valitsin vaihtoehdon 3.

Ensimmäisessä vaihtoehdossa tehtäisiin yksi taulu, johon tallennettaisiin muistion id, listan nimi, JSON-muotoinen data sekä ylemmän muistion id ja alempien muistioiden id:t. Tämä vaihtoehto on mielestäni kuitenkin huono. Tauluun tallennetaan turhaa tietoa, kuten esimerkiksi childids. Childids on kenttä muistion alla oleville muistioiden id:ille. Nämä kuitenkin saa luettua jsonData-kentästä, koska siihen on tallennettuna kaikki muistion tieto. Välimuistista poistaminen ja välimuistin muokkaaminen vaatisi myös monimutkaisia SQL-kyselyitä. Esimerkiksi yhden muistion muokkaaminen vaatisi, että myös kaikki sen alla olevat muistiot päivitettäisiin. Päivitykset olisi tehtävä myös kaikista eri listoista.

Vaihtoehdossa 2 tekisin kaksi eri taulua. List-tauluun tallennettaisiin lista ja item-tauluun muistio. Totesin tämänkin vaihtoehdon huonoksi, koska item-taulu on turha. List-taulun jsonData sisältää kaikki muistion tiedot, joten listan muistiot saadaan haettua siitä parseJson-funktiota käyttäen. Tällöin item-taulu on turha.

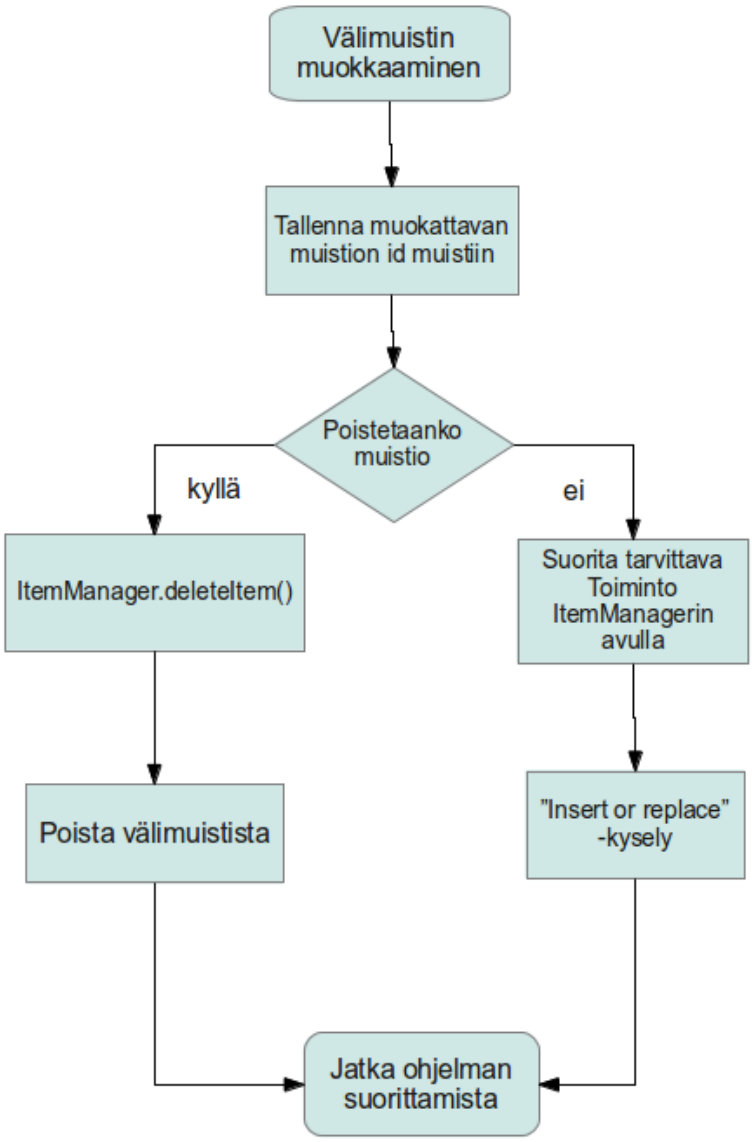
Valitsin parhaimmaksi vaihtoehdoksi vaihtoehdon 3. Tässä ratkaisussa vaihtoehdosta 2 on poistettu item-taulu. List-tauluun tallennetaan lista ja siihen kuuluvat muistiot. Muistiot saadaan jsonData-kentästä parsimalla. Välimuistin muokkaaminen (Kuva 15) ja välimuistista poistaminen ovat myös tässä ratkaisussa helppo tehdä. Muokkaamisessa vain korvataan jsonData-kenttä uudella datalla. Poistamisessa voidaan suoraan poistaa rivi tietokannasta, koska taulut eivät ole riippuvaisia mistään muusta taulusta. Päätin käyttää vaihtoehtoa 3.

Tietokannan saa luotua mobiilisovelluksessa SQL-lauseella.

```
var db = Ti.Database.open('itemlist');
db.execute("CREATE TABLE IF NOT EXISTS 'itemlist' ('itemId' VARCHAR,
'jsonData' TEXT, 'listName' VARCHAR)");
db.close();
```

Ensimmäisellä rivillä avataan tai luodaan tietokanta, jos sitä ei ole vielä luotu. Tietokannan nimi annetaan parametrina open-funktiolle. Execute-funktio ajaa SQL-tietokantakyselyn. ”CREATE TABLE IF NOT EXIST 'itemlist'” -kysely luo uuden itemlist-nimisen taulun, jos sitä ei ole aikaisemmin luotu. Suluissa olevissa käskyssä määritellään taulun kentät. ItemId-kenttä on VARCHAR-tyypinen kenttä. VARCHAR tarkoittaa merkinjonoa. Valitsin merkkijonon id-kentälle, koska muistiot ovat tallennettuina MongoDB-tietokantaan palvelimella ja MongoDB-

tietokannan id-kenttä on 12-bittinen binääriarvo, kuten esimerkiksi 47cc67093475061e3d95369d. Merkkijono on mielestäni hyvä tämän tallentamiseen, koska hakujen tekeminen on tällöin helppoa. JsonData kenttä on TEXT-tyyppinen. Valitsin TEXT-tyypin, koska data voi olla pitkä merkkijono riippuen muistioiden määrästä. TEXT-tyyppiseen kenttään voi tallentaa yli miljardi bittiä ja sitä voi tarvittaessa kasvattaa(SQLite 2012). ListName-kenttään tallennetaan listan nimi. Valitsin sille myös VARCHAR-tyypin, koska se on vain yksi sana.



Kuva 15. Välimuistin muokkaaminen

Kuvassa 15 näkyy, kuinka välimuistia käsitellään muistioiden muokkaus- tai poistotoiminnoissa. Muokattavan muistion id tallennetaan muistiin. Jos muistio halutaan poistaa, tehdään se muistiota hallitsevan ItemManagerin avulla. Item-

Manager ottaa yhteyden palvelimeen ja tekee tarvittavan viestinnän rajapinnan kanssa muistion poistamiseksi. Tämän jälkeen muistion tallennettua id:tä käyttäen muistio voidaan poistaa välimuistista DELETE-kyselyn avulla.

Jos muistiota halutaan muokata, tehdään tarvittava muokkaus ensin ItemManagerin avulla samalla tavalla kuin muistion poistamisessakin. Tämä jälkeen välimuistiin tehdään INSERT- tai REPLACE-kysely. Tämä tarkoittaa sitä, että vanha muistio korvataan uudella muistiolla REPLACE-kyselyn avulla. Jos muistiota ei ole aikaisemmin tallennettuna välimuistiin, niin silloin käytetään INSERT-kyselyä, joka tallentaa muistion uutena rivinä tietokantaan.

6.2 Toteutuksen kulku

Aloitin toteutuksen muokkaamalla muistiota hallitsevan ItemManagerin toteutusta. ItemManager hakee muistioita ja käsittelee niitä. Se myös pitää yhteyttä palvelimeen. Se on tärkeässä roolissa mobiilisovelluksessa, joten päätin aluksi muokata ja optimoida sen koodia. Tein yhden suuren muutoksen ItemManageriin, josta oli paljon hyötyä koko mobiilisovelluksen toiminnalle ja nopeudelle. Muutin muistioiden hakemisen tapahtumapohjaisesta toteutuksesta callback-pohjaiseksi (Kuva 16).

Kuva 16. ItemManagerin uusi toteutus

Kuvassa 16 näkyy uusi ItemManagerin toteutus. Load-funktioille on nyt annettava parametrina callback-funktio. Callback-funktio on parametrina annettava funktio, jota kutsutaan, kun funktio on suorittanut toimintonsa. Callback-funktiota kutsutaan, kun ItemManager on saanut haun palvelimelta tehtyä. Callback-funktio kutsuu ready-funktiota. Ready-funktio tarkistaa onko kaikki haut tehty. Kun kaikki haut on tehty, kutsutaan setData-funktiota, joka näyttää muistiot table-komponentissa. Callback-funktiot mahdollistavat sen, että haut voidaan tehdä rinnakkain, eikä event-tapahtumia tarvitse odottaa. Tämä nopeutti mobiilisovelluksen toimintaa todella paljon.

Titanium Mobilesta tuli uusi versio 1.8.0.1 joulukuussa 2011. Tämä aiheutti epäyhteensopivuuksia mobiilisovelluksessa, joten päivitin ennen välimuistin toteuttamista mobiilisovelluksen ohjelmakoodin yhteensopivaksi uusimman version kanssa. Korjaamista täytyi tehdä muun muassa sisäänkirjautumisessa. Mobiilisovellus käyttää Ti.Network.HttpClient-moduulia viestinnässä palvelimen kanssa. Moduulilla voi tehdä HTTP-pyyntöjä verkkoon. Palvelin käyttää evästeitä si-

säänkirjautuneiden käyttäjien tunnistamisessa. Ennen uusimman Titanium Mobilen päivitystä nämä evästeet tuli asettaa itse HttpClient-objektille, mutta uudessa versiossa tämä osasi asettaa ne palvelimen pyynnöstä ilman erillistä ohjelmakoodia. Muutamia pieniä käyttöliittymäkomponentteihin liittyviä päivityksiä täytyi myös tehdä.

Suurimman parannuksen päivitys aiheutti table-komponentissa. Table-komponenttia käytetään näyttämään muistiot allekkain ikkunassa. Tämä komponentti sai suuren nopeusparannuksen verrattuna edelliseen versioon. Muistioiden hakeminen palvelimelta on nopeaa, mutta niiden näyttäminen table-komponentissa vei paljon aikaa. Uuden version myötä nopeus parantui huomattavasti.

ItemManagerin muuttaminen callback-funktioilla toimivaksi ja Titanium Mobilen uusin versio paransivat ohjelman nopeutta ja toimintaa todella paljon. Päivityksen jälkeen ohjelma tuntui käytettävyydeltään hyvältä, koska käyttäjä ei joudu odottamaan ohjelman eri toimintojen suorituksia.

Aloitin tämän jälkeen välimuistin toteuttamisen. Tarkoituksenani oli tehdä suunnitelmieni pohjalta toimiva runko välimuistille ja saada perustoiminnot toimiviksi nopeasti. Toteutin funktioita uuteen cache-moduuliin. Funktiot on tehty kuvan 13 perusteella. Toteutin ensimmäiseen toimivaan välimuistitoteutukseen funktiot load, saveList ja loadList. Load-funktio alustaa tietokannan ja luo sen, jos sitä ei ole aikaisemmin tehty. SaveList-funktion avulla tallennetaan lista ja listan JSON-muotoinen data Android-laitteen tietokantaan. Funktio noudattaa kuvan 15 logiikkaa. Aluksi funktio tarkastaa, onko lista jo aikaisemmin tallennettu. Jos sitä ei ole tallennettu, lisätään uusi rivi tietokannan tauluun, johon lista tallennetaan. Jos lista on taas aikaisemmin tallennettu, kirjoitetaan entisen datan päälle. LoadList-funktiolla ladataan lista tietokannasta. Haku tapahtuu listan id:n ja nimen perusteella. Funktio palauttaa JSON-muotoisen datan. Jos listaa ei löydy tietokannasta, palauttaa funktio null-arvon. Tällöin loadList-funktiota kutsunut funktio voi päätellä, että data on haettava palvelimelta.

```
Sinser.cache.load();
var cacheJson = Sinser.cache.loadList(tempId,'today');
if(cacheJson != undefined){
        Sinser.ItemManager.parseJson(todaylist,cacheJson);
        callback(todaylist);
        return;
}
else{
var httpclient = Ti.Network.createHTTPClient();
…
```

Ylläolevassa koodiesimerkissä näytetään yksinkertaistettuna, kuinka muistio haetaan. Ensimmäisellä rivillä alustetaan laitteen välimuisti ja luodaan tietokanta, jos sitä ei ole vielä luotu. Seuraavaksi ladataan Today-lista. Muuttuja tempId sisältää haettavan listan id:n. Paluuarvo tallennetaan cacheJson-muuttujaan. If-lauseella tarkistetaan löytyikö haettavaa listaa välimuistista. Jos lista löytyi, parsitaan JSON-muotoisesta datasta todaylist-muuttujaan listan muistiot. Tämän jälkeen kutsutaan callback-funktiota, jolle annetaan muistiot parametriksi. Jos listaa ei olisi löytynyt, oltaisiin lista haettu palvelimelta.

Kun olin tehnyt yksinkertaisen ja toimivan version välimuistista, testasin sovellusta älypuhelimella. Havaitsin, että välimuisti toimii, koska verkosta ei haettu tietoa, kun lista oli aikaisemmin ladattu. En kuitenkaan huomannut paljoa eroa sovelluksen nopeudessa. Tuntui, että muistiot ja listat latautuivat yhtä nopeasti niin palvelimelta kuin älypuhelimen välimuististakin. Testasin muistioiden latautumista 0,5 megabitin mobiili-internetliittymällä, joten verkkoyhteys ei ollut testauksessa kovin nopea.

Pohdin tämän jälkeen havainnot tehtyäni mobiilisovellusta. Onko välimuisti mobiilisovelluksen kannalta tärkeä, jos yrityksen tavoitteena on saada ensimmäinen julkaistava versio käyttäjille mahdollisimman pian? Mobiilisovelluksessa on vielä tärkeitä toimintoja suunnittelematta ja toteuttamatta. Esimerkiksi muistioiden jakaminen muiden käyttäjien kanssa on vielä kesken. On tärkeää saada käyttäjälle keskeiset toiminnot ennen julkaisua tehtyä. Testieni mukaan välimuisti ei tuo paljoa lisänopeutta sovellukselle. Onko välimuistia ensimmäiseen mobiilisovelluksen versioon järkevä toteuttaa, jos siitä ei tule käyttäjälle paljoa hyötyä ja se ei ole käyttäjälle näkyvä ominaisuus? Mielestäni pitäisi keskittyä toteuttamaan muita toimintoja ensin valmiiksi ja toteuttaa välimuisti myöhempiin mobiilisovelluksen versioihin.

Kerroin päätelmistäni ja havainnoistani Mikko Mäkelälle ja hän oli samaa mieltä kanssani. Päätimme siirtää välimuistin toteutuksen jatkokehitykseen ja toteuttaa ensin tärkeämpiä toiminnallisuuksia ja ominaisuuksia mobiilisovellukseen. Sinser ei tuota vielä tuloja yritykselle. Tämän vuoksi on tärkeää saada palvelu ensin julkaistua, jonka myötä saada tuloja yritykselle.

6.3 Lopputulos

Lopputuloksena syntyi suunnitelma ja aloitettu toteutus välimuistista Sinser-mobiilisovellukseen. Tuloksena saatiin myös havaintoja välimuistin tarpeellisuudesta mobiilisovellukselle. Mobiilisovellus toimii ilman välimuistia, eikä sillä ollut testieni mukaan paljoa vaikutusta mobiilisovelluksen nopeuteen. Välimuistin avulla voidaan kuitenkin säästää siirretyn datan määrässä ja käyttää sovellusta myös osittain, kun verkkoa ei ole saatavilla.

Päätettiin, että välimuisti ei ole tarpeellinen ensimmäiselle julkaistavalle versiolle mobiilisovelluksesta. Välimuisti toteutetaan myöhemmissä mobiilisovelluksen versioissa. Välimuistin suunnitelmat ja alkutoteutus voidaan kuitenkin säilyttää jatkoa ajatellen.

7 Hälytyksien toteuttaminen Sinseriin

Tässä luvussa kerrotaan, kuinka hälytysten suunnitteleminen ja toteuttaminen Sinser-mobiilisovellukseen onnistui tässä opinnäytetyössä. Luvussa 7.1 vertaillaan eri vaihtoehtoja ja teknologioita toteuttaa hälytykset. Luvussa 7.2 kerrotaan valitusta teknologiasta ja luvussa 7.3 kerrotaan toteutuksen suunnitelmasta. 7.4 luvussa kerrotaan toteutusvaiheesta ja lopputulokset käsitellään luvussa 7.5.

7.1 Ajanhallinta ja hälytykset

Tavoitteena on saada aikaan hälytys käyttäjälle, kun mobiilisovellus vastaanottaa viestin tai joku toinen käyttäjä jakaa muistion. Hälytys on siis ilmoitus käyttäjälle. On tärkeää, että käyttäjä saa ilmoituksen heti. Olen keksinyt kolme sopivaa tapaa toteuttaa hälytys mobiilisovellukseen.

Ensimmäisessä tavassa mobiilisovellus hakee palvelimelta tiedon, onko viestejä tullut tai onko muistioita jaettu. Mobiilisovellus hakisi tiedon esimerkiksi 30 mi-

nuutin välein. Tapa on kuitenkin huono Sinser-mobiilisovellukseen. Jatkuva yhteydenpito kuluttaa puhelimen akkua ja käyttäjä saisi hälytyksen maksimissaan puolentunnin viiveellä.

Toisessa tavassa puhelin muodostaisi TCP-yhteyden sokettien avulla palvelimeen. Yhteyttä pidettäisiin jatkuvasti auki, jolloin mobiilisovellus saisi hälytyksen heti eikä viivettä syntyisi. Tämä aiheuttaisi kuitenkin paljon lisätyötä palvelimen päässä. Palvelimeen täytyisi ohjelmoida erillinen sovellus, joka vastaanottaa yhteyksiä mobiilisovelluksista, ylläpitää yhteyksiä ja lähettää hälytyksiä. Tämä tapa saattaa myös kuluttaa puhelimen akkua.

Kolmannessa tavassa käytettäisiin push notification -teknologiaa. Push notification tarkoittaa teknologiaa, jossa palvelin pyytää asiakasohjelmaa vastaanottamaan tietoa ja lähettää tiedon sitten asiakkaalle. Esimerkkinä push notification teknologiasta ovat pikaviestimet. Viestit lähetetään asiakasohjelmille heti, kun palvelin vastaanottaa ne. Omassa Android-älypuhelimessani olen havainnut, että sähköpostin saapuessa puhelin ilmoittaa siitä minulle heti, kun sähköposti ilmestyy postilaatikkoon. Tästä olen päätellyt, että sähköpostisovellus käyttää push notification -teknologiaa. Puhelimeni akku kestää myöskin tätä, joten se ei myöskään kuluta paljoa akkua. Tämä vaikuttaisi parhaimmalta tavalta toteuttaa hälytykset Sinser-mobiilisovellukseen.

Löysin kaksi palveluntarjoajaa push notification -teknologialle. Ensimmäinen palveluntarjoaja Google tarjoaa Android Cloud to Device Messaging Frame-Work -palvelua. Palvelun avulla pystyy lähettämään lyhyitä viestejä Android-laitteille. Tutkin Titanium Mobilen tukea kyseiselle palvelulle, mutta valmista komponenttia en kuitenkaan löytänyt. Päätin aluksi hylätä kyseisen vaihtoehdon ja etsiä muita vaihtoehtoja.

Toinen palveluntarjoaja on Urban Airship. Urban Airship tarjoaa push notification -palvelua, jossa miljoona ensimmäistä viestiä ovat ilmaisia. Tämän jälkeen hintaa tulee 0,001 dollaria viestiltä. Urban Airship tarjoaa myös pakettihinnoittelua esimerkiksi 10000 kuukausittaiselle käyttäjälle hintaan 199 dollaria kuukaudessa (Urban Airship, 2011). Titanium Mobile tukee tätä palvelua maksullisessa versiossa. Urban Airship -moduuli maksaa 29,99 dollaria kehittäjältä, joten sen

käyttäminen ei maksa yritykselle paljoa. Tämä vaihtoehto vaikuttaa parhaimmalta.

Kaikissa tapauksissa tulee palvelimelle toteuttaa erillinen komponentti. Tutkin Urban Airshipin dokumentaatiota ja löysin eri ohjelmointikielellä toteutettuja kirjastoja palvelimen yhdistämiseksi palveluntarjoajan rajapinnan kanssa. Löysin dokumentaatiosta PHP-ohjelmointikielellä tehdyn kirjaston, jolla palvelimen pystyy yhdistämään Urban Airship -palvelimeen.

Ehdotin Urban Airshipiä ratkaisuksi hälytyksille. Koska Sinser ei vielä tuota yritykselle tuloja ja Urban Airship -moduuli on maksullinen, päätettiin kuitenkin tutkia lisää Googlen Android Cloud to Device Messaging FrameWork -palvelua. Käytän palvelusta tästä eteenpäin lyhennettä c2dm. Tutkin lisää c2dm-palvelua ja sen yhteensopivuutta Titanium Mobilen kanssa. Löysin lopulta yhteensopivan moduulin.

7.2 C2dm ja Titanium Mobile

Tutkin avoimen lähdekoodin yhteisöprojekteja Github-palvelusta. Github on internetsivusto, jossa käyttäjät voivat kehittää ja jakaa omia ohjelmointiprojektejaan muille. Muut käyttäjät voivat ottaa osaa projekteihin tekemällä ohjelmakoodia ja lähettämällä sen projektin omistajalle. Projektin omistaja voi lisätä halutessaan koodin projektiin. Käyttäjät voivat myös etsiä virheitä ohjelmakoodista tai kirjoittaa esimerkiksi dokumentaatiota projektista. Githubista löysin projektin nimeltä titanium-c2dm. Projektissa on tehty natiivi Android-moduuli, joka on yhdistetty Titanium Mobileen.

Aloitin tutkimalla lisää c2dm-palvelua ja mitä vaatimuksia se asettaa palvelimelle sekä mobiilisovellukselle. C2dm-palvelun käyttö on yksinkertaista.

1. Kehittäjä rekisteröi oman sovelluksensa ja Google-tilin.
2. Google hyväksyy kehittäjän c2dm-palveluun.
3. Mobiilisovellus rekisteröi itsensä c2dm-palvelimelle ja saa registration_id:n.
4. Palvelin tunnistautuu c2dm-palvelimelle ja saa avaimen, jota tarvitaan viestien lähettämisessä.

5. Mobiilisovellus lähettää oman registration_id:n palvelimelle.
6. Palvelin lähettää viestin c2dm-palvelimelle. Viesti sisältää registration_id:n, jonka avulla c2dm-palvelin osaa lähettää push notificationin oikeaan puhelimeen. Viesti sisältää myös lähetettävän datan.
7. C2dm-palvelin lähettää push notificationin puhelimeen.
8. Mobiilisovellus vastaanottaa push notificationin, tekee toimenpiteitä ja ilmoittaa käyttäjälle.

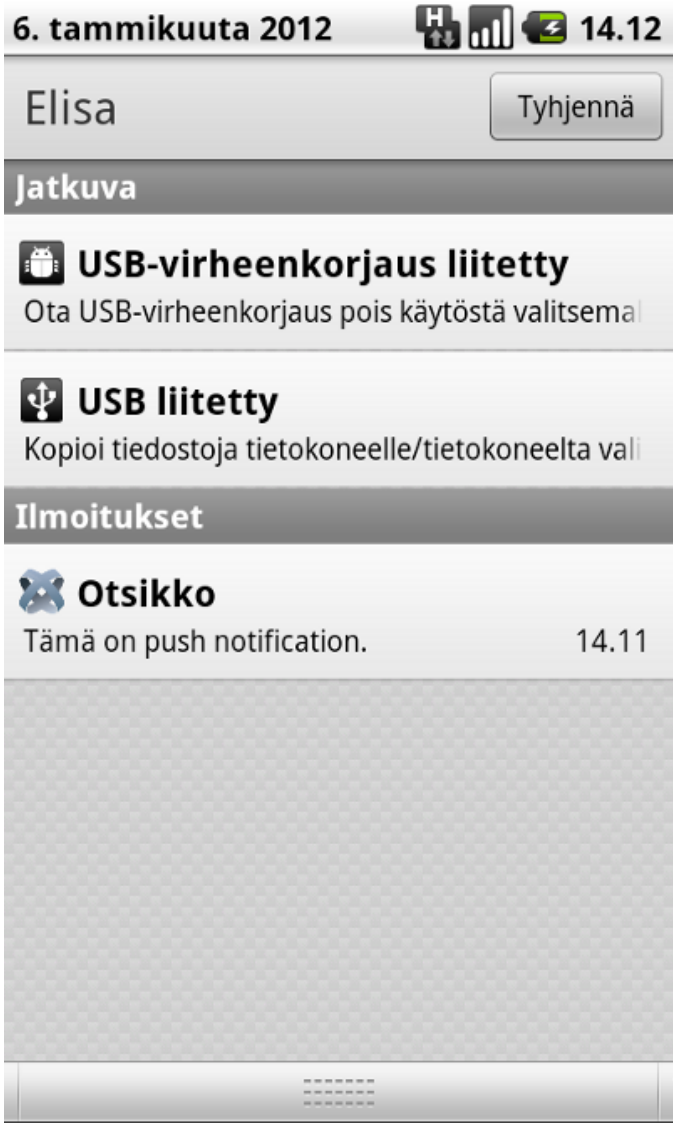
Palvelussa viestin koko on rajoitettu 1024:n tavuun ja viestien määrä 200000:n viestiin päivässä (Google 2011). Palvelu vaatii myös yhden Google-tilin, joka toimii lähettäjänä. Vaatimuksena on myös, että käyttäjällä on Google-tili kirjautuneena omassa älypuhelimessa sekä vähintään Androidin 2.2 versio.

Latasin moduulin ja päätin testata sitä tekemällä pienen ja yksinkertaisen testisovelluksen. Moduulin dokumentaatio oli huonoa ja vähäistä ja jouduin etsimään tietoa selailemalla eri keskustelupalstoja ja moduulin ohjelmakoodia. Sain lopulta pienen ohjelman valmiiksi. Testiohjelma rekisteröityy aluksi c2dm-palveluun ja jää tämän jälkeen odottamaan push notificationeita. Käytin Androidin omia työkaluja ohjelman testauksessa. Lokitulosteesta havaitsin, että rekisteröityminen oli onnistunut ja registration_id oli saatu. Seuraavaksi tein omalle testipalvelimelle yksinkertaisen scriptin, joka tunnistautuu c2dm-palveluun ja lähettää viestin. Löysin valmiin scriptin internetin keskustelupalstalta. Kokeilin lähettää viestiä, jonka jälkeen puhelimeeni tuli push notification ilmoitus (Kuva 17). Kaikki toimi hyvin.

Ongelmia syntyi kuitenkin 22. päivä joulukuuta, jolloin Appcelerator päivitti Titanium Mobilen versioon 1.8.0.1. Tämä oli suuri parannus Sinser-mobiilisovelluksen nopeuteen. Päivitys kuitenkin aiheutti epäyhteensopivuuden moduulien kanssa. Vanhat moduulit eivät toimineet uuden version kanssa. Koska päivitys oli suuri parannus mobiilisovellukselle, päätin yrittää päivittää c2dm-moduulia yhteensopivaksi uuden Titanium Mobilen kanssa.

Löysin dokumentaatiota muuttuneista funktioista ja asioista natiiveissa moduuleissa uudessa versiossa. Tutkin c2dm-moduulin lähdekoodia ja etsin sieltä dokumentaatiosta löytyviä kohtia. Tutustuin myös uuteen työkaluun nimeltä Ant. Ant on työkalu ohjelmakoodin kääntämisen helpottamiseksi. Ant-työkalua käyte-

tään lähdekoodin kääntämiseen moduuliksi. Kokeilin aluksi kääntää moduulia, mutta se tuotti epäyhteensopivuuksista johtuen monia virheilmoituksia. Dokumentaatiota apuna käyttäen ja lähdekoodia lukien, kävin jokaisen virheilmoituksen yksitellen läpi. Päivitin lähdekoodin yhteensopivaksi uuden version kanssa ja sain lopulta käännettyä moduulin. Testasin moduulin toimintaa testiohjelmassani ja totesin sen toimivaksi. Päätin myös lähettää työni Github-palveluun.

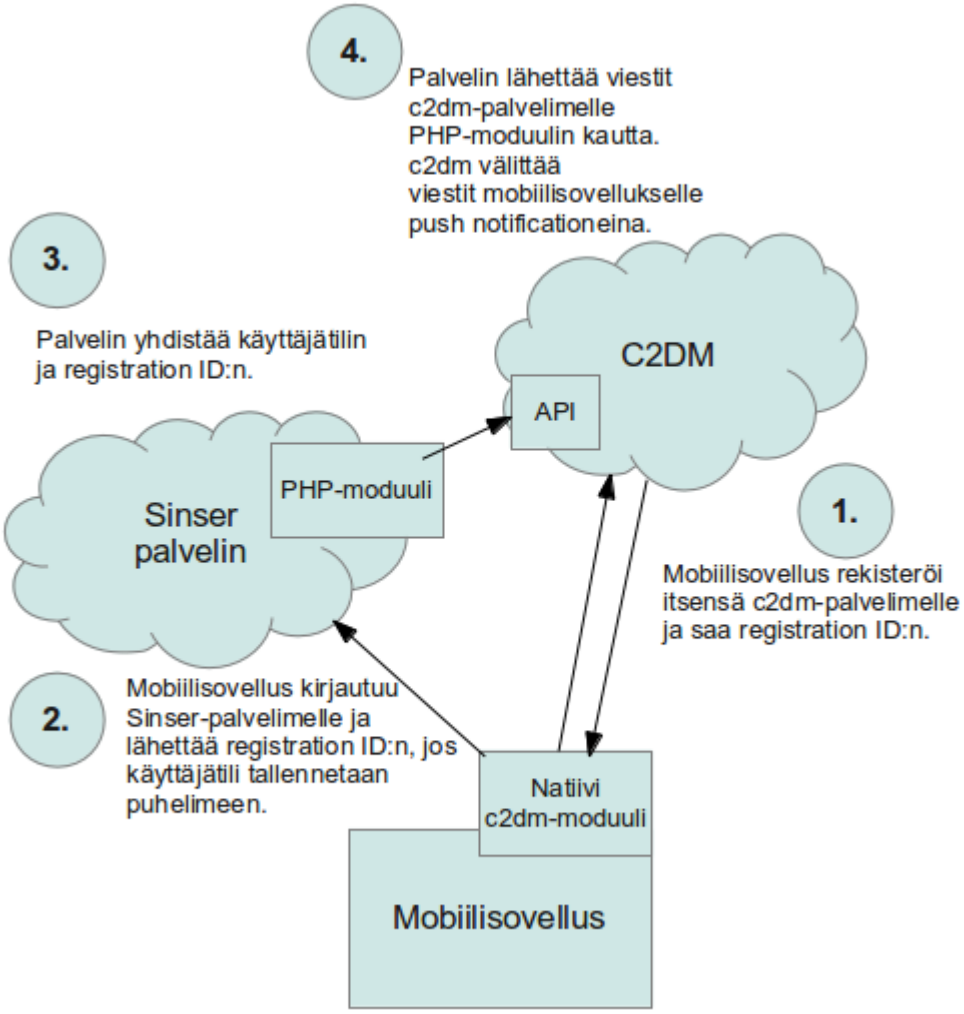


Kuva 17. Push notification

Kuvassa 17 näkyy, kun puhelin on vastaanottanut push notificationin. Seuraavaksi aloitin toteutuksen suunnittelemisen.

7.3 Suunnitelma

Tässä luvussa kerrotaan kuinka c2dm on suunniteltu toimimaan mobiilisovelluksessa (Kuva 18). Push notificationit vaativat muutoksia palvelimelle sekä c2dm-moduulin yhdistämisen Sinser-mobiilisovellukseen.



Kuva 18. Hälytyksien suunnitelma

Kuvassa 18 näkyy suunniteltu push notificationeiden toiminta. Suunnitelmassa c2dm-moduuli on yhdistettynä mobiilisovellukseen. Moduuli tekee rekisteröinnit c2dm-palveluun sekä kuuntelee push notification -viestejä ja ilmoittaa niistä käyttäjälle.

Ei ole aina varmaa, että käyttäjä kirjautuu Sinser-palveluun omalta henkilökohtaiselta puhelimelta tai Android-laitteelta. Tämän vuoksi rekisteröinti on hyvä tehdä vain silloin, kun käyttäjätili tallennetaan puhelimeen. Mobiilisovellus lähettää registration_id:n Sinser-palvelimelle, joka yhdistää sen käyttäjätiliin.

Palvelimen täytyy tunnistautua, ennen kuin se pystyy lähettämään viestejä c2dm-palvelimelle. Tunnistautuminen täytyy tehdä vain kerran, jotta saadaan viestien lähetyksessä tarvittava ”Auth token”. Tämä on yksilöivä tunniste, jonka avulla c2dm-palvelin tunnistaa palvelimen ja sallii sen lähettää viestejä.

Palvelin lähettää push notificationin mobiilisovellukseen, kun jokin tapahtuma sen vaatii. Esimerkiksi toinen käyttäjä jakaa muistion tai käyttäjälle saapuu viesti. Palvelin tekee tämän hakemalla ensin käyttäjän registration_id:n. Tämän jälkeen palvelin lähettää viestin c2dm-palvelimelle registration_id:n avulla.

Avain	**Selite**
registration_id	Yksilöllinen tunniste, jonka käyttäjä saa sovelluksen rekisteröidyttyä c2dm-palveluun. Tunnisteen avulla push notificationit pystytään lähettämään oikeaan älypuhelimeen.
collapse_key	Avain, jota käytetään viestien kokoamisessa, kun puhelin on poissa verkosta. Tällä varmistetaan, ettei monia viestejä lähetätä käyttäjälle kerralla puhelimen liittyessä takaisin verkkoon.
data.<key>	Tietoa voidaan lähettää avain-tieto pareina.
delay_while_idle	Avain, jolla kerrotaan viive viestien lähettämisessä, kun puhelin on niin sanotusti nukkumassa tai lepotilassa.
Tunnistautuminen: GoogleLogin Auth=[AUTH_TOKEN]	Tunniste, jolla palvelin tunnistautuu c2dm-palveluun.

Taulukko 1. C2dm-palvelimelle lähetettävät tiedot

Taulukossa 1 näkyvät tiedot, jotka palvelin lähettää c2dm-palvelimelle. Kun c2dm-palvelin vastaanottaa viestin, se lähettää push notificationin puhelimeen. C2dm-moduuli on käynnistänyt Android-järjestelmän taustalle prosessin, joka kuuntelee saapuvia push notificationeita.

7.4 Titanium-c2dm-moduulin yhdistäminen Sinser-mobiilisovellukseen

Moduulin yhdistäminen oli helppoa ja sain sen tehtyä keskustelupalstoilta löytyneiden keskustelujen sekä esimerkkisovelluksen avulla. Aluksi moduuli täytyi asentaa oikeaan hakemistoon projektissa. Moduulit tulee sijoittaa modules-kansioon ja sinne tulee tehdä moduulin Java-paketin mukainen kansio. C2dm-moduulin tapauksessa kansio on com.findlaw.c2dm. Tänne sijoitetaan moduulin tiedostot. Libs-kansio sisältää moduulin käyttämät kirjastot, jar-tiedosto sisältää käännetyn Java-ohjelmakoodin ja manifest-tiedosto sisältää tietoja kääntäjälle. Moduulin timodule.xml-tiedosto sijoitetaan projektin juurihakemistoon.

Timodule.xml-tiedostossa määritellään moduulin oikeudet Android-järjestelmässä. Esimerkiksi tiedostossa määritellään moduulin oikeus käyttää puhelimen verkkoyhteyksiä, vastaanottaa viestejä sekä käynnistää taustaprosessi, joka odottaa push notificationeita.

Projektin tiapp.xml-tiedostoa tulee myös muuttaa, jotta moduuli toimisi. Tiapp.xml-tiedosto sisältää tietoja ja asetuksia kääntäjälle sovelluksesta. Esimerkiksi tiedosto sisältää asetukset sovelluksen käännöksistä eri käyttöjärjestelmille, kuten Androidille, iOS-järjestelmälle tai Blackberryn käyttöjärjestelmälle. Tiedosto sisältää myös tiedot käytettävistä moduuleista, jotta kääntäjä tietää linkittää ne käännöksen aikana. Tämän jälkeen moduuli on asennettu ja valmis käytettäväksi projektissa.

```
var c2dm = require('com.findlaw.c2dm');
Unelmakone.c2dm.register = function(){
        c2dm.registerC2dm(senderId, {
                success : function(e) {
                Unelmakone.c2dm.sendRegistrationId(e.registrationId);
                },
                error : function(e) {
                       var message;
                       if(e.error == "ACCOUNT_MISSING") {
                              message = "No Google account found; you'll
need to add one (in Settings/Accounts) in order to activate
notifications";
                       }
                       Titanium.UI.createAlertDialog({
                              title : 'Push Notification Setup',
                              message : message,
                              buttonNames : ['OK']
                       }).show();
                }
        });
};
```

Ylläolevassa koodiesimerkissä näkyy kuinka rekisteröiminen c2dm-palveluun tehdään JavaScript koodissa. Ensimmäisellä rivillä ladataan c2dm-moduuli käytettäväksi ja se tallennetaan c2dm-muuttujaan. Tämän jälkeen moduulin funktioita voi käyttää seuraavasti: c2dm.funktionnimi. Rekisteröityminen c2dm-palveluun tapahtuu c2dm.register-funktion avulla. Funktio ottaa parametreikseen lähettäjänä toimivan Google-tilin osoitteen ja objektin, joka sisältää callback-funktiot. Callback-funktioita tarvitaan kaksi, joista toista kutsutaan virhetilanteessa ja toista onnistuneen rekisteröinnin jälkeen. Callback funktioille on annettu parametrinä e-muuttuja. E-muuttuja tulee sisältämään c2dm-palvelimen antaman vastauksen tai virheviestin.

Jos rekisteröinti onnistuu, täytyy saatu registration_id lähettää Sinser-palvelimelle. Olen tehnyt lähettämiselle oman funktion nimeltä sendRegistrationId(). Tämä funktio ottaa parametrikseen registration_id:n ja lähettää sen Sinser-palvelimelle rajapintaa käyttäen.

Mahdollisessa virhetilanteessa käyttäjälle tulee lähettää ilmoitus tapahtuneesta virheestä. Virhe näytetään käyttäjälle dialogina. Dialogi luodaan createAlertDialog()-funktiolla. Funktio ottaa parametreikseen dialogin tarvitsemat tiedot. Title tarkoittaa dialogin otsikkoa, message näytettävää viestiä ja buttonNames tarkoittaa näytettäviä painikkeita dialogissa. Dialogin saa näkyviin show()-funktion avulla. Virhekoodin saa selville e-muuttujasta. E-muuttujan ominaisuus e.error sisältää virheviestin, joka käsitellään if-lauseen avulla. Virheviesti asetetaan tapahtuneen virheen mukaan. Esimerkissä on käsiteltynä vain ”ACCOUNT_MISSING” -virhetilanne. Tämä tarkoittaa, että käyttäjän Google-tili ei ole kirjautuneena puhelimeen.

Moduulin yhdistämisessä tuli huomioida myös erikoistilanteet. Jos käyttäjä poistaa Sinser-mobiilisovelluksen puhelimestaan, niin palvelimelle jää tieto edelleen registration_id:stä. Käyttäjä voi myös tyhjentää puhelimensa muistin, jolloin tallennettu Sinserin käyttäjätili poistuu. Testieni mukaan c2dm-palvelu voi edelleen lähettää tässä tapauksessa push notificationeita puhelimeen.

Havaitsin, että käyttäjän poistaessa Sinser-mobiilisovelluksen, puhelin poistaa automaattisesti rekisteröinnin c2dm-palvelusta. Tällöin käyttäjä ei voi enää vastaanottaa push notificationeita. Myöskin moduulin käynnistämä taustaprosessi

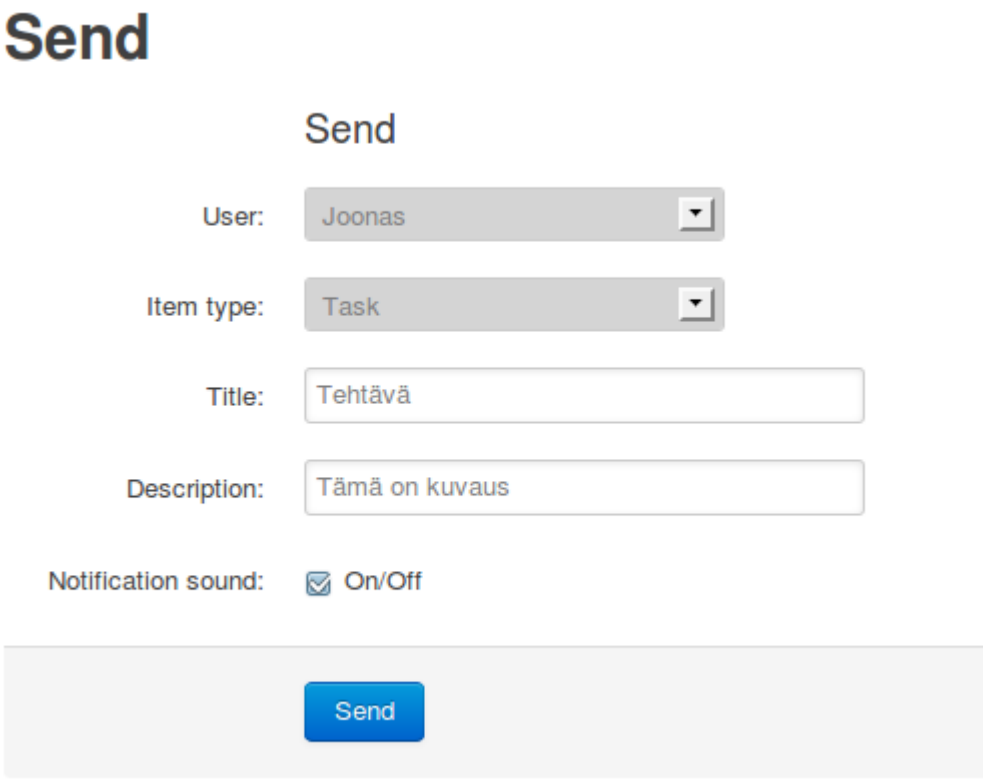
suljetaan ja poistetaan. Sinser-palvelimelta puhelimen registration_id voidaan poistaa, kun seuraavan kerran yritetään lähettää push notificaatio kyseisen registration_id:n puhelimeen. Sinser-pavelin saa tällöin c2dm-palvelusta virhekoodin, jossa ilmoitetaan, että kyseinen registration_id puuttuu. Sinser-palvelin voi tällöin poistaa sen tietokannastaan.

Toisessa erikoistilanteessa, jossa käyttäjä tyhjentää puhelimen muistin, ei testieni mukaa poistettu automaattisesti myös rekisteröintiä c2dm-palvelimelta. Muutin tätä tilannetta varten c2dm-moduulin koodia lisäämällä tarkistuksen. Moduuli tarkistaa push notificationin saapuessa, että puhelimeen on tallennettu käyttäjän Sinser-käyttäjätili. Jos sitä ei ole tallennettu, niin push notificationista ei ilmoiteta käyttäjälle ja rekisteröityminen poistetaan c2dm-palvelimelta. Jos käyttäjätili löytyy tallennettuna, niin push notificationista ilmoitetaan normaalisti.

Toteutin c2dm-moduuliin myös kaksi ominaisuutta, joita säädetään palvelimelta lähetettävillä parametreilla. Ensimmäinen ominaisuus on äänetön push notification hälytys. Tämä on toteutettu push notification viestin mukana lähetettävällä nosound-parametrillä, jonka arvo asetetaan nollaksi. Tällöin push notificationistä tulee käyttäjälle huomautus, mutta äänetön sellainen. Toinen ominaisuus on hälytystä koskeva muistio. Jos push notificationin mukana lähetetään id-parametri, niin Sinser-mobiilisovellus avaa kyseistä id:tä vastaavan muistion. Esimerkiksi käyttäjä on jakanut muistion toisen käyttäjän kanssa. Tästä lähetetään hälytys käyttäjälle. Käyttäjän puhelin vastaanottaa push notificationin. Käyttäjä painaa push notificationia, jolloin puhelin käynnistää Sinser-mobiilisovelluksen, joka avaa jaetun muistion käyttäjälle tarkasteltavaksi. Jos id:tä ei lähetetä, niin push notificationia painamalla käynnistyy vain mobiilisovellus.

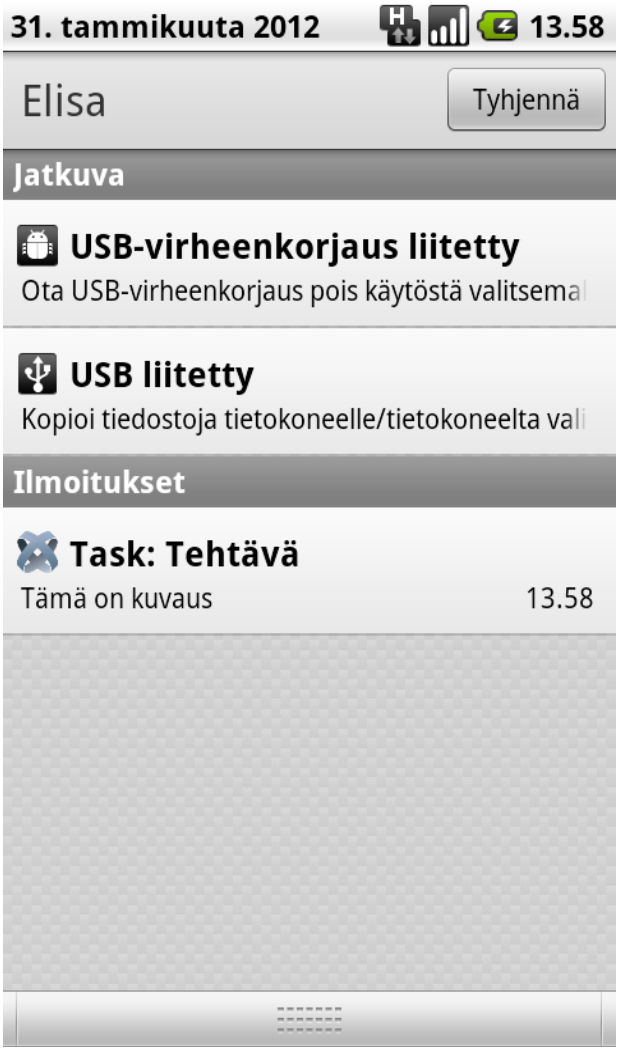
7.5 Lopputulos

Hälytykset onnistuivat mielestäni todella hyvin. Ratkaisu oli toimiva, halpa ja melko helppo toteuttaa. Vaikeuksia aiheutti ainoastaan moduulin muokkaaminen toimivaksi uusimmassa Titanium Mobilen versiossa. Testasin moduulia omalla testipalvelimellani. Konfiguroin palvelimen ajamaan push notificationin lähetysscriptin tunnin välein. Tein myös yksinkertaisen weblomakkeen (Kuva 19), jolla pystyi lähettämään push notificationeita halutuilla parametreilla.



Kuva 19. Testauslomake

Kuvassa 19 näkyy tekemäni yksinkertainen testilomake. Puhelin vastaanotti push notificationin jokaisella kerralla (Kuva 20). Testasin myös push notificationin mukana lähetettävät parametrit tarkoin Androidin omia kehitystyökaluja käyttäen. Testauksessa käytin ddms-ohjelmaa, jolla voi tarkastella puhelimen lokia. Lokitulosteesta näin push notificationin tarkat tiedot parametreineen ja mahdollisine virhetilanteineen. Testasin myös erikoistilanteet. Jos käyttäjä sammuttaa puhelimen ja käynnistää sen uudelleen, niin voiko puhelin tämän jälkeen tehdä hälytyksiä. Testieni mukaan c2dm-taustaprosessi käynnistyy itsestään ja hälytysten tekeminen onnistuu.

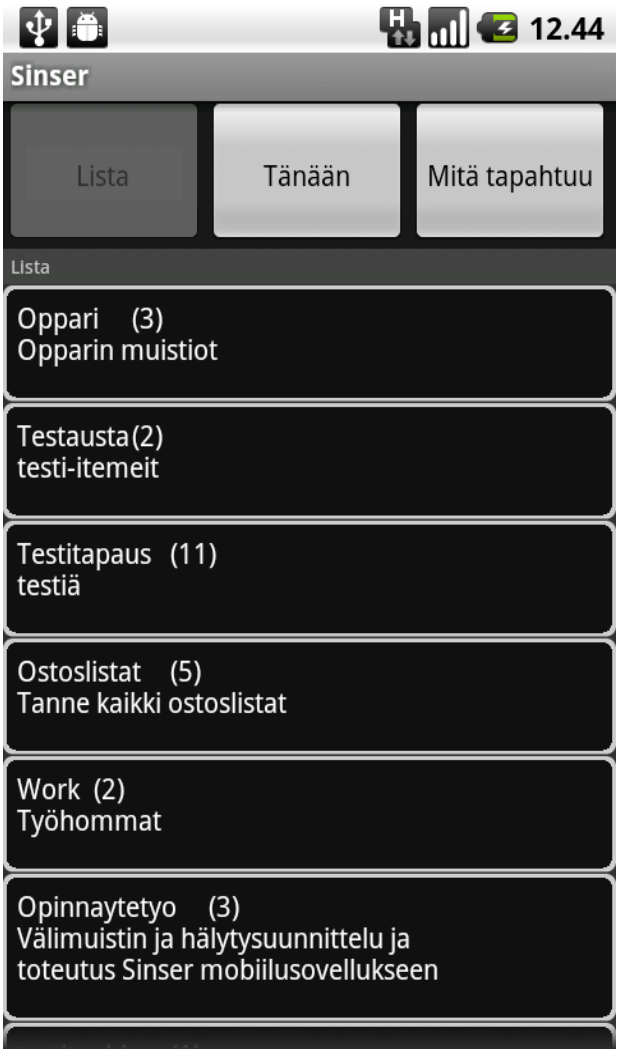


Kuva 20. Onnistunut push notification lähetys

Kuvassa 20 näkyy, miltä lopullinen push notification näyttää puhelimessa. Kuvasta voi nähdä, että toteutus on onnistunut. Lopputuloksena syntyi toimiva moduuli vastaanottamaan push notificationeita ja tekemään hälytyksiä. Moduuli yhdistettiin onnistuneesti Sinser-mobiilisovellukseen. Tavoitteena oli saada hälytyksiä käyttäjälle nopeasti eri tapahtumista. Tavoitteeseen päästiin.

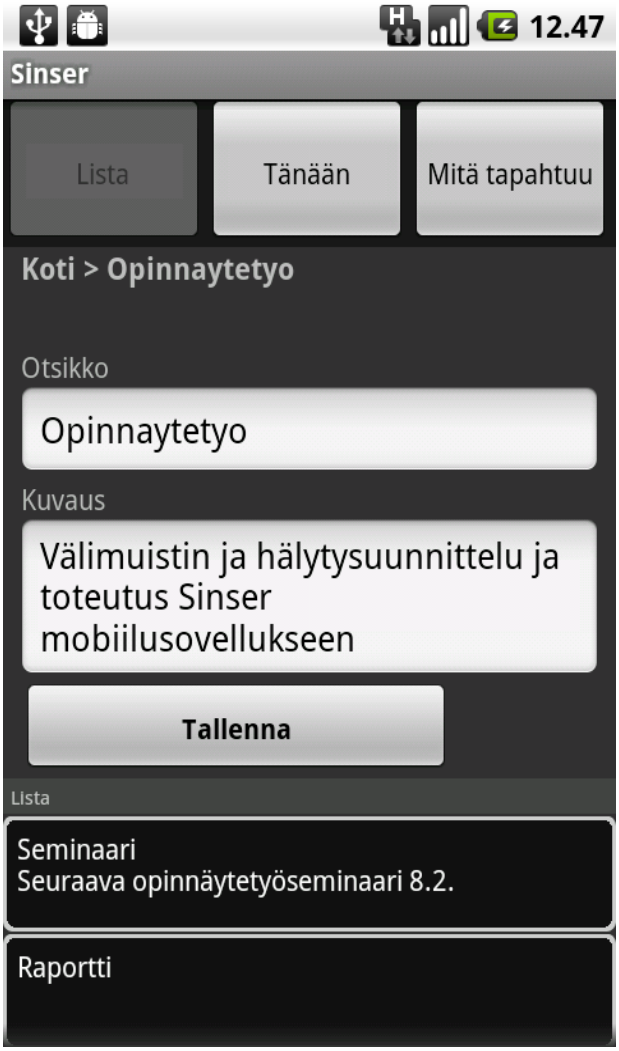
8 Lopputulos ja päätelmät

Mobiilisovelluksen kehittäminen eteni paljon opinnäytetyön aikana. Kehitystä tapahtui enimmäkseen mobiilisovelluksen sisäisissä toiminnoissa, mutta myös ulkoasullisia muutoksia tapahtui (Kuva 21).



Kuva 21. Sinser-mobiilisovelluksen uusi versio

Kuvassa 21 on opinnäytetyön toteutusvaiheen jälkeinen kuva mobiilisovelluksen kehitysversiosta. Jos vertaa kuvaa aikaisempaan versioon, kuten esimerkiksi kuvaan 3, voi huomata että ulkoasu on muuttunut yksinkertaisemmaksi sekä viimeisimmät lista on hävinnyt (Kuva 22). Ulkoasua on pyritty yhtenäistämään webversion kanssa. Ulkoasu tuskin tulee kuitenkaan olemaan lopullinen, vaan sitä hiotaan ja viimeistellään ominaisuuksien valmistuttua.



Kuva 22. Mobiilisovelluksen uuden version avattu muistio

Kuvassa 22 näkyy toinen kuva kehitysversiosta. Kuvassa on avattu Opinnäytetyö-niminen muistio. Sen alla näkyvät muistiot Seminaari ja Raportti. Muistiota voi muokata kentistä ja painamalla Tallenna-painiketta. Muokkausnäkymä tulee vielä todennäköisesti tulevaisuudessa muuttumaan.

8.1 Välimuisti

Välimuistin toteutuksessa opinnäytetyössä oli tavoitteena parantaa käyttäjäkokemusta mobiilisovelluksessa, vähentää siirretyn datan määrä, nopeuttaa mobiilisovellusta sekä mahdollistaa mobiilisovelluksen osittainen offline-käyttö. Nämä pyrittiin ratkaisemaan luomalla laitteelle välimuisti, johon tallennetaan kaikki väliaikainen tieto.

Tutkimusvaiheessa selvitin monia välimuistiin liittyviä asioita ja ongelmia. Ongelmana havaitsin datan tuoreuden selvittämisen ja sen kuinka varmistutaan siitä, että käyttäjälle näytetään tuorein tieto. Tutkin myös yhteensopivuuksia ja eri menetelmiä toteuttaa välimuisti Titanium Mobilella. Minkälaisia tekniikoita tiedon tallentamisessa puhelimeen on Titanium Mobilessa. Valitsin tekniikaksi SQLiten. Olen käyttänyt SQLiteä aikaisemmin eräässä toisessa projektissa, joten paljoa uuden opettelemista sen käytössä ei ollut. Opettelin kuitenkin kuinka SQLiteä voi käyttää Titanium Mobilessa.

Tein nopeasti ensimmäisen toimivan version välimuistista, kuten ketteriin menetelmiin kuuluu. Tämän jälkeen tein käytännön testejä puhelimellani, kuten olin luvussa 6.2 kuvannut. Kerroin luvussa havainneeni, ettei suurta eroa ole aikaisempaan versioon, jossa välimuistia ei ollut. Tässä kohtaa jouduin pohtimaan paljon yrityksen näkökulmasta ja yrityksen tavoitteiden kannalta Sinser-palvelussa. Esitin välimuistin toteutuksen siirtämistä jatkokehitykseen ja seuraaville Sinser-mobiilisovelluksen versioille, koska mielestäni välimuistin toteuttaminen ei ole tärkeä ominaisuus ensimmäisessä julkaistavassa mobiilisovelluksen versiossa. Yritykselle syntyy kustannussäästöjä, mobiilisovellus valmistuu nopeammin ja välimuistista ei ole paljoa näkyvää hyötyä käyttäjälle. Lopulta päätettiin, ettei välimuistia toteuteta ensimmäiseen julkaistavaan versioon. Ehdin jo kuitenkin suunnitella välimuistia ja aloitin toteutuksen. Työni ei kuitenkaan mene projektin kannalta hukkaan, vaan todennäköisesti suunnitelmani otetaan taas esiin jatkokehityksessä.

Opinnäytetyötä tehdessäni havaitsin ajattelutavassani monia muutoksia. Mielestäni tämä oli yksi tärkeimmistä opeistani opinnäytetyötä tehdessäni. Ennen ajattelin projekteissani, että uusia hienoja teknisiä ominaisuuksia on hieno toteuttaa ja en ajatellut tarpeeksi käyttäjää. Nyt opinnäytetyötä tehdessäni huomasin ajattelevan asioita enemmän ja enemmän käyttäjän sekä yrityksen näkökulmasta. Toimitusjohtaja Mikko Mäkelä on aina painottanut helppokäyttöisyyttä ja käyttäjän näkökulmasta ajattelua. Ymmärrän vähitellen tämänlaisen ajattelun. Huomaan myös ajattelevani enemmän yrityksen näkökulmasta kuin ennen. Yrityksen tavoitteena on saada voittoa sovelluksesta. Voittoa ei saada, jos käyttäjät eivät käytä sovellusta, joten sovellus on tehtävä mielenkiintoiseksi ja käytettävyydeltään hyväksi. Sovelluksesta on oltava hyötyä käyttäjälle. Ominai-

suuksien toteuttamisessa ei myöskään saa kulua liikaa aikaa, koska aika maksaa yritykselle ja viivästyttää sovelluksen valmistumista.

8.2 Hälytykset

Hälytyksien alkuperäisenä tavoitteena oli saada käyttäjälle ilmoitus tärkeästä tapahtumasta mahdollisimman nopeasti. Sinser-palvelussa on esimerkiksi mahdollista jakaa muistioita muiden käyttäjien kanssa. Yritysmaailmassa yrityksen työntekijä voi esimerkiksi luoda tehtäviä Sinser-palvelua käyttäen vaikka palaverin pohjalta. Työntekijä voi jakaa nämä tehtävät työkavereidensa kanssa Sinserissä. Osa tehtävistä voi olla tärkeitä, jolloin työntekijöille on saatava ilmoitus jaetuista tehtävistä nopeasti. Sinseriä voi käyttää myös esimerkiksi apuna etätöiden hallinnassa jaettaessa työtehtäviä. Toisena esimerkkinä voi olla yksinkertainen ostoslista. Perheenjäsen voi jakaa esimerkiksi Sinserissä tekemänsä kauppalistan kaupassa asioivalle toiselle perheenjäsenelle. Jos tämä perheenjäsen on ehtinyt jo lähteä kohti kauppaa, täytyy hänelle saada ilmoitus kauppalistasta. Opinnäytetyössä oli yhtenä tavoitteena keksiä ratkaisu ongelmaan, jossa käyttäjälle on saatava ilmoitus mahdollisimman nopeasti.

Tutkimusvaiheessa minulla oli monia toteutusvaihtoehtoja. Jouduin vertailemaan vaihtoehtoja monista eri näkökulmista. Onko vaihtoehto teknisesti yhteensopiva mobiilisovelluksen kanssa? Onko vaihtoehto skaalautuva kasvavalle käyttäjämäärälle? Toteutuuko tavoite saada hälytys nopeasti käyttäjälle? Kuinka paljon tekniikka vaatii työtä ja aikaa sekä minkälaisia kuluja yritykselle siitä mahdollisesti syntyy? Sain karsittua kysymysten pohjalta osan teknisistä ratkaisuista ja päädyin käyttämään push notification -teknologiaa. Tutkittuani monia eri push notification palveluntarjoajia, päädyin lopulta Googlen tarjoamaan c2dm-palveluun. Tutkin myös Urban Airship -nimisen yrityksen tarjoamaa tekniikkaa, mutta palvelun maksullisuuden vuoksi, päädyttiin käyttämään Googlen palvelua. Googlen palvelu on ilmainen ja sille löytyi myös moduulituki Titanium Mobilessa. Testauksessa valinta osoittautui hyväksi ja toimivaksi. Oletan sen myös skaalautuvan isoille käyttäjämäärille Googlen suuren palvelinkapasiteetin avulla.

Suurimpana haasteena hälytysten suunnittelussa ja toteuttamisessa oli päättää Googlen ja Urban Airshipin välillä. Alussa minulla ei ollut tietoa Googlen palvelun ja Titanium Mobilen yhteensopivuudesta, joten päätös vaati paljon tutkimista

ja vertailua. Päätös kuitenkin syntyi, kun löysin yhteensopivan moduulin Googlen palvelun kanssa Github-palvelusta. Toinen paljon haastetta tuonut asia oli c2dm-moduulin päivittäminen yhteensopivaksi uusimman Titanium Mobilen kanssa.

Osallistuin c2dm-moduulin päivittämisessä avoimen lähdekoodin projektiin Github-verkkopalvelussa. Muokkasin moduulin koodin toimimaan Titanium Mobilen uusimmassa versiossa ja lähetin tekemäni koodimuutokset takaisin projektiin muiden käytettäväksi. Ilmoitin myös tekemästäni päivityksestä keskustelupalstalla, joka käsittelee push notificationeita ja kyseistä moduulia. Olen saanut muilta kehittäjiltä palautetta ja avunpyyntöjä moduuliin liittyen. Olen pyrkinyt auttamaan muita kehittäjiä ja samalla luonut pientä mainetta itselleni. Aion myöskin jatkaa moduulin kehittämistä. Eräs käyttäjä esimerkiksi ilmoitti, että moduuli ei enää toimi Titanium Mobilen päivityttyä versioon 1.8.1. Aion selvittää virheen ja korjata moduulin jälleen toimivaksi. Pienestä maineesta voi olla hyötyä tulevaisuutta ajatellen.

Opin mielestäni paljon hälytyksiä tehdessä. Jouduin pohtimaan paljon yrityksen näkökulmasta. Esimerkiksi jokin omasta mielestäni mielenkiintoinen ja hyvältä vaikuttava tekniikka olisi ollut yrityksen kannalta kallis ja aikaa vievä toteuttaa. Oli löydettävä kompromissiratkaisu ja sellainen löydettiinkin. Lopulta valittu tekniikka ja ratkaisu osoittautuvat testeissä oikeaksi ja hyväksi. Opinnäytetyössä tutustuin minulle ennestään uuteen tekniikkaan push notificationeihin sekä paransin osaamistani Titanium Mobilessa moduuliohjelmoinnin osalta. En ollut aikaisemmin ohjelmoinut natiiveja moduuleja Titanium Mobilelle, mutta pääsin tutustumaan ja opettelemaan niiden ohjelmoimista c2dm-moduulia päivittäessäni. Parasta ja antoisinta tässä mielestäni oli se, että työstäni c2dm-moduulin parissa on hyötyä Sinser-mobiilisovelluksen lisäksi myös muille kehittäjille.

Hälytyksissä tavoitteisiin päästiin. Alkuperäinen tavoite toteutui suunnitelmien mukaan. Hälytysten toteutus oli onnistunut ja nyt se muodostaa yhden osan Sinser-mobiilisovelluksesta.

Kaiken kaikkiaan opinnäytetyö onnistui mielestäni hyvin. Se tehtiin nopeassa aikataulussa tehokkaasti. Ketterät menetelmät mahdollistivat nopean ja joustavan kehitystyön, eikä aikaa kulunut määrittelyiden tekemiseen. Mobiilisovelluk-

sen ja Sinserin kehityksen kulusta sovittiin lyhyillä palavereilla sekä sähköpostiviestinnällä. Ketterät menetelmien avulla muutoksia oli helppo mielestäni toteuttaa mobiilisovellukseen.

Jatkossa tulen kehittämään sovellukseen uusia ominaisuuksia, ulkoasua tullaan parantamaan ja käytettävyyttä hiotaan paremmaksi. Eräs yrityksen idea aiotaan myös mahdollisesti sovittaa ja testata mobiilisovelluksessa. Omasta mielestäni dokumentaatiota kannattaa myös tehdä. Nyt dokumentoituna on ainoastaa Sinser-palvelimen rajapinta, mutta mielestäni dokumentaatiota kannattaa tehdä myös mobiili- ja websovelluksen osalta. Esimerkiksi jos mobiilisovelluksesta halutaan tehdä iOS-versio iPhonelle, on välttämätöntä, että mobiilisovelluksesta on dokumentaatiota. Itselläni ei ole vielä kokemusta iOS-ohjelmoinnista Titanium Mobilella. Jos yrityksellä on tiukka aikataulu muuntaa sovellus toimivaksi iPhonella ja halutaan palkata aputyövoimaa tätä varten, on tällöin hyvä olla dokumentaatiota Sinser-palvelusta ja mobiilisovelluksesta.

Mobiilisovellukselle on myös hyvä tehdä käytännön testaamista. Sinser-palvelua ja mobiilisovellusta tulee testata pienessä ryhmässä käytännön tilanteissa? Onko joistakin ominaisuuksista hyötyä ja mitä ominaisuuksia puuttuu? Onko sovelluksen käytettävyys hyvä ja mitä mielipiteitä käyttäjillä on sovelluksesta?

Kaiken kaikkiaan opinnäytetyö onnistui mielestäni hyvin. Sain kiitettävää palautetta yritykseltä ja olen itse myös tyytyväinen saatuihin tuloksiin.

Kuvat

Taulukot

Lähteet

10Gen 2011. MongoDB. http://www.mongodb.org/. Luettu 10.12.2011

Appcelerator 2011. Product. http://www.appcelerator.com/products/. Luettu 16.12.2011

Appcelerator 2011. Titanium Mobile. http://www.appcelerator.com/products/titanium-mobile-application-development/, Luettu 17.12.2011

Google 2011. Android Cloud to Device Messaging Framework. http://code.google.com/android/c2dm/, Luettu 28.12.2011

IBM 2011. RESTful Web Services: The Basics. https://www.ibm.com/developerworks/webservices/library/ws-restful/. Luettu 10.12.2011

Internet Engineering Task Force 2011. RCF4627. http://tools.ietf.org/html/rfc4627. Luettu 8.12.2011

Lean Development Oy 2011. Company. http://leandevelopment.fi/company, Luettu 1.12.2011

Lean Development Oy 2011. Development. http://leandevelopment.fi/development, Luettu 1.12.2011

Sininen meteoriitti 2012. Ketteryys haltuun: Scrum pähkinänkuoressa. http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-scrum-pahkinankuoressa. Luettu 28.2.2012

SQLite 2012. About. http://www.sqlite.org/about.html, Luettu 19.1.2012

SQLite 2012. Limits In SQLite. http://www.sqlite.org/limits.html#max_length, Luettu 23.1.2012

Urban Airship 2011. Pricing. http://urbanairship.com/pricing/. Luettu 12.12.2011

W3C 2011. RFC 2616. http://www.w3.org/Protocols/rfc2616/rfc2616. Luettu 3.12.2011

Wikipedia 2011. Cache. http://en.wikipedia.org/wiki/Cache. Luettu 3.12.2011