OULUN SEUDUN
AMMATTIKORKEAKOULU

Jussi Witick

# ANDROID-ASIAKASOHJELMA TUNTIENSEURANTAJÄRJESTELMÄÄN

# ANDROID-ASIAKASOHJELMA TUNTIENSEURANTAJÄRJESTELMÄÄN

Jussi Witick
Opinnäytetyö
27.4.2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

## TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, Ohjelmistojen tuotanto

---

Tekijä: Jussi Witick
Opinnäytetyön nimi: Android asiakasohjelma tuntienseurantajärjestelmään
Työn ohjaaja(t): Kari Salonen, Eneris Solutions Oy; Pekka Alaluukas, Oulun seudun ammattikorkeakoulu
Työn valmistumislukukausi ja -vuosi: kevät 2011 Sivumäärä: 68

---

TIIVISTELMÄ

Tässä projektissa toteutettiin Eneris Solutions Oy:n omaan tuntienseurantajärjestelmään Android-mobiilialustalle asiakasohjelma. Työn tilaajana toimi Eneris Solutions Oy. Asiakasohjelma toteutettiin olemassa olevan serveri- ja protokollatoteutuksen kanssa yhteensopivaksi.

Ohjelman avulla käyttäjä voi merkitä, poistaa ja muokata tunteja, käyttäjiä ja projekteja. Sovellus tarjoaa myös karkean tason raportointitoiminnon. Ohjelma on suunniteltu toimimaan kaikilla Android-laitteilla, joissa on käytössä vähintään Android-alustan versio 2.1. Sovellus tukee myös kaikkia eri näyttökokoja.

Projekti on toteutettu Java-kielellä. Osa Java-koodista tuli serverille ja osa asiakasohjelmaan. Asiakasohjelman Java-koodi käyttää Android-kirjastoja. Sovellus käyttää alustariippumatonta JSON-viesteihin perustuvaa protokollaa keskustellessa Java-servletin kanssa serverillä.

Projektissa onnistuttiin toteuttamaan kaikki sille asetetut tavoitteet aikataulussa. Päätavoitteita oli opetella Android-alustan sovelluskehityksen perusteet sekä saada toimiva asiakasohjelma tuntienseurantajärjestelmään Android-alustalle.

---

Asiasanat: Android, Java, mobiililaite, Server/Client-arkkitehtuuri, JSON

**ABSTRACT**

Oulu University of Applied Sciences
Information Technology, Software Development

---

Author: Jussi Witick
Title of thesis: Android client for hourtracking software
Supervisor(s): Kari Salonen, Eneris Solutions Oy; Pekka Alaluukas, Oulu University of Applied Sciences
Term and year when the thesis was submitted: Spring 2011 Number of pages: 68

---

ABSTRACT

This project is an extension to an hour tracking software developed by Eneris Solutions Oy. The result of this project is an Android-based client program for mobile devices using the same server and protocol that previous client versions use. The client of this project is Eneris Solutions Oy.

With this client program the user can add, remove or edit hour entries, users and projects. This application also provides a simple hour reporting tool. The program is designed to work on all Android-based mobile devices, which are using at least Android version 2.1. It also supports various screen sizes of different devices.

The project has been developed with Java programming language. Part of the Java code was developed for the server environment and most of the code was for the Android client. The client's Java code uses Google's Android Java libraries and the program uses platform independent JSON based protocol when exchanging data with the server's Java -servlet.

This project was finished with all objectives completed in the time designated for the project. The main objective of this project was to learn the basics for Android-based software development and to finish a working version of Android-based client for the Eneris' hour tracking software.

---

Keywords: Android, Java, Mobile device, Server/Client architecture, JSON

# ALKULAUSE

Tämä opinnäytetyö on tehty Eneris Solutions Oy:lle keväällä 2011. Opinnäytetyössä oli mukana Eneriksen teknologiajohtaja Kari Salonen, jonka tehtävänä oli tarjota teknistä apua työn eri osa-alueilla, oikolukea opinnäytetyön dokumentti ja tarvittaessa sensuroida työn sisältöä ennen työn lähettämistä koulun ohjaavalle opettajalle, lehtori Pekka Alaluukkaalle. Projektissa oli myös mukana Eneriksen ohjelmistotestaaja Ville Kelloniemi, jonka tehtävänä oli käyttöliittymäsuunnittelu ja ohjelman testaus. Hän oli myös mukana teknisen tason vaatimusten luonnissa, muokkauksessa ja tulkinnassa koko projektin ajan. Ohjaavan opettajan tehtävänä oli tarkistaa työn tekninen sisältö ja tekstinohjaajan lehtori Tuula Hopeavuoren tehtävänä oli tarkistaa tekstin oikeellisuus.

Oulussa 27.4.2011

Jussi Witick

# SISÄLLYS

TIIVISTELMÄ
ABSTRACT
ALKULAUSE
SISÄLLYS 6
LYHENTEET JA TERMINOLOGIA 9
1 JOHDANTO 12
2 LÄHTÖKOHTA JA VAATIMUKSET 13
2.1 Projektin lähtökohta 13
2.2 Vaatimukset 13
2.2.1 Arkkitehtuuritason vaatimukset 14
2.2.2 Toiminnalliset vaatimukset 14
2.2.3 Tekniset vaatimukset 15
3 ANDROID-ASIAKASOHJELMA 16
3.1 Ohjelman toiminnot 16
3.1.1 Normaalin käyttäjän toiminnot 16
3.1.1.1 Tuntienkirjaus toiminto 16
3.1.1.2 Raportointitoiminto 20
3.1.2 Admin-tason toiminnot 22
3.1.2.1 Projektien hallinta 23
3.1.2.2 Käyttäjien hallinta 25
3.2 Arkkitehtuuri 29
3.2.1 Aktiviteetit 29
3.2.2 Käyttöliittymät 30
3.2.3 Managerit 30
3.2.4 Kansiorakenne 31
3.2.5 Android manifest -tiedosto 33
4 SERVER/CLIENT-PROTOKOLLA 35
4.1 Alkuperäinen protokolla 35
4.2 Alustariippumaton protokolla 36
5 TIMESHEET SERVER 38
5.1 Arkkitehtuuri 38

5.1.1 Servletit 38
5.1.2 Actionit 38
5.2 Projektispesifiset muutokset serverille 39
6 KÄYTETYT TEKNOLOGIAT 40
6.1 Android kohdealustana 40
6.2 Java 41
6.3 JSON 42
6.4 Tietokannat 43
6.4.1 TimeSheet serverin MySQL 43
6.4.2 Android-päätelaitteen SQLite 43
7 TYÖVÄLINEET 45
7.1 Eclipse 45
7.2 Android SDK 45
7.3 NetBeans IDE 46
7.4 MySQL command line 46
7.5 SVN command line 47
8 PROJEKTIN TOTEUTUS 49
8.1 Aloitus 49
8.2 Projektin hallinta 49
8.3 Versionhallinta 50
8.4 Suunnittelu 50
8.4.1 Käyttöliittymän suunnittelu 50
8.4.2 Serverin muutosten suunnittelu 52
8.5 Ohjelmiston kehitys 52
8.5.1 Käyttöliittymän toteutus 52
8.5.1.1 Android xml -tiedostot ja käyttöliittymän määritys 52
8.5.1.3 Androidin valmiit komponentit, eli widgetit 54
8.5.1.4 Androidin ja Qt:n käyttöliittymäkoodauksen eroja 55
8.5.2 Client-ohjelman toteutus 56
8.5.2.1 Managereiden toteutus 56
8.5.2.2 Aktiviteettien toteutus 57
8.5.2.3 Tilakone 62
8.5.3 Server-ohjelman toimintojen toteutus 63

8.6 Testaus ja laadunvarmistus 64
9 LOPPUSANAT 66
9.1 Jatkokehitys 66
9.2 Pohdinta 66
LÄHDELUETTELO 69

# LYHENTEET JA TERMINOLOGIA

**Client** – Tarkoittaa asiakasohjelmaa, jolla saadaan yhteys johonkin serveriin ja voidaan suorittaa ohjelmiston toteuttamia toimintoja etänä.

**Dedikoitu serveri** – Tarkoittaa serveriä, joka pyörii tietokoneella, joka on varattu ainoastaan kyseisen serverin käyttöön.

**Emulaattori** – Ohjelma, jota voidaan käyttää mobiililaitteiden ohjelmien testaukseen, ennen kuin niitä testataan itse mobiililaitteella.

**IDE** – Integroitu ohjelmistojenkehitysympäristö, joka yleensä sisältää tekstineditointiohjelmiston, ohjelmiston debuggaus- ja kääntämistoiminnot.

**JavaScript** – Ohjelmointikieli, jota käytetään yleensä dynaamisten web-sivustojen luonnissa.

**JSON** – Javascript Object Notation on tekstipohjainen, avoimeen standardiin pohjautuva ja luettava datan siirtomuoto.

**Landscape-moodi** – Tarkoittaa sitä graafista käyttöliittymän asettelua, kun mobiilipäätelaitetta pidetään vaakatasossa eli näytön leveys on suurempi kuin korkeus.

**Lokalisaatio** – Tarkoittaa ohjelmiston tekstien kääntämistä eri maiden kielille.

**Maemo** – Nokian Linuxiin perustuva mobiililaitteille kehitetty käyttöjärjestelmä ja ohjelmistoalusta. Maemo on avointa lähdekoodia.

**Metodi** – Java-ohjelmointikielen käyttämä nimi funktio-kutsuille eli luokkien tietyn asian toteuttaville ohjelmakoodipätkille.

**MySQL** – Relaatiotietokantojen hallintasysteemi, jota ajetaan serverinä.

**Ohjelmavirhe** – Virhe (bugi) ohjelmakoodissa, joka aiheuttaa vääränlaisia toimintoja tai odottamattomia ongelmia ohjelmaa ajettaessa.

**Periyttäminen** – Tarkoittaa sitä, kun jokin koodaajan luokka perii toisen luokan julkiset ja suojatut metodit ja muuttujat ja voi toimia pääluokkansa korvaajana.

**Portrait-moodi** – Tarkoittaa sitä graafista käyttöliittymän asettelua, kun näytön korkeus on leveyttä suurempi eli laitetta pidetään pystysuorassa.

**Qt** – Nokian omistama alustariippumaton ohjelmistokehitysympäristö, joka on laajasti käytössä GUI-ohjelmissa sekä työpöytä- että mobiilialustoilla.

**Relaatiotietokanta** – Tietokanta, jossa kaikkien taulujen suhteet on määritelty erinäisten ehtojen mukaan tiukasti noudatettavaksi.

**Servlet** –Java API Java EE:ssä, joka palvelee HTTP-kutsuja.

**Server** – Palvelinohjelma, joka palvelee yhtä tai useampaa asiakasohjelmaa eli clientiä. Yleensä ohjelmistot koostuvat sekä serveristä että clientistä.

**Staattinen muuttuja ja metodi** – Määritelmä, jonka mukaan tietyn muuttujan tai metodin arvo on tiedossa jo käännön aikana eikä näiden sisältö voi ajon aikana enää muuttua.

**Swing** – Java-ohjelmointikielen graafisten työpöytäsovellusten käyttöliittymien kehitykseen käytettävä kirjasto.

**Toast** – Androidin toiminto, jonka avulla mobiilaitteen näytölle voidaan tulostaa informaatiota halutuksi aikaa.

**Tomcat** – Apachen avoimeen lähdekoodiin pohjautuva WWW-palvelinohjelmisto, joka voi sisältää useita servlettejä ja toimia Java Servlet Containerina.

**Wrapper** – Tarkoittaa luokkaa, joka tekee tietyn asian yksinkertaisemmaksi koodaajalle piilottamalla jonkin laajemman luokan toiminnot yksinkertaisempien metodikutsujen taakse.

**Widget** – Graafinen kontrolli, joka näyttää käyttäjälle koodaajan valitsemaa dataa tietyssä muodossa.

# 1 JOHDANTO

Android-päätelaitteiden määrä on ollut suuressa kasvussa viime vuosina. Tästä johtuen myös Android-alustalle tarvittavien ohjelmistokehittäjien ja ohjelmistojen tarve on kasvussa. Mikäli Android-alustan kasvuvauhti pysyy samana, se lyö kilpailevat iOS:n ja Nokian alustat muutamassa vuodessa. Tällä hetkellä Applen AppStore on suurin ja suosituin ohjelmistokauppa, mutta Androidin kasvu on ollut lupaavampaa viime aikoina. Näistä syistä Eneriksellä päädyttiin ratkaisuun, että on hyvä olla osaamista Nokian alustojen lisäksi myös Android- ja iOS-alustoista.

Tässä projektissa laajennettiin Eneris Solutions Oy:n kehittämän TimeSheet Plus -tuntienseurantaohjelmiston toiminnallisuutta koskemaan myös Android-päätelaitteita. Projektin edeltäjänä toimi Qt-alustalle kehitetty vastaavanlainen sovellus. Ohjelman oli tarkoitus toteuttaa mahdollisimman pitkälti olemassa olevan työpöytäversion toiminnallisuudet. Projektin lähtöideana oli Android-alustan opiskelu mahdollisten tulevaisuuden Android-projektien varalle.

Projekti toteutettiin Java-ohjelmointikielellä Androidin 2.1 -versiolle Samsung Galaxy S -malliselle puhelimelle. Vaikka projekti suunnattiin tälle yhdelle puhelinmallille, projektin toteutus tehtiin siten, että ohjelmisto toimii myös muilla Android-puhelimilla näytön kokoon katsomatta. Projektissa käytettiin sovellettua scrum-mallia ja sitä oli tekemässä ohjelmoijan lisäksi testaaja. Projektissa käyttöliittymät ja testauksen suoritti Eneriksen ohjelmistotestaaja, ohjelman toteutuksen ja suunnittelun teki tämän opinnäytetyön kirjoittaja.

# 2 LÄHTÖKOHTA JA VAATIMUKSET

## 2.1 Projektin lähtökohta

Projektin lähtökohtana oli Eneris Solutions Oy:n halu laajentaa toimialansa sisältämään Android-ohjelmointia. Toinen syy projektin aloittamiselle oli työn tekijän tarve saada opinnäytetyö tehtyä koulunkäynnin valmiiksi saamiseksi. Nämä kaksi vaatimusta saatiin muokattua yhdeksi tämän opinnäytetyön muodossa. Työn tilaajana toimi Eneris Solutions Oy, eli työ tehtiin yrityksen sisäisesti, eikä ulkopuolisia tahoja ollut missään tekemisissä projektin kanssa. Eneriksellä ei ennestään ollut Android-osaamista. Mobiiliosaamista yrityksestä löytyi, mutta Android-projekteja ei vielä tähän mennessä ollut tehty. Koska Android perustuu Java-ohjelmointikieleen, oli Android-alustan opettelulla kuitenkin vahva pohja, sillä Eneris on painottunut juuri Java-ohjelmointiin asiakasprojekteissaan.

Näistä lähtökohdista projektia varten minun tuli opetella Android-alustalle ohjelmointi lähes tyhjästä, jolloin ainoastaan aiempi Java-osaaminen auttoi alkuun pääsyä. Alusta lähtien projektin saamaan suuntaan vaikuttivat suuresti myös aiemmat toteutukset TimeSheet Plus -ohjelmistossa. Näitä olivat TimeSheet Server, TimeSheet Client (Swing-client) sekä Qt-ohjelmointikielellä tehty mobiiliclient (myöhemmin Qt-client) Maemo-alustalle. Projektin toissijaisena tarkoituksena oli näin ollen myös laajentaa olemassa olevan tuntienseurantajärjestelmän TimeSheet Plus:n toiminnallisuutta tukemalla uutta mobiilialustaa, Androidia.

## 2.2 Vaatimukset

Projektissa oli käytännössä kolmen tason vaatimuksia. Ylemmät ohjaavan tason vaatimukset eli arkkitehtuuritason vaatimukset linjasivat sen, että projektin tulee toimia täysin yhteen aiempien TimeSheet Plus -toteutusten (server, Swing-client, Qt-client) kanssa. Toiminnalliset vaatimukset kertoivat sen, mitä

toimintoja clientin tulee toteuttaa. Koska kyseessä on päätelaitteelle tehtävä sovellus, oli täysin selvä, että joitakin toimintoja jouduttiin jättämään toteuttamatta. Tästä huolimatta ennen ohjelman toteuttamista kirjattiin ylös käytännössä kaikki alkuperäisen tuntienseurantaohjelmiston toiminnot, jotka sitten tulisi toteuttaa Android-clientissä. Kolmantena ryhmänä olivat teknisen tason vaatimukset. Teknisen tason vaatimuksia kirjattiin ylös sitä mukaa, kun ohjelman toteutus eteni. Koska Eneriksellä ei ollut Android-osaamista ennen tämän projektin aloittamista, ei teknisen tason vaatimuksia etukäteen juurikaan osattu kirjoittaa.

### 2.2.1 Arkkitehtuuritason vaatimukset

Karkeat ylemmän tason vaatimukset olivat lähinnä arkkitehtuuriin liittyviä vaatimuksia, kuten että uuden client-ohjelman tulee toimia saman serverin kautta kuin vanhat client-toteutukset, client-ohjelman tulee käyttää samaa protokollaa kuin Qt-ohjelmointikielellä tehty client ja kolmantena kolmansien osapuolien kirjastojen määrä tulee pitää minimissään, koska päätelaitteella on rajalliset resurssit ja kirjastot vievät aina ylimääräisiä resursseja. Android-clienttiä tehdessä näiden vaatimusten täyttäminen ei ollut missään vaiheessa ongelma lukuun ottamatta yhtä kolmannen osapuolen kirjastoa, jonka käyttäminen säästi huomattavan määrän työtunteja ohjelman toteutuksesta.

### 2.2.2 Toiminnalliset vaatimukset

Toiminnalliset vaatimukset kopioitiin käytännössä suoraan alkuperäisen client-ohjelman ja serverin toiminnoista. Näitä vaatimuksia ovat muun muassa käyttäjien hallinta, tuntienkirjaus, projektien hallinta, laajat raportointitoiminnot, yksiköiden hallinta ja bug-track-toiminnallisuus. Aikaisempaan Qt-client-projektiin näistä osista ohjelmalle varattujen resurssien johdosta toteutettiin vain tuntienkirjaus. Tälle Android-projektille varatuista resursseista johtuen päätettiin toteuttaa käyttäjien hallinta, tuntienkirjaus, projektien hallinta, raportointitoiminnot, yksiköiden hallinta sekä offline-toiminnallisuus. Näistä

vaatimuksista offline-toiminto jätettiin myöhemmin pois alkuperäisen suunnitelman puutteiden ja väärinkäsitysten vuoksi, sekä raportointitoiminnot toteutettiin hieman suunniteltua suppeammin. Muun muassa PDF-muotoisten raporttien luominen ja raportin lähettäminen sähköpostin välityksellä halutulle vastaanottajalle ja yksiköidenhallinta-toiminnallisuus jätettiin myöhemmin pois turhana.

### 2.2.3 Tekniset vaatimukset

Alimman tason eli teknisen tason vaatimukset muuttuivat lähes jatkuvasti projektin edetessä. Kun käyttöliittymäsuunnitelmaa alettiin toteuttaa, saattoi huomata, että jotakin asiaa ei kannatakaan tehdä niin kuin se oli suunniteltu, vaan tällöin päätettiin muuttaa jonkin toiminnon ideaa Android-alustalle sopivammaksi. Koko projektissa isona vaatimuksena oli se, että kaikkien toimintojen ja ulkoasun tulisi olla mahdollisimman pitkälti Androidin tyylisääntöjen sanelemassa muodossa. Tämä vaatimus toimi ikään kuin sääntönä ja apuna ohjelman toimintoja koodatessa. Teknisen tason vaatimuksia oli projektin opiskelumaisen luonteen vuoksi helppo muokata tarvittaessa. Tarvitsi vain hyväksyttää haluttu muutos Eneriksen toisen senior-tason työntekijän tai teknologiajohtajan kautta. Isossa roolissa teknisten vaatimusten tulkinnassa ja vaihtoehtoisten toteutusten mietinnässä oli mukana myös Eneriksen ohjelmistojen testaaja.

# 3 ANDROID-ASIAKASOHJELMA

## 3.1 Ohjelman toiminnot

Android-client toteuttaa lähes kaikki alkuperäisen Swing-clientin toteuttamat toiminnot. Edellisen Qt-client-projektin pohjalta Android-clientin suunnittelussa säästettiin todella paljon aikaa. Näin ollen toisin kuin Qt-client projektissa, Android-clientissä toteutettiin huomattavasti enemmän toimintoja kuin Qt-clientissä.

Android-clientin toiminnot jakaantuvat karkeasti neljään luokkaan: tuntien kirjaus, raportointi, käyttäjien hallinta ja projektien hallinta. Näistä luokista käyttäjien ja projektien hallinta ovat saatavilla vain admin-tason käyttäjälle.

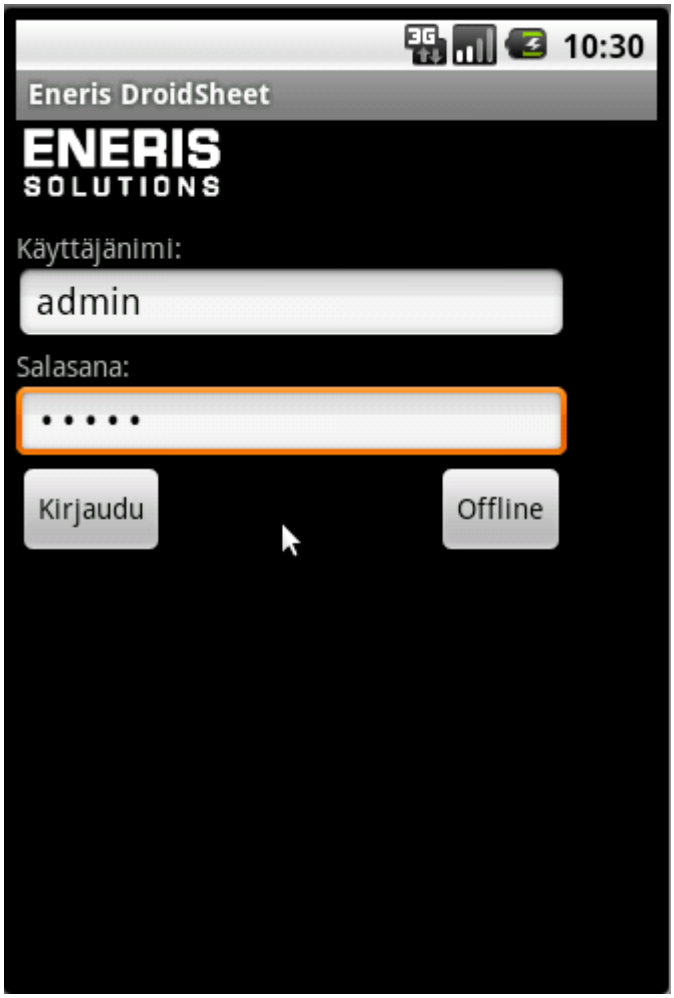
Kaikissa eri toiminnoissa käyttäjää informoidaan toimintojen onnistumisesta ns. Toast-viesteillä. Nämä ovat hetken näytöllä näkyviä tekstikenttiä, joista käyttäjä näkee, onnistuiko hänen juuri tekemänsä toiminto.

### 3.1.1 Normaalin käyttäjän toiminnot

Normaalin käyttäjän toiminnot sisältävät kaksi eri toimintokokonaisuutta: tuntien kirjauksen ja raportoinnin. Tuntien kirjaus sisältää ohjelmiston laajimmin käytetyn osion eli tuntimerkintöjen lisäämisen, poistamisen ja muokkaamisen. Raportointitoiminnossa käyttäjä saa haluamaltaan aikaväliltä raportin merkityistä tunneista.

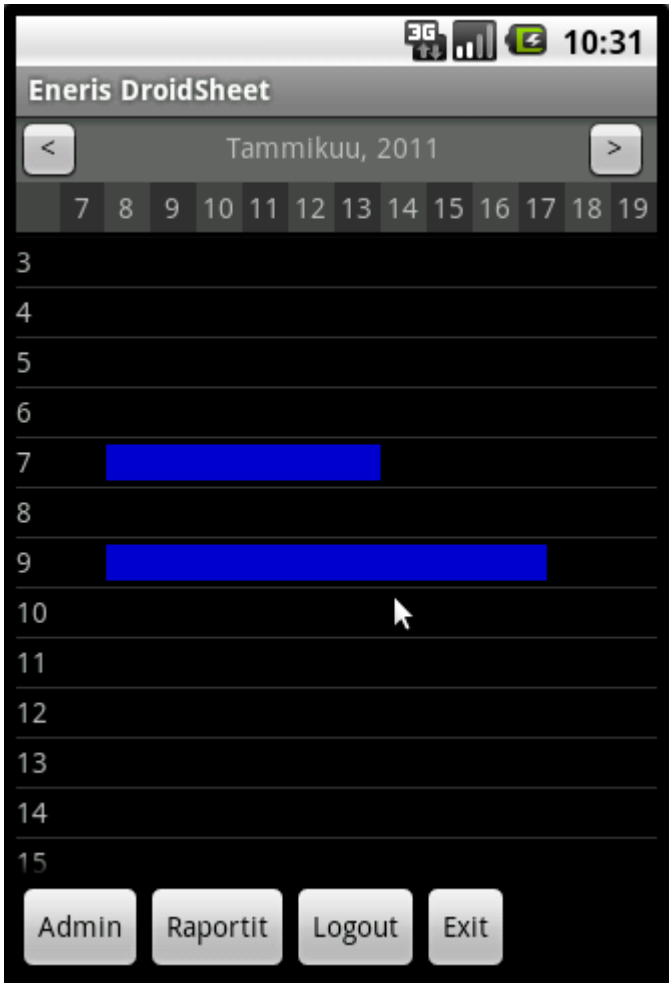
#### 3.1.1.1 Tuntienkirjaustoiminto

Tuntien kirjaus sisältää uusien tuntimerkintöjen lisäämisen, poistamisen ja muokkaamisen halutulle projektille valitulle päivälle. Kun käyttäjä avaa ohjelman, hänelle aukeaa kirjautumisnäkymä (kuva 1).




*KUVA 1. Kirjautumisnäkymä*

Kirjautumisnäkymän jälkeen käyttäjälle näytetään kuukausinäkymä (kuva 2), jossa näkyvät karkeasti kuukaudelle merkityt tunnit päivittäin. Tästä näkymästä voidaan päivää koskettamalla mennä päivänäkymään. Kuukausinäkymän yläpalkissa näkyy valittu kuukausi ja vuosi sekä näiden selausnapit. Itse päivät kuukausinäkymässä ovat yksittäisiä rivejä selattavassa listassa. Kuukausinäkymässä sijaitsevat myös admin- ja raportointitoimintoihin johtavat napit. Admin-nappi ei näy tavalliselle käyttäjälle lainkaan. Tässä näkymässä sijaitsee myös Logout- ja Exit-napit, joilla käyttäjä pääsee kirjautumaan ulos tai sulkemaan ohjelman. Myös puhelimen back-näppäin toimii uloskirjautumisena ja poistumisnappina.

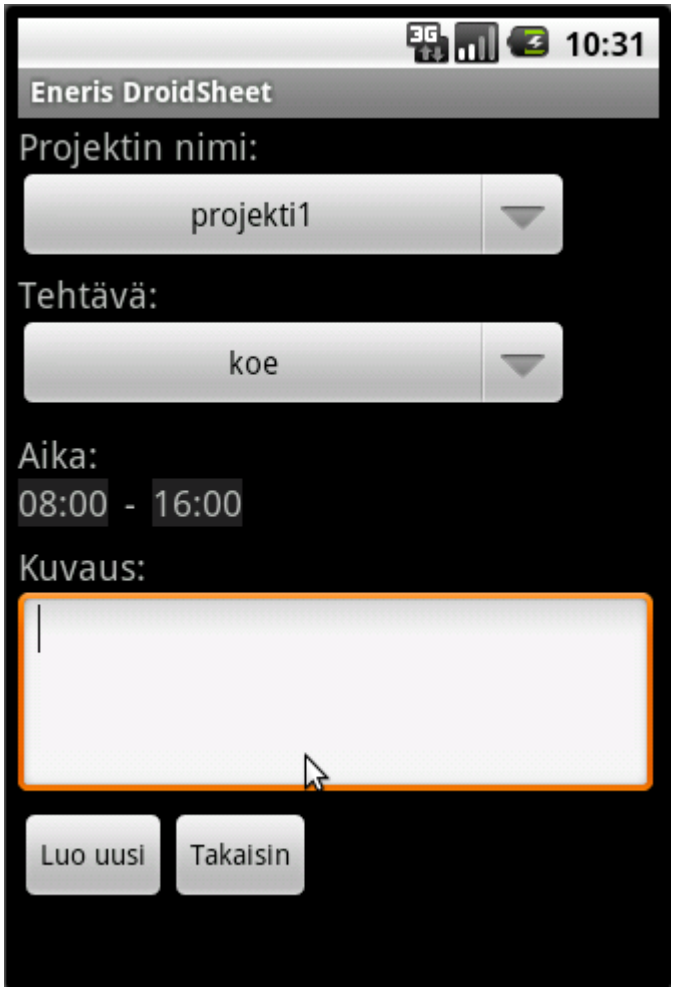



*KUVA 2. Kuukausinäkymä*

Päivänäkymässä (kuva 3) nähdään valitulle päivälle merkityt tuntimerkinnät. Tuntimerkintöjä voi olla useampia samalle päivää. Päivänäkymässä eri tuntimerkintää yläpalkissa koskettamalla nähdään merkinnän tarkemmat tiedot, eli mikä on tarkka aika, mille projektille merkintä on tallennettu sekä mille projektin tehtävälle se on merkitty. Tuntimerkinnät eivät vaadi projektin tehtävän valitsemista, mikäli projektilla ei tehtäviä ole. Päivänäkymässä sijaitsee myös Yhteenveto-taulukko, mistä käyttäjä näkee helposti, paljonko tunteja on merkittynä kyseiselle päivälle, viikolle ja kuukaudelle. Tässä näkymässä sijaitsee myös Lisää, Muokkaa- ja Takaisin-napit. Lisää- ja Muokkaa-napit toimivat käyttäjän valitsemalle tuntimerkinnälle, ja Takaisin-napilla pääsee takaisin kuukausinäkymään.


*KUVA 3. Päivänäkymä*

Tuntimerkinnän muokkaus- ja lisäysnäkymässä (kuva 4) käyttäjä valitsee projektin, projektin tehtävän ja tuntimerkinnän alkamis- ja päättymisajan sekä kirjoittaa kuvauksen merkinnälle. Tämän jälkeen joko lisätään uusi merkintä tai muokataan vanhaa. Takaisin-napin toiminta palauttaa käyttäjän takaisin päivänäkymään.

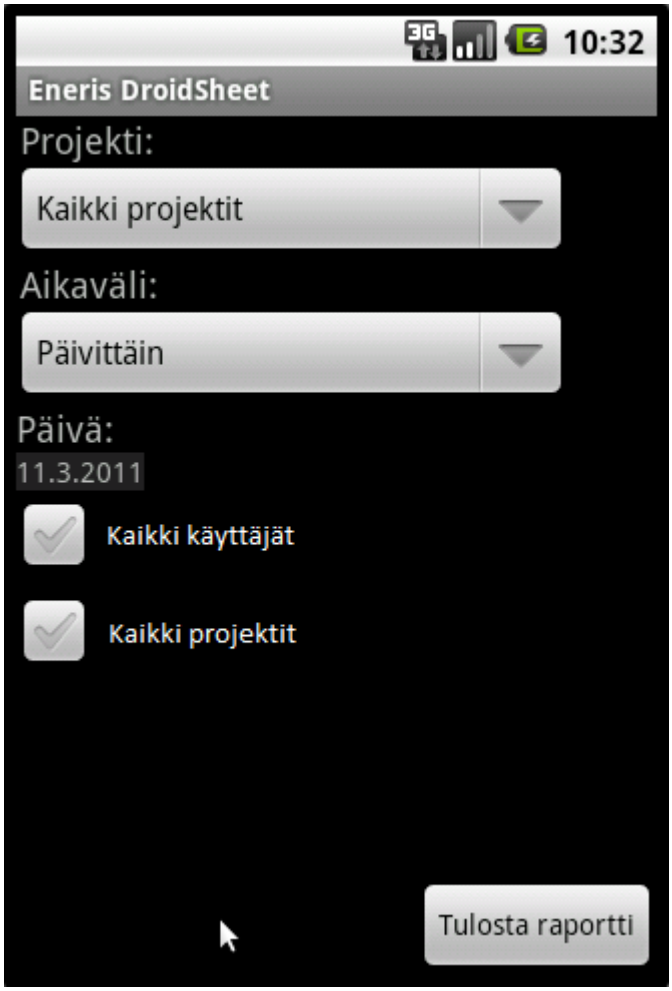



*KUVA 4. Tuntimerkinnän muokkaus- ja lisäysnäkymä*

### 3.1.1.2 Raportointitoiminto

Raportointitoiminnossa käyttäjälle avautuu raportinluontinäkymä (kuva 5), jossa hänen pitää valita projekti, jonka merkinnät halutaan näytettävän raportilla. Tämän jälkeen käyttäjä valitsee raportin tyylin. Eri raportin tyylejä ovat päivittäinen, kuukausittainen ja vapaa-valintainen raportti. Päivittäinen raportti näyttää yhdelle päivälle projektin tunnit. Kuukausittainen näyttää ne valitulle kuukaudelle. Vapaavalintainen raportti antaa käyttäjän itse valita raportin aloitus- ja lopetuspäivämäärät. Raportointinäkymässä on lisäksi kaksi checkbox-tyylistä kontrollia, Kaikki käyttäjät sekä Kaikki projektit. Nämä kaksi toimintoa näkyvät vain admin-tason käyttäjälle ja ovat keskenään poissulkevia. Mikäli katsotaan kaikkien projektien raportti, ei sitä voida näyttää kaikille

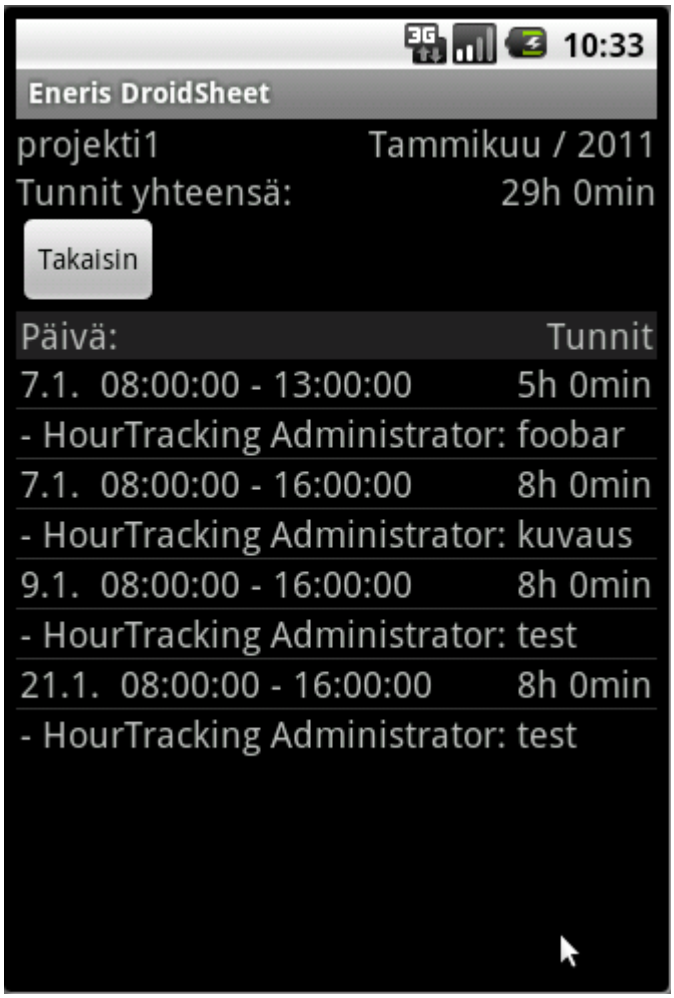
käyttäjille ja toisinpäin. Jos näistä valinnoista näytetään raportti kaikille käyttäjille, se tarkoittaa sitä, että valitusta projektista näytetään kaikki sille merkityt tunnit. Kaikki projektit -valinta tarkoittaa sitä, että näytetään valitun käyttäjän kaikki merkinnät aikaväliltä kaikille projekteille.




*KUVA 5. Raportinluontinäkymä*

Kun raportin luonti on valmis, käyttäjä painaa Tulosta raportti -nappia, jolloin raportti tulostetaan näytölle. Raporttinäkymä (kuva 6) koostuu kahdesta osasta: Yläpalkissa näkyy raportoitavan projektin nimi, raportin aikaväli sekä yhteen laskettu tuntimäärä valitulta ajalta. Yläpalkin alapuolella sijaitsee listanäkymä, jossa listataan jokainen yksittäinen tuntimerkintä valitulle ajalle. Jokainen merkintä vie kaksi riviä listasta. Ylemmällä rivillä sijaitsee merkinnän

päivämäärä, aika ja merkinnän pituus tunteina ja minuutteina. Alemmalla rivillä näkyy merkinnän tekijä ja merkinnän kuvaus.




*KUVA 6. Raporttinäkymä*

### 3.1.2 Admin-tason toiminnot

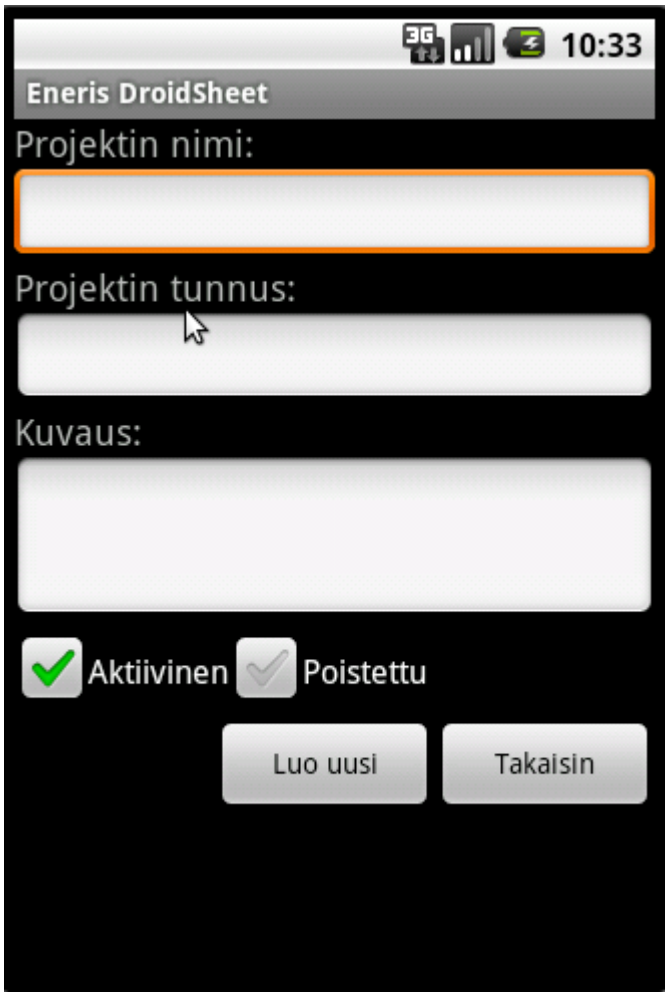
Admin-tason toiminnot sisältävät normaalin käyttäjän toimintojen lisäksi käyttäjien ja projektien hallinnan. Näihin toimintoihin päästään käsiksi kuukausinäkymästä, mikäli käyttäjällä on admin-tason oikeudet. Käyttäjien ja projektien hallinta sijaitsevat näkymässä, jossa on kaksi eri välilehteä. Oletuksena projektien hallinta -välilehti on aktiivisena.

### 3.1.2.1 Projektien hallinta

Projektit-sivulla (kuva 7) on neljä painonappia: Lisää projekti, Muokkaa projektia, Lisää tehtävä sekä Muokkaa tehtävää. Näiden nappien alapuolella sijaitsee spinner-lista, jossa on kaikki projektit, mitä serveriltä löytyy. Kun jokin projekti on valittu, näkyvät projektin tehtävät listana spinner-listan alapuolella. Käyttäjä voi valita jonkin näistä tehtävistä, jolloin Muokkaa tehtävää -nappi aktivoituu. Lisää projekti- ja Muokkaa projektia -napit avaavat projektin lisäys- ja muokkaussivun, kun taas Lisää tehtävä- ja Muokkaa tehtävää -napit avaavat tehtävän lisäys- ja muokkaussivun.


*KUVA 7. Projektienhallintanäkymä*

Projektin lisäys- ja muokkausnäkymässä (kuva 8) on kolme tekstikenttää: Projektin nimi, Projektin tunnus ja Kuvaus. Näiden tekstikenttien alapuolella on kaksi checkbox-tyylistä kontrollia, Aktiivinen ja Poistettu. Nämä toiminnot eivät ole toisiaan poissulkevia kahteen suuntaan. Poistettu projekti ei voi olla aktiivinen eikä näy käyttäjille normaalissa projektilistauksessa tuntimerkintöjä lisätessä. Mikäli projekti ei ole aktiivinen, se kuitenkin näkyy käyttäjille, mutta sille ei voi lisätä merkintöjä. Näkymässä on lisäksi kaksi nappia: Luo uusi ja Takaisin. Luo uusi -nappi muuttuu Tallenna-napiksi, mikäli valittuna on projektin muokkaaminen eikä lisääminen. Toisin kuin käyttäjien muokkaamisessa, projekteja ei voi poistaa. Ne voidaan vain merkitä poistetuiksi.




*KUVA 8. Projektin lisäys- ja muokkausnäkymä*

Projektin tehtävän lisäys- ja muokkausnäkymä (kuva 9) poikkeaa projektin lisäys- ja muokkausnäkymästä vain siten, että tehtävänäkymästä puuttuu Projektin tunnus -tekstikenttä sekä Aktiivinen- ja Poistettu-checkbox-kontrollit.


*KUVA 9. Projektin tehtävien lisäys- ja muokkausnäkymä*

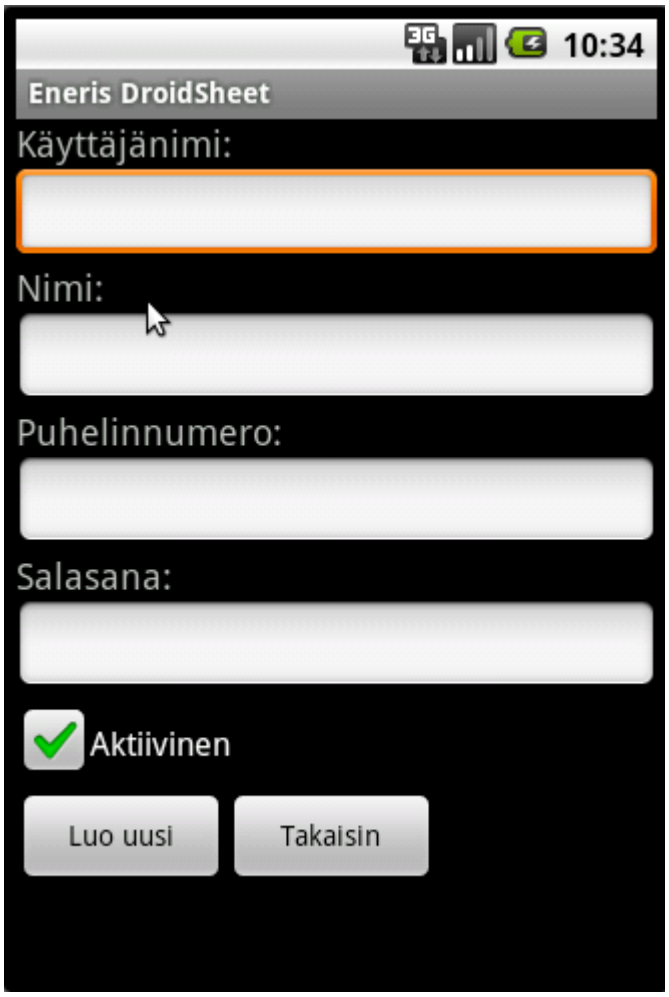
### 3.1.2.2 Käyttäjien hallinta

Käyttäjien hallinta sijaitsee admin-toimintojen toisella välilehdellä. Käyttäjien hallinnan päänäkymä (kuva 10) sisältää neljä painonappia: Lisää, Muokkaa, Projektit ja Tunnit. Näiden painonappien alapuolella on listaus kaikista käyttäjistä. Mikäli listassa ei ole käyttäjää valittuna, ovat Muokkaa-, Projektit- ja Tunnit-napit tummennettuna ja poissa käytöstä.


*KUVA 10. Käyttäjienhallinnan päänäkymä*

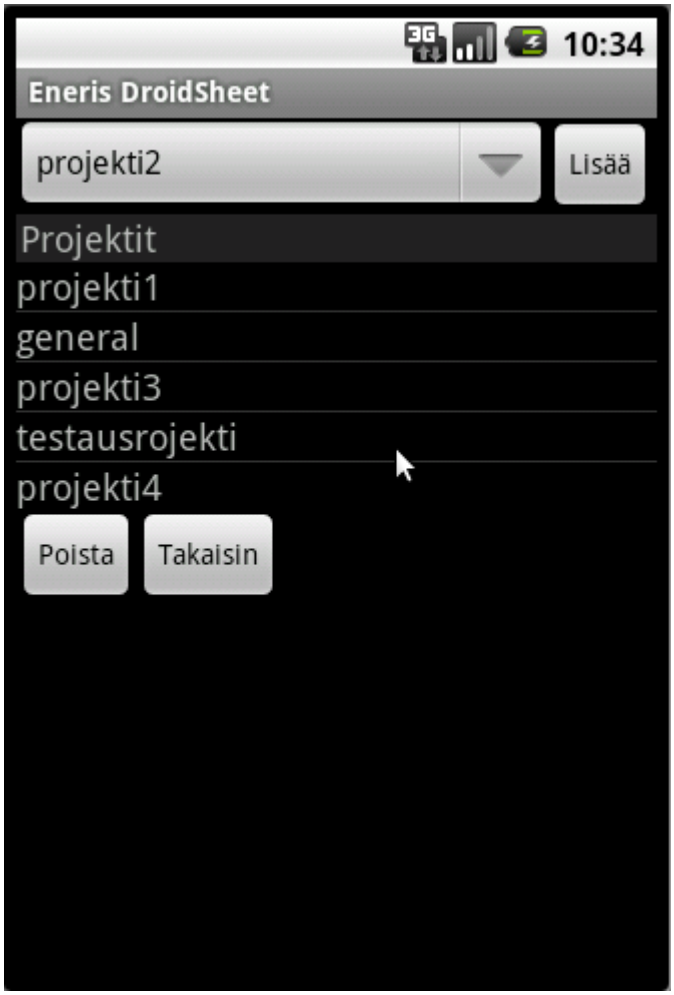
Käyttäjien muokkaus- ja lisäysnäkymässä (kuva 11) on neljä tekstikenttää: käyttäjätunnus, käyttäjän nimi, puhelinnumero ja salasana. Tekstikentistä salasanakenttä on toisista kentistä poikkeava. Käyttäjän muokkauksessa salasanakenttää ei täydennetä vanhalla salasanalla ollenkaan, vaan mikäli käyttäjä haluaa vaihtaa salasanan, tulee hänen kirjoittaa uusi salasana tähän kenttään. Mikäli salasanaa ei haluta vaihtaa, tulee kentän jäädä tyhjäksi. Kyseessä on myös ns. salasanakenttä, eli kirjaimet näkyvät käyttäjälle palloina, jolloin kukaan ei voi selän takaa lukea käyttäjän kirjoittamaa salasanaa. Näiden tekstikenttien alapuolella on yksi checkbox, Aktiivinen-toiminto. Aktiivinen-toiminto määrää sen, voiko käyttäjä merkitä tunteja ja ylipäänsä kirjautua järjestelmään sisään. Aktiivinen-toiminnon alapuolella sijaitsee kaksi painonappia: Luo uusi ja Takaisin. Luo uusi -nappi muuttuu Tallenna-napiksi,

jos muokataan vanhaa käyttäjää uuden lisäämisen sijasta. Muokkausnäkymässä on myös lisäksi Poista-nappi.



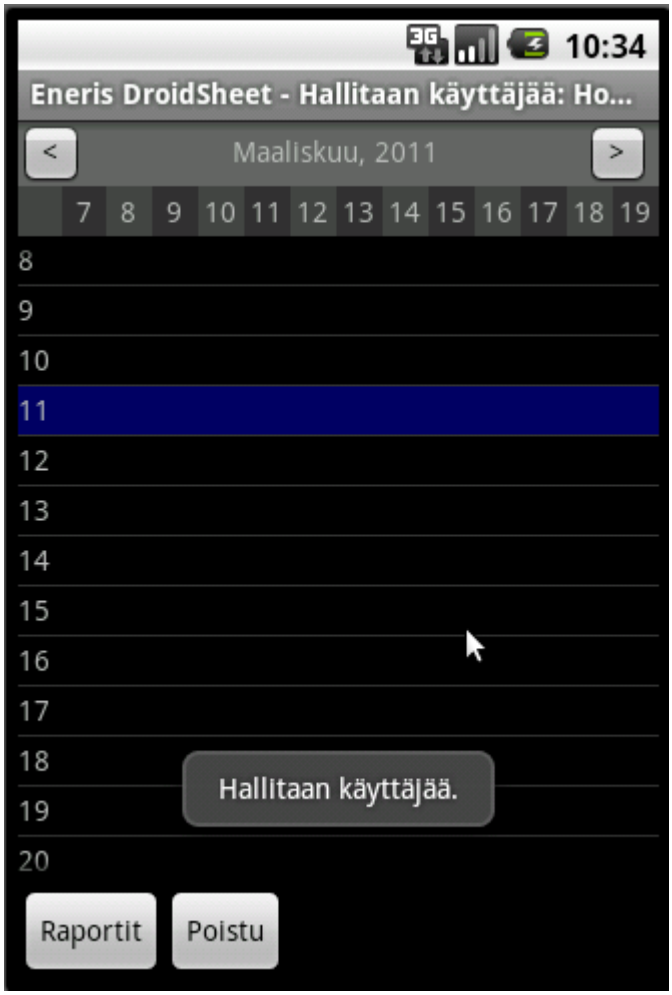

*KUVA 11. Käyttäjän lisäys- ja muokkausnäkymä*

Käyttäjien hallinnan pääsivulla oleva Projektit-nappi avaa näkymän, jossa käyttäjille yhdistetään projekteja (kuva 12). Tässä näkymässä ylimmäisenä on spinner-lista, jossa on listattuna kaikki sillä hetkellä olevat projektit, joita ei ole merkitty poistetuksi. Tämän spinner-listan vieressä on Lisää-nappi, joka yhdistää valitun projektin käyttäjälle jota juuri hallinnoidaan. Projektit spinner-listan alapuolella on tavallinen lista, jossa on listattuna käyttäjälle jo yhdistetyt projektit. Tämän listan alapuolella on kaksi painonappia, Poista ja Takaisin. Poista-nappi poistaa valitun projektin käyttäjältä, ja Takaisin-nappi päästää käyttäjän takaisin käyttäjien hallinnan päänäkymään.




*KUVA 12. Projektien yhdistäminen käyttäjälle*

Viimeisenä toimintona käyttäjien hallinnassa on päänäkymässä oleva Tunnitnappi. Tämä nappi antaa admin-tason käyttäjälle valitun käyttäjän näkemän näkymän (kuva 13), eli admin-käyttäjä hallitsee valittua käyttäjää ja voi merkitä tälle tunteja. Käyttäjän tunnit -näkymä ei poikkea normaalista kuukausi-, päivä- ja tuntimerkintänäkymästä muuten kuin siten, että ohjelman yläpalkissa näkyvän ohjelman nimen perässä näkyy myös, ketä käyttäjää sillä hetkellä hallinnoidaan.




*KUVA 13. Käyttäjän tunnit -näkymä. Näkymässä mukana toast-viesti kertomassa toiminnon onnistumisesta*

## 3.2 Arkkitehtuuri

### 3.2.1 Aktiviteetit

Android-ohjelman arkkitehtuuri perustuu aktiviteetteihin. Yleensä yksi aktiviteetti kuvaa yhtä käyttäjälle näkyvää näyttöä. Tässä projektissa on 14 eri aktiviteettia, joista useaa aktiviteettia on käytetty hyödyksi useampaan kertaan eri toimintoja toteutettaessa. Aktiviteettien uudelleenkäytöstä hyvänä esimerkkinä on muun muassa projektin muokkaus ja lisäys, käyttäjien muokkaus ja lisäys sekä projektin tehtävien muokkaus ja lisäys. Sen sijaan että näitä toimintoja varten olisi tehty jokaiselle muokkaus- ja lisäysparille oma aktiviteetti, nopealla ja

helpolla työllä ohjelman sai käyttämään samaa aktiviteettia molemmille toiminnoille. Toinen iso ajansäästö tuli admin-tason käyttäjän tunnit -toiminnosta, jossa sen sijaan että kuukausi-, päivä- ja tuntienkirjausnäkymät olisi koodattu toiseen kertaan, käytetään samoja aktiviteetteja normaalin käyttäjän toiminnoissa sekä admin-tason käyttäjän tunnit -toiminnoissa.

Jokaista aktiviteettia kohden on olemassa yleensä yksi käyttöliittymätiedosto. Käyttöliittymätiedostoja voi olla myös useampia yhdelle aktiviteetille, mutta käyttöliittymien vaihto lennosta ei ole suositeltavaa ohjelmointityyliä, sillä se aiheuttaa ennalta arvaamattomia ongelmia ajon aikana. Java-ohjelmoinnin periaatteiden mukaan jokainen aktiviteetti on oma luokkansa ja näin ollen sijaitsee omassa .java-päätteisessä ohjelmakooditiedostossa.

### 3.2.2 Käyttöliittymät

Tässä projektissa seurattiin Android-ohjelmoinnin tyylisääntöjä mahdollisimman hyvin, mistä johtuen käyttöliittymä- ja ohjelmakoodi sijaitsevat erillään toisista. Käyttöliittymäkoodi sijaitsee Android xml -tiedostoissa, ja se seuraa XML-standardin merkkaustapaa. Jokaisessa Android xml -tiedostossa mainitaan kyseisen XML-tiedoston käyttämä schema. Käytännössä projekti eteni käyttöliittymä- ja aktiviteettipareja tehden. Projektissa tein jokaiselle aktiviteetille oman käyttöliittymän, joka oli tehty vain kyseiselle aktiviteetille.

### 3.2.3 Managerit

Päädyin kokeilunhalusta testaamaan, miten manageri-mallinen ratkaisu toimii tämän kaltaisessa projektissa. Otin pohjaksi sellaisen idean, että jokainen iso kokonaisuutensa tarvitsee oman manageri-luokkansa. Lopulta managereita tuli vain muutama, mutta itse malli kuitenkin toimii loistavasti tämän kaltaisessa projektissa. Managereita ei tullut omaansa jokaiselle asialle, koska aktiviteettimallilla joitakin asioita ei ollut mitään järkeä toteuttaa omassa managerissaan, kun ne oli helppo toteuttaa niitä tarvitsevissa aktiviteeteissaan.

Tärkeimmät managerit tässä projektissa olivat NetworkManager, ContentManager ja JSON serializer/deserializer.

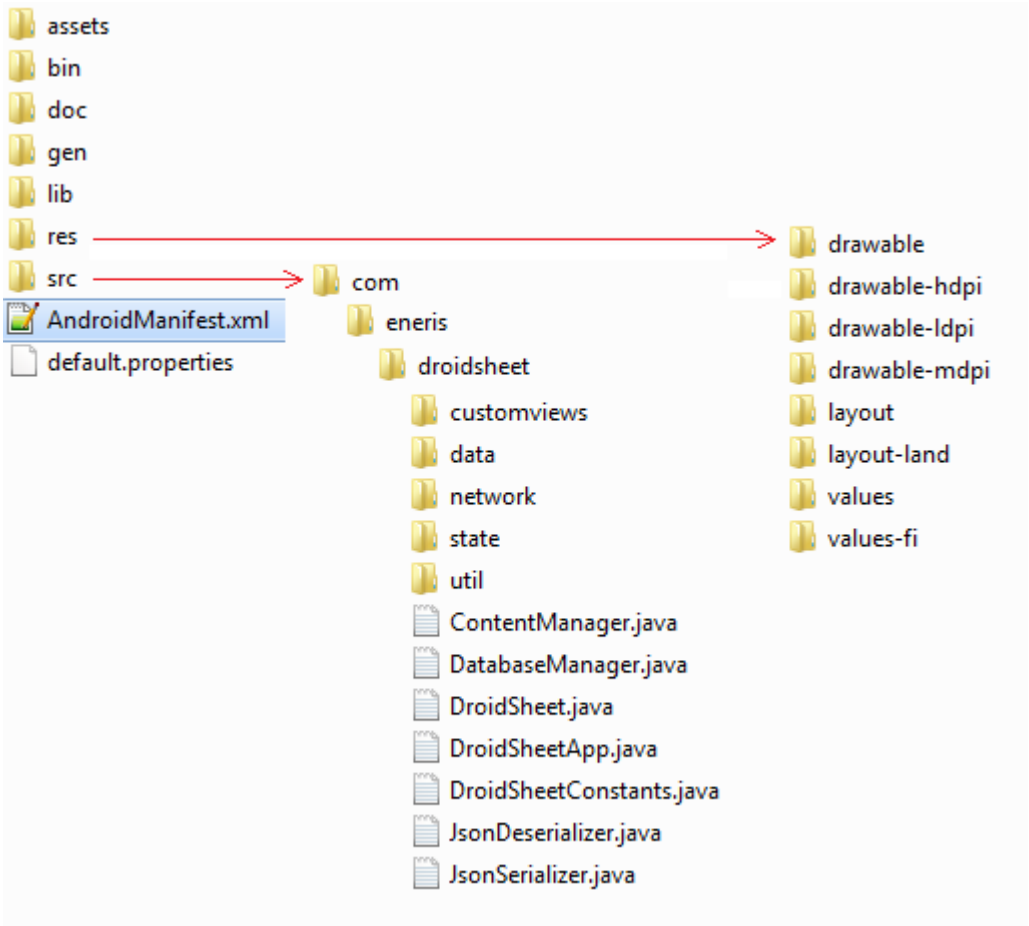
NetworkManager ja ContentManager toimivat tiiviissä yhteistyössä toistensa kanssa. NetworkManager hoitaa kaiken verkkoyhteyden serverin ja Android-clientin välillä ja toteuttaa alustariippumattoman protokollan toteutuksen. Kun NetworkManager saa jotain dataa serveriltä, se lähettää saadun viestin sellaisenaan ContentManagerille, joka parseroi viestin, käsittelee parseroidun datan ja tämän jälkeen tallentaa sen tietovarastoonsa. Jokainen eri aktiviteetti sisältää pääsyn NetworkManageriin ja ContentManageriin. Sen sijaan yksikään aktiviteetti ei tarvitse pääsyä JSON serializer/deserializer -luokkiin.

JSON serializer/deserializer -luokat ovat yksinkertaisia JSON-viestien pakkaus- ja parserointiluokkia. Nämät luokat ovat tärkeitä juuri alustariippumattoman protokollan toteutusta varten. JSON serializer/deserializer -luokat käyttävät projektissa ainoaa mukana olevaa kolmannen osapuolen kirjastoa, Flexjson-kirjastoa.

### 3.2.4 Kansiorakenne

Ohjelman resurssit sijaitsevat käyttöliittymätiedostojen ohella projektin luoman res-kansion alla. Res-kansiossa sijaitsee useampia alakansioita, kuten drawable, drawable-hdpi, drawable-lpdi, layout, layout-land, values, values-fi. Näistä kansioista drawable-alkuiset kansiot sisältävät ohjelman käyttämän grafiikan. Perus-drawable-kansio sisältää normaalit kuvat, ja hdpi-, ldpi- ja mdpi-kansiot sisältävät eri näyttötarkkuuksille tarkoitetut kuvat, mikäli käyttäjä haluaa tehdä omat kuvat erikokoisille näytöille. Layout-kansiossa sijaitsee ohjelman portrait-moodin käyttöliittymät ja layout-land kansiossa landscape-moodin käyttöliittymät. values- ja values-fi-kansioissa sijaitsee ohjelman lokalisaatio (kuva 14).

Ohjelman koodi sijaitsee src-kansion alla. Src-kansiossa jokaiselle projektin paketille (package) on oma kansiorakenteensa. Packaget ovat Javasta peräisin oleva idea, jossa samankaltaista ohjelmakoodia tai saman asiayhteyden omaavat tiedostot voidaan sijoittaa yhteiseen pakettiin. Käytännössä packageilla on merkitystä vain siinä mielessä, että käyttäjän on helpompi hahmottaa useita ohjelmakooditiedostoja sisältävän projektin kokonaisuus. Mikäli tiedostoja on useita satoja, voisi projektin selaaminen olla miltei mahdotonta ilman packageja. Hyvä esimerkki siitä, miten packageja käytetään tässä projektissa, on se, että ohjelmakooditiedosto on jaoteltu niin, että aktiviteetit ovat omassa packagessaan, managerit omassaan, dataluokat omassaan ja kantaoliot omassaan.




*KUVA 14. Android-client-projektin kansiorakenne*

### 3.2.5 Android manifest -tiedosto

Android-projektin manifest-tiedosto sisältää tiedon kaikista ohjelman käyttämistä aktiviteeteista, oikeuksista ja kohdealustan tiedoista. Androidin manifest-tiedosto on XML-formaatissa. Uutta projektia luodessa Eclipse IDE luo myös tarvittavan XML-tiedoston, mutta aktiviteetit, kohdealustan tiedot ja oikeudet on itse lisättävä sinne. Eclipsessä on mukana Androidin XML-editori, jonka avulla käyttäjä voi muokata manifest-tiedoston sisältöä, vaikka ei XML-merkkausta osaisikaan.

Aktiviteettien lisäys manifest-tiedostoon onnistuu seuraavanlaisella rivillä:

```
<activity android:name=".state.DayViewState"></activity>
```

Tällä rivillä android:name-määre viittaa aktiviteetin toteuttavan luokan pakettiin ja nimeen. Käyttäjän oikeuksien määrittäminen onnistuu rivillä

```
<uses-permission
android:name="android.permission.INTERNET"/>
```

Tällä rivillä ohjelma kertoo käyttöjärjestelmälle, että ohjelma tarvitsee oikeuden Internet-yhteyden käyttöön. Kun käyttäjä asentaa ohjelmaa, käyttöjärjestelmä kysyy häneltä, sallitaanko ohjelmalle lupa käyttää kaikkia sen pyytämiä oikeuksia. Kohdealustan versiointitiedot ilmoitetaan manifest-tiedostossa seuraavanlaisella rivillä:

```
<uses-sdk android:minSdkVersion="7" />
```

Tässä esimerkissä minSdkVersion kertoo sen, että ohjelma vaatii version 2.1 Android-alustasta toimiakseen. Viimeiseksi manifest-tiedosto kertoo ohjelman version ja version nimen. Tämä on tärkeä tieto siinä mielessä, että mikäli ohjelmasta tehdään uusi versio ja versionumeroa ei kasvateta, ei Android-alusta anna päivittää ohjelmaa vaan pakottaa asentamaan uuden kopion ohjelmasta

laitteeseen. Tämä ei ole haluttua toimintaa, joten versionumeron päivittäminen on tärkeä muistaa ohjelmistosta uutta versiota tehtäessä. Versionumeron määrittäminen manifest-tiedostossa tapahtuu seuraavanlaisilla määreillä:

```
android:versionCode="5"
android:versionName="1.05"
```

Näillä riveillä versionCode on numero, joka kuvaa kyseistä versiota. versionName tarkoittaa käyttäjän itse valitsemaa nimeä versiolle.

# 4 SERVER/CLIENT-PROTOKOLLA

## 4.1 Alkuperäinen protokolla

Alkuperäinen protokolla Java server / swing-client-toteutuksessa käytti yksinkertaista Java-luokkien serialisointia HTTP-yhteyden yli. Jokainen paketti lähetti autentikaatioon käytettävän hash-arvon (auth-hash) sekä ennalta tuntemattoman määrän serialisoituja Java-luokkia. Tämän Javan serialisointityylin johdosta oli huomattavasti yksinkertaisempaa suunnitella ja toteuttaa kokonaan uusi protokolla kuin tehdä vanhasta yhteensopiva uusien client-alustojen kanssa. Samalla uudesta protokollasta pystyttiin myös tekemään sellainen, jonka eri ohjelmointikielet ja eri alustat pystyvät alustariippumattomasti toteuttamaan. Näin serverille tuli vain kaksi eri protokollaa sen sijasta, että jokaisella päätelaitteella olisi oma.

Vanhan protokollan paketit sisältävät neljä eri muuttujaa: auth-hash, tieto pyynnön onnistumisesta, paketin ohjaustieto ja paketin data. Tieto onnistumisesta on yksinkertaisesti SUCCESS- tai FAILED-arvo, joka on tallennettuna enum-muuttujaan. Seuraavana oleva paketin ohjaustieto on niin ikään tallennettu enum-muuttujaan. Toisin kuin onnistumisen kertovassa muuttujassa, ohjaustiedon enumissa on lukuisia eri arvoja kentälle. Nämä ohjauskentän arvot ohjaavat paketin oikealle käsittelijälle, kun serveri alkaa käsitellä pakettia. Käytännössä kaikki verkkoliikenne serverin ja asiakasohjelman välillä omistaa yhden oman tiedon tässä ohjaustieto-enumissa. Muutamia esimerkkejä ohjaustietueessa olevista arvoista ovat esimerkiksi kentät LIST_TIMERANGES_BY_DAY, LIST_PROJECTS_BY_USER ja SAVE_TIME_RANGE, jotka nimensä mukaisesti pyytävät serveriä lähettämään pyydettävää dataa tai muokkaamaan vanhaa dataa.

Datakenttä sisältää serialisoituja Java-objekteja. Nämä Java-objektit voivat olla käytännössä mitä tahansa Java-luokkia, jotka toteuttavat java.io.Serializable-

rajapinnan ja sisältävät vähintään yhden muuttujan sekä getter/setter-parit jokaiselle muuttujalle. Kyseessä on siis yksinkertainen tieto-oliotyylinen Java-luokka. Myös Javan primitiivisiä tietotyyppejä int/double/long/.. voidaan serialisoida, mikäli ne ovat omien wrapper-luokkaolioidensa sisällä (Integer/Double/Long/…).

## 4.2 Alustariippumaton protokolla

Alustariippumatonta protokollaa suunniteltaessa tuli ottaa huomioon, että eri alustoilla on käytössä eri ohjelmointikieliä. Alustariippumaton protokolla tuli olla toteutettu dataformaatilla, jota jokainen mahdollinen kieli pystyy lukemaan ja kirjoittamaan. Kun jokainen ohjelmointikieli tukee valittua dataformaattia, on jokaiselle alustalle melko varmasti olemassa vähintään yksi kolmannen osapuolen kirjoittama avoimeen lähdekoodiin perustuva kirjasto, jota voi hyödyntää protokollan toteutuksessa.

Projektissa käytetty protokolla suunniteltiin alun perin kesällä 2010 Nokian N900-puhelimelle Maemo/Qt-alustalle tehdylle pääteohjelmalle, joka oli ideapohjalta tämän projektin edeltäjä. Tuossa Qt-client-projektissa tutkittiin useita eri mahdollisuuksia protokollan toteutukseen ja lopulta päädyttiin siihen lopputulokseen, että datan muoto on sopivin projektien vaatimukset huomioon ottaen JSON-dataformaatissa.

Uuden protokollan pakettien rakenne on hyvin samankaltainen kuin vanhan alustariippuvaisen protokollan paketit olivat. Suurin ja tärkein ero on kuitenkin se, että paketin mukana olevaa dataa ei serialisoida Javan java.io.Serializable-rajapintaa hyödyntäen, vaan data serialisoidaan JSON-formaattiin. Tästä johtuen dataa ei myöskään enää voitu lähettää päätelaitteelle samalla lailla kuin ennen. Uuden protokollan myötä paketti kasataan taulukkoon, joka koostuu tavuista. Kun koko taulukko on kasattu sisältöineen kasaan, taulukko lähetetään vastauspakettina päätelaitteelle, joka lähetti pyynnön. Pakettien lähetyksessä ja vastaanotossa suoritetaan tietyntasoinen datan kryptaus, jolla muuten

selväkielinen JSON-muotoinen data saatetaan muotoon, jossa sitä ei voi lukea. Vaikka tämä datan kryptaus on suhteellisen helppo purkaa, ohjelman laadun ja pienuuden huomioon ottaen se ei ole minkään tasoinen riski.

Toinen ero vanhaan protokollaan on pyynnön onnistumisarvon puuttuminen kokonaan. Tämä toiminto on korvattu siten, että paketin mukana tuleva JSON-muotoinen data on NULL-arvo (nolla), mikäli toiminto epäonnistui. Joihinkin clientin pyyntöihin NULL-arvo on hyväksyttävä vastaus serveriltä, koska näissä pyynnöissä client ei edes oleta serverin vastaavan, tai kyseisen toiminnon epäonnistuminen ei vaikuta clientin toimintaan mitenkään eikä vaadi pyynnön uudelleenlähettämistä serverille.

HTTP request/response -ideologian johdosta server/client-yhteys on tilaton. Tästä johtuen jokaiseen clientiltä tulevaan kutsuun vastataan, sillä serveri ei voi itse lähettää paketteja clientille muuta kuin vastauksena clientin lähettämään pakettiin. HTTP-kutsujen käyttö tämänkaltaisessa arkkitehtuurissa on siinä mielessä iso este, että se estää muun muassa täysin saman datan hallinnoimisen yhtä aikaa useammalta clientiltä. Myös pakettien autentikaatio ja tietoturva on astetta heikommalla tasolla kuin se olisi tilallisessa socket-yhteydessä serverin ja clientin välillä. Vanhassa protokollassa on myös sama ongelma, mutta päätelaitteiden ohjelmien arkkitehtuurin vuoksi tilallinen socket-yhteys ei olisi hyvä ratkaisu muutenkaan. Näin ollen edellä mainituista heikkouksista huolimatta tilaton HTTP-yhteys oli tässä tapauksessa paras vaihtoehto.

# 5 TIMESHEET SERVER

## 5.1 Arkkitehtuuri

### 5.1.1 Servletit

TimeSheet Plus Server (server) on Apache Tomcat web serverillä pyörivä ohjelmisto, joka palvelee kaikkia TimeSheet-client-ohjelmia. Serverin toiminnot perustuvat kahteen servlettiin, jotka toteuttavat kaksi käytössä olevaa protokollaa. Servletit ovat serverillä olevia moduuleja, jotka kuuntelevat niille ohjattua HTTP-liikennettä. Kun client-ohjelma lähettää HTTP-pyynnön serverille, servletti purkaa tulleen paketin haluttuun muotoon, toteuttaa paketissa tulleen pyynnön, kasaa vastauksen ja lähettää sen clientille.

Ensimmäinen servletti on ohjelman alkuperäinen servletti, joka palvelee ainoastaan Swing-clienttiä. Koska pakettien käsittely on syvälle integroitu servlettien toimintoihin, yhden servletin ei ollut mahdollista palvella kahta eri protokollaa. Näin ollen toinen servletti on tehty palvelemaan alustariippumattomia yhteyksiä. Tämä servletti tehtiin alun perin palvelemaan Qt-clientin määrittelemää alustariippumatonta protokollaa, mutta myöhemmin kun Android-client-projekti aloitettiin, uudempi servletti palvelee myös sitä.

### 5.1.2 Actionit

Kun servletti saa paketin clientiltä ja on purkanut tämän paketin, on servletillä tiedossa paketin mukana tullut header-tieto. Tämän tiedon avulla servlet etsii oikean Action-toiminnon, jonka suorittamalla se saa clientin haluaman vastauksen pyyntöön. Actionit ovat luokkia, jotka peritään AbstractAction-kantaluokasta. Perimisen vuoksi Actioneiden käsittely on yksinkertaisempaa, kuin jos ne olisi kaikki omia irrallisia luokkiaan. AbstractAction-kantaluokassa sijaitsee kaikki serverin tietokannan ja datan käsittely.

## 5.2 Projektispesifiset muutokset serverille

Qt-clienttiä varten serverille piti tehdä toinen servletti toteuttamaan alustariippumaton protokolla. Tuossa projektissa luotiin toinen servletti, muutamia uusia Actioneita ja JSON serializer ja deserializer. Android-projektissa tehtiin huomattava määrä Actioneita lisää ja erilaisten tietokantahakujen määrää lisättiin.

# 6 KÄYTETYT TEKNOLOGIAT

## 6.1 Android kohdealustana

Android on avoimeen lähdekoodiin perustuva ohjelmistokokoelma, joka sisältää käyttöjärjestelmän, keskitason ohjelmistoa ja avainohjelmia. Androidia kehitti alun perin Android Inc. Vuonna 2005 Google Inc. osti Android Inc:n ja jatkoi Androidin kehittämistä. Androidin käyttöjärjestelmä perustuu muokattuun versioon Linux kernelistä. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

Androidin kehittäminen aloitettiin vuonna 2003, kun Andry Rubin, Rich Miner ja muutama muu avainhenkilö perustivat Android Inc:n luodakseen paremman alustan älypuhelimia varten. Vuonna 2005 Google osti Android Inc:n ja teki siitä täysin Googlen omistaman tytäryhtiön. Android Inc:n avainhenkilöt pysyivät yrityksen palveluksessa myös yrityksen ostamisen jälkeen. Oston aikoihin Android Inc:stä ei vielä tiedetty kovin paljoa, ja yleisesti luultiin, että Google halusi vain mukaan älypuhelinten ohjelmistokehitykseen. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

Vuonna 2006 spekulaatiot Googlen aikomuksista hypätä mukaan mobiililaitteiden markkinoille jatkuivat. Vuonna 2007 InformationWeek-lehti julkisti artikkelin, jonka mukaan Google olisi hankkimassa useita patentteja mobiilipuhelinten alalta. Myöhemmin vuonna 2007 julkistettiin useiden yritysten liittouma Open Handset Alliance. Tähän liittoumaan kuului muun muassa Broadcom Corporation, Google, HTC, Intel, LG, Marvell Technology Group, Motorola, Nvidia, Qualcomm ja Samsung Electronics. Open Handset Alliancen tarkoitus oli kehittää avoimia standardeja mobiililaitteille. Samana päivänä Open Handset Alliance myös julkisti ensimmäisen tuotteensa, Androidin, joka oli Linuxin 2.6-kerneliä käyttävä mobiililaitteille rakennettu alusta. Vuoden 2008 lokakuussa Google julkisti koko Androidin lähdekoodin Apache Lisenssin alla.

Saman vuoden joulukuussa 14 uutta jäsenyritystä liittyi Open Handset Allianceen. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

Riippumaton teknologian analysointiyritys Canalys raportoi 2010 vuoden viimeisen neljänneksen älypuhelinten käyttöjärjestelmien myynnin prosenttiosuuksista. Tämän raportin mukaan Android-alusta pudotti Symbianin pois 10 vuoden mittaiselta ykköspaikalta. Android omistaa nyt 31,2 % Yhdysvaltojen älypuhelinmarkkinoista. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

Näissä parissa vuodessa, jona Android-alusta on ollut olemassa, se on saavuttanut valtavan ison osan älypuhelinmarkkinoista. Mikäli tämä tahti jatkuu, Androidista on tulossa lähitulevaisuudessa suurin älypuhelimissa käytetty alusta. Asiaa auttaa myös se, että Androidilla on todella laaja ohjelmistokehittäjien joukko, jotka tekevät ohjelmia Android-alustalle myytäväksi Googlen Android Marketissa. Suuri ohjelmien määrä vain lisää Androidin suosiota. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

## 6.2 Java

Java on ohjelmointikieli, jonka alun perin kehitti James Gosling Sun Microsystemsillä. Nykyään Javan omistaa Oracle Corporation. Javan syntaksi on tehty C/C++:n pohjalta, mutta sillä on yksinkertaisempi oliorakenne. Java-kääntäjät yleensä kääntävät Java-ohjelmat tavukoodi-muotoon, jota Javan virtuaalikone (Java Virtual Machine) osaa tulkata. Javan tavukoodi on alustariippumatonta, joten jokaisella alustalla oleva Javan virtuaalikone osaa tulkata Javan tavukoodia, vaikka se olisi eri alustalla käännetty. Java on laajasti käytetty ohjelmointikieli, erityisesti web-ohjelmistoissa. (1, hakusanalla Java programming language, hakupäivä 19.3.2011.)

Käytännössä Android-alustalle ohjelmoidaan samaa Javaa kuin normaali Java on. Erona näiden kahden välillä on muun muassa niiden käyttämät

virtuaalikoneet. Androidin Java-kirjastot ovat Googlen tekemiä kirjastoja, kun taas standardi-Javan käytetyt kirjastot ovat yleensä Sun Microsystemsin tekemiä. Java Virtual Machine (JVM) ja Dalvik Virtual Machine (DalvikVM) eroavat toisistaan käytännössä vain siinä mielessä, että DalvikVM on optimoitu mobiililaitteiden akunkäyttöä ja pienempiä resursseja huomioiden. (1, hakusanalla Android operating system, hakupäivä 14.3.2011.)

## 6.3 JSON

JSON eli JavaScript Object Notation on kevyt datanvälitysmuoto. Ihmisten on helppo lukea ja kirjoittaa sitä. Tietokoneiden on helppo ja nopea luoda ja purkaa JSON-muotoista dataa. JSON-datamuoto perustuu JavaScript-ohjelmointikieleen, joka perustuu ECMA-standardiin. JSON on tekstimuotoinen formaatti, joka ei ole riippuvainen mistään ohjelmointikielestä, vaikka se käyttää merkintätapoja, jotka ovat tuttuja muun muassa C:n, C++:n, C#:n, Javan, JavaScriptin, Perlin ja Pythonin käyttäjille. Nämä ominaisuudet tekevät JSONista ideaalisen datanvälitysformaatin. (2.)

JSON perustuu kahteen tietorakenteeseen: nimi/arvo-kokoelmaan ja järjestettyyn arvolistaan. Useissa ohjelmointikielissä nimi/arvo-kokoelmaa toteutetaan muun muassa objekteilla, kokoelmilla, tietueilla, hash-tauluilla ja dictionary-tietueilla. Järjestetty arvolista on ohjelmointikielissä yleensä toteutettu muun muassa taulukoilla, vektorilla, listalla tai sarjoilla. Nämä edellä mainitut tietorakenteet ovat yleisiä. Käytännössä kaikki nykyaikaiset ohjelmointikielet tukevat niitä jossain muodossa. Näin ollen on selvää, että hyvä datanvälitysformaatti perustuu juuri näihin tietorakenteisiin. (2.)

## 6.4 Tietokannat

### 6.4.1 TimeSheet Serverin MySQL

MySQL on avoimeen lähdekoodiin perustuva relaatiotietokantojen hallintasysteemi, joka serverinä ajettaessa tarjoaa useille käyttäjille pääsyn useisiin tietokantoihin. MySQL:n omisti ruotsalainen MySQL AB -yritys, kunnes Sun Microsystems osti sen vuonna 2008. Vuoden 2009 huhtikuussa Oracle Corporation osti Sun Microsystemsin ja sai kaupassa mukana tekijänoikeudet ja tavaramerkin MySQL:ään. (1, hakusanalla MySQL, hakupäivä 15.3.2011.)

TimeSheet Server käyttää MySQL-relaatiotietokantaa, joka pyörii MySQL Server 5.1:llä omalla Eneriksen lähiverkossa sijaitsevalla dedikoidulla tietokantapalvelimella.

### 6.4.2 Android-päätelaitteen SQLite

SQLite on avoimeen lähdekoodiin perustuva kirjasto, joka implementoi itsenäisen, palvelimettoman, transaktioihin perustuvan sulautetun SQL-tietokantamoottorin. SQLite ei sisällä serveritoimintoa ollenkaan, vaan kirjoittaa ja lukee tietoa suoraan tietokantatiedostosta levyjärjestelmästä. Täydellinen SQL-tietokanta monine tauluineen, indekseineen, triggereineen ja näkymineen on tallennettuna yhdessä tietokantatiedostossa. Tietokannan tiedostomuoto on täysin alustariippumaton – se toimii niin 32 ja 64 bitin järjestelmissä kuin myös big- ja little-endian-arkkitehtuureissa. (3.)

Näiden lukuisien ominaisuuksiensa ja alustariippumattomuutensa vuoksi Google on integroinut SQLiten Android-alustalle mukaan. Siksi oli looginen valinta valita SQLite käytetyksi tietokannaksi Android-client-projektissa. Android-client-projektissa on toteutettuna offline-toimintoa varten tarvittava laaja tietokanta ja tietokannan hallinnointiin tarvittavat metodit. Koska offline-toiminto hyllytettiin projektin edetessä, ei SQLite-kantaa käytetä ohjelmassa tällä

hetkellä. Päätettiin kuitenkin, että tietokannan muokkaukseen ja käyttöön tarvittava koodi pidetään projektissa mukana mahdollista jatkokehitystä varten.

# 7 TYÖVÄLINEET

## 7.1 Eclipse

Eclipse on The Eclipse Foundationin ylläpitämä avoimeen lähdekoodiin perustuva IDE (Integrated Development Environment). Eclipse IDE on alun perin IBM:n aloittama projekti, jota tukivat useat ohjelmistojen jälleenmyyjät. The Eclipse Foundation perustettiin vuonna 2004 takaamaan Eclipse-projektin läpinäkyvyys ja puolueettomuus. (4.)

Eclipse IDE valittiin tässä projektissa ohjelmointiympäristöksi siksi, että Android SDK on suunniteltu juuri Eclipsellä käytettäväksi. Vaikka iso osa muista IDE:istä jo tukee Android-ohjelmointia, on Eclipsellä silti paras tuki Android SDK:lle.

## 7.2 Android SDK

Android SDK on kokoelma työkaluja Android-ohjelmien kehitykseen. SDK sisältää muun muassa debuggerin, kirjastoja, mobiililaiteen emulaattorin, dokumentaatiota, esimerkkikoodia ja tutoriaaleja. Virallisesti tuettu ohjelmistonkehitysympäristö on Eclipse, jota käytetään ADT (Android Development Tools) -pluginin kanssa. SDK sisältää täydelliset työkalut paketoida Android-ohjelmia .apk-muodossa. (1, hakusanalla Android sdk, hakupäivä 19.3.2011)

Tässä projektissa Android SDK:sta käytettiin lähinnä ADT:tä, joka sisältää muun muassa logcat-ohjelman, jolla sai ohjelman debug-tekstin näkyviin. Myös ohjelman release-versioiden pakkaaminen ja digitaalinen allekirjoitus tapahtui SDK:n työkaluilla.

## 7.3 NetBeans IDE

NetBeans IDE on integroitu kehitysympäristö, jolla voi kehittää ohjelmia muun muassa seuraavilla kielillä: Java, JavaScript, PHP, Python, Ruby, Groovy, C ja C++. IDE on ohjelmoitu Java-ohjelmointikielellä ja se toimii kaikilla alustoilla, joissa on JVM (Java Virtual Machine) asennettuna. JDK (Java Development Kit) täytyy olla asennettuna Java-ohjelmien koodausta varten, mutta muut kielet eivät JDK:ta vaadi. NetBeans IDE on avoimeen lähdekoodiin perustuva projekti, jota tällä hetkellä kehittää Oracle Corporation. NetBeans IDE tukee kaikkia Java-koodauksen osa-alueita mukaan lukien Java SE, JavaFX, Java ME, web, EJB ja mobiiliohjelmistot. NetBeans tukee myös muun muassa Ant-pohjaista projektisysteemiä, koodin refaktorointia ja version hallintaa (CVS, Subversion, Mercurial, Clearcase). NetBeans IDE on lisensoitu tällä hetkellä kaksoislisenssillä, CDDL:llä (Common Development and Distribution License) ja GPL versio 2:lla. (1, hakusanalla NetBeans, hakupäivä 19.3.2011; 5.)

## 7.4 MySQL command line

MySQL command line on MySQL Serverin mukana tuleva ohjelma, jolla voi ottaa yhteyden MySQL Serveriin. Tällä yhteydellä pääsee käsiksi serverin tietokantaan ja voi tehdä haluttuja muutoksia siellä oleviin tauluihin tai tauluissa oleviin tietoihin.

Tässä projektissa käytin MySQL command line -työkalua hyödyksi lähinnä tietokannassa olevan datan tarkkailuun. Muun muassa kaiken Android-clientiltä tulleen datan muodon tarkastin MySQL command line -työkalulla. Tarvittaessa poistin väärämuotoista dataa, jotta TimeSheet Serverin toiminnan sujuvuus ei kärsisi. Tässä projektissa ei tarvinnut tehdä muutoksia tauluihin, mutta projekteissa, joissa tarvii, tauluihin tehdyt muutokset on todella helppo suorittaa MySQL command linen kautta.

## 7.5 SVN command line

Projektin ohjelmakoodin versionhallintaan viemisessä käytettiin SVN command lineä. Se on konsolipohjainen työkalu, jolla voidaan viedä koko projekti SVN:ään (Subversion), hakea projekteja ja päivittää vain projektissa paikallisesti muutetut tiedot svn:ään. Ohjelmalla voi myös hakea tiettyjä versioita SVN:stä tai palauttaa tiettyjä tiedostoja, tai koko projektin, tiettyyn revision versioon SVN:stä. Aina kun tehdään commit-toiminto eli viedään jotain muuttunutta tietoa SVN:ään, tälle toiminnolle tulee oma revision-numero, jonka avulla voidaan palata ohjelmakoodissa aina tiettyyn haluttuun pisteeseen. SVN command line tulee mukana Apachen Subversion -ohjelmistossa, joka on avoimeen lähdekoodiin perustuva projekti.

Yleisin projektissa tehty toiminto oli commit-toiminto. Siinä viedään vain muuttuneet tiedostot SVN-palvelimelle. Commit-toiminnossa käyttäjä menee projektin pääkansioon ja kirjoittaa rivin

```
svn commit
```

Tämän komennon jälkeen aukeaa terminaaliin tekstieditori, jossa annetaan commitin mukana näkyvä viesti. Viestin kirjoituksen jälkeen, kun tekstieditorin sulkee, lähetetään muuttuneet tiedostot SVN-palvelimelle. Mikäli projektiin on lisätty tiedostoja, joita ei ole vielä SVN:ssä olemassa, täytyy jokainen tiedosto lisätä SVN:ään käskyllä

```
svn add
```

Kaikkia tiedostoja ei tarvitse lisätä yksitellen, vaan add-käsky voi ottaa myös kokonaisia hakemistoja, jolloin se lisää automaattisesti kaikki tiedostot SVN:ään, joita hakemistosta löytyy. Kun tiedosto on merkitty lisättäväksi, seuraavassa commit-käskyssä kaikki lisätyksi merkityt tiedostot lähetetään SVN-palvelimelle. Kokonainen kansio sisältöineen lisättiin SVN:ään seuraavalla käskyllä:

```
svn add ./hakemisto/polku/*
```

(6, linkki documentation.)

# 8 PROJEKTIN TOTEUTUS

## 8.1 Aloitus

Android-client projekti alkoi Eneriksen sisäisellä suunnittelupalaverilla. Tämän jälkeen kun projektin arkkitehtuuritason ja toiminnalliset vaatimukset olivat kasassa, voitiin lähettää ehdotus opinnäytetyön aiheesta koululle. Tämän jälkeen koulu määräsi opinnäytetyön ohjaajan, jonka kanssa pidettiin aloituspalaveri Eneriksen toimitiloissa. Tästä projekti alkoi edetä vaatimusten tarkentamisella ja käyttöliittymäsuunnittelulla. Kun vaatimukset olivat tarpeeksi tarkat, alkoi ohjelman koodin kirjoitus ja käyttöliittymien toteutus. Kaikki ylimääräinen aika, joka koulun vastauksia odotellessa jäi, käytettiin Android-ohjelmointikielen opiskeluun.

## 8.2 Projektin hallinta

Projektin hallintaan käytettiin yksinkertaista tehtävälistaa ja pieniä palavereita. Koska projekti oli Eneriksen sisäinen ja projektin tekijöillä oli kaikilla muitakin projekteja työn alla, päätettiin heti projektin alussa, että projektin kehitykselle varataan huomattavasti normaalia pidempi aikataulu. Päätettiin, että projektin ohjelmakoodin tulisi olla valmis seitsemän kuukauden sisällä ja opinnäytetyön raportin kirjoittaminen suoritetaan ylimääräisellä ajalla eikä sitä lasketa projektin aikatauluun mukaan.

Projektissa ei pidetty yhtään isompaa yksittäistä palaveria aloituspalaverin lisäksi. Pienempiä palavereita pidettiin yleensä viikoittain ja tarvittaessa niin usein kuin jotain ongelmia ilmeni. Nämä palaverit olivat yleensä ihan muutaman minuutin mittaisia ja niissä käytiin läpi senhetkinen tehtävälista, mahdolliset senhetkiset ongelmat ja mahdollisesti tulevaisuudessa näkyvät ongelmia.

Käytetty tehtävälista oli yksinkertainen Excel-taulukko (kuva 15), missä tehtävät oli numeroitu, nimetty ja selitetty. Jokaisen tehtävän kohdalla näkyi myös

tehtävän lisäyspäivä, aloituspäivä ja lopetuspäivä. Tehtävälista toimi näin ollen perustana projektin hallinnalle.

| | TASK | Lisätty | Aloitettu | Valmis | Selite |
|---|---|---|---|---|---|
| 14 | 1 | 8.12.2010 | | | Offline tila |
| 15 | 2 | 8.12.2010 | 8.12.2010 | 20.12.2010 | Kirjautuminen |
| 16 | 3 | 8.12.2010 | 27.12.2010 | 29.12.2010 | Tuntien kirjaus |
| 17 | 4 | 8.12.2010 | 27.12.2010 | 29.12.2010 | Tuntien poistaminen |
| 18 | 5 | 8.12.2010 | 31.12.2010 | 4.1.2011 | Käyttäjien hallinta |
| 19 | 6 | 8.12.2010 | 28.12.2010 | 31.12.2010 | Projektien hallinta |
| 20 | 7 | 8.12.2010 | 28.12.2010 | 31.12.2010 | Projektin taskien hallinta |

*KUVA 15. Ote projektin tehtävälistasta*

## 8.3 Versionhallinta

Projektissa käytettiin Apachen Subversion (svn) -versionhallintaa. Yleisenä käytäntönä yrityksessä on, että jokaisen päivän tuotokset lähetetään työpäivän päätteeksi versionhallintaan. Kun projekti oli saatu ensimmäiseen release-versioon, siitä lähtien projektin versionumero aina kasvatettiin yhdellä, kun uusi release tehtiin. Näin projektin versionhallinta pysyi hyvin hallinnassa.

## 8.4 Suunnittelu

### 8.4.1 Käyttöliittymän suunnittelu

Ohjelman käyttöliittymän suunnitteli Eneriksen ohjelmistojen testaaja. Käyttöliittymiä ei voinut suunnitella suoraan Androidin tyyliohjeiden mukaisesti, koska projektin suunnitteluvaiheessa ei yrityksessä kenelläkään ollut vielä kokemusta Android-käyttöliittymistä. Tästä johtuen käyttöliittymistä tehtiin vain suuntaa antavat luonnokset ja ohjelman ajon etenemistä kuvaavat toiminnot.

Projektin edetessä käyttöliittymää jouduttiin suunnittelemaan useaan otteeseen uudestaan. Iso osa suunnitelmista meni uusiksi, koska joidenkin toimintojen toteuttaminen alkuperäisten suunnitelmien mukaan olisi vaatinut liikaa resursseja niiden käytöstä saatuun hyötyyn verrattuna. Hyvänä esimerkkinä tästä on kalenterikomponentin puuttuminen valitusta Android-alustan versiosta. Oman kalenterikomponentin tekeminen olisi ollut mahdollista, mutta pikapalaverissa käyttöliittymäsuunnittelijan kanssa päädyimme ratkaisuun, että on parempi idea toteuttaa listamainen kuukausinäkymä. Lopulta tästä ratkaisusta tuli toimivampi ja parempi ratkaisu kuin Qt-client-projektissa käytetystä kalenterinäkymästä. Muita ongelmia käyttöliittymän luonnissa aiheuttivat lokalisaatio ja näytön pieni koko. Qt-client-projektissa näytön pienuus ei ollut niin iso ongelma, vaikka näytön koko oli sama, koska Qt-clientissä toteutettiin huomattavasti vähemmän toimintoja. Suuremman toimintomäärän mahduttaminen näkymiin oli suurempi työ.

Lokalisaatio aiheutti ongelmia lähes jokaisessa näkymässä. Mobiililaitteen näyttö on alun perinkin pieni, ja joidenkin komponenttien koon täytyi olla suhteellisen iso näytön kokoon nähden. Lokalisaation vuoksi joidenkin painonappien ja tekstikenttien kokoa jouduttiin suurentamaan alkuperäisiä suunnitelmia suuremmaksi, mistä johtuen muilta tärkeiltä kontrolleilta väheni tila. Joissakin tapauksissa, kuten käyttäjien ja projektien listausnäkymässä, kontrolleista tuli sen verran pieniä, että niiden käytettävyyden taso laski liian alas. Näissä tapauksissa jouduttiin keksimään vaihtoehtoisia ratkaisuita käyttöliittymän kontrollien asetteluun tai joissakin tapauksissa vain etsimään vaihtoehtoisia suomen ja englannin kielen sanoja korvaamaan käyttöliittymää rikkovia pitkiä sanoja.

Käytännössä käyttöliittymän suunnittelu jatkui koko projektin kehityksen ajan. Muun muassa lokalisaatio oli lähes viimeinen tehtävä, joka projektissa toteutettiin. Alun perin käyttöliittymät suunniteltiin vain landscape-moodiin, koska Qt-client-projektissa portrait-moodin käyttöliittymää ei toteutettu ollenkaan. Projektin edetessä tuli kuitenkin huomattua, että on käytettävyyden

kannalta todella tärkeää, että myös portrait-moodin käyttöliittymät toteutetaan. Tässä vaiheessa käyttöliittymän suunnittelija joutui suunnittelemaan kaikkien näkymien käyttöliittymät uudestaan. Projektin loppuvaiheilla lokalisaatiota toteutettaessa käyttöliittymien suunnittelija osallistui hyvin aktiivisesti ohjelman kehitykseen.

### 8.4.2 Serverin muutosten suunnittelu

Kaikki suuret muutokset serverille mobiililaitteiden tukea varten tehtiin jo Qt-client-projektissa. Siksi serverin muutoksia ei tarvinnut suunnitella etukäteen juuri ollenkaan. Aina kun Android-client ohjelmaa kehittäessä tuli vastaan tilanne, jossa serveriltä tarvittiin uutta toiminnallisuutta tai kantahakuja, siinä vaiheessa toiminnot tai haut toteutettiin serverille sen kummemmin suunnittelematta. Näin pystyttiin tekemään, koska mitään uutta koodia serverille ei tullut, ainoastaan vanhojen toimintojen kopiointia ja pientä muokkausta.

## 8.5 Ohjelmiston kehitys

### 8.5.1 Käyttöliittymän toteutus

Käyttöliittymän toteutus alkoi heti projektin alussa. Sillä aikaa kun opettelin Androidin alkeita projektin aloittamista varten, käyttöliittymäsuunnittelija suunnitteli ensimmäiset näkymät, jotta projekti voisi alkaa niin pian kuin mahdollista. Heti alusta lähtien ohjelman kehityksen malli kulki iteroiden käyttöliittymien luonnin mukaan. Ensin luotiin suunnitelma kirjautumissivulle, sen jälkeen toteutettiin sen käyttöliittymä, jonka jälkeen toteutettiin sen toiminnot ohjelmakoodissa. Ohjelmassa on noin 20 eri näkymää eli käytännössä 20 eri iteraatiota, joista jokainen liittyi käyttöliittymän luontiin.

#### 8.5.1.1 Android xml -tiedostot ja käyttöliittymän määritys

Androidin käyttöliittymät määritellään Android xml -tiedostoissa. XML-tyylisten tiedostojen standardi vaatii, että ensimmäisellä rivillä määritellään käytetyn XML-standardin versionumero ja tiedoston enkoodaus. Tämä tapahtui seuraavalla merkkauksella:

```
<?xml version="1.0" encoding="utf-8"?>
```

Tällä rivillä kerrotaan, että xml-tiedosto käyttää XML-merkkauksen versiota 1.0 ja UTF-8 enkoodausta. Kaikki projektiin tehdyt käyttöliittymä xml-tiedostot käyttävät samaa versiota XML:stä ja samaa enkoodausta. Seuraavaksi Android xml -käyttöliittymätiedosto määrittelee jonkin suunnittelumallin. Nämä asettelumallit ovat ylimmän tason säiliöitä muille kontrolleille. Jokainen käyttöliittymätiedosto alkaa asettelumallin määrittelyllä. Näitä asettelumalleja voi olla useita sisäkkäin, minkä avulla voidaan luoda monimutkaisiakin käyttöliittymiä. Tässä projektissa ainut käyttämämme asettelumalli oli LinearLayout eli suora asettelu. Näitä asettelumalleja on joissakin tapauksissa käytetty jopa neljä kertaa sisäkkäin. Ensimmäisen asettelumallin eli xml-tiedoston rootin määrittely sisältää myös Androidin vaatiman skeeman määrittelyn, joka tapahtuu seuraavan kaltaisella merkkauksella:

```
xmlns:android="http://schemas.android.com/apk/res/android"
```

Jokaiseen kontrolliin täytyy myös määritellä sen koko. Kontrollin koko voi olla joko absoluuttinen pikselikoko, fill_parent-määre, jolloin kontrolli täyttää sen omistajan varaaman alueen näytöltä, tai wrap_content-määre, jolloin kontrollin koko on sen oman sisällön täyttämä alue. Kaikille näille määreille oli käyttöä tässä projektissa. Kontrollin koon määrittäminen tapahtuu merkkauksella:

```
android:layout_width="fill_parent"
android:layout_height="fill_parent"
```

Kontrolleilla on lukuisia muitakin mahdollisia määritelmiä. Näistä kuitenkin tärkeimmät ovat android:id, android:text, android:textSize, android:orientation ja android:gravity. android:orientation-kenttä kertoo asettelumallille, asetellaanko sen omistamat kontrollit vaaka- vai pystysuoraan näytölle. android:gravity-määre taas kertoo sen, mihin suuntaan näytöllä kontrollin sisältö (esimerkiksi teksti) asetellaan. Näitä määreitä on käytetty seuraavassa esimerkissä, joka on tekstikenttä admin-toimintojen käyttäjien hallinnassa.

```
<TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:textSize="18dip"
        android:background="#202020"
        android:paddingLeft="2dip"
        android:text="@string/users"/>
```

Tässä esimerkissä luodaan tekstikenttä, jonka leveys on isäntäkontrollin leveys, korkeus on tekstin korkeus, tekstin koko on 18 näytön kokoon suhteutettua pikseliä korkea, kontrollin taustaväri on harmaa (#202020), tekstin ja kontrollin vasemman laidan väli on 2 suhteutettua pikseliä, ja kontrollin sisältämä teksti haetaan string-tyyppisistä resursseista id:llä ”users”. Tässä tapauksessa users-tunnisteinen string-resurssi on ”Users” englanniksi ja ”Käyttäjät” suomeksi.

### 8.5.1.3 Androidin valmiit komponentit eli widgetit

Androidin käyttöliittymissä on lukuisia eri kontrolleja, kuten painonapit, tekstikentät, listakontrollit, spinner-listat ja syötekentät. On kuitenkin joitakin komponentteja, jotka koostuvat muista kontrolleista ja toteuttavat jonkin suuremman toiminnon. Näitä komponentteja kutsutaan widgeteiksi. Tässä projektissa valmiita widgettejä ei juurikaan käytetä. Admin-päänäkymässä on nähtävillä Androidin tabcontrol-widget ja normaalin käyttäjän raporttien

luontinäkymässä päivämäärien valintanäkymässä käytetään Androidin DatePicker-widgettiä (kuva 16).


*KUVA 16. Androidin DatePicker-widget raporttienluontinäkymässä*

### 8.5.1.4 Androidin ja Qt:n käyttöliittymäkoodauksen eroja

Android-clientin ja Qt-clientin käyttöliittymien luonti oli hyvin erilaista sekä määrittely- että kooditasolla. Qt-client projektissa käytössä oli loistava graafinen käyttöliittymiensuunnittelutyökalu Qt Creator. Tällä ohjelmalla pystyi luomaan käyttöliittymät drag and drop -periaatteella, eli saman tien näki, miltä käyttöliittymä tulee ohjelmassa näyttämään. Android-client projektissa heti alusta lähtien huomasimme, että käyttämämme Eclipse IDE:n ja Android SDK:n

tarjoama visuaalinen käyttöliittymiensuunnittelutyökalu oli sen verran huono, että oli nopeampaa ja helpompaa kirjoittaa käyttöliittymä suoraan xml-tiedostoon, jonka jälkeen katsoa miltä se näyttää ja testata ajonaikana emulaattorissa. Tässä projektissa käyttämämme tee ja testaa saman tien -tyyli oli huomattavasti hitaampaa kuin Qt-client projektissa. Kuitenkin tässä menettämämme aika saatiin takaisin käyttöliittymän ohjelmakoodin luomisessa, joka oli Android-projektissa helpompaa. Qt:n käyttämän C++-ohjelmointikielen vuoksi käyttöliittymän toiminnot oli hitaampaa toteuttaa.

### 8.5.2 Client-ohjelman toteutus

#### 8.5.2.1 Managereiden toteutus

Koko projektin toteutus alkoi NetworkManagerin toteutuksesta. NetworkManager toteuttaa serverin vaatiman alustariippumattoman protokollan toteutuksen. Tärkeintä oli heti projektin alusta lähtien saada yhteys serverille toimimaan, jotta muiden toimintojen toteutuksen aloittaminen voitiin aloittaa. NetworkManagerin toiminnallisuuden perusajatuksena oli se, että jokainen aktiviteetti, joka tarvitsee jotain dataa serveriltä, voi kutsua NetworkManagerin sendPost()-metodia. sendPost()-metodi ottaa parametrina käyttäjältä pyynnön header-arvon ja mahdollisen paketin mukana lähetettävän datan. Data, jota serveri voi vaatia, voi olla esimerkiksi se tieto, mille päivämäärälle tuntimerkinnät halutaan hakea tai mille projektille tehtävät halutaan hakea. Sen jälkeen sendPost()-metodi lähettää pyynnön serverille ja jää odottamaan vastausta. Vastauksen saatuaan se deserialisoi saadun JSON-muotoisen datan takaisin Java-luokka-muotoiseksi dataksi. Kun serverin vastaus on nyt halutussa muodossa, sendPost()-metodi lähettää datan ContentManager-luokalle ja palauttaa paluuarvona käyttäjälle tiedon siitä, onnistuiko datan haku serveriltä.

ContentManager-luokan tekeminen täytyi aloittaa lähes saman tien NetworkManagerin kanssa. NetworkManager käyttää ContentManageria

hyödyksi, mutta ContentManagerin ei tarvitse tietää NetworkManagerista mitään. ContentManager on vain tietovarasto, jossa on private-tyyppisiä listoja kaikista datoista, joita ohjelmassa käytetään, sekä getter- ja setterparit jokaiselle näistä listoista.

Projektia aloittaessa en tiennyt Androidista mitään, ja siksi meni kauan miettiä ratkaisu siihen, mikä olisi paras keino tallettaa ContentManagerin ja NetworkManagerin viittaukset, jotta jokainen niitä tarvitseva aktiviteetti voisi helposti niitä kutsua. Ratkaisuksi myöhemmin selkeni Androidin tarjoama ns. Application-luokka, eli pystyin tekemään luokan, jonka tyyppi kerrotaan Android manifest
-tiedostossa, ja tämä luokka on saatavilla jokaisesta aktiviteetista ohjelmassa. Tältä pohjalta tein luokan, joka yksinkertaisesti sisälsi viittaukset Network- ja ContentManageriin gettereineen ja settereineen. Ohjelman pääaktiviteetin luomisen alussa loin olion Network- ja ContentManagerista ja talletin niiden osoitteet Application-luokkaan. Tämän jälkeen jokainen aktiviteetti, jonka tarvitsee käyttää yhteyttä serverille, hakee Application-luokasta Network- ja ContentManagerin viittaukset ja tallettaa ne itselleen, jonka jälkeen ne voivat käyttää kyseisiä luokkia tarpeensa mukaan.

### 8.5.2.2 Aktiviteettien toteutus

Projektin suurin osuus käyttöliittymien toteuttamisen ohella oli aktiviteettien toteuttaminen. Lähes kaikki ohjelmassa oleva ohjelmakoodi on juuri aktiviteettien toteutusta. Uuden aktiviteetin toteuttaminen aloitettiin luomalla uusi Java-luokka, joka laitettiin perimään Androidin Activity-luokka. Jokaisen aktiviteetin tuli myös ylikirjoittaa Activity-kantaluokan onCreate()-metodi. Android-alusta kutsuu aktiviteettia luotaessa sen onCreate()-metodia, eli tässä metodissa luodaan aktiviteetin tarvitsemat tiedot, valitaan sille käyttöliittymä sekä yhdistetään käyttöliittymän toiminnot ohjelmakoodiin. onCreate()-metodin perusrakenne on seuraavanlainen:

```
@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
}
```

Esimerkissä @Override-tagi tarkoittaa sitä, että kantaluokan samannimistä metodia ei kutsuta automaattisesti, kun pääluokasta luodaan uutta oliota. Käytännössä tämä tarkoittaa sitä, että kun juuri luodusta aktiviteetista tehdään uutta oliota käyttäjän mennessä kyseiseen aktiviteettiin, ajetaan koodaajan tekemä onCreate()-metodi eikä kantaluokassa sijaitsevaa onCreate()-metodia. super.onCreate()-metodin kutsu tarkoittaa sitä, että tässä vaiheessa kutsutaan kantaluokan onCreate()-metodia. Koska sitä kutsutaan itse tässä vaiheessa, metodin suoritus jatkuu, jonka jälkeen koodaaja voi suorittaa omat haluamansa toiminnot aktiviteettia luotaessa.

Seuraavaksi onCreate()-metodissa tulee valita aktiviteetin käyttämä ulkoasu. Ulkoasu on pakko valita, mikäli halutaan, että aktiviteetissa näkyy käyttöliittymä. Ulkoasun valinta tapahtuu seuraavanlaisella koodirivillä:

```
setContentView(R.layout.main_form);
```

setContentView()-metodi ottaa parametrina käyttöliittymä-xml-tiedoston nimen, tässä tapauksessa main_form.xml-nimisen tiedoston. Tämän jälkeen jokaisessa tämän projektin aktiviteetissa, joka tarvitsee yhteyttä serverille, haetaan viittaus ohjelman Application-luokkaan. Tämä tapahtuu koodirivillä

```
DroidSheetApp app = (DroidSheetApp)getApplicationContext();
```

Kun Application-luokan viittaus (app) on saatu, voidaan sieltä hakea Network- ja ContentManagerin viittaukset aktiviteetin käyttöön. Viimeisenä yleisenä toimintona jokaiselle aktiviteetille täytyy luoda linkit käyttöliittymän kontrollien ja ohjelmakoodin välillä. Tämä linkin teko onnistuu ainakin kahdella eri tavalla:

joko tekemällä uusi kuuntelija-instanssi jokaiselle kontrollille tai laittamalla aktiviteetti implementoimaan kaikki tarvittavat kuuntelijat, jolloin kuuntelijana toimii itse aktiviteetti. Tässä projektissa on yhtä kontrollia lukuun ottamatta käytetty tyyliä, jossa aktiviteetti toteuttaa kaikki kuuntelijat. Näin ollen jokaista kuuntelijaa varten on olemassa vain yksi metodi, jossa on switch/case-rakenne päättelemään, mitä kontrolleista käyttäjä käyttää ja mitä ohjelmakoodia tulisi suorittaa, kun kyseistä kontrollia käytetään. Seuraavassa ohjelmakoodissa on esimerkki siitä, miten luodaan ensimmäisen mallinen kuuntelija kontrollille eli oma kuuntelija jokaiselle kontrollille. Kyseessä on siis ainut tämänkaltainen kontrollin kuuntelija tässä projektissa:

```
builder = new AlertDialog.Builder(this);
builder.setPositiveButton(R.string.set,
   new DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog,
                          int whichButton) {
         String value = input.getText().toString();
         currentYearFrom = Utils.parseInt(value);
         refreshDateFields(); }});
```

Tässä koodinpätkässä luodaan AlertDialog-tyyppistä widgetiä, jonka ok-painonappiin lisätään uusi kuuntelija. Toisessa koodiesimerkissä on kaikkialla muualla käytössä oleva toteutus, jossa aktiviteetti toteuttaa kuuntelijarajapinnat ja lisää itsensä jokaiselle kontrollille kuuntelijaksi:

```
public class ReportProjectsState extends Activity imple-
ments OnClickListener {

.
.
.
```

```
Button button = (Button) findViewById (
R.id.button_reports_projects_print);

if (button != null) {
   button.setOnClickListener(this);
}
```

Ylemmässä koodirivissä näkyy luokan esittely. Aktiviteetin nimi on ReportProjectsState, ja se perii Activity-kantaluokan ja toteuttaa OnClickListenerin-rajapinnan. Tämän toisessa koodiesimerkissä haetaan viittaus käyttöliittymässä olevaan print-painonappiin, ja mikäli viittaus löytyi, lisätään painonapille kuuntelijaksi tämä aktiviteettiluokka.

Koska tässä projektissa käytettiin aktiviteettia kuuntelijana, tulee aktiviteetin toteuttaa valitun rajapinnan metodit myös. Tämän mukaisesti edellisen esimerkin aktiviteetin tulee toteuttaa OnClickListener-rajapinnan metodit. Tämä onnistuu ylikirjoittamalla onClick()-metodi. onClick()-metodissa aktiviteetin tulee tarkistaa mikä kontrolli lähetti onClick-tapahtuman, ja sen jälkeen ajaa haluttu ohjelmakoodi. Seuraavassa koodiesimerkissä on edellisen esimerkkiaktiviteetin toteuttama onClick()-metodi ja esimerkissä linkitetyn print-painonapin ohjelmakoodin kutsuminen.

```
public void onClick(View v) {
   switch (v.getId()) {
   case R.id.button_reports_projects_print:
      showReportActivity();
      break;
   }
}
```

Näin ollen, kun käyttäjä painaa käyttöliittymästä Tulosta-painonappia, kutsutaan aktiviteetin showReportActivity()-metodia, joka luo uuden aktiviteetin näyttämään halutun raportin sisällön.

Joissakin aktiviteeteissa saattaa edellisten alustustoimintojen lisäksi olla tarve alustaa joitakin komponenttien tietoja myös. Hyvänä esimerkkinä näistä on kaikki listat ja spinnerit. Esimerkiksi päänäkymän, eli kuukausinäkymän, tuntilistaus täydennetään välittömästi pääaktiviteetin onCreate()-metodissa.

Yksinkertaisena esimerkkinä listan täydentämisestä on admin-toimintojen käyttäjien hallinnan käyttäjälistan täydentäminen. Listan täydentäminen tapahtuu seuraavanlaisella koodilla

```
private void populateUsersList() {
   String message = JsonSerializer
                    .getRequestUsersMessage();
   if (message != null) {
      if (nm.doSendPost( DroidSheetConstants
         .Request.LIST_USERS.getIntegerValue(), message)) {
         users = cm.getAllUsers();
      }
   }

   if (users.size() > 0) {
      String usersList[] = new String[users.size()];
      for (int i = 0; i < users.size(); i++) {
         usersList[i] = users.get(i).getName();
      }

      userListAdapter = new TaskListAdapter(this,
         R.layout.task_listitem, usersList);

      ListView listView = (ListView) findViewById
                          (R.id.admin_users_user_list);
      if (listView != null) {
         listView.setAdapter(userListAdapter);
```

```
        }
    }
}
```

Tässä esimerkissä näkyy NetworkManagerin käyttö. Ensimmäisenä koodinpätkässä luodaan JSON-muotoinen viesti, joka lähetetään NetworkManagerille doSendPost()-metodilla. Kun serveri palauttaa vastauksen pyyntöön, haetaan ContentManagerista getAllUsers()-metodilla talteen serveriltä saatu data users ArrayList-olioon. Tämän jälkeen, kun serveriltä on saatu data talteen aktiviteettiin, aletaan täyttää itse listaa. Luodaan String-array, jonka koko on serveriltä saatujen käyttäjätietojen määrän kokoinen. Tämän jälkeen String-arrayhyn laitetaan saatujen käyttäjien nimet. Sitten etsitään listakontrolli käyttöliittymästä ja tehdään sille uusi adapteri, joka hallinnoi listan sisällön piirtoa. Tämän koodipätkän jälkeen käyttäjien nimet tulevat halutulle listalle näkyviin.

### 8.5.2.3 Tilakone

Ohjelmistossa ei käytännössä toteutettu tilakonetta lainkaan. Tämä johtui siitä, että Android-sovelluksen elinkaari on monelta osin poikkeava normaalin työpöytäsovelluksen elinkaaresta. Käytännössä normaalin työpöytäsovelluksen elinkaaren toteutus olisi ollut paljon resursseja vaativaa ja kaiken lisäksi täysin vastoin Androidin tyylimääritelmiä. Näin ollen päätettiin, että tilakone toteutetaan Androidin suosittelemalla tavalla.

Ohjelman tilakoneeksi muodostuivat aktiviteetit. Pääaktiviteetti oli aina alimmaisena, ja jokainen eri toiminto avasi joko yhden tai useamman uuden aktiviteetin vanhan päälle. Näin ollen kun mobiililaitteella painettiin takaisin-nappia, tai ohjelman Takaisin-painonappeja painettiin, sai käyttäjä kuvitelman, että ohjelma palaisi takaisin edelliseen näyttöön, vaikka käytännössä edelliset aktiviteetit eivät ikinä taustalta pois lähteneetkään. Tämän kaltaisen tilakoneen idea oli itselleni hyvin uusi, ja monessa kohtaa opinnäytetyöprojektin edetessä

tuli ongelmia vastaan, jolloin omaa ajattelutyyliä täytyi muuttaa Androidille sopivaksi. Vastaan tuli myös tilanteita, missä oma tilakone olisi toiminut huomattavasti paremmin kuin tämä aktiviteettien kasaus -tilakone. Hyvänä esimerkkinä siitä, missä oma tilakone olisi toiminut paremmin kuin käyttöön valittu, oli se, että kun admin-toiminnoista hallitaan käyttäjää, on sillä hetkellä näkyvä aktiviteetti tarkalleen sama kuin se aktiviteetti, joka admin-käyttäjällä on jo auki pohjimmaisena. Näin tässä nimenomaisessa tilanteessa käyttäjällä oli kaksi tarkalleen samaa aktiviteettia auki aktiviteettipinossa, eli resursseja on mennyt hukkaan.

### 8.5.3 Server-ohjelman toimintojen toteutus

Projektissa serverille ei tullut perustavanlaatuisia muutoksia. Iso osa serverin muutoksista oli uusien Actioneiden eli toimintojen tekeminen. Jokaiselle Android-clientin toiminnolle, jota Qt-client ei toteuttanut, jouduttiin tekemään uusi Action. Koska Qt-client toteutti ainoastaan tuntien kirjauksen, Actioneiden määrä lähes kaksinkertaistui Android-clienttiä tehdessä. Vaikka alkuperäinen serveri tukee kaikkia toimintoja Swing-clienttiä varten, jouduttiin Android-clienttiä varten toteuttamaan samat Actionit joka tapauksessa. Syy tähän oli se, että mobiililaitteilla on käytössä vähemmän resursseja kuin normaalilla tietokoneella. Työpöytäsovelluksena toimiva Swing-client pystyi siis käsittelemään suurempia datamääriä kerralla nopeammin ilman mitään ongelmia. Mobiilialustalla resurssien käytön täytyi olla koko ajan mielessä ohjelman toimintoja luodessa.

Uusien Actioneiden luonti serverille toteutettiin niin, että ensimmäiseksi luotiin halutun toiminnon tarvitsema tietokantakysely. Kaikki serverin tietokantakyselyt ovat tallennettuna yhdessä staattisessa luokassa. Kun kysely oli valmis, voitiin tehdä uusi Action-luokka, joka perittiin AbstractAction-kantaluokasta. Tämän jälkeen AbstractAction-luokkaan tehtiin kantakyselyä käyttävä metodi, joka palauttaa halutun datan tekemällä pyynnön tietokantaserverille ja käsittelemällä sen vastauksen.

Seuraavana uuden toiminnon luonnissa serverille piti tehdä uusi header-tieto. Header-tiedot ovat enum-tietotyyppejä, joiden avulla serveri tietää, mitä Actionia sen tulee kutsua, kun pyyntö clientiltä saapuu. Kun clientiltä saadaan pyyntö, serverillä on switch/case-rakenne, joka sisältää kaikkien headereiden listan, ja tämän rakenteen avulla serveri ohjaa saadun pyynnön oikealle Actionille. Tämän jälkeen kutsutaan Actionin execute-metodia, joka suorittaa AbstractAction-kantaluokkaan tehdyn kantaa käyttävän metodin. Kun execute-metodin suoritus loppuu, se palauttaa kannasta saadun datan, joka lähetetään JSON-serializer-luokalle. JSON-serializer palauttaa sille syötetyn datan JSON-muodossa. Tämän jälkeen kasataan vastaus clientin lähettämään pyyntöön laittamalla sama header-tieto kuin pyynnössä, clientin auth-hash koodi sekä juuri serialisoitu JSON-muotoinen data yhteen pakettiin. Paketti lähetetään clientille ja servlettiosion suoritus loppuu.

Käytännössä kaikki serverille tulleet muutokset oli vain vanhan koodin kopioimista, hakuehtojen tarkentamista ja optimointia. Koko ajan täytyi kuitenkin pitää serveri täysin yhteensopivana aiempiin serverin versioihin, eli serverin tuli edelleen tukea Swing-clienttiä, jonka arkkitehtuuri ja protokolla poikkeavat huomattavasti mobiililaitteiden arkkitehtuurista ja alustariippumattomasta protokollasta.

## 8.6 Testaus ja laadunvarmistus

Ohjelman testaaminen aloitettiin aikaisessa vaiheessa. Heti kun ensimmäinen toiminnallisuus eli tuntien kirjaus oli valmis, Eneriksen ohjelmistojen testaaja alkoi testata ohjelmaa. Tämän jälkeen jokaisen uuden toiminnallisuuden jälkeen testaaja saman tien testasi sen. Kun jokin toiminnallisuus päätyi testaajalle asti, itse ohjelman kehitys jatkui normaalisti samaan aikaan testaamisen kanssa. Kun testauksessa ollut toiminta tuli täysin testatuksi, testaaja teki virheraportin. Sen avulla toiminnallisuudessa olleet ohjelmavirheet, puutteet ja asiavirheet saatiin korjattua. Aina kun virheraportti valmistui, se nousi ensimmäiseksi prioriteetiksi. Kun virheraportissa olleet huomiot oli saatu korjattua, toimitettiin

testaajalle korjattu versio toiminnallisuudesta. Tätä prosessia testaajan ja koodaajan välillä toistettiin niin kauan, kunnes toiminnallisuus toimi halutulla tavalla.

Virheraporttien tekemisen lisäksi Eneriksen testaaja toimi myös laadunvarmistajana. Jokainen virheraportti sisälsi myös parannusehdotuksia joihinkin toimintoihin. Näin mitään toimintoa ei päästetty lopulliseen versioon, ennen kuin testaaja oli varmistanut toiminnon laadun ja hyväksynyt sen. Laadunvarmistus vaati lähes yhtä paljon työtä kuin ohjelmavirheiden korjaaminen, sillä ohjelmistosta haluttiin tarkoituksella tehdä mahdollisimman laadukas.

# 9 LOPPUSANAT

## 9.1 Jatkokehitys

Android-client projektia kehitetään jatkossa ainakin virhekorjausten muodossa. Kun ohjelmaa käytetään mobiilipäätelaitteilla, ohjelmavirheitä tulee ilmaantumaan, sillä niitä kaikkia ei voi testausvaiheessa löytää. Mitään suurempia toimintojen lisäystä tai vanhojen muokkausta ei ole tiedossa tähän projektiin.

TimeSheet Plus -ohjelmistoon jatkokehitystä kuitenkin tulee. Suunnitelmissa on tehdä asiakasohjelma myös Applen iPhone-alustalle, koska se on jatkuvasti kasvava ja hyvin suosittu alusta. Kuitenkin iPhone-client-projektin tarkoitus tulee olemaan myös itse alustan opettelu, ei varsinaisesti ohjelman toteutus. Eneriksellä suunnitellaan laajentaa osaamista myös iOS-alustan ohjelmointiin. Hyvin todennäköisesti iPhone-clientin toteuttaminenkin vaatii muutoksia ja laajennuksia serverin koodiin.

## 9.2 Pohdinta

Projektin perustavoitteena oli laajentaa Eneriksen ohjelmisto-osaamista Android alustalle. Toinen tavoite oli saada laajennettua tuntienseurantajärjestelmän asiakasohjelmia sisältämään Android-päätelaitteet. Molemmat päätavoitteet saavutettiin korkein arvosanoin, sillä projektin laajuuden vuoksi sain opeteltua lähes kaikki Android-ohjelmistotuotannon peruselementit. Android-alusta oli itselleni täysin uusi osa-alue, sillä Androidia ei opiskelujeni aikana vielä koulussakaan opetettu. Uuden alustan opettelua auttoi huomattavasti kuitenkin aiempi ohjelmointikokemus. Mitään varsinaista ongelmaa ei tämän projektin aikana tullut. Ainoastaan Android-alustassa huomasin puutteita, minkä takia jouduin vähän väliä tekemään turhaa itseään toistavaa työtä tai tekemään asiat huonommin kuin olisin halunnut.

Android-alustan puutteista huolimatta näen Androidin tulevaisuuden erittäin valoisana. Projektin aikana huomasin, että Android-ohjelmistokehitys on erittäin yksinkertaista ja selkeää, minkä ansiosta yhä useammat ohjelmiston kehittäjät keskittyvät juuri Android-ohjelmistojen kehitykseen. Tästä syystä oli mielestäni hyvä idea Enerikseltä laajentaa ohjelmisto-osaamista koskemaan myös Android-alustoja.

Projektin luontee takia sen suunnittelu jäi vähän vähemmälle, kuin ehkä olisi ollut tarpeen. Koska ohjelmassa kopioitiin aiempien asiakasohjelmien toimintoja, ei tämä projekti toteuttanut varsinaisesti mitään uutta toimintoa. Tämän takia suurimman osan projektille varatusta ajasta pystyin kuluttamaan Android-alustan opiskeluun ja erilaisten Android-teknologioiden testaamiseen. Koska kyseessä oli opiskelumuotoinen projekti, jossa tuloksena tullut asiakasohjelma oli vasta toisiksi tärkein vaatimus, saatiin juuri tärkein tavoite eli alustan opiskelu toteutettua.

Projektin hallinta sujui suhteellisen helposti soveltamalla omaa scrum-malliani. Palavereita pidettiin aina silloin, kun oli tarpeen. Joinakin viikkoina oli useampia palavereita, kun taas toisina viikkoina niitä ei tarvinnut pitää lainkaan, ja usein palavereiden pituus oli vain muutaman minuutin. Tämä oli mielestäni ehkä huonoin osa tässä projektissa, koska tiimityöskentelystä ei saanut uutta kokemusta. Tosin opinnäytetyön luonteen vuoksi oli ehkä parempikin, että tämän projektin ohjelmakoodi tehtiin yksin.

Koko projektin ajan sain loistavaa tukea työkavereiltani jokaisella tarvitsemallani osa-alueella ja projektissa suhteellisen aktiivisesti mukana ollut Eneriksen ohjelmistojen testaaja ylitti täysin odotukset tarjoamalla loistavia ideoita eri toimintojen toteuttamiseen, vaikka hänellä ei ole ohjelmointitason kokemusta lainkaan. Tällaisissa tapauksissa huomasin itse sortuneeni koodareiden perusongelmaan, eli kun osaa koodata, ei välttämättä osaa katsoa joitakin ongelmia vähän kauempaa jolloin näkisi että mitään ongelmaa ei oikeasti edes ole.

Tämä projekti sopi mielestäni hyvin opinnäytetyön aiheeksi opiskelumaisen luonteensa johdosta. Mikäli projektin olisi tehnyt henkilö, jolla on jo laaja kokemus Android-alustasta, ei se olisi sopinut opinnäytetyön aiheeksi. Suurin osa projektiin varatusta ajasta meni Android-alustan opiskeluun ja erilaisten toimintojen toteutuksen suunnitteluun. Erityisesti aikaa kului siihen mietintään, miten jonkin toiminnon saa toteutettua Android-alustalla, kun saman toiminnon on tehnyt muilla alustoilla huomattavasti helpommin. Myös mobiililaitteiden pienemmät resurssit olivat koko projektin toteutuksen ajan mielen päällä. Suurin osa työtunteja kului juuri käyttöliittymien toteutukseen, vaikka ne olivatkin suunniteltu etukäteen. Projektin opiskelumaisen luonteen vuoksi ohjelmakoodin toteutuksessa oltiin hyvin joustavia ja tämän vuoksi lyhyitä projektipalavereita pidettiin normaalia ohjelmistoprojektia useammin.

# LÄHDELUETTELO


1. Wikipedia. 2011. Vapaa tietosanakirja. Saatavilla: http://www.wikipedia.org.

2. JSON. 2011. JSON – dataformaatin kotisivut. Saatavilla: http://www.json.org. Hakupäivä 15.3.2011.

3. SQLite. 2011. Alustariippumaton paikallinen tietokantasysteemi. Saatavilla: http://www.sqlite.org. Hakupäivä 10.3.2011.

4. Eclipse. 2011. Saatavilla: http://www.eclipse.org. Hakupäivä: 10.3.2011.

5. NetBeans. 2011. Saatavilla: http://netbeans.org. Hakupäivä 17.3.2011.

6. Apache Subversion. 2011. Versionhallintajärjestelmä. Saatavilla: http://subversion.apache.org. Hakupäivä 19.3.2011