OULUN SEUDUN
AMMATTIKORKEAKOULU

Aki Syri

IOS-ASIAKASSOVELLUS TUNTIENSEURANTAJÄRJESTELMÄÄN

IOS-ASIAKASSOVELLUS TUNTIENSEURANTAJÄRJESTELMÄÄN

Aki Syri
Opinnäytetyö
26.10.2011
Tietotekniikan koulutusohjelma
Oulun seudun ammattikorkeakoulu

TIIVISTELMÄ

Oulun seudun ammattikorkeakoulu
Tietotekniikan koulutusohjelma, ohjelmistojen tuotanto

Tekijä: Aki Syri
Opinnäytetyön nimi: iOS-asiakasohjelma tuntienseurantajärjestelmään
Työn ohjaaja(t): Kari Salonen, Eneris Solutions Oy; Pekka Alaluukas, Oulun seudun ammattikorkeakoulu
Työn valmistumislukukausi ja -vuosi: syksy 2011 Sivumäärä: 59

Tässä opinnäytetyöprojektissa toteutettiin iOS-käyttöjärjestelmälle sovellus, joka laajentaa Eneris Solutions Oy:n TimeSheet-tuoteperheen mobiililaitetukea. Projektissa keskityttiin käyttöliittymän toimivuuteen ja käytön nopeuteen. Projektissa tehty sovellus käyttää valmista serveritoteutusta.

Projektissa toteutetulla sovelluksella voidaan lisätä ja poistaa tuntimerkintöja halutulle projektille sekä tarkastella raportteja merkityistä tunneista. Lisäksi ylläpitäjä voi lisätä uusia käyttäjiä sekä muokata käyttäjien tietoja. Ylläpitäjä voi myös lisätä muille käyttäjille projekteja, joihin he voivat tehdä tuntimerkintöjä.

IOS-asiakassovellus tehtiin Objective-C-ohjelmointikielellä, xCode-kehitystyökalulla. Sovellus lähettää ja vastaanottaa tietoa serveriltä http-yhteydellä käyttäen aiemmin määriteltyäs JSON-pohjaista protokollaa.

Projektin lopputuotteena saatiin toimiva iOS-sovellus, joka toimii luotettavasti ja on nopea ja helppo käyttää.

Asiasanat: iOS, Objective-C, mobiililaite, asiakas-palvelinarkkitehtuuri, JSON

ABSTRACT

Oulu University of Applied Sciences
Information Technology, Software Development

Author: Aki Syri
Title of thesis: iOS 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: Fall 2011 Number of pages: 59

In this thesis project, an application was developed for iOS operating system, which expands mobile device support of TimeSheet product. The project was made for Eneris Solutions Oy. This project was focused on ensuring good user experience, easy to use user interface and quick usability. The application was developed to work with server developed earlier.

With the application which developed in this project, user can add and remove hour markings for projects and view reports from marked hours. As an administrator user, new users can be added and user information can be modified. Administrator can also attach projects to users so they can mark hours to it.

IOS client application was programmed using Objective-C language with xCode development tool. The application sends and receives data from server in JSON-format over http. The application uses JSON-based protocol developed earlier.

Final product in this project was working iOS-application which works reliably and it is fast and easy to use.

Keywords: iOS, Objective-C, Mobile device, Server/Client architecture, JSON

ALKULAUSE

Tämä opinnäytetyö on tehty keväällä ja kesällä 2011. Työn tilaajana oli Eneris Solutions Oy. Työssä oli mukana Eneris Solutions Oy:n toimitusjohtaja Tuomas Fabritius, joka oikoluki opinnäytetyön ennen työn lähettämistä lehtori Pekka Alaluukkaan tarkastettavaksi. Enerikseltä mukana oli myös yrityksen teknologiajohtaja Kari Salonen, joka osallistui opinnäytetyön sisällön määrittelyyn sekä antoi teknistä tukea opinnäytetyön aikana. Kari Salonen oli myös vahvasti mukana toteutusvaiheessa antamassa hyviä ideoita opinnäytetyön teknisen osan ratkaisuihin. Ohjaava opettajan Pekka Alaluukkaan tehtävänä oli työn teknisen sisällön tarkastaminen. Lehtori Tuula Hopeavuori opasti opinnäytetyön kielellisessä ulkoasussa sekä teki viimeiset kielitarkastukset.

Haluan kiittää koko Eneris Solutions Oy:n henkilökuntaa erinomaisesta tuesta ja ohjauksesta opinnäytetyössä sekä suuret kiitokset kaikille työhön osallistuville ohjaajille, tarkastajille sekä tukijoille.

Oulussa 26.10.2011

Aki Syri

SISÄLLYS

LYHENTEET JA TERMINOLOGIA

Client – Asiakasohjelma, jolla voidaan ottaa yhteys johonkin palvelimeen ja toteuttaa palvelimen tarjoamia toimintoja

IDE – (Integrated Development Environment) Integroitu ohjelmointiympäristö

iOS – Applen kehittämä käyttöjärjestelmä iPhone-matkapuhelimelle sekä iPad-taulutietokoneelle

JSON – (JavaScript Object Notation) Yksinkertainen tekstimuotoinen tiedonsiirtomuoto

JVM – (Java Virtual Machine) Java-virtuaalikone

MySQL – Relaatiotietokantaohjelmisto

Qt – Alustariippumaton ohjelmistojen ja graafisten käyttöliittymien kehitysympäristö

XML – (Extensible Merkup Language) Rakenteellinen kuvauskieli, joka auttaa jäsentämään laajoja tietomassoja selkeämmin.

1 JOHDANTO

Erilaisten mobiililaitteiden yleistyttyä työn tekeminen ei ole enää niin paikkariippuvaista kuin ennen. Nykyiset työnteon ja ajankäytön tehokkuuden vaatimukset luovat paineita tehdä myös työkaluista joustavia ja helposti käytettäviä. Erilaisten ohjelmistoratkaisujen päätelaitteina ei enää ole pelkästään samassa paikassa oleva pöytätietokone, vaan erilaiset mobiililaitteet soveltuvat monenlaisten sovellusten päätelaitevaihtoehdoiksi.

Näistä lähtökohdista Eneris Solutions Oy lähti laajentamaan oman tuotteensa laitetukea mobiililaitteille. Aiemmin mobiililaitetukea laajenettiin Android- ja mobiili-Qt-pohjalle, nyt myös suuressa suosiossa olevalle Applen iOS-alustalle. Opinnäyteyöprojektilla saatiin hankittua samalla kokemusta sovelluskehityksestä iOS-alustalle sekä TimeSheet-tuntienseurantasovellukselle laajempi laitetuki. Projektin lähtökohtana oli ensimmäisesi tehty TimeSheet-työpöytäsovellus, jonka ominaisuuksia ja toimintoja oli tarkoitus saada käytettäväksi iOS-alustalla.

Opinnäytetyöprojektissa toteutettiin Objective-C-ohjelmointikielellä TimeSheet-tuntienseurantajärjestelmään asiakasohjelma iOS 4 -käyttöjärjestelmälle. Projektin tekemistä helpotti huomattavasti Android-sovelluksiin verrattuna se, että iOS-käyttöjärjestelmää käyttäviä laitteita ei ole montaa eri tyyppiä. Matkapuhelimissa on vain kahta eri näytön kokoa ja muilta ominaisuuksiltaan laitteet ovat hyvin identtisiä.

2 LÄHTÖKOHTA JA VAATIMUKSET

2.1 Projektin lähtökohta

Projektin lähtökohtana oli laajentaa Eneris Solutions Oy:n TimeSheet-tuntienhallintasovelluksen laitetukea iOS-alustalle sekä hankkia yritykselle pohjaa iOS-ohjelmoinnista asiakasprojekteja varten. Opinnäytetyön tilaajana toimi Eneris Solutions Oy ja työ tehtiin yrityksen sisäisenä projektina. Yrityksellä oli aiempaa mobiiliosaamista Androidin, Symbianin sekä Windows-mobilen muodossa, mutta iOS-sovelluksia yrityksessä ei ollut aiemmin tehty.

Projektia aloitettaessa Objective-C:n opiskelu täytyi aloittaa täysin nollatasosta, joskin opiskelua hieman auttoi jonkinlainen kokemus C- ja C++-ohjelmoinnista. Koska yritys on painottunut toiminnassaan Java-ohjelmointiin ja iOS-ohjelmoinnista yrityksessä ei ollut aikaisempaa kokemusta, teknistä tukea ohjelmoinnin harjoitteluun ei juuri yrityksestä saanut. Tämä ei kuitenkaan tuottanut suuria ongelmia, koska Applen omat iOS-ohjelmoinnin oppaat ovat kattavia ja Internetistä löytyi hyvin paljon esimerkkejä harjoittelun tueksi.

2.2 Vaatimukset

2.2.1 Toiminnalliset vaatimukset

Sovelluksen toiminnalliset vaatimukset voitiin ottaa suoraan aiemmin tehdyistä asiakassovelluksista. Tärkeimpänä runkona toiminnallisille vaatimuksille oli aiempaa Android-asiakassovellusta varten tehty toimintolistaus. Yhtenä tärkeänä vaatimuksena oli tehdä helppo- ja nopeakäyttöinen sovellus, jossa säilyisi iOS-sovellusten ”look and feel”. Tästä johtuen iOS-asiakassovelluksen käyttöliittymästä olisi tehtävä yksinkertainen ja yksiselitteinen.

Tarkoituksena oli suunnitella iOS-asiakassovellus täysin puhtaalta pöydältä, joten käyttöliittymissä ei otettaisi mallia aiemmin tehdyistä asiakassovelluksista. Heti alkuvaiheessa tuli selväksi, että kaikkia ominaisuuksia ei voitaisi projektin aikarajan puitteissa toteuttaa, joten ominaisuudet karsiutuivatkin pian tuntien kirjaamiseen ja poistamiseen, yksinkertaiseen raporttinäkymään sekä käyttäjän asetuksiin. Ylläpitäjän hallintatoimintoja tehtäisiin lisäksi, jos aikaa siihen jää.

2.2.2 Tekniset vaatimukset

Sovelluksen teknisistä vaatimuksista tärkeimpänä oli tehdä sovellus, joka käyttää samaa tietokantaa sekä serveriä muiden TimeSheet-asiakassovellusten kanssa. Täytyi myös varautua tekemään muutoksia serverin lähdekoodiin, jotta se saataisiin toimimaan iOS-sovelluksen kanssa.

Koska sovelluksen täytyi käyttää samaa serveritoteutusta muiden TimeSheet-asiakassovellusten kanssa, iOS-asiakassovelluksen täytyi lähettää ja vastaanottaa tieto serverin käyttämässä JSON-muodossa. Tämä tarkoittaa sitä, että sovellukseen jouduttiin ottamaan käyttöön kolmannen osapuolen tekemä JSON-kirjasto.

Sovelluksen täytyi myös toimia tausta-ajossa, jolloin tuntien kirjaamisoperaation voisi jättää kesken ja jatkaa samasta pisteestä myöhemmin. Sovellusta suunniteltaessa oli lisäksi otettava huomioon, että se toimisi myös Applen iPad-tablet-tietokoneessa. Tätä vaatimusta ei kuitenkaan pidetty tärkeänä ottaen huomioon opinnäytetyöprojektille varatun ajan, joten sovelluksen käyttöliittymät ja toiminnot suunniteltiin vain iPhone-matkapuhelimelle.

3 IPHONE JA IOS-KÄYTTÖJÄRJESTELMÄ

3.1 IPhone

IPhone on kosketusnäytöllinen Applen älypuhelin iPod-musiikkisoittimen mediaominaisuuksilla. Applen ensimmäisen puhelinmallin lanseerasi Applen toimitusjohtaja Steve Jobs Macworld-messuilla 9. tammikuuta 2007. iPhone julkaistiin Yhdysvalloissa 29. kesäkuuta 2007 ja se saapui Eurooppaan loppuvuodesta 2007. Aluksi iPhone-matkapuhelinta myytiin vain Isossa-Britanniassa, Saksassa ja Ranskassa. Suomessa ja monessa muussa uudessa myyntimaassa iPhonen uudistetun 3G-version jakelu alkoi samanaikaisesti 11. heinäkuuta 2008. (1.) Tällä hetkellä uusin iPhone malli on iPhone 4S, joka julkaistiin lokakuussa 2011.

IPhone 4 -matkapuhelimessa on 3,5 tuuman kapasitiivinen kosketusnäyttö, jonka tarkkuus on 960 x 640 pikseliä. Puhelimessa on Applen A4-prosessori ja muistia siinä on mallista riippuen 16 tai 32 gigatavua. IPhone 4:ssä on 5 megapikselin kamera ja LED-kuvausvalo sekä mahdollisuus kuvata 720p:n tarkkuudella olevaa videokuvaa 30 kuvaa sekunnissa. Puhelimen edessä on VGA-tasoinen kamera videopuheluita varten. Applen iPhone 4 -versiossa tuli uutena ominaisuutena mahdollisuus moniajoon ja puhelinta kiertävä metallikehys toimii osana antennirakennetta. Antenniratkaisusta tuli pian julkaisun jälkeen moitteita käyttäjiltä, sillä pitämällä tietyllä tavalla puhelimesta kiinni matkapuhelimen signaalin voimakkuus laski huomattavasti. (2.)

3.2 IOS

IOS (aiemmin nimellä iPhone OS) on Applen kehittämä käyttöjärjestelmä, joka on käytössä Applen iPhone-, iPod Touch- ja iPad-laitteissa. IOS perustuu Darwin BSD -käyttöjärjestelmään ja joihinkin Mac OS X -jakelun komponentteihin. Käyttöjärjestelmä vie levytilaa noin puoli gigatavua. IOS on suunniteltu käytettäväksi kosketusnäytön avulla. Apple julkaisi käyttöjärjestelmän ensimmäisen version kesäkuussa 2007. (3.)

IOS-käyttöjärjestelmä tukee monisormieleitä, joita ovat pyyhkäisy, kosketus, nipistys ja käänteinen nipistys. Myös joitain kiihtyvyysanturiin pohjautuvia toimintoja on otettu mukaan, kuten se että puhelinta ravistettaessa voidaan perua kirjoitettu teksti.

4 TIMESHEET-ASIAKASSOVELLUS

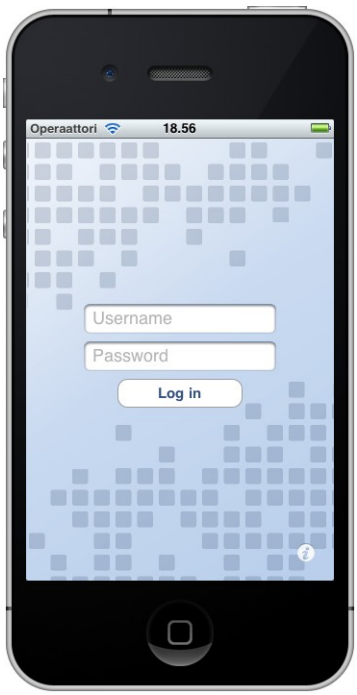
4.1 Sovelluksen toiminnot

4.1.1 Normaalin käyttäjän toiminnot

Tuntien kirjaaja on yleisin käyttäjämuoto sovelluksessa. Normaali käyttäjä (reporter) voi kirjata tunteja valittuihin projekteihin sekä tarkastella raportteja tehdyistä työtunneista. Normaalin käyttäjän toimintoihin kuuluvat tuntien kirjaaminen ja poistaminen, raporttinäkymä sekä käyttäjän asetukset.

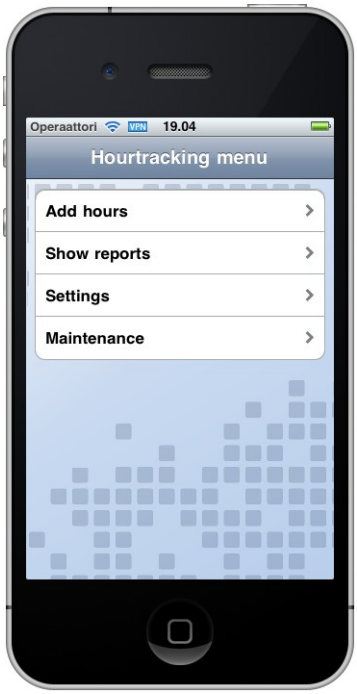
Käyttäjälle avautuu sovelluksen käynnistyttyä latausruutu (kuva 1), jonka jälkeen avautuu kirjautumisruutu (kuva 2). Kirjautumisruudusta pääsee asetusnäkymään, jossa voidaan vaihtaa palvelimen osoite. Tämän jälkeen asetettu osoite tallennetaan erilliseen asetustiedostoon.

KUVA 1. Latausruutu



KUVA 2. Kirjautumisnäkymä

Kirjautumisen jälkeen käyttäjälle avautuu päävalikko, jonka valinnat määräytyvät käyttäjätyypin mukaan. Jos käyttäjä on normaali tuntien kirjaaja, hallintatoimintoja ei näytetä päävalikossa (kuva 3).



KUVA 3. Päävalikko

Tuntien kirjaaminen

Tuntien kirjaaminen on sovelluksen käytetyin toiminto ja siihen kiinnitettiin työssä erityistä huomiota. Valittaessa tuntien kirjaus käyttäjälle avautuu projektinvalintanäkymä (kuva 4). Näkymässä olevat projektit ovat valittavissa asetustoiminnosta, jossa on valittavana kaikki ylläpitäjän käyttäjälle liittämät projektit.

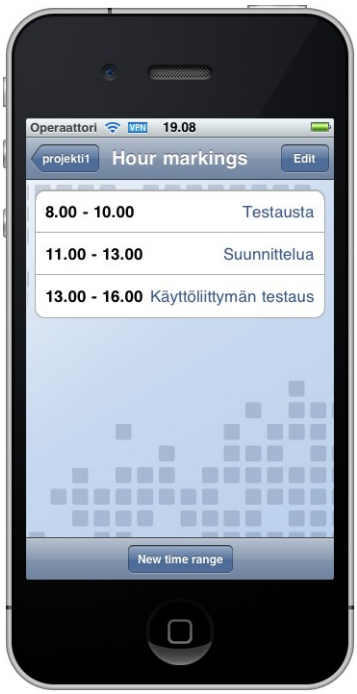
KUVA 4. Projektin valinta

Projektin valinnan jälkeen käyttäjä siirtyy päivän valintaan (kuva 5). Sovellus ehdottaa listassa nykyisen päivän sekä neljä päivää taaksepäin, tai halutessaan käyttäjä voi valita haluamansa päivän alapalkin napilla (kuva 6).

KUVA 5. Päivämäärän valinta

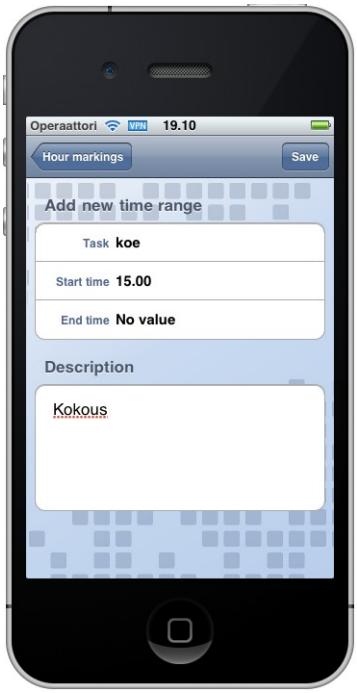
KUVA 6. Päivämäärän valitsin avattuna

Päivämäärän valinnan jälkeen käyttäjä siirtyy päivänäkymään, jossa näkyy valitulle päivälle ja valittuun projektiin tehdyt merkinnät (kuva 7). Jos merkintöjä ei päivälle ole, ruutu on tyhjä.



KUVA 7. Valitulle päivälle merkityt tunnit

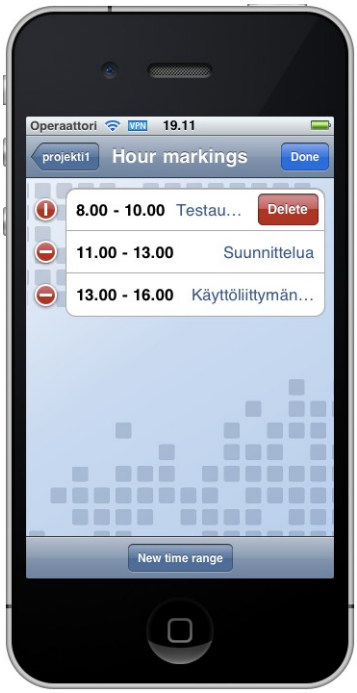
Uuden merkinnän lisäämisnäkymässä (kuva 8) käyttäjä voi halutessaan valita jonkin projektiin määritellyistä tehtävistä. Tuntien aloitus- ja lopetusajat valitaan iOS-järjestelmästä tutulla kellonajan valitsimella. Lopuksi lisätään merkintään kuvaus.



KUVA 8. Uuden merkinnän lisääminen

Tuntien poistaminen

Merkittyjen tuntien poistaminen tapahtuu koskettamalla päivänäkymässä edit-nappia tai pyyhkäisemällä sormella haluttua tuntimerkintää. Ensimmäisellä menetelmällä jokaisen merkinnän viereen ilmestyy liikennemerkkiä muistuttava punainen nappi, jota koskettamalla ilmestyy merkinnän poistonappi (kuva 9).



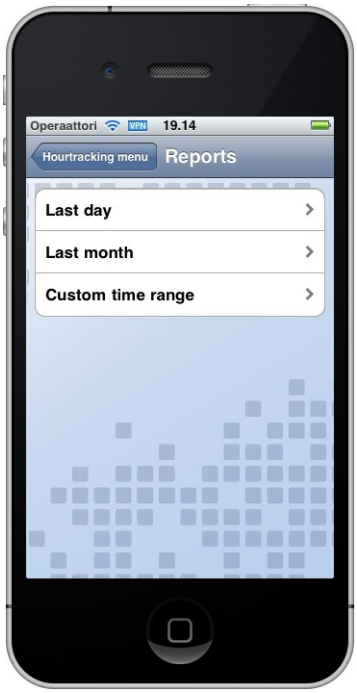
KUVA 9. Tuntimerkinnän poistotoiminto

Kun käyttäjä koskettaa poistonappia, ruudulle ilmestyy varmistusikkuna (kuva 10). Toinen tapa poistaa merkintöjä eroaa siten, että kun käyttäjä pyyhkäisee haluamansa merkinnän yli, merkinnän kohdalle ilmestyy suoraan poistonappi.

KUVA 10. Merkinnän poiston varmistus

Raporttinäkymät

Raporttitoiminnossa käyttäjä voi nopeasti tarkastella tehtyjä tunteja ja niiden kuvauksia valitulla aikavälillä. Samalla hän saa summan käytetystä ajasta. Sovellus mahdollistaa kaksi pikavalintaa raporttinäkymään: kuluneen päivän ja kuluneen kuukauden tunnit. Kolmantena vaihtoehtona käyttäjä voi itse määrätä aikavälin, jolla haluaa tarkastella merkintöjä. (Kuva 11.)



KUVA 11. Raporttinäkymän päävalikko

Valittaessa ensimmäinen tai toinen vaihtoehto päävalikosta sovellus siirtyy projektinvalintanäkymään. Kun projekti on valittu, käyttäjä näkee tuntimerkinnät kyseisellä aikavälillä ja yhteensä merkittyjen tuntien summan (kuva 12). Usein tuntimerkinnöissä kuvaukset ovat pidemmät kuin yksi tai kaksi sanaa, joten pidettäessä laitetta pystyasennossa voi tulla ahdasta. Sovellus on suunniteltu toimimaan sekä pysty- että vaakatasossa, jolloin laitetta kääntämällä saa näkyviin hieman enemmän merkinnän kuvauskentästä (kuva 13).

KUVA 12. Raporttinäkymä kuluneelta päivältä

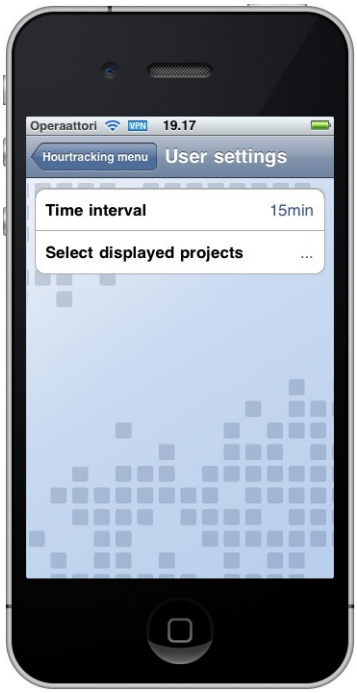
KUVA 13. Raporttinäkymä vaakatasossa

Kolmantena vaihtoehtona käyttäjä voi valita itse aikavälin, jolla raporttinäkymä luodaan (kuva 14).

KUVA 14. Raporttitoiminnon aikavälin valinta

Käyttäjän asetukset

Sovelluksen päävalikosta käyttäjä pääsee hallitsemaan omia henkilökohtaisia asetuksiaan (kuva 15). Asetukset tallennetaan tiedostoon ja jokaiselle samaa päätelaitetta käyttävälle käyttäjälle tallennetaan omat henkilökohtaiset asetukset. Käyttäjän asetuksista voi valita, halutaanko näyttää tuntimerkinnän luomisessa olevassa kellonaikavalitsimessa aika viidentoista vai kolmenkymmenen minuutin väleillä.



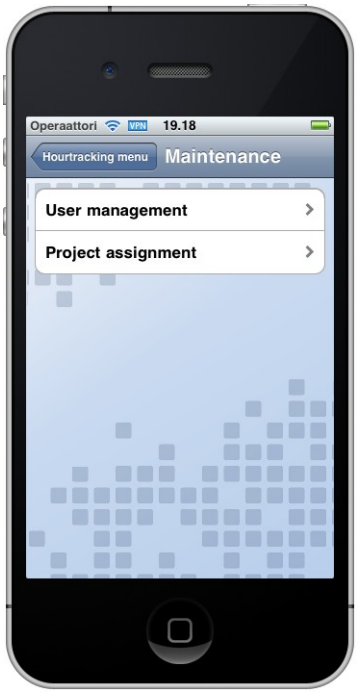
KUVA 15. Käyttäjän asetukset

Toisena asetusvalikossa käyttäjä voi valita projekteja, jotka näytetään muualla sovelluksessa projektinvalintanäkymissä (kuva 16). Tällä toiminnolla nopeutetaan sovelluksen käyttöä, jos ylläpitäjä on liittänyt käyttäjälle paljon projekteja, mutta vain muutamaan niistä tehdään merkintöjä aktiivisesti.

KUVA 16. Käyttäjäasetusten projektinvalinta

4.1.2 Ylläpitäjän toiminnot

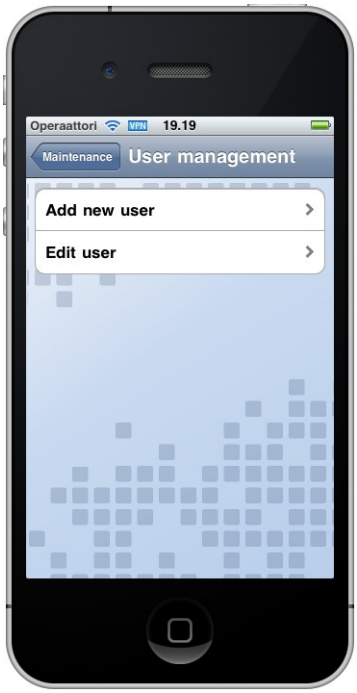
Normaalien tuntienkirjaamistoimintojen lisäksi ylläpitäjäksi (Admin) merkitty käyttäjä pääsee hallitsemaan muiden käyttäjien tietoja sekä liittämään projekte- ja käyttäjille. Näihin toimintoihin pääsee sovelluksen päävalikon kautta (kuva 17).



KUVA 17. Hallintanäkymän päävalikko

4.1.2.1 Käyttäjien hallinta

Käyttäjien hallinnassa ylläpitäjä pääsee luomaan uusia käyttäjiä sekä muokkaamaan käyttäjien tietoja (kuva 18).



KUVA 18. Käyttäjien hallintavalikko

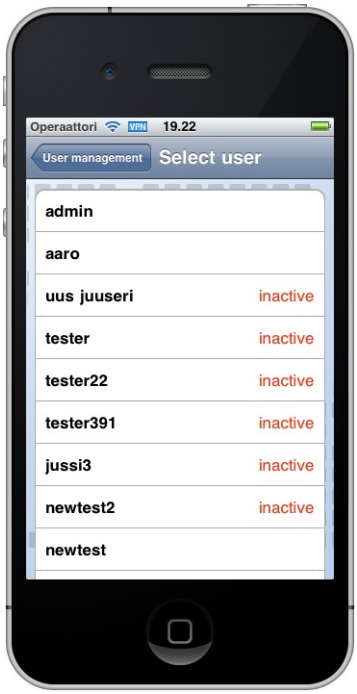
Käyttäjän luominen

Käyttäjä luodaan syöttämällä järjestelmään uuden käyttäjän nimi, käyttäjätunnus, salasana, puhelinnumero sekä käyttäjän rooli (kuva 19). Tässä sovelluksessa vain admin-tason käyttäjä pääsee hallintanäkymään ja manager-tason käyttäjä voi käyttää sovellusta samalla tavalla kuin normaali reporter-käyttäjä. Koska iOS-sovellus käyttää samaa tietokantaa laajemman PC-asiakassovelluksen kanssa, manager-tason käyttäjän luominen on lisätty myös iOS-asiakassovellukseen.

KUVA 19. Käyttäjän luontilomake

Käyttäjätietojen muokkaaminen

Kun halutaan muokata jonkin käyttäjän tietoja, esimerkiksi vaihtaa unohtunut salasana uuteen, ylläpitäjä saa listan kaikista järjestelmään rekisteröidyistä käyttäjistä (kuva 20). Listaan on merkitty punaisella ”inactive”-tekstillä käyttäjät, jotka on asetettu passiivisiksi. Passiiviset käyttäjät eivät voi kirjautua järjestelmään.



KUVA 20. Käyttäjälista

Kun käyttäjä on valittu, aukeaa käyttäjän tietojen muokkauslomake (kuva 21). Tässä näkymässä on lomakkeeseen esitäytettynä käyttäjän tiedot ja salasana-kentän paikalla on salasanan vaihtonappi. Kosketettaessa nappia napin paikalle ilmestyy tyhjä tekstikenttä, johon voi syöttää uuden salasanan. Jos nappia ei paineta, käyttäjän salasana pysyy ennallaan.

KUVA 21. Käyttäjän muokkauslomake

Viimeisenä valintana on käyttäjän aktiivisuusvalinta, jolla voidaan muuttaa käyttäjä passiiviseksi tai palauttaa passiivinen käyttäjä takaisin aktiiviseksi. Aktiivisuuden valinta ei näy kuvassa, koska näppäimistö on peittänyt valinnan puhelinnumeron muokkauksen ajaksi.

4.1.2.2 Projektien liittäminen käyttäjälle

Hallintatoiminnon päävalikossa toisena vaihtoehtona on projektien liittäminen käyttäjälle. Ensin ylläpitäjä valitsee käyttäjän, jolle projekteja halutaan liittää tai jolta niitä halutaan poistaa. Käyttäjän valintanäkymä on samanlainen kuin käyttäjätietojen muokkaustoiminnossa. Kun ylläpitäjä on valinnut käyttäjän, ruudulle tulee lista kaikista tietokannassa olevista projekteista (kuva 22).

KUVA 22. Projektien liitosnäkymä

Listaan on merkitty liitetyt projektit ja ylläpitäjä voi lisätä ja poistaa projekteja koskettamalla projektia listassa. Muutokset tallentuvat tietokantaan automaattisesti jokaisen muutoksen jälkeen.

4.2 Sovelluksen arkkitehtuuri

4.2.1 Navigointityypit

IOS-sovelluksissa on periaatteessa viisi eri mahdollisuutta toteuttaa käyttöliittymänavigointi. Näistä jokaisella on oma erityispiirteensä ja soveltuvuutensa erilaisille sovelluksille. Sovellusta suunnitellessa olikin tärkeää miettiä, millä tavalla navigointi kannattaa toteuttaa. Sovellusrakenteen pohjana on UiViewController-luokka ja kaikki navigointivaihtoehdot periytyvät tästä luokasta. Seuraavaksi esitellään kaksi yleisintä tapaa toteuttaa monimutkaisemman sovelluksen navigointi.

Navigointipohjainen sovellus

Tässä sovelluksessa päädyttiin käyttämään navigaatiopohjaista ratkaisua, joka tarjoaa mahdollisuuden siirtyä kätevästi näkymästä toiseen näytön yläosassa sijaitsevan navigointipalkin avulla. Navigointipohjaista sovellusta luotaessa kehitysympäristö generoi UiNavigationController-luokan, joka periytyy UiViewController-luokasta. (4.) Tämä kontrolleri tarjoaa navigointipalkin perustoiminnallisuuden eli paluunapin edelliseen näkymään ja nykyisen näkymän otsikon. Oletuksena navigaatiopohjaista sovellusta luotaessa UiNavigationControllerille luodaan ensimmäinen näkymä, jonka kontrolleri on UiTableViewController. Tämä taulukkonäkymä on ensimmäisenä objektina niin kutsutussa navigointipinossa, johon lisätään aina seuraava näkymä. Uuden näkymän luonti tapahtuu seuraavasti:

```
SecondViewController *svc = [[SecondViewController alloc initWithNibName:@”SecondView”
                            bundle:nil];
svc.title = @”New view”;
[self.navigationController pushViewController: svc animated: YES];
[svc release];
```

Luodaan objekti seuraavasta näkymäkontrollerista ja liitetään kontrolleriin haluttu näkymää kuvaava nib-tiedosto. Tämän jälkeen voidaan asettaa näkymälle otsikko ja mahdollisesti välittää muuta tietoa uudelle kontrollerille. Sitten lisätään objekti navigointipinoon ja valitaan, halutaanko näyttää siirtymisessä animaatiota vai ei. Lopuksi poistetaan viittaus objektiin.

Pinoon sijoitettu objekti säilyttää tilansa siihen asti, että se poistetaan pinosta. Toisin sanoen näkymän tila pysyy samana siihen asti, että palataan näkymästä taaksepäin.

Välilehtityyppinen sovellus

Toinen vaihtoehto toteuttaa navigointi sovelluksen sisällä on jakaa se välilehtiin. Tämä oli toinen vaihtoehto toteuttaa TimeSheet-asiakassovellus, mutta vaihtoehto hylättiin pian, koska pelkästään välilehdet eivät olisi riittäneet toteuttamaan

koko sovelluksen navigointia. Sovellukseen olisi siinä tapauksessa jouduttu rakentamaan toinen navigointimenetelmä rinnalle, jotta kaikki toiminnallisuudet saataisiin näytettyä selkeästi.

Välilehtiratkaisu eroaa teknisesti navigaatiopohjaisesta ratkaisusta kontrollereiden luonnin osalta. Luotaessa UiTabBarController ladataan jokaisen välilehden kontrolleri valmiiksi taulukkoon. Tällöin jokainen kontrollerin luoma näkymä säilyttää tilansa koko sovelluksen käytön ajan.

4.2.2 Sovelluksessa toteutettu rakenne

TimeSheet iOS -asiakassovelluksessa päädyttiin käyttämään perusrakenteitten kombinaatiota navigoinnissa. Kirjautumisruutu toimii sovelluksen päänäkymänä, johon lisätään serverinvalintanäkymä sekä muun sovelluksen navigointinäkymä modal-tyyppisenä näkymänä. Modal-tyyppisen näkymän etuna on se, että kirjautumisruudun käyttäjätunnus ja salasanakenttään syötetyt tiedot säilyvät taustalla avattaessa modal-tyyppinen näkymä. Esimerkiksi syötettyään tunnuksen ja salasanan käyttäjä voi vielä muuttaa tarvittaessa serverin osoitetta ja palata takaisin kirjautumaan sisälle järjestelmään syötetyillä tunnuksilla.

Käyttäjän kirjauduttua järjestelmään avautuu sovelluksen päävalikkonäkymä, joka toimii juurinäkymänä navigointipinossa. Tähän pinoon lisätään aina seuraava näkymä, johon siirrytään sovelluksessa.

Lisäksi sovellukseen luotiin muutamia yleisesti sovelluksessa käytettäviä luokkia, joista luotiin oliot sovelluksen pääluokassa eli appDelegate-luokassa. Tällä tavoin mistä tahansa sovelluksesta päästään käsiksi luotujen olioiden funktioihin luomatta niistä joka kerta omaa oliota.

NetworkManager-luokka on näistä tärkein luokka, jota kutsutaan eri puolilla sovellusta. NetworkManager-luokka hoitaa tietoliikenteen serverin ja iOS-asiakas-

sovelluksen välillä sekä hoitaa lähetettävän ja vastaanotettavan tiedon käsittelyn JSON-kirjaston avulla.

DateFormatter-luokka hoitaa päivämäärien muunnokset serveriltä saadusta muodosta ruudulla näytettävään tekstimuotoon sekä käyttäjän syöttämien valitsemien päivämäärien ja kellonaikojen muutoksen serverin vastaanottamaan muotoon.

Kolmantena AppDelegate-luokassa luodusta yleiskäyttöisestä oliosta on Data-luokka. Tähän luokkaan tallennetaan sovellukseen kirjautuneen käyttäjän tietoja sekä hoidetaan asetusten kirjoittaminen tiedostoon.

5 KÄYTETYT TEKNOLOGIAT

5.1 Objective-C

Objective-C-ohjelmointikieli on Brad J. Coxin 1980-luvun puolivälissä kehittämä ja nykyään Applen käyttämä ohjelmointikieli. Se on pieni oliolaajennus C-ohjelmointikieleen. (5.)

Objective-C mahdollistaa normaalin C-kielen syntaksin käyttämisen Objective-C-kielisen ohjelman seassa. Kaikki ei-oliopohjaiset operaatiot kuten primitiiviset muuttujat, funktiomäärittelyt, funktiokutsut jne. ovat syntaksiltaan identtiset C-kielen kanssa. Oliopohjaiset ominaisuudet taas on syntaksiltaan smallTalk-tyyppisiä. (5.)

Tyypillisimmät Objective-C-ohjelmointikielen ominaisuudet tulevat hyvin esille vertailtaessa syntaksia myös C-kieleen pohjautuvaan C++-ohjelmointikieleen. C++-kielessä otsikkotiedosto voisi olla seuraavanlainen:

```
class SomeClass : public SuperClass {
    protected:
        //instance variables
    public:
       //Class (static) functions
       static void * classMethod1();
       static return_type classMethod2(param1_type param1_varName);

       //Instance (member) functions
       return_type instanceMethod(param1_type param1_varName, param2_type
                                        param2_varName);

}
```

Vastaavanlainen toteutus Objective-C-kielellä olisi seuraava:

```
@interface ClassName : SuperClass
{
    //instance variables
}
+ classMethod1;
+ (return_type)classMethod2:(param1_type)param1_varName;

- (return_type)instanceMethod:(param1_type)param1_varName:(param2_type)param2_varName;
```

Plus-merkillä alkavat funktion määrittelyt ovat luokkafunktioita (class method), joita voidaan kutsua ilman, että luokasta luodaan olio. Jälkimmäinen miinus-merkillä alkava funktion määrittely on jäsenfunktio (instance method), jota voidaan kutsua vasta, kun kyseisestä luokasta on luotu olio.

Toinen erityispiirre Objective-C-ohjelmoinnissa on, että luokan jäsenmuuttujat (instance variable) voidaan määritellä propertyiksi otsikkotiedostossa:

```
@property(retain) NSSMutableString *someString;
```

Tämä määrittelee muokattavan tekstijono-objektin, jolle määritellään getter- ja setter-funktiot automaattisesti. Kun lähdekooditiedoston alkuun lisätään synthesize-määrittely, voidaan myöhemmin lukea ja muokata objektin arvoa getterien ja setterien avulla:

```
@synthesize someString;
.
.
.
self.someString = [[NSMutableString alloc] init];
[self.someString setString:@”fooBar”];
NSLog(@”someString value: @%”, [self someString]);
```

Edellä olevassa ohjelmakoodissa allokoidaan muistia someString-objektille, jonka jälkeen siihen sijoitetaan fooBar-merkkijono setter-funktion kautta. Tämän jälkeen tehdään lokimerkintä, joka tulostaa someString-objektin arvon.

5.2 JSON

JSON (JavaScript Object Notation) on kevyt tiedonvälitysmuoto. Se on ihmisille helppo lukea ja kirjoittaa sekä tietokoneille helppo purkaa ja luoda. Vaikka JSON perustuu JavaScript-ohjelmointikieleen ja JSON-tekstimuoto on täysin ohjelmointikieliriippumaton, se noudattaa merkintätapoja, jotka ovat tuttuja esimerkiksi C-, C++-, C#-, Java-, JavaScript-, Perl- ja Python-ohjelmointikielten käyttäjille. Näiden ominaisuuksien ansiosta JSON on erinomainen tiedonvälityskieli. (6.)

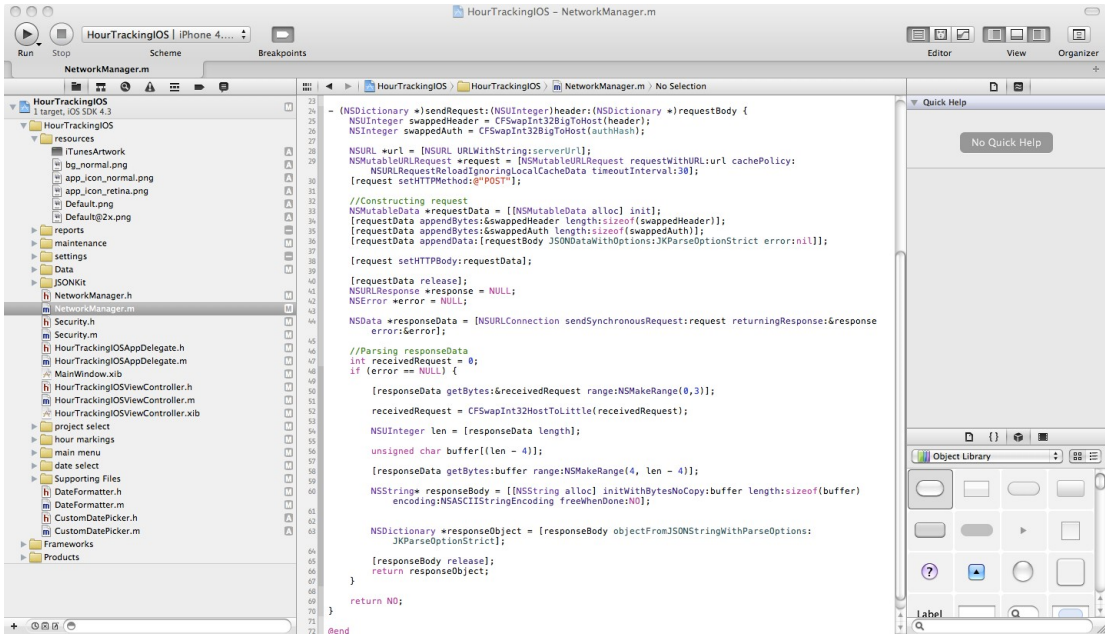
JSON perustuu kahteen erilaiseen rakenteeseen: nimi-arvo-pareihin sekä järjestettyihin arvolistoihin. Nimi-arvo-parikokoelmia käytetään useissa ohjelmointikielissä objekteissa, kokoelmissa, tietueissa, dictionary-tietueissa, hash-tauluissa, avainlistoissa ja assosiatiivisissa taulukoissa. Järjestettyjä arvolistoja käytetään useissa ohjelmointikielissä taulukoissa, vektoreissa, listoissa ja sarjoissa. (6.) Seuraavassa esimerkissä esitetään objekti, joka sisältää merkkijono-arvon, taulukon sekä boolean-tyyppisen arvon.

```
{”string”: ”String value”, ”array”: [1,2,3], ”boolean” : false }
```

6 TYÖVÄLINEET

6.1 xCode 4

XCode on Applen kehittämä IDE OS X- ja iOS-sovellusten kehitysympäristö (kuva 23) (7). Yleensä IDE sisältää vähintään lähdekoodieditorin, kääntäjän ja debuggerin (8). Näiden lisäksi xCode 4 sisältää myös käyttöliittymäsuunnittelijan, automaattisen virheentarkistuksen, iPhone/iPad-simulaattorin sekä Instruments-sovelluksen muistivuotojen ja resurssien käytön analysointia varten.



KUVA 23. xCode 4-kehitysympäristö

6.2 NetBeans IDE

Netbeans IDE on integroitu kehitysympäristö Java-, JavaScript-, PHP-, Python-, Groovy-, C-, C++-, Scala- ja Clojure-ohjelmointikielille. Netbeans on ohjelmoitu Java-ohjelmointikielellä ja toimii kaikilla alustoilla, johon on asennettuna JVM. (9.)

Netbeans-kehitysympäristöä käytettiin Java-ohjelmointikielellä kirjoitetun TimeSheet-serverin ohjelmakoodin tutkimiseen. Projektissa ei tarvinnut tehdä ollenkaan muutoksia serverin toteutukseen, mutta serverin koodiin tutustuminen oli välttämätöntä iOS-asiakassovelluksen toteuttamisessa.

6.3 MySQL command line

MySQL command line -ohjelmaa käytettiin lähinnä iOS-sovelluksen käyttämän tietokannan tarkkailuun. MySQL command line on osa MySQL-serveriä ja sillä voidaan hallita serverin tietokantoja. Projektin aikana itse tietokantaan ei tarvinnut tehdä muutoksia, mutta työkalua käytettiin hyvin paljon tietokantaan kirjoitettavan tiedon oikeellisuuden tarkastamiseen.

7 PROJEKTIN TOTEUTUS

7.1 Aloitus

Projektin alussa suurin osa sovelluksen vaatimuksista oli jo valmiina, koska samaan järjestelmään on jo tehty aiemmin kaksi mobiiliclient-sovellusta eri alustoille. Tästä johtuen sovelluksen toimintojen suunnittelu rajoittui käytännössä siihen, miten toiminnot esitetään iPhonen käyttöliittymässä ja mitkä kaikki PC-client -sovelluksen toiminnot otetaan mukaan iOS-sovellukseen.

Ennen varsinaista suunnittelutyötä aloitettiin iOS-alustaan tutustuminen pienten esimerkkisovellusten avulla. Yrityksessä ei ollut aikaisempaa kokemusta iOS-ohjelmoinnista, joten harjoittelu täytyi aloittaa aivan lähtöpisteestä.

Toisin kuin monet muut ohjelmointiympäristöt, iOS kehitysympäristön asennus onnistui odotettua helpommin. Kun xCode-paketti oli ostettu ja asennettu Applen AppStore-sovelluskaupasta, kehitysympäristö oli täysin valmis ilman mitään konfigurointeja. Käytännössä koko kehityspaketin asennus onnistui parilla hiiren näpäytyksellä.

Aiemmin pitkään PHP-kielellä ohjelmoineelle Objective-C:n syntaksi tuotti monessa kohtaa ongelmia harjoittelun aikana. Lisäksi xCode 4:n käyttöliittymäsuunnittelijan ja ohjelmakoodin liitokset hoidetaan raahausmenetelmällä. Tämä tarkoittaa sitä, että käyttöliittymäelementti voidaan liittää ohjelmakoodiin vetämällä ctrl-nappi pohjassa viiva suoraan ohjelmakoodiin. Raahausmenetelmä osoittautui hyvin nopeaksi ja havainnolliseksi tavaksi tehdä liitoksia, mutta alussa sillä oli vain sekoittava vaikutus.

Pienten harjoitusten aikana kartoitettiin sopivia käyttöliittymäratkaisuja TimeSheet-sovellusta varten. Näiden kokeilujen pohjalta alkoikin muotoutua selkeä

näkemys siitä, millaisilla ratkaisuilla sovelluksen käyttöliittymää lähdetään toteuttamaan.

7.2 Käyttöliittymän suunnittelu

Käyttöliittymä vei suurimman osan suunnitteluun käytetystä ajasta. Koko projektissa pääajatuksena oli kehittää helppo- ja nopeakäyttöinen sovellus, jonka käyttöönotto onnistuisi ilman ohjeiden lukemista.

Suunnittelun lähtökohtana oli pitää näkymät mahdollisimman yksinkertaisina. Jokainen toiminto toimisi omassa näkymässään. Myös erilaiset valinnat pilkottiin eri näkymiin, jolloin valintojen määrä yhdessä näkymässä pysyy hallinnassa. Siksi yksinkertaisia valikkonäkymiä tuli sovellukseen aika paljon, mutta tämän ratkaisun ansiosta sovellusta on helppo laajentaa myöhemmin.

Sovelluksen tärkeimmästä toimintosarjasta, eli normaalin käyttäjän tuntikirjauksesta, tehtiin käyttöliittymäkaavio Inkscape-vektorigrafiikkaohjelmalla (liite 1). Kaavion avulla saatiin selkeämpi käsitys siitä, mihin suuntaan käyttöliittymän muita toimintoja viedään.

Apple on luonut hyvin tarkat määrittelyt sille, miten sovelluksen käyttöliittymä olisi suunniteltava. iPhone SDK:n tarjoamat valmiit käyttöliittymäkomponentit ovat helppokäyttöisiä ja luovat yhtenäisen tuntuisen käyttöliittymän, jos Applen ohjeistusta noudatetaan. Tässä projektissa pyrittiin käyttämään mahdollisimman pitkälle oletuskäyttöliittymäkomponentteja, jolloin lopputulos olisi samassa linjassa muiden iOS-sovellusten kanssa.

Suunnitteluun suuren haasteen toi se, että kokemus Applen tuotteista, etenkin iOS-käyttöjärjestelmästä, oli olematon. Käyttöliittymää suunnitellessa oli vaikea miettiä kuhunkin tarkoitukseen sopivia ratkaisuja pelkästään paperilla, joten jo suunnittelun alkuvaiheessa alettiin rakentaa sovelluksen käyttöliittymää. Käyttö-

liittymä rakennettiin ilman serveriyhteyttä, joten aluksi valikoihin sijoitettiin vain ennalta määrättyä sisältöä.

7.3 Serveriyhteyden muodostaminen

Serveriyhteyden rakentaminen aloitettiin, kun sovelluksen käyttöliittymä oli saatu karkealla tasolla valmiiksi. Alussa täytyi etsiä sopiva JSON-kirjasto, jolla voidaan generoida serverille lähetettävä tieto sekä purkaa saatu vastaus. Pienen etsiskelyn jälkeen päädyttiin käyttämään JSONKit-nimistä kirjastoa (https://github.com/johnezang/JSONKit). Kirjaston etuna oli muihin vaihtoehtoihin nähden huomattavasti pienempi koko sekä useissa testeissä mainittu muita suurempi nopeus tietoja käsiteltäessä.

Serveriyhteyden luominen aloitettiin luomalla NetworkManager-luokka, joka sisälsi käytännössä yhden sendRequest-funktion tietojen lähetykseen. Funktio saa parametrina serverille lähetettävän toimintoa kuvaavan header-arvon sekä dictionary-objektin, joka sisältää serverille lähetettävän tiedon. Tämä tieto voi sisältää esimerkiksi tuntien kirjaamisvaiheessa projektin, johon tuntimerkinnät liitetään, tarkennetun tehtävän, aloitusajan, lopetusajan sekä merkinnän kuvauksen. Lisäksi serverille lähetettävän paketin alkuun liitetään kirjautumisen yhteydessä palautunut auth-hash-arvo. Tämä arvo pidetään NetworkManagerissa muistissa koko sovelluksen elinkaaren ajan, jolloin sitä ei tarvitse välittää näkymältä toiselle.

SendRequest-funktiossa tehdään pyynnön lähettämiseen tarvittavat määritykset eli määritellään sovellus lähettämään data http-POST-kutsuna sekä paketoidaan auth-hash-arvo, pyynnön header-arvo sekä JSON-muotoon muutettu tietosisältö.

Tässä sovelluksessa tiedonsiirto toteutettiin synkronisena, jolloin koko sovellus jää odottamaan serveriltä vastausta. Tämä ei testien mukaan tuottanut ongelmia, sillä lähetettävät ja vastaanotettavat datamäärät ovat hyvin pieniä. Synkro-

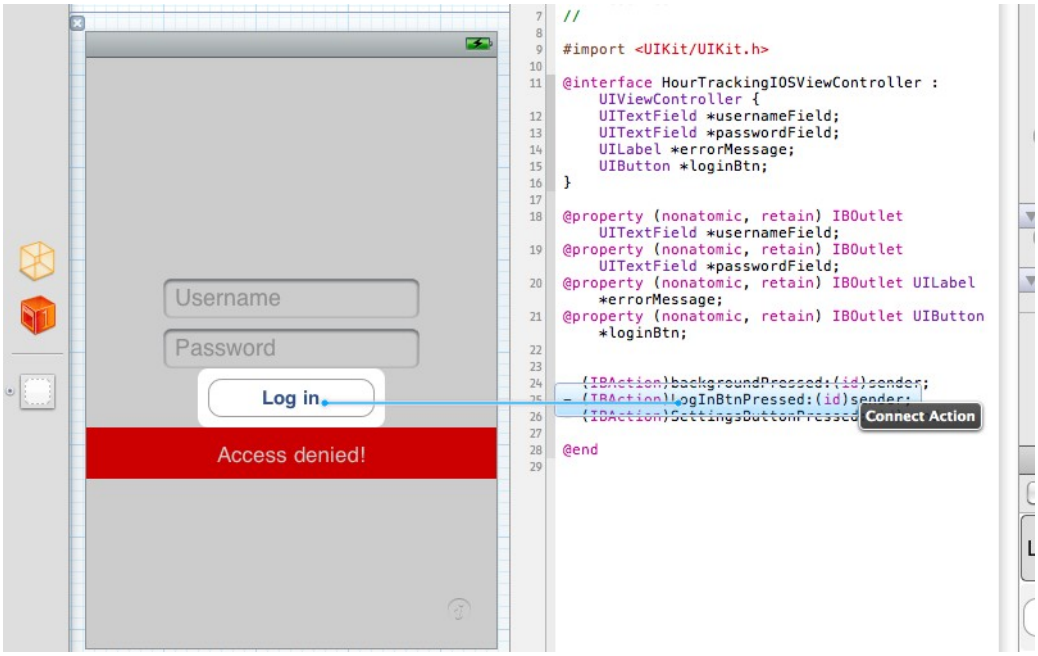
nisen yhteyden käyttäminen oli lähes välttämätöntä, sillä useimmissa tapauksissa tiedonsiirto tapahtuu siirryttäessä näkymästä toiseen ja serverin palauttamalla tiedolla on suuri merkitys seuraavan näkymän luonnissa.

Paluuarvona serverin palauttama data puretaan JSONkit-kirjaston avulla dictionary-objektiksi, joka palautetaan lähetysfunktiota kutsuneelle näkymäkontrollerille.

7.4 Käyttöliittymän ja niiden toiminnallisuuksien toteutus

Käyttöliittymiä voidaan rakentaa kolmella eri tavalla. Ensimmäinen ja helpoin tapa on luoda näkymät xCode-kehitysympäristön mukana tulevalla käyttöliittymäsuunnittelijalla. Suunnittelijalla voidaan raahata käyttöliittymäkomponentteja paikoilleen komponenttikirjastosta sekä muokata niiden ominaisuuksia selkeiden valikoiden avulla. Käyttöliittymää kuvaavat tiedostot (.xib -tiedostot) ovat xml-tyyppisiä tiedostoja, joten käyttöliittymän voisi rakentaa myös kirjoittamalla suoraan xml-kuvauksen näkymästä. Kolmantena vaihtoehtona on luoda koko käyttöliittymä ohjelmakoodin avulla. Tällöin erillisiä näkymätiedostoja ei tarvita.

Tässä projektissa kaikki näkymät toteutettiin käyttöliittymäsuunnittelijan avulla muutamia erikoistilanteita lukuunottamatta. Käyttöliittymäsuunnittelijalla voitiin tehdä helposti käyttöliittymäkomponenttien liitokset ja nappien tapahtumafunktioiden luonti raahaamalla viiva käyttöliittymästä ohjelmakoodiin (kuva 24).



KUVA 24. Tapahtumafunktion luonti kirjautumisnapille

7.4.1 Perusnäkymä

Sovelluksen kirjautumisruudussa, palvelinosoitteen valintanäkymässä, tuntimerkinnän lisätietonäkymässä sekä käyttäjien hallintalomakkeissa käytettiin pohjana tyhjää perusnäkymää. Perusnäkymään voi asetella käyttöliittymäelementtejä ilman rajoituksia, joten se sopi hyvin lomaketyyppisiin näkymiin. Kirjautumisnäkymä luotiin projektin perustamisessa generoidun ensimmäisen näkymän paikalle. Kirjautumisnäkymälle oli siis oma näkymäkontrolleri sekä xib-päätteinen näkymätiedosto. Käyttöliittymäsuunnittelijassa aseteltiin tarvittavat elementit paikoilleen sekä tehtiin liitokset käyttöliittymätiedoston elementtien ja näkymäkontrollerin välille.

Uutta näkymäkontrolleria luotaessa generoituu joukko funktioita valmiiksi. Tyypillisimmät funktiot normaalissa UIViewController-luokasta perityssä kontrollerissa ovat seuraavanlaisia. ViewDidLoad-funktiota kutsutaan silloin, kun käyttöliittymätiedosto on ladattu, mutta ennen kuin sitä piirretään ruudulle. Tässä vaiheessa on hyvä esimerkiksi tehdä muuttujien alustuksia ja muokata käyttöliitty-

mäelementtejä kooditasolla. Kirjautumisnäkymän kontrollerissa asetettiin salasanakenttä näyttämään palloja tekstin sijasta.

```
-(void)viewDidLoad {
     [super viewDidLoad];
     passwordField.secureTextEntry = YES;
}
```

ShouldAutorotateToInterfaceOrientation-funktiossa voidaan antaa näkymälle lupa kääntyä laitetta käännettäessä sekä voidaan tehdä kääntämiseen liittyviä operaatioita. Funktio palauttaa boolean-arvon YES, jos näkymälle on annettu lupa kääntyä. Seuraavassa esimerkissä on tuntien kirjaamisnäkymän kääntöfunktio. Tässä normaalin tarkistuksen lisäksi muutetaan kuvauskentän kokoa näytön ollessa vaakatasossa, jotta kaikki mahtuisi yhtä aikaa samaan ruutuun.

```
-(BOOL)shouldAutorotateToInterfaceOrientation:        (UIInterfaceOrientation)interfa-
ceOrientation {
     //Return YES for supported orientations
     switch (interfaceOrientation) {
          case UIInterfaceOrientationPortrait:
          {
               descriptionCellHeight = 130.0;
               return YES;
          }
          break;
          case UIInterfaceOrientationLandscapeLeft:
          {
               descriptionCellHeight = 80.0;
               return YES;
          }
          break;
          case UIInterfaceOrientationLandscapeRight:
          {
               descriptionCellHeight = 80.0;
               return YES;
          }
          break;
          case UIInterfaceOrientationUpsideDown:
          {
               return NO;
          }
     break;
     default:
```

```
            return NO;
        break;
        }
        [self.tableView reloadData];
}
```

ViewDidUnload-funktiossa vapautetaan kaikki alinäkymät eli käyttöliittymäkomponentit, jotka on liitetty outleteina käyttöliittymätiedostosta ohjelmakoodiin:

```
- (void)viewDidUnload {
        [self setUsernameField: nil];
        [self setPasswordField: nil];
        [self setErrorMessage: nil];
        [self setLoginBtn: nil];
        [super viewDidUnload];
 }
```

Dealloc-funktiossa poistetaan viittaukset alinäkymiin ennen näkymän tuhoamista:

```
- (void)dealloc {
        [usernameField release];
        [passwordField release];
        [errorMessage release];
        [loginBtn release];
        [super dealloc];
}
```

Näiden funktioiden lisäksi generoituvat myös seuraavat funktiot. ViewWillAppear-funktiota kutsutaan juuri ennen näkymän piirtoa ruudulle ja ViewDidAppear-funktiota kutsutaan taas näkymän piirron jälkeen. Näiden funktioiden käyttöä ei käsitellä tämän enempää.

7.4.2 Taulukkonäkymä

Taulukkonäkymä on yleisimmin käytetty näkymätyyppi sovelluksessa. Käyttöliittymäsuunnittelijassa taulukkonäkymän käsittely eroaa hyvin paljon tyhjän perus-

näkymän käsittelystä. Käyttöliittymäsuunnittelijassa voidaan tehdä vain hyvin suppeita muutoksia taulukkonäkymään. Suurin osa taulukkonäkymän muokkauksista tapahtuu itse ohjelmakoodissa. Luotaessa uusi taulukkonäkymä kehitysympäristö generoi uuden näkymäkontrollerin, joka periytyy UITableViewController-luokasta. Normaalissa näkymässä olevien vakiofunktioiden lisäksi generoituu seuraavia taulukoiden hallintaan liittyviä funktioita.

NumberOfSectionsInTableView-funktio palauttaa taulukon ryhmien lukumäärän. Tämän sovelluksen lähes kaikissa taulukkonäkymissä on vain yksi ryhmä, mutta taulukot on määritelty käyttöliittymäsuunnittelijassa ryhmitetyksi taulukoksi ulkonäöllisistä syistä. Seuraavassa esimerkissä on kyseinen funktio tyypillisimmillään tässä sovelluksessa.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {
     //Return number of sections
     return 1;
}
```

NumberOfRowsInSection-funktio palauttaa rivien lukumäärän kussakin ryhmässä. Yhden ryhmän käsittelyssä riittää, että palauttaa koko taulukon rivien lukumäärän, eli esimerkissä palautetaan päävalikon valintojen lukumäärä.

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section
{
     //Return the number of rows in the section
     return [self.menuItems count];
}
```

CellForRowAtIndexPath-funktiolla määritellään kunkin solun ominaisuudet taulukossa. Suurimmassa osassa näkymiä solujen määrittely hoidettiin seuraavan esimerkin mukaisesti.

```
- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:
              (NSIndexPath)indexPath {
     static NSString *CellIdentifier = @”Cell”;
     UITableViewCell *cell = [tableView dequeueReusableIdentifier: CellIdentifier];
```

```
    if(cell == nil) {
        cell = [[[UITableViewCell alloc] initWithStyle:UITableViewCellStyleDefault
                    reuseIdentifier:CellIdentifier] autorelease];
    }
    //Configure the cell
    cell.accessoryType = UITableViewCellAccessoryDisclosureIndicator;
    cell.textLabel.text = [menuItems objectAtIndex: indexPath.row];

    return cell;
}
```

Kun valitaan taulukosta jokin solu tai rivi, kutsutaan funktiota didSelectRowAtIndexPath. Funktio ottaa parametrina indexPath-objektin, jolla voidaan yksilöidä valittu rivi. Seuraavassa esimerkissä on sovelluksen päävalikon valintafunktio hieman lyhennettynä.

```
-(void)tableView:(UITableView *)tableView didSelectRowAtIndexPath
                        :(NSIndexPath *)indexPath {
    …
    if(indexPath.row == 3) {
        MaintenanceMenuViewController *maintenanceController =
                        [[MaintenanceMenuViewController alloc]
                        initWithNibName:@"MaintenanceMenuViewController"
                        bundle: nil];
        maintenanceController.title = @"Maintenance";
        [self.navigationController pushViewController:maintenanceController
                        animated: YES];
        [maintenanceController release];
    }
}
```

Suurimmassa osassa näkymistä taulukoiden arvojen näyttäminen sekä valintatoiminnallisuus saatiin tehtyä edellä esitettyyn tapaan. Normaaleissa valikkonäkymissä voitiin suoraan asettaa taulukkoon staattiset tekstit. Näkymät, jossa näytetään serveriltä haettua tietoa, eroavat siten, että viewDidLoad-funktiossa kutsutaan NetworkManagerin sendRequest-funktiota, joka hakee serveriltä tarvittavat tiedot. Seuraavassa esimerkissä haetaan serveriltä projektilista projektinvalintanäkymää varten. SendRequest-funktio ottaa parametrina lähetettävän

header-arvon sekä serverille lähetettävän datasisällön. Tässä tapauksessa serverille lähetetään ainoastaan header-arvo.

```
projectNames = [[NSMutableArray alloc] init];
projectDesctriptions = [[NSMutableArray alloc] init];

HourTrackingIOSAppDelegate* appDelegate = (HourTrackingIOSAppDelegate*)[[UIApplication
                                          sharedApplication] delegate];
NSDictionary *projects = [appDelegate.networkManager sendRequest: 6 :NULL];
[appDelegate.data setProjects: projects];

NSArray *projectsToShow = [appDelegate.data loadProjectList];

for(NSDictionary *object in projects) {
     for(NSNumber *id in projectsToShow) {
          if([id isEqualToNumber:[object valueForKey: @"id"]]) {
              [projectNames addObject: [object valueForKey: @"name"]];
              [projectDescription addObject: [object valueForKey: @"description"]];
          }
     }
}
```

7.4.3 Poikkeuksia rakenteeseen

Erilaisten käyttöliittymäkomponenttien käyttäminen oli sovellusta tehdessä helppoa ja monia komponentteja pystyi käyttämään suoraan ilman muutoksia. Muutamissa tilanteissa tarvittiin muokkaamista ja soveltamista, jotta saatiin sovellus toimimaan halutulla tavalla.

Sovelluksessa käytetään hyvin paljon päivämäärän- ja ajanvalitsinrullia. Jotkin listavalinnat on toteutettu samanlaisella valitsinrullalla. Ongelmaksi muodostui sovellusta tehtäessä, että valitsinrullat eivät toimikaan samalla tavalla kuin ajateltiin. Valitsinrullan haluttiin automaattisesti liukuvan näkyviin valittaessa tekstikenttä tai nappia painamalla. Tähän jouduttiin ratkaisuksi rakentamaan oma toteutus, jossa valintaa tehtäessä piirretään taulukkonäkymän päälle valitsinkom-

ponentti. Lisäksi valintakomponentin yläpuolelle täytyi tehdä työkalupalkki, johon sijoitetaan valintakomponentin sulkemisnappi.

Valitsinkomponenttien piirtäminen ruudulle, niiden toiminnallisuuden tekeminen ja lopuksi piilottaminen ei ollut loppujen lopuksi vaikea tehtävä. Suurempi ongelma oli saada komponentit skaalautumaan ja sijoittumaan oikein laitetta kääntäessä. Kaikki vakiokomponentit, kuten tekstikentät, napit ja taulukkonäkymät, skaalautuvat yleensä oikein laitetta kääntäessä ja käyttöliittymäsuunnittelijassa on tarvittaessa helppo muuttaa niiden käyttäytymistä. Rullavalitsimen ongelmana oli se, että se ei osannut skaalautua kunnolla käännettäessä laitetta pystyasennosta vaakaan. Tämän ongelman ratkaisuun kului hyvin paljon aikaa. Lopulta ratkaisu löytyi eräältä internet-sivulta (http://www.llamagraphics.com/developer/using-uidatepicker-landscape-mode). Oli tehtävä uusi UIDatePicker-luokan perivä alaluokka CustomDatePicker, joka toteuttaa tarvittavat korjaukset (liite 2).

Toinen suurempi poikkeus normaaleihin käyttöliittymän rakennusrutiineihin oli oman alanäkymän sijoittaminen taulukkonäkymän soluun. Normaalin taulukkonäkymän solussa on vain muutama erilainen vaihtoehto, miten sisältö solussa näytetään. Sovelluksen tuntien kirjausnäkymässä tuli kuitenkin tarve näyttää useampi rivi tekstiä tuntimerkinnän kuvauksen kohdalla. Tarve ratkaistiin luomalla käyttöliittymäsuunnittelijassa taulukon solua kuvaava komponentti, johon voitiin määritellä mitä tahansa sisältöä. Tämän luodun solun sisälle sijoitettiin monirivinen tekstikenttä, johon voitiin kirjoittaa merkinnän kuvaus. Samaa menetelmää käytettiin myös raporttinäkymässä, mutta siinä yhden solujen sisällön elementit ladattiin kokonaan eri tiedostosta.

Kolmas mielenkiintoinen poikkeus oli tuntimerkintöjen järjestäminen aloitusajan mukaan. Tuntimerkintöjen listausnäkymään haetut tiedot sijoitettiin taulukkoon, jossa oli tuntimerkintä-olioita. Kun luotiin merkintöjen lajittelufunktio, yllätykseksi funktion määrittely ei toiminut ollenkaan Objective-C:n syntaksilla, vaan se täytyi

kirjoittaa C-kielen syntaksia noudattaen. Allaolevassa ohjelmakoodilistauksessa esitetään lajittelufunktiota kutsuva komento sekä itse lajittelufunktio.

```
     ...
     [timeRanges sortUsingFunction: sortRanges context: nil];
}

int sortRanges(id a, id b, void *context) {
     TimeRange* tra = a;
     TimeRange* trb = b;
     if ([[tra.start hour] integerValue] < [[trb.start hour] integerValue]) {
          return NSOrderedAscending;
     } else if ([[tra.start hour] integerValue] > [[trb.start hour] integerValue]) {
          return NSOrderedDescending;
     } else {
          if ([[tra.start min] integerValue] < [[trb.start min] integerValue]) {
               return NSOrderedAscending;
          } else if ([[tra.start min] integerValue] > [[trb.start min] integerValue]) {
               return NSOrderedDescending;
          } else {
               return NSOrderedSame;
          }
     }
     return NSOrderedSame;
}
```

7.5 Asetusten tallentaminen

Sovelluksessa käyttäjien tekemät asetukset tallentuvat puhelimeen erilliseen xml-tyyppiseen plist-tiedostoon. Kun sovellus käynnistetään ensimmäisen kerran, asetustiedostoon kirjoitetaan serverin osoite sekä ensimmäisen kirjautuneen käyttäjän asetukset. Käyttäjän muuttaessa omia asetuksiaan tiedot tallennetaan käyttäjän kohdalle asetustiedostossa. Kun sovellusta käytetään samalla laitteella eri käyttäjätunnuksella, sovellus lisää uuden käyttäjän asetustiedostoon. Tällä tavoin kunkin käyttäjän asetukset pysyvät tallessa seuraavaa käyttöä varten. Käyttäjän asetukset olisi voinut hakea myös palvelimelta, jolloin ne olisivat olleet samat PC-asiakassovelluksen kanssa. IOS-sovellukseen päätettiin kuitenkin tehdä erilliset asetukset, koska käyttäjät haluavat varmasti rajoittaa näytettävien projektien määrää pienellä iPhonen näytöllä.

7.6 Sovelluksen testaus

Sovelluksen testaus hoidettiin pääsääntöisesti käyttäjätestauksena mobiililaitteella sekä simulaattorilla. Sovellusta testattiin koko ajan kehityksen edetessä, kun jokin suurempi kokonaisuus oli saatu valmiiksi. Tämä osaltaan myös johtui siitä, että kokemattomuuden takia sovellusta oli testattava tiheään toimivuuden tarkistamiseksi.

Sovellusta kokeiltiin ensimmäisen kerran oikealla puhelimella vasta, kun suurin osa sovelluksen päätoiminnallisuuksista oli jo tehty. Sitä ennen testausta tehtiin ainoastaan iOS-SDK:n mukana tulleella iPhone-simulaattorilla. Yllätykseksi simulaattori noudatti yllättävän tarkasti oikean laitteen toiminnot, eikä muutoksia tarvinnut tehdä juurikaan siirryttäessä testaamaan oikealla laitteella. Ainoastaan sovelluksen sammuttaminen ja sen jälkeen uudelleen käynnistäminen ei toiminut simulaattorissa. Sovellus jatkoi aina samasta tilasta kuin se oli ollut sammutettaessa, vaikka sen olisi pitänyt avautua kirjautumisruutuun uudelleen käynnistettäessä. Tämä kuitenkin toimi oikein oikealla laitteella testattaessa, joten se ei tuottanut lisätyötä.

Testaukseen käyttämäni päätelaite oli ensimmäinen Suomessa myytävä iPhone 3G, johon oli asennettu uusin iOS 4 -käyttöjärjestelmä. Tämä ei kuitenkaan tukenut moniajoa, joten moniajotoiminnan testaamiseen käytettiin erään työntekijän omaa iPhone 4-matkapuhelinta. Yllättäen moniajo toimi ihan oikein testattaessa, vaikka sovellukseen ei ollut rakennettu mitään moniajon mahdollistavia toimintoja. Sovelluksen moniajon tarkoitus oli muistaa tilansa laitettaessa se taustalle, jolloin käyttäjän ei tarvitse kirjautua uudelleen jatkaessaan tuntien kirjaamista. Tämän toiminnon hoiti suoraan iOS-käyttöjärjestelmä, jolloin se ei vaatinut mitään toimenpiteitä ohjelmakoodissa toisin kuin aikaisemmin ajateltiin. Tätä edesauttoi myös se, että sovelluksen tiedonsiirrosta tehtiin synkroninen, jolloin sovellus ei kommunikoi serverin kanssa ilman että käyttäjä tekee aloitteen operaation toteuttamiseksi.

Toinen testausmenetelmä oli iOS-SDK:n mukana tuleva Instruments-työkalu, jolla voidaan analysoida sovelluksen resurssien käyttöä ajon aikana. Se olikin hyvin merkittävä työkalu erityisesti muistivuotojen jäljittämisessä. Koska oma kokemukseni koostuu pääsääntöisesti korkeamman tason ohjelmointikielistä kuten PHP:sta, muistinhallinta vaati huomattavan paljon opiskelua ja istruments-työkalua käytettiin paljon koko sovelluksen testauksen ajan. Objective-C-kielessä on niin sanottu puoliautomaattinen muistinhallinta, eli jotkin oliot vapauttavat automaattisesti varaamansa muistin ja jotkin pitää vapauttaa erikseen. Tämän takia testauksessa kului paljon aikaa siihen, että kahlattiin ohjelmistoa läpi ja laskettiin olioon tehdyt viittaukset sekä viittausten vapautukset.

Loppujen lopuksi testaus oli hyvin tärkeässä osassa sovelluksen kehitystä. Sen lisäksi, että testauksella havaittiin sovelluksen tekniset virheet ja puutteet, löytyi myös uusia näkökulmia toimintojen toteuttamiseen. Ahkeran testauksen ansiosta sovelluksesta saatiin hiottua sujuvasti toimiva kokonaisuus, mikä oli yhtenä isona tavoitteena sovellusta suunniteltaessa.

8 LOPPUSANAT

Opinnäytetyöprojektin tavoitteena oli suunnitella ja toteuttaa TimeSheet-asiakassovellus iOS-käyttöjärjestelmälle sekä tutkia iOS-sovellusten tarjoamia mahdollisuuksia Eneris Solutions Oy:n myöhempiä projekteja varten. Lopputuloksena nämä molemmat tavoitteet toteutuivat ja saatiin aikaiseksi toimiva sovellus sekä hyvin kattava kokemus siitä, millaista on kehittää sovelluksia iOS-alustalle. Tämän lisäksi sovellus antaa Eneris Solutions Oy:lle mahdollisuuden kokeilla täysin uutta bisnesmallia jakamalla ja markkinoimalla sovellusta Applen kauppapaikan AppStoren kautta.

Tämän projektin aikana iOS-asiakassovelluksesta jätettiin monia ominaisuuksia pois ajan puutteen vuoksi. Sovellusta mahdollisesti jatkokehitetään lisäämällä ylläpitäjälle enemmän toimintoja ja tekemällä sovellus tukemaan useampia käyttökieliä.

IOS-asiakassovellus suunniteltiin alusta lähtien toimimaan myös iPad-tablet-tietokoneella. Sovellusta voitiin testata vain simulaattorilla, joten se ei anna varmaa tietoa siitä, toimiiko sovellus oikein tablet-laitteella. Toiseksi sovellus ei osaa hyödyntää iPad-laitteen suurempaa näyttöä, vaan se näkyy näytön keskellä saman kokoisena ikkunana kuin iPhone-näytöllä näkyisi. Tässä olisikin hyvä jatkokehitysmahdollisuus toteuttaa käyttöliittymä siten, että se hyödyntäisi kunnolla iPad-laitteen suurempaa näyttöä.

Aikaisempi kokemukseni Applen tuotteista oli varsin vähäinen ja iOS-alustalle ohjelmointi oli täysin tuntematonta. Kuullessani ehdotuksen projektin aiheesta, se tuntui hyvin kiinnostavalta opiskelupohjaisen luonteen sekä ennestään minulle tuntemattoman alustan vuoksi. Iso osa projektiin käytetystä ajasta kuluikin testaillessa erilaisia vaihtoehtoja toteuttaa jokin toiminto sekä opiskellessa alustan toimintaa. Lopputuloksena syntyi kuitenkin toimiva sovellus, joka täydentää hyvin TimeSheet-tuoteperhettä.

LÄHDELUETTELO

1. IPhone. 2011. Saatavilla: http://fi.wikipedia.org/wiki/IPhone. Hakupäivä 24.7.2011.

2. iPhone 4. 2011. Saatavilla: http://fi.wikipedia.org/wiki/IPhone_4. Hakupäivä 24.7.2011.

3. iOS. 2011. Saatavilla: http://fi.wikipedia.org/wiki/Ios. Hakupäivä 24.7.2011.

4. Table View Programming Guide for iOS: Navigating a Data Hierarchy With Table Views. 2011. Saatavilla: http://developer.apple.com/library/ios/#DOCUMENTATION/UserExperience/Conceptual/TableView_iPhone/TableViewAndDataModel/TableViewAndDataModel.html. Hakupäivä: 24.7.2011.

5. Objective-C - Wikipedia. 2011. Saatavilla: http://fi.wikipedia.org/wiki/Objective-C. Hakupäivä 14.7.2011.

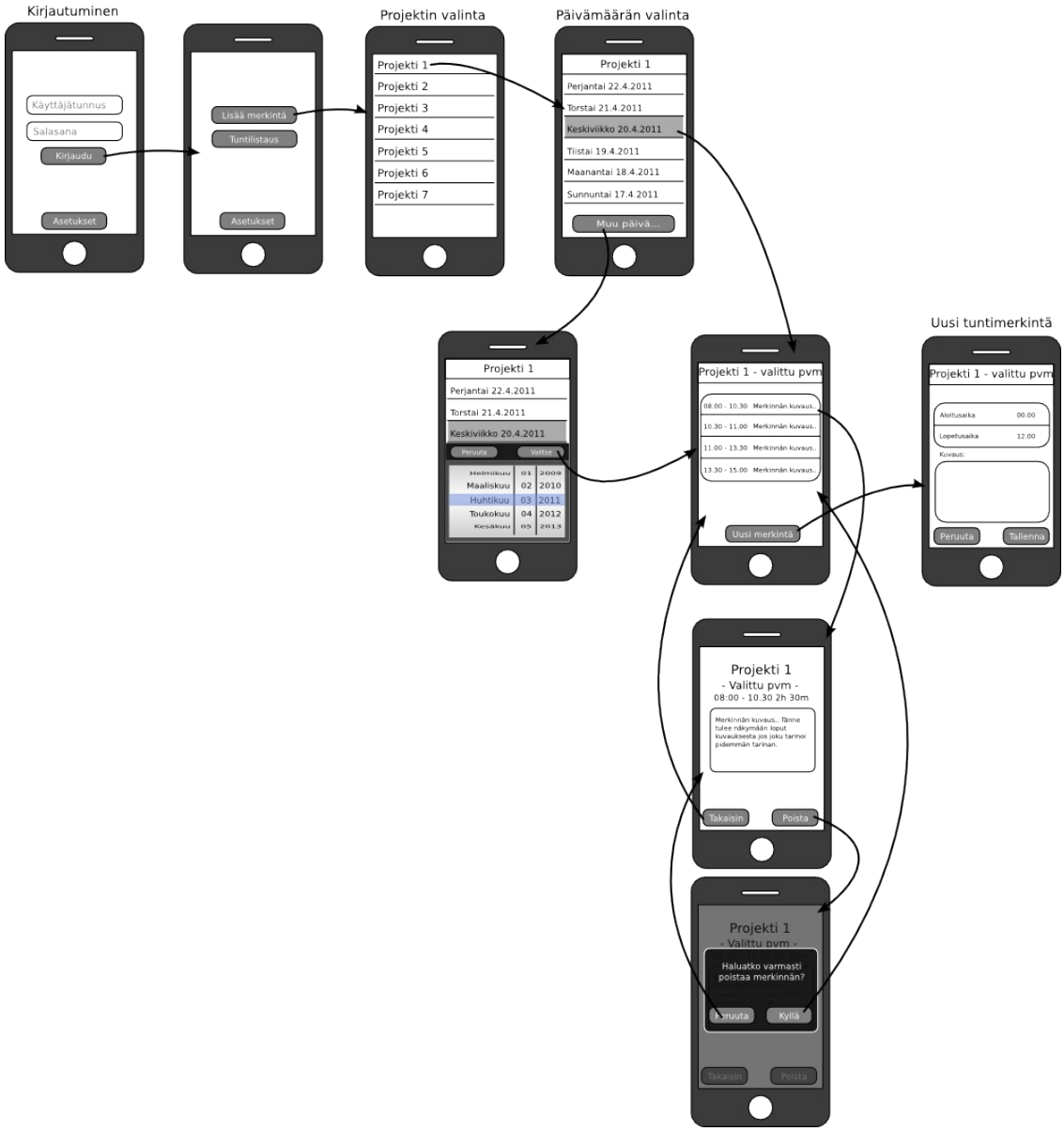
6. JSON. Saatavilla: http://www.json.org/. Hakupäivä 19.7.2011.

7. Xcode. 2011. Saatavilla: http://en.wikipedia.org/wiki/Xcode. Hakupäivä 19.7.2011.

8. Integrated development environment. 2011. Saatavilla: http://en.wikipedia.org/wiki/Integrated_development_environment. Hakupäivä 19.7.2011.

9. NetBeans. 2011, Saatavilla: http://en.wikipedia.org/wiki/NetBeans. Hakupäivä 24.7.2011.

Käyttöliittymäkaavio



CustomDatePicker-luokan koodilistaus

```
//
//  CustomDatePicker.m
//  HourTrackingIOS
//
//  Created by Aki Syri on 5/27/11.
//  Copyright 2011 Eneris Solutions. All rights reserved.
//

#import "CustomDatePicker.h"

@implementation CustomDatePicker

- (id)initWithFrame: (CGRect)frame {
    if (self == [super initWithFrame:frame]) {
        for (UIView * subview in self.subviews) {
            subview.frame = self.bounds;

        }
    }
    return self;
}

- (id) initWithCoder: (NSCoder *)aDecoder {
    if (self == [super initWithCoder: aDecoder]) {
        for (UIView * subview in self.subviews) {
            subview.frame = self.bounds;
        }
    }
    return self;
}
@end
```