jamk.fi

iOS-ohjelmistokehitys

Suunnitelmasta julkaisuun

Jarno Virtanen

Opinnäytetyö
Toukokuu 2015

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Virtanen, Jarno Henrik	Opinnäytetyö	21.05.2015
	Sivumäärä	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: (X)

Työn nimi
iOS ohjelmistokehitys:
Suunnitelmasta julkaisuun

Koulutusohjelma
Ohjelmistotekniikka

Työn ohjaaja(t)
Matti Mieskolainen

Toimeksiantaja(t)
Iwa Labs Oy

Tiivistelmä

Opinnäytetyön tarkoituksena oli saada kuvaus ohjelman kehityksen kulusta, jota voitaisiin sitten soveltaa toimeksiantajan ohjelmistokehitysprosessin parantamiseksi. Tarkoituksena oli paikantaa heikkouksia ja vahvuuksia projektin kulussa ja kirjata ylös askel askeleelta matka suunnitelmasta valmiiseen ohjelmaan. Opinnäytetyö sisältää sen lisäksi kuvauksen ohjelmiston kehityksessä syntyneistä ongelmista ja niiden ratkaisuista.

Tehdyn ohjelman tilasi Iwa Labsin asiakas. Tässä projektissa Iwan osuus oli kehittää mobiiliohjelmisto asiakkaan toimittamaa rajapintaa vastaan. Projektissa toimi vain yksi ohjelmistokehittäjä, joten ohjelmistokehittäjä otti kaiken vastuun ohjelmiston toteutuksesta ja valmistumisesta.

Projekti valmistui hyvissä ajoin vaikka kehityksen aikana ilmenikin paljon ongelmia. Iwa Labsin asiakas oli tyytyväinen projektin lopputulokseen ja ohjelman toimivuus oli hyvä.

Avainsanat (asiasanat)
iOS, iPad, Apple, Ohjelmistokehitys, Ohjelmointi, X-Code

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

Description

Author(s)	Type of publication	Date
Virtanen Jarno Henrik	Bachelor's thesis	21.05.2015
		Language of publication: Finnish
	Number of pages	Permission for web publication: (X)

Title of publication
iOS software development:
From planning to release

Degree programme
Software engineering

Tutor(s)
Matti Mieskolainen

Assigned by
Iwa Labs Oy

Abstract

The purpose of this bachelor's thesis was to describe the development process of an iOS application. The strengths and weaknesses of Iwa Labs' software production process were found and written down so that Iwa could use this document to improve it. The problems encountered and their solutions to help in this matter were also recorded.

The software was ordered from Iwa Labs by a third party. In this project the task of Iwa Labs was to write a mobile application that uses the client's API. There was only one software developer assigned to the project who also took the responsibility for the development process described here in the thesis.

The project was finished in time despite many problems encountered during the development. The customer of Iwa Labs was satisfied with the software and it worked as specified in the requirements.

Keywords/tags (subjects)
iOS, iPad, Apple, Software development, programming, X-Code

Miscellaneous

Sisällysluettelo

Kuviot

Lyhenteet ja sanasto

Agile (ketterä)	Agile-ohjelmistokehitys on ohjelmistokehitystä, jossa asetetaan yhteistyö asiakkaan kanssa ja toimivat ohjelmistot sopimusten ja dokumentaation edelle. Agile-metodit yleensä ovat nopeampia reagoimaan muutoksiin ja toimivat hyvin jatkuvassa kehityksessä.
API	Ohjelmistorajapinta, käyttää sovittuja sääntöjä, joiden mukaan ohjelmat kommunikoivat keskenään.
App	Mobiililaitteille kehitty ohjelmisto.
Apple	Apple Inc. yritys, joka vastaa Applen tuotteiden ja käyttöjärjestelmien valmistuksesta, jotka toimivat tämän opinnäytetyön alustana. http://www.apple.com
App Store	Applen hallinnoima ohjelmistokauppa, jossa opinnäytetyöprojekti asiakkaalle julkaistiin.
Basecamp	Verkkopalvelu, jossa yritykset tai yhteyshenkilöt voivat koordinoida
Cocoa Touch	iOS-käyttöliittymä SDK tunnetaan tällä nimellä. (ks. SDK)
Developer portaali	Applen verkkosivu, jonka kautta hallitaan kehitystunnuksia sekä profiileja, joita käytetään Applen tuotteille valmistettujen ohjelmien julkaisuissa. http://developer.apple.com

iOS	Applen ylläpitämä käyttöjärjestelmä, jota käytetään iPhone, iPad, iPod ja iWatch tuotteissa.
iPad	Sormitietokone (tabletti), jolla tässä opinnäytetyössä kuvailtu ohjelma suoritetaan. Käyttää Applen iOS käyttöjärjestelmää.
iTunes connect	Verkkosivu, jossa hallinnoidaan julkaisuja, päivityksiä ja raha-asioita koskien Applen laitteille tehtyjä ohjelmistoja. http://itunesconnect.apple.com
Mac	Macintosh, yleisnimitys Applen valmistamille tietokoneille.
Objective-C	Ohjelmointikieli, jota ohjelman tekemisessä käytettiin. Pohjautuu C-ohjelmointikieleen.
Ohjelmointikieli	Kieli, jolla kuvataan toimenpiteitä, joita tietokoneen halutaan tekevän.
OSX	Käyttöjärjestelmä Applen tietokonelinjastossa.
Palvelin	Tietokone, jonka tehtävänä on ylläpitää joitain palveluita, joita ihmiset tai ohjelmat voivat käyttää.
SDK	Software Development Kit on tarvikkeet ohjelmistokehittäjälle ohjelmiston julkaisuun jollain toisen osapuolen alustalla. Sisältävät kirjastot, joita käyttäen ohjelmistot voidaan rakentaa.

Swift	Applen kehittämä ohjelmointikieli, jota suunnitellaan korvaamaan Objective-C.
Tietokanta	On pysyvä tallennuspaikka ohjelman tiedoille.
Versionhallinta	Ohjelmistot, jotka on suunnattu projektien parissa yhteistyötä tekeville henkilöille. Sisältävät tarvittavat työkalut käyttäjien tekemän työn synkronointiin ja hallintaan jäsenten koneilla.
Windows	Microsoftin valmistama käyttöjärjestelmä koti- ja työkäyttöön. Windows sisältää myös kauppapaikan, jossa voi ostaa kolmannen osapuolen tekemiä ohjelmistoja.
XCode	Kehitysympäristö, jota käyttäen ohjelma koodattiin, käyttöliittymä ja tietokanta suunniteltiin.

1 Toimeksiantaja ja kehitysympäristö

1.1 Toimeksiantaja Iwa Labs Oy

Iwa Labs on pienehkö Helsingistä lähtöisin oleva ohjelmistoyritys, joka on perustettu vuonna 2009. Iwa on tasaisesti laajentanut toimintaansa monikulttuuriseksi yritykseksi viime vuosina. Tällä hetkellä Iwalla on toimisto Helsingin lisäksi myös Thaimaassa Khon Kaenin kaupungissa. Thaimaan lisäksi Iwalla on myyntihenkilöstöä myös Saudi-Arabiassa sekä Amerikassa. Iwan palveluksessa on tällä hetkellä 23 työntekijää, joista 15 on ohjelmistokehittäjiä.

Iwan tarjontaan tällä hetkellä kuuluvat räätälöidyt verkkopalvelut, iOS-, Android- ja Windows Phone -sovellukset, päätelaiteriippumattomat selainpohjaiset web app -sovellukset sekä mobiilioptimoidut HTML5-verkkopalvelut ja -sivustot. Käytetyimpiä kehityskieliä ovat Rails, PHP, JavaScript, HTML, Java, C# ja Objective-C. Iwa Labs suosii ketterää ja oppivaa ohjelmistokehitystä, jossa asiakkaat ja ratkaisun loppukäyttäjät otetaan mukaan kehitysprosessiin mahdollisimman aikaisessa vaiheessa. Laadunvarmistuksen peruspilareita ovat sitoutuneet ja ammattitaitoinen henkilöstö, toistettavat työmenetelmät, kurinalaiset laatukäytännöt sekä testivetoinen ohjelmistokehitys.

> *Iwa Labs suunnittelee ja toteuttaa asiakkailleen web-pohjaisia ratkaisuja ja ohjelmistoja. Tarjoamme räätälöidyt verkkopalvelut, iOS-sovellukset, Android- ja Windows Phone-sovellukset, päätelaiteriippumattomat selainpohjaiset mobiili web - ja web app -sovellukset sekä mobiilioptimoidut HTML5-verkkopalvelut ja -sivustot. (Iwa Labs Oy 2015)*

Iwan asiakkaita ovat sekä yritykset että julkishallinnon toimijat, muun muassa Metsätalouden kehittämiskeskus TAPIO, Geologian tutkimuskeskus, Kirkkohallitus ja Kansainvälisen liikkuvuuden ja yhteistyön keskus CIMO. Iwa Labsin tavoitteena on toimittaa paras ratkaisu jokapäiväisen liiketoiminnan tarpeisiin, ja näissä ratkaisuissa korostuvat ennen kaikkea positiivinen palvelu- ja käyttökokemus sekä korkea laatu.

Olen työskennellyt Iwan palveluksessa jo useita vuosia. Aikaisemmin olin työkomennuksella Thaimaan toimistolla. Siellä minun tehtäviini kuului ohjelmistokehitys ja suunnittelu, työntekijöiden koulutus sekä projektin vastuuhenkilönä toimiminen. Nykyään työskentelen pääasiassa tarvepohjalta tehden etätöitä projektien parissa, yleensä avustaen projektin pääasiallista kehittäjää. Minun roolini Iwa Labsin palveluksessa tässä projektissa oli ohjelmistokehittäjänä toimiminen asiakkaan tilaaman projektin parissa sekä ohjelmiston valmistuttua sen julkaiseminen App Storessa asiakkaan nimissä.

Asiakkaan sekä ohjelmiston oikeat nimet jätetään tässä opinnäytetyössä julkaisematta tietoturvan ja asiakkaan yksityisyyden suojaamiseksi.

1.2 Apple Inc.

Applen perustivat Steve Jobs ja Steve Wozniak 1. huhtikuuta 1976 alkujaan nimenä oli Apple Computer. Yritys aloitti toimintansa valmistamalla kotikäyttöön suunniteltuja tietokoneita, mutta alkuvaiheessa yrityksellä oli pulaa pääomasta, joka johti vaikeuksiin tuotantoon siirtymisessä. Saatujaan rahoittajia yritys lähti liikkeelle, ja ensimmäiset tunnetut Apple II-mikrotietokoneet julkaistiin vuonna 1977. Ensimmäinen graafisella käyttöliittymällä varustettu Apple-tietokone nimeltään Apple Lisa julkaistiin vuonna 1982, mutta se ei saanut laajaa suosiota hintansa ja vähäisten ohjelmistojen vuoksi. (Apple Inc. 2015)

Vuonna 1984 Apple lähti nousuun julkaistuaan ensimmäiset Macintosh-tietokoneet, jotka myivät hyvin ja saavuttivat kohtalaisen suosion. Macintosh-tietokoneiden myynti oli hiipumassa johtuen samoista syistä kuin Lisa-koneessa, kunnes ensimmäiset kohtuuhintaiset kotitulostimet julkaistiin ja kotitietokoneiden suosio lähti nousuun. Sisäisten valtataisteluiden, joissa perustajajäsen Steve Jobs lähti yrityksestä, jälkeen Apple Computers lähti kehittämään kalliimpia tietokonemalleja tehokäyttäjille.

Tämä strategia tuotti hyviä tuloksia aina 1989 vuoden joulumarkkinoihin asti, jonka jälkeen Apple julkaisi uudelleen myös edullisempia tietokonevaihtoehtoja. (Apple Inc. 2015)

Vuonna 1991 Apple julkaisi PowerBook kannettavan tietokoneen, jonka ulkonäkö on lähellä nykyään olevia kannettavia tietokoneita. Samana vuonna Apple julkaisi uuden version tietokoneidensa käyttöjärjestelmästä, jossa lisättiin värejä näytettävään käyttöliittymään. Tätä vuonna 1991 julkaistua käyttöjärjestelmää käytettiin Applen käyttöjärjestelmien pohjana aina vuoteen 2001 saakka. 1990-luvulla Applen myyntiä vähensivät myös nousevat Microsoft sekä UNIX ja OS/2 pohjaiset laitteet joita, valmistivat muun muassa Sun Microsystems. (Apple Inc. 2015)

Vuonna 1996 uuden toimitusjohtajan saattelemana Apple aloitti kauppaneuvottelut NeXT-yrityksen, jonka Steve Jobs perusti, sekä heidän valmistamansa NeXTSTEP-käyttöjärjestelmän ostamiseksi. Tämän kaupan mukana Steve Jobs palasi Applen palvelukseen vuonna 1997, kun kauppa viimeisteltiin. Vuonna 1998 Apple julkaisi ensimmäisen ”i-sarjan” laitteensa iMacin, jolle Apple rupesi yrityskauppojen seurauksena kehittämään isoa määrää ohjelmistoja. (Apple Inc. 2015)

Vuonna 2001 Apple avasi ensimmäisen jälleenmyyntiliikkeensä, ja myöhemmin samana vuonna julkaistiin iPod musiikkisoitin. iPod saavutti suuren suosion ja sitä myytiin yli sata miljoonaa kappaletta kuuden vuoden aikana. Vuonna 2003 Apple avasi iTunes kaupan, joka myi internetin ylitse ladattavaa musiikkia 99 senttiä kappale. iTunes kauppa oli integroitu iPod soittimen kanssa, ja se tavoitti näin laajan kuluttajakunnan. iTunes kauppa kasvoi nopeasti markkinajohtajan asemaan ja sai latauksia yli viisi miljardia kappaletta jo ennen 19. kesäkuuta 2008. (Apple Inc. 2015)

Macworld Expo-tapahtumassa vuonna 2007 Steve Jobs julkisti tiedon Applen nimen vaihdosta Apple Computersista lyhempään muotoonsa Apple johtuen yrityksen uudesta suuntauksesta kannettaviin laitteisiin. Samassa tapahtumassa julkistettiin myös

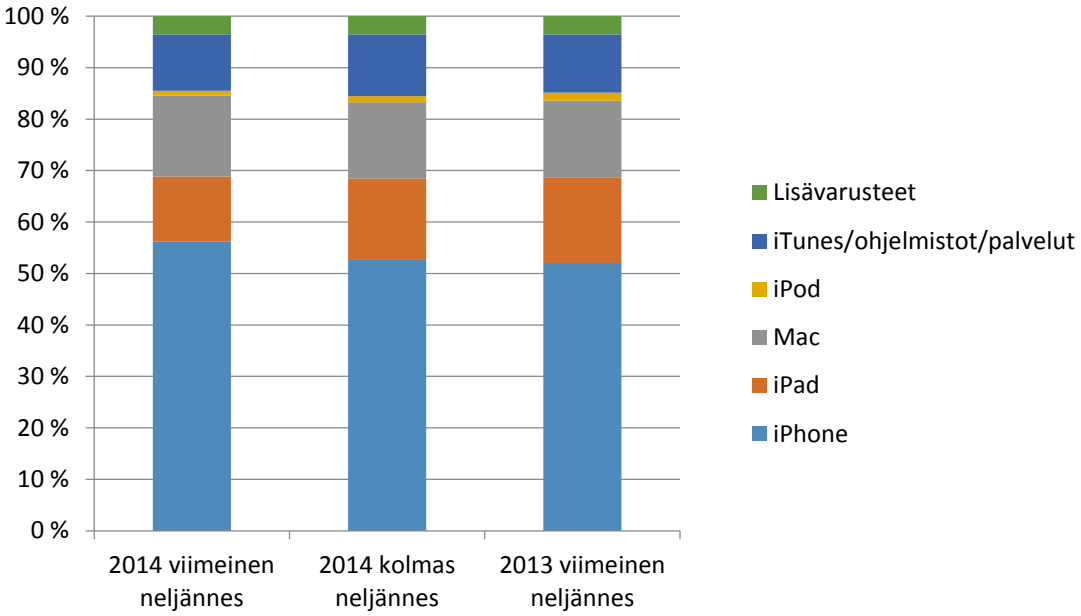
ensimmäinen iPhone-puhelinmalli sekä Apple TV. Seuraavana päivänä Applen osakkeet kävivät siihen mennessä suurimmissa lukemissaan. Myöhemmin samana vuonna Applen nettisivuilla oli kirjoitus, jossa Steve Jobs puhui mahdollisuudesta myydä musiikkia ilman käyttöoikeuksien rajoitusta, jos musiikin julkaisijat olisivat suostuvaisia teknologian poistamiseen. Ensimmäisenä levy-yhtiönä EMI lähti mukaan tähän uuteen malliin ja myöhemmin muut yhtiöt seurasivat EMI:n esimerkkiä. Viimein vuonna 2009 Apple julkaisi tiedotteen, jossa ilmoitettiin, että muutos toteutuu iTunes-kauppaan. Tämä teknologian muutos antoi mahdollisuuden kuunnella iTunes kaupasta ladattua musiikkia muillakin kuin iPod laitteilla. (Apple Inc. 2015)

Kesäkuussa 2008 Apple julkaisi App Storen, joka on kauppa kolmannen osapuolen ohjelmille iPhonelle sekä iPod Touchille, joka on uudempi kosketusnäytöllinen iPod malli. Kauppa saavutti suuren suosion, ja jo ensimmäisen kuukauden aikana sitä kautta myytiin yli kuusikymmentä miljoonaa ohjelmistoa ja se tuotti keskimäärin jopa miljoonan dollarin päivittäisen liikevaihdon. Lokakuussa 2008 Apple oli kolmanneksi suurin matkapuhelinten toimittaja maailmassa iPhonen saaman suuren suosion seurauksena. (Apple Inc. 2015)

Tammikuussa 2010 Apple julkisti ensimmäisen tabletti tuotteensa iPadin, joka käytti samaa käyttöjärjestelmää kuin iPhone, kehityksen. Saman käyttöjärjestelmän ansiosta monet iPhone-sovellukset olisivat suoraan toimivia iPadin kanssa. Tämän vuoksi iPadillä oli jo julkaisuvaiheessa kattava ohjelmistokatalogi. iPad myi ensimmäisen päivän aikana 300 000 kappaletta ja 500 000 kappaletta viikon loppuun mennessä. (Apple Inc. 2015)

Nykyään Apple on tunnettu kalliista, mutta tasokkaista laitteista ja minimalistisesta tai siististä suunnittelusta. Uusi puhelinten käyttöjärjestelmä on hyvä esimerkki tästä suuntauksesta. Applen uusin laitejulkaisu on iWatch, joka on älyrannekello, jonka voi yhdistää iPhonen tai iPadin kanssa. iPad laitteiden myynti on hieman laskussa, mutta iPad kehitys on silti tärkeä osa monen yrityksen kehitystä. iPad- ja iPhone-sovellukset

tavoittavat suuren määrän ihmisiä, ja monet yritykset tekevät sovelluksia näille tuotteille. Applen liikevaihto oli vuoden 2014 viimeisellä vuosineljänneksellä neljäkymmentä kaksi miljardia dollaria, josta suurin osa tulee iPhonen myynneistä. (ks. Kuvio 1)



Kuvio 1. Applen liikevaihdon jakautuminen (Q4 2014 Unaudited Summary Data 2015)

1.3 Objective-C

Objective-C on yleiskäyttöinen olio-ohjelmointikieli, jonka kehityksen aloittivat Brad Cox ja Tom Love 1980-luvun alussa. Objective-C pohjautuu C-ohjelmointikieleen ja yksi kehitysperiaatteista oli, että mihin tahansa Objective-C koodin joukkoon voisi lisätä C-koodia ja se toimisi normaalisti. Objective-C:n erottaa C:stä olioiden lisääminen sekä Smalltalk-ohjelmointikielen tyyppisestä syntaktista ja käytöstä. (Objective-C 2015)

Objective-C ohjelmointikieltä alkoi kattavasti käyttää vuonna 1988 NeXT-yhtiö heidän käyttöjärjestelmässään NeXTSTEP. Tämän seurauksena suurempi määrä ihmisiä teki töitä kielen parissa sekä sen avulla. Apple Inc. osti NeXT yhtiön vuonna 1996 ja aloitti

heidän Objective-C tuotteidensa käytön omassa Mac OS X käyttöjärjestelmässään, ja nykyään suurin osa Applen tuotteista pohjautuu tälle ympäristölle. (Objective-C 2015)

Vuonna 2014 kansainvälisessä kehittäjien konferenssissa (WWDC) Apple ilmoitti suunnitelmistaan korvata Objective-C kieli uudella Swift-ohjelmointikielellä, jonka kuvailtiin olevan Objective-C ilman C:tä. Tähän opinnäytetyöhön valittiin kuitenkin käytettäväks Objective-C ohjelmointikielen, koska vastaava ohjelmistokehittäjä on käyttänyt sitä jo muutaman vuoden ajan ja se sujuu häneltä hyvin. Tulevaisuutta ajatellen iOS ohjelmoijan pitää ehdottomasti opetella Swift ohjelmointikieli, mikäli aikoo jatkaa tai aloittaa iOS ohjelmoinnin parissa. (Objective-C 2015)

Objective-C syntaksi eroaa muista C kielien syntakseista näkyvimmin useiden hakasulkujen käytöllä. Esimerkiksi metodin kutsuminen Objective-C kielellä verrattuna C++ kieleen näyttää tältä:

```
[objekti metodi:argumentti];
```

Objective-C

```
objekti->metodi(argumentti);
```

C++

Ohjelmoinnin aloittamista Objective-C kielellä, on kuvailtu tuntuvan ensin hyvin vaikealta huomattavasti erilaisen syntaksin kirjoittamisen johdosta. Ajan kuluessa kuitenkin Objective-C kielen jäsentelystä sekä lievistä vapauksista saa hyötyä ja syntaksin kirjoittaminen helpottuu. Vaikka moni kehittäjä jääkin kaipaamaan joitain ominaisuuksia joita he olivat aiemmin käyttäneet, mutta kokonaistunne Objective-C:n kirjoittamisesta on kuvailtu olevan hyvin positiivinen.

1.4 iOS

iOD on Applen valmistama ja jakelema mobiililaitteiden käyttöjärjestelmä, joka on yksinoikeudella käytössä Applen valmistamissa tuotteissa. Käyttöjärjestelmästä tiedotettiin ensimmäisen kerran vuonna 2007 samaan aikaan ensimmäisen iPhonen

kanssa. Alussa kolmannen osapuolen ohjelmistot eivät olleet tuettuja, mutta maaliskuussa 2008 Apple julkaisi ensimmäisen SDK beeta version, joka mahdollisti natiivi sovellusten kehittämisen iOS päälle. Käyttöjärjestelmästä julkaistiin kolme versiota iPhone OS nimen alla ennen nimen vaihtoa iOS:ään. Nimen vaihto tuli tarpeelliseksi käyttöjärjestelmän levitessä muihinkin Applen mobiili päätelaitteisiin. Nykyään käyttöjärjestelmässä ollaan jo versiossa 8.3 ja versio 8.4 on tämän kirjoituksen aikana beeta versiossa. (iOS 2015)

Käyttöjärjestelmän työpöytä, tai SpringBoard, koostuu ohjelmistojen kuvakkeista. Nämä kuvakkeet on jaettuna sivuille joiden välillä voi raahata liikuttamalla sormea laitteen kosketusnäytön pinnalla sivulta toiselle. Käyttöjärjestelmä sisältää suoraan laitteen käyttämiseen tarvittavat ohjelmistot esimerkiksi iPhonen tapauksessa puhelinluettelon ja soittamiseen tarvittavan valintapaneelin. Käyttöjärjestelmä sisältää asetusvalikon, jossa laitteeseen liittyviä asetuksia, kuten salasanoja, hallinnoidaan. Lisäksi löytyy myös muutama perusohjelma arkikäyttöön, kuten karttaohjelma ja säätiedote, tämän lisäksi löytyy myös App Store josta pääsee ostamaan lisää ohjelmistoja. (iOS 2015)

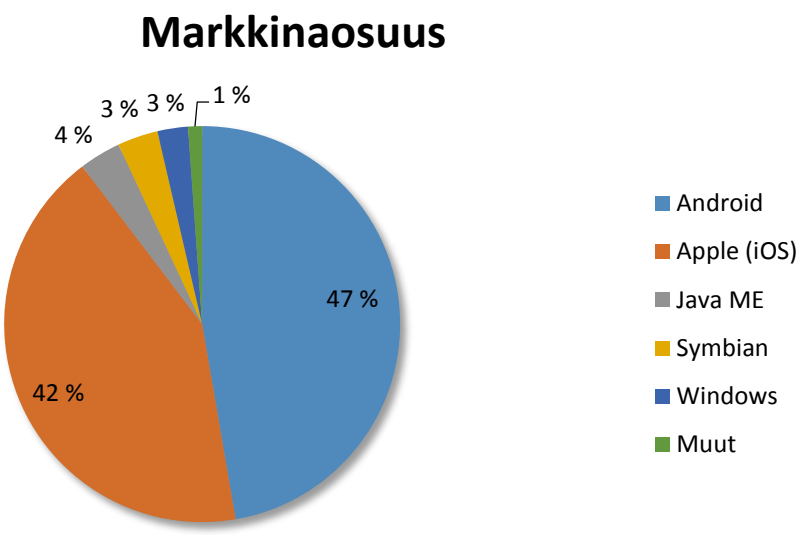
1.5 iPad

Ensimmäinen julkinen tiedotus koskien iPad laitteita oli 27. tammikuuta vuonna 2010 Applen lehdistötilaisuudessa San Franciscossa ja ensimmäiset langatonta lähiverkkoa tukevat laitteet päätyivät julkiseen kulutukseen 4. huhtikuuta samana vuonna. Ensimmäisen päivän iPad myyntiluvut nousivat kolmen sadan tuhannen kappaleen luokkaan ja kokonaisuudessaan iPad laitteita myytiin jopa viisitoista miljoonaa kappaletta. Seuraava iPad laite ”iPad 2” julkistettiin seuraavan vuoden maaliskuussa ja iPad tuotteita on julkaistu yhteensä yhdeksän kappaletta kahdessa eri mallissa. (iPad 2015)

iPad laitteet ovat muodoltaan hyvin samankaltaisia, mallien välillä eroavaisuudet koostuvat lähinnä paksuudesta ja ominaisuuksista. Uusimmat iPad laitteet ovat paksuudeltaan vain 6.1 millimetriä kun ensimmäinen iPad oli 13 millimetriä paksu. Samaan aikaan laitteen korkeus ja leveys ovat muuttuneet 243 ja 190 millimetristä 240 ja 170 millimetriin. Prosentuaalinen muutos on siis huomattavasti suurempi paksuudessa kuin leveydessä ja korkeudessa. Samaan aikaan paino on pudonnut ensimmäisten Wi-Fi + 3G iPadien 730 grammasta noin neljäkymmentä prosenttia uuden iPad Air 2 440 grammaan. Kaikissa laitteissa on vain kaksi porttia, toinen on liitin kuulokkeille tai muille äänilaitteille ja toinen on portti lataamista ja tietokoneeseen yhdistämistä varten. (iPad 2015)

Ensimmäisissä iPad laitteissa ei ollut lainkaan kameraa, vaan se lisättiin vasta iPad 2 versiossa. Vaikka iPad laitteet eivät sisällä puhelinominaisuuksia voi useilla ohjelmistoilla soittaa internetin ylitse puheluita ja uudemmat iPad laitteet sisältävät myös etupuolella olevan kameran videopuheluiden mahdollistamiseksi. iPad laitteet sisältävät 4 fyysistä näppäintä. Kotinäppäimestä, joka sijaitsee laitteen etupuolella näytön alla ja jonka toiminta vaihtelee painallusten pituuden ja nopeuden vaihteluiden mukaan, mutta yleisimpänä toimintona se palauttaa käyttäjän takaisin iOS työpöydälle tai kotinäkymään. Laitteessa on myös 3 muovista näppäintä laitteen sivuilla, jotka ovat virtanäppäin, äänenvoimakkuuden säätö sekä ohjelmisto näppäin jonka toimivuutta vai vaihtaa iOS käyttöjärjestelmässä. (iPad 2015)

Uusimmat iPad tuotteet iPad Air 2 ja kooltaan pienempi iPad Mini 3 ovat menestyneet melko hyvin vaikka viime vuosina Applen tablettien myynti on ollut laskussa. Kokonaisuudessaan silti iPad käyttäjiä on maailmalla miljoonia ja se on hyvin suosittu alusta niin kehittäjien kuin käyttäjienkin keskuudessa. Apple on tänä päivänä markkinajohtaja puhelin ja tabletti myynneissä valmistajittain, mutta kokonaisuudessaan häviää Android laitteille. (ks. Kuvio 2) Tästä suuresta suosiosta johtuen myös ohjelmistoja halutaan paljon iOS laiteille. Ja iOS kehittäjillä on hyvä kysyntä työmaailmassa. (iPad 2015)



Kuvio 2. Markkinaosuus käyttöjärjestelmittäin tableteissa ja puhelimissa (NetMarketshare 2015)

2 Opinnäytetyön tarve, idea ja tavoite

2.1 Tarve

Nykyään ohjelmistomaailmassa on alkanut ilmetä halua ohjelmistojen ostamisen ohella myös prosessikuvausten ostamiseen ja myymiseen. Yritykseltä saatetaan ostaa ohjelmiston lisäksi myös kuvaus siitä, miten ohjelmisto on valmistettu. Halutaan tietää, mitä ohjelmistoja on käytetty, miten suunnitteluprosessi on edennyt, mitä ohjelmistokehitysprosessia on käytetty sekä miten itse ohjelmointi on projektissa toteutettu. Tämä tulee usein ihan ohjelmiston tarpeen rinnalle, ja se saatetaan ostaa ohjelmiston jatkokehitystä ajatellen. Mutta joskus halutaan esimerkiksi startup-yritystä

varten prosessikuvaus, jotta yritys pääsee ohjelmistokehityksessä liikkeelle saaden valmiit ja varmennetut testaus- ja muut suunnitelmat omia projektejaan varten.

Tämä opinnäytetyö on kuvaus yhdestä projektista eikä sisällä tarkkaan harkittuja ja ennakkoon suunniteltuja kuvauksia, mutta se toimii lähtökohtana niin henkilökohtaiseen kuin yrityksen sisäiseenkin ohjelmistokehittämisen prosessin parantamiseen.

2.2 Idea

Idea lähti siitä, kun yritys ilmoitti, että olisi projekti tiedossa, jota voisin käyttää opinnäytetyössäni, mutta projekti ei itsessään sisältänyt mitään innovaatioivaa tekniikkaa tai teknologiaa. Suunniteltiin tehtävän kuvaus projektista ohjelmistokehittäjän näkökulmasta, päätöksistä sekä projektin kulusta. Tarkoituksena oli tutkia ohjelmointiprosessia ja löytää siitä parannettavaa, mutta se toimii hyvin myös Iwa Labsin eduksi ja auttaa paikantamaan mahdollisia vaikeuksia ohjelmistokehityksen kulussa.

2.3 Tavoite

Tämän opinnäytetyön tavoitteena on antaa lukijalle selvä kuvaus siitä, mitä iOS-ohjelman tekemiseen tarvitaan sekä miten ohjelma toimii ja miksi.

Tavoitteena oli saada, selkein yleisesti ymmärrettävin termein, välitettyä syyt ohjelmistokehittäjän päätöksiin projektin etenemisessä. Tätä varten pidettiin ohjelmointia suoritettaessa päiväkirjaa, johon merkittiin joka päivä tapahtumia ja etenemistä projektissa. Tätä päiväkirjaa apuna käyttäen aiotaan nyt ohjata lukija koko prosessin läpi, aloituksesta julkaisuun ja samalla kertoa ohjelmistokehittäjän kokemuksia ja näkemyksiä siitä, miten projekti eteni.

Lopuksi kerrotaan, miten tehty ohjelma toimii ja mitä vajavaisuuksia ja vahvuuksia siinä on. Samalla suoritetaan vertailua ohjelman eri osien välillä katsoen miten lopullinen toteutus eroaa alkuperäisestä suunnitelmasta ja mahdollisia syitä siihen miksi nämä eroavaisuudet ovat ohjelmaan syntyneet.

3 Projektista sopiminen ja sen aloitus

3.1 Sopimus

Iwa Labsilla oli aiempi tieto siitä, että projekti tarvitaan opinnäytetyötä varten. Toiveena oli projektin olevan maksimissaan kolme kuukautta pitkä, jotta ehdittäisiin tehdä projekti sekä raportti siitä kevään 2015 aikana. Marraskuussa 2014 sähköpostilla Iwa Labs ilmoitti, että sopiva projekti on löytynyt ja sen aloitusajankohta on 29.12.2014. Projektin piti valmistua yhdeksässä viikossa lähtien siitä päivästä kun Iwa Labsille toimitettiin tarvittavat API dokumentaatiot ja palvelin jota vastaan sovellusta voitaisiin testata. Iwan tehtävänä ennen sovelluskehityksen alkua oli tehdä käyttöliittymä suunnitelma, jonka näköiseksi ohjelmisto tehtäisiin.

Kuten asiakkaan kanssa oli sovittu, työt aloitettiin 29.12. täysipäiväisesti. Työsähköpostin sekä projektin basecamp työtilaa läpikäytäessä huomattiin, että käyttöliittymä suunnitelmia eikä api-dokumentaatiota ollut missään saatavilla. Projektin päälliköltä tarkistettiin oliko tarvittavia dokumentteja saatavilla, vai aloitettaisiinko työn teko vasta myöhemmin. Esimies oli sitä mieltä, että projekti tulisi aloittaa välittömästi vaikka asiakas ei ollutkaan toimittanut tarvittavia kuvauksia projektista, eikä hyväksynyt Iwa Labsin graafikon tekemää käyttöliittymäsuunnitelmaa.

3.2 Työtila

Basecamp verkkopohjainen työtila oli projektin pääasiallinen työtila, jonne piti lisätä kaikki dokumentaatio sekä asiakaskommunikaatio, jotta se olisi pysyvä paikka jossa projektin etenemistä pystyisi seuraamaan ja sieltä voisi aina löytää ohjelmiston valmistamiseen vaadittavat tiedot ja tiedostot. Pidettiin myös sisäistä tuntikirjanpitoa Basecamp työtilassa, jotta Iwa Labs pystyisi helposti arvioimaan etenikö projekti toivotulla tavalla, sekä sen kustannustehokkuutta.

3.3 Aloitus

Projektin parissa aloitettiin ensimmäiseksi päivittämällä käytetyn työkoneen käyttöjärjestelmä sekä oheisohjelmistot, sekä lataamalla uusin version kehitysympäristöstä ja SDK:sta.

Seuraavaksi piti ladata asiakkaan Windows applikaatio testaamista varten, mutta se jouduttiin lataamaan ohjelmistokehittäjän kotikoneelle, koska työkone on Macbook Pro jossa käyttöjärjestelmänä oli Applen yksityisomistuksellinen OSX. Tästä johtuen lähdettiin hakemaan ohjelmaa testausta varten Windows tietokoneelle, mutta jostain syystä sitä ei löytynyt Windows kaupasta. Projektipäällikkö ehdotti, että kokeiltaisiin asennusta jollain toisella laitteella, mikäli mahdollista. Vaihdettiin käyttämään toista Windows konetta, jonka seurauksena saatiin ladattua ohjelmisto testausta varten.

Ohjelma koostui kolmesta päänäkymästä. Ensimmäisessä näkymässä ladattiin palvelimelta rajapinnan välityksellä kohteet ohjelmaan. Tämän jälkeen näistä voitiin valita yksi kohde lähempään tarkasteluun, klikkaamalla kohdetta listassa. Seuraava näkymä sisälsi yksityiskohtaisia tietoja kohteesta ja listan kohteen sisäisistä alakohteista, joiden alle oli listattu alakohteiden yksityiskohdat. Kun näkymästä klikkaa jotain yksityiskohtaa, tulee näkyviin kolmas näkymä, jossa on muokkausmahdollisuus valitulle

yksityiskohdalle. Yksityiskohdan tallentamisen jälkeen ohjelma palaa edelliseen näkymään. Huomioitavaa on että yksityiskohtia muokatessa voi niihin tallentaa myös kuvia.

Testiohjelman lataamisen aikana lähettiin myös viestiä Iwa Labsin graafikolle, jossa pyydettiin keskeneräiset versiot käyttöliittymäsuunnitelmista, jotka toimitettiin hyvin nopeasti. Kun käyttöliittymäsuunnitelmat oli käyty läpi, aloitettiin testaamaan Windows sovellusta pitäen silmällä käyttöliittymäpiirroksia, etsien samoja ominaisuuksia sekä selvitettäessä niiden toimivuutta. Suurimpana erona Windows ohjelmiston ja käyttöliittymäpiirrosten välillä oli se että piirustuksissa ohjelma oli jaettu neljään osaan. Toisen ja kolmannen näkymän väliin oli lisätty vielä yksi välivaihe. Kohteen lisätietojen näkymässä ei ollutkaan suoraan listattuna yksityiskohtia, vaan piti ensin valita jokin alakohde jonka jälkeen yksityiskohdat listattiin. Suurin osa ominaisuuksista löytyi helposti ja löysin toimivuuden kaikille UI-suunnitelman elementeille. Käyttöliittymän suunnitelma näytti hyvin iOS tyyppiseltä ja väritys oli hyvä ja selkeä.

3.4 Dokumentaation tärkeys

Dokumentaation arvo helposti aliarvioidaan, ja tässäkin projektissa asiakkaalta olisi pitänyt pyytää paremmat tiedot ohjelmiston suunnittelusta ja käytöstä. Useita ominaisuuksia on vaikea löytää olemassa olevasta ohjelmasta, jos ei tiedä mitä etsii. Jopa yksinkertaiset ranskalaisilla viivoilla tehdyt dokumentit, varsinkin suunnitteluvaiheessa, helpottaisivat ohjelmistokehittäjän elämää huomattavasti. Parempi dokumentaatio olisi helpottanut myös graafikon käyttöliittymä suunnittelua.

3.5 Projektin tallentaminen ja toimitus

Projektin lähdekoodien tallennus ja varmennus tapahtui koko projektin ajan käyttäen versionhallinta ohjelmistoa nimeltään Git. Git on helppokäyttöinen ja ominaisuuksil-

taan rikas konsolipohjainen ohjelma, jolla versiointi voidaan tallentaa ja lähettää palvelimelle ja tarvittaessa synkronoida useamman käyttäjän kesken. Git sisältää myös ominaisuuden vanhojen versioiden tutkimiseen, joten ohjelman kehitystä pystyy helposti tutkimaan myös jälkikäteen.

Projekti piti lähettää Applen App Storeen valmistuttuaan Iwa labsin toimesta, sekä toimittaa asiakkaalle lähdekoodit joita ohjelmisto käyttää. Asiakkaan maksettua projektin kokonaisuudessaan, Iwa Labs luovuttaa kaikki immateriaalioikeudet asiakkaalle.

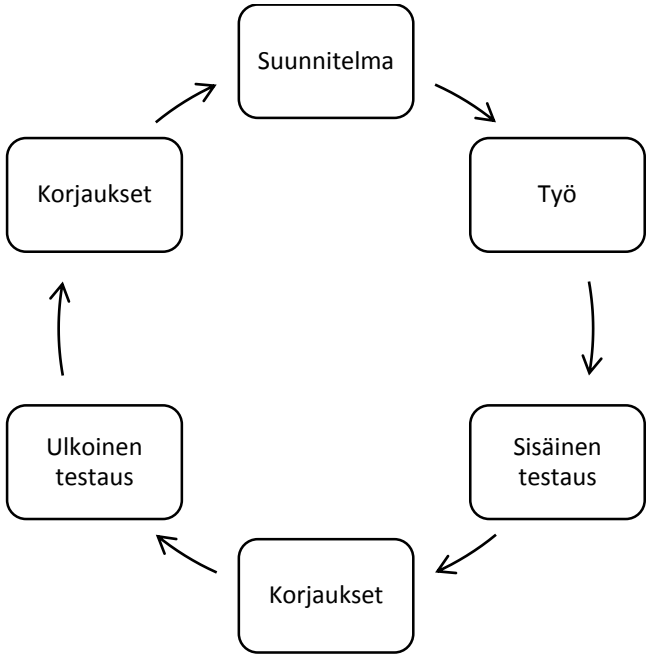
4 Suunnitelman laatiminen

4.1 Idea

Koska dokumentaatiota ei ollut, ei voitu varmistaa mitä kaikkea ohjelmaan tarvitsee tehdä. Tämän vuoksi päätettiin, että suunniteltaisiin ohjelma aluksi niin, että se on helposti mukautettavissa, jos ongelmia tai muutoksia ilmenee. Tässä vaiheessa ei vielä ollut API dokumentaatiota, ja toimittiin sillä oletuksella, että ohjelman käytössä pitää olla kokoaikainen yhteys internettiin, jotta ohjelma toimii oikein. Tämä päätös perustettiin kokemuksen, ja edellisten ohjelmistokehittäjän tekemien ohjelmien perusteella. Tästä syystä ei myöskään nähty tarvetta millekään pysyvälle tallennuspaikalle tai tietokannalle.

Iwa käyttää pääasiassa ketteriä ohjelmistokehitysmetodeja, eikä nähty mitään syytä rakenteellisen tai tavoitepohjaisen metodin käyttöön. Tarkoituksena oli iteroida ohjelmaa läpi kerros kerrokselta, jotta ohjelma saadaan nopeasti asiakkaan tarkasteltavaksi, ja he voivat antaa palautetta ohjelman toimivuudesta ja mahdollisista muutok-

sista tai muista korjauksista. Tästä syytä suunnitelma peilaa tätä ideaa, ja kehitys aloitettiin rungosta, jonka jälkeen tehtiin käyttöliittymä. Vasta kun käyttöliittymä olisi toimiva, jatkettaisiin kehitystä muihin alueisiin (ks. Kuvio 3)



Kuvio 3. Visuaalinen kuvaus suunnitelman yhdestä kierroksesta

4.2 Suunnitelma

Ohjelmiston suunnittelu aloitettiin käyttämällä kynää ja paperia. Tämä yhdistelmä tuntuu toimivan parhaiten, tämän kaltaista työtä tehtäessä. Se antaa vapauden piirtää ja kirjoittaa helposti ja nopeasti ideoita paperille, josta ne on myöhemmin helppo lukea ja tarkistaa. Oletetusta ohjelman kulusta piirrettiin vuokaavio, ja siihen merkittiin vaihe kerrallaan miten ohjelma etenisi.

Myös käyttöliittymäsuunnitelmaa käytiin läpi ja merkittiin ylös kohtia joissa oli epäselvyyksiä. Samalla katsottiin kuvista mallia, mitä luokkia tarvittaisiin ohjelmaa tehtäessä ja mitkä näkymät mahdollisesti voisi käyttää uudelleen. Kirjoitettiin myös hieman alustavaa suunnitelmaa palvelinkutsujen toteutuksesta ja tutkittiin miten se on

helpoin toteuttaa niin, että data pysyy synkronoituna palvelimen ja ohjelman välillä helposti.

Myöhemmin saatiin asiakkaalta linkki heidän palvelinkutsujen apusivustoon ja päästiin katsomaan hieman millaisia kutsuja tultaisiin käyttämään. Olemassa olevaa suunnitelmaa päivitettiin paperin toiselle puolelle, tehden tarkempia API kutsujen ja metodien suunnittelua. Kutsujen listaus sisälsi kymmeniä kutsuja, mutta asiakkaan viestin perusteella keskityttiin näistä kahteen, jotka vaikuttivat tärkeimmiltä.

4.3 Testaaminen

Ohjelmistokehittäjänä työhön kuuluu aina testata ohjelmaa koko kehityksen ajan, samalla kun sitä ohjelmoidaan, mutta itse tehdystä työstä on välillä hyvin vaikea löytää virheitä. Kun ohjelman on tehnyt, tietää miten sen pitää toimia, ja helposti testaa ohjelmaa vain niin, kuin sen on ajatellut toimivan. Tässä kuitenkin jää helposti monia virheitä ohjelmaan ja tästä syystä onkin tärkeää testauttaa ohjelmaa muilla henkilöillä. Tässä projektissa pääasiallisena testaajana ohjelmistokehittäjän lisäksi toimi projektin päällikkö, jolle ohjelmaa lähetettiin testattavaksi tärkeissä kehityksen vaiheissa.

Testaamisen helpottamiseksi, iOS laitteille oli kehitetty Testflight-verkkopalvelu, jonka kautta kehittäjät pystyivät jakelemaan kehitysversiota ohjelmistaan suoraan testaajien päätelaitteisiin. Tämä palvelu oli helppokäyttöinen ja ilmainen kaikille, mutta sen toiminta lopetettiin projektin puolivälin jälkeen. Apple siirsi testaamisen kolmannen osapuolen palvelusta omaan iTunesConnect verkkosivustoonsa ja integroi sen App Storeen lähetettävien ohjelmaversioiden rinnalle.

Tämä uusi palvelu tuotti pieniä sopeutumisvaikeuksia ohjelmistokehittäjälle. Suurin haitta kehityksen loppuvaiheessa oli palvelun joustamattomuus. Kun haluttiin lähet-

tää ohjelma ulkopuolisille testaajille, piti Applen omien työntekijöiden testata sitä ensin. Tähän testaukseen oli jonoa, jopa päiviä, ja se hidasti asiakkaan palautteen saamista huomattavasti. Jossain vaiheessa kuitenkin huomattiin, että jos versionumerointia ohjelmassa ei muuttanut, pystyi valitsemaan uuden vaihtoehdon testiversiota tehtäessä. Tämä uusi kohta oli valinta, jossa kysyttiin muutoksista ohjelmistoon. Jos vastasi, ettei ollut tehnyt suuria muutoksia, pystyi ohjelman lähettämään testaajille suoraan ilman Applen uutta hyväksyntää.

4.4 Ensimmäinen kierros

Ensimmäisen kierroksen suunnitelmana oli tehdä kaikki tarvittavat luokat tyhjinä toteutuksina sekä tehdä kaikki käyttöliittymä elementit pohjautuen keskeneräisiin käyttöliittymäsuunnitelmiin. Tässä toteutusvaiheessa ei olisi tarvetta keskittyä itse koodiin vaan ”suunnitella” ohjelma kehitysympäristössä. Käyttöliittymän ei tarvitsisi olla interaktiivinen vaan sen pitäisi näyttää oikealta, jotta asiakas voi hyväksyä ohjelman suunnan ja kertoa jos tarvitaan joitain muutoksia.

Kirjoitettiin ylös suunnitelmaan mitä Cocoa Touch komponentteja tulisi käyttää tämän käyttöliittymän toteutuksessa. Ohjelmistokehittäjälle tuli kysymyksiä siitä mihin suuntaan käyttöliittymän pitäisi rullata. Windows sovellus rullasi sivuttain ja käyttöliittymän asettelu kuvissa viittasi pystysuuntaan kulkevaan käyttöliittymään. Komponenttien saralla päädyin siihen tulokseen että CollectionView täyttäisi kaikki tarpeet ohjelman ulkoasun saralta. Ohjelmistokehittäjä kävi läpi muutaman käyttöohjeen tästä komponentista, koska se oli ohjelmistokehittäjälle uusi komponentti iOS ohjelmistossa.

Suunniteltiin myös hieman ohjelman sisäistä rakennetta, miten ohjelmassa liikutaan näkymästä toiseen ja missä järjestyksessä näkymät kannattaa asettaa navigointi pinoon. Päätös tehtiin, että ensimmäinen näkymä pinossa olisi aina alkunäkymä johon

ohjelma tulee ensimmäisen kirjautumisen jälkeen. Kirjautuminen tapahtuu vain harvoin ohjelman elinkaaren aikana ja se on helppo näyttää tarvittaessa. Ohjelmassa oli myös muutama komponentti, jotka eivät vaikuttaneet iOS omilta komponenteilta. Mietittiin myös miten nämä komponentit saisi helpoiten toteutettua, muuttamatta ohjelman yhtenäistä ulkoasua.

4.5 Toinen kierros

Asiakkaan rajapinta käyttöohjeesta ladattiin esimerkkipalautus, jonka palvelin antaa kun pyydetään kohteita palvelimelta. Tämä oli tarkoitus integroida ohjelmaan jotta voitaisiin simuloida rajapintaa, ilman että oikeasti tehtäisiin kutsuja sitä vastaan. Tämä on hyvä taktiikka kun tehdään kehitystä, varsinkin tässä projektissa, koska asiakkaan rajapinta ei ollut vielä valmiina projektia aloitettaessa. Tällä kierroksella on tarkoitus myös saada nappulat ja muut käyttöliittymä elementit interaktiivisiksi. Esimerkiksi tekstit vaihdettaisiin dynaamisiin teksteihin jotka luettaisiin testidatasta. Tämän tekeminen vaatii jo ohjelmointia ja metodien ja luokkien tekemistä, interaktiivisuuden saavuttamiseksi.

4.6 Kolmas kierros

Toinen kierros oli suunniteltu olemaan lievästi interaktiivinen runko ohjelmalle ja kolmannen kierroksen oli tarkoitus rakentaa sen päälle, tuoden mukaan käyttäjien ja muun datan tallentamisen ohjelmaan. Kolmas kierros oli suunniteltu jatkamaan toisen kierroksen toteutusta, kolmannen kierroksen jälkeen ohjelman oli tarkoitus olla valmis rajapintakutsuja lukuunottamatta.

4.7 Neljäs kierros

Viimeisenä ohjelmaan lisättäisiin rajapintakutsut. Tämä tarkoituksen mukaisesti on jätetty viimeiseksi sillä siitä löytyy eniten komponentteja johon ohjelmistokehittäjä

itse ei voi vaikuttaa tässä projektissa. Tämä on hyvä käytäntö jos on mahdollista saada esimerkkivastaukset ja kutsut ilman että niitä pitää oikeasti tehdä. Tämä poistaa ohjelmaa kehitettäessä internetyhteyksien hitauden ja mahdollisen puutteen. Ohjelma saadaan tehtyä kauttaaltaan valmiiksi ilman että kehitys hidastuu.

5 Ensimmäisen vaiheen toteutus

5.1 Toteutus

Ohjelman tekeminen aloitettiin luomalla uusi projekti ja antamalla sille osuvan nimi. Tarvittavat ohjelman kokoamiseen ja käyttämiseen vaaditut asetukset asetettiin kohdalleen. Kohdeversioksi ohjelmalle asetettiin iOS 8.0. Tämä tulee olemaan yksinomaan tabletti käytössä, joten sekin merkittiin asetuksiin. Kun ohjelma luodaan, syntyy siinä samalla monia tiedostoja ja perus tietorakenne, jonka päälle ohjelmaa on helppo lähteä kehittämään. Lisättiin myös omat kansiot ohjaimille, jotka kontrolloivat näkymiä, käyttöliittymäkomponenteille sekä apuluokille joita saatettaisiin ohjelman kehityksessä tarvita.

Iwa Labsin tekninen johtaja teki käytettäväksi vaaditun Git säiliön Iwan palvelimelle, jonne tultaisiin lähettämään lähdekoodi ja pitämään se muutoksien kanssa ajan tasalla. Heti sen valmistuttua lähetettiin sinne uusi ja puhtoinen projekti, jotta kehittäjällä olisi aina tilanne mihin palata, jos ohjelman kehityksen kanssa tulisi alussa joitain ongelmia.

Aiemmin luotuun kansioon lisättiin luokat näkymien ohjaimille, joita tiedettiin tarvittavan ohjelman kehityksessä. Luotiin myös omat ohjaimensa alkusivulle, kirjautumiselle, tietonäkymälle, valintaruudulle sekä muokkausnäkymälle. Myös ensimmäiset apuluokat lisättiin projektiin, esimerkiksi oma luokkansa väreille jotka graafikko oli valinnut ohjelmaan, jotta voitaisiin kutsua niitä tarvittaessa helposti.

```
+ (UIColor *)blackColour
{
    float i = 34.0/255.0;
    return [UIColor colorWithRed:i green:i blue:i alpha:1];
}
```

Seuraavaksi lisättiin tyhjiä näkymiä käyttöliittymiä varten, käyttämällä graafista käyttöliittymätyökalua joka oli liitettynä kehitysympäristöön. Käyttöliittymät suunniteltiin tiedostomalliin, jota kutsutaan nimellä Storyboard. Kaikki käyttöliittymätiedot voi lisätä tänne samaan tiedostoon ja se tekee helpoksi nähdä miten ohjelman ”tarina” kulkee. Aiemmin luoduille ohjaimille lisätiin näkymät ja ne liitettiin editorissa yhteen. Sitten näkymien välille lisättiin yhteydet, jotka näkyvät nuolina työympäristössä. Tässä vaiheessa jo pelkästään nimien ja nuolien avulla näki selkeästi minne ohjelma oli matkalla. Alussa myös yksi CollectionView komponentti lisättiin, ja myöhemmin poistettiin, jotta tätä uutta komponenttia pystyttiin paremmin testaamaan.

Ohjelman ajettiin simulaattori laitteelle, joka tuli kehitysympäristön mukana. Tämä simulaattori sisältää simuloidun version Applen iOS laitteista ja siinä voidaan valita näytön kokoja ja laitemerkkejä, sekä mitä iOS versiota siinä käytetään. Se sisältää myös funktiot laitteen kääntämiselle sekä nappuloiden painamiselle. Kameraa ja oikeita paikkatietoja simulaattori ei valitettavasti tue ja tästä tulee myöhemmin ongelma. Ohjelman ensimmäisellä käynnistyskerralla ei voitu vielä muokata mitään tai navigoida minnekään, mutta nähtiin juuri luotu aloitusnäkymä ja CollectionView komponentti jonka se sisälsi. Ohjelman suoritus pysäytettiin ja käytiin käsin lisäämässä CollectionView komponenttiin ”solun” joita näkymä näyttää ja ryhmittelee. Kun ohjelman käynnistettiin uudelleen, ilmestyi solu simulaattoriin. Tämän jälkeen ajettiin useita testejä solujen toimivuuden kanssa. Tarkistettiin miten eri määrät näkyivät ja miten osiot jakaantuivat. Tässä vaiheessa jo huomattiin että alkuperäisin layoutin tekeminen täysin samannäköiseksi tulisi olemaan hankalaa, koska osiot alkoivat aina uudelta riviltä, eli kahden osion alkaminen samalta riviltä tulisi olemaan kustannustehokkuuden vastaista rakentaa.

Loput tyhjät luokkaohjaimet lisättiin ohjelmistoon, esimerkiksi tyhjä API luokka, ja loput tyhjät näkymät. Tämän jälkeen lisättiin ensin suuria komponentteja ohjelmaan, ja tehtiin näkymien välillä liikkumisen mahdolliseksi. Kun kaikkien näkymien ulkoasu vastasi suurin piirtein käyttöliittymäsuunnitelmaa, lisätiin tarkempia yksityiskohtia, tekstejä ja nappuloita, näkymiin. Käyttöliittymän hiominen on yllättävän aikaa vievää työtä, kun näkymät yritetään saada menemää kuvapisteen tarkasti oikeaan kohtaan.

Käyttöliittymää tehtäessä pyritään aina käyttämään ”constraint” nimellä kulkevia ohjeita siitä miten käyttöliittymä pitäisi piirtää. Näihin kuuluu esimerkiksi tieto siitä kuinka monen näyttöpisteen päässä reunasta elementin pitäisi alkaa, elementtien etäisyys toisistaan näytöllä ja niin edelleen. Tämä on hyvin tärkeä osa iOS käyttöliittymäsuunnittelua sillä iOS laitteilla voi olla erilaisia kuvasuhteita. Nämä rajoitteet takaavat sen että ohjelmisto näyttää hyvältä missä tahansa laitteessa. Ne on myös hyvin tärkeitä, jos ohjelman pitäisi toimia molempina pysty- ja vaakasuuntaisena. Tässä ohjelmassa kyseistä vaatimusta ei ollut, vaan kaikki näkymät suunniteltiin käytettäväksi laitteen ollessa vaaka-asennossa.

Osan toisen vaiheen työstä tehtiin jo ennen ensimmäisen vaiheen valmistumista. Asiakaspalaveria varten. Palaverin jälkeen vielä hieman käyttöliittymään hiottiin ennen kuin jatkettiin toiseen vaiheeseen ohjelman tekemisessä.

5.2 Asiakaspalaveri

Ensimmäisen palaveri asiakkaan kanssa pidettiin kaksi ja puoli viikkoa kehityksen aloittamisen jälkeen. Tähän kokoukseen ohjelmistokehittäjä otti osaa etänä internetpuhelun välityksellä, koska kokous pidettiin Helsingissä. Projektin vastuuhenkilö oli kuitenkin paikalla henkilökohtaisesti. Iwa Labsin edustaja käytti puhelimensa nettiä, tietokoneessaan kun hän soitti minulle. Nettiyhteys kuitenkin katkesi kahdesti, palaverin jälkeen ihmeteltiin, että asiakas ei tarjonnut omaa verkkoaan kokouskäyttöön. Kokousta varten oli tehty listan kysymyksistä joita ohjelmistokehittäjällä oli, koskien

molempia käyttöliittymää sekä ohjelman toimintaa. Lista avattiin tietokoneella puhelun toisessa päässä samalla kun ohjelmistokehittäjä katsoi sitä omalta koneeltaan ja lista käytiin kohta kohdalta läpi. Samalla ohjelmistokehittäjä kirjoitti muistiinpanoja siitä mitä sovittiin, sekä lisäsi kysymysten perään vastaukset mitä asiakas antoi. Muutamaan kysymykseen ei asiakkaan edustaja juuri sillä hetkellä osannut vastata, mutta hän lupasi palata asiaan myöhemmin sähköpostilla.

Suurin osa kokouksen sisällöstä koski vain ohjelmistokehittäjän esittämiä kysymyksiä, ja oli melko suoraviivaista tiedon vaihtoa. Yksi poikkeus kuitenkin oli, jossain vaiheessa asiakas mainitsi, että ohjelman tulisi toimia ilman internet yhteyttä. Tämä tuli ohjelmistokehittäjälle suurena yllätyksenä, ohjelman tekemisessä oli edetty siinä uskossa että internetyhteys olisi pakollinen ohjelman käyttämisessä. Nyt kuitenkin selvisi että suurin osa suunnitelmista joita oltiin datan tallentamista ja rajapinta kommunikaatiota varten tehty, pitäisi tehdä uudelleen toiselta kantilta. Tämä pakotti siirron tietokantaan ohjelman taustarunkona ja tuotti huomattavasti enemmän töitä, kuin mitä olisi pitänyt tehdä jos olisi tiedetty, että ohjelman pitää toimia ilman internet yhteyttä.

Toinen hieman suurempi muutos, joka ohjelmaan piti tehdä, oli editointinäkymän asettelu. Sen sijaan että yksityiskohdan tila valittaisiin ja sitten painettaisiin tallenna näppäintä, asiakas halusi valintanäppäinten toimivan myös tallenna näppäiminä. Tämän seurauksena käyttöliittymään vapautui myös hieman tyhjää tilaa, joka pakotti asettelemaan muitakin elementtejä uudelleen. Valintanäppäimet siirrettiin tallenna nappulan tilalle ja valintanappuloiden toimintaa ja ulkonäköä muutettiin hieman, jotta olisi selvempää että ne ovat näppäimiä.

Näiden ja muiden pienempien asioiden valossa jota palaverissa tuli ilmi, oltiin entistäkin tyytyväisempiä siihen miten ohjelman koodaus oli suunniteltu. Kahden ja puolen viikon kohdalla oli saatu jo ensimmäisen vaiheen tehtyä, sekä jo jonkin verran

toista vaihetta. Melkein kaikki työ jota toiseen vaiheeseen oli siinä kohdassa tehty, korvattiin toisenlaisella toteutuksella myöhemmin projektin aikana.

5.3 Vaikeudet

Projektin ensimmäisen vaiheen ongelmat tulivat melkein kokonaisuudessaan käyttöliittymän suunnittelutyökalun oudosta käyttäytymisestä, sekä joidenkin muutosten epäjohdonmukaisista tuloksista. Ensimmäiset ongelmat kuitenkin tulivat puutteellisesta dokumentaatiosta, sekä keskeneräisistä käyttöliittymäsuunnitelmista.

Kun ohjelmaa ajettiin ensimmäistä kertaa simulaattorilla, automaattisesti oli valittuna iPad Air simulaattori, jonka erityisen suuren resoluution ansiosta virtuaalisen tabletin näytöstä mahtui ohjelmistokehittäjän työkoneen näytölle vain noin yksi neljäsosa. Simulaattorissa oli myös hieman oudosti näkyvät komponentit ja yläpalkki ei tullut näkyviin yläreunaan. Tarkoituksena oli kuitenkin käyttää eri simulaattoria vanhahkon työkoneen takia, ja kun simulaattori vaihdettiin iPad 2 versioon, ongelmat korjaantuivat ja työskentelyä pystyttiin jatkamaan. Tässä samalla mainittakoon että työkoneen oli erittäin hidas käyttää ja monessa tilanteessa jouduttiin odottelemaan pitkiäkin aikoja, jotta voitaisiin jatkaa kehitystä.

Oli myös ongelmia saada Cocoa Touch navigointi sekä työkalupalkit toimivaan halutulla tavalla. Navigointipalkki tuli välillä näkyviin vaikka se käyttöliittymäsuunnittelutyökalussa merkittiin piilotetuksi. Ongelman koitettiin ratkaista usealla eri tavalla suoraan käyttöliittymäsuunnittelutyökalussa, mutta jos navigointi piilotettiin ensimmäisestä näkymästä, katosi se niistä kaikista, ja jos se laitettiin näkyviin ensimmäiseen näkymään, ei se suostunut piiloutumaan enää myöhemmissä. Ratkaisu saatiin navigointi ongelmaan siten, että piilotettiin palkki koodissa suunnittelutyökalun sijasta. Ongelma työkalupalkin kanssa oli, ettei sitä saatu samannäköiseksi kuin navigaatiopalkkia, eikä nappuloita saatu helposti näyttämään oikeilta. Tämä ongelma ohi-

tettiin, eu niinkään korjattu, tekemällä näkymän yläreunaan palkki, joka näytti samalta kuin navigaatiopalkki. Se ei kuitenkaan ollut oikea palkki, vaan tyhjä näkymä, johon asetettiin taustaväri ja viiva alareunaan. Tämä saattaa tuottaa vaikeuksia ohjelman siirtämiseksi seuraavaan iOS versioon, jos palkkien ulkonäköä siinä vaihdetaan.

Käyttöliittymäsuunnitelmat oli tehty käyttäen enimmäkseen ”Lorem Ipsum” täytetekstiä, joka tuotti ongelmia selvittää, että mitä tekstiä oikeasti kuuluu minnekin käyttöliittymässä. Tämän ongelman tiimoilta kuitenkin pystyttiin onneksi kommunikoimaan Iwa Labsin graafikon kanssa ja satiin suurimpaan osaan kysymyksistä vastaukset, ja loput vastauksista tuli ensimmäisessä asiakaspalaverissa.

6 Toisen vaiheen toteutus

6.1 Toteutus

Tämän vaiheen toteutus aloitetiin tekemällä testi tiedosto, joka sisälsi asiakkaan rajapinta käyttöohjeesta otetun esimerkki vastauksen, sekä sen integroiminen metodiin API luokassa. Kutsumalla tätä metodia saatiin ohjelmaan kokoelma NSDictionary muodossa olevia kirjastoja, jotka koostuvat avaimista ja niihin liitetyistä arvoista. Kutsumalla kirjastosta metodia:

```
valueForKey:@”avain”
```

saatiin vastaukseksi tieto, joka avaimelle oli asetettu. Esimerkiksi: Kun haluttiin saada kirjastosta nimi kohteelle, kutsuttiin kirjastoa seuraavasti:

```
[kohde valueForKey:@”Nimi”]
```

tämä palautti kohteen nimen, ja tätä nimeä pystyttiin sitten käyttämään ohjelmassa, esimerkiksi näyttämällä sen käyttäjälle, jotta hän tietäisi mitä kohdetta hän on katselemassa. Seuraava vaihe oli yhdistää nämä tiedot tehtyyn käyttöliittymään asettamalla oikeat tiedot oikeisiin nimikkeisiin. Tämän kohta tehtiin jo ennen ensimmäistä

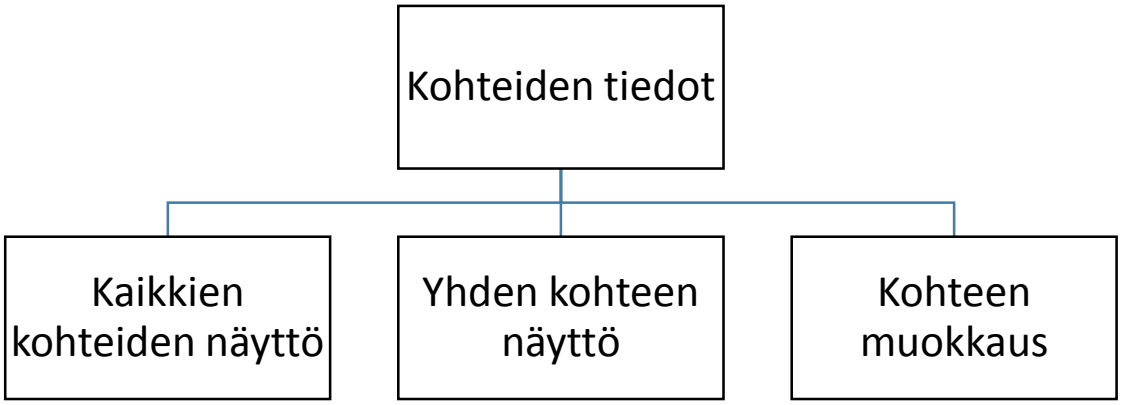
asiakaspalaveria, vaikka ei oltu vielä täysin valmiita käyttöliittymän kanssa. Tämä tehtiin projektipäällikön toivomuksesta, jotta asiakas näkisi että tieto kulkeutuu ohjelmistorajapinnasta näytölle.

Kun oltiin tyytyväisiä käyttöliittymään ulkonäköön, siirryttiin tekemään reaktiivisuutta käyttöliittymään, yhdistelemällä nappulat ja tekstikentät käyttöliittymästä koodin puolella oleviin muuttujiin. Tämän jälkeen siirryttiin metodien pariin, jotka keskittivät käyttöliittymän muokkaamiseen, sekä käyttäjän syötteiden hallinnoimiseen.

6.2 Muutokset ohjelman rakenteeseen

Jo ennen asiakaspalaveria huomattiin, että ohjelmassa kohteiden käsittely ei ollut hoidettuna optimaalisesti (*Kuvio 4*). Suurimmat ongelmat tulivat esille, kun muutosten tallentamisen jälkeen navigoi ohjelmassa takaisinpäin. Tehdyt muutokset eivät tulleet näkyviin muualla ohjelmassa, joten tietorakennetta piti hieman muuttaa. Taas käyttöön otettiin paperia ja kynä, sekä ryhdyttiin käsin suunnittelemaan erilaisia vaihtoehtoja. Otettiin esille kolme todennäköisimmin parasta ratkaisua ja piirrettiin näistä kaaviokuvat. Sitten listattiin ylös plussia ja miinuksia mitä eri toteutukset aiheuttaisivat ja näiden perusteella tehtiin päätös vaihtaa kohteiden tallentamisen keskitetysti omaan luokkaansa. Tämä oli hyvin vakaa ja monipuolinen ratkaisu jonka suurin miinus muihin vaihtoehtoihin nähden oli, että se oli työläin toteuttaa. Tässä vaihtoehdossa pystyin myös tekemään kaikkia kohteita koskevia metodeja helposti, implementoimalla staattisia metodeja tähän luokkaan (*Kuvio 5.*).

Kuvio 4. Lineaarinen toteutus



Kuvio 5. Uusi keskitetty toteutus

Asiakaspalaverin jälkeen, kun selvisi verkottoman tilan tarve, oltiin taas erityisen tyytyväinen valintaan jonka oltiin tehnyt jo ennen kuin sen tarpeellisuus tiedettiin. Pysyvän tallennustilan tarve oli kohtalaisen helposti korjattavissa sisällyttämällä se luokkaan, joka piti sisällään kohteiden tiedot, ilman muutoksia muualle ohjelmassa. Tietokannan käyttäminen oli itsestään selvä ratkaisu, mutta mitä tietokantaa projektissa kannattaisi käyttää, vaati tutkimustyötä. Tutkittiin internetistä erilaisista mahdollisuuksista, ohjelmistokehittäjä oli aikaisemmin käyttänyt SQLite tietokantaa muissa ohjelmissa, joten lähdettiin ensin tutkimaan sitä pääasiallisena vaihtoehtona. Tutkimustyön tuloksena kuitenkin huomattiin, että iOS kirjastot pitävät sisällään myös oman tietokantaobjektinsa nimeltään NSManagedObject, ja näiden objektien tallentaminen hoituu iOS SDK:n omilla metodeilla. Kehitysympäristö sisälsi myös graafisen työkalun tietokantarakenteen suunnitteluun. Projektiin piti lisätä SDK kirjasto CoreData, joka piti sisällään kaikki tietokannan toteutukseen tarvittavat metodit. Toisena etuna tässä implementoinnissa oli, että CoreData käytti tietokantanaan SQLite tietokantaa joka oli ohjelmistokehittäjälle tuttu entuudestaan. Ainoana ongelmana oli, että muu ohjelma käytti vanhaa NSDictionary rakennetta joten tehtiin funktiot, jotka muuttivat NSManagedObject kohteet NSDictionary muotoon, jotta muun ohjelman toimintaa ei tarvitsisi kirjoittaa uudelleen.

6.3 Vaikeudet

Käyttöliittymän tekemisessä oli suurimpana hankaluutena tekstilaatikko, joka oli näkyvillä kun tarkasteltiin yhtä kohdetta ohjelmassa. Se oli UICollectionView:n koko yläreunan kokoinen laatikko, jonka sisällä on tietoja tarkasteltavasta kohteesta. Näytettävät tiedot olivat aina vakioita, mutta niiden sisältöä ei ollut kokorajoiteltu mitenkään. Haasteena oli siis luoda korkeudeltaan dynaaminen näkymä, joka asetteli sisältönsä kolumneihin näkymän leveyden mukaan. Ensimmäinen kohta mitä piti ohjelmaan tehdä, oli otsikkojen ja kirjaston avainten yhdistäminen pareiksi. Tämän tehtiin eritoten siksi että haluttiin käydä kaikki mahdolliset arvot läpi silmukassa, eikä haluttu mahdollisten muutosten ja korkeusvaihteluiden takia koodata mitään kiinteitä arvoja ohjelmaan.

```
NSMutableDictionary *whatToDisplay = [NSMutableDictionary new];
[whatToDisplay setValue:@[@"osoite", @"nimi", @{@"Postal": @"postinumero",
@"City":@"postitoimipaikka"}] forKey:@"kohde"];
```

Jos käymme tätä koodia hieman läpi rivi riviltä, niin koodin tarkoitus ehkä selviää hieman. Ensimmäisellä rivillä näkyvä NSMutableDictionary on sama kuin aikaisemmin puhuttu NSDictionary, tässä Mutable avainsana kertoo kirjaston olevan muokattava, joka mahdollistaa metodin setValue: forKey: käyttämisen kirjaston yhteydessä. Tämä on vastametodi aiemmin mainitulle valueForKey: metodille ja asettaa arvon kirjastoon annetulle avaimelle. Vasemmalla puolella määrittelemme uuden muuttujan joka on kirjasto tyyppiä ja oikealla puolella käytämme new avainsanaa luodaksemme uuden tyhjän kirjaston. Toinen rivi näyttää sekavalta, mutta muuttuu paljon helpommaksi ymmärtää, kun osaa Objective-C syntaksia. Merkintä @[] luo uuden tietueen, jonka arvot erotellaan pilkulla, Objective-C perus tietue on tyypiltään NSArray, ja @{} luo uuden NSDictionary kirjaston. Teksti ennen kaksoispistettä kertoo avaimen ja kaksoispisteen jälkeen tulee tallennettava arvo. Kirjaston tällaisessa luomismallissa avain ja arvo parit erotetaan toisistaan pilkulla. Tämä toinen rivi sisältää sisäkkäisiä asetuksia tietueessa, jotta saataisiin useampi arvon yhden otsikon alle. Sisäpuolella oleva kirjasto on tehty, jotta kirjaston sisällä olevat arvot menisivät samalle riville. Ja käytin tässä kirjastoa tietueen sijasta, koska jollekin riville saatettiin tarvita kiinteätä

tekstiä ja kirjaston otsikoinnilla pystyin erottelemaan rivityypit toisistaan. Tämän koodin alla lisättiin kirjastoon noin 10 riviä lisää arvoja.

```
float usableWidth = collectionView.bounds.size.width -  edgeInsets.left -
edgeInsets.right;
float rows = ( floorf(usableWidth / 320.0) == 0.0 ? 1.0 : floorf(usableWidth /
320.0) );
int columnWidth = floorf(usableWidth/rows)-COLUMNMARGIN+10;
```

Seuraavaksi laskettiin kuinka paljon tilaa oli jokaiselle kolumnille tekstiä. Tässä lasketaan ensin käytettävissä oleva leveys, joka sitten jaetaan osiin käyttäen yhden UICollectionView solun kokoa. Ja lopuksi käytettävä leveys jaetaan kolumnien määrällä ja saadusta koosta poistetaan tyhjä tila joka jää kolumnien ympärille. Seuraavaksi käydään silmukassa läpi kaikki tiedot ja tehdään näistä käyttöliittymäkomponentteja jotka tallennetaan pareittain tietueeseen. Tästä saadaan myös kokonaiskorkeus kaikille komponenteille, joista lasketaan keskimääräiskorkeuden kolumnille.

```
for (int i=0;i<displayGroups.count;i++)
{
     UILabel *keyLabel = [[displayGroups objectAtIndex:i] objectAtIndex:0];
     [keyLabel setFrame:CGRectMake(MARGIN+((COLUMNMARGIN+columnWidth)*column),
     currentColumn, columnWidth, LABELHEIGHT)];
     [myBackgroundView addSubview:keyLabel];
     currentColumn += LABELHEIGHT + MARGIN;
```

Lopuksi käydään läpi kaikki otsikkoparit ja asetetaan ne oikeille kohdilleen näkymässä, kasvattaen nykyistä sijaintia aina nimikkeen korkeudella ja vaaditulla reunuksella. Kun kolumnin korkeus ylitti keskimääräisen korkeuden, vaihdettiin komponenttien piirtämistä seuraavaan kolumniin.

```
if (currentColumn > columnHeight)
{
     if (biggest<currentColumn)
       biggest = currentColumn;
     currentColumn = MARGIN;
     column++;
}
```

Näin viimeinen kolumni jäi hieman lyhemmäksi kuin aiemmat ja antoi tyylikkäämmän lopputuloksen. Pidettiin myös kirjaa korkeimmasta kolumnista, koska UICollectionView tarvitsi tiedon kuinka korkeaksi tämä solu piti piirtää.

Toinen ongelma, joka aiheutti paljon turhaa työtä, oli CoreData tietokantojen nimeämiskäytäntö. Haluttiin nimetä tietokannan arvot ja taulut samalla tavalla kuin mitä asiakas oli ne omassa rajapintatoteutuksessa nimennyt. Ongelmana oli CoreData:n joustamattomuus nimeämiskäytännössä. Kaikkien arvojen nimet piti olla kokonaan pienillä kirjaimilla. Tästä rajoituksesta ei tiedetty mitään ennen implementointia, koska ei ennen ollut käytetty CoreData tietotyyppejä. Harmaita hiuksia tuotti, kun yritettiin ensimmäistä tietuetta nimetä ja kehitysympäristö ei antanut mitään virheilmoitusta, arvon nimi vain ei muuttunut miksikään kun sille kirjoitettiin uutta nimeä useasti. Lopulta huomattiin, että vika oli isot kirjaimet arvon nimessä ja arvot saatiin näin nimettyä, jonka jälkeen siirryttiin nimeämään tauluja. Ohjelmistokehittäjä meinasi saada hermoromahduksen, kun hän yritti kirjoittaa taululle nimeä pienillä kirjaimilla ja taaskaan mitään ei tapahtunut. Ei virheilmoituksia, eikä nimi vain muuttunut miksikään muuksi. Vika kuitenkin löytyi viimein kun taulun nimeksi yritettiin asettaa kirosanoja isoilla kirjaimilla, ja tämä uusi nimi hyväksyttiin. Taulujen nimet siis piti aloittaa isolla alkukirjaimella.

CoreData tuotti myös ongelmia kannan käyttämisen kanssa. XCode jota jäytettiin kehitysympäristönä menee välillä hieman sekaisin ja automatisoidut metodien luomiset eivät toimi. Kaikki ohjeistukset joita CoreData:n käyttämisestä luettiin, neuvoivat käyttämään automaattisesti luotuja metodeja, jotka alustavat kannan ja näyttävät miten sitä manipuloidaan, mutta kyseisiä metodeja luotu automaattisesti projektissa. Lyhyen internet etsinnän tuloksena löydettiin esimerkkiprojekti, joka oli tallennettu internet sivustolle, ja kopioitiin sieltä tarvittavat metodit tietokannan käyttämiseen. Samaan aihepiiriin liittyen ei ensin saatu luotua tietokanaa auki, graafinen suunnittelutyökalu tekee tiedoston päätteellä ”xcdatamodel” mutta kun sitä koodissa avataan pitää tiedostopääte muuttaa muotoon ”momd” jotta se löytyy ohjelman koodista. Tämä johtuu kääntäjän toiminnasta joka muuttaa tiedostot koneystävällisempään muotoon.

7 Kolmannen vaiheen toteutus

7.1 Toteutus

Osa kolmannen vaiheen tavoitteista tuli jo toisen vaiheen aikana johtuen halusta näyttää dynaamisia tekstikenttiä jo asiakaspalaverissa. Ei ollut järkevää enää verkottoman tilan tarpeen ymmärtämisen jälkeen edetä ohjelman kehityksessä ennen kuin tietokanta olisi ohjelmassa käytössä. Kolmanteen vaiheeseen jäi kuitenkin vielä paljon tehtävää ja se oli tärkeä vaihe ohjelman tekemisessä.

Kolmas vaihe aloitettiin tekemällä valmiiksi kohteiden muokkaaminen ja tallentaminen tietokantaan. Toisessa vaiheessa oli keskitytty lähinnä kannasta tiedon lukemiseen, ja tässä kolmannessa vaiheessa tehtiin toteutus ottamaan mukaansa myös tallentamisen ja kohteiden tilojen ylläpidon. Tätä varten tarvittiin funktio muutosten tallentamiseen, ja lisätä muutama arvo lisää kohde tauluun tilan ja tallennuspäivän ylläpitämiseksi.

```
NSManagedObject *object = [self getObject];
NSSet *spaces = [object valueForKey:@"Kohteet"];
for (NSManagedObject *space in [spaces allObjects])
{
     NSSet *details = [space valueForKey:@"Yksityiskohdat"];
     for (__strong NSManagedObject *possible in [details allObjects])
     {
       if ([[possible valueForKey:@"id"] intValue] == [[detail valueForKey:@"id"]
       intValue])
       {
              // Oikea yksityiskohta löydetty
```

Yläpuolella on toteutus oikean tilan etsimiseen, tämä toteutus on viimeisestä versiosta ja näyttää hyvin monimutkaiselta. Monimutkaisuus johtuu siitä, että kolmannen vaiheen puolessa välissä selvisi, että kenttä id, joka on lyhennys sanasta identifier. Tarkoittaa tunnistetta, ei olekaan ainutlaatuinen. Yleensä kenttä id on projekteissa käytetty merkitsemään ainutlaatuista tunnistetta josta tietokannan arvot erottaa toisistaan, mutta tässä projektissa se saattaa olla sama useammassa kohteessa. Tämän

takia jouduttiin ensin etsimään pääobjekti ja käymään pääobjektin kaikki aliobjektit läpi löytääkseen oikean paikan tietojen tallentamiseen.

Tässä vaiheessa toteutettiin myös käyttäjien tallentaminen ja käyttäjätietojen lukeminen tietokannasta. Haluttiin siirtää käyttäjiä koskevat metodit omaan tiedostoonsa, koodin selkeyden ylläpitämiseksi, ja tästä syystä jouduttiin rikkomaan tietokantaa koskevat metodit kolmeen eri tiedostoon. Kaikki itse tietokantaa, mutta ei tauluja koskevat funktiot, siirrettiin omaan tiedostoonsa ja tehtiin niistä staattisia, jotta niitä voi kutsua tekemättä ensin oliota ja tallentamatta sitä muuttujaan. Sitten siirrettiin käyttäjäfunktiot omaan tiedostoonsa ja luotiin tähän tiedostoon yhteys staattisten funktioiden kanssa. Yhdistetiin staattiset funktiot myös kohteiden manipulointitiedostoon, ja muutettiin metodit kutsumaan näitä uusia funktioita.

Liitos:

```
#import "PersistentStorage.h"
```

Ja kutsu metodissa:

```
NSManagedObjectContext *context = [PersistentStorage context];
```

7.2 Identifier ongelma

Tietokanta oli rakennettu sillä olettamuksella, että id olisi ainutlaatuinen tieto, jota voitaisiin aina käyttää tunnistamaan eri kohteet, joita ohjelmassa tallennetaan. Tämä on hyvin yleinen käytäntö ja koska ei ollut muuta dokumentaatiota olemassa, oletettiin tämän olevan käytäntö myös tässä projektissa. Kuitenkin kun oltiin jo melkein valmiita tietokannan kehityksessä, pidettiin puhelinpalaverin asiakaan kanssa. Tässä palaverissa asiakkaan piti antaa kommentteja senhetkistä ohjelmistosta, ja antaa vastauksia kysymyksiin joita ohjelmistokehittäjälle oli tullut kehityksen aikana. Keskustelun aikana kävi ilmi, että id kenttä jonka rajapinta palauttaa ei ole uniikki tieto vaan jokaisen kohteen uniikki avaintieto saadaan yhdistämällä kohteen id numero sekä kohteeseen liitetyn käyttäjän tilin nimi. Tämä tuli suurena yllätyksenä ja se aiheutti monia ongelmia ohjelman nykyisen rakenteen kanssa.

Tehtiin tarvittavat muutokset jokaiseen kohteeseen, jotta avaintieto jota käytettäisiin, ei olisi enää pelkkä id numero. Vaan id numeroon liitettäisiin myös tilin nimi, jotta oikea kohde löytyisi jonkin yksiselitteisen tiedon takaa. Tässä vaiheessa muutettiin myös monien tietojen haku menemään tämän pääobjektin, eli kohteen, kautta. Tässä vaiheessa tehtiin suuri virhe. Olisi ehdottomasti pitänyt lisätä jokaiseen alikohteeseen, jonka tallensin tietokantaan, oma yksityinen tunnus. Tämän tunnuksen olisi voinut johtaa ottamalla pääkohteen yhdistetyn tunnisteen ja lisäämällä siihen aina alikohteen tunnisteen. Tämä ei kuitenkaan tullut mieleen muutosta tehtäessä ja myöhemmin se aiheutti monia silmukoita, jotka kuluttivat turhaan prosessoritehoja ja hidastivat ohjelman käyttöä. Tästä johtuen myös ylempänä oleva tallennusesimerkki sisältää monia sisäkkäisiä silmukoita sen sijaan että haettaisiin tietokannasta oikealla tunnuksella tarvittava tietue.

7.3 Vaikeudet

Ensimmäiset vaikeudet tämän vaiheen kanssa tulivat, kun muutettiin tapaa, jolla haetaan tietoja tietokannasta. Alussa aina kun haettiin tietoa jostain kohteesta, haettiin se suoraan tietokannasta. Tämä toteutus toimi alussa hyvin, mutta kun lisättiin kohteiden määrää ja kelattiin listaa näytöllä, rupesi toteutus hidastumaan. Tultiin siihen tulokseen tutkinnan jälkeen, että hitaus tuli aina uuden solun näyttämistä tehtäessä. Hitaus siis tuli aina kun ohjelma joutui kesken rullauksen hakemaan tietoja tietokannasta. Vaihdettiin toteutus niin, että kaikkien kohteiden tiedot haettiin heti tietokannasta ja koko lista oli valmiina käyttämistä varten. Kun käytiin kohteita läpi silmukassa, tuli siitä usein loputon silmukka, eikä syy ollut heti näkyvillä. Kaikki kohteen elementit käytiin läpi ja muutettiin ne kirjastoksi. Laittamalla pysäytyspisteitä koodin joukkoon huomattiin, että alikohteen viimeinen elementti osoitti takaisin alkuperäiseen kohteeseen. Tästä muodostuikin ongelma, miten saada rajoitettua dynaamisesti tätä toiminnollisuutta kaikille kohteille, alikohteille sekä yksityiskohdille. Onneksi ob-

jektista löytyi ominaisuuden nimi, jota pystyttiin vertaamaan tähän elementin nimeen, joka viitasi aina yläelementtiin. Lisäämällä metodiin, yläelementin nimen, arvoksi, pystyttiin vertaamaan aina jokaista elementtiä tätä nimeä vastaan, ja jos ne olivat sama, niin ohittamaan kohteen.

```
+ (id) subvalueIterator:(id)object parent:(NSString*)parent
{
     NSMutableDictionary *returnValue = [NSMutableDictionary new];
     NSEnumerator *keyEnumerator = [[[object entity] propertiesByName]
     keyEnumerator];
     NSString *key;
     while ((key = [keyEnumerator nextObject]))
     {
       while ([key isEqualToString:[Dictionary stringDICTtoDB:parent]] || [key
       isEqualToString:@"images"])
              if (!(key = [keyEnumerator nextObject]))
                     return returnValue;
       [returnValue setValue:[self subvalueIterator:[object valueForKey:key]
       parent:[[object entity] name]] forKey:[Dictionary stringDBtoDICT:key]];
     }
     return returnValue;
}
```

Ylläoleva koodi käy läpi kohteen kaikki arvot järjestyksessä. NSEnumerator luokka pitää sisällään listaa avaimista ja rivillä seitsemän pyydämme seuraavaa avainta. Tässä on hyvä huomioida että enumerator palauttaa tyhjän arvon jos kaikki arvot ovat jo käyty läpi ja tämä katkaisee silmukan toistorakenteen. Ylläolevasta esimerkistä on poistettu muutama lauseke tilan säästämiseksi.

Kun tietokantaa päästiin kuormittamaan vielä suuremmalla määrällä kohteita, huomattiin ohjelman etusivun aukeavan todella hitaasti. Hieman aikaisemmin oli juuri muutettu kohteiden lataamista toisenlaiseksi ja nyt ilmeni uutta hitautta. Ongelmaa tutkiessani, vika löytyi juuri tehdystä latausfunktiosta. Ohjelmaa avatessa saattoi mennä jopa 10 sekuntia kohteiden läpikäyntiin ja kirjastomuotoon muokkaamiseen. Tämä oli ensimmäinen tilanne jossa huomattiin, että kirjasto implementaatio ei tule pitkällä tähtäimellä toimimaan kunnolla. Ensimmäinen näkymä, jossa kohteita oli paljon, päätettiin muuttaa käyttämään tietokannan omaa nimeämiskäytäntöä, asiakkaan rajapinnan nimeämiskäytännön sijasta. Ja samalla otettiin kirjastomuotoon

muuttaminen pois tästä näkymästä. Tämä tuotti jonkin verran työtä, kun jouduttiin rivi riviltä käymään koodia läpi ja muuttamaan kaikki viittaukset kaikkiin avaimiin. Kun kirjastoon muutos jätettiin pois, ja otettiin suoraan NSManagedObject tyyppiset objektit, saatiin latausaika putoamaan alle sekuntiin samalla määrällä kohteita.

```
NSFetchRequest *request = [[NSFetchRequest alloc] init];

NSEntityDescription *entityDesc = [NSEntityDescription entityForName:@"Kohteet"
inManagedObjectContext:context];
[request setEntity:entityDesc];

NSPredicate *pred = [NSPredicate predicateWithFormat:@"(lahetys = '' OR lahetys =
nil)"];
[request setPredicate:pred];

NSError *error;
NSMutableArray *results = [context executeFetchRequest:request error:&error];

if (error || [results count] == 0)
     return [NSMutableArray new];

return results;
```

Yllä olevassa koodissa haetaan tietokannasta kaikki kohteet joiden lähetyspäivää ei ole vielä asetettu tai se on asetettu tyhjäksi. Virheen sattuessa, tai jos kohteita ei löytynyt, palautetaan tyhjä lista mahdollisten virheiden estämiseksi muualla ohjelmassa.

8 Neljännen vaiheen toteutus

8.1 Toteutus

Viimeistä vaihetta lähdettiin toteuttamaan integroimalla kirjautuminen ohjelman käyttäjille. Tämä oli loogisin lähtökohta, sillä käyttäjän id numerot tarvittiin kohteiden hakemiseen rajapinnasta. Kutsu itsessään ei ollut monimutkainen, mutta sen ollessa ensimmäinen kutsu, jota lähdettiin implementoimaan ohjelmaan, piti suunnitella mitä ylätunnisteita (header) piti lisätä kun rajapintaa kutsuttiin verkon yli.

```
+ (void) makeContentHeaders:(NSMutableURLRequest**)request
{
     [(*request) setValue:@"application/json" forHTTPHeaderField:@"Content-Type"];
     [(*request) setValue:@"application/json" forHTTPHeaderField:@"Accept"];
     [(*request) setValue:@"iPad_v0.1" forHTTPHeaderField:@"Versio"];
}
```

Tässä on esimerkki toteutustavasta johon päädyttiin tunnistetietoja lisätessä. NSMutableURLRequest on tietotyyppi mukautettavalle kutsulle, johonkin web-osoitteeseen. Tuplatähti notaatio tässä tarkoittaa, että kutsuun liitetään muistiosoite kyseiselle objektille, jotta tämä funktio pystyy muokkaamaan alkuperäistä web-kutsua. Käytännössä, kun tämä funktio loppuu, sisältää kutsuvassa metodissa oleva objekti tässä esitetyt tunnistetiedot. Toteutus on siksi suunniteltu tällä tavalla, että tehtäessä muun laisia kutsuja rajapintaa vastaan, voidaan kaikista metodeista kutsua tätä funktiota, ja lisätä nämä kentät helposti mihin tahansa tehtävään kutsuun.

Seuraavaksi lähdettiin toteuttamaan kutsua, jolla saadaan käyttäjän tiedot rajapinnasta. Ensin luotiin muokattavissa oleva palautuskirjasto, johon asetettiin perusoletamukseksi kutsun epäonnistuneen. Web-osoite objekti tehtiin tekstimuotoisesta osoitteesta, jonka jälkeen luotiin kutsuobjekti, johon annettiin parametreina osoite. Samassa luomiskutsussa on myös parametreina ilmoitettu, että ei tule käyttää laitteen välimuistia, vaan kaikki kutsut pitää tarkastaa palvelimelta. Viimeinen parametri kertoo ohjelmalle, että pyyntö tulee aikakatkaista jos rajapinta ei vastaa kymmenen sekunnin kuluessa. Lopussa asetetaan kutsun olevan hakutyyppiä, sekä lisätään tunnisteet staattisella funktiolla, sekä lisätään käyttäjätiedot kutsuun.

```
NSMutableDictionary *returnDictionary = [NSMutableDictionary new];
[returnDictionary setValue:@"failure" forKey:@"success"];

NSURL *target = [NSURL URLWithString:@"https://rajapinta.asiakas.fi/kayttajat"];

NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:target
cachePolicy:NSURLRequestReloadIgnoringCacheData timeoutInterval:10.0];

[request setHTTPMethod:@"GET"];
[Api makeContentHeaders:&request];
[request setValue:username forHTTPHeaderField:@"Kayttaja"];
[request setValue:password forHTTPHeaderField:@"Salasana"];
```

Kun kutsu on alustettu valmiiksi, lähetetään se palvelimelle ja käsitellään vastauksena saatavat tiedot. Ensin kutsu lähetetään http yhteyttä käyttäen rajapinnalle ja siitä saatavat vastaukset tallentuvat annettuihin muuttujiin. Jonka jälkeen tarkistetaan, että kutsu on onnistunut, tämä tapahtuu tarkistamalla että kutsun tilakoodi on kaksisataan tai yli, ja alle kolmesataa. Tarkistetaan myös mahdollisen virhetietojen palautus ja varmistetaan, että rajapinta palautti tietoja. Jos näistä kriteereistä täyttyy yksikään, on kutsun vastaus hylättävä suoraan. Tässä tapauksessa palautetaan kirjasto jossa alkuarvoksi oli jo asetettu epäonnistunut kutsu.

```
NSError *error;
NSHTTPURLResponse *response;
NSData *returnedData = [NSURLConnection sendSynchronousRequest:request
returningResponse:&response error:&error];

if (floor(response.statusCode/100.0) != 2 || error || !returnedData)
     return returnDictionary;

NSMutableDictionary *user = [[NSJSONSerialization JSONObjectWithData:returnedData
options:NSJSONReadingAllowFragments error:&error] mutableCopy];

if ([user isKindOfClass:[NSDictionary class]])
{
     [returnDictionary setValue:@"success" forKey:@"success"];
     [user setValue:password forKey:@"salasana"];
     [returnDictionary setValue:user forKey:@"user"];
}
return returnDictionary;
```

Lopussa muutetaan saatu tieto luettavaan muotoon tekemällä siitä kirjasto-objekti. Jos tässä vaiheessa tapahtuu virheitä, palautuu metodista taas tieto epäonnistumisesta. Kun viimein käyttäjätiedot on onnistuneesti muutettu kirjastomuotoon, asetetaan palautuskirjastoon tieto kutsun onnistumisesta, sekä lisätään saatu käyttäjä palautuskirjastoon.

Lisätiin myös toteutukset kohteiden haulle ja niiden tallentamiselle käyttäen samoja periaatteita kuin käyttäjätietojen tarkastuksessa. Tarkastuksia haettaessa jouduttiin tekemään hieman enemmän työtä, koska kaikkien kirjautuneiden käyttäjien kohteet piti hakea samalla kerralla rajapinnasta. Rajapinta kuitenkin hyväksyy vain yhden tarkastajan id numeron kutsua kohden, joten jouduttiin tekemään pieni silmukkarakenne. Tämän lisäksi rajoitetiin kohteiden hakemista kahteenkymmeneen kappaleeseen kerrallaan, joten jokaisen käyttäjän kohdalla piti myös pitää kirjaa jo haettujen ja haettavien kohteiden lukumäärästä. Tämän toteutus tehtiin, koska yhden käyttäjän kohteiden määrä saattoi olla todella suuri ja mobiili internet voi olla paikoitellen hidas käyttää. Tämä olisi saattanut aiheuttaa todella pitkiä latausaikoja joissain tilanteissa ja ohjelman reaktiivisuus on tärkeä osa käyttökokemusta.

8.2 Ladattujen kohteiden tallennus ja päivitys

Yllättävän suuren työn tässä vaiheessa jouduttiin tekemään kohteiden tallentamisen parantamiseksi. Asiakas ilmoitti, että kohteet tulisi päivittää palvelimelta saatavilla tiedoilla, mutta käyttäjän ohjelmassa tekemiä muutoksia ei saisi korvata. Aikaisemmin kohteet, jotka tulivat rajapinnasta, ohitettiin, jos samalla id numerolla löytyi jo kohde tietokannasta. Nyt lähdettiin korjaamaan ongelmaa lisäämällä kaksi uutta parametria tallennus funktioon. Ensimmäinen oli kohde johon nämä päivitetyt tiedot haluttiin kirjoittaa ja toinen oli totuusarvomuuttuja, josta näki oliko kyseessä päivitysoperaatio. Käyttämällä olemassa olevaa objektia kaikkeen tallentamiseen, pystyttiin antamaan funktiolle, joko tietokannassa jo olleen kohteen, tai lisäämään sinne uuden

ilman lisätöitä. Ongelmaksi kehkeytyi muutosten testaaminen, koska ohjelmistokehittäjällä ei ollut pääsyä asiakkaan rajapintaan. Testattiin muutosta vaihtamalla hetkellisesti takaisin ohjelman sisäiseen esimerkkitiedostoon, ja käynnistämällä ohjelman uudelleen, pystyttiin muuttamaan tietoja, joita tämä esimerkkitiedosto sisälsi. Ratkaisusta puuttui kuitenkin vielä tärkeä palanen, kun asiakas testasi ohjelmaa, ei päivitys heidän laitteillaan toiminutkaan. Tämä eroavuus johtui ohjelman uudelleenkäynnistämisestä oman testaukseni aikana. Ohjelma ei päivittänyt tietoja tietokannasta ilman erillistä käskyä. Vaikka muutokset kirjoitettiinkin tietokantaan oikein, eivät ne asiakkaan tietojen synkronoinnissa näkyneet. Tämä oli kuitenkin nopeasti korjattu lisäämällä päivityspyyntö aikaisemmin mainittuun kaikkien kohteiden hakuun tietokannasta.

```
for (NSManagedObject *object in results)
     [context refreshObject:object mergeChanges:true];
```

Tallennus ongelma huomattiin, kun saatiin päivitys toimimaan, ja ladattiin ensimmäiset käyttäjän kohteet iPad simulaattorille. Kohteiden tallennus ensimmäisellä kerralla kesti melkein viisitoista sekuntia. Tähän oli ensitilassa saatava muutos, sillä tämä oli ensimmäinen asia mitä käyttäjä huomaa kirjauduttuaan sisään. Viidentoista sekunnin odotusaikana hätäinen käyttäjä ehtii jo sulkemaan ohjelman luullessaan sen olevan toimimaton. Kohteiden tallentamisen optimointi aloitettiin etsimällä tarkasti, missä ongelma sijaitsi. Ohjelman toimintaa ajastettiin käyttämällä NSDate aikaa kuvaavia objekteja, ja kirjoitettiin saadut tulokset kehitysvaiheessa näkyvään loki tiedostoon. Suurin hidastuminen tapahtui, kun kohteiden arvoja tallennettiin tietokantaan. Niitä jouduttiin käymään läpi yksitellen, tallentaen olemassa oleviin objekteihin tietoa kaikille arvoille. Optimoinnin helpottamiseksi, jaettiin tallennus metodi kahteen eri osioon, joista toinen oli tarkoitettu vain kokonaan uusille kohteille, ja toinen oli keskittynyt päivittämään olemassa olevia kohteita. Näin pystyttiin helposti nopeuttamaan ensimmäistä tallennuskertaa ja luomaan uudet tietueet kaikille arvoille jotka kirjastosta löytyi.

```
NSManagedObject *newItem = [NSEntityDescription
insertNewObjectForEntityForName:[Dictionary stringDBtoDICT:key]
inManagedObjectContext:context];
```

Tämä pudotti tallentamiseen kuluvan ajan alle sekuntiin. Päätettiin samalla myös optimoida hieman kohteiden päivitys funktiota, joka vaikutti nyt huomattavasti hitaammalta uusien kohteiden tallentamiseen verrattuna. Vanha funktio, joka tarkisti tietueet tietokantakutsulla, vaihdettiin uuteen nopeampaan funktioon, joka tarkisti olemassa olevat tietueet suoraan objektista.

```
NSManagedObject *newItem = [item valueForKey:key];
```

8.3 Vaikeudet

Suurin osa viimeisen vaiheen ongelmista tuli asiakkaan rajapinnan toiminnasta ja sen dokumentaatiosta. Asiakas päivitti rajapintaa uuteen versioon samaan aikaan kun iPad ohjelmistoa kehitettiin ja tämä aiheutti katkoksia palvelimen toiminnassa. Jonain päivinä en pystynyt lähettämään yhtään kutsua palvelimelle, joka ei joko vastannut tai palautti ilmoituksen virheestä palvelimen puolella. Tämä johti suuriin viivästyksiin ohjelman viimeisen vaiheen kehityksessä.

Rajapinnalle löytyi dokumentaatio verkosta, mutta senkin toimivuus oli välillä huonoa. Välillä nämä verkkosivut olivat kokonaan pois päältä ja välillä saatiin vain virheilmoituksia, kun yritettiin lukea tietoja rajapinnan toiminnasta. Tämän lisäksi vielä dokumentaatio oli hieman vajavaisesti kirjoitettu, ja usein tietojen kuvaukset eivät kertoneet mitä kentät tekivät tai mitä ne sisälsivät, ja mihin sitä tietoa tuli käyttää. Esimerkiksi kun kohdetta haettaessa sen alakohteet sisälsivät kentän ”tila”, jonka kuvauksena oli ”alakohteen tila”. Tämä tieto oli vain numeroarvo enkä tiennyt mitä mikäkin numero tarkoitti ja jouduin ottamaan yhteyttä asiakkaan vastaavaan henkilöön päästäkseni kyseisessä asiassa eteenpäin.

Asiakas myös muutti palvelimelta tulevia tietoja, välillä ilman varoitusta, muuttamatta dokumentaatiota. Esimerkiksi päivämäärässä palautettiin ensimmäiset kaksi

kuukautta sadasosasekunnit myös mukana, mutta sitten ne jätettiin sieltä pois. Tämä johti siihen että kaikki päivämäärät katosivat näkyviltä ohjelmasta. Ja kaikki koodi päivämäärään liittyen jouduttiin korjaamaan, jotta ne saatiin takaisin näkyville ohjelmaan.

9 Ohjelman lähetys asiakkaalle

9.1 iTunesConnect

Jo kehityksen alussa päätettiin, ettemme lisää asiakkaan testiprofiileja Iwa Labsin Apple tunnuksien alle, vaan asiakas luo oman Apple tilin, jonka alle kaikki projektiin liittyvä Applen sivuilla tehtävä työ laitettaisiin. Asiakas loikin pyynnöstämme uuden tilin, mutta kun siellä käytiin tarkastamassa, ei asiakas ollut liittynyt vielä maksulliseen iOS Developer ohjelmaan. Lähetimme uuden viestin asiakkaalle, joka siirsi asian edelleen asiakkaan tilinpito osaston puolelle. Tässä vierähti aikaa, ja tämän aikana vain Iwa Labsin projektipäällikkö testasi ohjelmistoa. Kun asiakas sai omassa päässään kaikki kuntoon, käytiin lisäämässä Iwa Labsin ohjelmistokehittäjän Apple kehitystili tehokäyttäjäksi asiakkaan profiiliin.

Ohjelmiston lähettäminen asiakkaalle iTunesConnect palvelun kautta oli vaikeampaa kuin oletettiin. Ensimmäisenä ongelmana oli kaksi eri testiryhmää, sisäinen ja ulkoinen. Sisäiseen ryhmään ei voinut kutsua kuin täysin uusia henkilöitä eikä kutsuttava sähköpostiosoite saanut olla käytössä missään Applen tietojärjestelmissä. Iwan projektipäällikkö loi toisen tilin tätä kautta, mutta asiakas ei luonut uusia tilejä. Tästä johtuen asiakkaalle lähtevät versiot piti käyttää Applen testijärjestelmän kautta ja heidän testauksensa oli aina muutaman päivän kehitystä jäljessä.

9.2 Vastaanotto

Asiakas oli tyytyväinen ohjelmiston toimintaan ja ulkonäköön, mutta muutaman päivän kuluttua he ottivat yhteyttä sähköpostilla, johon he olivat liittäneet suuren listan puuttuvista ominaisuuksista ja korjauksista. Saimme kehuja käyttöliittymän toiminnasta ja eritoten tehdyistä nopeutuksista ohjelmiston toimintaan. Pieniä käyttöliittymä risuja tuli virheellisten tekstikenttien muodossa.

9.3 Puuttuvat ominaisuudet

Windows ohjelmiston testauksessa oli ollut puutteita molempien, ohjelmistokehittäjän ja käyttöliittymäsuunnittelijan toimesta. Emme olleet huomanneet muutamaa suurta ominaisuutta, jotka piti kuitenkin sopimuksen mukaan toteuttaa. Vika oli ohjelmistokehittäjän kohdalla kokemattomuudesta Windows 8 sovellusten toimintaan. Ohjelmistossa oli lisättynä erilaisia valikkoja jotka ilmestyivät ohjelman alareunaan oikealla hiirenpainikkeella. Vikaa löytyi myös asiakkaan puolelta, koska he eivät kehityksen alkuvaiheessa huomauttaneet meille puutteita käyttöliittymäsuunnitelmissa. Jos he olisivat huomauttaneet heti, esimerkiksi yksityiskohtien monivalinnan puutteesta, olisimme pystyneet lisäämään nämä tärkeät toiminnollisuudet ohjelmiston ollessa vielä täydessä kehityksessä.

Heti puutteiden ilmoituksen jälkeen palattiin Windows sovelluksen pariin ja käytiin läpi toiminnollisuudet. Tässä vaiheessa ei enää oikein pystynyt tekemään suurempia suunnitelmia, joten ryhdyttiin kehittämään uusia ominaisuuksia heti testauksen jälkeen.

10 Applikaation viimeistely

10.1 Bugikorjaukset

Ohjelmistossa oli muutama virhe, jotka olivat jääneet huomaamatta sisäisessä testauksessa, osittain koska emme tienneet mitä näissä tekstikentissä oli tarkoitus olla. Yksi virheistä oli väärä paikka kohteen lyhenteen hakemiselle, joka näkyi väärin melkein kaikissa kohteissa. Kun asiakas ohjasi käyttämään oikeaa kenttää rajapintakutsusta, saatiin korjattua virhe nopeasti.

Asiakas oli myös lisännyt rajapintaan testikäyttäjälle muutaman uuden kohteen joihin he olivat syöttäneet erimittaisia testitekstejä eri kenttiin. Näistä yhdessä kentässä piti olla vain lyhyt tieto, mutta yhteen testikohteeseen tähän kenttään kirjoitettiin todella pitkä teksti. Tämä rikkoi ohjelman ulkoasun, koska ei ollut aiemmin ohjelmaan saatu kohdetta, jonka tekstit olisivat näin oudon mittaisia. Tämänkin ongelman korjaus oli yksinkertainen, tarkistettiin tekstin pituus tässä kentässä.

10.2 Uudet ominaisuudet

Uudet ominaisuudet tuli toteuttaa kiireellisinä ohjelmistoon, ja niille ei ollut käyttöliittymäsuunnitelmia. Käyttöliittymän suunnittelun jäi ohjelmistokehittäjän vastuulle, ja hän yritti tyylitellä nämä uudet käyttöliittymäelementit mahdollisimman samannäköisiksi olemassaolevien komponenttien kanssa.

Ensimmäisenä uutena ominaisuutena lähdettiin toteuttamaan monivalintaa yksityiskohtia varten. Aiemmin käyttöliittymässä yksityiskohta valittiin napauttamalla sitä ruudulla, nyt piti toteuttaa monivalinta tämän vanhan valintatavan lisäksi. Toteutustavaksi valittiin pitkä painallus, joka muutti normaalin painalluksen toimintaa. Kun monivalinta oli käynnistetty, yksityiskohdan näpäytys lisäsi sen valittujen elementtien listaan. Ohjelmassa eteneminen monivalintaa tehtäessä toteutettiin lisäämällä yksi näppäin ohjelman reunaan joka ilmestyy näkyville kun monivalinta käynnistetään.

Toinen tarvittava ominaisuus, oli kohteeseen lisätietojen lisääminen. Tätä ominaisuutta varten tarvittiin kokonaan uusi näkymä Storyboardiin, ja muutama lisäys kohteen näyttämiseen. Kohteen näyttöikkunaan piti lisätä näppäin, joka avasi uuden näkymän näiden tarvittavia lisätietoja varten. Lisätietojen määrä oli suuri, ja näkymään piti lisätä useampi kymmenen tekstikenttää. Toteutus ei kuitenkaan ollut monimutkainen, vain aikaa vievä. Uudesta näkymästä oli tehtävä rullattava, sillä tekstikentät eivät kaikki mahtuneet tabletin näytölle samaan aikaan.

Oltiin hyvin tyytyväisiä käytettyyn kehitysmalliin, kun kuultiin näistä tarvittavista lisäyksistä. Sen sijaan, että meidän olisi pitänyt pitää erillisiä palavereita kehityksen ja ajankäytön päättämiseksi, pystyttiin ohjelmistokehitys käynnistämään suoraan ominaisuuksien parissa.

12 Valmis projekti

12.1 Toiminta

Ohjelman viimeisen version toimituksen jälkeen, saatiin asiakkaalta palautetta ohjelmiston toimivuudesta, myös projektin johtaja Iwan puolelta arvioi projektin toimintaa. Ohjelman toimivuus sai kehuja molemmilta tahoilta. Ohjelmisto on vakaa eikä sen suorituskyvyssä ole ongelmia. Viimeisimmän version testaajien käsittelyssä ohjelmisto ei ole tuottanut yhtään virhettä.

Ohjelmisto sisältää kaikki pyydetyt toiminnot, sekä hyvää käyttökokemusta varten, on siihen lisätty iPad ohjelmissa usein tavattuja ominaisuuksia. Esimerkiksi käyttäjän kirjautuessa ohjelmistoon, voi tabletin näppäimistöltä painaa näppäintä, ja siirtyä käyttäjätunnuksen syöttökentästä salasanan syöttökenttään.

12.3 Ulkonäkö

Ohjelman lopullinen ulkonäkö ja visuaalinen toteutus saivat myös positiivista palautetta, niin asiakkaan kuin Iwankin vastaavilta henkilöiltä. Käyttöliittymä vastasi suunnitelmia erinomaisesti ja sen käytettävyys oli hyvä. Pientä eroa oli kuitenkin huomattavissa lisätietojen syöttämisessä, mutta se ei häirinnyt ohjelman käyttökokemusta.

12.3 Koodi

Ohjelman lähdekoodi ei saanut yhtä hyvää vastaanottoa kuin ohjelman toimivuus ja käyttökokemus. Asiakkaan toimistotiloissa pidetyssä lähdekoodin luovutuspalaverissa käytiin läpi lähdekoodin rakennetta ja käytetyimpiä ohjelmointitapoja. Asiakkaan edustaja ilmaisi huolensa muutamassa kohdassa lähdekoodia, eritoten rajapintakutsuja läpikäytäessä.

En myöskään ollut itse tyytyväinen lähdekoodin rakenteeseen. Lähdekoodi on hyvin sirpaloitunutta, ja siitä puuttuu yhtenäisyys. Tämä johtuu osittain muuttuneista vaatimuksista ja osittain käytetystä ohjelmointi prosessista. Koska lähdekoodia on iteroitu läpi useasti, on sen yhtenäisyys ja rakenne hieman kärsineet. Kun lähdekoodia käytiin läpi luovutuspalaveria varten, suurin osa näistä ongelmista huomattiin, mutta niitä ei enää voinut korjata. Jos iterointia olisi voinut jatkaa vielä yhdellä kierroksella, olisi myös ohjelman lähdekoodi saatu yhtenäistettyä, ja ohjelman toimivuutta mahdollisesti parannettua.

12.4 Asiakkaan palaute

Iwa Labs suoritti asiakastyytyväisyyskyselyn projektin lopussa, ja asiakas antoi Iwa Labsille hyvän arvioinnin. Iwan asiakaskyselyn tärkein kysymys, koskee mahdollista suosittelua muille firmoille, samantyyppisten projektien toteutusta koskien. Tähän kysymykseen vastattaessa alhainen arvosana tarkoittaa, että yritys ei suosittelisi Iwa

Labsia muille kumppaneilleen, ja korkea arvosana tarkoittaa että olisi hyvin todennäköistä, että asiakas suosittelisi Iwa Labsia muille yrityksille. Asiakas antoi projektille arvosanaksi 9.

13 Päätelmät

13.1 Projektin eteneminen

Projektin eteneminen oli erittäin hidasta ja siinä tuli todella paljon virheitä, joita olisi pitänyt korjata aikaisemmin. Ohjelman rakenteen suunnittelu oli huonoa, ja lähdekoodia olisi pitänyt iteroida läpi useamman kerran. Rakenteen suunnittelun virheiden vastuu, voidaan suurimmalta osalta asettaa puuttuneen dokumentaation syyksi.

Alkupään suunnitelman puutteeseen olisi auttanut, jos asiakkaalle olisi painotettu voimakkaammin hyvän dokumentaation tarvetta ja käyttöliittymäsuunnitelma olisi käyty läpi kohta kohdalta ja lyöty lukkoon. Toinen mahdollinen ratkaisu ongelmaan olisi ollut oikean ohjelmistotestaajan palkkaaminen, joka olisi voinut käydä asiakkaan sovelluksen läpi ammattitaitoisemmin, ja koonnut toiminnollisuus listan, jonka mukaan ohjelma olisi voitu tehdä. Iwa Labs olisi myös voinut listata kaikki löytämänsä ominaisuudet asiakkaan ohjelmistosta, ja dokumentoida ne, jonka jälkeen hyväksyttää listä asiakkaalla. Näin ohjelman rakenteen suunnittelu olisi viety heti oikeaan suuntaan, eikä radikaaleja korjaustoimenpiteitä olisi välttämättä tarvittu.

13.2 Omien virheiden tunnistaminen

Myös minä olisin voinut helpottaa työmäärää omilla toimenpiteilläni. Nyt projektin ohjelmointivaiheessa tulleet muutokset ohitettiin mahdollisimman pienillä muutoksilla, työmäärän vähentämiseksi. Tämä lopulta vain lisäsi projektiin käytettyä työmäärää, kun muutoksen kasaantuivat toinen toistensa päälle. Mikäli muutoksia olisi tullut

vähemmän, olisivat nämä nopeat muutokset olleet tehokkaita. Heti kun huomattiin ohjelman olevan huomattavasti vaatimusten ulkopuolella, olisi minun pitänyt korjata lähdekoodin suuntaa voimakkaammin. Tämä olisi tuottanut enemmän työtä siinä ohjelmoinnin vaiheessa, mutta olisi vähentänyt kokonaistyömäärää.

Esimerkiksi tietokannasta kohteiden tietojen ottaminen, kirjastomuotona. Tämä toiminnollisuus olisi pitänyt vaihtaa pois, heti kun tarve tietokannan käyttöön selvisi. Tietokanan ja rajapinnan eroavuuksien takia päätin, että erilainen nimeämiskäytäntö tietueissa olisi käytössä vain tietokannassa, ja muualle ohjelmistoon ei tarvitsisi tehdä muutoksia. Nimeämiskäytäntö olisi pitänyt tehdä samaksi ohjelman sisällä ja käyttää erilaista nimeämistä vain rajapintakutsuja tehtäessä. Muutoksen tekeminen ohjelmistossa hidasti ohjelman toimintaa, ja optimointivaiheessa muutos jouduttiin tekemään kahteen näkymään kuitenkin. Jos koko ohjelma olisi tehty tietokannan nimeämiskäytännön mukaan heti, olisi ohjelmistossa voitu käyttää NSManagedObjectia aina.

13.3 Mitä opin

Projektin kuluessa opin käyttämään uusia elementtejä ja objekteja iOS kehityksessä. En ollut koskaan ennen käyttänyt NSManagedObjecteja jotka tulivat minulle nyt hyvin tutuiksi. Myös ohjelman keskeisin käyttöliittymäobjekti UICollectionView oli minulle uusi, ja sitä jouduin käyttämään hyvin paljon projektin aikana.

Opin myös projektin suunnittelusta paljon uutta. Tämä projektin suunnittelu meni heikosti, mutta siinä minulle tuli monta hyvää oppituntia, joista tärkeimpänä pidän ehdottomasti omien virheiden tunnistamista ja korjaamista.

Suuri positiivinen kokemus minulle oli myös päiväkirjan pitäminen projektin aikana. Kirjoitin kehitysprosessin aikana noin 15 000 sanaa päiväkirjaani, ja se tuli suureksi

hyödyksi kehityksessä. Aamulla kun aloitin työt, kävin aina läpi edellisen päivän kirjoitukseni. Muistin aina mikä metodi jäi kesken tai mitä piti korjata. Viikonlopun jälkeen erityisesti, päiväkirja helpotti työntekoon takaisinpääsyä. Olenkin päättänyt pitää päiväkirjaa kaikissa ohjelmistoprojekteissani. Tekstin määrää voi kyllä hieman vähentää, sillä tässä projektissa kirjoitin erityisen tarkasti muistiinpanoja opinnäytetyön kirjoitusta varten.

Opinnäytetyön tekeminen oli ehdottomasti tärkein kurssi koko ohjelmistotekniikan koulutuksen aikana. Tätä projektia tehdessä, ja työtä kirjoittaessa, olen oppinut paljon, ja päässyt käytännössä kokemaan koulutukseni hyödyn. Ainoa asia, joka opinnäytetyössä minua hieman harmitti, oli sen kieliasun ja ulkonäön tärkeyden painotus. Olisin mieluummin kuullut paljon enemmän palautetta sisällöstä ja vähän kieliasusta, mutta ymmärrän kyllä kieliasunkin tärkeyden raportoinnissa.

Lähteet

Iwa Labs Oy. 2015. http://www.iwa.fi

NetMarketShare. 2015. http://www.netmarketshare.com/

Q4 2014 Unaudited Summary Data. 2015. http://images.apple.com/pr/pdf/q4fy14datasum.pdf

Gartner Says Tablet Sales Continue to Be Slow in 2015. 05.01.2015 http://www.gartner.com/newsroom/id/2954317

andreberg/Meslo-Font. 2015. https://github.com/andreberg/Meslo-Font

Apple Inc. 2015. http://en.wikipedia.org/wiki/Apple_Inc.

iPad. 2015. http://en.wikipedia.org/wiki/IPad

Objective-C. 2015. http://en.wikipedia.org/wiki/Objective-C

iOS. 2015. http://en.wikipedia.org/wiki/IOS