IOS-VERSION TOTEUTTAMINEN ANDROID-SOVELLUKSESTA

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Ammattikorkeakoulun opinnäytetyö

Tietojenkäsittelyn koulutusohjelma

Visamäki, syksy 2014

Tuomas Kurunsaari

TIIVISTELMÄ

HAMK
HÄMEEN AMMATTIKORKEAKOULU

Visamäki
Tietojenkäsittelyn koulutusohjelma
Systeemityö

Tekijä Tuomas Kurunsaari **Vuosi** 2014

Työn nimi iOS-version toteuttaminen Android-sovelluksesta

TIIVISTELMÄ

Opinnäytetyön tarkoituksena oli portata Ambientia Oy:n kehittämä LviDroid Android-sovellus iOS-käyttöjärjestelmälle. Sovelluksen avulla on mahdollista hakea LVI-tuotteiden tietoja Rest-rajapinnasta. Tuotteita voi hakea kirjoittamalla tuotteen LVI-numeron tai käyttämällä mobiililaitteen kameraa viivakoodin lukuun. Ambientia Oy toimi myös opinnäytetyön toimeksiantajana. Opinnäytetyön tavoitteena oli tutustua iOS-ohjelmointiin ja sen eroihin verrattuna Android-käyttöjärjestelmään.

Työn teoriaosuudessa tutustuttiin Androidista iOS:lle siirtymiseen, tekniikoihin ja työkaluihin joita käytännön osuuden suorittamisessa käytettiin sekä Applen tarjoamiin oppaisiin, joita noudattamalla sovellus on mahdollista julkaista App Storessa.

Opinnäytetyön lähteinä käytettiin useaa iOS-ohjelmointia käsittelevää kirjaa sekä internetistä löytyviä oppaita ja artikkeleita. Sovellus kehitettiin Xcode-ohjelmointiympäristössä ja siihen liitettiin RestKit-ohjelmistokehys tietojen hakua varten. Sovelluksen testausta suoritettiin Xcoden tarjoamalla simulaattorilla sekä iPad-tabletilla.

Opinnäytetyön aikana selvisi, että vaikka iOS- ja Android-sovelluskehitys eroavat merkittävästi toisistaan, on molemmissa käytettävä oliopohjainen ohjelmointi hyvin samankaltaista. Molemmissa ohjelmointikielissä luodaan luokkia ja metodeja, joita taas voidaan kutsua. Opinnäytetyön tuloksena kehitettiin LviDroid-sovellus iOS-käyttöjärjestelmälle. Sovelluksessa on samat toiminnot kuin alkuperäisessä Androidille kehitetyssä LviDroid-sovelluksessa. Sovellus ei kuitenkaan nykytilassa ole vielä valmis App Storessa julkaistavaksi. Työn lähdekoodit luovutettiin toimeksiantajalle mahdollista jatkokehitystä varten.

Avainsanat iOS, sovelluskehitys, Objective-C,

Sivut 26 s.

HAMK
UNIVERSITY OF APPLIED SCIENCES

Visamäki
Degree Programme in Business Information Technology
System Engineering

Author Tuomas Kurunsaari **Year** 2014

Subject of Bachelor's thesis Android application to iOS

ABSTRACT

The purpose of the thesis commissioned by Ambientia Oy was to port LviDroid-Android application to iOS operating system. LviDroid application was developed by Ambientia Oy. Using LviDroid, it is possible to retrieve information about HVAC products from Rest interface. Products can be searched by writing the product's HVAC number or using the camera of the mobile device to read the barcode.

The theory part contains information on how to transition from Android to iOS and information about techniques and tools used in the practical part of the thesis. It also contains information about Apple's guides for application development. By following these guides, it is possible to publish an application at App Store.

The source material of the thesis contains multiple iOS programming books and also articles and guides on the internet. The application was developed in Xcode programming environment, and it was merged with RestKit software framework for data retrieval. The application was tested in Xcode's simulator and iPad tablet.

In writing the thesis it became clear that even though iOS and Android application development differ significantly from each other, they share a similar object-oriented programming style. In both programming languages classes and methods are created which can be invoked. As a result of the thesis LViDroid application to iOS was created. The application includes the same features as the original Android based version. However, the application is not yet ready to be published in App Store in its current state. All source codes of the application are possessed by the commissioner for possible further development.

Keywords iOS, application development, Objective-C

Pages 26 p.

Termistö

iOS(aiemmin iPhone OS) on Applen kehittämä mobiilikäyttöjärjestelmä jota käytetään yksinomaan Applen kehittämissä laitteissa kuten iPhone-, iPad- sekä iPod-laitteissa. Ensimmäinen versio iOS:sta nähtiin vuonna 2007, ja uusin julkaisu on tällä hetkellä iOS 8.

OS X on Applen kehittämä käyttöjärjestelmä, joka on suunniteltu käytettäväksi Mac-tietokoneilla. Sen ensimmäinen työpöytäversio julkaistiin vuonna 2001 nimellä Mac OS X v10.0 ”Cheetah” ja sen uusin versio on vuonna 2014 julkaistu Mac OS X v.10.10 ”Yosemite”.

Framework eli ohjelmistokehys on nippu tiedostoja, joita sovellus voi käyttää käyttääkseen tiettyä osaa käyttöjärjestelmästä. Esimerkiksi Core Graphics-ohjelmistokehys sisältää metodit, luokat ja resurssit grafiikan piirtämistä varten.

Objective-C on oliopohjainen ohjelmointikieli jota Apple käyttää iOS sekä OS X käyttöjärjestelmissä sekä niihin kuuluvissa ohjelmointi rajapinnoissa kuten Cocoa ja Cocoa Touch.

HIG(iOS Human Interface Guidelines) tarkoittaa Applen laatimaa dokumentaatiota, jossa on ohjeistukset iOS-sovelluksen käyttöliittymän rakentamiseen.

Xcode on Applen kehittänä ohjelmointiympäristö, jonka avulla voidaan luoda iOS- sekä OS X-sovelluksia. Se julkaistiin ensimmäisen kerran vuonna 2003 ja sen uusin versio on tällä hetkellä Xcode 6.1.

REST(Representational state transfer*)* on ohjelmistoarkkitehtuurimalli, joka hyödyntää jo olemassa olevia tekniikoita ja protokollia joita World Wide Web tarjoaa.

Android on mobiilikäyttöjärjestelmä jota käytetään monien eri valmistajien puhelimissa ja muissa mobiililaitteissa. Sen kehityksestä vastaa Open Handset Alliance sekä Google.

MVC (Model-View-Controller) on yleinen ohjelmistojen suunnittelumalli, jossa erotellaan ohjelman eri osat omiin luokkiinsa jotta ohjelmasta ei tule liian monimutkaista.

IOS SDK(Software Development Kit) on kokoelma tiedostoja jotka tarjoavat Xcodelle kaiken tarvittavan jotta natiivi-sovelluksia pystyy kehittämään iOS-käyttöjärjestelmälle.

Jailbreak(iOS) on keino jonka avulla voidaan ajaa kolmannen osapuolen sovelluksia Applen iOS-käyttöjärjestelmässä. Laitteet jotka on jailbreakattu sallivat allekirjoittamattomien sovellusten ajon.

URI(Uniform Resource Identifier) tarkoittaa merkkijonoa, joka kertoo tietyn resurssin nimen tai sijainnin. Yleisin URI:n erikoistapaus on URL(Uniform Resource Locator), jota käytetään viittaamaan WWW-sivuihin.

Web(World Wide Web) on hajautettu hypertekstijärjestelmä, joka toimii Internet-verkossa.

HTTP(Hypertext Transfer Protocol) tarkoittaa protokollaa, jota WWW-palvelimet ja selaimet käyttävät tiedonsiirtoon.

Porttaamisella tarkoitetaan tässä opinnäytetyössä prosessia, jossa muunnetaan sovellusta niin, että sitä voidaan käyttää ympäristössä, johon sovellusta ei ole alun perin suunniteltu.

SISÄLLYS

1 JOHDANTO

Opinnäytetyön aiheeksi olen valinnut Android-sovelluksen porttaamisen iOS-alustalle. Opinnäytetyön toimeksiantajana toimii Ambientia Oy. Ambientia on sähköiseen liiketoimintaan ja viestintään sekä yhteisöllisiin ratkaisuihin erikoistunut asiantuntijayritys. Ambientia suunnittelee ja toteuttaa erilaisiin tarpeisiin räätälöityjä verkkopalveluja sekä verkkosovelluksia.

Ambientialla on jo kehityksessä Android-sovellus nimeltään LviDroid, jonka avulla on mahdollista hakea viivakoodin tai LVI-numeron perusteella port.lvi-info.fi-palvelusta LVI-tuotteiden tietoja. Palvelu on kehitetty LVI-numero Oy:ta varten, joka ylläpitää LVI-numeroiden tuoterekisteriä. Palvelusta on myös web-sovellus osoitteessa www.lvi-info.fi. Nyt Ambientialla on tarvetta iOS-sovellukselle, jotta iPhone ja iPad-käyttäjät pääsisivät helposti palvelua käyttämään.

Itseäni aihe kiinnostaa, sillä olen jo aiemmin tutustunut Android-ohjelmointiin ja ollut mukana muutaman sovelluksen rakentamisessa. Kuitenkin työssä esiintyvä iOS-alusta on itselleni aivan uusi asia. Pohtiessani opinnäytetyön aihetta, päädyin nopeasti lopputulokseen, että haluaisin tehdä sen johonkin ohjelmointiin liittyvästä aiheesta. Tämän takia tartuin heti tilaisuuteen, kun kuulin että olisi mahdollisuus tehdä opinnäytetyö iOS-ohjelmoinnista.

Työn tavoitteena on siis tutustua Androidin ja iOS-kehityksen eroihin sekä tietysti kehittää toimiva versio LviDroid-sovelluksesta iOS-alustalle. Yhtenä tavoitteista on opetella iOS-ohjelmointia ja tutustua Applen laatimiin ohjeistuksiin, jotta tehty sovellus olisi mahdollisimman lähellä julkaisukuntoa. Opinnäytetyö vastaa seuraaviin tutkimuskysymyksiin: Mitkä ovat erot iOS- ja Android-ohjelmoinnin välillä? Miten Android-sovellus käännetään iOS-alustalle? Millaiset ovat Applen ohjeistukset sovelluksen rakentamiseen?

2 IOS JA OBJECTIVE C

Tässä luvussa kerrotaan tarkemmin iOS-mobiilikäyttöjärjestelmästä jolle LviDroid-sovellusta on kehitetty sekä laitteista, joille kyseinen sovellus on kehitetty eli iPhonesta ja iPadista. Luvussa käsitellään myös Objective-C-ohjelmointikieltä sekä opinnäytetyössä käytettyä Xcode-ohjelmointiympäristöä.

2.1 iOS

IOS on Applen kehittämä mobiilikäyttöjärjestelmä, joka on tehty yksinomaan Applen laitteistoilla toimivaksi. Käyttöjärjestelmä julkistettiin vuonna 2007 iPhonelle, minkä jälkeen siihen on lisätty tuki Applen muille laitteille kuten iPod Touch, iPad, iPad Mini sekä Apple TV-laitteille. Alun perin käyttöjärjestelmän nimi oli iPhone OS, ja sinä se pysyikin neljä vuotta, kunnes se vaihtui iOS 4 -julkaisun aikana iOS-muotoon. Kesäkuussa 2014 Applen sovelluskaupassa eli App Storessa oli tarjolla yli 1,2 miljoonaa iOS-sovellusta.(Apple Inc. 2014a.)

IOS:n käyttöliittymä perustuu täysin kosketuksella ohjaamiseen, ja se tukee myös monikosketuseleitä. Käyttöliittymän ohjaukseen tarkoitettuja elementtejä ovat erilaiset liukusäätimet, kytkimet ja painikkeet. IOS:n kosketusnäytön ohjaamiseen tarkoitettuja eleitä ovat napautus, veto, pyyhkäisy ja nipistys, joille jokaisella on oma toimintonsa iOS-käyttöjärjestelmässä. Jotkin iOS-sovellukset käyttävät laitteen sisäistä kiihtyvyysanturia esimerkiksi kumoa-toimintona, jolloin käyttäjä voi puhelinta ravistamalla kumota tekemänsä toiminnon. Toinen yleinen kiihtyvyysanturin käyttökohde on sovelluksen näkymän vaihtaminen erilaiseksi, kun käyttäjä kääntää laitteensa pystyasennosta vaaka-asentoon.(Apple Inc. 2014b, 9, 20, 23, 45.)

Opinnäytetyön kirjoitushetkellä iOS-käyttöjärjestelmän uusin versio on iOS 7, ja merkittäviä versioita käyttöjärjestelmästä julkaistaan vuosittain. IOS käyttää muutamia samoja ohjelmistokehyksiä kuin OS X kuten Core Foundation ja Foundation. IOS kuitenkaan ei ole yhteensopiva OS X:n kanssa, sillä käyttöliittymässä käytetään Cocoa Touch-ohjelmistokehystä OS X:n Cocoan sijaan. (Apple Inc. 2014)

2.2 Objective-C

Objective-C on yleiskäyttöinen olipohjainen ohjelmointikieli, joka on laajennettu C-ohjelmointikielestä lisämällä siihen Smalltalk-tyylinen oliomalli. Apple käyttää Objectice-C:tä OS X sekä iOS-käyttöjärjestelmissä sekä niihin kuuluvissa ohjelmointirajapinnoissa kuten Cocoa Touchissa.

Objective-C:n olio-ohjelmointimallissa on moni eroja verrattuna Simulasta johdettuihin ohjelmointikieliin kuten C++, Java ja C#. Sen syntaksi on johdettu lähes suoraan Smalltalk-ohjelmointikielestä, jossa olioille lähetetään viestejä sen sijaan että kutsuttaisiin tiettyä metodia. Simulasta johdetussa kielissä täytyy tietää olion tyyppi ennen kuin sen metodia voi kutsua. Objective-C:ssä tämä ei ole pakollista sillä olio, jolle viesti lähetetään, yrittää

tulkita vastaanottamansa viestin, mutta ei ole takuuta siitä vastaako se siihen. Mikäli olio ei ymmärrä viestiä, se jättää sen huomioimatta ja paluttaa arvon nil. Tämänkaltaisen oliomallin avulla ei ole jatkuvasti tarvetta suorittaa tyyppimuunnoksia, jotta viestin vastaanottaja ymmärtäisi viestin. (Allan 2013, 28.)

Toinen merkittävä ero Objective-C:n ja Simulasta johdettujen kielien välillä on muistinhallinnassa. Esimerkiksi Javassa käytetään roskien keräystä muistinhallintaan, kun taas Objective-C:ssä se tapahtuu viittausten laskennalla. Aikaisemmin se suoritettiin käsin, mutta iOS 5:sta julkaisun yhteydessä esiteltiin uusi ominaisuus nimeltään ARC eli Automatic Reference Counting, joka yksinkertaisti tätä operaatiota huomattavasti. (Allan 2013, 28–29, 63.) ARC on ohjelman kääntämisen aikana toimiva ominaisuus, joka lisää automaattisesti koodin sekaan tietyt metodikutsut, jotta viittaukset olioihin joita ei tarvita vapautetaan muistista.

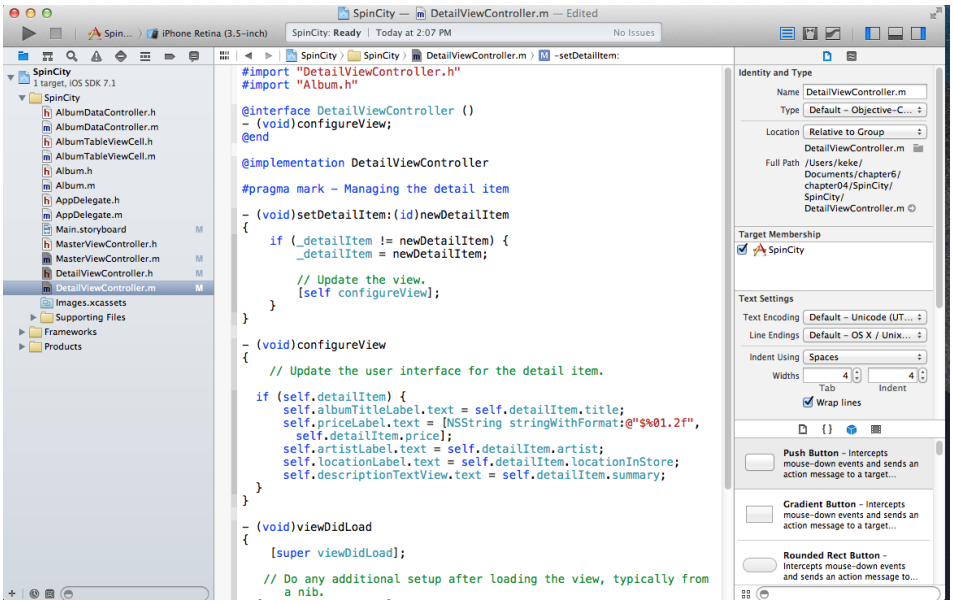
2.3 iPhone & iPad

iPad on Applen kehittämä taulutietokone, joka julkaistiin vuonna 2010. Siitä on julkaistu kuusi erilaista versiota. Sen uusin versio iPad Air 2 julkaistiin lokakuussa 2014. iPad Mini on iPadia pienemmällä näytöllä varustettu taulutietokone ja siitä on julkistettu kolme eri versiota, joista uusin iPad Mini 3 julkaistiin lokakuussa 2014.Laitteen käyttöliittymä on rakennettu tukemaan monikosketusta ja se sisältää virtuaalinäppäimistön. Kesäkuuhun 2014 mennessä iPadeja on myyty yli 200 miljoonaa kappaletta sen julkistamisen jälkeen. (GSMArena, 2014)

iPhone on Applen kehittämä vuonna 2007 julkaistu älypuhelin. iPhonen uusimman sukupolven mallit iPhone 6 ja iPhone 6 Plus julkaistiin syksyllä 2014. (Apple Inc. 2014e)

2.4 Xcode

Xcode on Applen kehittämä ohjelmointiympäristö jonka avulla pystyy kehittämään sovelluksia OS X sekä iOS-käyttöjärjestelmille. Nykyaikainen sovelluskehitys vaatii suuren määrän erilaisia sovelluksia, kuten editoreja, kääntäjiä, syntaksin tarkistajia, salauksen allekirjoittajia, debuggereita, simulaattoreita, suorituskyvyn analysoijia ja monia muita. (Bucanek 2014, 3.) Kuitenkaan niistä ei tarvi itse huolehtia, sillä Xcode sisältää kaikki tarvittavat yksittäiset työkalut, jotta kehittäjä voi keskittyä sovelluksen kehittämiseen. Alapuolella kuva Xcoden käyttöliittymästä. (Kuva 1)



Kuva 1. Xcode-ohjelmointiympäristö.

Työkalujen lisäksi Xcodessa on SDK:t iOS sekä OS X- sovelluksia varten. SDK on kokoelma tiedostoja joita Xcode tarvitsee sovellusten suorittamiseen tietylle käyttöjärjestelmälle, kuten iOS 7:lle. SDK:t sisältävät yhden tai useamman ohjelmistokehyksen jotka kertovat Xcodelle kuinka jokin sovellus voi käyttää iOS-palveluita. Vaikka sovelluksen voi ohjelmoida tekemään lähes mitä tahansa, suuri osa sovelluksen toiminnasta on pyyntöjen lähettämistä iOS:in valmiiksi kirjoitetuille palveluille ohjelmointirajapinnan avulla. Tälläisia palveluita ovat esimerkiksi: hälytyksen näyttäminen, kuvan ottaminen, musiikkikappaleen soittaminen.(Bucanek 2014, 3).

Ohjelmointiympäristön lisäksi Xcode sisältää suurimman osan Applen kehittäjä-dokumenteista sekä Interface Builderin jonka avulla pystyy kehittämään sovelluksen graafista käyttöliittymää. Uusin Xcode-versio on 6.1 ja sen pystyy lataamaan ilmaiseksi Mac App Storesta.

2.5 iOS Developer

Mikäli haluaa testata tekemäänsä sovellusta oikealla laitteella, täytyy rekisteröityä iOS-kehittäjäksi. Tämän lisäksi iOS-kehittäjänä pystyy myymään sovelluksiaan Applen App Storessa ja asentamaan sovelluksiaan muiden henkilöiden laitteille ilman App Storea. iOS-kehittäjänä pystyy myös kehittämään sovelluksia jotka käyttävät sovelluksen sisäisiä ostoja, push notifikaatioita tai kommunikoivat Applen servereiden kanssa. IOS-kehittäjänä saa myös oikeudet Applen Developer-keskustelu-alueelle. Yksityiselle henkilölle liittyminen iOS-kehittäjäksi maksaa 99 dollaria ja jäsenyys kestää vuoden. Kun sovelluksen testaukseen käytetään oikeaa laitetta iOS-simulaattorin sijaan, pystytään testaamaan sovelluksia jotka käyttävät esimerkiksi GPS:ää, monikosketusta, kiihtyvyysanturia tai kameraa. (Bucanek 2014, 4.)

2.6 iOS-natiivisovellusten historiaa

Kun iPhone alun perin julkaistiin vuonna 2007, ei sille löytynyt vielä natiivi-SDK:ta. Applen mukaan sellaista ei tarvittu ja laitteelle kehitettäisiin web-sovelluksia käyttäen JavaScriptiä, CSS:sää ja HTML:ää. Tämän ansiosta saataisiin hyödynnettyä web-sovellusten hyvät puolet kuten sen helppo ylläpito ja halvemmat kehityskustannukset verratuna natiivisovelluksiin. Lisäksi sovelluksen löytäminen olisi helpompaa, sillä se voi löytyä käyttäjän tekemän haun perusteella esimerkiksi Google-hakukoneella. (Budiu, 2013.) Tämä ei kuitenkaan ollut kehittäjien mieleen, sillä he halusivat suoran pääsyn laitteistoon sekä integraation Applen omiin sovelluksiin (Allan 2013, 1). Morgenstein (2013) mainitsee web-sovelluksia käsittelevässä artikkelissaan, että Applen idea web-sovelluksien käytöstä oli järkevä, mutta aikaansa edellä. Selaintekniikat eivät olleet vielä tarpeeksi kehittyneitä iPhonen julkaisun yhteydessä.

Vaikka Apple oli lukinnut iPhonen estääkseen natiivi-sovellusten kehittämisen, ei kulunut montaa kuukautta kun avoimen lähdekoodin yhteisöjen jäsenet olivat ohittaneet estot ja takaisinmallintaneet SDK:n ja rakentaneet ilmaisen avoimen lähdekoodin työkalun, jonka avulla pystyi kehittämään natiivisovelluksia iPhonelle. Tämä johti siihen että yli kolmannes iPhonen omistajista ”jailbreakkasi” laitteensa, jotta pystyisi asentamaan siihen kolmannen osapuolen kehittämiä sovelluksia. Apple kuitenkin vastasi kehittäjien odotuksiin ja julkaisi virallisen natiivi-SDK:n alle vuosi ensimmäisen iPhonen julkaisun jälkeen. (Allan 2013, 1.)

Lukuisat kehittäjät eivät olleet tyytyväisiä Applen ratkaisuun julkaista natiivi-SDK. Heidän mielestään koodin kirjoittaminen pelkästään iPhonelle Objective-C:llä teki sovellusten porttaamisesta muille alustoille vaikeampaa, sillä web-sovellusten porttaaminen onnistui yksinkertaisimmillaan CSS-tyylimallia vaihtamalla halutun käyttöjärjestelmän mukaiseksi. Kuitenkin näytti siltä että iPhonen käyttäjät olivat eri mieltä, sillä web-sovellukset eivät täyttäneet käyttäjien odotuksia. Käyttäjät halusivat sovelluksia jotka näyttivät ja tuntuivat samalta kuin Applen kehittämät natiivi-sovellukset (Allan 2013, 1–2.)

Käyttämällä natiivi-SDK:ta sovellusten kehittämiseen, pystyy tekemään asioita joita web-sovelluskehitys ei tarjoa. Esimerkiksi ensimmäisen sukupolven lisätyn todellisuuden- sovellukset, jotka tarvitsevat tiiviin integroinnin iPhonen sensoreihin, kuten GPS:ään, kiihtyvyysanturiin, digitaaliseen kompassiin ja kameraan. Yksi natiivi-sovellusten etu verrattuna web-sovelluksiin on se, ettei sovellus välttämättä tarvitse internet-yhteyttä tai edes joitain osia sovelluksesta voi käyttää ilman yhteyttä internetiin, tämän avulla saadaan myös akunkestoa parannettua.(Allan 2013, 2.)

3 SIIRTYMINEN KÄYTTÖJÄRJESTELMÄSTÄ TOISEEN

Alkuperäinen LviDroid-sovellus on kehitetty Androidille eli se on kirjoitettu Javalla, joka on syntaksiltaan yksi helpoimmista oliopohjaisista ohjelmointikielistä.(Liao 2014, 28) Jotta siirtyminen Androidista iOS:lle sujuisi, on molempien ohjelmointikielien perusteiden oltava hallussa. Liao (2014, 27) mainitsee että paras keino uusien ohjelmointikielien ja työkalujen opetteluun on käytännön kokemus sekä harjoittelu.

Tässä kappaleessa tutustutaan suurimpiin eroihin joihin törmättiin siirtyessä Androidista iOS:lle. Lisäksi tutustutaan J2ObjC-työkaluun, joka mahdollistaa Java-koodin kääntämisen Objective-C:lle.

3.1 Androidista iOS:lle

Porttaamisella tarkoitetaan prosessia, jossa muunnetaan sovellusta jotta sitä voidaan käyttää ympäristössä, johon sovellusta ei alunperin ole suunniteltu. Tässä opinnäytetyössä LviDroid-sovellus luotiin tyhjästä, eikä sovelluksen kehitykseen käytetty porttausta tukevia työkaluja kuten J2ObjC-työkalua.

iOS-sovellukset suunnitellaan ja rakennetaan samaan tapaan kuin Android-sovellukset. Objective-C:n syntaksi näyttää erilaiselta kuin Java, mutta oliopohjainen ohjelmointi pysyy samankaltaisena, eli luodaan luokkia ja metodeja, joita taas voidaan kutsua. Androidille kehittäessä riittää Android Developer Toolsin asennus, kun taas Mac OS:llä vastaavat kehityöstyökalut saa käyttöönsä asentamalla Xcoden. (Liao 2014, 59.) Alla olevassa taulukossa on esitetty Objective-C:n ja Javan syntaksin eroja.

Taulukko 1. Objective-C:n ja Javan syntaksien vertailu.

Objective-C	Java
#import "abc.h"	import packagename.abc
@interface Abc: NSObject	public class Abc extends Object
@protocol Abc	public interface Abc
@interface Abc: NSObject <Efg>	class Abc extends Object implements Efg
@property int i	private int i;
-(id)init	public Abc();
Abc* obj = [[Abc alloc] init];	Abc obj = new Abc();
-void doWork:(NSString*) arg;	public void doWork(String arg)
[obj doWork:arg];	obj.doWork(arg);

Objective-C:ssä ei ole Javasta tuttuja public, protected ja private -määreitä joilla metodien ja muuttujien näkyvyys määritetään. iOS:ssa jokaisella luokalla on oma otsikkotiedostonsa(header file), jossa julkiset metodit ja muuttujat määritetään. Otsikkotiedostossa määritettyjä metodeja ja muuttujia pystyy käyttämään samaan tapaan luokan ulkopuolelta kuin Javassa julkisia metodeja. Oliopohjaisessa ohjelmoinnista rajapinta kuvaa käyttäytymistä, jota rajapinnan toteuttavan luokan täytyy toteuttaa. Javassa käytetään rajapinnasta avainsanaa interface, kun taas Objective-C:ssä käytetään @protocol-avainsanaa, joka toimii kuitenkin samanlailla kuin Javan rajapinnat. Tämä saattaa olla harhaanjohtavaa kun siirrytään Androidista Objective-C:hen sillä Objective C:ssä normaalit luokat määritellään interface-avainsanalla. (Liao 2014, 35)

Suuri osa sovelluksen siirtämisestä Androidilta iOS:lle on sovelluksen yleisten ominaisuuksien korvaamista. Useat data-tyypit ja symbolit löytyvät sekä Javasta että Objective-C:stä eri avainsanoilla. Alla olevassa taulukossa on esitelty Javan ja Objective-C:n symboleita ja syntaksia, jotka on mahdollista korvata toisen kielen vastineella.

Taulukko 2. Objective-C:n ja Javan symboleiden ja syntaksin vertailua.

	Objective-C	Java
Olioviittaus	*	Ei ole.
Self-viittaus	self.	this.
String-luokka	NSString	String
String-merkkijono	@"abc"	"abc"
Taulukko	NSArray	ArrayList tai JSONArray
Hajautustaulu	NSDictionary	HashMap tai JSONObject
Boolean-tietotyyppi	BOOL, YES/NO	Boolean, true/false
Integer-tietotyyppi	NSInteger, NSUInteger	Integer
Null-arvo	nil	null
Olio	Id tai NSObject	Object
Metodin julistus	-(void), -(NSString*)	private void, private String
Staattinen metodi	+(void), +(NSString*)	private static void, private String
Vakio	#define var	public static final var

Androidille kehittäessä käytetään Eclipsen graphical layot editoria käyttöliittymän luonnissa ja käyttöliittymäkomponenttien muokkauksessa. iOS:lle kehitettäessä tähän käytetään Xcoden storyboard-työkalua tai Interface Builderia. (Liao 2014, 62)

Suurin osa sovelluksista koostuu muustakin kuin ohjelmoidusta koodista, esimerkiksi tyypillinen Android-projekti sisältää Java-lähdekoodit, sovellusresurssit kuten kuvia, audio-tiedostoja ja XML-tiedostoja joilla määritellään animaatioita, valikkoja, tyylejä, värejä ja ruutuja. Samaan tapaan myös

iOS-projekti sisältää Objective-C lähdekoodin lisäki erilaisia kirjastoja, yhden tai useamman storyboard-tiedoston, kuvia tai multimediatiedostoja. Xcode kokoaa ja luo koko projektin ja niputtaa kaikki osat yhdeksi toimivaksi kokonaisuudeksi. (Liao 2014, 46)

3.2 J2ObjC

J2ObjC on Googlen kehittämä avoimen lähdekoodin työkalu, joka mahdollistaa Java-koodin kääntämisen Objective-C:lle iOS-alustaa varten.Työkalun avulla alkuperäinen Java-koodi saadaan osaksi iOS-sovellusta, eikä generoitujen tiedostojen muokkaaminen ole välttämättä tarpeellista. J2ObjC tukee suurinta osaa Java-ohjelmointikielen ominaisuuksista, joita asiakasohjelmien kehitys vaatii. Tuettuja ominaisuuksia ovat esimerkiksi poikkeukset, anonyymit luokat, sisäluokat, geneeriset tyypit ja säikeet. J2ObjC:n kehitystyö on tällä hetkellä Alpha ja Beta- vaiheen välillä. (J2ObjC homepage, 2014)

Kun J2ObjC:n kehitys alkoi, moni kehittäjä luuli sen toiminnan olevan mahdotonta iOS:n tiukkojen käyttöliittymästandardien takia. Googlen mielestä ainoa keino kehittää maailmanluokan iOS-sovelluksia, jotka toimivat nopeasti, on kirjoittaa ne Objective-C:llä käyttäen Applen omia ohjelmistokehyksiä. Tämän takia J2ObjC tukee pelkästään sovellusten business-logiikan kääntämistä ja käännettyjen sovellusten käyttöliittymät on tarkoitus luoda iOS:n ohjelmistokehyksiä käyttäen. (J2ObjC wiki, 2014)

4 TEKNIIKAT JA TYÖKALUT

Tässä luvussa kerrotaan tekniikoista ja työkaluista, joita opinnäytetyön käytännön osuudessa on käytetty. LviDroid-sovellus, joka opinnäytetyössä kehitettiin käyttää RestKit-ohjelmistokehystä LVI-tuotteen tietojen hakemiseen Rest-rajapinnasta. Tämän takia halusin tutustua tarkemmin siihen mitä Rest oikeastaan tarkoittaa. Myöskin iOS-sovelluskehitykseen vahvasti liittyvään MVC-suunnittelumalliin tutustuttiin.

4.1 Rest

Rest eli Representational State Transfer on arkkitehtuurityyli palveluiden kehittämiseen, jonka esitteli Roy Fielding vuonna 2000 väitöskirjassaan ”Architectural styles and design of network-based software architectures”. Rest on joukko rajotteita, jotka perustuvat Word Wide Webin arkkitehtuurityyliin. Nämä rajoitteet ovat kuitenkin enemmän ohjeistuksien tyyppisiä eikä niinkään sääntöjä, joita olisi pakko noudattaa Rest-pohjaisia web-rajapintoja kehittäessä. Jotkut palvelut noudattavat kaikkia rajotteita, kun taas jotkut noudattavat vain muutamia niistä. Webin arkkitehtuuri perustuu muutamaan perusperiaatteeseen, joiden ansiosta Webin suosio on kasvanut sen nykyiseen tilaan. (Flanders 2009, 1.) Nämä perusperiaatteet on esiteltynä seuraavissa luvuissa.

4.1.1 Resurssien osoitteellisuus ja URI

Tärkeimpänä näistä periaatteista Flanders (2009, 1) mainitsee osoitteelliset resurssit, mikä tarkoittaa sitä että käytetään URIa määrittelemään yksittäisen resurssin sijainti. Resursseja ovat esimerkiksi HTML-tiedostot, kuvat ja muut mediatyypit. Resurssit voivat olla staattisia kuten esimerkiksi kuvat, jotka on otettu tiettynä päivänä tai ne voivat olla dynaamisia kuten elokuvateattereiden sivustoilla olevat näytösajat, jotka muuttuvat päivittäin. Nykypäivänä URIt ovat sijoitettu näkyvästi esimerkiksi mainoksissa, joista käyttäjä voi ne helposti poimia ja kirjoittaa omaan selaimeensa jonka jälkeen selain palauttaa oikean resurssin käyttäjälle. (Flanders 2009, 1–2.)

Hyvin suunnitellun URIn hyödyllisyys on lähes itsestäänselvää, minkä huomaa jos on joskus itse selvittänyt websivun URI:n löytääkseen tietyn resurssin vaikka on tiennyt vain osan websivun osoitteesta. Restin tapauksessa URIen pitäisi olla loogisia perustuen resursseihin joihin URI viittaa. URI joka on selkokielinen kelle tahansa käyttäjälle on myös helposti käytettävä esimerkiksi ohjelmalle, joka käyttää kyseistä URIa ohjelmallisesti. Helposti luettavat URIt helpottavat Rest-pohjaisen rajapinnan testausta ja debuggausta, mutta eivät ole ehdottoman välttämättömiä palvelun luokittelemiseksi Rest-pohjaiseksi. (Flanders 2009, 7.)

Erinomainen esimerkki helposti ymmärrettävistä URIsta on Flickr-yhteistöpalvelusivusto, joka erikoistuu valokuvien ja videoiden jakamiseen. (Flanders 2009, 6.)

```
https://www.flickr.com/people/{user-id}/
https://www.flickr.com/photos/{user-id}/
https://www.flickr.com/photos/{user-id}/{photo-id}
https://www.flickr.com/photos/{user-id}/sets/
https://www.flickr.com/photos/{user-id}/sets/{pho-
toset-id}
```

Esimerkki 1. Flickr-palvelun URIt.

Yllä olevista esimerkistä huomaa, että kyseiset URIt ovat loogisia ja perustuvat resursseihin. Hakasulkeiden sisään asetetaan joko user-id, photo-id tai photoset-id jonka avulla yksilöidään resurssi.

4.1.2 Standardit mediatyypit

Osa Webin suuresta suosiosta johtuu siitä että Webissä olevat resurssit ovat standardin mukaisia. Tämän ansiosta ohjelmat ja käyttäjät pääsevät käsiksi web-serverin resursseihin käyttämällä mitä tahansa modernia käyttöjärjestelmää ja selainta. (Flanders 2009, 2.) Virallista rekisteriä mediatyypeistä hallinnoi Internet Assigned Numbers Authority.

4.1.3 Yhtenäinen rajapinta

Yhtenäinen rajapinta on yksi Rest-arkkitehtuurityylin rajotteista. Käyttämällä yhtenäistä rajapintaa ei tarvitse luoda uutta rajapintaa aina uuden palvelun luonnin yhteydessä. (Flanders 2009, 7.) Webin yhtenäinen rajapinta perustuu HTTP-protokollaan. HTTP on avoin ja hyvin tunnettu protokolla, joka määrittelee standardin tavan selaimen ja palvelimen väliseen tiedonsiirtoon. Tiedonsiirto perustuu HTTP-pyyntöjen lähettämiseen selaimelta palvelimelle. HTTP-metodeista yleisimmät ovat GET, DELETE, POST ja PUT-metodit. (Flanders 2009, 2.)

GET-metodia käyttämällä kerrotaan palvelimelle että selain haluaa vain luku- tyylisen esityksen resurssista. Yksi tärkeimmistä GET-metodin ominaisuuksista on mahdollisuus tallentaa vastauksena saatu resurssi välimuistiin. GET-metodi on myös turvallinen, joka tarkoittaa sitä että sen käyttäminen ei saa vaikuttaa resursseihin. Kuitenkin resurssi voi vaihtua kahden GET-pyynnön välissä esimerkiksi palvelun itsenäisen toiminnan takia. POST-metodia käyttämällä ilmaistaan että halutaan luoda uusi resurssi. PUT-metodin yleinen käyttötarkoitus on resurssin muokkaus tai uuden resurssin lisäys mikäli selaimella on tieto miten määrittää URI uudelle resurssille. DELETE-metodia käyttämällä ilmoitetaan että halutaan poistaa resurssi. (Flanders 2009, 2.)

4.1.4 Tilattomuus ja hyperlinkit

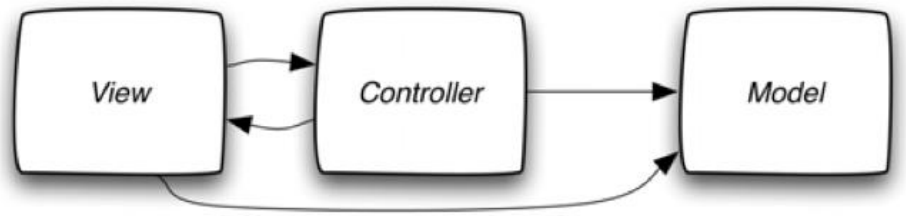
HTTP ja Web on suunniteltu olemaan tilattomia. Tilaton palvelu pystyy käsittelemään asiakkaalta saapuvan pyynnön pelkästään pyynnössä olevien tietojen perusteella. Kun jokainen pyyntö on riippumaton edellisistä pyynnöistä voidaan pyyntö käsitellä palvelinfarmilla millä tahansa palvelimella tahansa, jolloin saadaan luotua skaalautuva ja kestävä ympäristö. (Flanders

2009, 3.) Hyperlinkit resurssien välillä ovat myös tärkeä osa Webin menestystä. Resurssi voi linkittää toiseen resurssiin, johon voidaan navigoida selaimen avulla.

4.2 MVC-suunnittelumalli

MVC eli model-view-controller (malli-näkymä-ohjain) on yleinen suunnittelumalli, jota käytetään apuna sovelluksien kehittämisessä. MVC-suunnittelumallia käytetään yleisesti web-kehityksessä, Windows- sekä Mac-ohjelmoinnissa ja mobiilikehityksessä. MVC-suunnittelumallista on hyötyä lähes jokaiselle sovellukselle, jossa on graafinen käyttöliittymä. (Franco & Mendelowitz 2013, 39.) MVC-suunnittelumalli on hallitseva suunnittelumalli Mac- ja iOS-ohjelmoinnissa. IOS-sovelluskehittäjän on tärkeää ymmärtää kuinka MVC-suunnittelumalli toimii, jotta sovelluksen luokat ja käyttöliittymä voidaan suunnitella järkevästi. MVC-suunnittelumallin ideana on jakaa kaikki sovelluksen komponentit kolmeen eri rooliin. Komponenttien jako eri rooleihin tekee sovelluksen osista toisistaan riippumattomia, helposti konfiguroitavia ja uudelleenkäytettäviä.(Harris 2014, 39.)

Mallilla tarkoitetaan luokkia, joissa sovelluksen data säilötään. Näkymällä tarkoitetaan ikkunoita ja elementtejä joita käyttäjä näkee ja käyttää, kuten esimerkiksi painike. Ohjaimella tarkoitetaan koodia jonka avulla Malli ja Näkymä yhdistetään toisiinsa. Ohjain sisältää sovelluksen toiminnallisuuden, joka päättää kuinka käyttäjän syötteisiin reagoidaan. (Mark & Nutting & LaMarche & Olsson 2013, 44.) Oikein käytettynä MVC-suunnittelumalli vähentää huomattavasti koodin kirjoittamisen määrää ja tekee sovelluksen yleisestä arkkitehtuurista helposti ymmärrettävää, mikä auttaa tilanteessa, jossa uusi henkilö on liittymässä projektin pariin kehittämään sovellusta.(Harris 2014, 39.) Alapuolella on MVC-suunnittelumallia havainnollistava kuva. (Kuva 2)



Kuva 2. MVC-suunnittelumalli havainnollistettuna. (Franco & Mendelowitz 2013, 23)

4.2.1 Malli

Mallilla tarkoitetaan iOS-kehityksessä normaaleja Objective-C-luokkia, jotka on luotu kuvaamaan sovelluksen dataa. Data voi olla mitä tahansa sovelluksessa olevia arvoja kuten nimiä, osoitteita tai kuvia. (Mark & Nutting & LaMarche & Olsson 2013, 45). Esimerkiksi yhteystiedot-sovelluksessa Malli-luokan sisältö voisi näyttää seuraavanlaiselta: Etunimi, Sukunimi, Puhelinnumero, Sähköposti ja Kotiosoite.

Malli-luokan tulisi piilottaa tieto siitä miten data on säilötty, eli sen pitäisi olla kapseloitu. Kapseloidun luokan täytyy tarjota sovelluksen muille komponenteille yksinkertainen rajapinta, jota käyttäen sovelluksen muut komponentit voivat saada tarvitsemansa datan, ilman tarkkaa tietoa siitä miten data on Malli-luokassa kuvattu tai säilötty. (Bucanek 2014, 218.)

Malli-luokan ei pitäisi sisältää yhtään muuttujaa tai koodia, jotka eivät ole tekemisissä sovelluksen datan kanssa tai datan säilömisen kanssa. Malli on MVC-suunnittelumallin puhtain rooli, mikä tarkoittaa sitä, että Malli-luokan ei pitäisi tietää mitään Näkymä tai Ohjain-komponenteista. Malli-luokassa ei siis pidä olla viittauksia Näkymä-olioihin tai metodeja joilla esitetään dataa käyttöliittymässä tai käsitellään käyttäjän syötettä. (Bucanek 2014, 218.)

4.2.2 Näkymä

Näkymällä tarkoitetaan sovelluksen käyttöliittymää eli ikkunoita ja elementtejä, joita käyttäjä näkee ja käyttää. Näkymän ensisijainen tehtävä on näyttää käyttäjälle dataa, jota Malli-luokka säilöö. Tämän takia Näkymällä täytyy olla jotain tietoa Mallista. IOS-kehityksessä näkymät luodaan pääosin Xcoden Interface Builderia käyttämällä, mutta joskus on myös tarvetta muokata Näkymää tai luoda kokonainen Näkymä pelkkää koodia käyttäen. (Mark & Nutting & LaMarche & Olsson 2013, 45.)

Näkymän ei kuitenkaan välttämättä tarvitse tietää paljoa Mallista, esimerkiksi yksinkertaisessa sovelluksessa Näkymälle riittäisi, että se tietää yhden merkkijonon arvon, jotta se pystytään näyttämään käyttöliittymässä. Tämän lisäksi Näkymää käytetään tulkitsemaan käyttöliittymän tapahtumia, kuten pyyhkäisy tai nipistys-ele, jotka Näkymä kääntää viesteiksi kuten -nextPage: tai -zoomOut:, jotka Näkymä lähettää Ohjaimelle. (Bucanek 2014, 219).

4.2.3 Ohjain

Ohjaimella tarkoitetaan koodia joka yhdistää Mallin ja Näkymän toisiinsa. Kun Mallissa tapahtuu muutoksia, saa Ohjain tästä tiedon ja päivittää Näkymän vastaamaan Mallia, mikäli se on tarpeen. Tämä toimii myös toisinpäin, esimerkiksi jos käyttäjä on muokannut Näkymässä merkkijonoa, joka pitää tallentaa tietokantaan. Tässä tapauksessa Näkymä ilmoittaa Ohjaimelle käyttäjän tekemistä toimista ja Ohjain tarkistaa merkkijonon ja välittää tiedon Mallille siitä, että kyseisen merkkijonon arvo on muuttunut ja se pitää tallentaa tietokantaan. IOS-sovelluskehityksessä Ohjain koostuu yleensä luokista, jotka on luotu ja määritelty erityisesti kehitettävälle sovellukselle. Ohjain-luokat voivat olla täysin tyhjästä rakennettuja eli NSObject-luokan aliluokkia, mutta yleensä ne ovat aliluokkia yleisimmistä UIKit-ohjelmistokehyksen tarjoamista luokista, kuten UIViewController-luokasta. Käyttämällä näitä luokkia, saadaan runsaasti toiminnallisuutta sovellukseen kirjoittamatta itse koodia. (Mark & Nutting & LaMarche & Olsson 2013, 45.)

4.3 RestKit

RestKit on Objective-C ohjelmistokehys iOS:lle ja OS X:lle. Sen tarkoituksena on tehdä Rest-pohjaisten web-palveluiden käyttämisestä yksinkertaista ja nopeaa. Se yhdistää yksinkertaisen HTTP-ohjelmointirajapinnan sekä tehokkaan olioiden mappaus-järjestelmän ,minkä ansiosta koodin kirjoittamisen määrä vähenee. RestKitin päämääränä on se, että sovelluskehittäjä voisi keskittyä ajattelemaan enemmän sovelluksensa datamallia, ja jättää vähemmälle pyyntöjen lähettämisen, vastaustan parsimisen ja esitysten rakentamisen resursseille.(RestKit homepage 2014)

RestKitissä oleva olioiden mappaus-järjestelmä on integroitu Applen Core Data-ohjelmistokehyksen kanssa ja RestKitin HTTP-ohjelmointirajapinta hyödyntää AFNetwork-ohjelmistokehyksen verkkotoimintoja. (RestKit homepage 2014)

Esimerkkinä RestKitin käytöstä otetaan kuvitteellinen sovellus, joka lataa uutisartikkeleita Rest-pohjaisesta web-palvelusta. Artikkelilla on otsikko, uutinen, kirjoittaja sekä julkaisupäivä. Palvelusta tuleva JSON näyttäisi seuraavanlaiselta.

```
{ "artikkelit": [{
            "otsikko": "Talvi yllätti autoilijat",
            "uutinen": "Ihmiset ajoivat ojaan",
            "kirjoittaja": "Kimmo Kirjoittaja",
            "julkaisuPaiva": "28/9/2014"
}]}
```

Esimerkki 2. Uutinen JSON-muodossa.

Sovelluksessa olisi taulukkonäkymä joka näyttää ylläolevat tiedot sekä Objective-C luokka datamallia varten. Esimerkki Objective-C luokasta alla.

```
@interface Artikkeli : NSObject
@property (nonatomic, copy) NSString* otsikko;
@property (nonatomic, copy) NSString* uutinen;
@property (nonatomic, copy) NSString* kirjoittaja;
@property (nonatomic) NSDate*   julkaisuPaiva;
@end
```

Esimerkki 3. Artikkeli-luokan datamalli.

Seuraavaksi hyödynnetään muutamia RestKitin mappaus-ominaisuuksia, jotta aiemmin mainittu JSON saadaan muutettua taulukoksi. Tämä taulukko sisältää kaikki artikkelit, joita web-palvelu palauttaa haun perusteella. Alla olevan esimerkin ensimmäisellä rivillä luodaan ilmentymä eli olio nimeltään articleMapping luokasta RKObjectMapping ja mapattavaksi luokaksi asetetaan Artikkeli. ArticleMapping olion avulla konfiguroidaan säännöt, siitä miten JSON-muodossa olevat artikkelit muutetaan Artikkeli-luokan olioiksi. Esimerkissä on käytetty samoja muuttujan nimiä web-palvelun JSON-dokumentissa ja Artikkeli-datamalli luokassa, joten mappauksen konfiguraatio onnistuu mahdollisimman helposti.(RestKit wiki 2014)

```
RKObjectMapping* articleMapping = [RKObjectMapping
mappingForClass:[Artikkeli class]];

[articleMapping addAttributeMappingsFromDiction-
ary:@{

            @"otsikko": @"otsikko",

            @"uutinen": @"uutinen",

            @"kirjoittaja": @"kirjoittaja",

            @"julkaisuPaiva": @"julkaisuPaiva"

}];
```

Esimerkki 4. JSONin muuntaminen taulukoksi.

Tämän jälkeen luodaan RKResponseDescriptor-luokasta olio, jonka avulla tarkastetaan vastaako aiemmin määritelty articleMapping-olio HTTP-vastausta, joka web-palvelusta saadaan. Oliolle on määritelty keyPath-arvoksi artikkelit, jonka täytyy löytyä HTTP-vastauksena tulleesta dokumentista. Esimerkki tästä alapuolella.(RestKit wiki 2014)

```
RKResponseDescriptor *responseDescriptor =

            [RKResponseDescriptor responseDescriptor-
            WithMapping:articleMapping

            method:RKRequestMethodAny

            pathPattern:nil

            keyPath:@"artikkelit"

            statusCodes:RKStatusCodeIndexSetForClass

            (RKStatusCodeClassSuccessful)];
```

Esimerkki 5. HTTP-vastauksen tarkistus.

Kun articleMapping sekä responseDescriptor- oliot on saatu konfiguroitua, voidaan artikkelit ladata käyttämällä RKObjectRequestOperation luokan oliota. Alla olevassa esimerkissä konfiguroidaan NSURLRequest-olio, jolla tarkennetaan mistä osoitteesta resurssit halutaan ladata. (RestKit wiki 2014) Tämän jälkeen alustetaan RKObjectRequestOperation-olio request-oliolla sekä aikaisemmin konfiguroidulla responseDescriptror-oliolla, joka sisältää mappauksen Artikkeli-luokkaa varten.

```
NSURL *URL = [NSURL URLWithString:@"http://esi-
merkki.fi/artikkelit"];

NSURLRequest *request = [NSURLRequest requestWith-
URL:URL];

RKObjectRequestOperation *objectRequestOperation =
[[RKObjectRequestOperation alloc] initWithRe-
quest:request responseDescriptors:@[ re-
sponseDescriptor ]];
```

Esimerkki 6. Artikkeleiden lataaminen.

Seuraavaksi tälle oliolle on luotu kaksi lohkoa, joita kutsutaan riippuen miten HTTP-pyyntö onnistuu. Mikäli HTTP-pyyntö palauttaa mapattavan esi-

tyksen resurssista status koodilla 200, näytetään ladatut artikkelit käyttäjälle. Mikäli HTTP-pyyntö epäonnistuu, siirrytään failure-lohkoon ja näytetään käyttäjälle virheilmoitus. Viimeisellä rivillä aloitetaan operaation toteuttaminen, joka tapahtuu asynkronisesti. (RestKit wiki 2014)

```
[objectRequestOperation setCompletionBlockWithSuc-
cess:^(RKObjectRequestOperation *operation, RKMap-
pingResult *mappingResult)

            {RKLogInfo(@"Ladatut artikkelit %@", map-
            pingResult.array);}

            failure:^(RKObjectRequestOperation *oper-
            ation, NSError *error)

            {RKLogError(@"Operaatio epäonnistui,
            virhe : %@", error);}];

[objectRequestOperation start];
```

Esimerkki 7. Lohkot sekä operaation toteuttaminen.

4.4 Versionhallinta

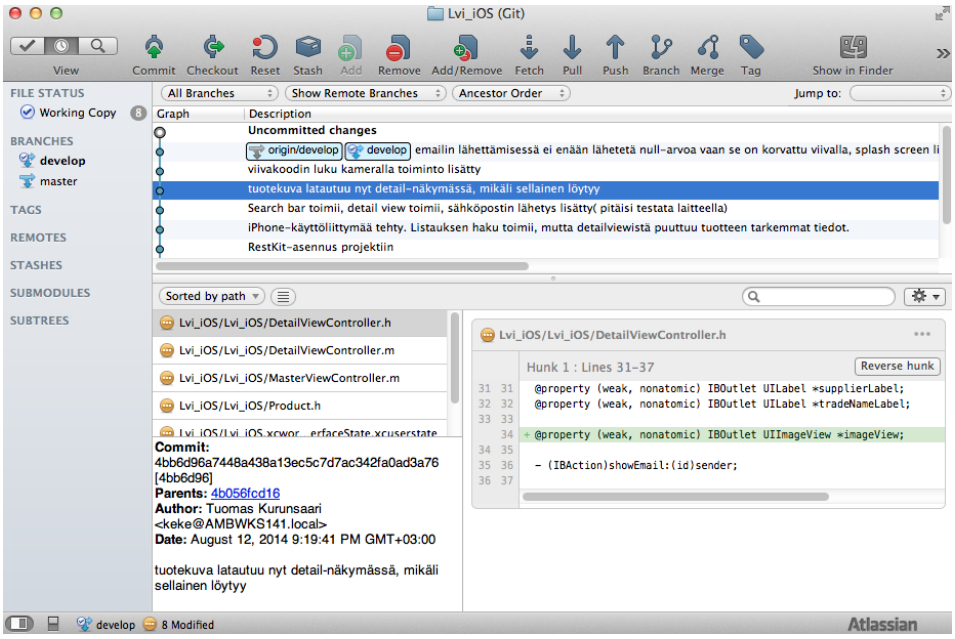
Opinnäytetyön käytännön osuudessa käytettiin Git-versionhallintajärjestelmää. Gitin kanssa käytettiin SourceTree-asiakasohjelmaa, joka helpottti Gitin käyttöä. SourceTreen avulla Git-komennot on helppo suorittaa graafisen käyttöliittymän avulla. Sovelluksen versiot puskettiin Bitbucket-isännöintipalveluun.

4.4.1 Git

Git on avoimeen lähdekoodiin perustuva hajautettu versionhallintajärjestelmä. Se on tarkoitettu sekä pienille että isoille projekteille. Gitin avulla on mahdollista nähdä projektin aiemmat vaiheet ja näyttää erot eri vaiheiden välillä. Projekti on myös mahdollista jakaa useampaan osaan, joita kutsutaan haaroiksi. Nämä haarat mahdollistavat projektin eri osien kehittämisen itsenäisesti. Määräajoin eri haarat on tarkoitus yhteensovittaa yhdeksi kokonaisuudeksi. Git mahdollistaa monien henkilöiden työskentelyn projektissa samanaikaisesti.(Silverman 2013, 1)

4.4.2 SourceTree ja BitBucket

SourceTree on ilmainen Mercurial- ja Git-asiakasohjelma Windowsille ja Macille. Se tarjoaa kokonaan graafisen käyttöliittymän Gitin käyttämiseen. Juuri tämän takia se sopii hyvin Gitin käytön aloittelijoille, sillä sitä käyttämällä Git-komentoja ei tarvitse suorittaa komentoriviltä. SourceTreen avulla pystyy hallinnoimaan sovellusvarastoja paikallisesti tai käyttäen joitain verkkopohjaisista isännöintipalveluista.(SourceTree homepage, 2014) Alapuolella kuva SourceTreen käyttöliittymästä. (Kuva 3)



Kuva 3. SourceTreen käyttöliittymä.

Bitbucket on isännöintipalvelu, joka mahdollistaa tietovarastojen säilyttämisen ja hallitsemisen pilvessä projekteille, jotka käyttävät Git- tai Mercurial-versionhallintajärjestelmiä. Bitbucket tarjoaa ilmaiset tietovarastot projekteille, joissa työskentelee enintään viisi henkilöä. Sen avulla pystyy hallinnoimaan ja jakamaan tietovarastoja projektin jäsenien kanssa. (Bitbucket homepage, 2014)

5 APPLEN OHJEISTUKSET

Suuri ero Android- ja iOS-sovelluskehityksen välillä liittyy niiden sovelluskauppoihin. Applen App Store on suodatettu ja aidattu yhteisö, joka tarkoittaa sitä että Applella on yksinoikeus hyväksyä tai hylätä sovelluksia omiin kriteereihinsä perustuen. Tällä varmistetaan sovellusten yhdenmukaisuus ja laatu. Apple sensuroi sovelluksia, joten väkivaltaiset tai mauttomat sovellukset eivät pääse App Storen valikoimaan. (Pilone & Pilone, 181.) Apple tarjoaakin iOS-kehittäjille oppaita, joihin jokaisen iOS-kehittäjän täytyy tutustua, mikäli haluaa julkaista oman sovelluksensa App Storessa.

5.1 iOS Human Interface Guidelines

IOS Human Interface Guidelines eli HIG, on Applen luoma dokumentti joka tarjoaa iOS-kehittäjälle suosituksia ja määräyksiä siitä millainen sovelluksen käyttöliittymän täytyy olla. Näiden suositusten ja määräysten laiminlyönti saattaa johtaa sovelluksen hylkäämisen App Storen hyväksymisprosessissa. HIG:n ideana on se että käyttäjälle tarjotaan johdonmukainen käyttöliittymä ja visuaalinen ilme eri sovellusten ja käyttöjärjestelmien välillä. Esimerkiksi suurin osa Mac OS X työpöytäsovelluksista omaavat saman ilmeen ja käyttäytyvät samalla tavalla. Alustan pitkäaikaiset käyttäjät suhtautuvat yleisesti varauksella sovelluksiin, jotka eivät noudata HIG:iä, ja myös uudemmat käyttäjät saattavat huomata että jokin sovelluksessa on pielessä. (Allan 2013, 425.)

Minimalistisuus on tällä hetkellä suosiossa Webissä ja myöskin iOS-käyttöliittymissä (Pilone & Pilone, 195). Painikkeista on poistettu varjot ja turhat koristeet, eikä käyttöliittymän elementeillä yritetä luoda syvyysvaikutelmaa, vaan kaikki näyttää tasaiselta. iOS-sovelluksille on määritelty tietty värisävy, joka tarkoittaa että tällä värisävyllä olevat elementit on tarkoitettu sovelluksen käyttämiseen, hyvä esimerkki tästä on iOS 7:n kalenterisovellus, jossa on käytetty punaista värisävyä. (Kuva 4).

Kuva 4. iOS 7:n kalenterisovellus.

iOS 6:een asti Apple rohkaisi kehittäjiä jäljittelemään käyttöliittymissään tosielämän asioita kuten esimerkiksi Applen iBook-kirjanlukusovellus, jonka käyttöliittymä on suunniteltu näyttämään kirjahyllyltä.(Kuva 5). Kuitenkin iOS 7:n tulon myötä tämä trendi loistaa poissaolollaan.

Kuva 5. iOS 6:n iBooks-kirjanlukusovellus.

Vaikka Apple itsekin ajottain poikkeaa HIG:stä ja jotkut kehittävät ovat skeptisiä sen suhteen tarvitseeko HIG:iä noudattaa tiukasti, on HIG pysynyt hyvänä vertailukohteena käyttökokemuksen mittaamiselle. (Allan 2013, 425.)

5.2 App Store Review Guidelines

App Store Review Guidelines on Applen luoma dokumentti, joka sisältää tapaukset, jonka takia sovellus voidaan hylätä App Storen hyväksymisprosessissa. Apple päivittää dokumenttia aina tarpeen mukaan vastaamaan nykytilannetta. Ohjeistuksissa kielletään lapsille sopimattoman sisällön näyttö sovelluksessa ja kerrotaan että sovelluksen täytyy tehdä jotain hyödyllistä eikä näyttää siltä että se olisi kehitetty huolimattomasti muutamassa päivässä. (Apple Inc. 2014c.) Apple on todella tarkka sen suhteen, minkälaisia linkkejä sovelluksissa esiintyy. Mikäli sovelluksessa esiintyvä linkki vie sivustolle, missä myydään digitaalisia kirjoja, musiikkia, elokuvia tai mitä tahansa muuta, sovellus hylätään, sillä Apple haluaa että sovellusten sisäiset ostot tehdään App Storen In App Purchase-toimintoa käyttäen. (Pilone & Pilone, 182.)

5.3 App Programming Guide for iOS

App Programming Guide for iOS on Applen luoma dokumentti, jossa tuodaan esille keskeisiä ominaisuuksia, joiden avulla sovelluksesta saadaan mahdollisimman hyvin toimiva iOS-alustalla. Sen lisäksi että sovellus tarjoaa hyvin suunnitellun käyttöliittymän, käyttökokemus sisältää myös muita tekijöitä. Käyttäjät odottavat iOS-sovelluksien olevan nopeita ja herkästi reagoivia samalla kuluttaen virtaa niin vähän kuin mahdollista. Sovellusten täytyy tukea kaikkia uusimpia iOS-laitteita, samalla kuitenkin näyttäen siltä että sovellus olisi suunniteltu juuri käytettävälle laitteelle. Sovelluksen täytyy sisältää tiettyjä resursseja ja dataa, kuten sovelluksen kuvake, aloitusruutu ja ja tieto siitä millaisen laitteiston sovellus vaatii. App Store päättelee tämän avulla, voiko käyttäjä asentaa sovelluksen laitteeseensa. Sovellusten täytyy myös toimia tehokkaasti moniajo-ympäristössä. (Apple Inc. 2014d.)

6 LVIDROID-SOVELLUS

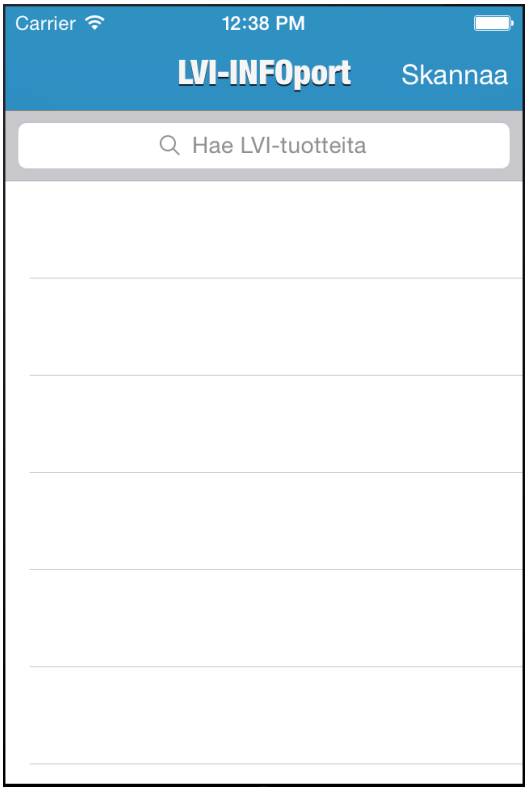
Tässä luvussa tutustutaan sovellukseen, joka opinnäytetyön aikana kehitettiin. Aluksi käydään läpi alkuperäisen Androidille kehitetyn LviDroid-sovelluksen toiminnot, jonka esitellään iOS:lle kehitetty LviDroid-sovellus. Lopuksi kerrotaan kehitetyn sovelluksen käyttöliittymän suunnittelusta sekä sovelluksen toiminallisuuden kehittämisestä.

6.1 LviDroid-Android

Opinnäytetyön alussa tutustuttiin Ambientian kehittämään Android-versioon LviDroid-sovelluksesta. Sovelluksen avulla pystyy etsimään LVI-numeron tai viivakoodin avulla tietoa erilaisista LVI-tuotteista port.lvi-info.fi-palvelusta. Palvelu on kehitetty LVI-numero Oy:ta varten, joka ylläpitää LVI-numeroiden tuoterekisteriä. Palvelusta on olemassa myös web-sovellus osoitteessa www.lvi-info.fi, jonka avulla tuotteita voi selata. LviDroid sovelluksen käyttäminen on varsin yksinkertaista. Sovelluksen avautuessa ilmestyy tekstikenttä, johon käyttäjä voi kirjoittaa haluamansa tuotteen LVI-numeron jonka jälkeen painetaan Hae-nappia, tämän jälkeen sovellus näyttää tuotteen tarkemmat tiedot omana listanaan. Tuotteita pystyy hakemaan myös kirjoittamatta koko LVI-numeroa, jolloin sovellus listaa kaikki LVI-tuotteet jotka alkavat kyseisellä numerolla, jonka jälkeen käyttäjä voi valita tuotteen painamalla sitä, jolloin tarkemmat tiedot tuotteesta näytetään käyttäjälle. Tämän lisäksi käyttäjä voi hakea tiettyä tuotetta lukemalla viivakoodin laitteensa kameraa hyödyntäen. Sovelluksessa on myös mahdollista lähettää tietyn tuotteen kaikki tiedot sähköpostilla haluttuun sähköpostiosoitteeseen. Sovelluksen käyttöliittymän on suunniteltu tablet-laitteille.

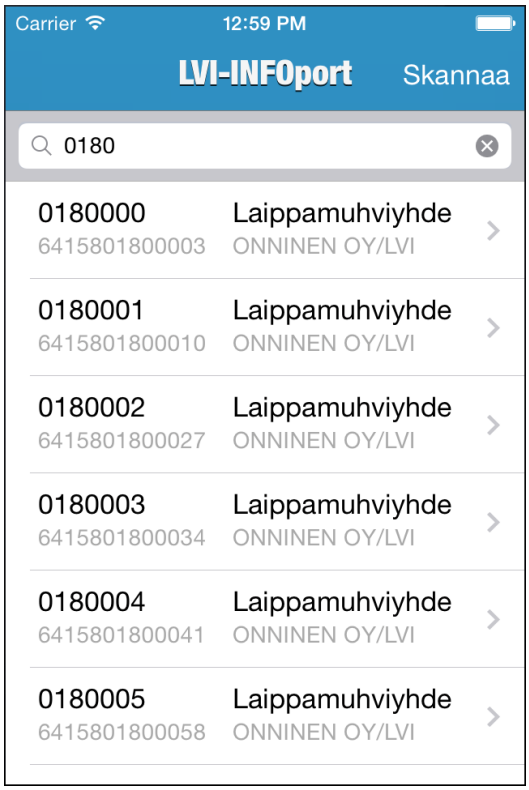
6.2 LviDroid-iOS

LviDroidin iOS-versiota lähdettiin kehittämään ensisijaisesti iPhone-käyttöliittymälle. Sovellus kehitettiin iOS 7:lle, mutta testattiin toimivaksi myös iOS 8-käyttöjärjestelmällä. Sovellus sisältää samat toiminnot kuin alkuperäinen Androidille kehitetty LviDroid-sovellus. Sovelluksen käynnistyessä käyttäjälle avautuu alla olevan kuvan näköinen näkymä. (Kuva 6)



Kuva 6. LviDroid-iOS sovelluksen aloitusnäkymä.

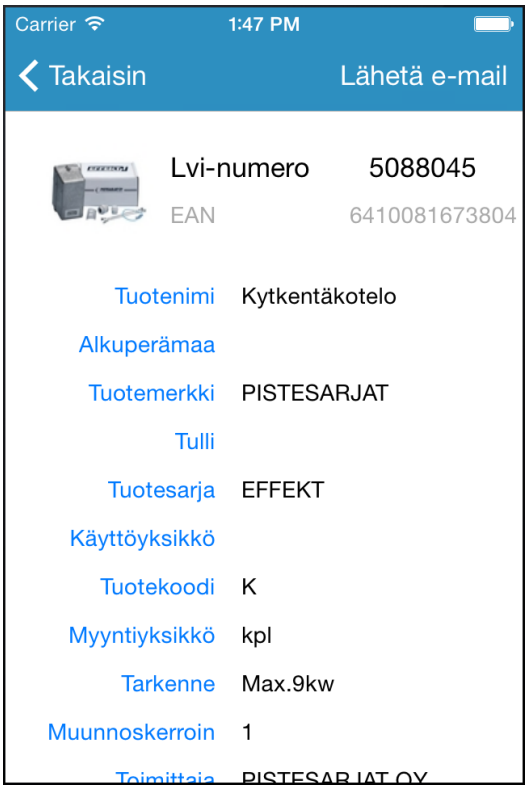
Hae LVI-tuotteita-hakukenttään voi kirjoittaa haluamansa LVI-tuotteen LVI-numeron tai osan siitä, jolloin sovellus listaa kaikki hakua vastaavat tuotteet hakukentän alapuolelle taulukkonäkymään. Alla olevassa esimerkissä tuotteita on haettu pelkästään osalla LVI-numeroa eli tässä tapauksessa numerolla 0180. (Kuva 7)



Kuva 7. Listaus hakua vastaavista LVI-tuotteista.

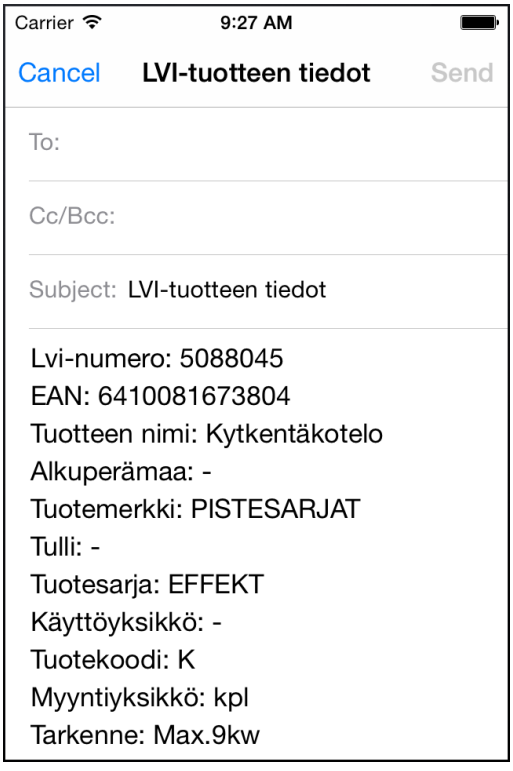
Tuotteen tarkempiin tietoihin pääsee käsiksi painamalla haluttua tuotetta jolloin sovellus siirtyy uuteen näkymään, jossa tuotteen yksityiskohtaiset tiedot ovat esillä. Sovelluksessa pystyy hakemaan LVI-tuotetta myös viivakoodin perusteella. Aloitusnäkymässä näkyvää Skannaa-painiketta painaessa sovellus siirtyy käyttämään laitteen kameraa, jonka avulla viivakoodi voidaan lukea. Kun viivakoodi on kameran avulla tunnistettu, siirtyy sovellus uuteen näkymään jossa näkyy löydetyn LVI-tuotteen tarkemmat tiedot.

Tuotteen tarkemmista tiedoista näkyvät tuotteen LVI-numero, ean, tuotteen nimi, alkuperämaa, tuotemerkki, tulli, tuotesarja, käyttöyksikkö, tuotekoodi, myyntiyksikkö, tarkenne, muunnoskerroin, toimittaja, kauppanimi sekä kuva tuotteesta mikäli sellainen löytyy. (Kuva 8)



Kuva 8. LVI-tuotteen yksityiskohtaiset tiedot.

Näkymässä jossa tuotteen yksityiskohtaiset tiedot näkyvät, on Lähetä e-mail-painike jota painamalla avautuu näkymä sähköpostin lähettämiseen. (Kuva 9) Sovellus asettaa viestikenttään automaattisesti tuotteen tiedot ja sähköposti lähetetään laitteeseen liitetyltä sähköpostitililtä.



Kuva 9. Sähköpostin lähetys.

6.3 Käyttöliittymän suunnittelu

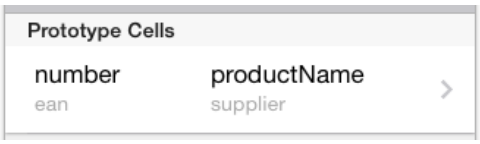
Sovelluksen käyttöliittymän suunnittelu pohjautuu suurin osin Xcoden tarjoamaan Master-Detail-Application- malliin. Master- näkymä on sovelluksen aloitusnäkymä, johon listataan haetut tuotteet, kun taas sovelluksen Detail-näkymässä ovat tuotteen yksityiskohtaiset tiedot. Sovellukseen on lisätty alkuperäisestä mallista poiketen yksi ylimääräinen näkymä, johon siirrytään skannaamaan viivakoodia laitteen kameran avulla. Sovelluksen käyttöliittymä pyrkii noudattamaan iOS 7:n yksinkertaista ja minimalistista muotoilua. Sovelluksen käyttöliittymä on suunniteltu laitteella pystysuunnassa käytettäväksi, mutta on helposti käytettävä myös vaakasuunnassa.

IOS 7:n käyttöliittymä on riisuttu ylimääräisistä koristeista ja tarpeettomat palkit ja painikkeet on poistettu. IOS 7:n käyttöliittymä pyrkii olemaan huomaamaton, jotta sen käyttävät voivat keskittyä paremmin sovellusten sisältöön. (Apple Inc. 2013.) Vaikka iOS 6:n ja iOS 7:n käyttöliittymä muotoilut ovatkin hyvin lähelle toistensa näköisiä, on niissä kuitenkin merkittäviä eroja. Näistä eroista kenties suurin on se kuinka iOS 6 ja iOS 7 käsittelevät yksittäisiä käyttöliittymän elementtejä. IOS 6:ssa lähes jokaisen elementin ympärille on selkeästi määritelty reunus ja usein sovellusten tekstuurit ovat muotoiltu imitoimaan tosielämän asioita. IOS 7:ssa vain muutamat elementit ovat selvästi rajattu, jolloin sisältö ja ohjaimet sekottuvat keskenään tuottaen sujuvan käyttökokemuksen. Otetaan esimerkiksi yleinen UIButton-elementti, joka on iOS 6:ssa rivi tekstiä pyöristetyn suorakulmion sisällä, jota varjo ympäröi korostamassa että se todellakin on painike. IOS 7:ssa taas tämä painike on yksinkertaisesti rivi paksunnettua tekstiä ilman reunoja. (Buttfielf-Addison & Manning & Nugent 2014, 90–91.)

6.4 Sovelluksen toiminnallisuuden toteutus

Sovelluksen toteutus käynnistyi tutustumalla iOS:lle kehitettyihin ohjelmistokehyksiin, joita voitaisiin käyttää sovelluksessa LVI-tuotteiden tietojen hakemisessa Rest-rajapinnasta. Käyttäjien kokemuksia ja iOS-tutoriaaleja hyödyntäen päädyttiin RestKit-ohjelmistokehykseen, joka vaikutti helppokäyttöiseltä ja oli todella kattavasti dokumentoitu. Ennen RestKitin liittämistä projektiin täytyi asentaa CocoaPods-paketinhallintajärjestelmä, jonka avulla voidaan määritellä ja hallita kolmannen osapuolen kirjastoja. RestKitin asennus CocoaPodsin avulla luo uuden .xcworkspace-päätteisesen tiedoston alkuperäisen .xcodeproj-päätteisen tiedoston rinnalle. Asennuksen jälkeen projekti avataan .xcworkspace-päätteisestä tiedostosta, johon RestKitin koodit ovat asentuneet.

Ensimmäiseksi kehitettiin sovelluksen aloitusnäkymässä olevaan Taulukko-näkymään mukautetut solut. Vakiona taulukon soluissa on vain yksi otsikko, ja sovellusta varten soluun tarvittiin neljä otsikkoa alla olevan kuvan mukaan. (Kuva 10) Taulukon solussa näkyy LVI-tuotteen LVI-numero, nimi, ean-koodi ja tuotteen toimittaja. Mukautettua solua varten luotiin oma Objective-C-luokka, joka on UITableViewCell-luokan aliluokka.



Kuva 10. Taulukkonäkymän mukautettu solu.

Sovellus tarvitsee viivakoodin lukemiseen kameraa ja tämän takia tutustuttiin ohjelmistokehyksiin, joiden avulla viivakoodeja voidaan lukea. Ohjelmistokehyksiä viivakoodien lukemiseen löytyi monia, mutta jotkut näistä olivat vanhentuneita ja joissan ohjelmistokehyksen dokumentaatio oli todella vähäistä. Lopulta päädyttiin iOS7:ssa päivittyneeseen AVFoundation-ohjelmistokehykseen, joka mahdollistaa muunmuassa QR-koodien sekä viivakoodien lukemisen kameran avulla. Tämän takia sovelluksen viivakoodin lukua ei voida käyttää iOS6:tta käyttävissä laitteissa, mutta iOS8:ssa viivakoodin luku on mahdollista.

Sovelluksen täytyy myös pystyä lähettämään sähköpostia. Sähköpostin lähettämiseen käytetään iOS:n omaa MFMailComposeViewController-luokkaa. Luokka tarjoaa käyttöliittymän jonka avulla sähköpostiviestiä voidaan muokata ja lopuksi lähettää. Sähköposti lähetetään laitteeseen liitetyltä sähköpostitililtä.

Alla olevalla esimerkkikoodilla sähköpostinäkymä saadaan näkyviin. Ensin tarkistetaan, voiko sovellusta käyttävä laite lähettää sähköpostia, jonka jälkeen asetetaan arvot otsikkoa, viestin runkoa ja sähköpostia varten. Näitä valmiiksi asetettuja arvoja käyttäjä voi muokata kun sähköpostinäkymä ilmestyy ruudulle.

```
if ([MFMailComposeViewController canSendMail]){
            MFMailComposeViewController *mail =
            [[MFMailComposeViewController alloc]
            init];
            mail.mailComposeDelegate = self;
            [mail setSubject:@"Esimerkki Otsikko"];
            [mail setMessageBody:@"Viestin runko!"
            isHTML:NO];
            [mail setToRecipients:@[@"Email@esi-
            merkki.com"]];
            [self presentViewController:mail ani-
            mated:YES completion:NULL];
            }
            else{
             NSLog(@"Tämä laite ei pysty lähettämään
            e-mailia");}
```

Esimerkki 8. Sähköpostinäkymän esille tuonti.

7 YHTEENVETO

Opinnäytetyön alkuperäisenä tavoitteena oli portata LviDroid-sovellus iOS-alustalle. Alkuperäinen LviDroid-sovellus oli kehitetty Android-käyttöjärjestelmälle opinnäytetyön toimeksiantajana toimivan Ambientia Oy:n toimesta. Tavoite onnistui hyvin ja tutkimuskysymyksiin onnistuttiin vastaamaan. Yhtenä tavoitteista oli testata myös J2ObjC-työkalun käyttöä Androidilta iOS:lle porttauksessa, mutta ajanpuutteen takia tämä jäi toteuttamatta käytännön osuudessa.

Kysymykseen iOS:n ja Androidin eroista saatiin vastaus tutustumalla lukuisiin iOS-ohjelmointia käsitteleviin kirjoihin sekä internetistä löytyviin tutoriaaleihin. Kysymykseen Applen ohjeistuksista sovellusten kehittämiseen liittyen, saatiin vastaus tutustumalla tarkemmin Applen luomiin dokumentteihin, jotka käsittelevät tarkasti sen miten sovellus täytyy Applen mukaan kehittää.

Opinnäytetyön aikana saatiin kehitettyä sovellus, jossa on samat toiminnot kuin alkuperäisessä LviDroid-sovelluksessa. Kuitenkaan sovellus ei toistaiseksi ole valmis App Storessa julkaisuun, sillä sovelluksen julkaisuprosessi rajattiin opinnäytetyöstä pois. Muutamia ominaisuuksia kuten graafista suunnittelua vaativat sovelluksen kuvakkeet ja aloitusruutu puuttuvat sovelluksesta. Sovellus tarjoaa kuitenkin hyvän lähtökohdan LviDroid-sovelluksen julkaisuun App Storessa, mikäli sen kehitystä halutaan jatkaa.

Opinnäytetyön alussa iOS-ohjelmointi oli minulle kokonaan uutta asiaa. Minulla oli ennestään kokemusta Android-ohjelmoinnin perusteista, mikä kuitenkin sisältää huomattavan paljon samankaltaisuuksia iOS:n kanssa. Vaikka Objective-C:n ja Javan syntaksi eroaa merkittävästi toisistaan, ovat molemmat kuitenkin oliopohjaisia ohjelmointikieliä. Molemmissa kielissä luodaan luokkia ja metodeja. Molemmilla alustoilla myös erotetaan sovelluksen käyttöliittymä ja logiikka toisistaan. Opinnäytetyön aikana saatiin hyvää kokemusta iOS-sovelluskehityksestä ja tietämys kasvoi myös Rest-arkkitehtuurimallin ja MVC-suunnittelumallin perusteista.

LÄHTEET

Allan, A. 2013. Learning iOS Programming, Third Edition. O'Reilly

Apple Inc. 2013. iOS 7 Design. Viitattu 26.9.2014. https://www.apple.com/fi/ios/design

Apple Inc. 2014. Cocoa Touch Frameworks. Viitattu 17.7.2014. https://developer.apple.com/technologies/ios/cocoa-touch.html

Apple Inc. 2014a. Apple Brings Vibrant Colors & iSight Camera to Most Affordable iPod touch Model. Viitattu 13.7.2014. http://www.apple.com/pr/library/2014/06/26Apple-Brings-Vibrant-Colors-iSight-Camera-to-Most-Affordable-iPod-touch-Model.html

Apple Inc. 2014b. iPad Käyttöopas. Viitattu 16.7.2014. http://manuals.info.apple.com/MANUALS/1000/MA1595/fi_FI/ipad_kayttoopas.pdf

Apple Inc. 2014c. App Store Review Guidelines. Viitattu 20.10.2014. https://developer.apple.com/app-store/review/guidelines/

Apple Inc. 2014d. App Programming Guide for iOS. Viitattu 28.10.2014 https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html

Apple Inc. 2014e. Apple Announces iPhone 6 & iPhone 6 Plus—The Biggest Advancements in iPhone History. Viitattu 9.12.2014. https://www.apple.com/pr/library/2014/09/09Apple-Announces-iPhone-6-iPhone-6-Plus-The-Biggest-Advancements-in-iPhone-History.html

Bitbucket homepage. 2014. Bitbucket Features. Viitattu 6.12.2014 https://bitbucket.org/features

Bucanek, J. 2014. Learn iOS 7 App Development. Apress

Budiu, R. 2013. Mobile: Native Apps, Web Apps, and Hybrid Apps. Viitattu 17.7.2014, http://www.nngroup.com/articles/mobile-native-apps/

Buttfield-Addison, P., Manning J. & Nugent, T. 2014. Learning Cocoa with Objective-C. O'Reilly

Flanders, J. 2009. RESTful.NET. O'Reilly

Franco, L & Mendelowitz, E. 2013. Hello! IOS Development. Manning

Silverman, R. 2013. Git Pocket Guide. O'Reilly

GSMArena, 2014, Apple iPhones reach 500 million sales, iPad pass 200 million. Viitattu 1.12.2014 http://www.gsmarena.com/apple_iphones_reach_500_million_sales_ipad_pass_200_million-news-8683.php

Harris, N. 2014. Beginning iOS Programming. Wrox

J2ObjC homepage, 2014, A Java to iOS Objective-C translation tool and runtime. Viitattu 3.12.2014
https://github.com/google/j2objc

J2ObjC wiki, 2014. The Motivation behind J2ObjC. Viitattu 3.12.2014 https://github.com/google/j2objc/wiki

Liao, S. 2014. Migrating to Android for iOS Developers. Apress.

Mark, D., Nutting, J., LaMarche, J. & Olsson, F. 2013. Beginning iOS 6 Development. Apress

Morgenstern, D. 2013. Do iOS users really want mobile web applications. Viitattu 17.7.2014. http://www.zdnet.com/do-ios-users-really-want-mobile-web-applications-7000013326/

Pilone, T & Pilone, D. 2014. Head First iPhone & iPad Development, 3rd Edition. O'Reilly

RestKit homepage, 2014, RestKit. Viitattu 29.9.2014. http://restkit.org/

RestKit wiki, 2014, Object Mapping Overview. Viitattu 1.10.2014. https://github.com/RestKit/RestKit/wiki/Object-Mapping

SourceTree homepage, 2014, A free Git & Mercurial client for Windows or Mac. Viitattu 6.12.2014 http://sourcetreeapp.com/