Juha Leppä

Palvelinyhteydet iOS-käyttöjärjestelmästä

Opinnäytetyö

Kevät 2013

Seinäjoen ammattikorkeakoulu

Tietotekniikka

Seinäjoen ammattikorkeakoulu
SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

SEINÄJOEN AMMATTIKORKEAKOULU

Opinnäytetyön tiivistelmä

Koulutusyksikkö: Seinäjoen ammattikorkeakoulu

Koulutusohjelma: Tietotekniikka

Suuntautumisvaihtoehto: Ohjelmistotekniikka

Tekijä: Juha Leppä

Työn nimi: Palvelinyhteydet iOS-käyttöjärjestelmästä

Ohjaaja: Petteri Mäkelä

Vuosi: 2013 Sivumäärä: 37 Liitteiden lukumäärä: 0

Opinnäytetyössä tutkitaan palvelinyhteyksiä iOS-käyttöjärjestelmässä. Opinnäytetyössä perehdytään aluksi iOS ohjelmointiympäristönä, mobiilipalvelinyhteyksiin yleisellä tasolla ja palvelinyhteyksiin. Työssä perehdytään myös iOS-sovelluskehityksessä käytettävään Xcode-ohjelmointiympäristöön.

Opinnäytetyössä toteutetaan palvelinratkaisu, minkä on oltava mahdollisimman helposti laajennettavissa eri alustoille, toimintavarma, tietoturvallinen, sekä monia palvelinyhteyksiä mahdollistava ratkaisu.

Opinnäytetyössä toteutetun sovelluksen tietokanta sijaitsee palvelimella, johon otetaan kulloinkin paras mahdollinen datayhteys. Datayhteyden avulla tarvittavat tiedot ladataan sovellukseen. Sovelluksesta määritellään opinnäytetyössä vaatimukset, käyttötapaukset sekä perehdytään valmiiseen toteutukseen. Sovellus tehdään siten, että se toimii sekä iPadillä että iPhonella.

Avainsanat: palvelin, iOS, iPad, iPhone, ohjelmointi, sovellus

SEINÄJOKI UNIVERSITY OF APPLIED SCIENCES

Thesis abstract

Faculty: School of Technology

Degree programme: Information Technology

Specialisation: Software Engineering

Author: Juha Leppä

Title of thesis: Network Connections of the iOS operating system

Supervisor: Petteri Mäkelä

Year: 2013 Number of pages: 37 Number of appendices: 0

The objective of this thesis was to study the network connections of the iOS operating system. The study was started with the iOS development environment, mobile network connections and iOS operating system from the perspective of network connections. Also Xcode development program was briefly studied.

The outcome of this thesis is a software application. The application has a database which is server based. Each time the best possible connection is made to the database. A description of the iOS application that was designed and implemented was a part of the documentation of this study.

Keywords: iOS, Apple, server, programming, application

SISÄLTÖ

Kuvio- ja taulukkoluettelo

Käytetyt termit ja lyhenteet

Apple	Apple on yhdysvaltalainen yritys, joka valmistaa IT-alan tuotteita ohjelmistoista laitteisiin
App Store	Applen verkkokauppa sovelluksille
API – Application Programming Interface	Ohjelmointirajapinta
Cocoa	Mac OS X -sovellusten ohjelmointirajapinta
Cocoa Touch	iOS-sovellusten ohjelmointirajapinta
iCloud	Pilvipalvelu, mihin voi tallentaa sisältöä Applen tekemistä laitteista. Palveluun tallennetaan esimerkiksi varmuuskopiot iOS-järjestelmää käyttävistä laitteista
Interface Builder	Xcode-ohjelmointityökaluun integroitu käyttöliittymän kehitystyökalu
iOS	iOS on Applen kehittämä käyttöjärjestelmä, jota käytetään iPhone-, iPad-, iPod Touch- ja Apple TV -laitteissa
iOS-Simulator	Xcode-ohjelmointityökaluun integroitu simulointityökalu, jolla simuloidaan sovelluksen suoritusta.
iPad	Applen taulutietokone
iPhone	Applen älypuhelin
iTunes	Applen kehittämä musiikkisoitinohjelma, johon on integroitu sovelluskauppa ja iOS-laitteiden synkronointi
Objective-C	Ohjelmointikieli, jolla kehitetään sovelluksia Mac-tietokoneille ja iOS-laitteille

Push Notification Applen valmistama palvelu, joka välittää viestejä palvelimen kautta kolmannen osapuolen sovelluksista Applen laitteisiin.

SDK – Software Development kit

Sovellusten kehittämiseen tarkoitettu ohjelmistotyökalu

URI – Uniform Resource Identifier

Merkkijono, jolla kerrotaan tiedon paikka (URL) tai nimi (URN)

Web Service Verkkopalvelu, jota tarjotaan tietoteknisten järjestelmien ja laitteiden käytettäväksi.

WiFI Langaton lähiverkkotekniikka

Xcode Applen Mac OS X -käyttöjärjestelmälle kehittämä ohjelmointiympäristö useille laitteille ja ohjelmointikielille

1 JOHDANTO

1.1 Työn tausta

iOS-käyttöjärjestelmä on yksi maailman suosituimmista alustoista sovellusten kehitykseen. Työn tarkoituksena oli koota tärkeimmät tiedot, joita tarvitaan, kun suunnitellaan palvelinyhteyksiä käyttävää sovellusta iOS-käyttöjärjestelmälle.

1.2 Työn tavoite

Opinnäytetyön tavoitteena on toteuttaa palvelinyhteyttä käyttävä sovellus iOS-käyttöjärjestelmälle. Opinnäytetyössä perehdytään iOS-käyttöjärjestelmään ohjelmointialustana, Objective-C-ohjelmointikieleen, Xcode-ohjelmointiympäristöön ja iOS-käyttöjärjestelmää käyttäviin mobiililaitteisiin sekä palvelinratkaisuihin. Opinnäytetyössä tutkitaan eri palvelinratkaisuja, suunnitellaan ja toteutetaan palvelinratkaisua käyttävä sovellus. Työssä perehdytään myös mobiilipalvelinyhteyksien perusperiaatteisiin, yhteysmuotoihin, palvelinratkaisuihin ja sovelluksen toimintaan heikon yhteyden tai katkenneen yhteyden aikana.

Opinnäytetyössä toteutetaan palvelinratkaisu, jonka on oltava mahdollisimman helposti laajennettavissa eri alustoille, toimintavarma, tietoturvallinen, sekä monia palvelinyhteyksiä mahdollistava. Sovelluksen tietokanta sijaitsee palvelimella, mihin otetaan kulloinkin paras mahdollinen datayhteys. Datayhteyden avulla tarvittavat tiedot ladataan sovellukseen.

1.3 Työn rakenne

Opinnäytetyön toisessa osiossa perehdytään iOS-käyttöjärjestelmään ohjelmointialustana yleisellä tasolla, aluksi iOS-käyttöjärjestelmään, sen jälkeen Objective-C-ohjelmointikieleen. Lopuksi tutustutaan kehitystyökaluihin.

Kolmannessa osiossa tutustutaan yleisellä tasolla iOS:n mahdollistamiin verkkopalveluihin. Luvussa esitellään Web Service, Push Notification ja iCloud.

Neljännessä osiossa perehdytään mobiiliverkko-ohjelmoinnin perusperiaatteisiin. Luvussa tulee tutuksi, mitä tulee ottaa huomioon, kun suunnittelee verkkoyhteyksiä käyttävää sovellusta.

Viidennessä osiossa tutustutaan verkko-ohjelmointiin iOS-käyttöjärjestelmässä. Lisäksi esitellään ohjelmoinnissa käytettäviä rajapintoja.

Kuudennessa osiossa tutustutaan opinnäytetyössä toteutettuun sovellukseen. Aluksi luvussa määritellään ohjelmistovaatimukset. Seuraavaksi tutustutaan käyttötapauksiin ja toteutettuun sovellukseen.

2 iOS-järjestelmä

Mobiililaitteet eivät ole vain pieniä tietokoneita, vaan ne ovat luoneet kokonaan uudenlaisen tavan käyttää tekniikkaa ja täysin uudenlaisen käyttäjäkokemuksen. Koska laite on aina mukana ja päällä se mahdollistaa kaiken tärkeän informaation mukana kantamisen. iPhonesta tekee erilaisen tapa, jolla sitä käytetään. Kosketusnäyttö on luonnollisempi käyttää, kuin vanhan malliset älypuhelimet tai tiekoneet (Goldstein 2012, 432). Sovellukset iOS-käyttöjärjestelmälle tehdään Mac-tietokoneilla käyttämällä Xcode-ohjelmointiympäristöä. Ohjelmointikielenä käytetään Objective-C:tä.

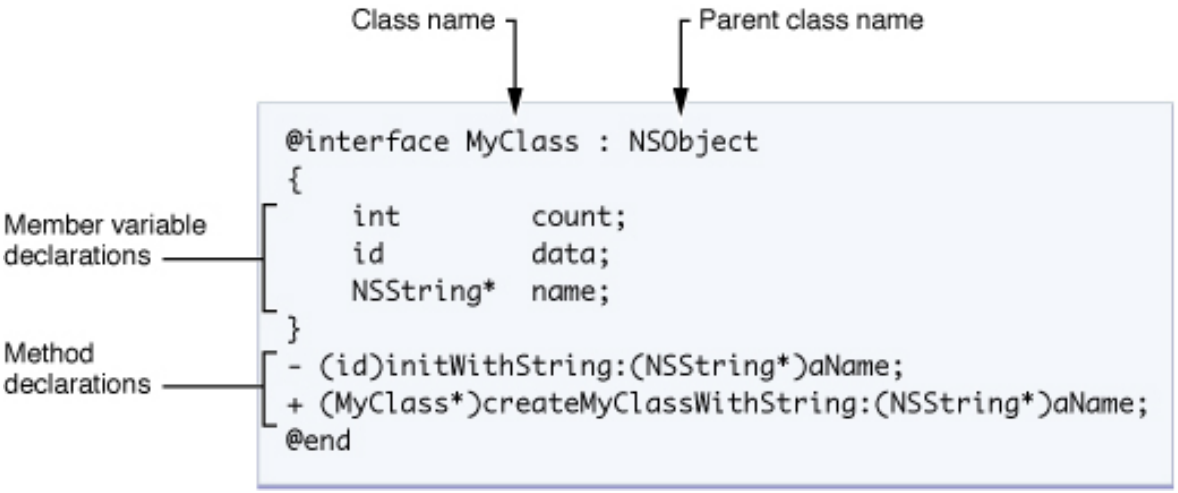
2.1 iOS-käyttöjärjestelmä

iOS-käyttöjärjestelmää käytetään iPhonessa, iPadissa ja iPod Touchissa. Käyttöjärjestelmä pitää yllä laitteen toimintoja ja siihen perustuvat teknologiat, jolla luodaan sovellukset. Käyttöjärjestelmän mukana tulee myös standardisovellukset kuten Mail, Phone ja Safari. iOS:n sovelluskehitystyökalut pitävät sisällään työkalut ja käyttöliittymät kehitykseen, asennukseen ja sovellusten testaamiseen. Natiivit sovellukset tehdään käyttämällä iOS:n ohjelmointirajapintoja ja Objective-C-ohjelmointikieltä ja niitä ajetaan suoraan laitteessa. (Apple, 2012a.)

iOS-arkkitehtuurissa on eri tasoja. Korkeimmalla tasolla iOS toimii komentojen välittäjänä laitteiston ja suoritettavan sovelluksen välillä. Sen sijaan sovellukset keskustelevat rajapintojen avulla laitteiston kanssa, mikä tarjoaa paremman mahdollisuuden laitteistopäivityksille. Tämä malli tarjoaa mahdollisuuden tehdä sovelluksia helposti eri tehoisille laitteille. iOS-teknologioita voidaan kuvata myös kerroksittain. Alimmilla tasoilla on peruspalvelut ja tekniikat, joita kaikki sovellukset käyttävät. Korkeammilla tasoilla on hienostuneempaa tekniikkaa. (Apple, 2012a.) Ohjelmoitaessa iOS-järjestelmälle tulisikin käyttää korkean tason rajapintoja, koska ne tarjoavat olio-ohjelmointitason malleja.

2.2 Objective-C

Objective-C on korkean tason olio-ohjelmointikieli. Se perustuu C-ohjelmointikieleen. Se tukee C-ohjelmointikielen perussyntaksia. iOS-sovellukset ovat ohjelmoitu käyttäen Objective-C-ohjelmointikieltä ja Cocoa Touch -kirjastoa. Cocoa Touch on kokoelma Objectice-C luokkia. Kun ohjelmoidaan iOS-käyttöjärjestelmälle, suurin osa ajasta kuluu olioiden tekemiseen. Oliot ovat Objective-C luokkien instansseja. (Conway & Hillegass 2011, 31.) Luokan määrittelyyn vaaditaan luokan rajapinta ja toteutus. Rajapinta sisältää luokan, muuttujien ja metodien määrittelyn. Rajapinta on yleensä .h tiedostossa. Toteutusosa sisältää luokan metodien koodin. Toteutus on yleensä .m-tiedostossa. Kuviossa 1 on esimerkki, mikä toteuttaa luokan MyClass. Luokan toteutus alkaa kohdasta @interface ja päättyy kohtaan @end. Luokan nimestä seuraavana on kantaluokan nimi. (Apple, 2010b.)



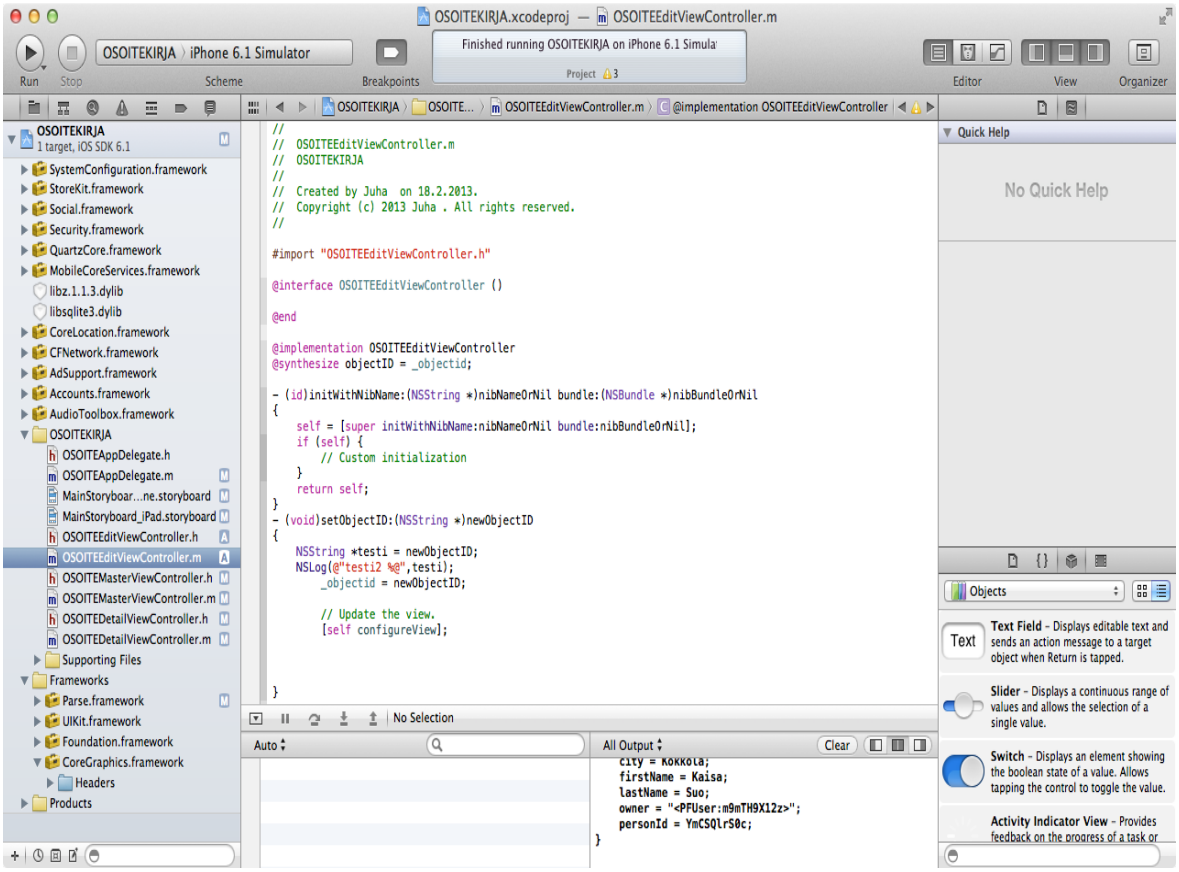
Kuvio 1. Objective-C -luokka
(Apple, 2012b)

2.3 Sovellukset

Seuraavassa osiossa käsitellään ohjelmointiympäristöä, jolla tehdään sovellukset iOS-järjestelmälle. Osiossa perehdytään sovelluksien ominaisuuksiin ja käyttötarkoituksiin.

2.3.1 Xcode

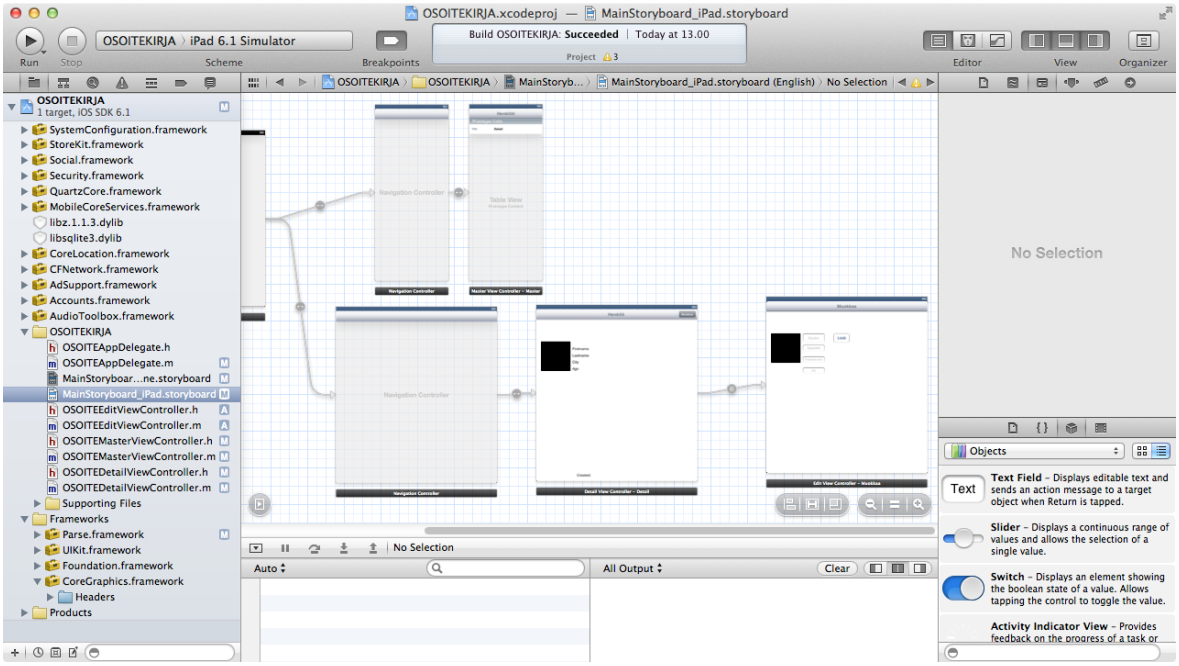
Xcode on ohjelmointiympäristö, jota käytetään kaikkiin kehitysvaiheisiin, ohjelmointiin, käyttöliittymäsuunnitteluun, testaamiseen ja optimointiin sekä sovelluksen toimittamiseen App Storeen (kuvio 2). Xcodella ohjelmoidaan sovellukset Mac-tietokoneille ja iOS-käyttöjärjestelmää käyttäville laitteille. Xcodella ohjelmoidaan aluksi sovellus, minkä jälkeen sovellusta voi testata iOS-simulaattorilla tai suoraan tietokoneeseen liitetyllä laitteella. Testaamisen jälkeen käytetään Instruments-sovellusta, joka löytyy Xcodesta. Instruments-ohjelmalla pystyy tarkistamaan tehdyn sovelluksen suorituskyvyn. Jos aikoo kehittää sovelluksia oikeille laitteille, Apple vaatii liittymisen iOS-kehitysohjelmaan (iOS Development Program) ja rekisteröimään laitteen kehitystarkoitusta varten. (Apple, 2012c.)



Kuvio 2. Xcode-kehitystyökalu

2.3.2 Interface Builder

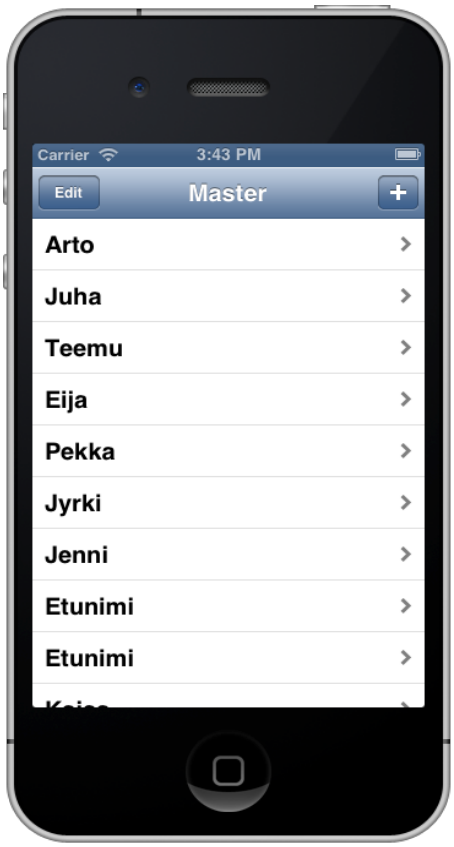
Interface Builder on käyttöliittymäsuunnitteluun käytettävä työkalu. Monissa muissa käyttöliittymäsuunnittelutyökaluissa määritellään aluksi, miltä sovelluksen halutaan näyttävän, minkä jälkeen generoituu koodia. Interface Builder on objektieditori, sillä tehdään ja muokataan objekteja, kuten ikkunoita, nappeja ja tekstejä. Kun objektit ovat valmiita, ne tallennetaan arkistoon. Arkistoa, joka sisältää tiedot käyttöliittymästä kutsutaan XIB-tiedostoksi. (Conway & Hillegass 2011, 5.) Interface Builder mahdollistaa käyttöliittymän suunnittelun ja esikatselun (kuvio 3).



Kuvio 3. Interface Builder -käyttöliittymäkehitystyökalu

2.3.3 iOS Simulator

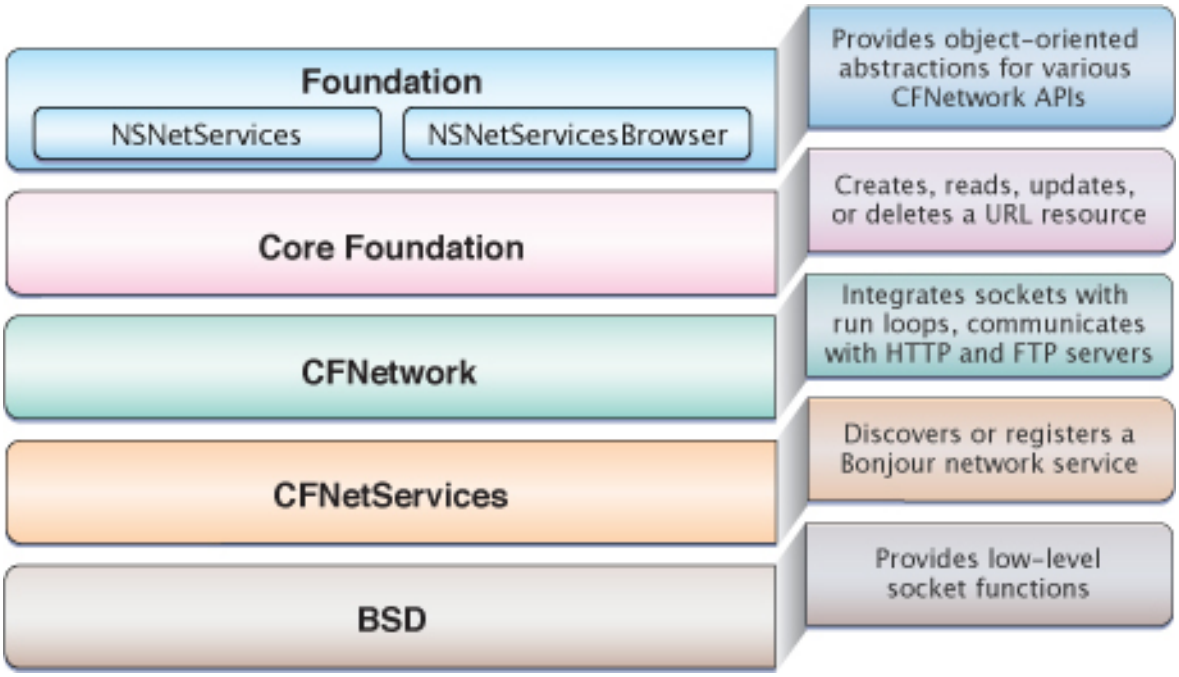
iOS Simulator -simulointityökalulla testataan valmistunut sovellus (kuvio 4). Se mahdollistaa nopean testaamisen kehitysprosessin aikana. Se auttaa löytämään ohjelmistovirheet kehityksen aikana. Sillä simuloidaan iPhonen tai iPadin toimintaa. Sitä voidaan ajatella ensimmäisenä testausvaiheena, minkä jälkeen vasta testataan aidolla laitteella. Sillä voidaan myös simuloida eri käyttöjärjestelmäversioita. Simuloinnilla nähdään miltä sovellus näyttää aidoilla laitteella. Jokaista simuloitua ohjelmistoversiota pidetään omana simulointiympäristönä, itsenäisenä muista versioista, jolla on omat asetuksensa ja tiedostonsa. (Apple, 2013d.)



Kuvio 4. iOS-simulaattorilla suoritetaan Osoitekirja-sovellusta

3 iOS-verkkopalvelut

iOS sisältää useita rajapintoja ja kirjastoja, minkä avulla voidaab lisätä verkko- ja internet-pohjaisia ratkaisuja sovellukseen. Kehittäjät pääsevät käsiksi protokolliin ja palveluihin Foundation- ja Core Foundation -rajapinnoista, ja myös CFNetwork- ja BSD-Socketeista (kuvio 5). Kun käytetään näitä rajapintoja, ei tarvitse kehitystyötä tehdessä valita käytetäänkö WiFi-, vai matkapuhelinverkkoa (kuvio 9). Rajapinta tarkistaa automaattisesti parhaimman siirtotien verkkoon ja vaihtaa sen huomaamattomasti toiseen tarvittaessa. (Apple, 2011e.)



Kuvio 5. iOS-verkkorajapinnat
(Apple, 2011e)

3.1 Web Services (www-sovelluspalvelut)

Web Services on tapa, jolla laite pyytää toiselta laitteelta verkon yli palveluita. Web Servicessä vähintään kaksi elektronista laitetta kommunikoi keskenään. Asiakas-palvelin-malli jakaa sovelluksen toiminnan palvelun tarjoajaan eli palvelimeen ja palvelua käyttäviin eli asiakkaaseen. Asiakas-palvelin-malli on koko internetin toiminnan pohjana. Internetin protokollat kuten HTTP, SMTP ja DNS on rakennettu asiakas-palvelin-mallin pohjalta. Web Servicen avulla iOS-laitteesta saadaan siir-

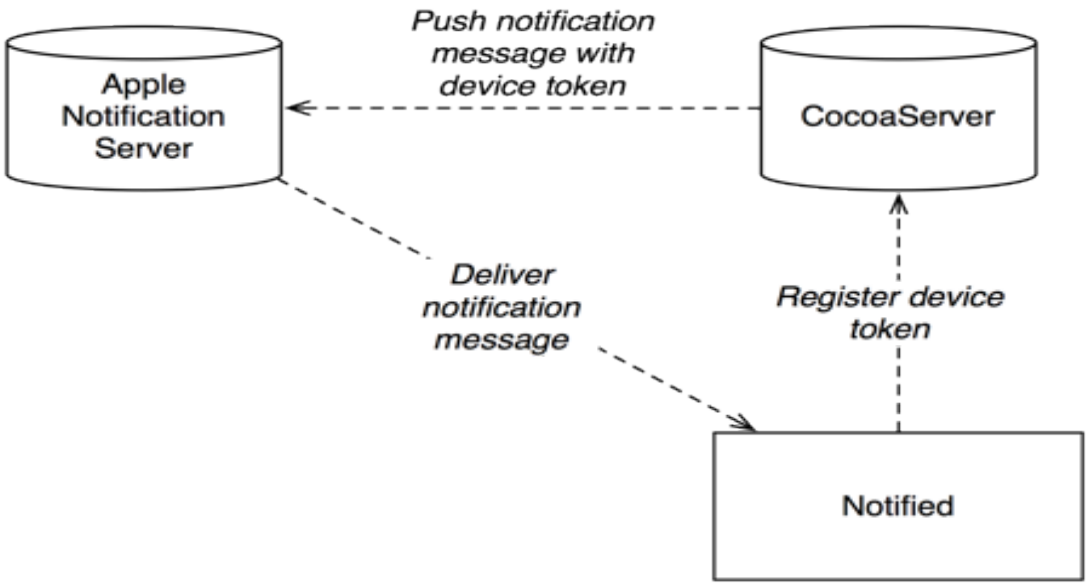
rettyä datan tallennus tai raskasta laskentatyötä palvelimelle. (Conway & Hillegass 2011, 406.)

Internetselain käyttää HTTP-protokollaa palvelimen kanssa kommunikointiin. Yksinkertaisin selaimen ja palvelimen välinen kommunikointi tapahtuu, kun selain lähettää URL-käskyn palvelimelle. Palvelin vastaa kyselyyn lähettämällä selaimelle pyydetyn internetsivun. (Conway & Hillegass 2011, 407.)

Vaikeammissa tapauksissa selaimen kysely voi sisältää parametreja, kuten dataa. Palvelin prosessoi datan ja palautta sen takaisin muokattuna. iOS:lle voi tehdä asiakassovelluksen, mikä hyödyntää HTTP-infrastruktuuria. Koska HTTP-protokolla ei välitä mitä dataa se kuljettaa, asiakassovellus voi vastaanottaa ja lähettää monimutkaista dataa. Data on yleensä XML- tai JSON-muodossa. Web Servicen käyttäminen vaatii iOS-sovelluksessa useimmiten datan muuntamista lähetystä varten XML- tai JSON-muotoon. Datan lähettämisen pitää tapahtua HTTP-käskynä, samoin vastaanoton. Lopuksi vastaanotettu XML- tai JSON-data täytyy muuntaa haluttuun muotoon. (Conway & Hillegass 2011, 408.)

3.2 Push Notification

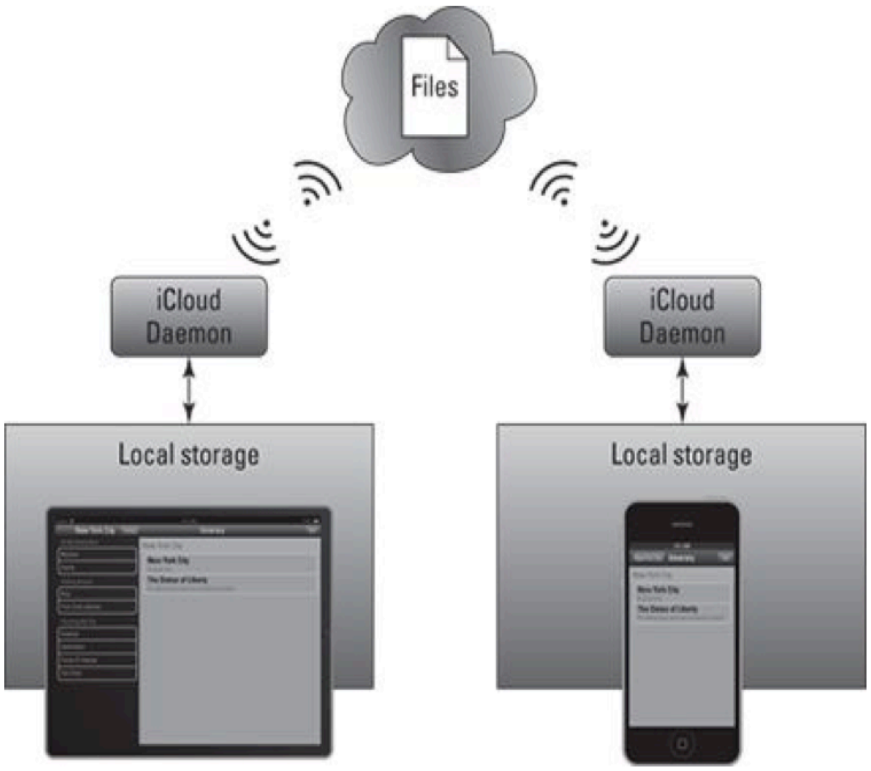
iOS-käyttöjärjestelmässä vain yksi sovellus voi olla päällä kerrallaan. Joissain tilanteissa on hyötyä, että sovellus lähettää laitteen näytölle viestin, kun jokin muutos tapahtuu. Esimerkiksi kun vastaanotetaan viesti tai kokous on alkamassa viiden minuutin päästä. Push Notification on yksi tapa ratkaista ongelma. Se on viesti, joka lähetetään Applen palvelimelta laitteeseen (kuvio 6). Käyttäjä näkee näytöllä viestin ja ohjelman nimen, mistä ilmoitus tuli. Ilmoituksesta kuulu myös ääni, jos laite ei ole äänettömällä. Jos sovelluksen kehittäjä haluaa Push Notification -viestit sovellukseensa. Hänen täytyy hankkia SSL-sertifikaatti ja sovelluksen tulee olla Applen hyväksymä (Conway & Hillegass 2011, 481).



Kuvio 6. Push Notification -viestijärjestelmän toimintaperiaate (Conway & Hillegass 2011, 481)

3.3 iCloud

iCloud on Applen teknologia, mikä tarjoaa tavan jakaa dataa sovelluksien välillä, jos sovellusta suoritetaan monella eri laitteella. Sovellus tallentaa yleensä datan siihen laitteeseen, missä sitä sillä hetkellä suoritetaan, mutta kopio datasta tallennetaan myös iCloudiin. Kun sovellus tekee muutoksen dataan yhdellä laitteella muutokset siirtyvät aluksi iCloud:n taustalla. Onnistuneen tiedonsiirron jälkeen iCloudiin, kopio datasta lähetetään toisille laitteille, joihin sovellus on asennettu (kuvio 7). Mikäli sovellusta ei sillä hetkellä ole käynnissä toisilla laitteilla, muutokset siirtyvät silloin kun sovellus seuraavan kerran käynnistetään. Jos kaksi samaa sovellusta käyttävää laitetta on samassa WiFi-verkossa, muutokset siirtyvät suoraan laitteesta laitteeseen eikä Applen palvelimen kautta. Muutoksien siirtyminen suoraan laitteiden välillä nopeuttaa toimintaa huomattavasti. (Goldstein 2012, 4586.)



Kuvio 7. iCloud-pilvipalvelun toimintaperiaate
(Goldstein 2012, 4586)

4 Mobiiliverkko-ohjelmointi

Seuraavassa kappaleessa tutustutaan mobiiliverkko-ohjelmoinnin perusperiaatteisiin. Asioihin, jotka tulee ottaa huomioon sovelluksen suunnitteluvaiheessa.

4.1 Perusperiaatteet

Täydellisessä maailmassa verkkoyhteydet toimivat aina. Verkkoyhteys on luotettava, viive on lyhyt ja yhteys on nopea. Todellisessa maailmassa verkkoyhteys pelaa vain suurimman osan ajasta, ja se katkeaa mitä erikoisemmilla tavoilla. Tietenkään hyvin kirjoitettu verkko-ohjelmointikoodi ei korjaa oikeasti katkennutta verkkoyhteyttä, mutta huonosti suunniteltu verkkoyhteyskoodi voi helposti tehdä asioista paljon huonompia. Ylikuormittuneelta palvelimelta saattaa kestää vastata kyselyihin 45 sekuntia. Kun sovellus yhdistää palvelimeen 30 sekunnin aikarajalla, se lisää palvelimen työmäärää, mutta ei saa koskaan vastauksena yhtään dataa. Vaikka verkkoyhteys toimisi täydellisesti, huonosti suunniteltu koodi aiheuttaa monia ongelmia mobiililaitteen käyttäjälle. Akun kesto on heikkoa ja sovelluksen toiminta hidastuu tai jopa pysähtyy. (Apple. 2013f.)

4.1.1 Virran ja kaistan säästö

Joka kerta kun sovellus lataa verkkoon tai verkosta, se maksaa käyttäjälle aikaa ja rahaa. Verkkoyhteys vie käyttäjältä aikaa, koska käyttäjän pitää odottaa jonkin sovelluksen toiminnon loppumista ennen kuin voi jatkaa sovelluksen käyttämistä. Datasiirrot käyttävät paljon akkua, joten on selvittävä mahdollisimman vähällä datasiirrolla. Kun suunnittelee verkkoyhteyksiä käyttävää sovellusta, on ohjelmoijan vastuulla minimoida datayhteyden ja virran käyttö. (Apple, 2013g.)

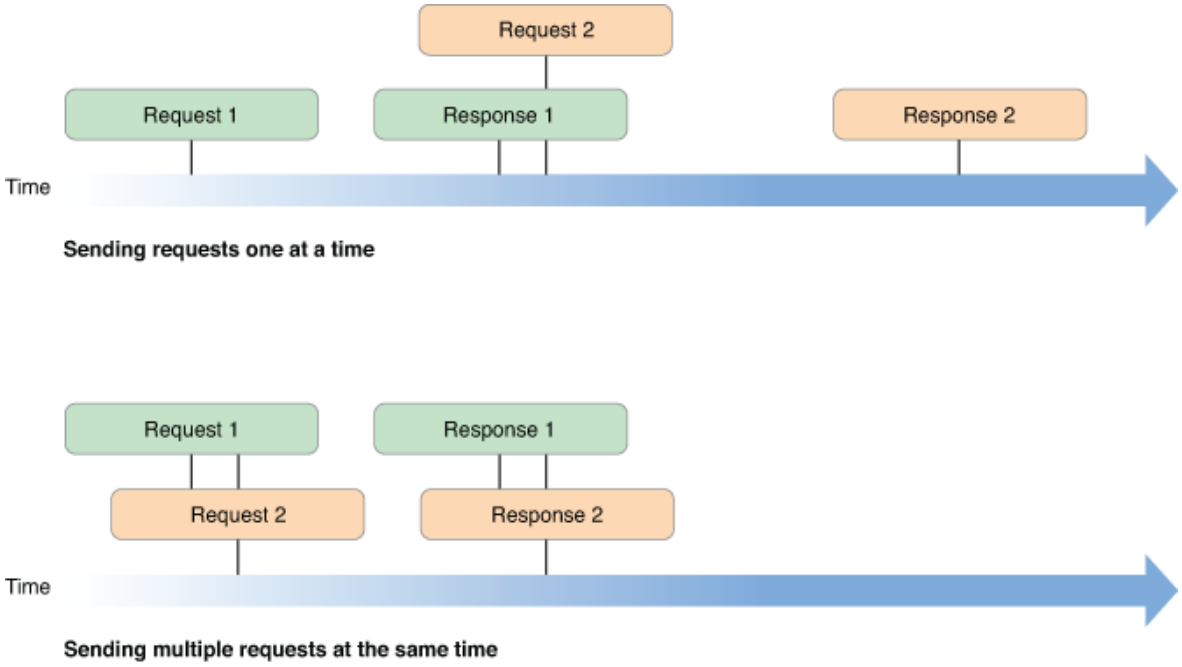
4.1.2 Vaihtelevat verkkoyhteyden nopeudet

Sovellus on suunniteltava siten, että se ottaa huomioon vaihtelevat verkkoyhteyden nopeudet, vaikka sen hetken yhteystapa ei muuttuisi. Pienikin sijainnin muu-

tos voi vaikuttaa nopeuteen huomattavasti. Jopa huoneen vaihtaminen talon sisällä voi muuttaa yhteyden nopeutta huomattavasti. Verkkoyhteyden nopeuden voi selvittää vain käyttämällä yhteyttä. Kun on ladannut pienen määrän dataa voi laskea keskiarvon verkkoyhteyden nopeudelle. Verkkoyhteyden nopeutta pitäisi seurata jatkuvasti. (Apple, 2013h.)

4.1.3 Verkkoyhteyden viive

Kun suunnittelee sovelluksen, tulisi ottaa huomioon myös verkkoyhteydestä johtuva viive. Korkea viive on yleistä tietyn tyyppisillä matkapuhelinverkoilla. Esimerkiksi EDGE-verkossa viive lasketaan sekunneissa. Jo puolen sekunnin viive voi aiheuttaa ongelmia, jos sovellusta ei ole suunniteltu ottamaan huomioon viivettä. Kun sovellus tekee monta verkkokyselyä, se odottaa että ensimmäinen vastaus saapuu ennen kuin se tekee toisen kyselyn. Viiveestä tulee kerrannaista. Ensimmäisen kyselyn viive lisätään toisen kyselyn viiveeseen (kuvio 8). Tämän ongelman voi välttää siten, että sovellus ei jää odottamaan ensimmäisen viestin vastausta, ennen seuraavan viestin lähetystä. Tämä onnistuu vain silloin, kun viestit eivät ole toisistaan riippuvaisia. (Apple, 2013i.)



Kuvio 8. Viiveen vaikutus
(Apple, 2013i)

4.1.4 Verkkoyhteyttä käyttävän sovelluksen testaus

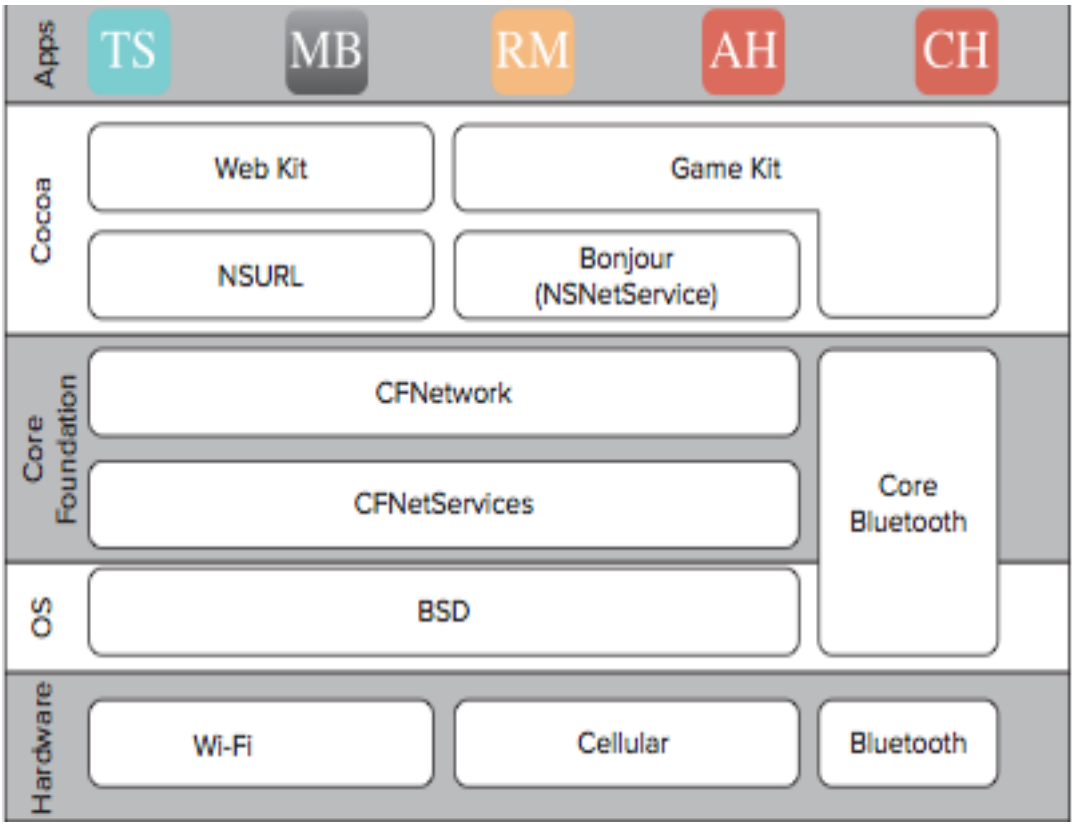
Sovellusta tulisi testata vaihtelevissa olosuhteissa. Xcode tarjoaa testaamiseen työkalun Network Link Conditioner. Sillä voi simuloida erilaisia verkko-ongelmia. Tärkeintä on testata sovelluksen toimintaa vaihtelevissa olosuhteissa. Sovelluksen pitäisi pysyä käytettävänä, vaikka on hidas verkkoyhteys. Kaistan käyttö on saatava minimiin, että tämä onnistuu. Jos verkon viive on useita sekunteja, sovelluksen toiminnot eivät saa hidastua kuin sekunteja, ei minuutteja. Kun verkkoyhteys kadottaa paketteja, sovelluksen toiminta saa vain hidastua, ei lopettaa toimintaansa kokonaan. (Apple, 2013j.)

5 iOS verkko-ohjelmointi

Hyvän iOS-sovelluksen tekee toimiva ja yksinkertainen käyttöliittymä. Yhtä tärkeää on hyvin suunnitellut sovelluksen verkkoyhteydet, jos sovellus käyttää internet-palveluita. Kun suunnittelee sovelluksen arkkitehtuuria tulee tietää, mitä tarkoittavat eri verkkoyhteysrajapinnat. (Cox, Jones, Szumski, Tang 2012, 3.)

5.1 Verkkorajapinnat

On tärkeää ymmärtää Objective-C:n verkkorajapinnat, kun aloittaa verkkoyhteyksiä käyttävän iOS-sovelluksen teon (kuvio 9). Kaikki iOS-sovellukset ovat verkkorajapinnan päällä, mikä koostuu neljästä kerroksesta. Päällimmäisenä on Cocoa-taso, joka sisältää Objectice-C:n ohjelmointirajapinnat URL:n lataamiseen, Bonjourin ja Game Kitin. Cocoan alapuolella on Core Foundation, joka sisältää C-ohjelmointirajapintoja. Esimerkiksi CFNetwork sisältää lähdekoodin suurimmalle osalle sovellustason verkko-ohjelmointikoodista. CFNetwork sisältää helpon verkkorajapinnan, joka sijaitsee CFStream:n ja CFSocket:n päällä. BSD-soketit ovat alin hierarkiataso. Mitä alemmas rajapintatasoja siirtyy, sitä enemmän on oikeuksia, mutta ohjelmointi ei ole yhtä helppoa. Apple suosittelee, että ohjelmoijat pysyvät CFNetwork-tasolla ja sen yläpuolella. CFNetwork on alin rajapinta, joka käynnistää automaattisesti verkkoyhteyden, joten verkkoyhteyden käynnistävää koodia ei tarvitse tehdä. Ohjelmointirajapinnat tuovat paljon toimintoja ja käytettävyyttä ohjelmoijalle. Mitä ylemmäksi tasoja nousee, abstraktion määrä kasvaa. Abstraktio ei kuitenkaan tule ilmaiseksi, osa mahdollisuuksista poistuu, vaikkakin suurimman osan sovelluksista pystyy tekemään ylimpien tasojen rajapintojen avulla. (Cox, Jones, Szumski, Tang 2012, 3, 4.)



Kuvio 9. iOS-verkkoyhteysohjelmointirajapinnat
(Cox, Jones, Szumski, Tang 2012, 4)

5.1.1 NSURLConnection

NSURLConnection on Cocoa-tason ohjelmointirajapinta, joka tarjoaa helpon tavan käsitellä URL-käskyjä, jotka voivat olla yhteydessä Web Service:n. Sen avulla pystyy lataamaan kuvia ja videoita. NSURLConnection-instanssi pystyy kommunikoimaan palvelimen kanssa kahdella tavalla: synkronisesti tai asynkronisesti. Synkroninen yhteystapa on käytännössä unohdettu, koska ohjelman suoritus pysähtyy yhteyden ajaksi. (Cox, Jones, Szumski, Tang 2012, 5.)

NSURLConnection tarvitsee tiedoksi web-sovelluksen sijainnin ja dataa mitä lähettää palvelimelle. Se tarvitsee myös delegaatin. Kun yhteys on käynnistynyt, se alkaa lähettää dataa ja mahdollisesti vastaanottaa. NSURLConnection päivittää joka vaiheessa delegaattinsa tärkeällä tiedolla. (Connway & Hillegass 2011, 410.)

5.1.2 Game Git

Game Kit tarjoaa vertaisverkkovaihtoehdon iOS-sovelluksille. Game Kit on rakennettu Bounjourin päälle. Se ei tarvitse verkkoinfrastruktuuria toimiakseen. Se luo ad-hoc-verkon Bluetooth-yhteyden yli. Se on hyvä vaihtoehto laitteiden väliselle tiedonsiirrolle, jos tarvitaan suora yhteys kahden iOS-laitteen välille, ja WiFi-yhteyttä ei ole tarjolla. Game Kit tarvitsee vain yhteyden tunnisteen, nimen ja yhteystavan kun verkkoa muodostetaan. Se ei tarvitse asetuksia soketeille, eikä muita alemman tason yhteyksiä. (Cox, Jones, Szumski, Tang 2012, 5.)

5.1.3 Bonjour

Bonjour on Applen ratkaisu verkkoyhteyteen, joka ei tarvitse minkäänlaisia asetuksia. Bonjour tarjoaa mekanismin löytää ja yhdistää laitteet verkkoon. Se myös poistaa tarpeen tietää laitteen verkko-osoitteen. Bonjourin avulla löytää toisten laitteiden tarjoamat palvelut automaattisesti, jos molemmilla laitteilla on käytössä Bonjour. (Cox, Jones, Szumski, Tang 2012, 5.)

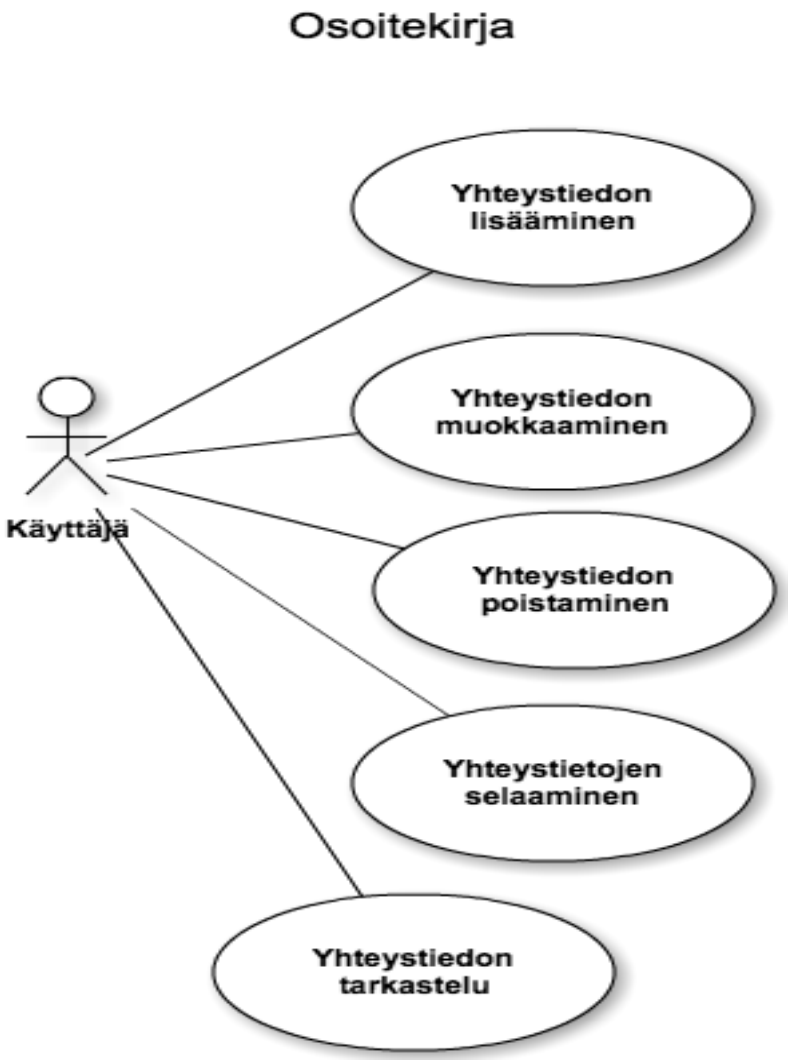
6 Osoitekirja-sovellus

Valmistunut sovellus tarjoaa iOS-käyttäjille vaihtoehtoisen tavan tallentaa yhteystiedot. Tässä opinnäytetyössä sovellus tehdään vain iOS-laitteille. Jatkossa sovellus on tarkoitus tehdä myös muille alustoille. Yhtenä sovelluksen käyttötarkoituksena on tallentaa yhteystiedot palvelimelle. Toisena käyttötarkoituksena on saada yhteystiedot helposti siirrettyä eri laitteiden välillä iOS-alustalla, aluksi vain iOS-laitteiden väilillä, mutta jatkossa myös samojen yhteystietojen siirto eri valmistajien laitteiden välillä.

6.1 Ohjelmistovaatimukset

Osoitekirjasovelluksen lähtökohtana on tarjota erittäin helppokäyttöinen ja tyylikäs sovellus iPhonelle ja iPadlile. Sovelluksella pystyy lisäämään, muokkaamaan ja poistamaan yhteystietoja. Yhteystiedot ovat tallennettuna palvelimelle. Sovelluksen käyttäjinä ovat kaikki, jotka sen lataavat App Storesta. Palvelinratkaisun tulee olla mahdollisimman vakaa, sen tulee toimia eri alustoilla ja sen pitää pystyä käsittelemään mahdollisimman paljon yhteyksiä. Tietoturva on myös erittäin tärkeä, koska palvelimelle tallennetaan ihmisten yhteystietoja.

Käyttötapauskaaviossa kuvataan sovelluksen toiminnot, sekä sovelluksen käyttäjä (kuvio 10).



Kuvio 10. Osoitekirja-sovelluksen käyttötapauskaavio

6.2 Käyttötapaukset

Seuraavassa kappaleessa esitellään Osoitekirja-sovelluksen käyttötapaukset. Käyttötapauksia on neljä yhteystiedon lisääminen, yhteystiedon muokkaaminen, yhteystiedon poistaminen, yhteystietojen selaaminen ja yhteystiedon tarkastelu.

6.2.1 Yhteystiedon lisääminen

Yhteystiedon lisääminen -käyttötapauksessa käyttäjä pääsee lisäämään yhteystietoja sovellukseen. Lisääminen tapahtuu valitsemalla napin, joka sisältää Plus-merkin (kuvio 12). Sovellus lisää aluksi tietokantaan oletusyhteystiedon, minkä voi muokata haluamakseen käyttötapauksessa Yhteystiedon muokkaaminen. Lisätty

yhteystieto tallentuu automaattisesti taustalla palvelimelle. Kun tallennus tapahtuu, sovelluksen toiminta ei pysähdy, ja käyttäjä voi jatkaa sovelluksen käyttöä normaalisti.

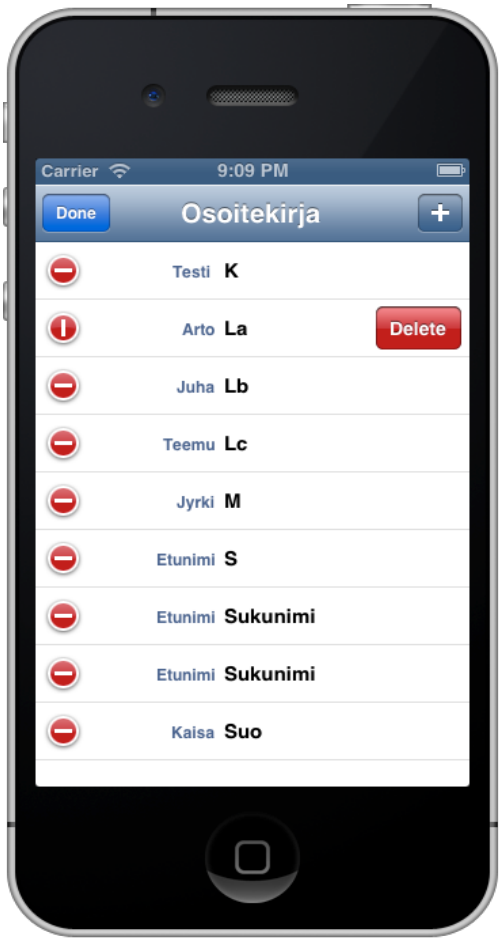
6.2.2 Yhteystiedon muokkaaminen

Yhteystiedon muokkaaminen -käyttötapauksessa käyttäjä voi olemassa olevan yhteystiedon valittuaan muokata sen tietoja. Kun käyttäjä valitsee tekstikentän, näppäimistö tulee automaattisesti esiin. Tallennus tapahtuu silloin, kun käyttäjä valitsee Tallenna-komennon (kuvio 11).

Kuvio 11. Käyttöliittymäkäyttötapauksesta yhteystiedon muokkaaminen

6.2.3 Yhteystiedon poistaminen

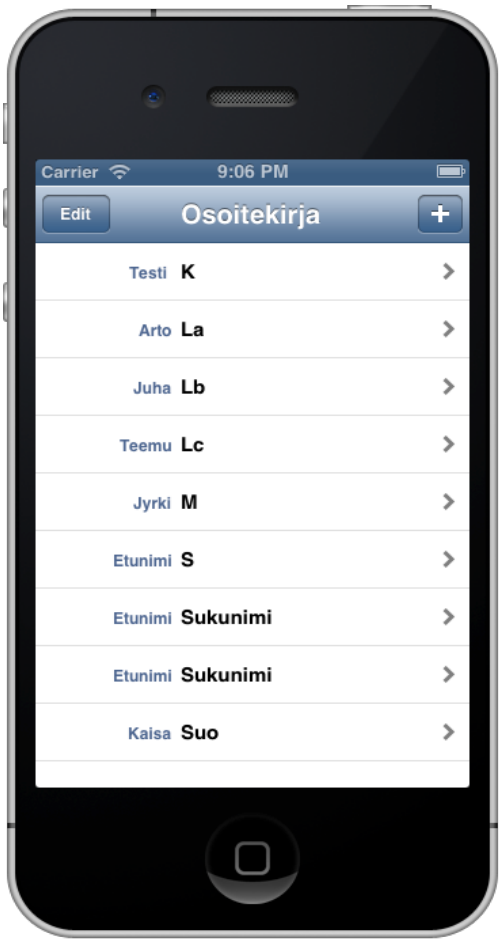
Yhteystiedon poistaminen -käyttötapauksessa käyttäjä voi poistaa olemassa olevan yhteystiedon. Käyttäjä valitsee listalta, joka ladataan palvelimelta, yhteystiedon tai yhteystietoja, jotka haluaa poistaa. Kun Poista-nappia painaa, yhteystiedot poistuvat palvelimelta ja sovelluksesta (kuvio 12).



Kuvio 12. Käyttöliittymäkäyttötapauksista yhteystiedon poistaminen ja lisääminen

6.2.4 Yhteystietojen selaaminen

Yhteystietojen selaaminen -käyttötapauksessa käyttäjä voi selata olemassa olevia yhteystietoja vierittämällä ruutua alas ja ylöspäin (kuvio 13). Yhteystiedot ladataan palvelimelta ohjelmaan taustalla, kun sovelluksen käynnistää.



Kuvio 13. Käyttöliittymäkäyttötapauksesta yhteystietojen selaaminen

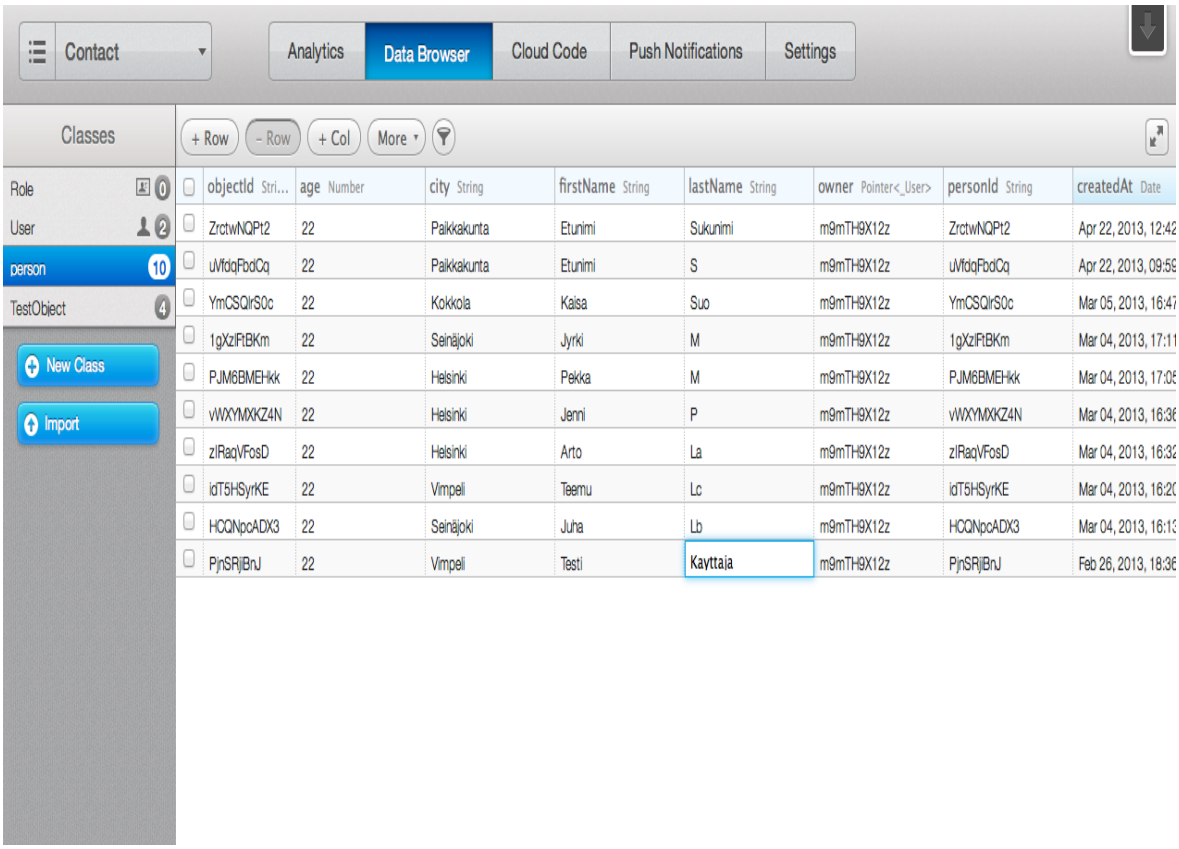
6.2.5 Yhteystiedon tarkastelu

Yhteystiedon tarkastelu -käyttötapauksessa käyttäjä pääsee tarkastelemaan yhteystiedon tietoja, aluksi valittuaan halutun yhteystiedon Yhteystietojen selaus -käyttötapauksessa. Tarkastelu-käyttötapauksesta käyttäjä pääsee muokkaamaan yhteystietoa painamalla Muokkaa (kuvio 14).

Kuvio 14. Käyttöliittymäkäyttötapauksesta yhteystiedon tarkastelu

6.3 Palvelinratkaisu

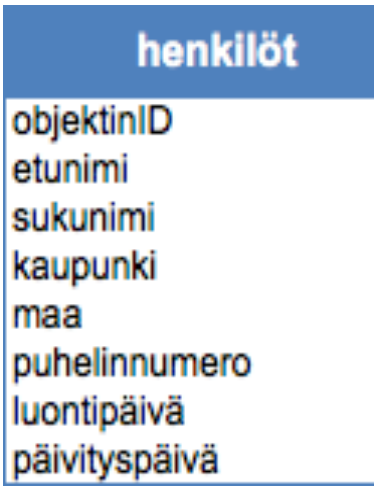
Sovelluksen palvelinratkaisuksi valittiin Parse. Parse on pilvipalvelu iOS-, Android-, JavaScript-, Windows 8-, Windows Phone 8- ja OS X -käyttöjärjestelmille. Parsen avulla sovellukselle saa helposti skaalattavan palvelinratkaisun, eikä itse tarvitse huolehtia palvelimen toiminnasta. Parse tarjoaa sovellukselle palvelut, kuten datan tallennuksen ja sosiaalisen median integroinnin. Myös oman sovelluskoodin ajaminen palvelimella on mahdollista. Se tarjoaa myös tehokkaan datan hallinnan web-käyttöliittymällä (kuvio 11). Sen avulla voi tallentaa helposti perusdataa, kuten kuvia ja sijainteja. Hakutoiminnossa on valmiit filtterit, millä saa halutun tuloksen. Ilmaiseksi palvelusta saa miljoona yhteyspyyntöä kuukaudessa, mikä riittää helposti osoitekirjasovellukselle. (Parse, 2013.)



Kuvio 15. Osoitekirja-sovelluksen tietokannan web-hallintatyökalu

6.4 Tietokanta

Sovelluksen tietokanta sijaitsee Parse-pilvipalvelun palvelimella. Tietokantaan tallennetaan osoitekirjan sisältö. Sovelluksen tietokannassa on yksi taulu (kuvio 12). Tietokannasta ladataan tietosisältö sovellukseen. ObjectId yksilöi tietokannassa olevan sisällön. Sen avulla tunnistaa kaksi samannimistä henkilöä toisistaan. Luontipäivä kertoo milloin kontaktitieto on luotu. Päivityspäivä viimeisen päivityksen.



Kuvio 16. Relaatiokaavio

6.5 Sovelluksen toteutus

Sovellus on toteutettu käyttämällä hyviä ohjelmointitapoja. Se on helposti laajennettavissa eri alustoille ja palvelimen hallinta on ulkoistettu Parselle, joten palvelimen valvontaan ja hallintaan ei mene resursseja läheskään yhtä paljon verrattuna siihen, jos olisin toteuttanut palvelimen ylläpidon itse. Käyttöliittymästä on tehty myös mahdollisimman helppokäyttöinen ja yksinkertainen (kuvio 12). Tässä kappaleessa kerrotaan sovelluksen ohjelmointivaiheista koodiesimerkein.

6.5.1 Yhteystiedon poistaminen

Aluksi yhteystiedot haetaan tietokannasta käyttämällä PFQuery:ä. PFQuery hakee person-nimisestä kannasta kaiken tietosisällön.

```
if (editingStyle == UITableViewCellEditingStyleDelete) {

    PFQuery *query = [PFQuery queryWithClassName:@"person"];
```

Seuraavaksi käyttäjän valitseman yhteystiedon tiedot tallennetaan NSMutableDictionaryyn. Valitun henkilön tiedot haetaan tietokannasta personId:n avulla.

```
NSMutableDictionary *personsq  = [_objects objectAtIndex:indexPath.row];

PFObject *person = [query getObjectWithId:[personsq objectForKey:@"personId" ]];
```

Lopuksi valitun henkilön tiedot poistetaan palvelimelta ja sovelluksesta ilman, että sovelluksen toiminta lakkaa vaikka yhteys olisi verkkoon katkennut.

```
[person deleteInBackground];
[_objects removeObjectAtIndex:indexPath.row];

    [tableView deleteRowsAtIndexPaths:@[indexPath] withRowAnimation:UITableViewRowAnimationFade];
```

6.5.2 Yhteystiedon muokkaaminen

Aluksi henkilön tiedot haetaan objectID:n avulla tietokannasta. ObjectId saadaan käyttötapauksesta, missä käyttäjä tarkastelee yhteystietoa.

```
PFQuery *query = [PFQuery queryWithClassName:@"person"];
PFObject *person = [query getObjectWithId:[self.objectID description]];
```

Yhteystiedon päivitetyt tiedot haetaan tekstikentistä käyttäjän muokattua niitä.

Tallennus tapahtuu, kun käyttäjä painaa Tallenna-nappia. Tallennus tapahtuu taustalla saveInBackround avulla.

```
[person setObject:self.firstName.text forKey:@"firstName"];
[person setObject:self.lastName.text forKey:@"lastName"];
[person setObject:self.city.text forKey:@"city"];
```

6.5.3 Testaus

Sovellus testattiin aina jokaisen valmistuneen osan jälkeen iOS Simulaattorilla. Sovellusta testattiin niin, että virheitä ei valmistuneesta osasta löytynyt ja sovelluksen vasteaika oli alle sekunti kaikissa käyttötapauksissa. Fyysisellä laitteella sovellusta ei päästy testaamaan, koska Apple vaati Developer-lisenssin, joka maksaa 99 euroa vuodessa. Simulaattoritestausta voi pitää kuitenkin opinnäytetyön tavoitteisiin nähden riittävän hyvänä testaustapana. Testauksen aikana varmistettiin, että sovellus toimii käyttötapausten mukaisesti.

7 Pohdinta

Opinnäytetyö rajattiin käsittelemään yhteyksiä iPhonesta yhteyksiä palvelimelle. Projektia ennen olin tutustunut iOS-sovellusohjelmointiin oma-aloitteisesti. Aiemmasta C- ja C++-ohjelmointikielien osaamisesta oli paljon hyötyä Objektive-C-ohjelmointi kielen oppimisessa.

7.1 Tulokset

Projektin aluksi selvitin taustoja, kuten iOS-järjestelmää, Xcode-kehitysympäristöä, yleisesti iOS:lle tarjolla olevia palvelinratkaisuja, mobiilipalvelinyhteyksien perusperiaatteita ja iOS-verkko-ohjelmoinnin perusratkaisuja. Taustojen selvittämisen jälkeen suunnittelin ja toteutin Osoitekirja-sovelluksen. Hyvin tehtyjen taustatöiden jälkeen minun oli helppo lähteä toteutusvaiheeseen. Sovelluksen ohjelmointi vei projektissa kuitenkin eniten aikaa. Sovellukseen toteutin ohjelmistovaatimuksissa vaaditut toiminnallisuudet.

7.2 Jatkokehitys

Projektin aikana toteutettu sovellus toimii hyvänä lähtökohtana kehitystyölle, missä sovellukseen toteutetaan kehittyneempi käyttöliittymä. Käyttöliittymä on tällä hetkellä todella helppokäyttöinen, mutta siitä puuttuu hienoudet, joita kaupallisilta sovelluksilta vaaditaan. Kun olen saanut käyttöliittymän vaadittavalle tasolle, sovellus on mahdollista viedä App Storeen ladattavaksi. Tällä hetkellä taustalla oleva palvelinratkaisu kestää isonkin käyttäjämäärän. Yksi iso jatkokehityksen mahdollisuus on sovelluksen luonti toisille käyttöjärjestelmille, kuten Androidille ja Windows Phone 8:lle.

LÄHTEET

Apple. 2012a. About the iOS Technologies. [WWW-dokumentti]. Apple. [Viitattu 2.3.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/Introduction/Introduction.html#//apple_ref/doc/uid/TP40007898

Apple. 2010b. Learning Objective-C: A Primer. [www-dokumentti]. Apple. [Viitattu 15.3.2013].Saatavana: https://developer.apple.com/library/mac/#referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/

Apple. 2012c. iOS Developer Tools. [www-dokumentti]. Apple. [Viitattu 21.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/Miscellaneous/Conceptual/iPhoneOSTechOverview/iPhoneOSDeveloperTools/iPhoneOSDeveloperTools.html#//apple_ref/doc/uid/TP40007898-CH7-SW1

Apple. 2013d. About iOS Simulator. [WWW-dokumentti]. Apple. [Viitattu 20.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html

Apple. 2011e. Networking & Internet Starting Point. [WWW-dokumentti]. Apple. [Viitattu 21.4.2013] Saatavana: https://developer.apple.com/library/ios/#referencelibrary/GettingStarted/GS_Networking_iPhone/index.html#//apple_ref/doc/uid/TP40007301

Apple. 2013f. Designing for Real-World Networks. [WWW-dokumentti]. Apple. [Viitattu 24.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/WhyNetworkingIsHard/WhyNetworkingIsHard.html%23//apple_ref/doc/uid/TP40010220-CH13-SW1

Apple. 2013g. Using Power And Bandwidth Efficiently. [WWW-dokumentti]. Apple. [Viitattu 22.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/WhyNetworkingIsHard/WhyNetworkingIsHard.html%23//apple_ref/doc/uid/TP40010220-CH13-SW1

Apple. 2013h. Design for Variable Network Speed. [WWW-dokumentti]. Apple. [Viitattu 22.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/WhyNetworkingIsHard/WhyNetworkingIsHard.html%23//apple_ref/doc/uid/TP40010220-CH13-SW1

Apple. 2013i. Design for High Latency. [WWW-dokumentti]. Apple. [Viitattu 22.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/WhyNetworkingIsHard/WhyNetworkingIsHard.html%23//apple_ref/doc/uid/TP40010220-CH13-SW1

Apple. 2013j. Test Under Various Conditions. [WWW-dokumentti]. Apple. [Viitattu 22.4.2013]. Saatavana: https://developer.apple.com/library/ios/#documentation/NetworkingInternetWeb/Conceptual/NetworkingOverview/WhyNetworkingIsHard/WhyNetworkingIsHard.html%23//apple_ref/doc/uid/TP40010220-CH13-SW1

Conway, J. & Hillegass A. 2011. iOS Programming: The Big Nerd Ranch Guide. Indianapolis, IN, USA: Pearson Technology Group.

Cox, J. , Jones, N. , Szumski, J & Tang, J. 2012. Professional iOS Network Programming: Connecting the Enterprise to the iPhone® and iPad®. Indianapolis, IN, USA: John Wiley & Sons, Inc, 4

Goldstein, N. 2012. iOS Cloud Development For Dummies. Hoboken, NJ, USA: John Wiley & Sons.

Parse. 2013. Overview. [Verkkosivu]. Parse. [Viitattu 24.4.2013]. Saatavana: https://www.parse.com/about

LIITTEET