SAVONIA

OPINNÄYTETYÖ - **AMMATTIKORKEAKOULUTUTKINTO**
TEKNIIKAN JA LIIKENTEEN ALA

FYSIAPP IOS-ALUSTALLE

TEKIJÄ: Maria Seppänen

SAVONIA-AMMATTIKORKEAKOULU

OPINNÄYTETYÖ
Tiivistelmä

Koulutusala Tekniikan ja liikenteen ala	
Koulutusohjelma Tietotekniikan koulutusohjelma	
Työn tekijä(t) Maria Seppänen	
Työn nimi FysiApp iOS-alustalle	
Päiväys 13.9.2015	Sivumäärä/Liitteet 34
Ohjaaja(t) Lehtori Sami Lahti, lehtori Jussi Koistinen	
Toimeksiantaja/Yhteistyökumppani(t) PhysioBit Oy	

Tiivistelmä

Opinnäytetyö tehtiin PhysioBit Oy:lle ja aiheena oli luoda FysiApp-sovellus iOS-alustalle. Tarkoituksena oli saada aikaan sovellus, jonka avulla asiakas voisi suorittaa terveysalan ammattilaisen, fysioterapeutin tai personal trainerin suunnittelemia harjoitusohjelmia. Lisäksi sovelluksen avulla asiakas ja ammattilainen voivat viestitellä keskenään.

Opinnäytetyön teko aloitettiin ensin tutustumalla iOS maailmaan ja Swift-ohjelmointikieleen. Ensimmäisenä sovellukseen tehtiin kirjautuminen ja luotiin tietokantayhteys Microsoft Azuren Mobile Serviceen. Sovelluksen käyttöliittymä suunniteltiin ja toteutettiin yhdessä PhysioBit Oy:n muotoilijoiden kanssa.

Lopputuloksena oli toimiva ja selkeä sovellus, jolla voi suorittaa ammattilaisen tekemiä harjoitusohjelmia, lähettää ja vastaanottaa viestejä ammattilaisilta sekä ryhmiltä, lukea ilmoituksia ja vastata mittauspyyntöihin. Valmis sovellus on ladattavissa Applen App Storesta.

Avainsanat
FysiApp, iOS, Swift

SAVONIA UNIVERSITY OF APPLIED SCIENCES

THESIS
Abstract

Field of Study Technology, Communication and Transport			
Degree Programme Degree Programme in Information Technology			
Author(s) Maria Seppänen			
Title of Thesis FysiApp for iOS platform			
Date	13 September 2015	Pages/Appendices	34
Supervisor(s) Mr. Sami Lahti, Lecturer & Mr. Jussi Koistinen, Lecturer			
Client Organisation /Partners PhysioBit Ltd.			

Abstract

The purpose of this thesis was to create FysiApp for the iOS platform. The goal of this project was to make a mobile application where the client can complete training programs made by healthcare professionals such as physiotherapists and personal trainers. In addition to that, the client can communicate with professionals with the application.

The project was started by getting to know the iOS world and the Swift programming language. The first features in the application were the login and database connection to Microsoft Azure Mobile Service. The user interface of the application was planned and implemented in co-operation with PhysioBit Ltd.'s designers.

As a result of this thesis a user friendly and working mobile application was made where the client can complete training programs, send and receive messages from professionals as well as from groups, read announcements and respond to measurement requests. The application can be found from Apple's App Store.

Keywords
FysiApp, iOS, Swift

ESIPUHE

Haluan kiittää koko PhysioBit Oy:n tiimiä työn aiheesta sekä saamastani suuresta avusta ja tuesta koko opinnäytetyön aikana. Lisäksi haluan kiittää Savonia-ammattikorkeakoulun lehtoreita Sami Lahtea ja Jussi Koistista opinnäytetyön ohjauksesta.

Kuopiossa 30.8.2015
Maria Seppänen

SISÄLTÖ

TERMIT JA LYHENTEET

API	Application programming interface. Ohjelmointirajapinta, jonka avulla ohjelmat lähettävät pyyntöjä ja vaihtavat dataa keskenään.
AppDelegate	Xcode projektin protokolla, joka vastaa tärkeistä tapahtumista. Se myös huolehtii siitä, että sovellus toimii oikein järjestelmän ja muiden sovellusten kanssa.
Framework	Paketti, joka sisältää jaettuja resursseja, kirjastoja ja dokumentaatioita. Luotu helpottamaan ohjelmiston kehitystä.
MVC	Model-View-Controller (malli-näkymä-käsittelijä) ohjelmistoarkkitehtuurityyli.
Objective-C	Oliolaajennus C-ohjelmointikieleen.
OS X	Mac tietokoneiden käyttöjärjestelmä.
SDK	Software development kit. Ohjelmiston kehitykseen tarkoitettu työkalu.

1 JOHDANTO

Opinnäytetyön toimeksiantajana toimi PhysioBit Oy. Sen pilottituotteensa FysiApp on suunnattu terveys- ja hyvinvointialan ammattilaisille. FysiAppin avulla ammattilaiset voivat lähettää fysioterapiaan ja kuntoutukseen suunniteltuja harjoitusohjelmia asiakkaan mobiilisovellukseen paperisten ohjeiden sijaan. FysiApp-mobiilisovellus on jo Android- ja Windows Phone -alustoilla ja kysynnän vuoksi yrityksellä oli tarve saada kyseinen sovellus myös iOS-alustalle.

Työn aihe on siis tehdä FysiApp-sovellus iOS-alustalle jo olemassa olevien Android- ja Windows Phone -sovellusten pohjalta. Sovelluksessa tulee pystyä suorittamaan harjoitusohjelmia sekä lähettää viestejä ammattilaiselle.

Tässä raportissa kerrotaan enemmän toimeksiantajasta sekä FysiAppista. Lisäksi käydään läpi työn luomisessa käytettyä tekniikkaa sekä sovelluksen toiminnot ja näkymät.

2 FYSIAPP JA PHYSIOBIT OY

Opinnäytetyön toimeksiantaja oli PhysioBit Oy. Kyseinen hyvinvointiteknologian yritys on perustettu lokakuussa 2014 ja se toimii Kuopiossa. Yrityksen pilottituotetta, FysiApp:a, lähdettiin suunnittelemaan jo kesällä 2013. FysiApp-tuotteeseen kuuluu maksullinen sovellus Windows- ja Mac-tietokoneille sekä ilmaiset mobiilisovellukset eri alustoille. FysiApp on ollut pilottitestauksessa keväällä ja kesällä 2015 ja se tulee markkinoille syksyllä 2015.

FysiApp on suunniteltu terveysalan ammattilaisille, mm. fysioterapian ja kuntoutuksen helpottamiseksi niin ammattilaiselle kuin asiakkaalle. Ajatus FysiAppista on lähtenyt fysioterapeuteilta, jotka halusivat vaikuttaa asiakkaan omatoimiseen harjoitteluun, jotta kuntoutus ja tavoitteisiin pääsy onnistuisivat.

FysiApp-sovellus toimii ammattilaisten ja asiakkaiden henkilökohtaisten tapaamisten lisänä. Tarkoituksena olisi helpottaa harjoittelua ja lisäksi vanhat paperiset ohjeet vaihdetaan aina mukana kulkevaan mobiililaitteeseen. Sovelluksesta ammattilainen voi seurata asiakkaidensa harjoittelua ja kehitystä. Salatun viestiyhteyden ansiosta ammattilainen voi vastata asiakkaan kysymyksiin ja antaa lisäohjeita harjoitteluun. FysiAppin käyttö edellyttää henkilökohtaista tapaamista ammattilaisen kanssa. (PhysioBit Oy 2015.)

3 TEKNIIKKA JA TYÖKALUT

3.1 iOS

iOS on Applen kehittämä käyttöjärjestelmä, jota käytetään Applen omissa mobiililaitteissa, joita ovat iPhone, iPad, iPad Mini ja iPod Touch. iOS-käyttöjärjestelmä julkaistiin vuonna 2007 ja se tunnettiin aikaisemmin nimellä iPhone OS. (Ramnath ja Loffing 2014, 20.) iOS:sta on kehitetty Apple Watch -kellon käyttämä watchOS-käyttöjärjestelmä (Apple Inc. 2015b). Viimeisin versio iOS 8 julkaistiin syksyllä 2014 ja se on saatavilla iPhone 4s -puhelimelle ja sitä uudemmille laitteille (Apple Inc. 2015a). Seuraava versio, iOS 9, julkaistaan syksyllä 2015 (Apple Inc. 2015i).

Apple julkaisi vuonna 2014 Health-sovelluksen iOS 8 -versioon. Sovellus kerää käyttäjän terveys- ja fitness-dataa ja luo niiden perusteella erilaisia kuvaajia. Health on saatavilla vain iPhone- ja iPod Touch -laitteissa. (Apple Inc. 2015c.)

HealthKit-frameworkin avulla Health ja muut sovellukset voivat jakaa keskenään dataa (Apple Inc. 2015c). Health-sovelluksen data on salattua ja muut sovellukset eivät pääse siihen käsiksi, jos käyttäjä ei ole hyväksynyt sitä tai jos sovellus pyörii taustalla tai jos puhelin on lukittuna. Sovellus voi käyttää HealthKit-frameworkia vain, jos kyseinen sovellus on suunniteltu tuottamaan terveys- tai fitness-palveluita. (Apple Inc. 2015e.) HealthKitin ansiosta Health-sovelluksen tietoja voisi tulevaisuudessa käyttää myös FysiApp-sovelluksessa. Näin voitaisiin lähettää dataa ammattilaiselle esimerkiksi asiakkaan nukkumisesta, ravinnosta ja jopa arkiliikunnasta harjoittelun tai kuntoutuksen ohella.

Alun perin sovellus tuki sekä iOS 7 - ja iOS 8 -versioita, mutta myöhemmin päätettiin, että iOS 7 -käyttöjärjestelmää ei tueta. Tämä johtui yksinkertaisesti siitä, että iOS 7-järjestelmässä näkymästä toiseen siirtyminen tapahtui erilailla kuin iOS 8:ssa ja navigointipalkki katosi.

3.2 Xcode

Xcode on Applen tarjoama ilmainen ohjelmointiympäristö, jolla luodaan ohjelmia OS X - ja iOS-ympäristöille. Se tarjoaa kaiken, mitä kehittäjä tarvitsee luodakseen, testatakseen ja jakaakseen ohjelmaa. Syyskuussa 2014 ilmestyi Xcoden viimeisin versio Xcode 6. Se toi mukaan uutena ominaisuutena Swift-ohjelmointikielen käytön sekä ”Playground”-nimisen ominaisuuden. Xcodesta löytyy myös Fix-it-ominaisuus, joka kertoo, mikäli koodissa on virhe, ja osaa myös korjata sen yhdellä napin painalluksella (Apple Inc. 2015d). Lisäksi siihen tuli SDK:t OS X Mavericks ja Yosemite ja iOS 8.0:sta 8.3:een. (Knott 2014, 17.)

Playground-ominasuutta voisi kuvailla eräänlaisena hiekkalaatikkona. Sen avulla kehittäjä voi testata erilaisia koodin pätkiä, esimerkiksi kuvaajia tai jopa yksinkertaisia silmukoita, ilman muuta taustakoodia. Lisäksi Playground-tiedostoja on helppo jakaa muille, ilman että tarvitsee kopioida koodia itse lähdekoodista tai lähettää koko projektia. (Knott 2014, 375.)

Ohjelmien ulkoasujen kehitykseen käytetään Storyboardia. Uutta projektia luodessa voi valita tekeekö ohjelman iPhonelle, iPadille vai käyttääkö universaalia ulkoasua. Universaalia ulkoasua tehdessä Storyboardissa pystyy määrittämään miten elementit asettuvat niin pienellä kuin isollakin näytöllä. On jopa mahdollista määrittää että, mikäli tietty elementti näkyy vain iPhonessa, mutta ei iPadillä. Myös näkymästä toiseen siirtyminen on mahdollista tehdä Storyboardilla ilman koodia.

Opinnäytetyö tehtiin käyttäen Xcoden versioita 6.3.2 ja se vaihdettiin myöhemmin päivitettyyn versioon 6.4.

3.3 Swift

Swift on Applen kehittämä ja vuonna 2014 julkaisema ohjelmointikieli, jota käytetään iOS- ja OS X -käyttöjärjestelmien kehityksessä. Swift on ottanut käyttöönsä C- ja Objective-C-ohjelmointikielten parhaimmat ominaisuudet, mutta tehnyt syntaksista helpompaa. Swift on suunniteltu toimimaan Applen Cocoa ohjelmointirajapinnan, Cocoa Touch -frameworkin sekä C:n ja Objective-C kanssa. (Apple Inc. 2014, 2.)

Objective-C:n yhteensopivuuden ansiosta yhdessä projektissa on mahdollista käyttää molemmilla kielillä kirjoitettuja tiedostoja (Apple Inc, 2015f). Tämän ansiosta opinnäytetyössä voitiin käyttää Windows Azure Mobile Services frameworkia, joka on kirjoitettu Objective-C-kielellä. Projektiin täytyi vain lisätä bridging header, joka mahdollistaa kahden ohjelmointikielen käytön samassa projektissa. Bridging header nimetään ”projektin_nimi-Bridging-Header.h”:ksi ja lisätään build-asetuksiin.

Opinnäytetyön ohjelmointikieleksi valittiin Swift, koska se on uusi. Se myös todettiin helpommaksi kuin Objective-C, varsinkin koska aikaisempaa kokemusta iOS- tai OS X -ohjelmoinnista ei ollut.

3.4 Microsoft Azure

Azure on Microsoftin pilvipalvelu, jota voidaan käyttää kehitys- ja virtuaalipalvelimien alustana. Lisäksi se tarjoaa käyttäjälleen mahdollisuuden rakentaa erilaisia tietokantoja, mobiili- ja web-applikaatioita. Azuren työkalut, valmiiksi rakennetut mallit (template) ja palvelut (managed services) mahdollistavat erilaisten applikaatioiden helpon luomisen ja hallitsemisen. (Microsoft 2015a.)

3.4.1 Mobile Services

Mobile Services on osa Azure App Serviceä, ja se tarjoaa laajasti skaalautuvan ja maailmanlaajuisesti tarjolla olevan mobiilisovellusten kehitysalustan. Sen avulla käyttäjä pystyy rakentamaan natiivisovelluksen Android, iOS tai Windows -alustalle tai cross-platform Xamarin- tai Cordova-sovelluksia. (Microsoft 2015b.)

Jotta sovelluksesta saadaan yhteys Mobile Serviceen, täytyy ladata Windows Azure Mobile Service Frameworks -paketti. Tämän jälkeen luodaan kuvan (KUVA 1) osoittamalla tavalla MSClient (Mobile

Service Client), jonka avulla tieto saadaan haettua tietokannasta. Client alustetaan Mobile Servicen osoitteella ja sovellusavaimella. Tässä sovelluksessa MSClient alustetaan AppDelegatessa.

```
var client:MSClient?

func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {

    self.client = MSClient(
        applicationURLString:"APPURL",
        applicationKey:"APPKEY"
    )

    return true
}
```

KUVA 1. Yhteys Microsoft Azuren Mobile Serviceen

Tietokannasta haettaessa käytetään MSTablea (Mobile Service Table). Sille kerrotaan, minkä nimisestä taulusta tietoa haetaan ja millä ehdoilla. Tämän jälkeen haun tuloksena tulleet tiedot lisätään ”allPrograms”-listaan NSDictionary-muodossa (KUVA 2). Tämän jälkeen voidaan ”allPrograms”-listan tiedot tallentaa objekteina Core Dataan.

Haku kannasta

```
UIApplication.sharedApplication().networkActivityIndicatorVisible = true
self.table!.readWithPredicate(predicate) {
    result, error  in

    UIApplication.sharedApplication().networkActivityIndicatorVisible = false
    if error != nil {
        println("Error: " + error.description)
        return
    } else {

        self.allPrograms = result.items as! [NSDictionary]

        //SAVE PROGRAM INFO
        for item in self.allPrograms {

            let p = item as NSDictionary
```

Objektin tallennus Core Dataan

```
println("ADD NEW PROGRAM")

let newProg = NSEntityDescription.insertNewObjectForEntityForName("Programs",
            inManagedObjectContext: self.managedObjectContext) as! Programs

newProg.id = (p.valueForKey("id") as? String)!
newProg.date = (p.valueForKey("date") as? NSDate)!
newProg.programid = (p.valueForKey("programid") as? String)!
var pid = (p.valueForKey("programid") as? String)!
```

KUVA 2. Tietokannasta haku ja objektien tallennus

3.4.2 Notification hubs

Azure Notification Hubs tarjoaa helppokäyttöisen ja monialustaisen push-infrastruktuurin. Sen avulla voi lähettää push-ilmoituksia mille tahansa mobiilialustalle. Push-ilmoitukset toimitetaan alustakohtaisten infrastruktuuriin, Platform Notification Systems (PNS), josta se lähetetään laitteeseen. Jos lähetetään ilmoitus iOS-laitteeseen, otetaan yhteys Apple Push Notification Service:en (APNS), joka lähettää ilmoituksen. (Microsoft 2015c.)

Jotta sovellus voi ottaa vastaan push-ilmoituksia, sille täytyy luoda App ID Applen omilla developer-sivuilla. Kyseiselle App ID:lle valitaan App Services-palveluista ”Push Notifications”. Tämän jälkeen luodaan samalle App ID:lle ”Provisioning Profile”, jonka avulla sovellus voi käyttää APNS:ää. Profiili tulee valita sen mukaan mitä tarvitsee, esimerkiksi testaukseen tarvitaan oma profiili ja App Storen

kautta jaettavaan sovellukseen oma. Lopuksi tulee luoda SSL-sertifikaatti. Sen avulla luodaan ”.p12”-loppuinen sertifikaatti, joka viedään Azuren Mobiili Service -portaaliin.

Sovelluksessa luodaan UserNotificationSettings, jotka rekisteröidään sovellukseen (KUVA 3). Azuren Notification hubiin rekisteröinti tapahtuu sovelluksen AppDelegatessa (KUVA 4).

```
var pushSettings = UIUserNotificationSettings(forTypes: UIUserNotificationType.Badge | UIUserNotificationType.Alert | UIUserNotificationType.Sound, categories: nil)
UIApplication.sharedApplication().registerUserNotificationSettings(pushSettings)
UIApplication.sharedApplication().registerForRemoteNotifications()
```

KUVA 3. UserNotificationSettings

```
func application(application: UIApplication, didRegisterForRemoteNotificationsWithDeviceToken deviceToken: NSData) {

    client?.push.registerNativeWithDeviceToken(deviceToken, tags: nil, completion: { (error) -> Void in

        if error != nil {
            println("Error registering for notifications: \(error)")
        } else {
            println("Registered notifications for device")
        }
    })
}

func application(application: UIApplication, didFailToRegisterForRemoteNotificationsWithError error: NSError) {
    println("Failed to register for remote notifications: \(error)")
}
```

KUVA 4. Rekisteröinti notification hubiin

FysiApp-sovellukset ottavat vastaan push-ilmoituksia uusista harjoitusohjelmista, viesteistä, ilmoituksista ja mittauspyynnöistä, sekä valmentajapyynnöistä. iOS-sovelluksissa push-ilmoitukset vastaanotetaan AppDelegatessa (KUVA 5). Alla olevassa kuvassa on esimerkkinä uuden viestin vastaanotto. Tulevan push-ilmoituksen perusteella ladataan API:n kautta uudet viestit.

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo: [NSObject : AnyObject]) {

    println(userInfo)

    if let apsPayload = userInfo["message"] as? NSDictionary {

        if let context = self.managedObjectContext {
            let dataAccess = DataAccess(managedObjectContext: context)

            dataAccess.loadMessagesFromAPI(apsPayload["client"] as! String)
        }

        if self.window?.rootViewController as? UITabBarController != nil {

            var tabBarController = self.window?.rootViewController as! UITabBarController
            tabBarController.selectedIndex = 2

        }

    }

}
```

KUVA 5. Push-ilmoituksen vastaanotto uudesta viestistä

4 FYSIAPP IOS-ALUSTALLE

4.1 Core Data

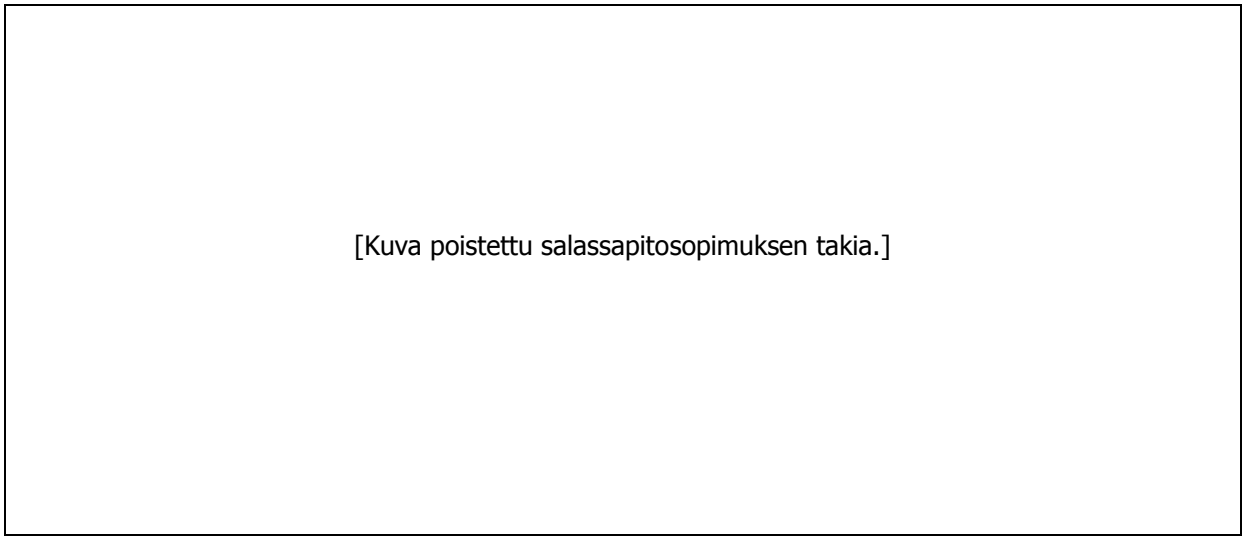
Jotta käyttäjän ei tarvitsisi odottaa joka kerta sovelluksen käynnistyksen jälkeen, että tietoja haettaisiin tietokannasta sovellukseen, täytyy ne tallentaa puhelimeen. Tallennukseen käytetään Core Dataa.

Core Data-framework hallinnoi ohjelman mallia (MVC-arkkitehtuurissa) ja sen objekteja. Näitä objekteja kutsutaan myös nimellä ”managed objects”, koska mallin objekteja hallitaan ”managed object context”:lla. Core Datan avulla voidaan ladata, tallentaa ja lukea objekteja.
(Ramnath ja Loffing 2014, 149)

Core Data tallentaa objektit SQLite-tietokanta tiedostoon. SQLite on relaatiotietokanta, mutta Core Data mahdollistaa sen käytön; tiedon haun, lisäykset, muokkaukset ja poistot, ilman SQL-lausekkeita. Core Datassa tietokannan taulut on kuvattu entiteetteinä eli objektikokonaisuuksina ja sarakkeita kutsutaan attribuuteiksi (Conway ja Hillegass 2012, 461, 464).

Tässä projektissa objektikokonaisuudet (KUVA 6) luotiin pitkälti jäljittelemällä Azuren tietokantaa, josta tiedot haetaan. WeekDays on ylimääräinen kokonaisuus, jota tarvitaan vain viikkonäkymän toiminnan kannalta. WeekDays-kokonaisuuden tarpeellisuutta on selvennetty enemmän luvussa 5.2 sivulla 18.

Normaalista poiketen Azuren tietokannassa ei ole luotu taulujen välille yhteyksiä, joten ne jätettiin myös pois näiden kokonaisuuksia väliltä. Lukuun ottamatta WeekDays, joka on yhdistetty Programs-kokonaisuuteen yhden suhde moneen (One-to-Many) yhteydellä. Myös Exercises on yhdistetty Programs kokonaisuuteen yhden suhde moneen yhteydellä. Näin Programs kokonaisuus sisältää NSSet listan kaikista liikkeistä, jotka kyseiseen harjoitusohjelmaan kuuluu. Lisäksi Updates-kokonaisuus pitää yllä päivämäärä- ja kellonaikatietoa siitä, milloin uudet ohjelmat tai viestit on haettu tietokannasta, sekä lisäksi tieto siitä milloin käyttäjä on viimeksi käyttänyt ohjelmaa.



KUVA 6. Tietokantaa jäljittelevät objektikokonaisuudet

Jotta luokka voi hakea dataa Core Datasta, tulee sille implemoida NSFetchedResultsControllerDelegate-protokolla. Dataa hakeva NSFetchedResultsController käyttää kyseisen protokollan metodeita tiedottaakseen delegaatille, mikäli tietoa on lisätty, poistettu, siirretty tai päivitetty. (Apple Inc. 2015g.)

NSFetchRequest-instanssi hakee dataa tietystä entiteetistä tiettyjen hakukriteerien ja jaottelujen mukaan. Haku suoritetaan ”managedObjectContext”:n executeFetchRequest-komennolla ja instanssi palauttaa joukon kriteereitä vastanneita objekteja. NSPredicate määrää hakukriteerit. Se vastaa SQL-lauseissa ”where”-ehtoa. NSSortDescriptor määrittelee, kuinka haun objektit jaotellaan. (Apple Inc. 2015h.)

4.2 Käyttöliittymä

Mobiilisovelluksen käyttöliittymä suunniteltiin ja toteutettiin yhdessä PhysioBit Oy:n muotoilijoiden kanssa. Ulkoasu on Android-sovelluksen kanssa samanlainen, mutta toteutettu iOS-tyyliohjeiden mukaisesti. Esimerkiksi Android-sovelluksen navigointipalkki on muutettu tähän sovellukseen alalaidassa olevaksi UITabBariksi. Sovelluksen yleinen ulkoasu ja ikonit muuttuivat useaan kertaan opinnäytetyön aikana.

iOS-applikaatiolle tyypillisiä elementtejä; UINavigationBar, UITabBar ja UITableView, on käytetty myös tässä projektissa. UINavigationBar huolehtii sivulta toisella siirtymisen. Esimerkiksi kun siirrytään viikko-näkymästä yhteen harjoitusohjelmaan, navigointipalkkiin ilmestyy ”< Back”-teksti, josta päästään takaisin viikko-näkymään. Eikä siihen tarvita minkäänlaista koodia. UITabBar on iOS-sovelluksille tyypillinen navigointipalkki, joka sijaitsee sovelluksissa alalaidassa. Tässä sovelluksessa UITabBar:ssa on seuraavaksi-, harjoitukset-, viestit- ja profiili-välilehdet.

Tässä sovelluksessa eniten käytetty elementti on UITableView. Se on eräänlainen lista, johon saa helposti sisältöä esimerkiksi Core Datasta. UITableView-elementit ovat helposti kustomoitavissa, kuten myös niiden solut. Elementin erilaista käyttöä voi nähdä esimerkiksi viikko-näkymässä ja viestien keskustelussa.

Usealla välilehdellä on mahdollista liikkua ”swipe”-toiminnolla näkymästä toiseen. Pyyhkäisytoiminto on tehty UIPageViewControllerilla. Toisin kuin UINavigationBarin ja UITabBarin kanssa, UIPageViewControllerin toimimiseen tarvitsee koodissa määritettynä sivut, joita pyöritetään näkymässä. Tämän lisäksi se tarvitsee UIPageControllerDelegaten metodeita, joilla se osaa näyttää kyseisessä indeksissä kyseisen sivun.

5 TOIMINNALLISUUDET JA NÄKYMÄT

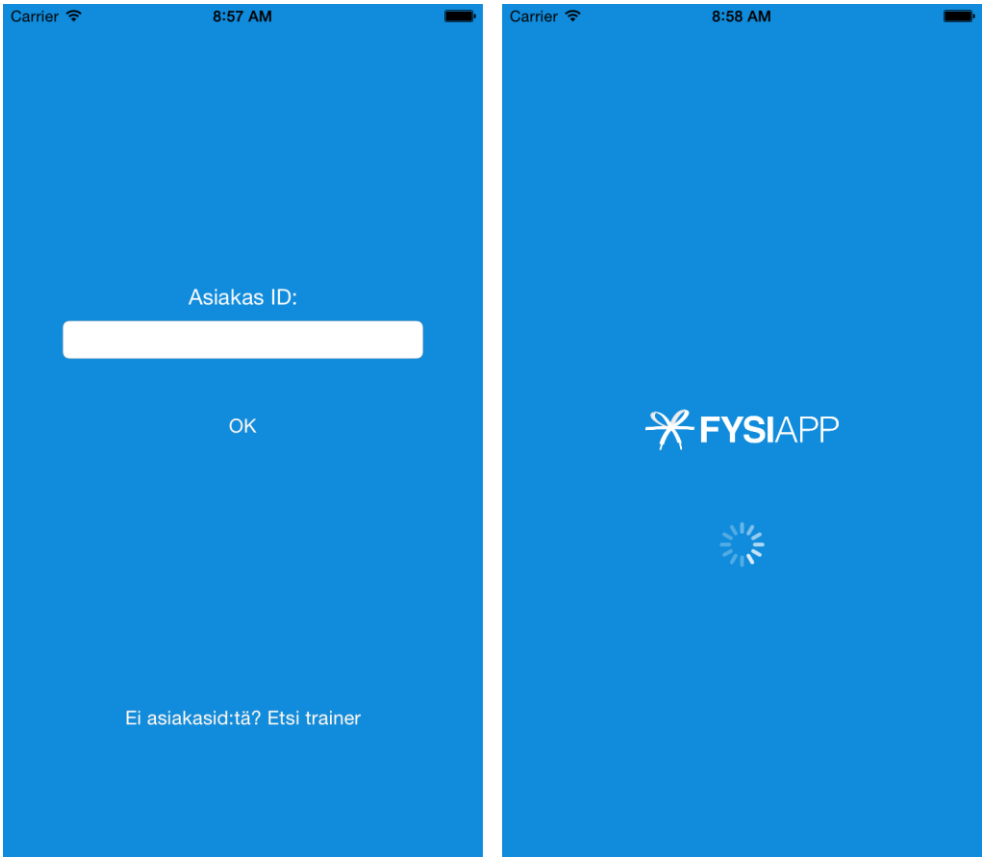
Android- ja Windows Phone -sovelluksissa on erittäin paljon toiminnallisuuksia. Toimintoja ovat harjoitusohjelmien näyttäminen ja suorittaminen, liikkeiden ohjeet kuvilla ja videoilla, viestit ammattilaisen ja ryhmien välillä, ilmoitukset ja mittauspyynnöt. Lisäksi löytyy käyttäjän ja ryhmän profiilit, profiilin ja sovelluksen asetukset sekä mittauksista muodostettavat kuvaajat.

Koska iOS-maailma ei ollut minulle niin tuttu, päätettiin, että iOS-sovellukseen ei tulisi olemaan niin paljoa ominaisuuksia opinnäytetyön kannalta. Tärkeimmiksi ominaisuuksiksi koettiin kirjautuminen, ohjelmat ja niiden suoritus sekä viestit.

Opinnäytetyön tekeminen sujui varsin mutkattomasti, joten vähimmäisvaatimusten lisäksi sovellukseen saatiin myös ilmoitukset, mittauspyynnöt, asiakkaan profiili ja asetuksiin profiilin muokkaus.

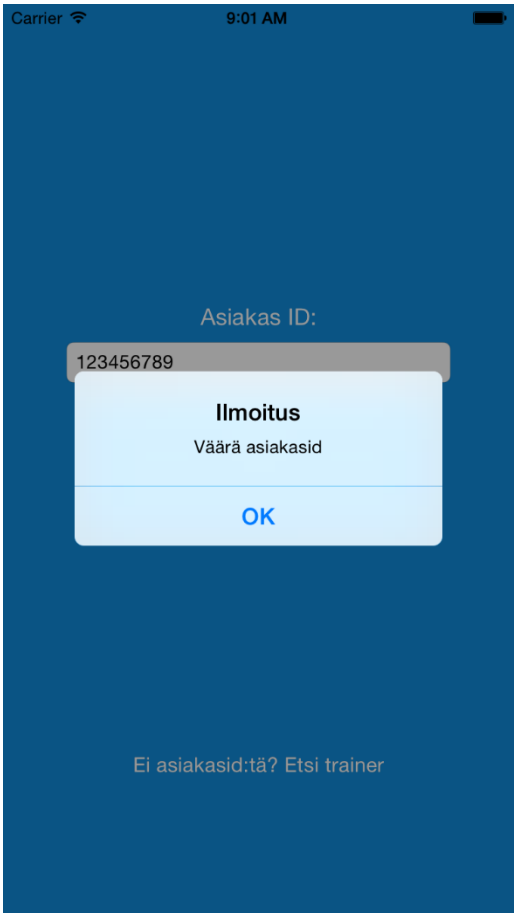
5.1 Kirjautuminen

Käyttääkseen FysiApp-mobiilisovellusta asiakas saa ammattilaiselta uniikin asiakas-id:n. Mobiilisovelluksen käyttöönotto alkaa rekisteröinnillä, jossa asiakas ensin kirjautuu sisään Google tunnuksilla ja sen jälkeen syöttää saamansa asiakas-id:n (KUVA 7).



KUVA 7. Asiakas-id:n syöttö- ja latausikkuna

Sovellukseen kirjaudutaan vain kerran, sovelluksen asennuksen jälkeen. Asiakas-id:n syötön jälkeen otetaan yhteys Azuren päähän rakennettuun clientLogin-APIin, joka tarkistaa, onko syötetty asiakas id olemassa. Mikäli asiakas id on oikea, jatketaan ohjelmaan, muutoin näytetään ilmoitus virheellisestä id:stä asiakkaalle (KUVA 8).



KUVA 8. Virheilmoitus väärästä asiakastunnuksesta

Ensimmäisen kirjautumisen yhteydessä tallennetaan Core Dataan kirjautuneen käyttäjän google id:n ja access tokenin. Näiden avulla alustetaan MSUser (Mobile Service User), joka asetetaan MSClient:n nykyiseksi käyttäjäksi.

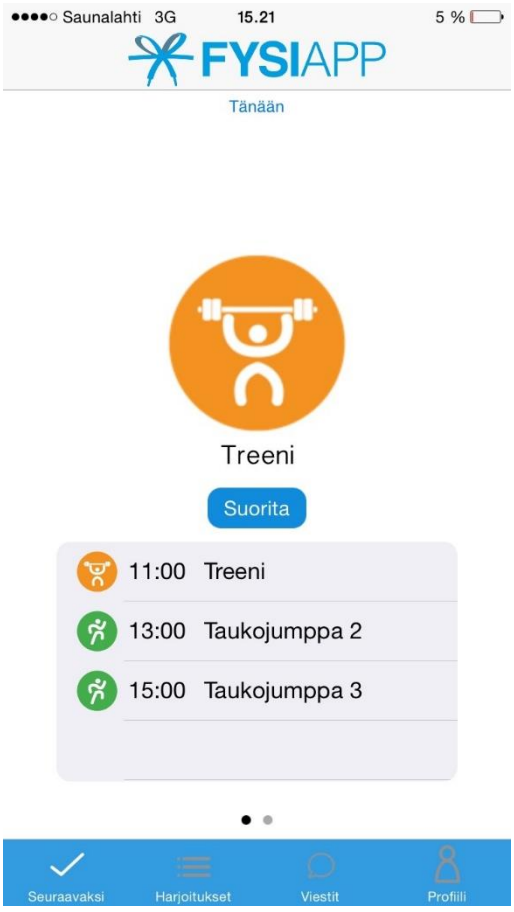
Kuvasta (KUVA 9) näkyy, kuinka ensin haetaan Authority-kokonaisuudesta, löytyykö asiakasta. Mikäli tiedot löytyvät, alustetaan MSUser ja asetetaan se käyttäjäksi. Mikäli mitään ei löydetä, jatketaan google-kirjautumiseen. Ilman MSCliet.currentUser-asetusta sovellus ei pysty hakemaan dataa Azuren tietokannasta.

```
//CHECK CLIENT DATA
let clientFetchRequest = NSFetchRequest(entityName: "Authority")

if let clientLogin = managedObjectContext?.executeFetchRequest(clientFetchRequest, error: nil) as? [Authority] {

    if clientLogin.count > 0 {
        println("client found from coredata")

        //SET CURRENTUSER
        var auth:NSManagedObject = clientLogin[0]

        let googleID = auth.valueForKey("identity") as! String
        let tokenID = auth.valueForKey("accesstoken") as! String
        let customerID = auth.valueForKey("customerid") as! String

        var msUser:MSUser = MSUser(userId: googleID)
        msUser.mobileServiceAuthenticationToken = tokenID

        self.delegate.client?.currentUser = msUser

        self.delegate.customerID = customerID

        //CHECK UPDATES FOR PROGRAMS, MESSAGES, ...
        if let context = self.managedObjectContext {
            let dataAccess = DataAccess(managedObjectContext: context)
            dataAccess.checkUpdates()
        }

    } else {
        println("no client > google login")
```

KUVA 9. MSUser:n alustus

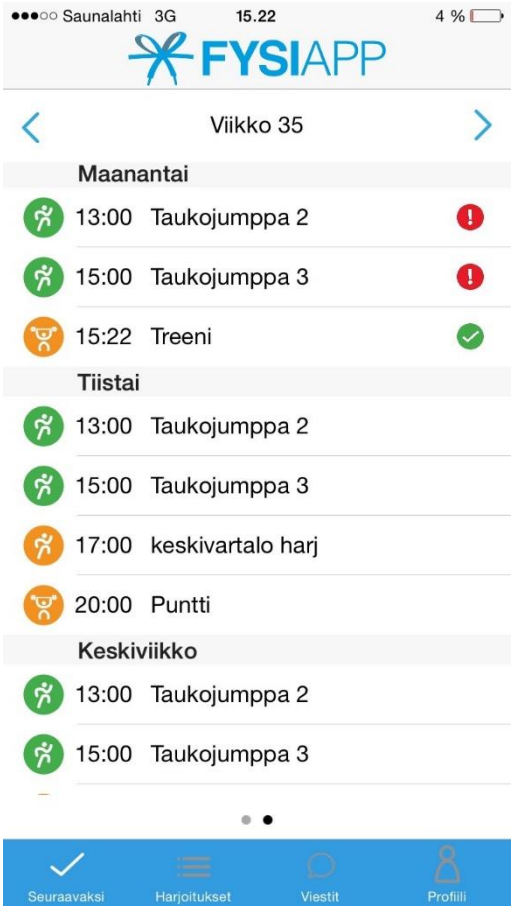
5.2 Seuraavaksi- ja viikko-näkymä

Sovelluksen aloitusnäkymänä on Seuraavaksi-näkymä (KUVA 10). Siitä asiakas näkee kunkin päivän mahdolliset harjoitukset ja ”Suorita”-napin kautta pääsee suoraan sen suoritukseen. Kyseisessä näkymässä on myös lyhyt lista, jossa on listattuna koko päivän harjoitukset, mikäli niitä on useampi kuin yksi.



KUVA 10. Seuraavaksi-näkymä

Seuraavaksi-näkymästä päästään Viikko-näkymään ”swipe”-toiminnolla eli pyyhkäisemällä näyttöä vasemmalle. Viikko-näkymässä oletuksena näkyvät kuluva viikko ja kaikki sen viikon harjoitukset päivien mukaan lajiteltuna (KUVA 11KUVA 1). Jokaisen suoritetun harjoituksen perässä on vihreä check. Mikäli harjoitus on jäänyt suorittamatta, perässä on punainen huutomerkki.



KUVA 11. Viikko-näkymä

Jotta viikkolistan päivien jako onnistui, täytyi luoda WeekDays-kokonaisuus, jonka avulla saadaan annettua Programs-objektille viikonpäivän tieto ohjelman päivämäärän mukaan. Kun ohjelmia haetaan viikkolistaan NSFetchedResultsControllerilla, sille annetaan SectionKeyPathName-tietoon, että ohjelmat jaotellaan viikonpäivän mukaan.

Ongelmana oli, että NSFetchedResultsControllerille tekstit ovat samanarvoisia ja se listaa ne aakkosjärjestyksessä. Tämän takia WeekDays-kokonaisuuden objekteille annettiin myös numeroarvot: maanantai on 1, tiistai on 2 jne. Näin NSFetchedResultsController jakaa harjoitukset ryhmiin WeekDays numeroarvon mukaan. Kun tietoja haetaan listaan, sen omassa titleForHeaderInSection-metodissa asetetaan ryhmälle viikonpäivän nimi numeron mukaan.

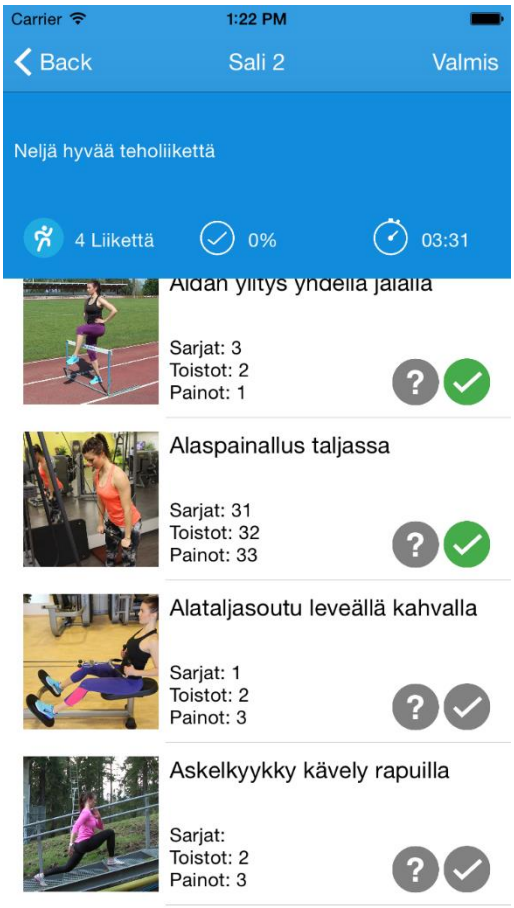
Android-sovelluksessa viikko-näkymän lisäksi on myös kuukausikalenteri. Koska Xcodesta ei löydy omaa kalenteri-elementtiä, jätettiin kalenteri pois iOS-sovelluksesta ja sen sijaan luotiin selattava viikko-näkymä. Näin käyttäjä voi selata tehtyjä ohjelmia viikko kerrallaan.

5.3 Harjoitusohjelmat

Ammattilainen luo harjoitusohjelmia asiakkaan tarpeiden mukaisesti ja asiakas pystyy suorittamaan niitä mobiililaitteestaan. Asiakas voi suorittaa annetun harjoitusohjelman aikaisintaan neljä päivää aikaisemmin suunniteltuun aikaan nähden tai neljä päivää jälkeenpäin. Mikäli harjoitusohjelma suoritetaan muulloin kuin suunniteltuun ajankohtaan, sen päivämäärä muuttuu suoritushetkeksi.

5.3.1 Harjoitusohjelman suoritus

Harjoitusohjelman suoritus -näkymässä on listattu kaikki ohjelmaan kuuluvat liikkeet UITableView-elementtiin (KUVA 12). Näkymässä ylimpänä on ohjelman kuvaus ja sen alla liikkeiden lukumäärä ja suoritusprosentti, joka kertoo, kuinka paljon harjoituksesta on suoritettu. Viimeisenä on reaaliajassa kulkeva aika.



KUVA 12. Suoritus-näkymä

Mikäli harjoitus on suoritettavissa, näkyy navigointipalkissa oikealla ”Aloita”-nappi. Sitä klikkaamalla kello käynnistyy. Kello on luotu käyttäen NSTimer-elementtiä. Ongelmana sen kanssa oli, että se pysähtyi, mikäli puhelimen näyttö lukittiin tai jos ohjelma laitettiin tausta-ajoon. Kun ohjelma palasi takaisin etualalle, kello jatkoi siitä mihin jäi. Samanlaista ongelmaa ei ollut Xcoden simulaattoreissa. Tämän takia oikealla laitteella harjoitusohjelman kesto saattoi jäädä lyhyeksi.

Projektin AppDelegate reagoi, mikäli sovellus menee tausta-ajoon tai palaa sieltä. Mutta tiedon saanti takaisin ohjelman suoritus-luokkaan oli monimutkaisempaa. Ongelma korjattiin käyttämällä NSNotificationCenteriä. Se on mekanismi, jonka avulla voidaan ilmoittaa erilaisista tapahtumista sovelluksen sisällä. Näin ohjelman suoritus-luokassa saatiin tieto, kun sovellus meni tausta-ajoon (KUVA 13).

Kun sovellus menee tausta-ajoon, kello pysäytetään. Kun sovellus palaa takaisin, lasketaan aika, kuinka kauan sovellus oli taustalla. Kyseinen aika lisätään jo kellossa olevaan aikaan ja käynnistetään se uudelleen. Näin harjoituksen suoritusaika on oikean mittainen.

```
NSNotificationCenter.defaultCenter().addObserver(self, selector: Selector("appWentBackground"), name: UIApplicationDidEnterBackgroundNotification, object: nil)
NSNotificationCenter.defaultCenter().addObserver(self, selector: Selector("appCameForeground"), name: UIApplicationWillEnterForegroundNotification, object: nil)
```

KUVA 13. NSNotificationCenter kertoo, kun sovellus menee tausta-ajoon tai palaa sieltä.

Jotta liikelistasta saatiin halutunlainen, täytyi luoda oma kustomoitu UITableViewCell tätä varten (KUVA 14). Solussa on liikkeen kuva, mikäli se on saatavilla, liikkeen nimi ja ammattilaisen antamat sarjat, toistot ja painot. Oikeassa alalaidassa on kaksi nappia, kysymys- ja check-merkki. Jälkimmäisellä voi pitää ylhäällä jo tehdyt liikkeet. Klikkaamalla nappia sen väri muuttuu vihreäksi, mutta sen voi muuttaa takaisiin harmaaksi uudelleen klikkaamalla. Näillä pidetään yllä suoritusprosenttia. Kysymysmerkistä aukeaa liikkeen ohjeet.

KUVA 14. Ohjelman suoritus -näkymän kustomoitu UITableViewCell

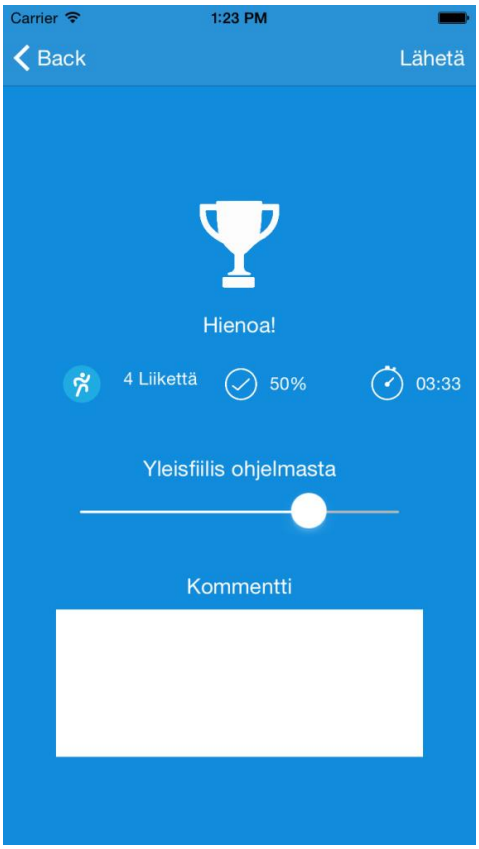
Check-merkin klikkauksen kanssa oli ongelmana, että yhtä klikatessa toisen ruudun check-merkki muuttui myös vihreäksi. Tämä johtuu siitä, että UITableView käyttää uudelleen listan soluja. Check-merkit ovat nappeja ja jokaisella laitettiin tag-tietoon rivin numero. Sitten luotiin bool-lista jonka tietoina oli tag-tiedon rivinumero sekä tieto ”true” tai ”false” sen mukaan oliko check-merkit vihreitä vai ei.

Check-merkin klikkauksesta kutsutaan ”setExerciseDone”-metodia (KUVA 15). Tässä metodissa asetetaan napille joko vihreä tai harmaa check-merkki vaihtamalla napin kuvaa sekä päivitetään bool-listan tietoa kyseiselle riville.

```
func setExerciseDone(sender: UIButton) {

    if startbtnClicked {

        if sender.currentImage == UIImage(named: "check") {

            let row = sender.tag
            selectedCheckButtons[row] = true
            var checkImage = UIImage(named: "check_done")
            sender.setImage(checkImage, forState: .Normal)

            selectedButtons++

        } else {

            let row = sender.tag
            selectedCheckButtons[row] = false
            var checkImage = UIImage(named: "check")
            sender.setImage(checkImage, forState: .Normal)

            selectedButtons--

        }
    }

}
```

KUVA 15. ”setExerciseDone”-metodi

Kun harjoitusohjelma on suoritettu, käyttäjä klikkaa suoritus-näkymän oikealla ylälaidassa olevaa ”Valmis”-nappia. Tästä aukeaa suorituksen lähetys -näkymä (KUVA 16), jossa asiakas näkee harjoitukseen kuluneen ajan ja sen kuinka paljon ohjelmasta suoritti. Lisäksi hän lähettää omat tuntemuksensa harjoituksesta. Asiakas voi myös lisätä vapaaehtoisen kommentin harjoituksesta. Kaikki nämä tiedot lähetetään ammattilaiselle.



KUVA 16. Suoritettu harjoitus

5.3.2 Liikkeiden ohjeet

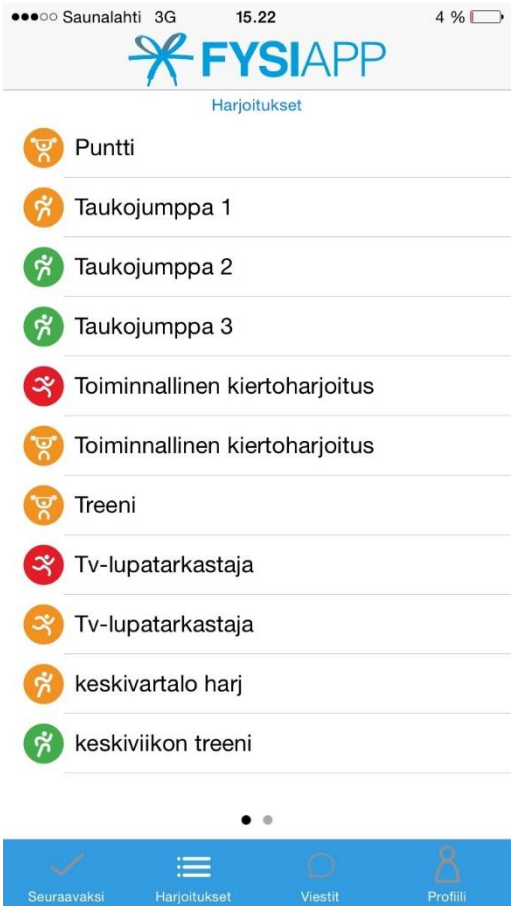
Toisin kuin muut näkymät, liikkeen ohje -näkymä (KUVA 17) on luotu kokonaan koodissa. Liikkeen ohjeet sisältävät liikkeen nimen ja sen jälkeen listattuna liikkeen vaikutukset. Mikäli liikkeellä on video-ohje, se näytetään ensin omien, lyhyiden tekstiohjeiden kanssa, jos ne on kirjoitettu. Liikkeellä on kuva- ja tekstiohjeet, jotka listataan allekkain näkymään. Ohjeiden jälkeen ovat liikkeeseen liittyvät huomiot. Liikkeiden kuvat tallennetaan Core Dataan, jolloin ohjeita voi lukea, vaikka internet ei olisi kytketty tai saatavilla.

Näkymä piti luoda kokonaan koodillisesti, koska ei ole varmaa onko liikkeellä video- ja/tai kuvaohjeet ja kuinka monta kuvaa liikkeellä on. Näkymä on rakennettu UIScrollView-elementin sisälle, joka mahdollistaa pitkän näkymän selauksen. Koodissa on pidettävä yllä, kuinka korkeita kaikki elementit ovat, jotta UIScrollView ei jätä näyttämättä sisältöä.

KUVA 17. Liikkeen ohje -näkymä

5.3.3 Kaikki harjoitusohjelmat ja liikepankki

Sovelluksen toisella välilehdellä on kaikki asiakkaalle tehdyt harjoitusohjelmat (KUVA 18). Tästä näkymästä asiakas voi suorittaa hänelle tehdyn harjoitusohjelman uudelleen. Kun ohjelma avataan, ”Aloita”-napin sijasta on ”Suorita”-nappi. Nämä harjoitusohjelmat ovat niin sanotusti omia ylimääräisiä ohjelmia, jotka lähetetään ammattilaiselle. Lisäksi tämä ohjelma tulee näkyviin viikko-näkymään.



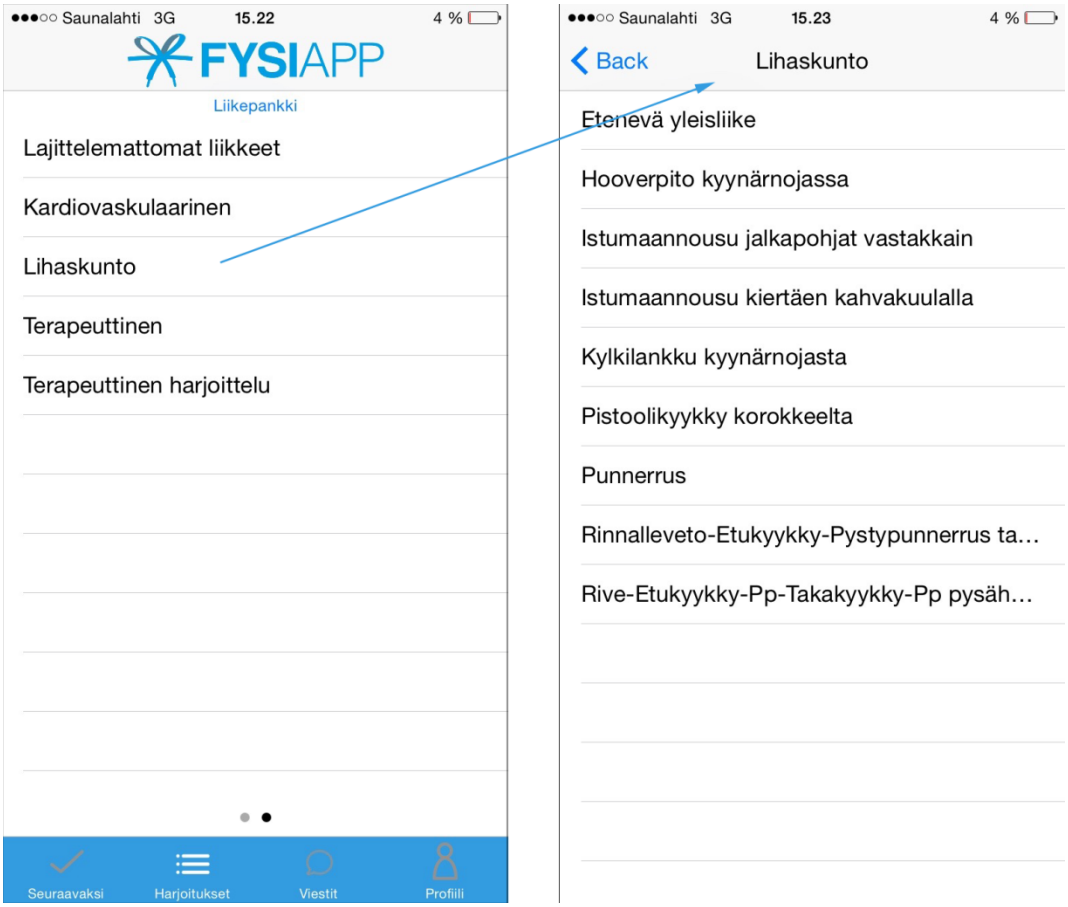
KUVA 18. Kaikki harjoitusohjelmat -näkymä

Jokaisella harjoitusohjelmalla on yksilöllinen id. Kun asiakas suorittaa ylimääräisen harjoitusohjelman, siitä tulee uusi ohjelma ja sille täytyy luoda oma uniikki id:nsä. Omalle ohjelmalle luotu id on muotoa ”asiakas_id” + ”OMA” + epoch (aika millisekunteina päivästä 1.1.1970 laskettuna) heksadesimaalimuodossa (KUVA 19).

```
println("uusi treeni")

var timestamp = UInt64(floor(NSDate().timeIntervalSince1970 * 1000))
var newProgID = self.delegate.customerID! + "OMA" + String(timestamp, radix: 16, uppercase: false) + String(customerCounter, radix: 16, uppercase: false)
customerCounter++
```

KUVA 19. Oman ohjelman id:n generointi

Kun näkymässä pyyhkäistään vasemmalle, aukeaa liikepankki-näkymä (KUVA 20). Täältä asiakas löytää kaikki harjoitusohjelmissa olleet liikkeet. Jokaisella liikkeellä on niin sanottu tunniste, joka kertoo, millainen liike on kyseessä. Tunniste voi olla esimerkiksi ”lihaskunto” tai ”terapeuttinen”. Ensimmäisenä on listattu nämä tunnisteet, joita klikkaamalla pääsee tarkistelemaan liikkeitä. Jokaisesta liikkeestä aukeaa ohjeet.

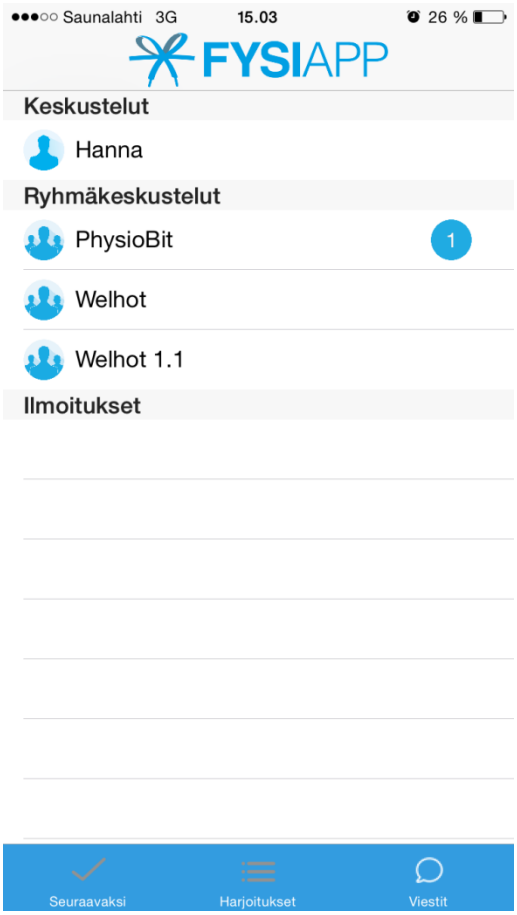


KUVA 20. Liikepankki-näkymä

5.4 Viestit

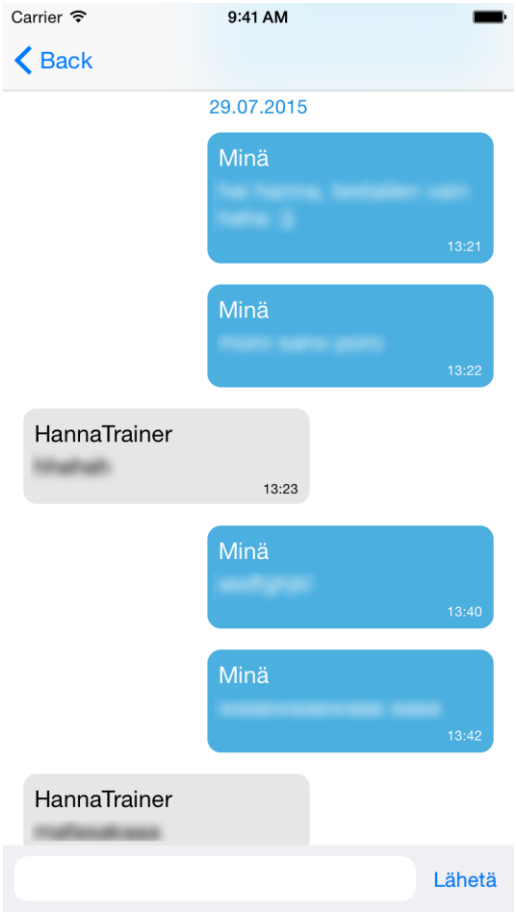
Viesti-näkymästä asiakas voi keskustella ammattilaisen kanssa. Näkymässä on myös ryhmäkeskustelut niiden ryhmien kanssa, joihin kyseinen asiakas kuuluu. Keskustelut on laitettu samaan UITableView-listaan, johon listataan myös mittauspyynnöt ja ilmoitukset (KUVA 21). Mikäli keskusteluissa on lukemattomia viestejä, keskustelun nimen perään ilmestyy sininen pallo, jonka sisällä oleva numero kertoo lukemattomien viestien määrän.

Viestit säilytetään tietokannassa salattuina, jottei niitä voisi lukea ketkään muut kuin asianomaiset. Tämän takia viestit haetaan sovellukseen API:n kautta, jota muuttaa viestit luettavaan muotoon. API:lle lähetetään parametrina tieto, kenen viestejä haetaan, tässä tapauksessa joko sovelluksen käyttäjän asiakas id:llä tai ryhmän id:llä.



KUVA 21. Viesti-näkymä

Keskustelua klikkaamalla aukeaa itse keskustelu (KUVA 22). Myös keskustelu-näkymän toteutukseen käytettiin UITableView-elementtiä. Kyseistä näkymää varten luotiin kaksi erilaista UITableViewCell-elementtiä, joista toista käytetään omiin viesteihin ja toista muiden.



KUVA 22. Keskustelu-näkymä

Keskustelu-näkymään haluttiin myös päivämääräjakaja. Aluksi se vaikutti hiukan mahdottomalta tehtävältä, koska viikko-näkymässä päiviin jako oli tuottanut ongelmia. Mutta pienen tutkimisen jälkeen, saatiin selville, että viestien päivämääränjakajan luominen olikin erittäin yksinkertaista. Jakajan luomiseen käytettiin metodia, joka palauttaa viestin päivämäärän ilman aikaa (KUVA 23). Ja tätä kyseistä funktiota käytettiin NSFetchedResultsControllerin SectionKeyPathName-tietona. Päivämääräjakajalle tehtiin kustomoitu ulkoasu, jolloin siitä tuli läpinäkyvä, vaalean sinisellä tekstillä.

```
func returnDate() -> String {

    var dayFormat = NSDateFormatter()
    dayFormat.dateFormat = "dd.MM.yyyy"

    var ret = dayFormat.stringForObjectValue(self.ms_createdAt)

    if ret == nil {
        ret = "01.01.1970"
    }

    return ret!

}
```

KUVA 23. Viestin päivämäärän palauttava metodi

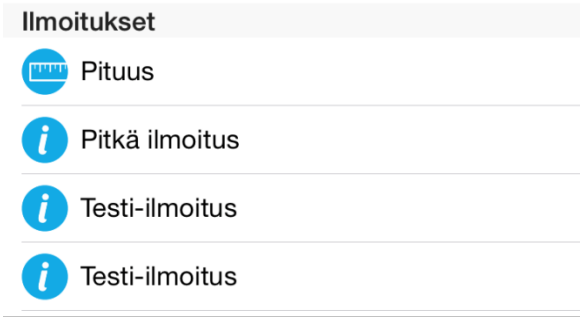
Viesti-näkymässä piti hakea tietoa neljästä entiteetistä samaan listaan. UITableView-elementti osaa ottaa tietoa vastaan vain yhdeltä NSFetchedResultsControllerilta. Ongelmana oli, kun ensimmäinen haku oli ohi, UITableView-elementti päivitti oman sisältönsä vaikka toinen haku oli vielä käynnissä ja

sovellus kaatui sen takia. Ongelma ratkaistiin käyttämällä NSMutableArray-listaa, johon pystyttiin lisäämään objekteja useammasta objekti-kokonaisuudesta.

5.5 Ilmoitukset ja mittauspyynnöt

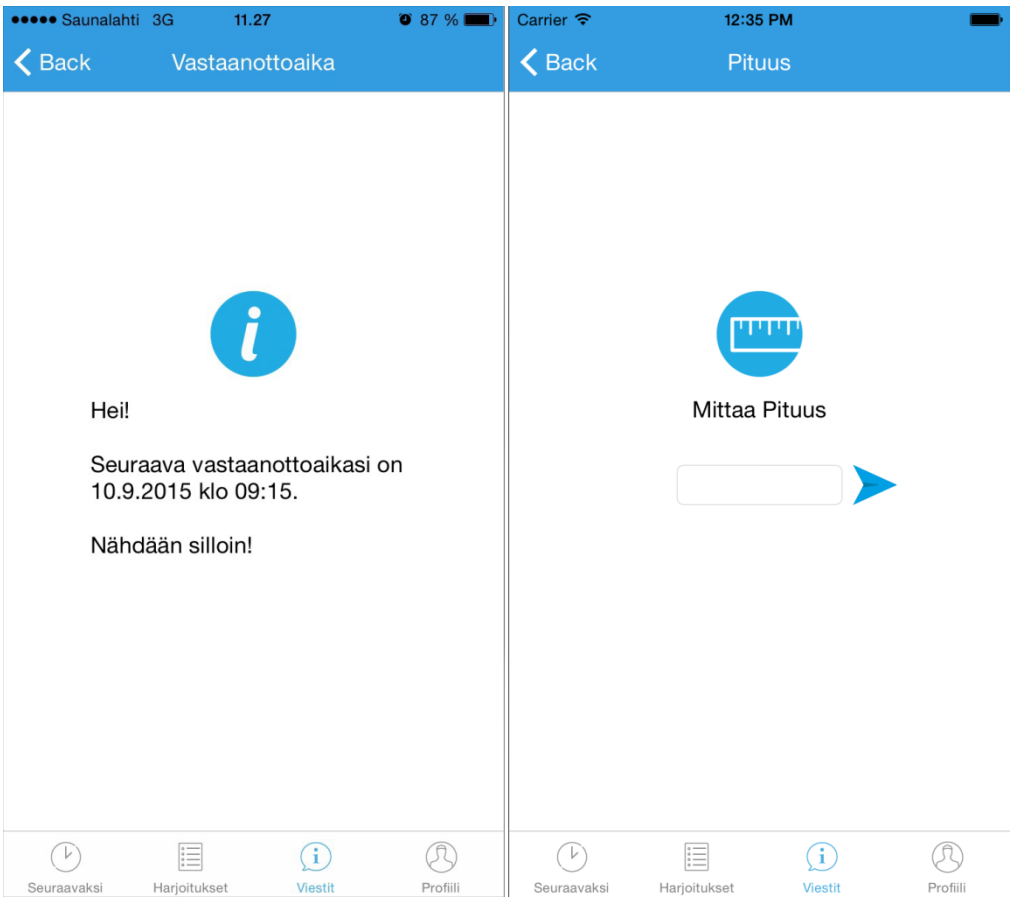
Ammattilainen voi lähettää asiakkaalle erilaisia ilmoituksia. Ilmoitukset voivat sisältää erilaisia informaatio tekstejä tai esimerkiksi kuntotestituloksia. Lisäksi ilmoitukseen voi olla lisättynä linkki. Ilmoitukset näkyvät viesti-näkymässä ilmoitus-osiossa (KUVA 24).

Samassa osiossa näkyy myös mahdolliset mittauspyynnöt. Ammattilainen voi lähettää asiakkaalle pyynnön jostain tietystä mittauksesta, esimerkiksi paino tai verenpaine. Mittaukset näkyvät ilmoitus-osiossa vain silloin kun niihin halutaan asiakkaan mittaus. Ja kun mittaus on lähetetty ammattilaiselle, kyseinen pyyntö katoaa listalta.



KUVA 24. Ilmoitukset

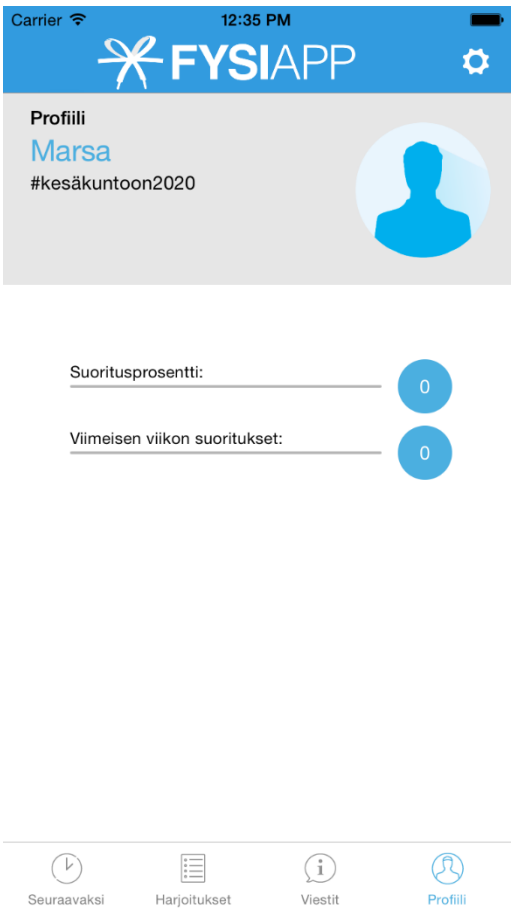
Ilmoitus- ja mittauspyyntö-näkymät käyttävät samaa UIViewControlleria, jonka sisältö muuttuu sen perusteella avataanko siihen ilmoitus vai mittauspyyntö (KUVA 25). Mittauspyynnön syöttökenttä on vapaamuotoinen eli asiakas voi syöttää pelkän numeron tai numeron ja yksikön.



KUVA 25. Ilmoitus- ja mittauspyyntö-näkymä

5.6 Profiili

Profiilistaan asiakas näkee nopeasti omat tietonsa sekä suoritusprosentin ja viikon suoritukset (KUVA 26). Profiilin oikeasta ylänurkasta pääsee asetuksiin. Käyttäjä voi muokata profiilikuvaansa klikkaamalla profiilissa olevaa kuvaa. Siitä aukeaa lista, josta käyttäjä voi valita, ottaako kuvan kamerasta vai mobiililaitteensa galleriasta (KUVA 27). Kuvan valinta on tehty UIAlertControllerilla, jolle on asetettu tyyliksi ActionSheet.



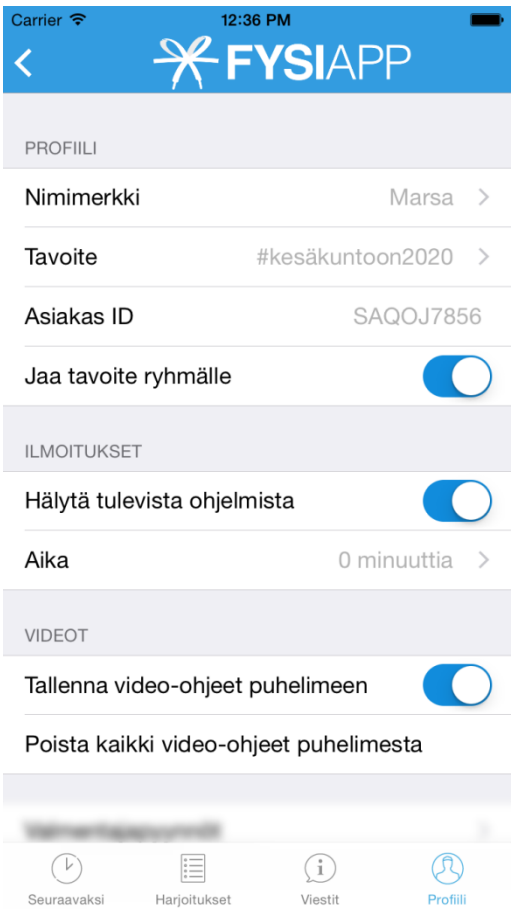
KUVA 26. Profiili-näkymä

KUVA 27. Profiilikuvan valinta

5.7 Asetukset

Sovelluksen asetuksiin pääsee profiili-näkymän ratas-kuvakkeen kautta. Asetuksista löytyy profiili-, ilmoitus- ja video-asetukset (KUVA 28).

Profiiliasetuksista käyttäjä voi muokata käyttäjänimeä ja tavoitetta sekä haluaako jakaa tavoitteensa muille ryhmäläisille. Ilmoitusasetuksista käyttäjä voi valita haluaako ilmoituksen tulevista harjoitus-ohjelmista, sekä kuinka monta minuuttia aikaisemmin haluaa ilmoituksen. Videoasetuksista käyttäjä voi määrittää, mikäli videot tallennetaan hänen puhelimeensa, kun harjoitusohjelmia tallennetaan. Jo tallennetut videot voi myös poistaa muistista.



KUVA 28. Asetukset-näkymä

5.8 Paikalliset ilmoitukset

Serverin päästä tulevien push-ilmoitusten lisäksi sovelluksessa on paikallisia ilmoituksia, Local Notifications. Jos käyttäjä on asetuksista valinnut haluavansa ilmoitukset tulevista harjoitusohjelmista, silloin ilmoitukset luodaan kaikille ohjelmille automaattisesti (KUVA 29).

Paikalliselle ilmoitukselle annetaan ”fireDate”-päivämäärä jolloin ilmoitus halutaan näyttää. Tässä sovelluksessa ilmoitusaika on ohjelman suoritusajankohta, josta on vähennetty minuutteja sen perusteella, kuinka paljon aikaisemmin asiakas ilmoituksen haluaa. Asetuksista käyttäjä voi valita ajaksi 0, 15, 30, 60 tai 90 minuuttia.

Lisäksi ilmoitukselle luodaan userInfo-tietoon, mihin ohjelmaan kyseinen ilmoitus liittyy. Näin voidaan tarkistaa, ettei samalle ohjelmalle ole useampaa ilmoitusta.

```
var localNotification:UILocalNotification = UILocalNotification()
localNotification.alertBody = "Sinulla on tänään harjoitus"
localNotification.fireDate = fireNotification
localNotification.soundName = UILocalNotificationDefaultSoundName
var info:NSDictionary = NSDictionary(dictionary: ["progid":progid])
localNotification.userInfo = info as [NSObject : AnyObject]
UIApplication.sharedApplication().scheduleLocalNotification(localNotification)
```

KUVA 29. Paikallisen ilmoituksen luonti

6 TESTAUS

Sovellusta testattiin koko kehitysvaiheen ajan Xcoden omilla simulaattoreilla. Xcode tarjoaa simulaattoreiksi puhelimista iPhone 4s ja sitä uudemmat puhelimet sekä iPad Air- ja iPad Retina-tabletit. Testauksessa käytettiin eniten iPhone 6:sta, mutta ulkoasun testauksessa tarvittiin jokaista. Lisäksi sovellusta testattiin aluksi iPhone 4-puhelimella. Mutta siitä luovuttiin, koska kyseisessä laitteessa oli iOS 7-käyttöjärjestelmä, jota sovellus ei enää tukenut. Tämän jälkeen testauksessa käytettiin PhysioBit Oy:n työntekijöiden iPhone 6-puhelimia.

Ensimmäinen käyttöönotto testaus aloitettiin heinäkuun puolessa välissä 2015. Tällöin testaajina toimi PhysioBit Oy:n työntekijät, joilla oli iOS-laitteita.

Toinen testaus aloitettiin, kun sovellus saatiin Applen App Storeen ladattavaksi. Tällöin testaajina toimi salibandyseura Welhojen miesten edustusjoukkue. SB Welhot ottivat FysiAppin testikäyttöön elokuussa 2015 ja joukkueesta löytyy myös iPhone-käyttäjiä.

7 YHTEENVETO

7.1 Yhteenveto

Opinnäytetyön tavoitteena oli luoda FysiApp-sovellus iOS-alustalle, jolla käyttäjä voisi suorittaa ammattilaisen luomia harjoitusohjelmia sekä viestitellä tämän kanssa. Muita ominaisuuksia ovat ilmoitukset, mittauspyynnöt, käyttäjän profiili, asetukset sekä push- ja paikalliset ilmoitukset. Työ toteutettiin käyttäen Applen Xcodea sekä Swift-ohjelmointikieltä. Aikaisempaa kokomusta niistä ei ollut, mutta niiden oppiminen oli erittäin helppoa.

Opinnäytetyötä tehtiin kesällä 2015, jonka aikana pidettiin pari kokousta opinnäytetyönohjaajien kanssa sekä pidettiin yhteyttä sähköpostitse kahden viikon välein ja kerrottiin missä mennään. Lisäksi yrityksen edustajien kanssa oli viikoittain lyhyt tilannepalaveri.

Työn tekeminen oli aluksi haastavaa, koska paljon aikaa meni uuden ohjelmointikielen opiskeluun. Myös tietokantayhteyden luonti tuotti hiukan ongelmia vähäisten dokumentaatioiden takia. Mutta loppua kohden sovellus eteni nopeaan tahtiin.

Tämän opinnäytetyön ansiosta opin kokonaan uuden ohjelmointikielen, josta on varmasti hyötyä tulevaisuudessa. Olen erittäin tyytyväinen saamaani lopputulokseen. En osaa eritellä mikä meni hyvin tai huonosti, sillä suunnitellut ominaisuudet joko onnistuivat ensimmäisellä yrityksellä ja ongelmatilanteisiin löytyi lopulta jokin toimiva ratkaisu, välillä jopa parempi kuin aikaisemmin suunniteltu. Tästä hyvä esimerkki on viesti-näkymä, jossa piti hakea neljästä entiteetistä tietoa yhteen listaan. Ratkaisu oli erittäin yksinkertainen, mutta sen löytämiseen meni paljon aikaa.

Jos tekisin jotain uudelleen, se olisi viikko-näkymä. Tällä tiedolla mitä nyt omaan, en käyttäisi WeekDays-entiteettiä ollenkaan. Silloin saisin varmasti aikaan viikko-kalenterin, jossa näkyvät myös päivät, joissa ei ole harjoituksia.

7.2 Jatkokehitys

FysiApp-sovellusta tullaan jatkokehittämään syksyllä 2015. Tulevia ominaisuuksia ovat mittaustulosten perusteella tehtävät kuvaajat sekä suoritettuihin harjoitusohjelmiin tulee näkyviin valmentajan antamat kommentin suorituksesta. Asetukset-näkymään tulee lisää ominaisuuksia, kuten valmentajapyynnöt, palautteen ja vikailmoitusten lähetys sekä synkronointi. Viikko-näkymän lisäksi kalenterinäkymä on suunnitteilla.

LÄHTEET

APPLE Inc. 2015a. What is iOS [verkkojulkaisu]. [Viitattu 2015-06-21.] Saatavissa: https://www.apple.com/ios/what-is/

APPLE Inc. 2015b. Watch [verkkojulkaisu]. [Viitattu 2015-06-21.] Saatavissa: https://www.apple.com/watch/

APPLE Inc. 2015c. Healt [verkkojulkaisu]. [Viitattu 2015-07-26.] Saatavissa: https://www.apple.com/ios/whats-new/health/

APPLE Inc. 2015d. Xcode IDE [verkkojulkaisu]. [Viitattu 2015-06-28.] Saatavissa: https://developer.apple.com/xcode/features/

APPLE Inc. 2015e. HealthKit Framework Reference [verkkojulkaisu]. [Viitattu 2015-07-26.] Saatavissa: https://developer.apple.com/library/ios/documentation/HealthKit/Reference/HealthKit_Framework/

APPLE Inc. 2015f. Swift and Objective-C in the Same Project [verkkojulkaisu]. [Viitattu 2015-06-21.] Saatavissa: https://developer.apple.com/library/mac/documentation/Swift/Conceptual/BuildingCocoaApps/MixandMatch.html#//apple_ref/doc/uid/TP40014216-CH10-XID_78

APPLE Inc. 2015g. NSFetchedResultsControllerDelegate Protocol Reference [verkkojulkaisu]. [Viitattu 2015-07-28.] Saatavissa: https://developer.apple.com/library/prerelease/ios/documentation/CoreData/Reference/NSFetchedResultsControllerDelegate_Protocol/index.html

APPLE Inc. 2015h. NSFetchRequest Class Reference [verkkojulkaisu]. [Viitattu 2015-07-39.] Saatavissa: https://developer.apple.com/library/prerelease/ios/documentation/Cocoa/Reference/CoreDataFramework/Classes/NSFetchRequest_Class/index.html#//apple_ref/occ/cl/NSFetchRequest

APPLE Inc. 2015i. iOS 9 Preview [verkkojulkaisu]. [Viitattu 2015-08-02.] Saatavissa: https://www.apple.com/ios/ios9-preview/

APPLE INC. 2014. The Swift Programming Language. Julkaisija: Apple Inc.

CONWAY, Joe ja HILLEGASS, Aaron 2012. iOS Programming: The Big Nerd Ranch Guide. 3. Painos. Atlanta: Big Nerd Ranch, Inc.

PHYSIOBIT OY 2015. FysiApp [verkkojulkaisu]. [Viitattu 2015-08-26.] Saatavissa: http://www.fysiapp.fi/

KNOTT, Matthew 2014. Beginning Xcode: Swift Edition. New York: Apress.

MICROSOFT 2015a. What is Microsoft Azure? [verkkojulkaisu]. [Viitattu 2015-06-25.] Saatavissa: http://azure.microsoft.com/en-us/overview/what-is-azure/

MICROSOFT 2015b. What are Mobile Apps? [verkkojulkaisu]. [Viitattu 2015-06-25.] Saatavissa: https://azure.microsoft.com/en-us/documentation/articles/app-service-mobile-value-prop-preview/

MICROSOFT 2015c. Azure Notification Hubs [verkkojulkaisu]. [Viitattu 2015-08-22.] Saatavissa: https://azure.microsoft.com/en-us/documentation/articles/notification-hubs-overview/

RAMNATH, Rajiv ja LOFFING, Cheyney 2014. Beginning iOS Programming For Dummies. New Jersey: John Wiley & Sons, Inc.