Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Yrittäjyys ja sähköinen liiketoiminta

2015

Alex Kivikoski

ALUSTARIIPPUMATTOMAN SOVELLUKSEN KEHITTÄMINEN KUNTOUTUMISEN TYÖKALUKSI

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

TURUN AMMATTIKORKEAKOULU

Tietojenkäsittelyn koulutusohjelma | Yrittäjyys ja sähköinen liiketoiminta

15.12.2015 | 40 sivua

Päivi Killström

Alex Kivikoski

ALUSTARIIPPUMATTOMAN SOVELLUKSEN KEHITTÄMINEN KUNTOUTUMISEN TYÖKALUKSI

Tämän opinnäytetyön tavoitteena oli kehittää HealthFOX-palvelua varten mobiilisovellus iOS-, Android- ja Windows Phone-käyttöjärjestelmille. Sovelluksen arkkitehtuuri suunniteltiin sellaiseksi, että se tukee sovelluksen helppoa laajennettavuutta jatkossa. Tämän pohjalta rakennettiin sovellukset kaikille kolmelle alustalle.

Projektin tilaaja, HealthFOX, on palvelu, joka tuo potilaalle työkalut nopeampaan kuntoutumiseen vammautumisesta. Palvelun keskeisiä elementtejä ovat audiovisuaaliset harjoiteohjeet, oman voinnin edistymisen seuraaminen, kommunikaatio lääkärin ja fysioterapeutin kanssa sekä ruokavalioon ja vammaan liittyvä opastus. Palvelun käyttäjiä ovat potilaat, fysioterapeutit ja lääkärit. Opinnäytetyössä keskityttiin potilaan käyttöliittymään. Työn tekijä on HealthFOX Oy:n perustajajäsen.

Sovellus kehitettiin käyttäen Visual Studio 2015-kehitysympäristöä. Monialustakehityksen tukena käytettiin Xamarin Platform -alustaa ja MvvmCross-sovelluskirjastoa. Sovelluksen arkkitehtuuri suunniteltiin MVVM-rakennetta noudattaen, jolloin sovelluksen näkymät ja niiden taustalla toimivat toiminnot ja tietorakenteet jaetaan omiksi tasoikseen. Tässä projektissa osa näistä tasoista jaettiin eri alustojen kesken. Opinnäytetyössä esitellään vaaditut askeleet kunkin alustan sovelluksen pohjan toteuttamiseksi.

Sovelluksen laajennusprojekti kolmelle alustalle todettiin onnistuneeksi ja sen tavoitteet hyvin täyttyneiksi. Sovellus vastaanotettiin hyvin potilaiden joukossa ja on nyt käytössä TYKS:n polven eturistisidevammasta kuntoutuvien potilaiden parissa. Jatkokehitystarpeina nousi esiin etäyhteystoimintojen rakentaminen ja sovelluksen laajentaminen tukemaan uusia käyttöalueita.

ASIASANAT:

Alustariippumaton, sovelluskehitys, Xamarin, MvvmCross, MVVM, fysioterapia, lokalisointi

BACHELOR´S THESIS | ABSTRACT

TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology | Entrepreneurship and e-Business

15.12.2015 | 40 pages

Päivi Killström

Alex Kivikoski

CROSS-PLATFORM APPLICATION DEVELOPMENT FOR REHABILITATION TOOL

The aim of the thesis is to develop a new application for HealthFOX rehabilitation service for mobile devices running on Android, iOS or Windows Phone 8. The architecture of the application was designed to support easily manageable future extensions.

The project's commissioner, HealthFOX, is a service that gives patients tools for faster recovery from injuries. The service's key elements are audio-visual exercise instructions, rehabilitation process visualization, communication tools for patients and medical staff, and instructions related to nutrition and injuries. The service is used by patients, doctors and physiotherapists. This thesis focuses on the patients' user interface. The author of the thesis is a founding member of HealthFOX Oy.

The application was developed by using Microsoft Visual Studio 2015. Xamarin Platform was used to enable cross platform development and the code sharing capabilities where further enhanced with MvvmCross library. The application architecture was designed based on MVVM design principles, which requires the application to be divided to layers of user interfaces, functions related to them and models that relate to the application data. Some of these layers where completely shared across the three platforms in this project. This thesis presents the required steps to create a working base for a cross platform solution on all three platforms.

The project was successful and the set goals were achieved. The application was received well among patients and is now in use by ACL injury patients at Turku University Hospital. Areas that were found to require further development were new communication tools for patients and medical staff, and studying new areas where the application could be used in.

KEYWORDS:

Cross platform, application development, Xamarin, MvvmCross, MVVM, physiotherapy, localization

SISÄLTÖ

LIITTEET

KUVAT

SANASTO

Termi	Termin selitys (Lähdeviite)
MVVM	Model-View-Viewmodel –arkkitehtuuri. Malli, jossa sovelluksen toiminnot jaetaan kolmeen tasoon: malliin, näkymään ja näkymämalliin. Näistä malli hallitsee tiedon säilyttämistä ja näkymämalli tietojen hakemista mallista käyttöliittymää varten. Näkymä käyttää näkymämallissa luotuja toimintoja ja on käyttöliittymä käyttäjälle.
PCL	Portable Class Library. Sovelluskirjasto, joka toimii rajatussa nimiavaruudessa niin, että se voidaan siirtää ilman muutoksia alustalta toiselle.
Nuget-pakettienhallinta	Työkalu, jolla ulkoisia sovelluskirjastoja voidaan helposti liittää projektiin ottaen näiden vaatimat muut sovelluskirjastot huomioon.
Lokalisointi	Tässä yhteydessä: sovelluksen eli kieliversioiden tuottaminen.
Referenssi	Projektiin lisätty sovelluskirjasto
Ohjelmistokehys	Framework
ACL	Polven eturistiside
Kuntoutus	Prosessi, jonka vammautunut potilas käy toimintakyvyn palauttamiseksi. Tähän kuuluu fysioterapeutin määräämiä harjoitteiden suorittamisia.

1 JOHDANTO

Opinnäytetyön kirjoittaja on ollut mukana perustamassa HealthFOX Oy:tä ja konseptoimassa sekä rakentamassa HealthFOX-palvelua. Projektin parissa on toiminut myös muita Turun Ammattikorkeakoulun opiskelijoita. **Marko Patanen** työskenteli Android-sovelluksen kehityksen apuna ja kirjoitti aiheeseen liittyen opinnäytetyön (Creating Android application using BLE sensor: a knee rehabilitation monitoring system, 2014). **Jaakko Paukamainen** toimi etäyhteystoiminnon parissa. ICT-talon **Gapstone-ryhmä** teki markkina-analyysiä HealthFOX-palvelun asiakkailta ja kehitti aikajanatoimintoa sovellukseen.

HealthFOX on palvelu, joka auttaa vammautunutta potilasta kuntoutumaan kokonaisvaltaisesti. Sen keskeisiä elementtejä ovat diagnosoituun vammaan ja sen tasoon pohjautuvat audiovisuaaliset kuntoutumisohjeet sekä kuntoutumisen seurantatyökalut lääkärin, potilaan ja fysioterapeutin välillä. Palvelu on suunniteltu lyhentämään kuntoutumiseen kuluvaa aikaa. HealthFOX palvelee vakuutusyhtiöiden tarpeita vähentämällä sairaslomien kustannuksia ja tarjoamalla työkalut potilaiden kuntoutumisen seurantaan.

Kun sovelluksen laajentaminen tukemaan useampaa alustaa tulee ajankohtaiseksi suunnitella tulevaisuudessa helposti ylläpidettävä arkkitehtuuri sovellukseen ja sen ympärille. HealthFOX-sovelluksen tapauksessa tuottavia alustoja ovat iOS, Android ja Windows Phone 8. Monialustakehityksen mahdollistavia ohjelmistokehyksiä ja muita tekniikoita on useita. Projektin kohteena on sopivan työkalun valinta ja toimivan sovelluksen rakentaminen niitä käyttäen. Opinnäytetyö keskittyy helposti hallittavan monialustakehitystä tukevan arkkitehtuurin suunnitteluun. Lopputuloksena on kolmella alustalla toimiva palvelu, jossa kunkin alustan käyttöliittymä on rakennettu erikseen ja näiden taustalla olevat toiminnot on toteutettu jaetusti.

Palvelun kolme keskeistä käyttäjäryhmää ovat potilaat, fysioterapeutit ja lääkärit. Näistä kullakin on omat roolinsa ja käyttöliittymänsä palveluun. Tässä opinnäytetyössä käsitellään ainoastaan potilaan käyttöliittymän toteutusta.

Sovelluksen käyttötapaukset ovat samat, kuin aiemmin toteutetussa Windows Phone-pilottisovelluksessa.

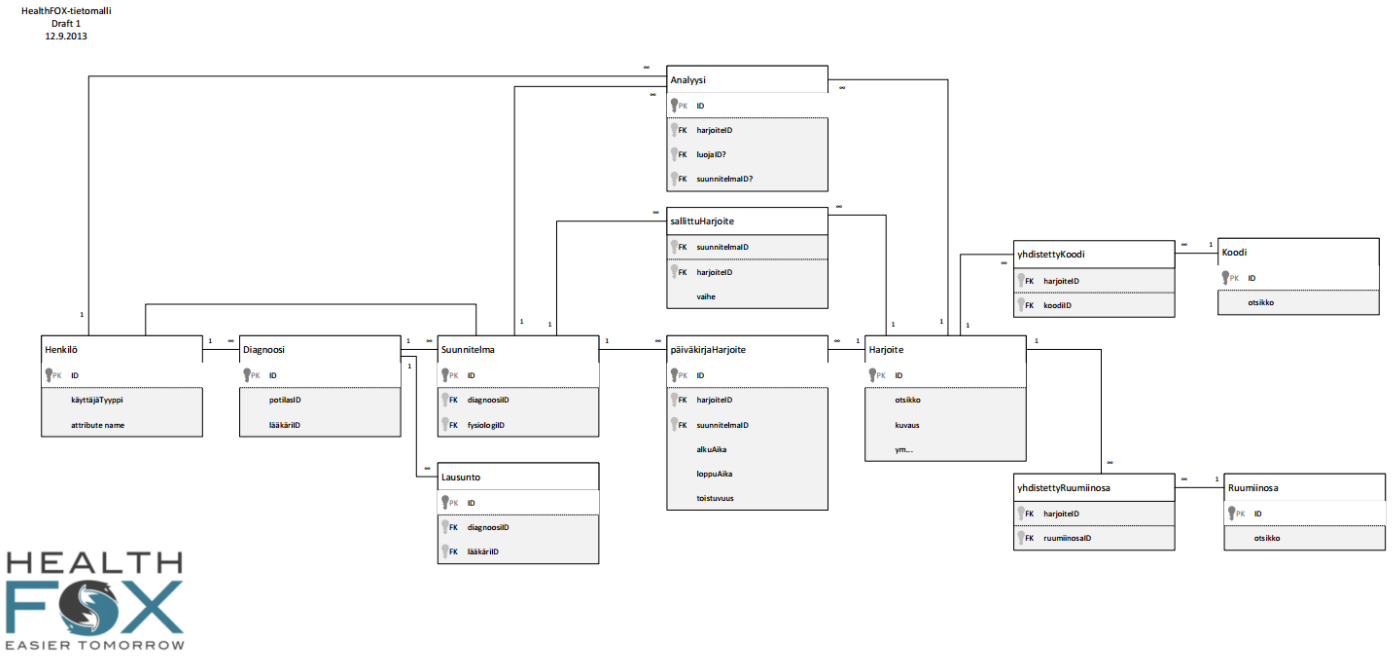
Kehitystyökalun valinnan kriteereinä olivat käyttäjäkokemus, lähdekoodin hallittavuus sekä tuetut ominaisuudet, käyttäjätuki ja kustannukset. Sovelluksen vaatimuksina on toimia päivittäisenä työkaluna muistuttamalla suoritettavista harjoitteista ja mahdollistamalla niiden sujumisen raportoinnin lääkärille ja fysioterapeutille. Sovelluksesta tulee löytyä niin offline- kuin online-tilassa potilaalle luotu liikekirjasto, hänelle soveltuvat ruokavalio-ohjeet, kuvaukset vammasta ja kuntoutumiskaaresta, potilaan oma historia sekä, kun mahdollista, etäyhteys lääkäriin ja fysioterapeuttiin.

Johdannossa olen esitellyt HealthFOX-palvelun. Tarve mobiilisovelluksen laajentamiselle kolmelle alustalle aloitti projektin, johon tämä opinnäytetyö perustuu. Seuraavaksi käsittelen tarkemmin projektin lähtökohtia.

2 PROJEKTI JA SEN LÄHTÖKOHDAT

Tässä luvussa käsittelen lähtökohtia, josta HealthFOX-palvelu lähtee laajentamaan mobiilisovellusta useammalle alustalle. Tarve helposti ylläpidettävälle arkkitehtuurille HealthFOX-palvelussa on ilmeinen. Esittelen lyhyesti sovelluksen graafisen ohjeistuksen ja taustalla olevan tietokannan sekä toteuttamiseen valitun työkalun.

Projektin lähtökohtana oli ympäristö, jossa palvelun tiedot sijaitsivat pilvipalveluna toimivana palvelimella tietokannassa. Tietokannan tietomalli oli suunniteltu aiemmin. Tämä malli (kuva 1) vastasi palvelun ydintoiminnallisuuksista ja sitä on sittemmin laajennettu. Tärkeimmät tietokannassa sijaitsevat tiedot olivat: henkilöt, diagnoosit, ICD-10-koodit, kuntoutussuunnitelmat, harjoitteet, harjoiteanalyysit, kuntoutustavoitteet, harjoitussuoritukset, opastusdokumentit, raportit voinnista ja käyttöoikeudet.



Kuva 1. Varhainen versio HealthFOX-tietokannan tietomallista

Projektin alkaessa oli valittava, kehittämmekö sovelluksen kunkin alustan natiivityökaluilla, vai käytämmekö työkaluja jotka sallivat osan tai kaikkien resurssien jakamisen näiden alustojen kesken. Natiivityökaluilla tarkoitetaan tässä kehitysympäristöä ja kieltä, jolla kullekin alustalle tehdyt sovellukset normaalisti toteutetaan. Projektia koskevista ympäristöistä Androidilla ohjelmointikielenä on Java ja kehitysympäristönä esimerkiksi Eclipse, iOS:lla Objective C ja XCode sekä Windows Phonella 8:lla C# ja Visual Studio. Lisäksi kunkin alustan käyttöliittymäsuunnittelu perustuu erilaisiin ennalta määrättyihin ohjenuoriin ja niille kehitettyjen sovellusten arkkitehtuuri ja testaaminen eroavat toisistaan. Kaikkien näiden opetteleminen vaatii paljon resursseja. Työkalut kuten Xamarin, Phonegap ja Titanium yksinkertaistavat kehitystä mahdollistamalla kaikille alustoille kehittämisen samalla työkalulla ja kielellä.

HealthFOX-sovelluksen toteuttamiseen valittiin työkaluksi Xamarin. Projektin puitteissa sen suurimmiksi eduiksi osoittautuivat ennalta tuttu käyttöympäristö ja ohjelmointikieli sekä laaja käyttäjäkunta. Koodin hallittavuutta pystyttiin parantamaan entisestään merkittävästi käyttämällä MvvmCross-kirjastoa.

Xamarin Platform on alusta, joka mahdollistaa Windows Phone 8-, iOS- ja Android-sovellusten kehittämisen C#-ohjelmointikieltä ja Visual Studio -kehitysympäristöä käyttäen. Xamarin kääntää sovelluksen iOS- ja Android-alustoilla natiivisovellukseksi (Xamarin 2015).

MvvmCross on sovelluskehys, joka helpottaa monialustasovelluksen kehittämistä MVVM-arkkitehtuuria käyttäen (MvvmCross 2015a). MvvmCross esitellään tarkemmin luvussa 4.

Kuva 2. Varhaiset käyttöliittymäpiirrokset sovelluksesta.

Sovelluksen käyttöliittymien suunnittelu perustui ennalta määriteltyihin käyttötapauksiin ja graafisen suunnittelijamme tekemiin ohjeistuksiin (kuva 2). Näiden ohjeistusten pohjalta oli jo aiemmin toteutettu pilottisovellus Windows Phone 8:lle.

Projektin toteutustapa

Projektin toteutustapa valikoitui tiimin ennalta hankitun osaamisen ja vahvuuksien mukaiseksi. Oma osaamiseni pohjautuu Microsoft Student Partner -tehtävissä hankittuun Windows- ja Visual Studio -osaamiseen sekä lukuisiin toteutettuihin Windows Phone -peleihin ja työkaluihin. Lisäksi toimimista tuki Nokian Windows Phone -tiimissä tehty harjoittelu, jonne pääsy oli valtakunnallisen kilpailun voittamisen ansiota.

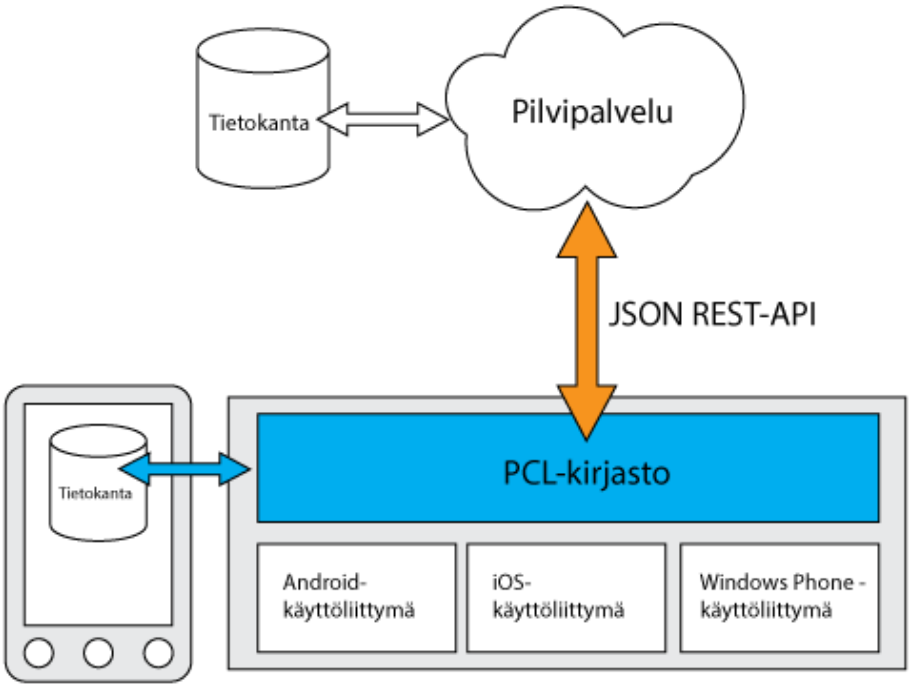
Toimintamallinani oli etätyöskentely ja viikoittainen projektinseuranta tiimin kesken. Versionhallintaa (Microsoft Team Foundation Server) käytettiin yhteistyön mahdollistamiseen eri yhteistyökumppaneiden kanssa. Määritellyt toiminnot purettiin niiden vaatimien työtehtävien selvittämisen myötä tehtäviksi, jotka osoitettiin Scrum-projektityömallin mukaisesti viikoittaisille työjaksoille. Valmistuneet

ominaisuudet julkaistiin kunkin alustan betatestauskanavia käyttäen koekäyttäjille. Koekäytössä huomattuihin ongelmiin puututtiin muun kehityksen ohella sitä mukaa kun niitä ilmeni.

Tässä luvussa kävin läpi projektin lähtökohdat: olemassa olevan tietokannan, valitun kehitystyökalun ja sovellusta varten laaditun graafisen ohjeistuksen sekä projektin toteutustavan. Seuraavassa luvussa käsittelen monialustakehitykseen soveltuvan arkkitehtuurin suunnittelua.

3 SOVELLUKSEN ARKKITEHTUURI

HealthFOX-palvelu, sen tarve kolmella alustalla toimivalle sovellukselle ja toteutukseen valittava työkalu on nyt esitelty. Tässä luvussa käsittelen monialustakehitystä tukevan sovellusarkkitehtuurin suunnittelua. Kerron HealthFOX-sovelluksen eri tasoista ja esimerkkitapauksen kautta havainnollistan, mitä kukin niistä tekee palvelussa. Esimerkkitapauksena toimii potilaan kivun tuntemusten tason tallentaminen järjestelmään.



Kuva 3. HealthFOX-palvelun arkkitehtuuri.

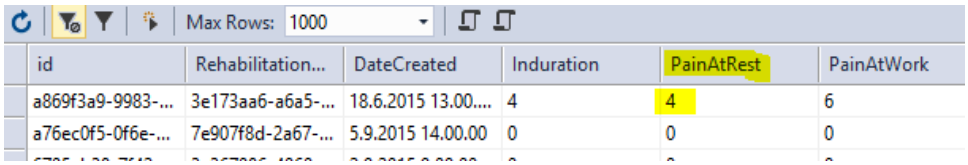
Usealle alustalle sovellusta kehitettäessä sovelluksen arkkitehtuurin suunnittelu on erityisen tärkeää. Suunnittelemalla järkevästi ylläpidettävä, laajennettava ja testattava pohja voidaan välttää ylimääräistä työtä sovelluksen elinkaaren aikana. Ideaalitilanteessa sama sovellus toimisi kaikilla alustoilla, jolloin kehitystyö tarvitsisi tehdä vain kerran. Tämä ei kuitenkaan nykyisillä käyttöjärjestelmillä ole mahdollista, pois lukien selainpohjaiset sovellukset, joiden maksimaalinen offline-

tallennustila ei riitä HealthFOX-palvelun satojen megatavujen tilatarpeeseen. Sovellus tai palvelu voidaan kuitenkin jakaa useisiin tasoihin, joista osa on jaettuja eri alustojen kesken. HealthFOX-sovelluksen käyttämät tasot on kuvattu kuvassa 3 ja tarkemmin sen jälkeen tässä luvussa.

Tietokanta varastoi palvelun tiedot. On tietoturvan kannalta järkevintä rajata suoraa käyttöoikeutta näihin tietoihin. Tyypillinen malli on rakentaa rajapinta eri sovelluksia varten. HealthFOX-palvelun tapauksessa tämä sijaitsee pilvipalvelussa. HealthFOX-tietokannan olennaisimmat taulut on esitelty aiemmassa luvussa (luku 2, kuva 1).

HealthFOX-palvelussa kaikki potilastieto sijaitsee palvelimella. Lisäksi potilaan käyttämässä sovelluksessa on paikallinen tietokanta, joka käyttää samaa tietomallia. Tähän paikalliseen tietokantaan kopioidaan potilaan oma data ja se synkronoidaan tarpeen tullen takaisin palvelimen tietokantaan.

Esimerkkitapaus: potilas on tallentanut kivun levossa tasoksi 4. Tieto on tallessa palvelimen tietokannassa (kuva 4).



Kuva 4. Kivun tuntemus levossa näkyvissä tietokannan rivinä.

Pilvipalvelu toimii sovelluksen apuna silloin, kuin laitteella on verkkoyhteys käytettävissä. Tällöin osa sovelluksen laskutoimituksista tai muusta logiikasta voidaan suorittaa palvelimella. Palvelimella toimivaan sovellukseen voidaan rakentaa standardoitu rajapinta, joka tukee monialustakehityksessä kaikkia haluttuja alustoja.

HealthFOX-sovelluksessa taustalla toimivaa palvelua ja palvelinta käytetään etänä ja ainoastaan internet-yhteyden yli, eli pilvipalveluna. Palvelu rakentaa sovelluksen pyytäessä palvelimen tietokannasta paketin potilaan kuntoutussuunni-

telmasta, harjoitusohjeista, aiemmista suoritteista, vamman tiedoista sekä ruokavalio-opastuksesta ja kuntoutuskaaren vaiheista. Sovellus lataa tiedot palveluun tehtyä rest-rajapintaa käyttäen ja tallentaa ne laitteen paikalliseen tietokantaan (kuva 3). Tämän jälkeen sovellus pystyy toimimaan itsenäisesti ilman pilvipalvelua. HealthFOX-pilvipalvelu vastaanottaa sovelluksesta, kun potilas suorittaa harjoituksia ja merkitsee tavoitteita saavutetuksi. Jos yhteyttä ei ole saatavilla, yritetään näiden tietojen lähettämistä uudelleen myöhemmin. Lisäksi HealthFOX-pilvipalvelussa on rajapinnat CameraFOX-sovellusta (fysioterapeutin työkalu potilaan harjoitusten kuvaamiseen ja analysoimiseen) ja selainkäyttöliittymää varten. Ulkoisille sidosryhmille on tarjolla rajapinnat, joilla uusia potilaita voidaan tuoda järjestelmään ja näiden raportteja viedä ulkoisiin tietojärjestelmiin.

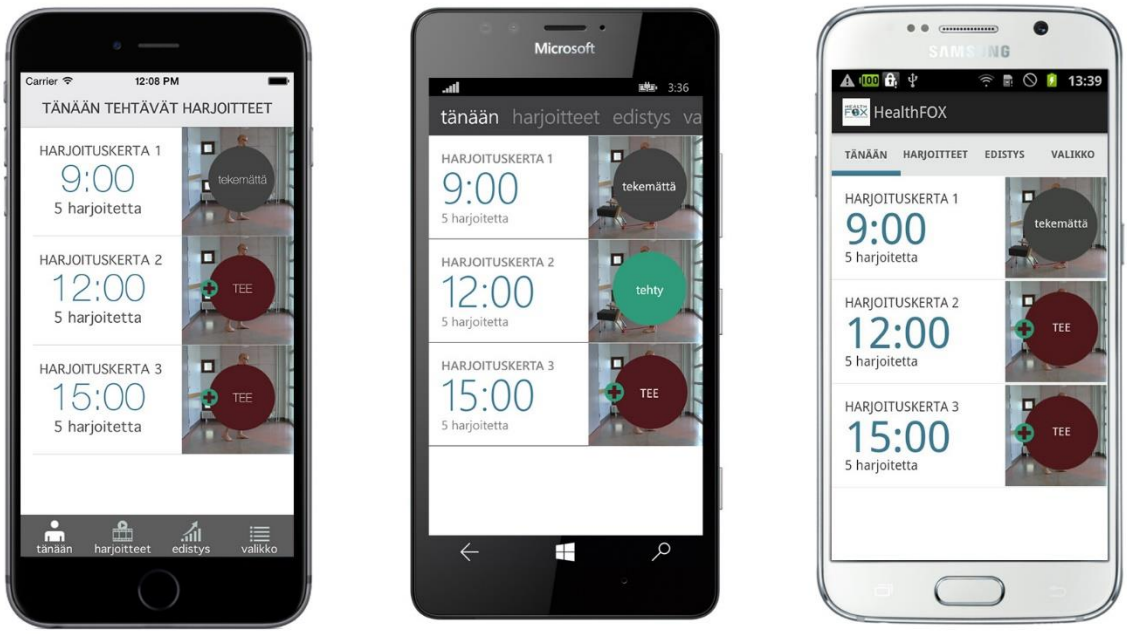
Sovelluskirjasto

Sovelluksen toiminnot voidaan jakaa useisiin tasoihin ja ne voidaan sulkea omaksi kirjastokseen. Tyypillisesti ainakin käyttöliittymä ja sen taustalla tapahtuva toiminnallisuus erotellaan toisistaan. Tähän ohjaavia malleja on useita, tunnetuimpia ovat MVC (Model-View-Controller), MVP (Model-View-Presenter) ja MVVM eli Model-View-Viewmodel (DotNetTricks 2014).

MVVM-arkkitehtuuri jakaa sovelluksen toiminnan kolmeen osaan: malliin, näkymämalliin ja malliin. Tämän kaltaisen arkkitehtuurin noudattaminen tekee koodista yleiskäyttöisempää, testattavampaa ja helpommin ylläpidettävää. MVVM on MVC- ja MVP-malleihin verrattuna helpommin toteutettava ja hallittava (Lehtinen 2011, 15). MVVM-malli on suunniteltu erityisesti hyödyntämään monipuolisia tiedon kytkemisominaisuuksia (data binding) käyttöliittymään.

HealthFOX-sovellus noudattaa MVVM-arkkitehtuuria. Sovelluksessa arkkitehtuurin malli- ja näkymämalli-tasot ovat erotettuna kirjastoksi, jota käytetään jaetusti kaikilla kolmella alustalla. Tämä tarkoittaa valtaosaa koko sovelluksen interaktiosta. Arkkitehtuuri on suunniteltu niin, että kaikki olennaiset toiminnot tarvitsee kirjoittaa vain kerran – näihin tehdyt muutokset toimivat välittömästi kaikilla kolmella alustalla. Tämä kirjasto toimii myös rajapintana puhelimen tallennustilaan ja tietokantaan (kuvassa 3 alempi tietokanta), jonne pilvipalvelusta ladatut tiedot

tallennetaan. Tämän mahdollistaa MvvmCross-ohjelmistokehys. MvvmCross on ohjelmistokehys (framework), joka tarjoaa joustavat työkalut MVVM-arkkitehtuurin mukaisen sovellukseen rakentamiseen. Se on suunniteltu toimimaan joustavasti Xamarin-ohjelmistokehyksen kanssa. Tällaista alustalta toiselle siirrettävissä olevaa sovelluskirjastoa kutsutaan PCL-kirjastoksi (MSDN 2011). Sovelluskirjastosta kerron lisää luvussa 4.

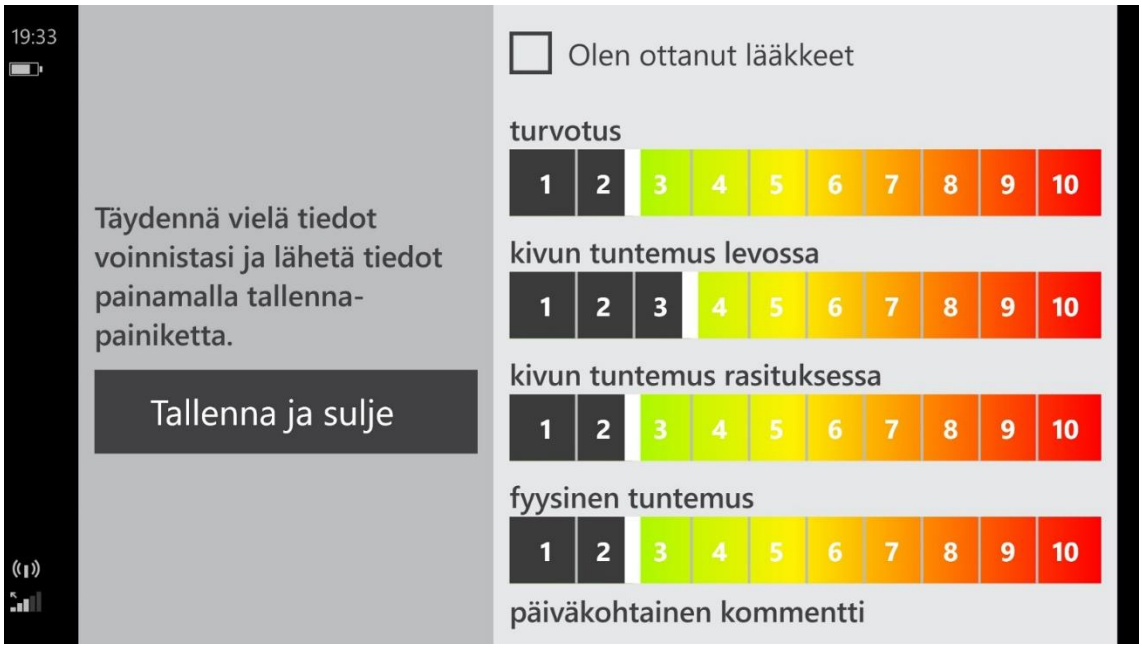


Kuva 5. HealthFOX-sovelluksen käyttöliittymä kolmella alustalla.

Sovelluksen käyttöliittymä

Työkalusta riippumatta monialustakehityksen lopputuotoksena on oltava kunkin alustan ymmärtämä sovellus. Se voi olla osittain tai kokonaan automaattisesti käännetty jonkin kehitystyökalun avulla. Alustakohtaisesti on syytä huomioida erityisesti käyttöliittymäsuunnittelun ohjenuorat. Käyttämällä alustalla suositeltuja navigaatio- ja muita yleisiä käytäntöjä käyttäjän on helpompi oppia käyttämään sovellusta, koska sovelluksen elementit ovat välittömästi tunnistettavissa (kuva 5).

HealthFOX-sovelluksen tapauksessa kunkin alustan sovellus käyttää samaa yhteistä sovelluskirjastoa. Alustakohtaisesti toteutetaan vain käyttöliittymien merkkaus. Tämä nopeuttaa sovelluskehitystä huomattavasti: kun haluttu toiminto on määritelty sovelluskirjastoon – esimerkiksi sisäänkirjautumissivulla sisäänkirjautuminen – kytketään kullakin alustalla vain haluttu nappi tai muu käyttöliittymäelementti tähän toimintoon. Lisäksi alustakohtaisesti toteutetaan toiminnot, joihin ei löydy valmista ratkaisua Xamarin- tai MvvmCross-ohjelmistokehyksistä. Tällainen ominaisuus on HealthFOX-sovelluksessa esimerkiksi hälytysten luominen puhelimen kalenteriin.



Kuva 6. Kivun tuntemuksen raportoiminen sovelluksessa.

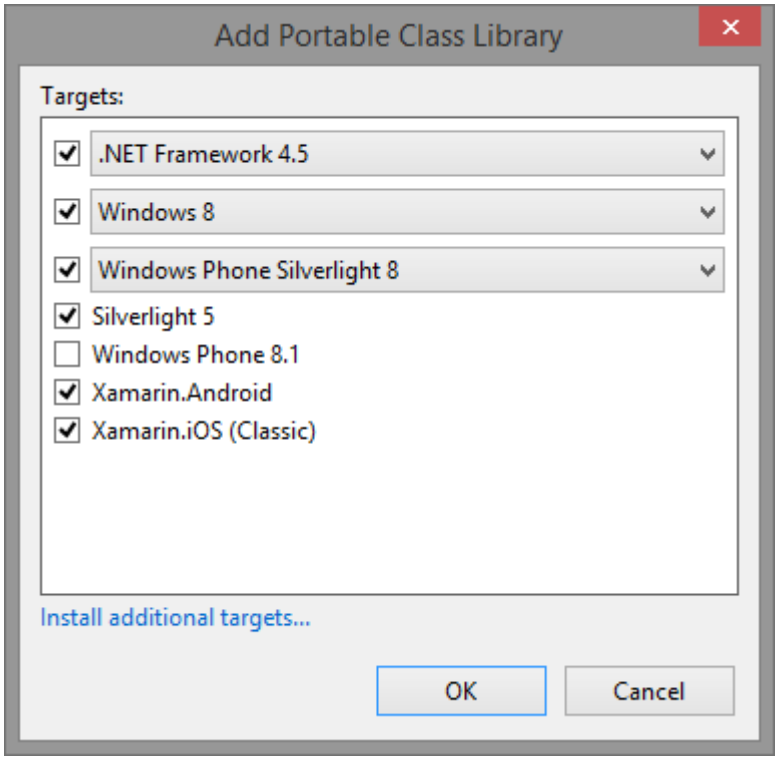
Esimerkkitapaus: Potilaan kivun tuntemusten raportointi on suunniteltu hyödyntämään terveydenhuollossa käytössä olevaa VAS-kivunarviointiasteikkoa. Potilaan käyttöliittymänä tuntemusten arviointiin toimivat liukupalkit (kuva 6). Nämä on kytketty PCL-kirjastoon (kuva 4) ja puhelimen tietokantaan, josta tieto siirtyy yhteyden ollessa saatavilla eteenpäin.

4 ALUSTARIIPPUMATON TASO – PCL-KIRJASTO

Aiemmissa luvuissa esittelin HealthFOX-palvelun ja sen arkkitehtuurin eri tasoineen. Tarve sovelluksen toteuttamiseen on esitelty. Tässä luvussa kerron sovelluksen ylemmän tason toteuttamisesta. Se on taso, jota kaikki kolme eri alustalle tehtyä sovellusta käyttävät.

Alustariippumattomia sovelluksia kehitettäessä haasteena on eri alustojen vaihteleva tuki käytetyille kirjastoille. Esimerkiksi Microsoftin omat Windows 8 ja Windows Phone 8 –alustat käyttävät suurilta osin samaa .NET-avaruutta, mutta täysin samaa nimiavaruutta ei voida niille kirjoitetuissa sovelluksissa käyttää. Tällaisissa tilanteissa voidaan alustojen kesken jaettavaa koodia kirjoittaa Portable Class Library (PCL)-tyyppiseen projektiin. PCL-projektit käyttävät vain määrättyä osaa .NET-nimiavaruudesta. Tämä mahdollistaa niihin viittaamiseen myös alustoilla, joilla kaikki .NET-nimiavaruuden osat eivät ole käytettävissä (Lähde? Xamarin). Käytettävä .NET-subset valikoituu käyttäjän valitsemien tuettavien alustojen perusteella. MVVM-arkkitehtuurissa PCL-kirjastoja käytetään tyypillisesti mallin ja näkymämallin kuvaamiseen (MSDN 2015). MvvmCross-projekteissa PCL-aliprojekti nimetään tyypillisesti .core-päätteellä (Lodge 2013).

HealthFOX-sovelluksessa alustariippumaton kehitys aloitettiin PCL-kirjaston luomisesta. Se tapahtuu klikkaamalla avoinna olevaa ratkaisua (solution) hiiren oikealla painikkeella ja valitsemalla Add > Add New Project. Aukeavasta ikkunasta valitaan tyypiksi Class Library (Portable). Visual Studio kysyy, mitä alustoja luodulla kirjastolla halutaan tukea – tässä tapauksessa valitaan .NET Framework 4.5, Windows 8, Windows Phone Silverlight 8, Silverlight 5, Xamarin.Android ja Xamarin.iOS (kuva 7). Kahden viimeisen näkyminen listalla edellyttää Xamarin-sovelluskehyksen asentamista.

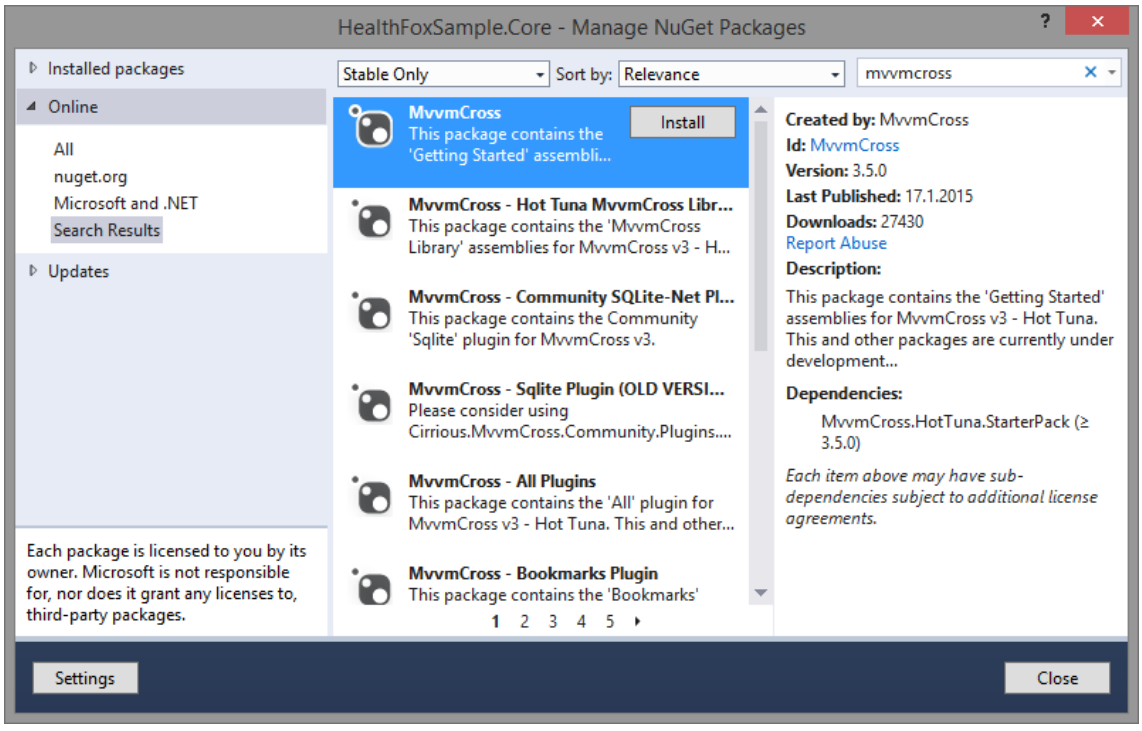


Kuva 7. Tuettujen alustojen valitseminen.

Kun PCL-kirjasto on luotu, siihen toteutetaan MVVM-arkkitehtuurin malli- ja näkymämalli-tasot sekä niiden apuna toimivat luokat. Nämä rakennetaan MvvmCross-kirjastoa hyödyntäen. Käsittelen kutakin näistä seuraavissa luvuissa.

4.1 MvvmCross-kirjaston lisääminen projektiin

MvvmCross-kirjasto lisätään projektiin Nuget-pakettienhallintajärjestelmän avulla painamalla projektin nimeä hiiren oikealla painikkeella ja valitsemalla valikosta "Manage Nuget Packages...". Aukeavasta pakettienhallintanäkymästä haetaan hakukenttää käyttäen MvvmCross-paketti ja se asennetaan klikkaamalla Install-painiketta (kuva 8). Nuget-pakettienhallinta tekee tarvittavat muutokset ja lisää tarvittavat tiedostot projektiin automaattisesti. MvvmCross-paketti sisältää yhden valmiin esimerkkinäkymämallin (FirstViewModel.cs), jonka voi halutessaan poistaa.



Kuva 8. MvvmCross-kirjaston asentaminen Nuget-pakettienhallinnasta.

Tämän jälkeen lisätään vastaavalla tavalla tarvittavat laajennokset. HealthFOX-projekti käyttää seuraavia MvvmCross-laajennoksia:

- MvvmCross.Community.Plugins.Sqlite
  - ✓ Rajapinta puhelimen paikalliseen tietokantaan

- MvvmCross.Localization
  - ✓ Mahdollistaa sovelluksen kieliversioinnin PCL-projektin tasolla
- MvvmCross.Plugins.DownloadCache
  - ✓ Automatisoi ladattujen kuvien tallentamisen välimuistiin
- MvvmCross.Plugins.File
  - ✓ Mahdollistaa tiedostojen tallentamisen puhelimen muistiin PCL-projektista käsin
- MvvmCross.Plugins.Json
  - ✓ Toiminnot json-tyyppisen tiedon käsittelyyn
- MvvmCross.Plugins.Messenger
  - ✓ Joustavat toiminnot viestien välittämiseen sovelluksen eri tasojen välillä

Asennetut laajennokset lisäävät vain tarvittavat rajapinnat PCL-projektiin. Jotta toimintoja voidaan käyttää, on myös kunkin alustan sovellukseen lisättävä kyseisen alustan versio laajennoksesta. Kun laajennokset on asennettu, tuodaan projektiin sovelluskohtaiset tiedot. Käyn tässä opinnäytetyössä seuraavaksi läpi HealthFOX-sovellusta koskevat toiminnot.

4.2 Tietomallin tuominen projektiin

Sovelluksen tietomallin käyttämät mallit lisätään Models-hakemistoon. MvvmCross ei aseta niille erityisiä vaatimuksia, mutta projektissa käytetty Sqlite.Community-kirjasto toimii vain yksinkertaisten tietotyyppien kanssa. Se vaatii, että tietokantaan tallennettavien luokkien avainkentät on merkitty [PrimaryKey]-määreellä. Lisäksi voidaan merkitä [AutoIncrement] inkrementaalisesti kasvavan avaimen arvon määrittelemiseksi.

HealthFOX-sovelluksen käyttämät mallit pohjautuvat palvelimen tietomalliin. Pohja malleille on haettu HealthFOX-pilvipalvelun generoimasta ado.net-tietomallista ja sen luomista luokista. Pääavaimien guid-tietotyyppi osoittautui ongelmalliseksi Sqlite.community-ohjelmistokirjaston kanssa. Ongelman kiertäminen

onnistui luomalla haasteita aiheuttaneisiin tauluihin kokonaisluku-tyyppinen pääavain ja poistamalla vanhasta guid-avainkentästä pääavain-määre.

4.3 Näkymämalleja tukevien toimintojen luominen

Hyvän ohjelmistosuunnittelun lähtökohtana on abstraktien rajapintojen suunnittelu. Tämä on oleellista yksikkötestauksen mahdollistamiseksi (MSDN). Näkymämallien käyttämät toiminnot, kuten tiedon hakeminen ja tallentaminen tietokantaan ja erinäiset verkkokutsut, toteutetaan palveluina (service), jotka implementoidaan omina luokkinaan. MvvmCross huolehtii näiden olioiden luomisesta ja löytämisestä (Injection of Service, IoC). Suurin osa sovelluksen bisneslogiikasta sijaitsee näissä palveluissa.

4.3.1 Tietokannan käsittely palvelussa

Data sovellukseen haetaan Microsoftin Azure SQL-tietokannasta JSON REST-apin avulla. Rajapintaa kutsutaan HTTPS-kutsulla ja vastauksena saadaan JSON-tyyppinen merkkijono. Sen deserialisointiin (deserializing) käytetään HealthFOX-projektissa MvxJsonConverter-luokkaa. Tiedon lähettäminen tapahtuu vastaavalla tavalla. Toiminnot on toteutettu asynkronisesti, jolloin verkkokutsujen kesto ei vaikuta käyttöliittymän toimintaan.

HealthFOX-projektissa käytetään Sqlite-tietokantaa MvvmCross-kirjaston Sqlite.Community-laajennoksen avulla. Sitä käytettävissä on huomioitava, että laajennoksen nykyinen versio toimii Androidilla vain yksisäikeisesti, joten useita toimintoja ei ole mahdollista suorittaa rinnakkain. Tämä aiheutti projektin aikana muutostarpeita palveluun.

Kaikki sovelluksen tarvitsema tieto tulee paikallisesta tietokannasta ja puhelimen massamuistiin tallennetuista mediatiedostoista, mikä on edellytyksenä vaatimusmäärittelyissä listatulle offline-toiminnallisuudelle. Sovellusta varten koodattu datapalvelu huolehtii siitä, että tiedot päivitetään, mikäli verkkoyhteys on saatavilla,

ja muissa tilanteissa muutokset tehdään vain paikalliseen tietokantaan. Verkkoyhteyden saatavuuden tarkistus on toteutettava sovelluksen tasolla.

Tietokantakutsujen vähentämiseksi ajon aikana muuttumattomat tiedot ladataan sovelluksen datapalvelussa vain kerran. MvvmCross huolehtii IoC-toiminnallisuuden avulla siitä, että ne sisältävä olio on olemassa.

4.3.2 Mediatiedostojen lataaminen ja tallentaminen

Mediatiedostojen tallentamiseen puhelimen muistiin käytetään HealthFOX-projektissa MvvmCross-kirjaston File-laajennosta. Sen avulla palvelimelta vastaanotettu data voidaan tallentaa alustariippumattomasti sovellukselle varattuun massamuistiin. Videotiedostojen koodekiksi valittiin h.264, jolle löytyy tuki kaikilta vaadituilta alustoilta. Ladattavien tiedostojen verkko-osoitteet saadaan JSON-merkkijonona (luku 3, kuva 3) ja niiden lataamista varten kehitettiin asynkroninen palvelu. Suunnittelussa otettiin huomioon se, että tiedostojonon lataaminen saattaa keskeytyä, jolloin uudelleen yritettäessä on jatkettava siitä tiedostosta, mihin edellinen yritys keskeytyi.

4.3.3 Kieliversioiden huomiointi

Sovelluksen tekstiresurssit on sijoitettu PCL-projektiin .resx-tiedostona, jonka muokkaaminen Visual Studiossa on helppoa. Sen hyödyntämiseen kirjoitettiin esimerkin pohjalta ResourceManager-luokkaa hyödyntävä palvelu. Lisäksi näkymäluokkien pohjaluokkaan lisättiin tätä palvelua käyttävä aputoiminto niin, että kaikilla alustoilla käyttöliittymän tekstit voidaan liittää suoraan PCL-projektin tekstiresurssiin.

4.4 Näkymämallit (ViewModel-luokat)

ViewModels-kansioon luodaan kutakin sovelluksen näkymää vastaa näkymämalli. Se määrittelee näkymässä näytettävät tiedot ja siinä suoritettavissa olevat

toiminnot, ja mitä kyseisten toimintojen yhteydessä tapahtuu. Tyypillisesti tiedon hakemiseen ja toimintojen toteuttamiseen käytetään yhtä tai useampaa palvelua (service). MvvmCross huolehtii automaattisesti näkymämallin yhdistämisestä oikeaan näkymään luokan nimen perusteella (MvvmCross 2015b). Näkymämallin konstruktorissa on välitettävä tarvittavat palveluiden rajapinnat, jotta IoC voi palauttaa tarvittavat oliot automaattisesti.

MvvmCross-näkymämallien on perittävä MvxViewModel-luokka. Sen kenttien (property) on reagoitava muuttuneisiin arvoihin esimerkiksi käyttämällä RaisePropertyChanged()-metodia, jolloin näkymä päivittyy automaattisesti vastaamaan uutta arvoa. MvvmCross-näkymämalleja kehitettäessä on hyödyllistä lisätä Visual Studioon pikakoodit (code snippet), jotka nopeuttavat uusien kenttien ja toimintojen lisäämistä näkymämalliin (StackOverflow 2013).

```
private DailyInput dailyInput;
public DailyInput DailyInput
        {
             get { return dailyInput; }
             set {
                         if (dailyInput == value) return;
                         dailyInput = value;
                         RaisePropertyChanged(() => DailyInput);
             }
        }

// -- Ote sivun lataamisesta:

DailyInput = _dataService.GetDailyInputs().Where(s => s.id ==
guid_id).FirstOrDefault();
```

Kuva 9. Kivun tuntemukset sisältävän objektin lataaminen näkymämalliin.

HealthFOX-projektissa näkymämalleille yhteiset toiminnot sijoitettiin BaseViewModel-nimiseen luokkaan, jonka kukin näkymämalli perii. Kutakin sovelluksen näkymää varten toteutettiin näkymämalli: näitä ovat esimerkiksi LoginViewModel.cs, ExerciseViewModel.cs ja WelcomeViewModel.cs. Aiemmin mainittuja palveluja käytetään näissä tietojen hakemiseen ja tallentamiseen.

Esimerkkitapaus: kivun tuntemukset raportoidaan DailyInputView-nimisessä näkymässä. Tiedot sisältävät DailyInput-luokassa, joka ladataan näkymämalliin (kuva 9).

4.5 Converter-luokat

Converters-kansioon lisätään tarpeelliset MvxValueConverter<T,T>-luokan perivät luokat, joilla voidaan sovelluksen ajon aikana tehdä muunnoksia esimerkiksi käyttäjän palautteesta tietokantaan tallennettavaan tietoon.

Hyvän MVVM-suunnittelun lähtökohtana on, että mahdollisimman vähän sovelluskohtaista logiikkaa on koodattuna näkymäluokkiin (Views). Tätä voidaan edistämällä toteuttamalla erinäiset tietotyyppien väliset ja niiden sisäiset muunnokset omassa luokassaan. On kuitenkin aiheellista miettiä, tarvitaanko tätä ajonaikaista muunnosta vai voidaanko tarvittavat muunnokset tehdä näkymämallissa, joka on ajoittain suorituskyvyn kannalta edullisempaa. Muunnoslogiikan "piilottaminen" converter-luokkaan saattaa myös tehdä koodista vaikeammin hahmotettavaa (Hubbard 2014).

Tässä luvussa esittelin, kuinka HealthFOX-sovelluksen alustariippumaton kerros, PCL-kirjasto, luotiin. Se on rakennettu MVVM-rakennetta noudattaen. MVVM-rakenteen tärkeimmät osat ovat malli (model), näkymä (view) ja näkymämalli (view model). PCL-kirjastossa sijaitsevat näistä kaksi ensimmäistä, sekä lisäksi näitä tukevat luokat kuten Converter-luokat ja palvelut (services). Nämä toteutettiin MvvmCross-sovelluskehystä käyttäen.

Seuraavaksi käyn läpi kunkin alustan sovelluksen toteuttamisen tätä PCL-kirjastoa hyödyntäen.

5 ANDROID-SOVELLUS

Projektin Android-sovellus on Xamarinin Android-projekti, joka referoi PCL-sovellusta ja tarvittavia MvvmCross-kirjastoja. Rakenteeltaan se vastaa natiiviprojektia ja myös käyttöliittymä toteutetaan identtisellä axml-merkkauksella. Muutoin kielenä on C#.

Projekti luodaan Visual Studiossa valitsemalla projektityypiksi Xamarin-kohdan alta löytyvä Android Project. Projektin on oltava samassa ratkaisussa (solution) ja nimiavaruudessa PCL-projektin kanssa, ja se nimetään tyypillisesti .droid-päätteellä.

5.1 Tarvittavien kirjastojen lisäämien projektiin

Ensimmäiseksi projektiin lisätään referenssi PCL-projektiin. Tämän jälkeen lisätään Nuget-pakettienhallinnalla (luku 4, kuva 8) MvvmCross-kirjasto ja kaikki PCL-projektin käyttämät laajennokset. MvvmCross-paketti luo ja muokkaa automaattisesti vaaditut tiedostot ja esimerkkinäkymän. Lisäksi Nuget-pakettienhallinnasta asennettiin taaksepäin yhteensopivuutta lisäävät Xamarin.Android.Support.v4- ja Xamarin.Android.Support.v7.AppCompat-kirjastot.

5.2 Käyttöliittymien rakentaminen

Xamarinin Android-projekteissa käytetään normaalia axml-käyttöliittymämerkkausta. MvvmCross-kirjastoa käytettäessä voidaan käyttöliittymäelementtien ominaisuuksia (attribute) kytkeä (bind) näkymää vastaaviin näkymämalleihin. Tämä tapahtuu MvxBind-ominaisuuden asettamalla.

Esimerkkitapaus: Kivun tuntemuksen arviointi tapahtuu liukupalkkia koskettamalla. Tieto on kytketty näkymämalliin (kuva 10).

```
<TextView
            android:text="Medium Text"
            local:MvxBind="Text [PainAtRest]"
            style="@style/BlueTitle"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:id="@+id/textView2"
            android:layout_marginLeft="10dp" />
<SeekBar
            style="@style/PainBar"
            local:MvxBind="Progress DailyInput.PainAtRest"
            android:id="@+id/seekBar2"
            android:layout_marginRight="10dp"
            android:layout_marginLeft="5dp" />
```

Kuva 10. Arvon kytkeminen käyttöliittymään Android-näkymässä.

Android-laitteille sovellusta suunnitellessa on syytä ottaa huomioon eri kokoiset ruudut, jolta sovellusta on mahdollista käyttää. Keinoja tähän ovat muun muassa mittojen määritteleminen resoluutiosta riippumattomin dp-yksiköin ja erillisten näkymien ja resurssien määrittely eri kokoisille näytöille (Android Developers 2015).

5.3 Aktiviteetit (Activity)

Android-näkymät toteutetaan aktiviteeteilla (activity), jotka voivat täyttää koko sivun tai toimia esimerkiksi kelluvina ikkunoina. Vain yksi aktiviteetti voi toimia kerrallaan aktiivisena ja reagoida käyttäjän toimintaan. Aktiviteetit liitetään käyttöliittymämerkkaukseen Xamarin-projekteissa SetContentView()-metodilla. MvvmCross yhdistää näkymän sitä vastaavaan ViewModel-luokkaan ja näkymästä toiseen siirryttäessä huolehtii näiden luomisesta. Tällöin aktiviteettien on perittävä MvxActivity-luokka. Useissa tilanteissa kaikki näkymän tarvitsema interaktio syntyy näkymämallin ja näkymän merkkauksen avulla eikä aktiviteettiin tarvitse lisätä koodia.

5.4 Näkymien toteutus fragmentoimalla

Uusien Android-sovellusten toteutuksessa on suositeltua jakaa käyttöliittymäelementit omiksi palasikseen (fragment), joita voidaan eri aktiviteeteissa (Activity)

asetella yksi tai useampi rinnakkain. Ne käyttävät samaa käyttöliittymämerkkausta kuin pelkät aktiviteetit, mutta niiden elinkaari on erilainen. MvvmCross huolehtii palasien ja aktiviteettien elinkaaresta. ViewModel-luokasta tuleva tieto yhdistetään palaseen BindingInflate()-metodilla. MvvmCross-toimintoja käytettäessä on perittävä MvxFragment-luokka.

HealthFOX-sovelluksessa palasia hyödynnetään etusivun näkymässä, jossa eri näkymät on sijoitettu ViewPager-elementtiin niin, että niitä voidaan selata sivusuunnassa näyttöä pyyhkäisemällä.

5.5 Alustakohtaiset toiminnallisuudet

Alustakohtaisia toiminnallisuuksia ovat esimerkiksi kalenterimuistutukset, videotiedostojen toisto, huomautukset (notifications) sekä puheluiden luominen. Osaa näistä on mahdollista käyttää PCL-projektista käsin valmiin laajennoksen avulla, osa vaatii omaa toteutusta. Toteuttaminen on Xamarin-projekteissa mahdollista niin natiivi- (Java) kuin C#-kielelläkin.

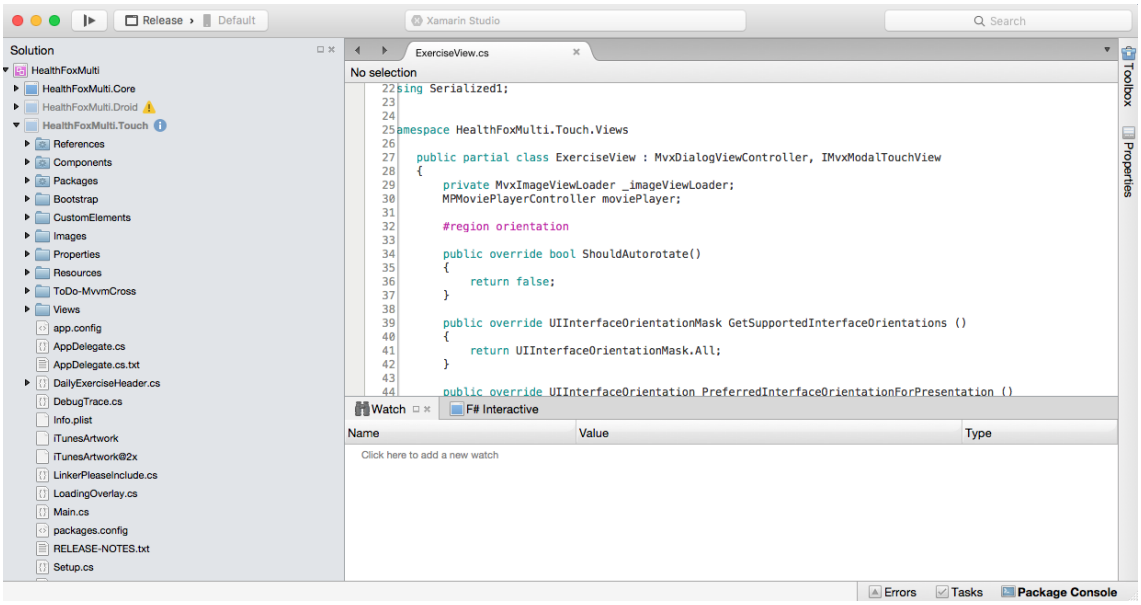
HealthFOX-sovellus luo puhelimeen oman kalenterinsa, jossa näkyvät potilasta varten luodut harjoitteet. Tämä on toteutettu ContentProvider-luokkien avulla. Lisäksi alustakohtaisista toiminnallisuuksista käytetään videon toistamista, jota varten Androidilla on rakennettava myös oma näkymänsä. Se käyttää VideoView- ja MediaController-luokkia.

5.6 Yhteensopivuus

Android-sovelluksissa taaksepäin yhteensopivuus voidaan taata käyttämällä esimerkiksi Android Support Library v7 AppCompat-kirjastoa. Se lisää vanhoihin Android-versoihin tuen uusille ominaisuuksille, kuten projektin käyttämälle ActionBar-toimintopalkille. Vanhin tuettu Android-versio asetetaan AndroidManifest.xml-tiedostossa. Jos sovellus vaatii oikeuksia esimerkiksi kalenteriin kirjoittamiseen tai hälytysten luomiseen, on nämä myös kyseiset toiminallisuudet määriteltävä tässä tiedostossa.

6 IOS-SOVELLUS

Xamarin iOS-projekti luodaan ja on muokattavissa Visual Studiossa, mutta sen kääntäminen vaatii Mac OS X-käyttöjärjestelmällisen tietokoneen. Projektin toteutushetkellä MvvmCross-yhteensopivien näkymien luominen vaatii Xcoden käyttämistä. Sovelluksen kehitykseen käytettiin Mac OS X-käyttöjärjestelmää ja Xamarin Studio-sovellusta (kuva 11).



Kuva 11. Xamarin Studio-kehitysympäristö käytössä.

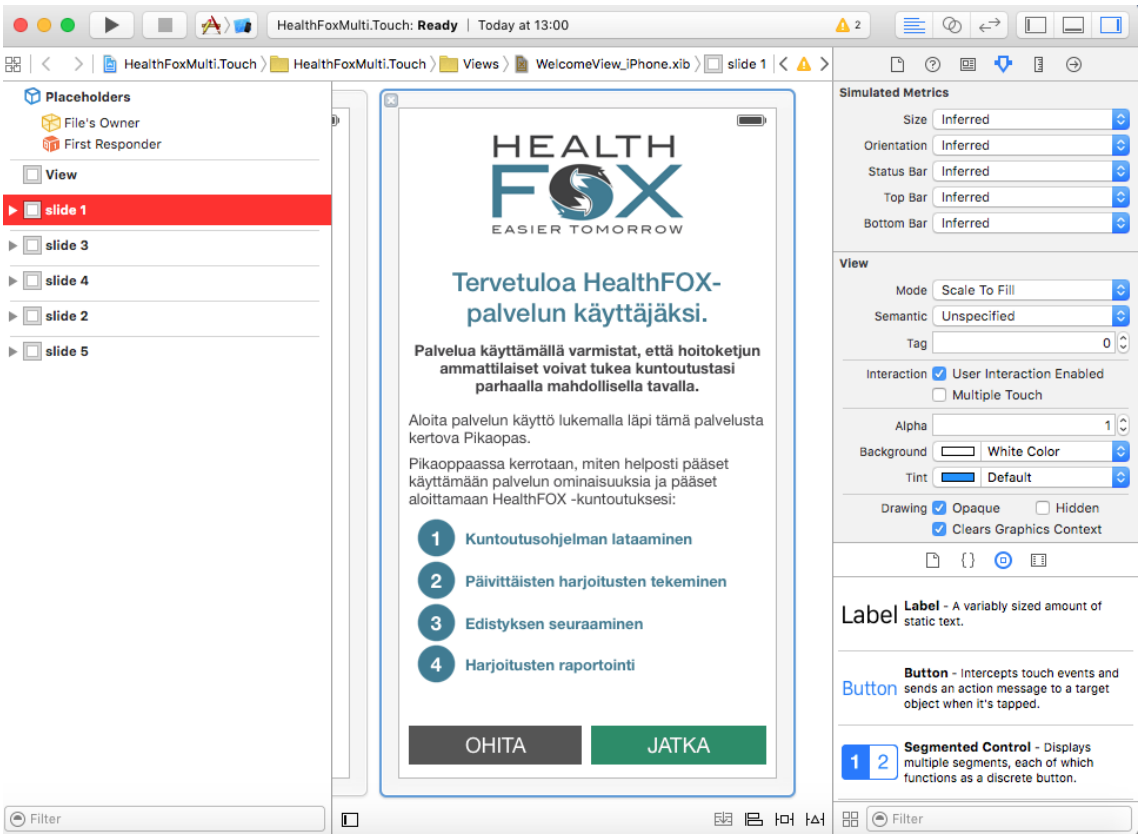
6.1 Näkymien luominen

Xamarin iOS-projektien näkymät sijoitetaan Views-hakemistoon valitsemalla x-valikosta New Universal View Controller (tai vaihtoehtoisesti iPhone- tai iPad View Controller tarpeen mukaan). Ne koostuvat .xib-merkkaustiedostosta, Xamarinin generoimasta .designer.cs-tiedostoista ja näkymää hallitsevasta .cs-tiedostoista. Käyttöliittymät voidaan nykyään luoda sekä Xamarin Forms-työkalulla että XCoden omalla editorilla. Projektin alkaessa vain jälkimmäinen tapa oli olemassa, joten näkymät luotiin sillä. Käyttöliittymäelementit voidaan luoda myös suoraan koodista käsin.

6.1.1 Käyttöliittymän rakentaminen XCodella

Valmis näkymä voidaan rakentaa XCodella (kuva 12). Storyboard-toiminnot eivät projektin rakentamishetkellä olleet tuettuna MvvmCrossin ja Xamarinin kanssa. Näkymän elementtien kytkeminen PCL-projektin ViewModel-luokkiin edellyttää, että kytkettäville elementeille luodaan XCodessa outlet-rivit .designer.cs-tiedostoon. Tämä tapahtuu avaamalla kyseisen näkymän .h-tiedosto Assistant Editor-näkymään ja raahaamalla control-näppäin pohjassa haluttu elementti Assistant Editoriin.

Monimutkaisempia näkymiä voidaan koota taulukkonäkymien avulla. Taulukon soluille voidaan luoda oma View Controller ja sille lisätä outlet-rivit samalla tavalla kuin näkymillekin. Solujen View Controller kytketään taulukkoon .cs-tiedostosta käsin.



Kuva 12. Käyttöliittymän luonti XCodessa.

Näkymä voidaan koota myös ilman .xib-tiedostoa luomalla halutut elementit .cs-tiedostossa.

6.1.2 Näkymämallin kytkeminen näkymään

Jos näkymän elementtejä halutaan yhdistää sitä vastaavaan näkymämalliin, on kytkettävä joka elementti erikseen CreateBindingSet<>()- ja Bind()-metodeilla. Tältä osin MvvmCross-näkymät eroavat iOS-alustalla Androidista ja Windows Phonesta, joilla yhdistäminen tapahtuu näkymän merkkauksessa.

Taulukkonäkymiä täydentävät solut kytketään näkymämalliin DelayBind()-metodilla.

6.1.3 Käyttöliittymien rakentamien Mono Touch Dialog -kirjastolla

Mono Touch Dialog-kirjaston avulla on helppo rakentaa monipuolisia automaattisesti aseteltuja näkymiä Xamarin-sovellukseen .cs-koodista käsin. Se sisältää valmiit luokat tekstikentille, päivämääräkentille, napeille ym. yleisesti käytetyille elementeille. Mono Touch Dialog luo halutulla tavalla järjestellyn näkymän. MvvmCross sisältää muokatun version tästä kirjastosta ja mahdollistaa näiden kenttien kytkemisen näkymää vastaavaan ViewModeliin.

HealthFOX-sovelluksessa Mono Touch Dialog-sivuja käytetään laajalti. Sovellusta varten rakennettiin myös omia luokkia täydentämään Mono Touch Dialogin kenttätyyppejä. Se onnistuu perimällä sopiva luokka kirjastosta ja laajentamalla haluttuja toimintoja. Mono Touch Dialog-kentät voivat käyttää myös .xib-näkymiä.

6.2 Alustakohtaiset toiminnallisuudet

Kalenterimuistutukset ja videoiden toisto toteutettiin niitä vastaavissa näkymissä. Muistutusten lisäämiseen puhelimen muistiin käytettiin UIApplication.Share-

dApplication.ScheduleLocalNotification()-metodia. Tämän käytössä on huomioitava tietotyyppimuunnos Xamarinin käyttämän .NET String-tyypin ja iOS-alustan NSString-tyypin välillä. Videoiden toistamiseen käytettiin MPMoviePlayerController-luokkaa.

7 WINDOWS PHONE 8 -SOVELLUS

Toisin kuin Android- ja iOS-projektit, Windows Phone-projekti on normaali natiiviprojekti. Se käyttää PCL-projektia ja tarvittavia MvvmCross-kirjastoja. Erona normaaliin Windows Phone-projektiin on vain näkymien luokat, jotka ovat MvxPhonePage-tyyppisiä.

MvvmCross-kirjaston lisääminen projektiin

Kun projekti on luotu, lisätään referenssit PCL-projektiin ja MvvmCross-kirjastoihin vastaavasti kuin Androidilla ja iOS:lla. Projektia hiiren oikealla näppäimellä klikkaamalla avataan Nuget-pakettienhallinta ja lisätään aiemmin mainitut ohjelmakirjastot.

Windows Phone-projektissa käytettiin lisäksi Telerik.Windows.Data, Telerik.Windows.Core ja Telerik.Windows.Controls-kirjastoja potilaan edistymistä kuvaavien kaavioiden luomiseen.

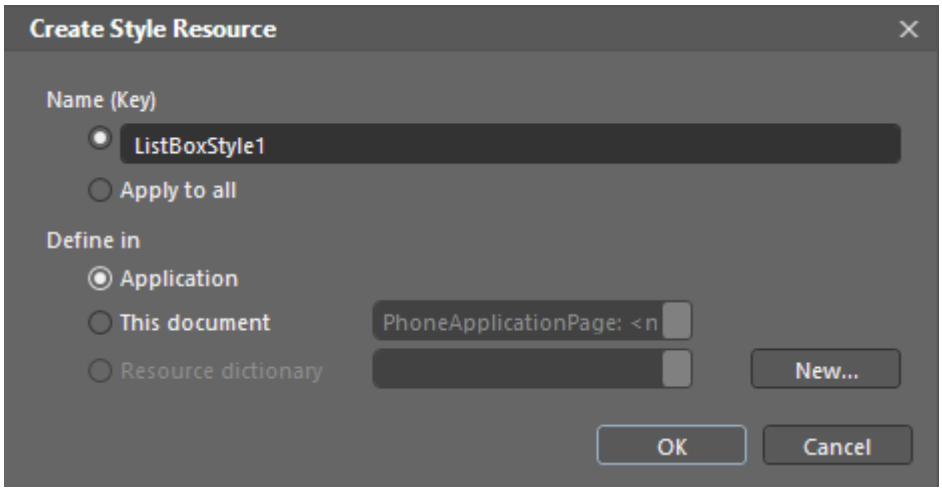
7.1 Näkymien merkkaus

Jotta näkymän elementit voitaisiin kytkeä näkymää vastaavaan näkymämalliin, on sivun oltava MvxPhonePage-tyyppinen. Itse kytkeminen voidaan tehdä normaalisti {Binding}-merkkauksella (kuva 13). Tällöin käyttöliittymä hakee merkittyihin kenttiin arvot automaattisesti näkymallista, kun näkymä ladataan tai kun näkymämallin tiedot muuttuvat. Jos kytkettyä arvoa käyttäessä halutaan käyttää Converter-luokkia tietotyypin tai muun ominaisuuden muunnokseen, on Windows phone-projektissa kyseisestä Converter-luokasta tehtävä oma apuluokka MvxNativeValueConverter-luokka perien ja lisättävä siihen viittaus projektin app.cs-tiedostosta.

```
<TextBlock Text="{Binding [PainAtRest], FallbackValue=Pain at rest}"
Style="{StaticResource DarkCaption}"/>
<toolkit:PhoneSlider  Value="{Binding DailyInput.PainAtRest, Mode=TwoWay}"
Style="{StaticResource HealthFoxSlider}" />
```

Kuva 13. Tiedon kytkeminen käyttöliittymään, Windows Phone.

Toistuvat tyylit ja esimerkiksi listaelementit on järkevää rakentaa omiksi tyyliresursseikseen (Style Resource), jolloin niiden päivittäminen käy yhdestä paikasta. Uusien mallien rakentaminen käy helposti Expression Blend-työkalulla. Muokatun käyttöliittymäelementin kohdalla klikataan hiiren oikealla näppäimellä ja valitaan Edit Template > Edit a Copy. Jos tyyliä on tarkoitus käyttää jaetusti usealla sivulla, on kohteeksi valittava Application tai Resource dictionary (kuva 14).



Kuva 14. Tyyliresurssin luominen Blend-työkalulla.

7.2 Näkymien toteutus (.cs-tiedostot)

Sivujen toteutuksessa ei ole eroa normaaliin muita kuin luokan (MvxPhonePage) osalta. Bisneslogiikka on toteutettu PCL-projektissa, joten useimmilla sivuilla .cs-tiedosto on sisällöltään minimaalinen. Alla oleva kuva antaa esimerkin näkymän toteutuksesta: tässä tapauksessa sisällöltään monimutkaisen kuntoutuskaari-sivun käyttöliittymän taustalla on vain 12 riviä koodia. Tämä on suuri MVVM-arkkitehtuurin tuoma hyöty ja helpottaa erityisesti monialustakehitystä. MvvmCross-näkymämalleja käyttävien sivujen on perittävä MvxPhonePage-luokka (kuva 15: rivi 5).

Kuva 15. Esimerkki näkymän toteutuksesta (Windows Phone).

7.3 Esimerkkidata (Sample data)

ViewModel-luokkien pohjalta on mahdollista generoida esimerkkidataa, jolla sivun elementit täytetään suunnittelunäkymässä. Tämä tapahtuu Expression Blend-sovelluksen Data-välilehdellä (Add -> From Class...). Näkymää vastaava näkymämalli valitaan esimerkkidatan pohjaksi. Tällöin Expression Blend generoi sisältöä kaikille näkymämallin julkisille tietueille ja näkymään tehdyt {Binding}-kytkökset toimivat jo suunnittelunäkymän yhteydessä (kuva 10 – huomaa automaattisesti generoidut paikkamerkkitekstit). Näillä keinoin esimerkkidata helpottaa sivujen suunnittelua ja vähentää tarvetta sovelluksen ajamiseen näkymän testaamiseksi.

Kuva 16. Esimerkkidataa käyttöliittymäsuunnittelussa.

7.4 Alustakohtaiset toiminnallisuudet

Windows Phone-projektissa kalenterimuistutukset tehdään Reminder -luokkaa hyödyntäen. Videoiden toistamiseen käytetään valmista MediaPlayerLauncher-luokkaa.

Windows Phone-sovellus on nyt esitelty. Se on toteutukseltaan Android- ja iOS-sovelluksia yksinkertaisempi, ja hyötyy suuresti MVVM-arkkitehtuurin mahdollistamasta tietojen kytkemisestä käyttöliittymään ja näkymämallin pohjalta luodusta esimerkkidatasta. Seuraavassa luvussa teen yhteenvedon projektista.

8 YHTEENVETO

Työn tavoite ja saavutukset

Projektin alussa asetettiin tavoitteeksi löytää helposti ylläpidettävä ja laajennettava arkkitehtuuri monialustakehityksen tueksi ja sen pohjalta rakentaa uudet iOS-, Android- ja Windows Phone 8-sovellukset HealthFOX-palveluun. Haasteina olivat erityisesti kehittäjien pieni lukumäärä tiimissämme ja rajalliset mahdollisuudet työn ulkoistamiselle. Työn pohjana toimi aiemmin kehitetty Windows Phone –pilottisovellus. Sovelluksen käyttäjiä ovat polvivamman kärsineet potilaat, joiden kuntoutumista pyritään tukemaan entistä paremmilla ohjeistuksilla ja kommunikaatiolla hoitohenkilökunnan kanssa.

Käyttötapausten määrittelyn jälkeen ja sovelluksen monipuolisten toimintojen asettamien rajauksien myötä kehitystyökaluksemme valikoitui Xamarin-alusta. Koodin modulaarisuutta ja jaettavuutta entisestään parantamaan valittiin Mvvm-Cross-ohjelmistokehys.

Kehitystyö oli aikaa vievä ja paljon oppimista vaatinut prosessi, mutta yksiäkään suuria haasteita ei tullut vastaan kehityksen aikana. Testiversioita tiimin sisäiseen käyttöön julkaistiin sitä myötä kun sovellusta kehitettiin ja tämän ansiosta virheet huomattiin ja korjattiin aikaisessa vaiheessa. Näitä testiversioita kertyi kullakin alustalla yli 30.

Kokemukset Xamarinista ja MvvmCross-kirjastosta

Xamarinin avulla Android- ja iOS-sovellusten kehittäminen kävi odotettua helpommin. Tiimissä oli valmistauduttu kehittämisen ulkoistamiseenkin, mutta lopulta kaikkien alustojen sovellukset tuli toteutettua kaikkine ominaisuuksineen yhden kehittäjän voimin. Eniten tutkimista vaativat tiettyjen lisäominaisuuksien kuten tyylitellyn Action Barin käyttäminen Androidilla tai iOS-sovelluksen Tab Bar-alapalkki. Myöskään kuvien näyttäminen offline-muistista ja erityisesti niiden taustalla latautumiseen reagoiminen vaativat suhteellisen paljon työtä. PCL-pro-

jektin ydintoiminnallisuudet toimivat mutkatta kaikilla alustoilla, mikä säästi varmasti valtavasti aikaa verrattuna siihen, mitä natiivisovellusten kehittämiseen olisi kulunut.

Sovelluksen kehittämisen aikana törmäsin myös ainakin kahdesti kolmannen osapuolen kirjastojen virheisiin, joiden kiertäminen aiheutti natiivisovelluksen kehittämiseen verrattuna turhaa vaivaa. Samalla tuli kuitenkin opittua myöhemmin hyödylliseksi käyneitä taitoja esimerkiksi iOS-sovelluksen käyttöliittymäelementtien muokkaamisen osalta.

MvvmCross osoittautui ohjelmistokehyksenä erittäin joustavaksi ja hyvin rakennetuksi. Sen perustoiminnallisuudet riittivät täysin projektin tarpeisiin, mutta ratkaisuja etsiessä ilmeni, että mikään toimintatapa ei siinä ole sidottu yhdenlaiseksi. Ilman MvvmCross-ohjelmistokehystä sovelluksen arkkitehtuuri ei olisi pysynyt yhtä hallittuna.

Testaus, käyttöönotto ja potilaspalaute

Sovellus otettiin käyttöön jo kehitysvaiheessa vaiheittaisen käyttöönoton avulla. Android-sovelluksen beta-testaus järjestettiin Play Storen betatestaustoiminnon avulla, ja testatut versiot siirrettiin tuotantoon normaalisti. TestFlight mahdollisti iOS-sovelluksen betatestauksen.

Kehitysversion testaamiseen osallistui 24 hengen ryhmä ACL-polvivamman kärsineitä potilaita TYKS:ssa ja Salon aluesairaalassa. Sovellus sai erittäin positiivisen vastaanoton (liite 2) ja koettiin hyödylliseksi erityisesti kuntoutuksen alkuvaiheessa, jolloin rutiinia kuntoutusliikkeiden suorittamiseen ei ole vielä syntynyt. Potilailta saadun palautteen perusteella suunniteltiin sovelluksen jatkokehitystä ja kuntoutuksen myöhemmissä vaiheessa potilasta motivoivia ominaisuuksia.

Jatkokehitys

Mobiilikehityksen osalta HealthFOX-palvelussa on käynnissä kaksi prosessia: nykyisten toimintojen parantelu ja käyttöliittymän viimeistely sekä uusien kameratoimintojen laajentaminen kaikille alustoille. Käyttöliittymän viimeistely koskee erityisesti iOS-versiota, jonka käyttöliittymän työstäminen osoittautui Android- ja

Windows Phone –versioita työläämmäksi. Uusi käyttöliittymä päivittäisten harjoitusten suunnitteluun on työn alla ja se on jo testauksessa valituilla Android- ja Windows Phone –käyttäjillä. Uusina ominaisuuksina kehitetään monipuolisempia kommunikaatiokanavia potilaan ja henkilökunnan välille. Lisäksi tutkimme uusia käyttökohteita sovellukselle.

Kameratoimintojen laajentaminen Windows Phone –alustalta iOS- ja Android-alustoille koskee erityisesti CameraFOX-sovellusta, jota fysioterapeutit käyttävät potilaiden harjoitusten kuvaamiseen ja analysoimiseen. Lisäksi kameratoimintojen tuomista HealthFOX-sovellukseen ja mahdollisiin uusiin kuntoutusympäristöihin on suunniteltu.

Sovelluksen kehitys on jatkuva prosessi, eikä nykymaailmassa sovellus ole välttämättä koskaan valmis. On luontevampaa puhua sen kulloisesta versioista, kun viitataan haluttuun hetkeen sen elinkaaressa. Opinnäytetyön valmistuessa potilaiden käytössä on HealthFOX 4.0.7 (tai versionumero 40). Se toimii kaikilla kolmella alustalla ja sisältää kaikki vaatimusmäärittelyssä kirjatut käyttötapaukset ja enemmänkin. Muun muassa tavoitteiden täyttymisen kirjaaminen ja sen myötä uusien (vaativampien) harjoitusten avautuminen lisättiin kehityksen aikana käyttötapauksiin hoitohenkilökunnan pyynnöstä ja toteutettiin viimeisten päivitysten yhteydessä.

LÄHTEET

Xamarin 2015. Mobile Application Development to Build Apps in C#. Viitattu 12.11.2015 www.xamarin.com/platform

MvvmCross 2015a. README.me. Viitattu 12.11.2015 https://github.com/MvvmCross/MvvmCross

DotNetTricks 2014. Understanding MVC, MVP and MVVM Design Patterns. Viitattu 2.11.2015 http://www.dotnet-tricks.com/Tutorial/designpatterns/2FMM060314-Understanding-MVC,-MVP-and-MVVM-Design-Patterns.html

Lehtimäki, A. 2015. Malli-näkymä-arkkitehtuurin soveltaminen WPF-teknologialla. Tampereen teknillinen yliopisto: diplomityö.

MSDN 2011. Portable Class Libraries. Viitattu 2.11.2015 https://msdn.microsoft.com/library/gg597391%28v=vs.100%29.aspx?f=255&MSPPError=-2147217396

MSDN 2015. Using Portable Class Library with Model-View-View Model. Viitattu 4.11.2015 https://msdn.microsoft.com/en-us/library/hh563947(v=vs.110).aspx#Anchor_2

Lodge, S. 2013. A first MvvmCross Application. Viitattu 4.11.2015 http://slodge.blogspot.co.uk/2013/04/n0-first-mvvmcross-application-n1-days.html

MvvmCross 2015b. Service Location and Inversion of Control. Viitattu 12.11.2015 https://github.com/MvvmCross/MvvmCross/wiki/Service-Location-and-Inversion-of-Control

StackOverflow 2013. MVVMCross Code Snippets for Visual Studio. Viitattu 5.11.2015 http://stackoverflow.com/questions/18200679/mvvmcross-code-snippets-for-visual-studio

Hubbard, D. 2014. ValueConverters. Viitattu 6.11.2015 https://github.com/MvvmCross/MvvmCross/wiki/Value-Converters

Android Developers 2015. Supporting Multiple Screens. Vittattu 3.11.2015 http://developer.android.com/guide/practices/screens_support.html

Asiakaspalautetta HealthFOX-palvelusta

KOKEMUKSIA HEALTHFOX PALVELUKONSEPTISTA

HEALTH FOX EASIER TOMORROW

HealthFOX ohjelmisto on mielenkiintoinen ja katsoo kuntoutusta eteenpäin, tulevaisuuteen, johon suuntaan henkilökohtaisesti ajatellen katson toiminnan kehittyvän.

EEVA-LEENA RASIUS
JOHTAVA LÄÄKÄRI
KUNTOUTUSKESKUS KANKAANPÄÄ

HealthFOX ohjelmakokonaisuus tuo kustannustehokkuuden lisäksi merkkitävää terveydenhuollon laatua monipolvisesti, niin palveluntuottajalle kuin sen käyttäjälle.

ARI PETÄJÄVAARA DR.(DBA), MBA

HealthFOX muistutti ja motivoi minua tekemään harjoitteet. Ohjelma katkaistiin hetkeksi, jolloin kipu ja turvotus muistutti tekemättömistä harjoitteista sänkyyn mennessä.

JONNA 24V

HealthFOX ratkaisu on mielenkiintoinen, pitkälle mietitty ja hyvin tarpeellinen.

JAN SCHUGK,
YLILÄÄKÄRI,
ELINKEINOELÄMÄN KESKUSLIITTO

Yksilöllisten harjoitteiden kuvaamisesta on ollut hyötyä, muuten pienet tärkeät huomiot olisivat puuttuneet omatoimisesta harjoittelusta.

TOMMI 25 V

Paperiharjoitteet ovat olleet aina hukassa, nyt harjoitukset olivat aina mukana.

HEIDI 26 V

En ollut käyttänyt älypuhelinta ennen testiä ja opin ilman käyttöohjekirjaa käyttämään HealthFOX sovellusta!

ERNO 35 V

Yhdessä fysioterapeutin kanssa kuvatut videot auttoivat minua keskittymään myös kotona harjoituksen ytimeen.

HEIDI 26 V

Minulle luetun ymmärtäminen on helppoa, mutta ei kaikille potilaille. HealthFOX auttaa ja helpottaa kuntoutumista.

JONNA 24 V