Joni Suntio

# TREENIPÄIVÄKIRJASOVELLUKSEN TOTEUTUS WINDOWS 10 -LAITTEILLE

Opinnäytetyö

Tietotekniikka / Peliohjelmointi

Maaliskuu 2016

KYAMK
University of Applied Sciences

| Tekijä/Tekijät | Tutkinto | Aika |
|---|---|---|
| Joni Suntio | Tietotekniikan insinööri | Maaliskuu 2016 |
| **Opinnäytetyön nimi**<br><br>Treenipäiväkirjasovelluksen toteutus Windows 10 -laitteille | | 35 sivua<br>4 liitesivua |
| **Toimeksiantaja**<br><br>GameLab | | |
| **Ohjaaja**<br><br>Laboratorioinsinööri Marko Oras | | |


**Tiivistelmä**

Opinnäytetyön tavoitteena oli luoda yksinkertainen treenipäiväkirjasovellus Windows 10 -laitteille. Tällä sovelluksella käyttäjä pystyy suunnittelemaan treeniohjelmia ja pitämään kirjaa suorituksista. Treeniohjelma pitää sisällään treeniliikkeet ja tehtävien sarjojen ja toistojen määrät. Suorituksiin käyttäjä merkitsee käytetyt painot ja voi halutessaan kirjoittaa lyhyen muistiinpanon. Sovellus suunniteltiin yksinkertaiseksi ja helppokäyttöiseksi, jotta käyttäjä tietäisi aina, miten hän voi suorittaa halutun toiminnon.

Sovellus toteutettiin UWP-sovelluksena, minkä avulla sovellus toimii lähes kaikilla Windows 10 -laitteilla. Ohjelmointikielenä käytettiin C#:ia ja merkintäkielenä XAML:ää. Toteutuksessa käytettiin MVVM-arkkitehtuurimallia, jossa koodin rakenne jaetaan kolmeen kerrokseen: malliin, näkymään ja näkymämalliin. Näkymä ja näkymämalli keskustelevat keskenään ja näkymämalli ja malli keskustelevat keskenään. Näin ohjelman eri osat saadaan eriteltyä niin, että yhden osan muokkaaminen ei vaikuta muiden osien toiminnallisuuteen.

Suunnitteluvaiheessa tehdyt tavoitteet onnistuttiin toteuttamaan ja sovelluksesta tuli vaatimusten mukainen.



**Asiasanat**
Windows 10, sovellus, ohjelmointi, XAML, UWP, C#

| Author (authors) | Degree | Time |
|---|---|---|
| Joni Suntio | Bachelor of Engineering | March 2016 |
| **Thesis Title** | | |
| Implementation of a Workout Journal Application for Windows 10 Devices | | 35 pages<br>4 pages of appendices |
| **Commissioned by** | | |
| GameLab | | |
| **Supervisor** | | |
| Marko Oras, Laboratory Engineer | | |


**Abstract**

The objective of this thesis was to create a simple workout journal application for Windows 10 devices. With this application, a user can design workout routines and keep a record of completed workouts. A workout routine includes exercises and the amount of doable sets and repetitions. In the completions, the user can record the used weight amount and, if desired, leave a small note. The application was designed to be simple and easy to use, so it is obvious to the user how to do a certain action.

The application was implemented as a UWP application, which makes it work on most Windows 10 devices. C# was used as the main programming language and XAML as the markup language. MVVM pattern was used in the implementation. In MVVM the code structure is divided into three layers: model, view and viewmodel. View and viewmodel communicate with each other, and viewmodel and model communicate with each other. With this pattern the different parts of the application are separated in such a way that editing one part will not have a negative effect on other parts' functionality.

The objectives set in the designing phase were successfully implemented, and the application met the criteria.



**Keywords**

Windows 10, application, programming, XAML, UWP, C#

SISÄLLYS

LYHENTEIDEN SELITYKSET ..... 6
1 JOHDANTO ..... 6
2 KÄYTTÖLIITTYMÄN SUUNNITTELU ..... 6
3 NÄKYMÄT ..... 8
3.1 Etusivu ..... 9
3.2 Treeniohjelmat ..... 10
3.3 Kalenteri ..... 11
3.4 Asetukset ..... 11
3.5 Lisänäkymät ..... 12
3.5.1 Treeniohjelman lisäys ..... 12
3.5.2 Treeniohjelman luonti ja muokkaus ..... 13
3.5.3 Treeniohjelman suorittaminen ..... 14
3.5.4 Treeniohjelman tarkastelu ..... 15
4 KÄYTETTÄVÄT TYÖKALUT ..... 16
4.1 Visual Studio Community 2015 ..... 16
4.2 Universal Windows Platform ..... 17
4.3 XAML ..... 17
5 TOTEUTUS ..... 18
5.1 MVVM-Malli ..... 19
5.1.1 Näkymä ..... 20
5.1.2 Näkymämalli ..... 21
5.1.2.1 Observable Collection ..... 22
5.1.2.2 ICommand ..... 23
5.1.3 Malli ..... 24
5.1.4 Poikkeamat ..... 25
5.2 Staattiset manageriluokat ..... 26
5.2.1 Workout Manager ..... 26
5.2.2 Day Manager ..... 27
5.2.3 Completion ja Exercise Manager ..... 28

5.2.4 Navigation Manager ..... 29
5.2.5 Settings Manager ..... 30
5.2.6 Save File Manager ..... 31
6 LOPPUSANAT ..... 33
LÄHTEET ..... 35
LIITTEET
Kuvankaappaukset sovelluksesta

LYHENTEIDEN SELITYKSET

HTML (Hypertext Markup Language) on merkintäkieli, mitä yleensä käytetään nettisivujen luomisessa.

MVVM (Model-view-viewmodel) on arkkitehtuurimalli.

UWP (Universal Windows Platform) on ohjelma-arkkitehtuuri. UWP:n pohjalle tehdyt ohjelmat toimivat kaikissa UWP:tä tukevissa laitteissa, kuten Windows 10:tä käyttävissä laitteissa.

XAML (Extensible Application Markup Language) on Microsoftin kehittämä XML:ään pohjautuva merkintäkieli.

XML (Extensible Markup Language) on merkintäkieli.

# 1 JOHDANTO

Opinnäytetyön tavoitteena on toteuttaa treenipäiväkirjasovellus Windows 10 -käyttöjärjestelmää käyttäville laitteille. Sovelluksen tärkeimmät ominaisuudet ovat treeniohjelmien suunnittelu ja treenipäiväkirjan pitäminen.

Idea pohjautuu tavanomaisiin treeniohjelmien paperiversioihin, joissa näkyy treenissä tehtävät liikkeet, sarjojen ja toistojen määrät, käytettävät painot ja mahdollisesti myös päivämäärät. Toteutettavan sovelluksen tulee säilyttää paperiversioiden selkeys ja helppokäyttöisyys. Ohjelman käyttö pitäisi olla uudelle ja kokeneellekin käyttäjälle helppoa ja luonnollista.

Toteutusprosessi koostuu käyttöliittymän suunnittelusta ja toiminnallisuuksien ohjelmoinnista. Ulkoasun suunnittelussa käytetään Microsoftin omia ohjeistuksia, sekä aiheeseen liittyvää kirjallisuutta.

Jotta sovellus pyörisi kaikissa Windows 10 -laitteissa, se toteutetaan UWP-sovelluksena. Ohjelmoinnissa käytettään C#- ja XAML-ohjelmointikieliä.

# 2 KÄYTTÖLIITTYMÄN SUUNNITTELU

Käyttöliittymän suunnittelun pohjana käytetään tavanomaista paperille tehtyä treeniohjelmaa (taulukko 1). Se näyttää käyttäjälle kaiken tarpeellisen tiedon, kuten mitä tehdään, kuinka paljon tehdään ja millä vastuksella tehdään. Kun tätä mallia yritetään siirtää sähköiseen muotoon, se ei välttämättä tule pysy-

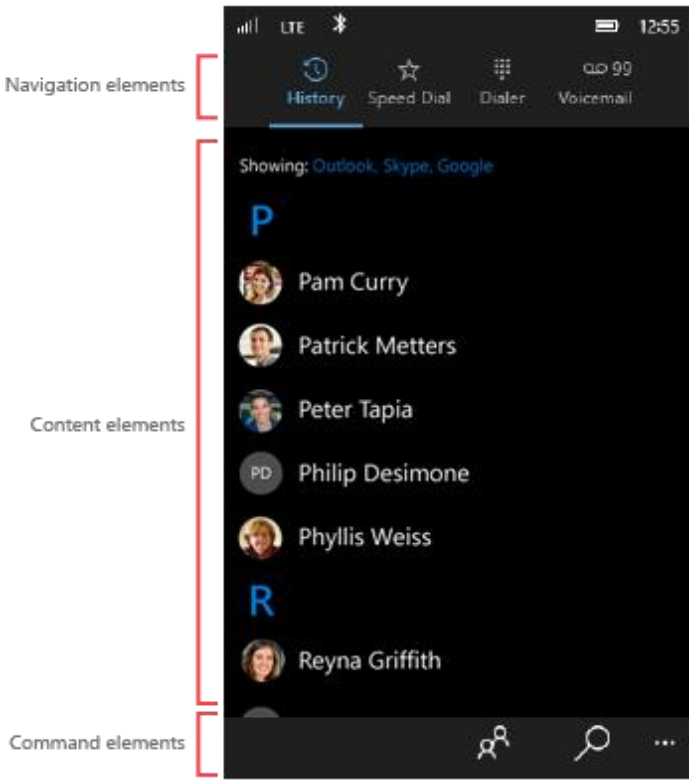
mään yhtä yksinkertaisena. Käyttäjän tulee pystyä suunnittelemaan treeniohjelmia, täyttämään niitä rajattomasti ja pitämään historiaa tehdyistä treeneistä.

Taulukko 1. Tavanomainen treeniohjelma paperilla.

| Liike | Sarjat | Toistot | Paino |
|---|---|---|---|
| Penkkipunnerrus | 3 | 12 | 60 |
| Hauis tangolla | 3 | 12 | 20 |
| Pystysoutu | 3 | 12 | 25 |

Käyttöliittymän tulee olla selkeä ja helppokäyttöinen. Steve Krugin (2014, 24) mielestä hyvä käyttöliittymä ei laita käyttäjää miettimään. Tällä hän tarkoittaa sitä, kun käyttäjää katsoo sovellusta tai nettisivua, hän ymmärtää minkälainen ohjelma on kyseessä ja kuinka sitä käytetään. Esimerkiksi tämän työn sovellusta käyttäessä, käyttäjä tietää heti, että kyseessä on treeniaiheinen sovellus. Kun käyttäjä haluaa luoda uuden treeniohjelman, hän pystyy päättelemään miten hän pääsee oikeaan näkymään ja miten haluttu toiminto toimii. Jokaisesta käytettävästä elementistä on siis tultava ilmi pelkällä vilkaisulla, mitä varten se on ja mitä se tekee.

Käytettäviä elementtejä ovat navigaatio-, sisältö- ja komentoelementit (kuva 1). Navigaatioelementit auttavat käyttäjää navigoimaan ympäri ohjelmaa. Näitä voivat olla välilehdet ja hyperlinkit. Sisältöelementit näyttävät sovelluksen sisältöä. Toteutettavassa sovelluksessa sisältöelementti voi esimerkiksi näyttää päivän treeniohjelmat. Komentoelementtien avulla käyttäjä pystyy käsittelemään sisältöä. Komentoelementtejä ovat esimerkiksi nappulat ja komentorivi. Kaikkia elementtityyppejä ei tarvitse käyttää ja elementit voivat vaihdella paljonkin riippuen sovelluksen näkymästä. (UI basics for Universal Windows Platform (UWP) apps 2015.)




Kuva 1. Sovellusten elementtityypit. (UI basics for Universal Windows Platform (UWP) apps 2015)

Elementtien sijoittelussa käytetään hyväksi länsimaista tapaa lukea vasemmalta oikealle ja ylhäältä alas. Tätä voidaan hyödyntää sijoittamalla tärkeimmät ja usein käytetyt elementit vasemmalle. Näin voidaan minimoida halutun elementin löytämiseen kuluva aika.

Toteutettavan sovelluksen värit määräytyvät käyttäjän käyttöjärjestelmän teeman mukaan, kun mahdollista. Muulloin sovelluksen väripaletti seuraa Windows 10:n sinistä teemaa. Näin sovelluksen värimaailma on yhtenäinen käyttöjärjestelmän värimaailman kanssa. Jotkin napit käyttävät teemasta poiketen muita väreillä, kuten vihreää tai punaista, korostaakseen niillä tehtävien toimintojen merkittävyyttä.

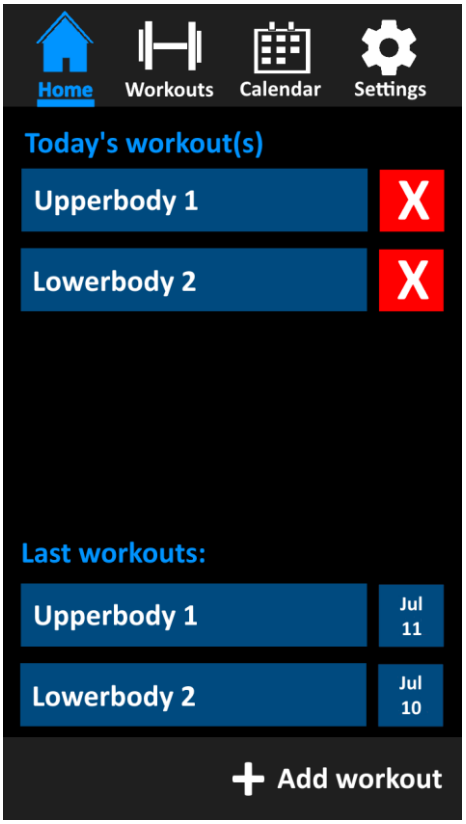
# 3 NÄKYMÄT

Näkymillä tarkoitetaan tässä yhteydessä toteutettavan sovelluksen sivuja. Näkymät on pyritty suunnittelemaan ulkoisesti yhtenäisiksi. Jotta käyttöliittymän koosta ei muodostuisi ongelmaa millään alustalla, on suunnittelu tehty ensisijaisesti mobiililaitteet mielessä pitäen. Näkymien luonnokset poikkeavat lopullisesta visuaalisesta ulkoasusta. Muutoksien määrä on pidetty minimissä. Ohjelmassa on neljä päänäkymää: etusivu, treeniohjelmat, kalenteri ja asetukset. Kaikissa näissä näkymissä on ylhäällä navigointirivi, jossa on nappulat jokai-

selle päänäkymälle. Näissä nappuloissa on tunnistamista helpottavat ikonit ja ikonin alla päänäkymän nimi. Navigaatiorivin luonnos näkyy kuvassa 2 ylhäällä. Valitun päänäkymän nappulan väri poikkeaa neutraalista valkoisesta ja nappulan alle ilmestyy pieni palkki.

## 3.1 Etusivu

Etusivu on näkymä, joka tulee ensimmäisenä käyttäjän eteen tämän avatessa sovelluksen. Se toimii myös sovelluksen vakionäkymänä. Tämän takia olisi kätevää, jos etusivu näyttäisi sisältöä, johon käyttäjä todennäköisesti haluaisi päästä käsiksi. Tässä suunnitelmassa oletettiin, että käyttäjä haluaisi nähdä päivän treeniohjelman ja kirjata tulokset ylös. Etusivu näyttää tästä syystä päivän treeniohjelmat.




Kuva 2. Etusivun luonnos.

Sisältöosuus näyttää päivän treeniohjelmat ja pari viimeisintä suoritettua treeniä. Päivän treeniohjelma näkyy nappulana, jota painamalla pääsee treenin suorittaminen -lisänäkymään. Nappulan vieressä näkyy punainen poistonappula, jonka avulla treenin voi poistaa päivän ohjelmasta. Poistotoimintojen yhteydessä käyttäjälle näytetään ponnahdusikkuna, joka kysyy haluaako käyttäjä varmasti suorittaa poiston. Suoritettujen treenien vierellä näkyy poistonap-

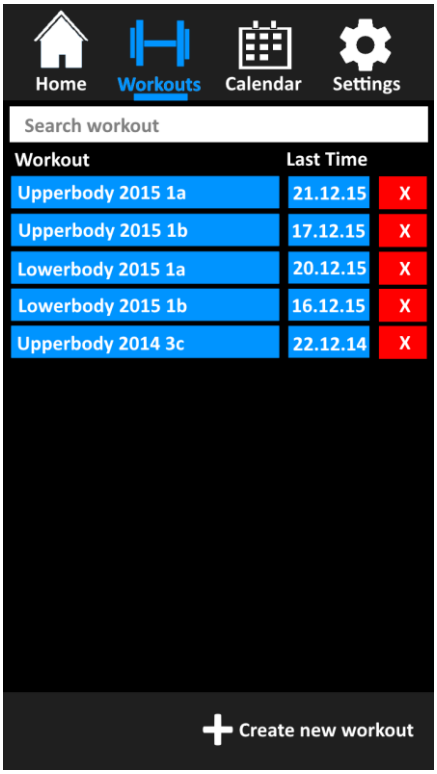
pulan sijaan päiväysnappula, josta pääsee päivän tarkastelu -lisänäkymään. Sisältöosuus on vieritettävä kuten muissakin näkymissä.

Näkymän alaosassa on yksittäinen harmaa palkki treenin lisäys–toimintoa varten. Tällä toiminnolla käyttäjä voi lisätä treenin päivän ohjelmaan. Useassa näkymässä on vastaavaa yksittäistä toiminnallisuutta varten nappula alalaidassa. Tällainen toiminnallisuus koskee yleensä lisäystä tai luontia.

## 3.2 Treeniohjelmat

Treeniohjelmat–näkymässä (kuva 3) käyttäjä voi tarkastella, luoda ja muokata treeniohjelmia. Sisältöosuus koostuu treeniohjelmista ja hakupalkista. Hakupalkkiin kirjoittaessa treeniohjelmista jää näkyviin ne, joiden nimistä löytyy hakupalkkiin kirjoitettu merkkijono. Tämä tekee halutun treeniohjelman löytämisestä helpompaa, kun treeniohjelmia on paljon.

Treeniohjelmat näkyvät lähes samalla tavalla kuin etusivulla. Treeniohjelma- ja päiväysnappuloiden toiminnot ovat samat. Poistonappula poistaa treeniohjelman kokonaan. Näkymän alalaidan palkissa on toiminto treeniohjelmien luonnille. Treeniohjelman luonnilla on oma näkymänsä, jonka tämä toiminto avaa.




Kuva 3. Treeniohjelmat-näkymän luonnos.

## 3.3 Kalenteri

Kalenteri-näkymässä (kuva 4) käyttäjä pystyy tarkastelemaan ja lisäämään treeniohjelmia päiviin. Näkymää hallitsee kalenteri, johon treeniohjelman sisältävät päivät on väritetty vihreiksi. Valittu päivä on väritetty Windows 10:n oletusteemassa siniseksi. Kalenterin alla on nappula, jota painamalla avataan valitun päivän tarkastelu -lisänäkymä. Alapalkin toimintona on sama treenin lisäys-toiminto kuin etusivulla.


Kuva 4. Kalenteri-näkymän luonnos.

## 3.4 Asetukset

Navigaatiopalkin viimeisenä on luultavasti vähiten käytettävä asetukset-näkymä (kuva 5). Ohjelmien asetus-valikot yleensä pursuavat vaihtoehtoja, mutta toteutettavassa ohjelmassa ei ole paljon ominaisuuksia mitä muuttaa. Ainut asetus on painoyksikön vaihto, jossa vaihtoehdot ovat painojen näyttäminen kilogrammoissa ja paunoissa. Alapalkissa löytyy toiminto oletusasetusten palauttamiselle. Tämä toiminto on toistaiseksi suhteellisen hyödytön, mutta jos asetusten määrä kasvaa, kasvaa myös tämän toiminnon hyödyllisyys.


Kuva 5. Asetukset-näkymän luonnos.

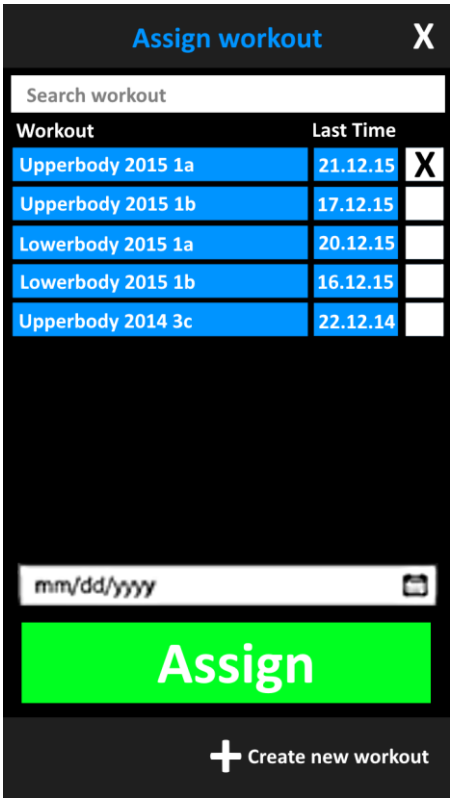
## 3.5 Lisänäkymät

Kuten joissakin päänäkymissä, lisänäkymissä voidaan tarkastella sisältöä. Sisällön muokkaus ja luonti tosin tapahtuu vain lisänäkymissä. Lisänäkymiin ei pääse navigaatiorivin kautta, eikä niissä ole navigaatioriviä. Lisänäkymiin pääsee päänäkymistä tai toisista lisänäkymistä. Esimerkiksi treeniohjelman luonti -lisänäkymään pääsee treeniohjelmat-näkymästä tai treeniohjelman lisäys -lisänäkymästä.

Navigaatiorivin tilalla näkyy lisänäkymän nimi ja oikeassa yläkulmassa lisänäkymän sulkemista varten oleva nappula. Muulla tavoin lisänäkymät eivät eroa päänäkymistä.

### 3.5.1 Treeniohjelman lisäys

Treeniohjelman lisäys -lisänäkymässä (kuva 6) liitetään yksi tai useampi treeniohjelma haluttuun päivään. Sisältöosuus muistuttaa treeniohjelmat-näkymän vastaavaa. Yläosassa on treeniohjelmien hakua helpottava hakupalkki. Hakupalkin alla treeniohjelmilla on napit treeniohjelman ja kyseisen treenin viimeisen suorituspäivän tarkasteluun. Oikealta löytyy treeniohjelman valintaruutu.

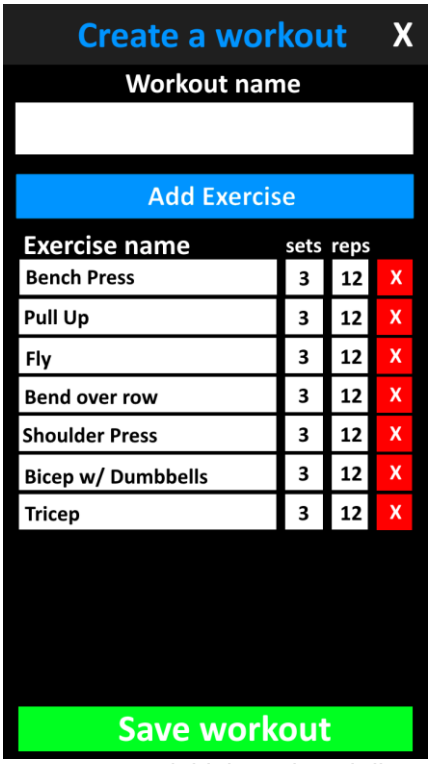



Kuva 6. Treeniohjelman lisäys –lisänäkymän luonnos.

Sisältöosuuden alaosassa on päiväyksen valinta, mitä painamalla avautuu kalenteri. Päiväyksen oletusarvo on käyttöpäivä tai valittu päivä kalenterinäkymässä, jos lisänäkymä on avattu kyseisessä näkymässä. Päiväyksen valinnan alla on vihreä hyväksymisnappi, jota painamalla valitut treeniohjelmat liitetään valittuun päivään. Alapalkissa on toiminto uuden treeniohjelman luontia varten.

### 3.5.2 Treeniohjelman luonti ja muokkaus

Treeniohjelman luonti (kuva 7) on lisänäkymä, jossa luodaan uusi treeniohjelma. Luonti-näkymän sisältöosuuden yläosassa on tekstipalkki treeniohjelman nimeämistä varten. Tämän tekstipalkin alla on nappula, jolla lisätään harjoitusliike treeniohjelmaan. Yksittäinen treeni koostuu treenin nimestä ja sarjojen ja toistojen määrästä. Näiden muokkausta varten löytyvät muokattavat tekstiruudut. Treenien oikealta puolelta löytyy poistonappulat. Alalaidassa on nappula treeniohjelman tallentamista varten.



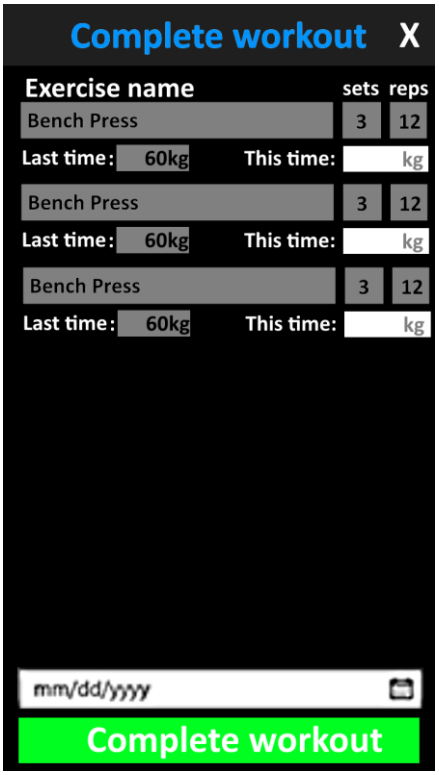

Kuva 7. Treeniohjelman luonti -lisänäkymä.

Treeniohjelman muokkaus -lisänäkymässä muokataan jo olemassa olevia treeniohjelmia. Näkymältään treeniohjelmien muokkaus muistuttaa luontia. Treenien poisto- ja lisäysnapit on poistettu, koska treeniohjelman on pysyttävä yhtenäisenä suoritushistorian kanssa. Koko treeniohjelman poistolle on lisätty nappula.

### 3.5.3 Treeniohjelman suorittaminen

Treeniohjelman suorittaminen -lisänäkymässä (kuva 8) lisätään treeniohjelman suoritus. Tätä näkymää käytetään eniten treenaamisen yhteydessä tai pian treenin jälkeen. Tämän takia on näkymässä oltava selvää, mitä tehdään, kuinka paljon tehdään ja mihin voi täyttää suorituksen arvot. Sisältöosuus koostuu lähinnä tehtävistä liikkeistä, joista näytetään nimi ja sarjojen ja toistojen määrät. Näiden alla näkyy edellisellä suorituskerralla käytetty paino kyseisessä liikkeessä. Tämän jälkeen on muokattava tekstiruutu uudella suorituskerralla käytettävää painoa varten.

Näkymän alaosassa on päiväyksen valinta, jonka vakioarvona on käyttöpäivä. Päiväyksen valinnan alla on iso vihreä hyväksymisnappula treeniohjelman suorituksen hyväksymistä varten.



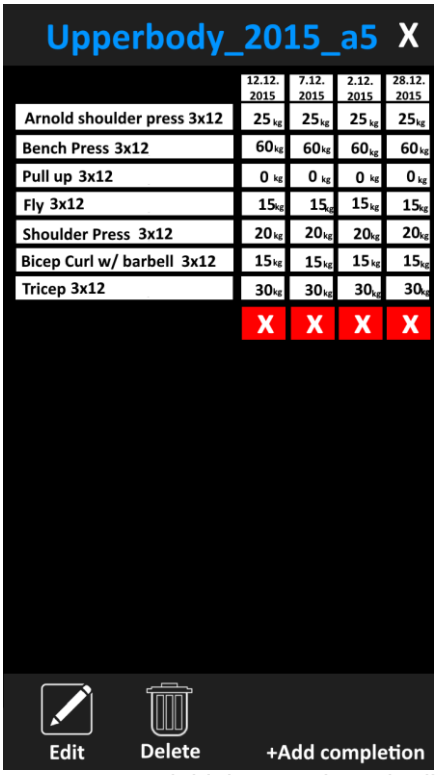

Kuva 8. Treeniohjelman suorittaminen -lisänäkymän luonnos.

### 3.5.4 Treeniohjelman tarkastelu

Treeniohjelman tarkastelu -lisänäkymässä (kuva 9) käyttäjä voi tarkastella treeniohjelman sisältöä. Sisältöön kuuluu treeniliikkeet sarjoineen ja toistoineen ja treeniohjelman suoritushistoria. Koska suorituskertoja voi kertyä huomattava määrä, näytettävän tiedon määrä voi olla suuri. Tästä syystä kaikki näytettävä tieto pitää tiivistää mahdollisimman pieneen tilaan säilyttäen silti tiedon helppolukuisuuden.

Sisältöosuus koostuu taulukosta, josta kaikki treeniohjelman tiedot löytyvät. Ensimmäisessä sarakkeessa on treeniliikkeiden nimet, joihin on lisätty perään sarjojen ja toistojen määrät. Toisesta sarakkeesta alkavat suorituskertojen merkinnät. Suorituskerran ylimmällä rivillä on suorituspäivä ja alimmalla suorituskerran poistonappi. Näiden välissä näkyy suorituskerralla käytetyt painot kyseisten rivien treeniliikkeissä. Suorituskerroille varattu tila on rullattava.

Alapalkissa on nappulat treeniohjelman muokkaamiselle ja poistamiselle ja toiminto treeniohjelman suorituksen lisäämistä varten.




Kuva 9. Treeniohjelman tarkastelu -lisänäkymän luonnos.

# 4 KÄYTETTÄVÄT TYÖKALUT

Koska sovellus tehdään Microsoftin Windows 10 -käyttöjärjestelmälle, on sovelluksen toteutuksessa käytetty lähinnä Microsoftin omia työkaluja.

## 4.1 Visual Studio Community 2015

Visual Studio Community 2015 on viimeisin ilmainen versio Microsoftin Visual Studio -ohjelmointiympäristöstä. Visual Studiolla voi luoda tietokone-, mobiili- ja web-sovelluksia. Ohjelmistosta löytyy työkalut lähdekoodin editoimiseen, sovelluksien testaamiseen ja käyttöliittymien suunnitteluun. Visual Studio tukee sellaisenaan Microsoftin ohjelmistonkehitysalustoja ja ohjelmointikielistä C:tä, C++:aa, C#:a, Visual Basiciä ja F#:a. Visual Studio on helposti laajennettavissa ladattavien lisäosien avulla, joiden avulla ohjelmiston voi saada tukemaan esimerkiksi muita ohjelmointikieliä.

Sovelluksen toteutuksessa on käytetty Visual Studio Community 2015:n koodieditoria ja käyttöliittymän suunnittelunäkymää. UWP-sovelluksen luontia varten asennettiin Universal Windows App -kehitystyökalut, jotka saatiin ladattua Visual Studion kautta suoraan.

## 4.2 Universal Windows Platform

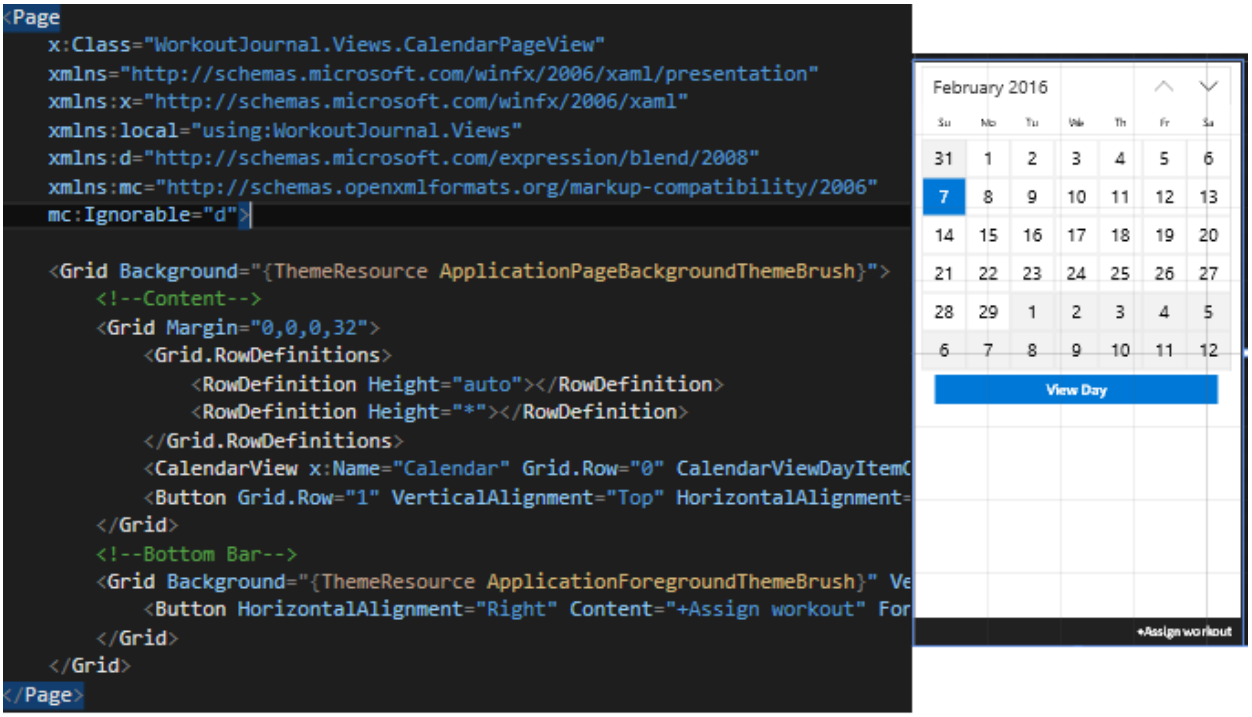
Universal Windows Platform, tai lyhyesti UWP, on Microsoftin kehittämä ohjelmistoarkkitehtuuri, jonka tarkoituksena on taata sovelluksen toimivuus kaikissa Windows 10 -käyttöjärjestelmää käyttävissä laitteissa. UWP on kehitetty Windows 8:lle suunnitellusta Windows Runtimesta, jonka tarkoitus oli tehdä mahdolliseksi sovelluksen kehitys samaan aikaan tietokoneille ja mobiililaitteille. UWP pystyy kutsumaan Windows Runtimen ohjelmointirajapintoja, jotka ovat samat kaikilla Windows 10 -laitteilla. Tämän lisäksi se pystyy kutsumaan laitekohtaisia ohjelmointirajapintoja, kuten Win32- ja .NET-rajapintoja. Käytännössä tämä tarkoittaa, että sovelluksen voi rajoittaa toimimaan vain tietyissä laitteissa. (Guide to Universal Windows Platform (UWP) apps 2016.)

UWP tarjoaa yhtenäiset alustariippumattomat perustarpeet sovelluksen luonnille. Universaalit kontrollit toimivat alustalla kuin alustalla, koska niiden toiminnallisuudet pysyvät samoina. Näitä ovat kaikki yleisimmät lomakekontrollit, kuten tavalliset nappulat, radionappulat ja tekstilaatikot. Universaalien tyylien ansiota ohjelma näyttää samalta kaikilla alustoilla. Nämä tyylit tarjoavat myös perusanimaatioita käyttäjän eri vuorovaikutuksia varten, kuten nappuloiden kutistuminen ja suureneminen painalluksen mukaan. (Introduction to Universal Windows Platform (UWP) apps for designers 2016.)

UWP-sovelluksen luontia varten tarvitaan Windows 10 -käyttöjärjestelmä, joka on kehittäjätilassa, ja Visual Studio 2015, jossa on asennettuna Universal Windows App -kehitystyökalut. Toiminnallisuuksien ohjelmoinnissa voi käyttää C#:a, Visual Basiciä, C++:aa tai JavaScriptiä. Ulkoasun määrittelyssä voi käyttää riippuen ohjelmointikielestä joko XAML:ää tai HTML:ää. Toteutetussa sovelluksessa käytettiin C#:a ja XAML:ää. (Develop apps for the Universal Windows Platform (UWP) 2016.)

## 4.3 XAML

XAML, eli Extensible Application Markup Language, on Microsoftin kehittämä XML:ään pohjautuva merkintäkieli. XAML:n avulla määritellään sovelluksen ulkoasu samalla tavalla kuin HTML:n avulla määritellään nettisivuston ulkoasu. Koodi muistuttaakin paljon HTML:ää. (What is XML? 2016.)




Kuva 10. Näkymän xaml-koodi ja esikatselu Visual Studiossa.

XAML:n tiedostopääte on .xaml. Tiedostossa (kuva 10 vasemmalla) aloitustunnisteessa määritellään sivun tyyppi. Aloitustunnisteen sisällä on määritelty näkymän luokan nimi ja käytetyt nimiavaruudet. Aloitustunnisteen sisäiselle tasolle voi määritellä vain yhden elementin. Tämä yksi elementti toimii pohjana koko sivun ulkoasun määrittelylle.

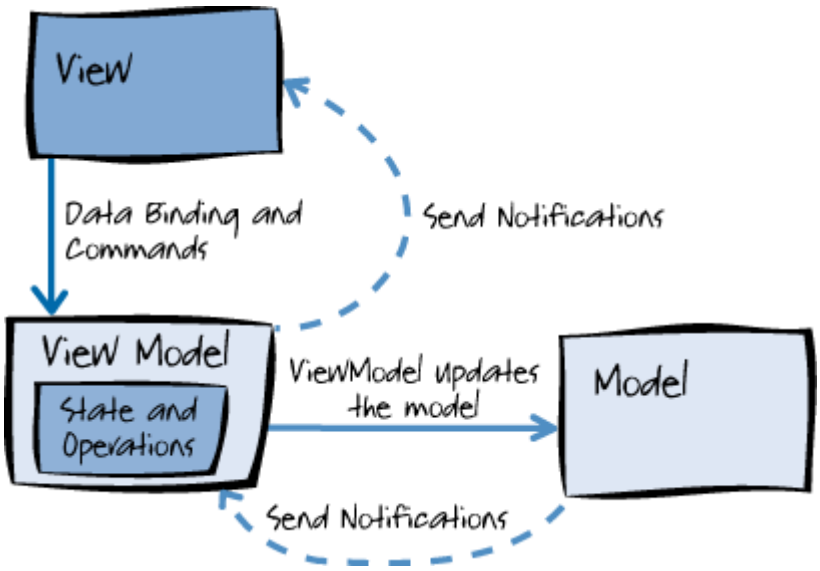
Jokainen elementti on hyvin muokattavissa. Muokattaviin ominaisuuksiin kuuluu muun muassa taustaväri, tekstin väri, leveys, korkeus ja sijainti isäntäelementtiin nähden. Toteutuksessa kaikkein usein käytetty elementti on Grid, eli ruudukko. Ruudukon avulla pystyy helposti säätämään muiden elementtien sijainteja. Tämä on helpointa tehdä määrittelemällä ruudukkoon rivejä ja sarakkeita ja määritellä niiden korkeudet ja leveydet. Tämän jälkeen ruudukon sisäiset elementit voi sijoittaa haluamalle riville ja/tai sarakkeelle laittamalla elementin tunnisteeseen haluttu sijainti. Esimerkiksi jos ruudukkoon on määritelty kaksi riviä ja elementti halutaan sijoittaa toiselle riville, laitetaan elementin tunnisteen Grid.Row=”2”.

# 5 TOTEUTUS

Sovellus toteutettiin lähes kokonaan käyttäen MVVM-mallia. Joidenkin UWP-kontrollien rajallisuus esti MVVM-mallin käytön parissa näkymässä.

## 5.1 MVVM-Malli

MVVM, eli model-view-viewmodel, on arkkitehtuurimalli (kuva 11). Sen tarkoituksena on tarjota selvä ero käyttöliittymän kontrollien ja näiden logiikan välillä. MVVM-mallissa on kolme ydinkomponenttia; malli, näkymä ja näkymämalli. Jokaisella näistä on erottuva ja toisistaan erilainen rooli.




Kuva 11. MVVM-malli. (The MVVM Pattern 2016)

Komponentit ovat toisistaan irrallaan, mikä mahdollistaa niiden vaihtamisen ja muokkaamisen rikkomatta muita komponentteja. Koska komponentit toimivat itsenäisesti, niitä voidaan testata erillään muista. On tärkeää tietää jokaisen komponentit vastuut ja kuinka ne vuorovaikuttavat toistensa kanssa. Näkymä tietää näkymämallin olemassaolosta, näkymämalli tietää mallin olemassaolosta ja malli tietää vain oman olemassaolonsa. Näkymämalli erottaa näkymän malliluokista ja antaa mallin toimia itsenäisesti erillään näkymästä. (The MVVM Pattern 2016.)

MVVM-mallin heikkouksia on mallin kehittäneen John Gossmanin (2006) mielestä muutamia. Hyvin yksinkertaista ohjelmaa varten mallin käyttö voi olla ylilyöntiä. Isommissa ohjelmissa taas voi olla hankalaa suunnitella näkymämallit sopiviksi useampaa näkymää varten. Näkymien ja näkymämallien välinen datasidonnan suorituskyky on varsin hyvä, mutta se luo jonkin verran kirjanpitotietoa. Jos yhteen objektiin liittää useamman datasidoksen, näiden sidosten käyttämän muistin määrä saattaa hyvinkin kasvaa tarpeettoman suureksi. Normaalikokoisessa ohjelmassa datasidosten määrä ei pitäisi olla kovin suuri, mutta niihin on hyvä kiinnittää suorituskyvyn kannalta huomiota.

### 5.1.1 Näkymä

Näkymä määrittelee kaiken käyttäjän näkemän rakenteen, asettelun ja ulkoasun. Ihanteellisessa tilanteessa näkymä määritellään käyttäen lähinnä XAML:ää ja näkymää tukeva koodi ei sisällä business-logiikkaa. Näkymällä voi olla oma näkymämalli tai se voi periä isäntänäkymän näkymämallin. Näkymä saa tietonsa näkymämallista datasidosten avulla tai kutsumalla näkymämallin metodeja. Sovelluksen ajon aikana näkymä muuttuu, kun sen kontrollit vastaavat näkymämallien ominaisuuksien nostamiin muutosilmoituksiin. (The MVVM Pattern 2016.)

```
<Page
    x:Class="WorkoutJournal.Views.WorkoutsView"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="using:WorkoutJournal.Views"
    xmlns:viewmodels="using:WorkoutJournal.ViewModels"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d">

    <Page.DataContext>
        <viewmodels:WorkoutsPageViewModel />
    </Page.DataContext>
```

Kuva 12. Näkymämallin liittäminen näkymään.

Jotta näkymämallin voi liittää näkymään, pitää ensin määritellä näkymämallin sijainti. Esimerkissä (kuva 12) luodaan linkki näkymämallien sijaintiin kirjoittamalla sivun määrittelyyn esimerkiksi xmlns:viewmodels=”using:WorkoutJournal.ViewModels”. Xmlns on nimiavaruus, johon elementtien nimet määritellään. Tässä tapauksessa määritellään nimi ”vievmodels”, johon liitetään näkymämallien sijainti.

Varsinainen näkymämallin liittäminen tapahtuu laittamalla haluttu näkymämalli Sivun tietokontekstiin. Esimerkissä (kuva 12) tunniste Page.DataContext tarkoittaa sivun tietokontekstia. Tunnisteen sisällä kutsutaan haluttua näkymämallia.

```
<ListView Grid.Row="1" x:Name="lv" SelectionMode="None" ItemsSource="{Binding Workouts}"
```

Kuva 13. Pätkä Listanäkymän määrittelystä.

Jotta näkymämallin tietoja voidaan näkymässä esittää, pitää kontrollien ja näkymämalliin ominaisuuksien välille luoda datasidos. Tämä tehdään laittamalla kontrollin tunnisteeseen sisällönlähteeksi esimerkiksi {”Binding Workouts”}

(kuva 13). Binding viittaa datasidokseen tietokontekstissa määriteltyyn näkymämalliin ja tämän jälkeen tulee haluttu näkymämallin ominaisuus.

### 5.1.2 Näkymämalli

Näkymämalli toimii näkymän ja mallin välillä ja on vastuussa näkymän logiikan käsittelyssä. Tyypillisesti näkymämalli on vuorovaikutuksessa mallin kanssa kutsumalla malliluokkien metodeja. Näkymämalli hakee tietoja mallilta ja muuttaa tarvittaessa ne sellaiseen muotoon, että näkymä voi ne esittää. näkymämalli myös tarjoaa toteutuksia komennoista, joita käyttäjä kutsuu näkymästä. Esimerkiksi kun käyttäjä painaa käyttöliittymän jotain nappia, se voi kutsua näkymämallissa sijaitsevaa komentoa. (The MVVM Pattern 2016.)

Jotta näkymämallissa tapahtuvat muutokset päivittyvät tarvittaessa näkymässä, pitää sen ominaisuuksien kutsua PropertyChanged-tapahtumaa. Tämän toteuttamiseksi näkymämallien luokat perivät INotifyPropertyChanged-rajapinnan. Rajapinnan toteuttamiseksi riittää PropertyChangedEventHandler-tyyppisen ja PropertyChanged-nimisen tapahtuman toteutus. Tätä tapahtumaa kutsuttaessa sen kuuntelijat, joita tämän sovelluksen kohdalla useimmiten ovat näkymän kontrollit, voivat toimia tarpeen mukaan.

```
public class BaseViewModel : INotifyPropertyChanged
{
    public event PropertyChangedEventHandler PropertyChanged;

    protected void RaisePropertyChanged(string PropertyName)
    {
        var handler = this.PropertyChanged;
        if(handler != null)
        {
            handler(this, new PropertyChangedEventArgs(PropertyName));
        }
    }
}
```

Kuva 14. Näkymämallien kantaluokka.

Sovelluksen toteutuksessa määriteltiin BaseViewModel-kantaluokka, jonka kaikki näkymämallit perivät (kuva 14). Tähän luokkaan on toteutettu INotifyPropertyChanged-rajapinta. PropertyChanged-tapahtuman kutsumista helpottamaan luotiin RaisePropertyChanged-metodi, joka ottaa parametrina muuttuvan ominaisuuden nimen merkkijonona. Metodia kutsuttaessa tarkistetaan onko kyseisen näkymämallin ominaisuuksilla kuuntelijoita. Jos kuuntelijoita löytyy, ilmoitetaan tämän muuttuneen ominaisuuden kuuntelijoille muutoksesta.

```
Exercise _exercise = new Exercise();

public Exercise Exercise
{
    get { return _exercise; }
    set { _exercise = value;
        RaisePropertyChanged("Exercise");
    }
}
```

Kuva 15. Harjoitus-muuttuja ja –ominaisuus.

RaisePropertyChanged-metodia kutsutaan aina ominaisuuden arvon vaihdon yhteydessä (kuva 15). Tiedon muutoksesta saatuaan kuuntelijat hakevat päivittyneen tiedon ominaisuudesta.

#### 5.1.2.1 ObservableCollection

Jos halutaan näyttää jonkinlaista tietokokoelmaa, kuten esimerkiksi listaa kaikista treeniohjelmista, pitää listan ilmoittaa mahdollisista muutoksista. Tätä varten sovelluksessa käytettiin tietokokoelmien näyttämisessä ObservableCollection-kokoelmatyyppiä. ObservableCollection perii INotifyPropertyChanged-rajapinnan ja kutsuu PropertyChanged tapahtumaa aina, kun kokoelmaa muokataan.

```
private ObservableCollection<ExerciseViewModel> _exercises = new ObservableCollection<ExerciseViewModel>();
public ObservableCollection<ExerciseViewModel> Exercises
{
    get { return _exercises; }
    set {_exercises = value;
        RaisePropertyChanged("Exercises");
    }
}
```

Kuva 16. Harjoitusliikkeistä koostuva ObservableCollection-muuttuja ja –ominaisuus.

ObservableCollection määritellään kuin mikä tahansa muukin muuttuja tai ominaisuus (kuva 16). Sovelluksessa ObservableCollection-kokoelmat olivat yleensä tarvittavan näkymämallin tyyppisiä. Tämän avulla pystyttiin määrittelemään mitä yksittäisen kokoelman objektista haluttiin näyttää. Esimerkiksi treeniliikkeen näkymämallissa on muuttuja treeniliikkeen malliluokalle ja ominaisuudet tämän malliluokan ominaisuuksiin, joilla taas pääsee käsiksi malliluokan instanssin muuttujiin. ObservabeCollectionia pystyy näkymässä esittämään joko ListView- tai BoxView-elementeissä.

#### 5.1.2.2 ICommand

MVVM-mallissa näkymän elementteihin ei voi suoraan liittää näkymämallissa olevia metodeja. Elementteihin sidottujen toimintojen täytyy olla tyyppiä, joka on perinyt ICommand-rajanpinnan. ICommandin toteuttaminen vaatii CanExecute ja Execute nimiset metodit, joilla on object-tyyppinen parametri, ja CanExecuteChanged nimisen tapahtuman. CanExecute tarkistaa toimintoa kutsuttaessa, voiko toiminnon suorittaa. Jos voi, kutsutaan Execute-metodia.

Tätä varten luotiin kaksi luokkaa, CustomCommand (kuva 17) ja CustomCommand<T>, joista jälkimmäinen luotiin parametrillisia metodeja varten. Käytettävää metodia varten luotiin Action-tyyppinen _action-muuttuja, johon metodi lisätään delegaattina CustomCommand-instanssia luodessa. CanExecute palauttaa aina arvon tosi, koska CustomCommandia käytettiin kaikissa toiminnoissa, eikä löytynyt tilannetta, jossa epätosi-arvolle olisi ollut käyttöä. Execute-metodissa tarkistetaan, että delegaatti on lisätty onnistuneesti ja haluttu metodi suoritetaan.

```
public class CustomCommand : ICommand
{
    public event EventHandler CanExecuteChanged;
    Action _action;

    public CustomCommand(Action action)
    {
        _action += action;
    }

    public bool CanExecute(object parameter)
    {
        return true;
    }

    public void Execute(object parameter)
    {
        if(_action != null)
        {
            _action();
        }
    }
}
```

Kuva 17. CustomCommand-luokka, joka perii ICommand-rajapinnan.

Toiminto toteutetaan näkymämallissa (kuva 18). Näkymään liittämistä varten luodaan ICommand-tyyppinen ominaisuus, jota haettaessa luodaan uusi CustomCommand-instanssi, joka ottaa parametrina halutun metodin delegaattina.

```
private void AddExerciseExecute()
{
    Exercises.Add(new ExerciseViewModel());
}

public ICommand AddExercise { get { return new CustomCommand(AddExerciseExecute); } }
```

Kuva 18. Komennon toteutus näkymämallissa.

### 5.1.3 Malli

MVVM-mallissa malli on kerros, joka käsittelee ja tallentaa tietoja. Sovelluksessa tämä kerros toteutettiin luomalla tietoja varten omat luokat ja näiden käsittelyä varten manageriluokat. Tietoja tässä tapauksessa ovat treeniohjelmat, harjoitusliikkeet, suoritukset ja päiväykset. Treeniohjelman malliluokka sisältää muuttujat nimelle, yksilölliselle tunnusluvulle ja kokoelmamuuttujat harjoitusliikkeille ja suorituksille. Treeniohjelman suorituksen lisäämiselle ja viimeisimmän suorituksen haulle on tehty apumetodit. Harjoitusliikkeen malliluokka (kuva 19) sisältää muuttujat liikkeen nimelle ja sarjojen ja toistojen määrille. Päiväyksen malliluokka sisältää muuttujan itse päiväykselle ja kokoelmamuuttujat kyseiselle päivälle suunnitetuille ja suoritetuille treeniohjelmille.

```
public class Exercise
{
    string _name = "";
    int _sets = 0;
    int _reps = 0;

    public string Name
    {
        set { _name = value; }
        get { return _name; }
    }

    public int Sets
    {
        set { _sets = value; }
        get { return _sets; }
    }

    public int Reps
    {
        set { _reps = value; }
        get { return _reps; }
    }
}
```

Kuva 19. Harjoitusliikkeen malliluokka.

Suoritukset on jaettu kahteen malliluokkaan. Ensimmäinen on treeniohjelman suorituksen malliluokka. Tämä luokka sisältää muuttujan suorituspäivälle ja kokoelmamuuttujan harjoitusliikkeiden suorituksille. Harjoitusliikkeen suorituk-

sen malliluokka sisältää muuttujat käytetylle painolle, liikkeen suorituksen varten tehtävälle muistiinpanolle ja apumuuttuja käytetyn painon yksikkömuunnokselle.

Muuttujien lisäksi luokissa löytyy ominaisuudet, joilla pääsee käsiksi muuttujiin. Kaikilla malliluokilla on vastaava näkymämalliluokka. Tällaisessa näkymämalliluokassa on yleensä muuttuja malliluokalle ja ominaisuudet itse malliluokalle ja malliluokan ominaisuuksille.

### 5.1.4 Poikkeamat

MVVM-mallia pyrittiin käyttämään aina kuin mahdollista. Kalenteri-näkymän kohdalla MVVM-mallia ei voinut käyttää CalenderView-elementin rajoitteiden takia. Suunnitelmissa haluttiin, että kalenterissa päivät, jotka sisältävät treeniohjelman väritetään vihreiksi. Ongelmaksi muodostui, ettei näkymämallissa päästy käsiksi CalenderView-elementin yksittäisiin päiväobjekteihin. Tämä esti niiden muokkaamisen näkymämallissa kokonaan. Toinen ongelma oli, ettei valitun päivän vaihtamisen tapahtumaan saatu liitettyä ICommand-tyyppistä komentoa. Tämä tapahtuma vaatii metodin, joka ottaa parametreiksi CalendarView-tyyppisen lähettäjän ja CalendarViewSelectedDatesChangedEventArgs-tyyppisen tapahtumaviestin.

MVVM-mallista poiketen nämä piti toteuttaa näkymään liitettyyn luokkaan. Tämä luokka on liitetty näkymään vakiona ja muokkaamattomana se vain alustaa näkymän komponentit. Tähän luokkaan luotiin tarvittavat metodit. Näiden metodien liittäminen näkymässä halutun elementin tapahtumiin eroaa näkymämallin ICommand-tyyppisten komentojen liittämisellä siten, että näkymän luokan metodit ei tarvitse datasidosta, vaan liittäminen tapahtuu suoraan (kuva 20). Metodien ei myöskään tarvitse olla julkisia.

```
CalendarViewDayItemChanging="Calendar_CalendarViewDayItemChanging"
```

Kuva 20. Näkymän luokassa sijaitsevan metodin lisääminen CalendarView-elementin tapahtumaan.

Toinen poikkeus on sovelluksen pääikkuna, johon kuuluu navigointirivi ja Frame-elementti, johon päänäkymät ladataan. Koska näkymän elementteihin ei pääse käsiksi näkymämallista, pääikkunan toiminnallisuudet on tehty näkymään liitettyyn luokkaan. Näitä toiminnallisuuksia ovat ikkunan Frame-

elementin tallentaminen NavigationManager-luokkaan ja navigaatiorivin nappuloiden käsittely.

## 5.2 Staattiset manageriluokat

Manageriluokalla viitataan luokkaan, joka yleensä käsittelee tietynlaista tietoa. Näihin luokkiin pääsee yleensä käsiksi mistä päin koodia tahansa. Jotta tämä on mahdollista, manageriluokasta tehdään joko singleton-luokka tai staattinen luokka. Singleton tarkoittaa, että luokasta luodaan vain yksi staattinen instanssi.

Sovelluksessa manageriluokista tehtiin staattisia. Tämä tarkoittaa sitä, että luokka itsessään ja sen muuttujat, metodit ja ominaisuudet ovat staattisia. Staattista luokkaa ei voi luoda muodostimella.

### 5.2.1 Workout Manager

WorkoutManager on staattinen luokka, joka käsittelee treeniohjelmia. Se säilyttää ajon aikana treeniohjelmat Dictionary-kokoelmamuuttujaan. Tämän muuttujan arvoina ovat treeniohjelmat ja avaimina treeniohjelmien tunnusluvut. Esimerkiksi päiväykset eivät sisällä treeniohjelman instanssia, vaan niihin on liitetty treeniohjelman tunnusluku, jonka avulla ne voivat hakea luokan instanssia WorkoutManagerista.

WorkoutManager pitää yllä myös valmista näkymämallien kokoelmaa, jonka näkymämallit saavat nopeasti haettua ilman, että sellainen pitäisi aina luoda erikseen. Tämä kokoelma päivitetään aina kun treeniohjelmien kokoelmaa muokataan jollain tavalla. Näkymämallien kokoelman muuttuessa kutsutaan WorkoutManagerin ListChanged-toimintaa. Tähän toimintaan voivat luokat lisätä metodin delegaattina, jos ne haluavat jotain suoritettavan toimintaa kutsuttaessa.

WorkoutManagerista löytyy myös kokoelma valituille treeniohjelmille. Tämä kokoelma ei sisällä treeniohjelmien instansseja, vaan treeniohjelmien tunnuslukuja. Tätä kokoelmaa käytetään apuna, kun halutaan lisätä treeniohjelmia vaikka päiväyksiin. Kyseistä tapausta varten on tehty apumetodi AddSelectedWorkoutsToADay, joka ottaa parametrina päiväyksen DateTimeOffset-tyyppisenä. Metodi lisää valittujen treeniohjelmien tunnusluvut parametrina annettuun päivään.

Viimeisenä WorkoutManagerista löytyy merkkijonomuuttuja väliaikaiseen treeniohjelman tunnusluvun säilytykseen. Tähän muuttujaan voidaan tallentaa haluttu tunnusluku, johon voidaan viitata vielä näkymän vaihdoksen jälkeenkin. Tämä todettiin tarpeelliseksi, kun näkymille ei voinut luoda muodostinta, johon olisi voinut laittaa parametrina haluttuja arvoja näkymän vaihdon yhteydessä.

### 5.2.2 Day Manager

DayManager on staattinen luokka, joka käsittelee päiväyksiä. Se säilyttää päiväyksiä Dictionary-tyyppisessä kokoelmassa, jossa arvoina ovat päivä-luokan instanssit ja avaimina ovat päiväykset DateTime-tyyppisinä. Koska päiväykset näyttävät vain päivälle suunnitellut ja suoritetut treeniohjelmat, ei ole tarvetta luoda päivien näkymämalleille kokoelmaa. Sen sijaan suunniteltujen ja suoritettujen treeniohjelmien lisäämiselle, poistamiselle ja näkymämallikokoelmien luonnille (kuva 21) on luotu metodit.

```
public static ObservableCollection<WorkoutViewModel> GetCompletedWorkoutsForDay(DateTimeOffset date)
{
    ObservableCollection<WorkoutViewModel> workouts = new ObservableCollection<WorkoutViewModel>();
    if (Days.ContainsKey(date.Date))
    {
        List<string> removableIds = new List<string>();
        if (Days[date.Date].CompletedWorkoutsID.Count > 0)
        {
            for (int j = 0; j < Days[date.Date].CompletedWorkoutsID.Count; j++)
            {
                WorkoutViewModel newWorkout = new WorkoutViewModel();
                if (WorkoutManager.Workouts.ContainsKey(Days[date.Date].CompletedWorkoutsID[j]))
                {
                    newWorkout.Workout = WorkoutManager.Workouts[Days[date.Date].CompletedWorkoutsID[j]];
                }
                else
                {
                    removableIds.Add(Days[date.Date].CompletedWorkoutsID[j]);
                    continue;
                }
                newWorkout.Date = date.Date;
                workouts.Add(newWorkout);
            }
        }
        //remove workoutIDs that belong to no workout
        if (removableIds.Count > 0)
        {
            for (int i = 0; i < removableIds.Count; i++)
            {
                Days[date.Date].CompletedWorkoutsID.Remove(removableIds[i]);
            }
        }
    }
    return workouts;
}
```

Kuva 21. Metodi suoritettujen treeniohjelmien näkymämallikokoelman luonnille.

DayManagerista löytyy DateTimeOffset-tyyppinen _tempDay-muuttuja, johon voi tallentaa väliaikaisesti halutun päivämäärän. Muuttujaan pääsee käsiksi TempDay-ominaisuuden avulla. Tätä muuttujaa käytetään lähinnä halutun päivän hakuun uuden näkymän alustuksessa (kuva 22).

```
public ViewDayViewModel()
{
    PlannedWorkouts = DayManager.GetPlannedWorkoutsForDay(DayManager.TempDay);
    CompletedWorkouts = DayManager.GetCompletedWorkoutsForDay(DayManager.TempDay);
    DayManager.ListChanged_Removed += OnWorkoutsChanged_Removed;
}
```

Kuva 22. Päivän näkymämallin muodostin. Huomaa DayManagerin TempDay-ominaisuuden käyttö.

### 5.2.3 Completion ja Exercise Manager

Completion- ja ExerciseManager ovat staattisia luokkia, joiden tarkoituksena on auttaa suoritusten ja harjoitusten poistamisessa ja lisäämisessä kokoelmiin ja ilmoittaa tehdyistä muutoksista. Jos harjoituksia ja toimintoja varten tarvitsisi lisää toimintoja, toteutettaisiin ne näihin luokkiin.

```
public static class CompletionManager
{
    public static Action<WorkoutCompletionViewModel> CompletionAdded;
    public static Action<WorkoutCompletionViewModel> CompletionRemoved;

    public static void RaiseCompletionAdded(WorkoutCompletionViewModel wcvm)
    {
        if(CompletionAdded != null)
        {
            CompletionAdded(wcvm);
        }
    }

    public static void RaiseCompletionRemoved(WorkoutCompletionViewModel wcvm)
    {
        if (CompletionRemoved != null)
        {
            CompletionRemoved(wcvm);
        }
    }
}
```

Kuva 23. CompletionManager-luokka.

Nämä luokat tehtiin, koska sovellusta ajettaessa näkymän kokoelmissa yksittäiset objektit eivät tiedä isäntäkokoelmistaan mitään. Tämä tuottaa ongelman, koska ei tiedetä mistä kokoelmasta objekti halutaan poistaa tarvittaessa. Tätä varten esimerkiksi CompletionManager-luokkaan (kuva 23) luotiin poistamiselle CompletionRemoved-niminen toiminto, johon voi liittää WorkoutCompletionViewModel-tyyppisen parametrin omaavan metodin delegaattina. Toiminnan kutsumista varten tehtiin RaiseCompletionRemoved-metodi, joka ottaa parametrinä WorkoutCompletionViewModelin instanssin. Metodi tarkistaa onko CompletionRemoved-toimintoon liitetty metodeja ja kutsuu toimintoa, jos metodeja on liitettynä vähintään yksi.

Näitä käytetään siten, että näkymämalliluokan (kuva 24), jossa näytettävä kokoelma on, muodostimessa lisätään haluttu metodi manageriluokan toimintoon delegaattina. Hajottimessa metodi poistetaan siltä varalta, jos luokan instanssi poistetaan ja metodia yritetään vielä tämän jälkeen kutsua.

```
    CompletionManager.CompletionRemoved += RemoveCompletion;
}

~ViewWorkoutViewModel()
{
    CompletionManager.CompletionRemoved -= RemoveCompletion;
}
```

Kuva 24. Metodien liittäminen ja poistaminen CompletionRemoved-toimintoon ViewWorkoutViewModel-näkymämalliluokassa.

Poistettavan objektin näkymämalliluokan poistamismetodissa (kuva 25) käsiteltävä objekti voidaan poistaa manageriluokissa sijaitsevista kokoelmista pelkän tunnusluvun avulla. Näkymämallin kokoelmasta objekti poistuu, kun kutsutaan manageriluokan poistotoimintoa, johon on liitetty tarvittavat metodit.

```
public ICommand DeleteWorkoutCompletion { get { return new CustomCommand(ConfirmDeleteWorkoutCompletion); } }

void DeleteCompletion()
{
    if (WorkoutManager.Workouts.ContainsKey(WorkoutManager.TempID))
    {
        WorkoutManager.Workouts[WorkoutManager.TempID].Completions.Remove(WorkoutCompletion);
        DayManager.RemoveCompletedWorkoutFromDay(WorkoutManager.TempID, WorkoutCompletion.CompletionDay.Date);

        CompletionManager.RaiseCompletionRemoved(this);
    }
}
```

Kuva 25. CompletionRemoved-toiminnon kutsuminen WorkoutCompletionViewModel-näkymämalliluokassa.

### 5.2.4 Navigation Manager

NavigationManager on staattinen luokka, jonka tarkoituksena on auttaa sovelluksen näkymien vaihtamisessa. Pääikkunaan viittaamista varten on tehty Frame-tyyppinen _mainFrame-muuttuja. Sovelluksen käynnistyessä tähän tallennetaan pääikkuna. Tähän avataan lisänäkymät ja poiston varmistamisnäkymä. Päänäkymiä varten oleva Frame-tyyppinen muuttuja löytyy pääikkunanäkymän luokasta.

Avatun päänäkymän väliaikaista tallentamista varten on määritelty _savedViewType-muuttuja. Kun päänäkymästä avataan lisänäkymä, tallennetaan tähän muuttujaan päänäkymän tyyppi. Tätä tarvitaan oikean päänäkymä

avaamiseen, kun palataan lisänäkymistä. Lisänäkymistä palaaminen avaisi muuten vain etusivun, koska päänäkymiä ei avata itse pääikkunaan.

Näkymien avaamiselle ja edelliseen näkymään palaamiselle on tehty OpenView- ja OpenPreviousView-metodit (kuva 26). OpenView ottaa parametrina avattavan näkymän tyypin. Näkymän vaihtaminen tapahtuu kutsumalla _mainFrame-muuttujan Navigate-metodia, joka ottaa parametrina OpenView-metodin parametrin.

OpenPreviousView-metodissa hyödynnetään _mainFrame-muuttujan ylläpitämää historiaa avatuista näkymistä. Aluksi tarkistetaan _mainFrame-muuttujan CanGoBack-metodilla, onko edellisiä näkymiä avattavana. Jos näin on, avataan edellinen näkymä. Koska sovelluksessa ei haluta palata näkymissä takaisin eteenpäin, tyhjennetään _mainFrame-muuttujan näkymähistoria seuraavien näkymien kohdalta ja säästetään samalla hieman muistia.

```
public static void OpenView(Type sourcePageType)
{
    MainFrame.Navigate(sourcePageType);
}

public static void OpenPreviousView()
{
    if(MainFrame.CanGoBack)
    {
        MainFrame.GoBack();
        MainFrame.ForwardStack.Clear();
    }
}
```

Kuva 26. OpenView- ja OpenPreviousView-metodit.

Poiston varmennusnäkymälle tehtiin oma OpenConfirmationView-metodi, koska kyseinen näkymä vaatii ICommand-komennon, käsiteltävän objektin nimen merkkijonona ja objektin tyypin. Nämä tiedot laitetaan metodiin parametreina ja asetetaan ConfirmationViewModel-näkymämalliluokan staattisiin muuttujiin.

### 5.2.5 Settings Manager

SettingsManager (kuva 27) on staattinen luokka, jonka tarkoituksena on säilyttää asetuksia sovelluksen ajon aikana. Tässä luokassa on Settings-luokan instanssi ja oletusasetusten palauttamista varten tehty ResetToDefaults-metodi. Metodi luo Settings-luokan instanssin uudelleen, palauttaen näin oletusasetukset.

```
public static class SettingsManager
{
    private static Settings _settings = new Settings();
    public static Settings Settings
    {
        get { return _settings; }
        set { _settings = value; }
    }

    public static void ResetToDefaults()
    {
        Settings = new Settings();
    }
}
```

Kuva 27. SettingsManager-luokka.

Settings-luokassa (kuva 28) on kaikki asetukset muuttujina, joita sovellus tarvitsee. Näitä ovat käytettävä painoyksikkö, joita on kilogrammat ja paunat, ja varmistuksen kysyminen treeniohjelmaa tai treeniohjelman suoritusta poistattaessa.

```
public class Settings
{
    WeightUnit _weightUnit = WeightUnit.kg;
    public WeightUnit WeightUnit
    {
        get { return _weightUnit; }
        set { _weightUnit = value; }
    }

    bool _confirmWorkoutDeletion = true;
    public bool ConfirmWorkoutDeletion
    {
        get { return _confirmWorkoutDeletion; }
        set { _confirmWorkoutDeletion = value; }
    }

    bool _confirmCompletionDeletion = true;
    public bool ConfirmCompletionDeletion
    {
        get { return _confirmCompletionDeletion; }
        set { _confirmCompletionDeletion = value; }
    }
}
```

Kuva 28. Settings-luokka, joka pitää sisällään kaikki asetukset muuttujina.

### 5.2.6 Save File Manager

SaveFileManager on staattinen luokka, joka käsittelee tietojen tallentamista ja lataamista. Treeniohjelmien, päiväyksien ja asetusten tiedot tallennetaan omiin tiedostoihinsa. Asetukset tallennetaan Windows-käyttäjän Roaming-kansioon. Tässä kansiossa saman sovelluksen asetukset pysyvät synkronoi-

tuina eri laitteiden välillä. Synkronoitavien tiedostojen yhteinen enimmäiskoko on rajallinen, joten tänne tallennetaan tiedostot, joiden koko pysyy samana.

Koska treeniohjelmien ja päiväysten tiedostot kasvavat normaalikäytössä, ei voida varmistaa, ettei näiden koko ylitä synkronoitavien tiedostojen enimmäiskokoa. Tästä syystä nämä tiedostot tallennetaan paikallisesti laitteelle. Näitä tiedostoja voi tosin käyttäjä itse siirtää laitteelta toiselle tarvittaessa.

Tietojen lukeminen ja kirjoittaminen tiedostosta vaatii koodissa tahdistamattomien toimintojen käyttöä. Näitä toimintoja kutsuttaessa pitää käyttää await-avainsanaa, muuten koodia ajettaessa ei välitetä onko toiminto suoritettu loppuun vai ei. Kaikki metodit ja muut toiminnot, joissa käytetään await-avainsanaa, pitää myös olla tahdistamattomia. Metodien sijaan tahdistamattomat tehtävät ovat sovelluksessa Task-tyyppisiä, koska tätä tyyppiä käyttäessä toiminnolle voi määrittää tarvittaessa palautusarvoja. Task-tyyppi kuvastaa tahdistamatonta toimintoa (Task Class 2016).

```
public static async Task SaveEverything()
{
    await SaveWorkouts();
    await SaveDays();
    await SaveSettings();
}


public static async Task LoadEverything()
{
    await LoadSettings();
    await LoadWorkouts();
    await LoadDays();
}
```

Kuva 29. Tehtävät tietojen tallentamista ja lataamista varten.

SaveFileManagerissa tallennus- ja lataamistehtävät eri tiedoille ovat hyvin samanlaiset. Tallentamista ja lataamista varten on tiedoille määritelty merkkijonomuuttujaan tiedoston nimi.

Tallentamistehtävässä (kuva 30) aloitetaan määrittelemällä StorageFile-tyyppinen tallentamistiedosto, joka luodaan joko paikalliseen Local-kansioon tai synkronoitavaan Roaming-kansioon. Jos samanniminen tiedosto on jo olemassa, se korvautuu uudella tiedostolla. Tämän jälkeen luodaan datavirta tallennustiedostoon. Itse tietojen kirjoittamista varten luodaan DataContract-Serializer-tyyppinen serializer-muuttuja, joka kirjoittaa tallennustiedostoon halutut tiedot käyttäen datavirtaa.

```
private const string _workoutsSaveFileName = "workoutsSave.txt";

public static async Task<bool> SaveWorkouts()
{
    try
    {
        StorageFile saveFile =
            await ApplicationData.Current.LocalFolder.CreateFileAsync(_workoutsSaveFileName, CreationCollisionOption.Repla

        using (Stream writeStream = await saveFile.OpenStreamForWriteAsync())
        {
            DataContractSerializer serializer = new DataContractSerializer(typeof(Dictionary<string, Workout>));

            serializer.WriteObject(writeStream, WorkoutManager.Workouts);
            await writeStream.FlushAsync();
            writeStream.Dispose();
        }
        return true;
    }
    catch(Exception e)
    {
        throw new Exception("ERROR, unable to save Workouts", e);
    }
}
```

Kuva 30. Treeniohjelmien tallennus.

Lataamistehtävässä (kuva 31) tarkistetaan aluksi, onko tallennustiedostoa olemassa. Jos sitä ei ole, poistutaan tehtävästä. Muuten luodaan tietovirta ja serializer-muuttuja, kuten tallennustehtävässä. Sitten luetaan tiedot tallennustiedostosta ja ladataan haluttuun paikkaan.

```
public static async Task LoadWorkouts()
{
    //Check if the file exists
    if (!System.IO.File.Exists(string.Format(@"{0}\{1}", ApplicationData.Current.LocalFolder.Path, _workoutsSaveFileName))
    {
        return;
    }

    var readStream = await ApplicationData.Current.LocalFolder.OpenStreamForReadAsync(_workoutsSaveFileName);

    if(readStream == null)
    {
        return;
    }

    DataContractSerializer serializer = new DataContractSerializer(typeof(Dictionary<string, Workout>));

    WorkoutManager.Workouts = (Dictionary<string, Workout>)serializer.ReadObject(readStream);
}
```

Kuva 31. Treeniohjelmien lataaminen.

# 6 LOPPUSANAT

Sovelluksesta tuli lähestulkoon suunnitelmien mukainen. Kaikki minkä piti olla mukana, saatiin toteutetuksi. Suunnitteluvaiheessa ei ollut vielä varmaa, mitä työkaluja tullaan toteutusvaiheessa käyttämään ja tämä johti toteutusvaiheessa joidenkin suunnitelmien muokkaamiseen toteutuskelposeksi.

Suurimmaksi haasteeksi muodostui päivitettyjen UWP-materiaalien löytäminen. Suurin osa aiheeseen liittyvistä materiaaleista koski UWP:tä edeltäneitä ohjelma-arkkitehtuureja ja vain osa näistä oli käyttökelpoisia. UWP-sovelluksen luomisesta tai XAML:n käytöstä ei ollut aikaisempaa kokemusta,

joten toteutusvaiheen alku oli hidas ja kului lähinnä opetteluun. Noin kahdessa viikossa työskentelystä tuli sujuvaa.

Sovelluksessa on markkinoitavaa potentiaalia. Vastaavanlaisia sovelluksia löytyy etenkin Android- ja iOS-laitteilla muutamia, mutta nämä eivät ole samalla tavalla selkeitä ja pelkistettyjä. Kesken treenin käyttäjä vaatii sovelluksesta yksinkertaisuutta ja selkeyttä, jota tämä sovellus tarjoaisi. Tosin sovellus toimii nykyisellään vain Windows-laitteilla ja toimiakseen muilla alustoilla se pitäisi koodata kokonaan alusta.

Ominaisuuksien osalta ohjelmaa voidaan kehittää vielä eteenpäin. Esimerkiksi treeniohjelmien tulostaminen olisi kätevää jos käyttäjällä ei ole yhteensopivaa puhelinta. Myös treeniohjelmien ja suoritusten yksinkertaisempi synkronointi laitteiden välillä olisi käyttömukavuuden kannalta tärkeää.

LÄHTEET

Develop apps for the Universal Windows Platform (UWP). s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/dn975273.aspx [viitattu 6.2.2016]

Gossman, J. 2006. Advantages and disadvantages of M-V-VM. Saatavissa: http://blogs.msdn.com/b/johngossman/archive/2006/03/04/543695.aspx [viitattu 13.2.2016]

Guide to Universal Windows Platform (UWP) apps. s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/dn894631.aspx [viitattu 6.2.2016]

Introduction to Universal Windows Platform (UWP) apps for designers. s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/fi-fi/library/windows/apps/dn958439.aspx [viitattu 6.2.2016]

Krug, S. 2014. Don’t make me think, revisited. A common sense approach to web usability. Yhdysvallat: New Riders.

Task Class. s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/system.threading.tasks.task.aspx?f=255&MSPPError=-2147217396 [viitattu 5.3.2016]

The MVVM Pattern. s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/hh848246.aspx [viitattu 13.2.2016]

UI basics for Universal Windows Platform (UWP) apps. s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/windows/apps/dn958432.aspx?f=255&MSPPError=-2147217396 [viitattu 20.12.2015]

What is XAML? s.a. Microsoft. Saatavissa: https://msdn.microsoft.com/en-us/library/cc295302.aspx [viitattu 7.2.2016]