**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ – ВАРНА**

**Факултет по Изчислителна Техника и Автоматизация**

**КАТЕДРА „Софтуерни и интернет технологии“**

„Разработка на програма за фитнес тренировки“

**Дипломна работа за придобиване на образователно-квалификационна степен „бакалавър“**

**Дипломант: Научен ръководител:**

Димо Миленов Петров /х. доц. М. Митев/

**Вapнa**

**2019**

Съдържание

[Глава 1. Въведение 4](#_Toc18313050)

[1.1. Въведение във фитенес тренировките 4](#_Toc18313051)

[1.2. Цел 4](#_Toc18313052)

[1.3. Задачи които произтичат от целта 4](#_Toc18313053)

[Глава 2. Обзор на програмите за организация и управление на фитнес тренировки 5](#_Toc18313054)

[2.1 Jefit 5](#_Toc18313055)

[2.2 MyWorkoutPlan 6](#_Toc18313056)

[2.3 PTminder 7](#_Toc18313057)

[2.4 Определяне изискванията към дипломната разработка след направения обзор. 8](#_Toc18313058)

[2.4.1 Общи изисквания: 8](#_Toc18313059)

[2.4.2 Функционални изисквания: 8](#_Toc18313060)

[Глава 3. Технология на обработка 9](#_Toc18313061)

[3.1 Описание на данните 9](#_Toc18313062)

[3.1.1 Структурни бизнес правила 9](#_Toc18313063)

[3.1.2 Концептуален модел 10](#_Toc18313064)

[3.1.3 Логически модел 11](#_Toc18313065)

[3.1.4 Релационен модел 12](#_Toc18313066)

[3.1.5 Физически модел 12](#_Toc18313067)

[3.1.6 Таблици и полета 13](#_Toc18313068)

[3.2 Използвани технологии 16](#_Toc18313069)

[3.2.1 Десктоп приложение 16](#_Toc18313070)

[3.2.2 WPF(Windows Presentation Foundation) 16](#_Toc18313071)

[3.2.3 XAML 17](#_Toc18313072)

[3.2.4 Code-Behind 17](#_Toc18313073)

[3.2.5 Visual Studio 17](#_Toc18313074)

[3.2.6 Prism 18](#_Toc18313075)

[3.2.7 O/RM Entity Framework 18](#_Toc18313076)

[3.2.8 Microsoft SQLServer 20](#_Toc18313077)

[3.2.9 GitHub 20](#_Toc18313078)

[3.2.10 MahApps.Metro 20](#_Toc18313079)

[Глава 4. Разработка на програмна система за фитнес тренировки 22](#_Toc18313080)

[4.1 Изисквания и прототипи на потребителския интерфейс 22](#_Toc18313081)

[4.1.1 Изисквания 22](#_Toc18313082)

[4.1.2 Use-Case 23](#_Toc18313083)

[4.1.3 Прототипи 23](#_Toc18313084)

[4.2 MVVM (Model-View-ViewModel) 27](#_Toc18313085)

[4.3 Модели и достъп до базата данни 27](#_Toc18313086)

[4.3.1 Класове на множествата(entities) 28](#_Toc18313087)

[4.3.2 DbContext 29](#_Toc18313088)

[4.3.3 Хранилища(Repositories) 30](#_Toc18313089)

[4.4 Lookups 31](#_Toc18313090)

[4.5 Dependency Injection 32](#_Toc18313091)

[4.6 Връзки между View и ViewModel 33](#_Toc18313092)

[4.7 Валидация на полетата 35](#_Toc18313093)

[Глава 5. Тестове на програмна система 36](#_Toc18313094)

[5.1 Тест на модул за въвеждане на данни за нов клиент 36](#_Toc18313095)

[5.2 Тест на модул за въвеждане на данни за ново упражнение 38](#_Toc18313096)

[5.3 Тест на модул за въвеждане на данни за нова тренировка 38](#_Toc18313097)

[5.4 Тест на модул за въвеждане на данни за начин на изпълнение на упражнението в дадена тренировка 39](#_Toc18313098)

[Глава 6. Ръководство на потребителя 41](#_Toc18313099)

[Начален екран на приложението 41](#_Toc18313100)

[Структура на прозореца 41](#_Toc18313101)

[Раздели(Tabs) 42](#_Toc18313102)

[Меню- Създаване 43](#_Toc18313103)

[Създаване на нов клиент 44](#_Toc18313104)

[Създаване на ново упражнение 44](#_Toc18313105)

[Създаване на нова тренировка 45](#_Toc18313106)

[Меню- Редактиране 47](#_Toc18313107)

[Редактиране, добавяне и премахване на Мускулни групи 48](#_Toc18313108)

[Редактиране, добавяне и премахване на Оборудване 49](#_Toc18313109)

[Глава 7.Заключение и бъдещо развитие 50](#_Toc18313110)

[Глава 8. Използвана литература 51](#_Toc18313111)

[Обем на чертежите 52](#_Toc18313112)

[Приложение 1: Протоколи от проведените тестове 52](#_Toc18313113)

[Приложение 2: Текст на програмата 60](#_Toc18313114)

# Глава 1. Въведение

## Въведение във фитнес тренировките

Фитнесът в най-широк смисъл е общата физическа подготвеност на организма на човека. Физическата форма е общо здравословно състояние и благосъстояние, и по-конкретно, е способността да се извършват различни видове спорт и спортни занимания. Физическата форма обикновено се постига чрез правилно хранене, умерено-тежък физически труд, спорт и почивка.

Преди индустриалната революция, фитнеса бил капацитет за извършване на дейностите за деня, без излишна умора. Въпреки това, с автоматизацията и промените в начина на живот, фитнес е мярка за способността на организма да функционира ефикасно и ефективно в работата и дейностите в свободното време, да е здрав, да може да се противопоставя на хипокинетичните заболявания, както и да се справя със стресови ситуации.

Дипломната работа разглежда един нов начин за проследяване на целия кръг- от срещата между фитнес инструктор и клиент, опознаването на индивидуалните потребности и желания на клиента, поставянето на конкретна цел и стъпките към постигането на желания резултат.

## Цел

Основната цел на настоящата дипломна работа е проектиране и разработка на десктоп базирана информационна система, за организация и съставяне на тренировъчни програми и списъци с информация за бъдещи и настоящи клиенти. Информационната система е с насоченост главно към фитнес треньори.

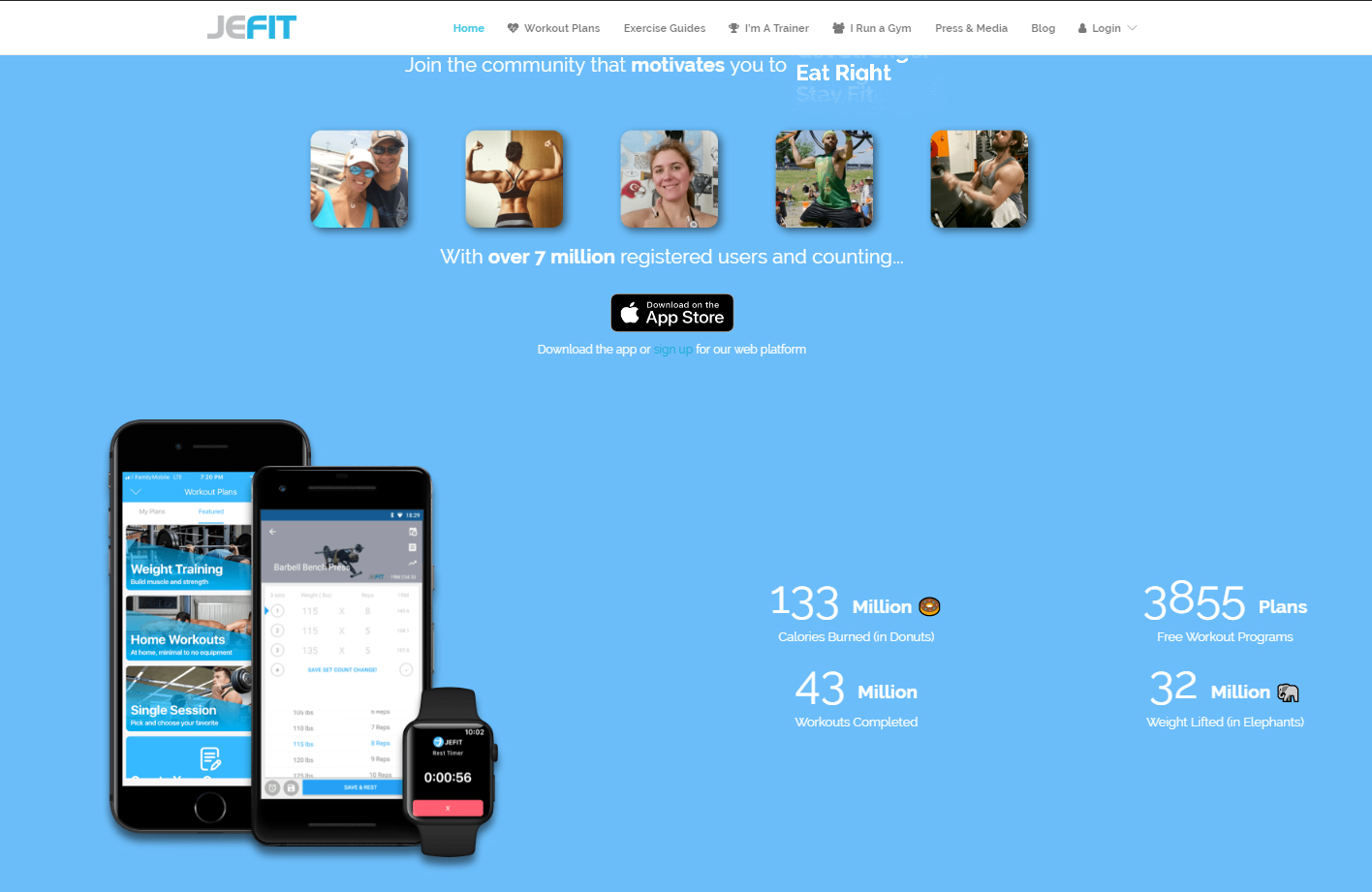
## Задачи, които произтичат от целта

* + 1. Да се анализират начините за изготвяне и структуриране на фитнес тренировки и техните характеристики.
    2. Да се анализира типа и структурата на информацията за всеки отделен клиент, нужна на треньора.
    3. Да се проектира и разработи база от данни, в която да се съхранява информация за тренировъчните планове, клиентите и техните характеристики, които ще бъдат част от информационната система.
    4. Да се проектира и разработи функциониращо десктоп базирано приложение, което да оперира с разработената база от данни.

# Глава 2. Обзор на програмите за организация и управление на фитнес тренировки

## Jefit

Jefit [1] е многоплатформено приложение за съставяне на персонални тренировъчни планове. Основната целевата група от хора на тази платформа, са хора които тренират и искат да следят своя прогрес.



Фиг.1 Екранна снимка от уеб сайта Jefit

Основни функции:

* съставяне на фитнес тренировъчни планове
* запазване и следене на информация за изминали тренировки
* следене и запазване на телесни измервания

Предимства:

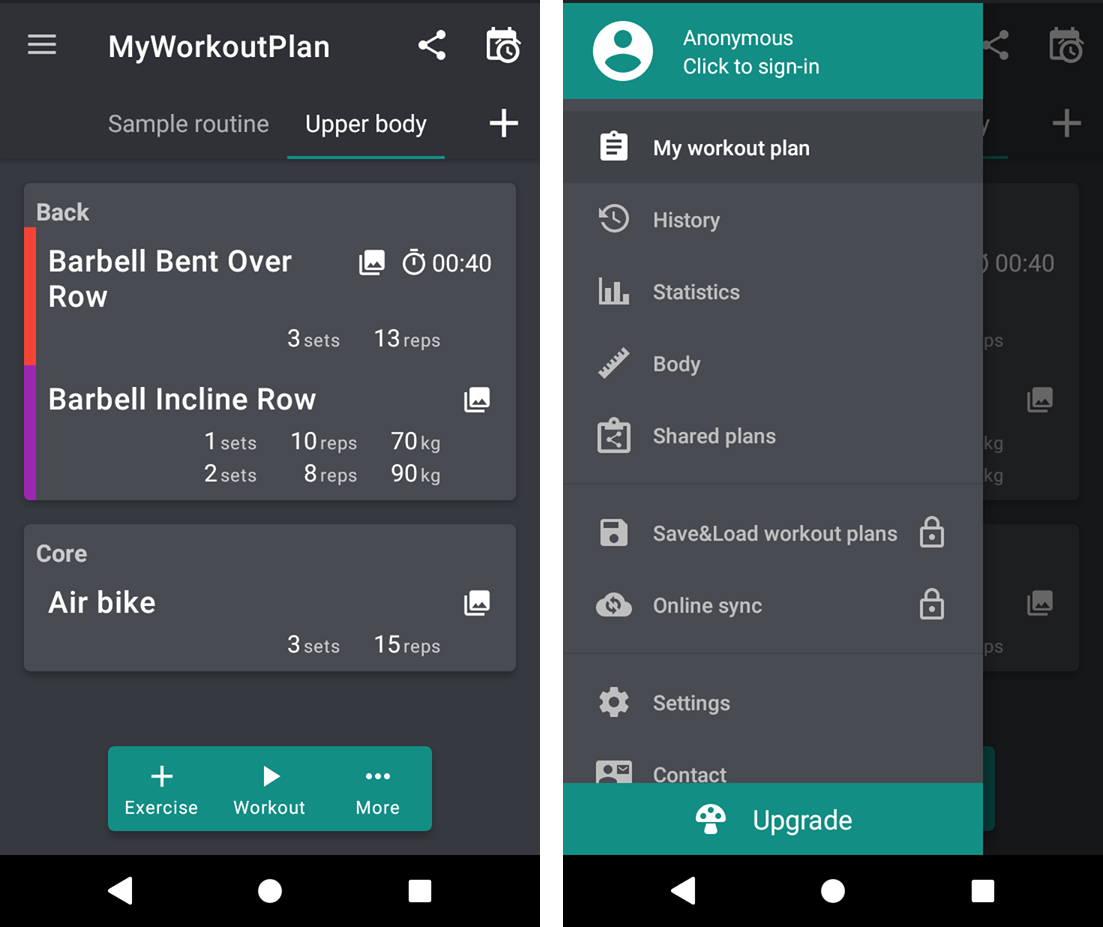
* платформата предоставя библиотека с голям набор от упражнения
* предоставя възможност за споделяне на тренировъчни планове с други потребители.
* предоставя възможност за автоматично изготвяне на анализи относно прогреса на потребителя във времето.
* дава възможност за запазване на персонализирани упражнения

Недостатъци:

* Интерфейсът е лошо структуриран и претрупан
* От потребителя се изисква прекалено много интеракция, за използване на основните функции.
* Наличен е само на английски език

## MyWorkoutPlan

MyWorkoutPlan [2] е мобилно приложение за устойства, които работят с операционната система Android. Приложението е насочено главно към хора, които тренират и желаят да изготвят сами своите тренировки.



Фиг.2 Екранни снимки от мобилното приложение MyWorkoutPlan

Основни функции:

* изготвяне на фитнес тренировъчен план
* добавяне на упражнение към всеки тренировъчен план
* групиране на упражненията по мускулни групи
* задаване на почивки между сериите и упражненията, както и цялостното време за изпълнение на тренировката
* възможност за записване на телесни измерения(тегло, размер на талия, гръдна обиколка и др.)

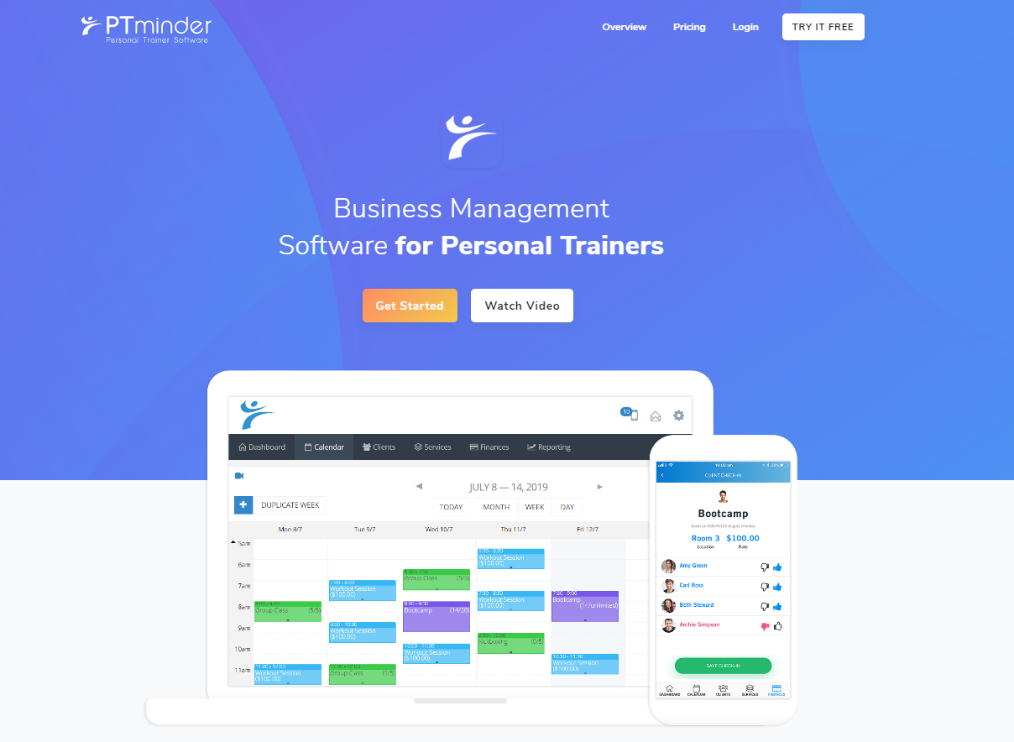
Предимства:

* потребителският интерфейс на приложението е добре структуриран.
* приложението дава възможност на потребителя да зададе цветови код на всяко упражнение в тренировъчния план.
* приложението дава възможност за споделяне на тренировъчните планове с други потребители.

Недостатъци:

* кратко или неясно описание на отделните функционалности на приложението
* липсва набор от упражнения, което задължава потребителя ръчно да въвежда информация за всяко едно упражнение за отделните тренировки.
* няма възможност за запазване на персонализирани упражнения

## PTminder

PTminder [3] е клауд базиран уеб сайт и мобилно приложение, което помага за управление на бизнеса на фитнес треньорите.

Фиг.3 Екранна снимка от уеб сайта на PTminder

Основни функции:

* създаване и управление на клиенти
* управление на плащанията
* създаване на тренировъчни планове за индивидуални клиенти или група от клиенти

Предимства:

* Удобен и добре структуриран интерфейс
* Лесна и удобна навигация
* Дава възможност за въвеждане на персонализирани упражнения.
* Пълно и ясно описание на функционалностите на сайта

Недостатъци:

* Не е достъпно на български език.

## Определяне изискванията към дипломната разработка след направения обзор.

### Общи изисквания:

* Опростен и добре структуриран интерфейс
* Постигане на желаните резултати с възможно най-малко стъпки
* Приложението да е леко и оптимизирано
* Приложението да е достъпно на български език

### Функционални изисквания:

* Създаване и управление на клиенти
* Възможност за добавяне на персонализирани упражнения
* Добавяне и премахване на мускулни групи и оборудване
* Създаване и управление на тренировъчени планове
* Избор на клиенти които да са част от тренировъчния план
* Търсене на упражнения, които да се добавят към тренировъчната програма
* Навигационни менюта за вече създадените клиенти, упражнения и тренировки

# Глава 3. Технология на обработка

## Описание на данните

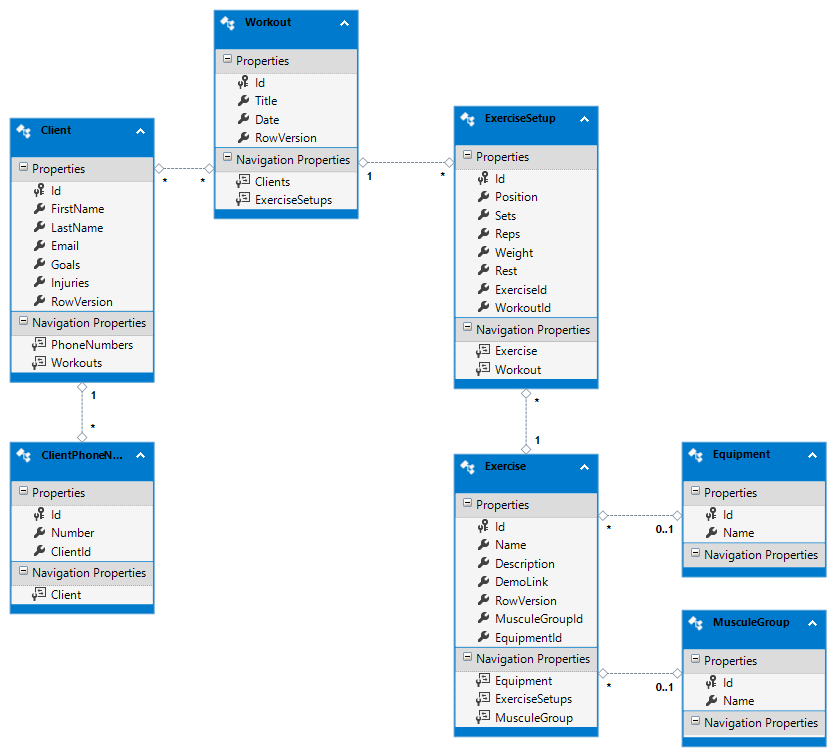
### Структурни бизнес правила

Структурните бизнес правила определят информацията, която трябва да се съхранява в базата данни. Те са следните:

* Един клиент може да има много телефонни номера
* Един телефонен номер може да принадлежи само на един клиент
* Един клиент може да е част от много тренировки
* Една тренировка може да има много клиенти
* Една мускулна група може да принадлежи на много упражнения
* Едно упражнение може да има една мускулна група
* Едно оборудване може да принадлежи към много упражнения
* Едно упражнение може да има само едно, или да няма оборудване
* Едно изпълнение на упражнението принадлежи само на една тренировка и съдържа само едно упражнение
* Едно упражнение има много изпълнения на упражнението, за много тренировки
* Една тренировка има много изпълнения на упражнението, за много упражнения
* Мускулните групи- съдържат наименование
* Оборудванията- съдържат наименование
* Телефонен номер- съдържа: номер, клиент
* Клиентите- съдържат: име, фамилия, Email, Цели, Заболявания
* Тренировките- съдържат: наименование и дата
* Изпълнение на упражнението- съдържа: брой серии, брой повторения, позиция, тежест, почивка, упражнение, тренировка
* Упражнението- съдържа: име, описание, линк, мускулна група, оборудване

Концептуалният модел на базата данни е представен на Фиг.6, логическия на Фиг.7, а релационния на Фиг.8

### Концептуален модел

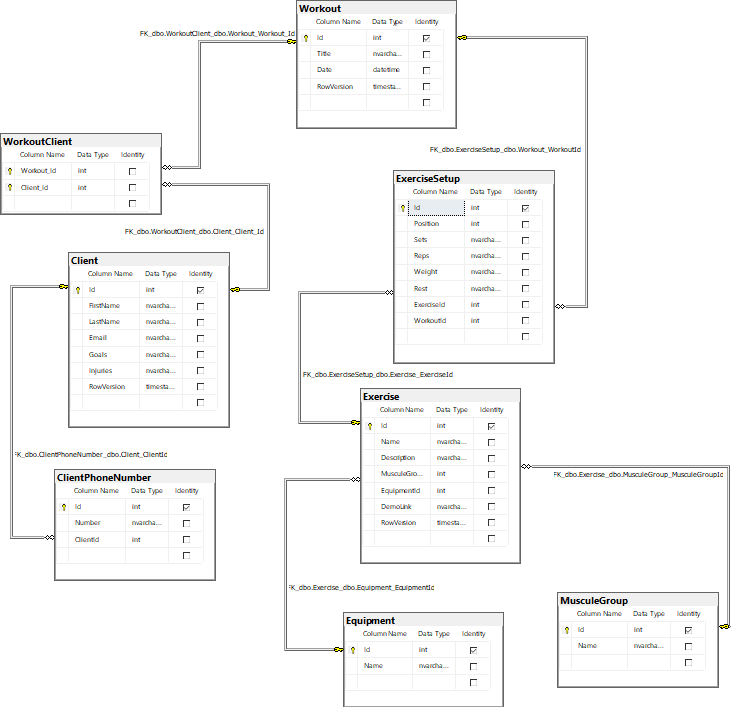
Концептуалният модел(фиг.6) е процес на изграждане модела на данните, свързан с дейността на дадена организация, който е независим от физическите фактори. Описва понятия от предметната област (домейн), техните взаимовръзки, както и налаганите ограничения на данните. [4]

Фиг.6 Концептуален модел на базата данни

### logicalModelЛогически модел

Фиг.7 Логически модел на базата данни

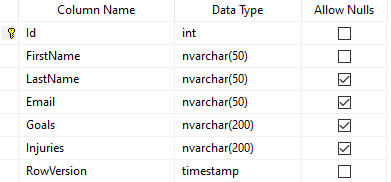
### Релационен модел

Фиг.8. Релационен модел на базата данни

### Физически модел

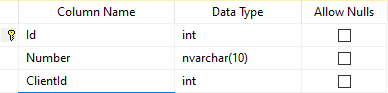
Описаният на Фиг.8 релационен модел отговаря на физическия моделна базата данни.

### Таблици и полета

**Таблица Client(Клиент)**

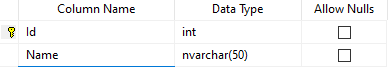
Фиг.9 Таблица Client, която съдържа информация за клиентите

* Id- идентификационен номер, int, задължително поле
* FirstName- име на клиента, nvarchar(50), задължително поле
* LastName- фамилно име на клиента, nvarchar(50), незадължително поле
* Email- адрес на електронна поща, nvarchar(50), незадължително поле
* Goals- цели на клиента, nvarchar(200), незадължително поле
* Injuries- заболявания на клиента, nvarchar(200), незадължително поле
* RowVersion- версия на записа, timestamp, задължително поле

**Таблица ClientPhoneNumber(Телефонен номер)**

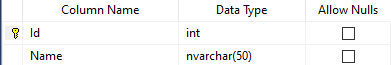
Фиг.10 Таблица ClientPhoneNumber, която съдържа информация за телефонните номера на клиента

* Id- идентификационен номер, int, първичен ключ
* Number- Телефонен номер, nvarchar(10), задължително поле
* ClientId- Вторичен ключ за клиента, който притежава този телефонен номер, свързан с първичния ключ към таблицата Clients, int, задължително поле

**Таблица MusculeGroup(Мускулна група)**

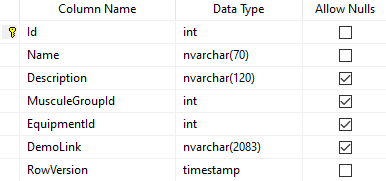
Фиг.11 Таблица MusculeGroup, която съдържа информация за мускулните групи

* Id- идентификационен номер, int, първичен ключ
* Name- Наименование на мускулната група, nvarchar(50), задължително поле

**Таблица Equipment(Оборудване)**

Фиг.12 Таблица Equipment, която съдържа информация за оборудването

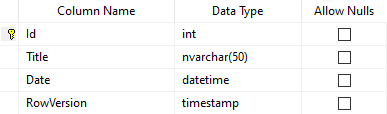
* Id- идентификационен номер, int, първичен ключ
* Name- Наименование на оборудването, nvarchar(50), задължително поле

**Таблица Exercise(Упражнение)**

Фиг.13 Таблица Exercise, която съдържа информация за упражненията

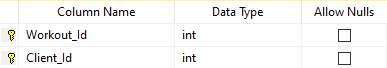
* Id- идентификационен номер, int, първичен ключ
* Name- наименование на упражнението, nvarchar(70), задължително поле
* MusculeGroupId- Вторичен ключ за мускулната група, към която се отнася упражнението, свързан с първичния ключ към таблицата MusculeGroup, int, незадължително поле
* EquipmentId- Вторичен ключ за оборудването, което е нужно за даденото упражнение, свързан с първичния ключ към таблицата Equipment, int, незадължително поле
* DemoLink- Url адрес, nvarchar(2083), незадължително поле
* RowVersion- версия на записа, timestamp, задължително поле

**Таблица Workout(Тренировка)**



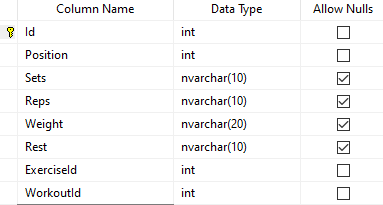
Фиг.14 Таблица Workout, която съдържа информация за тренировките

* Id- идентификационен номер, int, първичен ключ
* Title- заглавие на тренировката, nvarchar(50), задължително поле
* Date- дата и час на тренировката, datetime, задължително поле
* RowVersion- версия на записа, timestamp, задължително поле

**Таблица WorkoutClient**

Фиг.15 Таблица WorkoutClient, която служи за осъществяването на връзка от тип M:N

* Client\_Id- Вторичен ключ за клиента, който е част от тренировката, свързан с първичния ключ към таблицата Client, int, задължително поле
* Workout\_Id- Вторичен ключ за тренировката, от която е част клиента, свързан с първичния ключ към таблицата Workout, int, задължително поле

**Таблица ExerciseSetup(Начин на изпълнение на упражнение)**

Фиг.16 Таблица ExerciseSetup, съдържаща информация за начинът на изпълнение на дадено упражнение за дадена тренировка

* Id- идентификационен номер, int, първичен ключ
* Position- ред на изпълнение на дадено упражнение, int, задължително поле
* Sets- Серии, nvarchar(10), незадължително поле
* Reps- Повторения, nvarchar(10), незадължително поле
* Weight- Тежести, nvarchar(20), незадължително поле
* Rest- почивка, nvarchar(10), незадължително поле
* ExerciseId- Вторичен ключ за упражнението, към която се отнася начинът на изпълнение, свързан с първичния ключ към таблицата Exercise, int, задължително поле
* WorkoutId- Вторичен ключ за тренировката, към която се отнася начинът на изпълнение, свързан с първичния ключ към таблицата Workout, int, задължително поле

## Използвани технологии

### Десктоп приложение

Десктоп приложението е програма, на която потребителският интерфейс и клиентската логика на приложението се изпълняват самостоятелно на настолен или преносим компютър. За разлика от уеб приложенията, които се изпълняват в уеб браузър, десктоп приложенията се изпълняват от операционната система.

### WPF(Windows Presentation Foundation)

Windows Presentation Foundation (WPF) [5] позволява създаването на клиентски приложения за настолни компютри за операционната система Windows. В ядрото на WPF стои независима от разделителната способност и векторно базирана библиотека за графично изобразяване, която е създадена с цел да се възползва от съвременния графичен хардуер. WPF разширява ядрото с голям набор от функции за разработка на приложения които включват декларативен маркъп език за приложения (XAML), контроли, свързване на данни, двумерни и тримерни графики, шаблони, анимации, стилове, текст и типография и други. WPF позволява използването на XAML за визуализиране на приложението, както и управляващ език за програмиране(code-behind), за реализация на неговото поведение. Това разделение между визуализация и поведение на приложението има следните предимства:

* Усилията и цената за разработка и поддръжка на приложението са намалени, защото визуализационната и поведенческата част не са тясно свързани.
* Разработката е по-ефикасна, защото дизайнерите могат да имплементират визуализацията на приложението едновременно с програмистите, които имплементират логиката и поведението.

Предимства на WPF пред WinForms:

* XAML спомага за лесното създаване и редактиране на потребителския интерфейс и позволява работата да бъде разделена между дизайнера (XAML) и програмиста (C#,VB.NET и т.н.)
* DataBinding – позволява по- ясното разделение между данни и оформление.
* По-нов и следователно по-близък до текущите стандарти.
* По-гъвкав, така, че може да напарвите повече неща без да трябва да пишете или купувате нови контроли.
* За изрисуване на потребителския интерфейс, WPF използва графичния процесор, за по-добра производителност.

### XAML

XAML [6] е XML-базиран маркъп език, който има за цел да имлементира визуализирането на приложението, декларативно. Обикновено се използва за създаване на прозорци, диалогови кутии, страници и потребителски контроли(user controls) и запълването им с контроли, фигури и графики.

### Code-Behind

Основното поведение на приложението е да реализира функционалността, която реагира на взаимодействията на потребителите, включително обработка на събития (например, щракване върху меню, лента с инструменти или бутон) и извикване на бизнес логика и логика за достъп до данни в отговор. В WPF това поведение се реализира в код, който е свързан с маркъп(XAML). Този тип код е известен като code-behind.

### Visual Studio



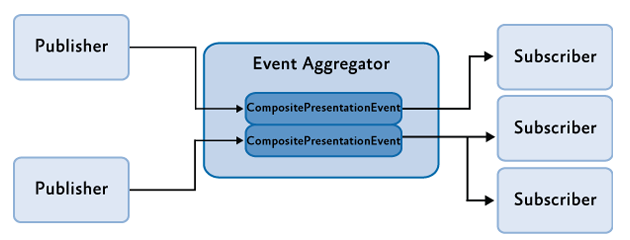
Visual Studio [7] предоставя мощна интегрирана среда за писане на код, компилиране, изпълнение, дебъгване (както за високо така и за машинно ниво), тестване на приложения, дизайн на потребителски интерфейс (форми, диалози, уеб страници, визуални контроли и други), моделиране на данни, моделиране на класове, изпълнение на тестове, пакетиране на приложения и стотици други функции. Използва се за разработка на конзолни и графични потребителски интерфейс приложения, както и Windows Forms или WPF приложения, уеб сайтове, уеб приложения и уеб услуги.

Могат да се добавят и плъгини, които повишават функционалността на почти всяко ниво – включително добавянето на поддръжка за source-control системи (като Subversion и Visual SourceSafe), добавяне на нови инструменти като редактори и визуални дизайнери за domain-specific languages или инструменти за други аспекти (като например: Team Foundation Server, Team Explorer).

### Prism



Библиотеката Prism [8] предлага механизъм за събития, който позволява комуникацията между слабо свързани компоненти в приложението. Този механизъм, базиран на услугата за агрегиране на събития, позволява на издателят (publisher) и абонатите (subscriber) да комуникират чрез събития, и въпреки това, да нямат пряка връзка един към друг.

Event Aggregator [9] предоставя мултикаст функционалност за публикуване/абонамент. Това означава, че може да има няколко издатели(publishers), които да публикуват едно и също събитие, както и много абонати(subscribers), които „слушат“ за едно и също събитие. Фиг.4.

Фиг.4 Схематично представяне на мултикаст функционалността публикуване/абонамент на EventAggregator

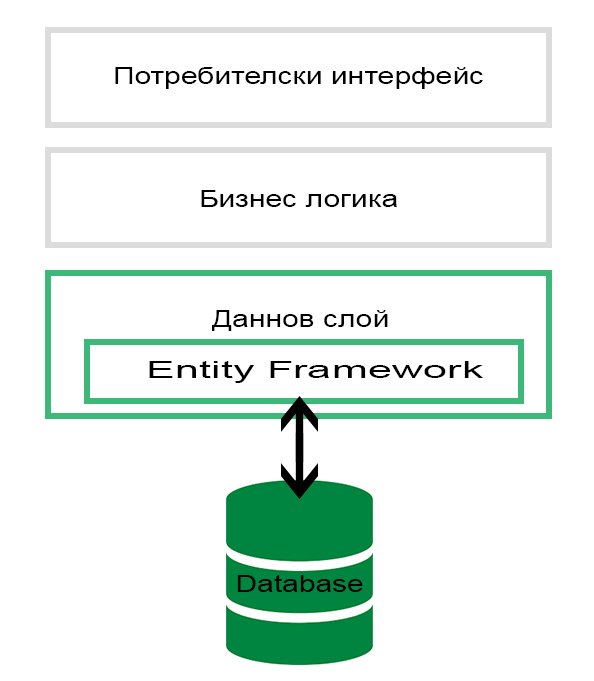
### O/RM Entity Framework

* **O/RM**

Обектно-релационно картографиране (ORM) е техника, която ви позволява да търсите и манипулирате данни от базата данни, използвайки обектно-ориентирана парадигма. Когато говорим за ORM, повечето хора се позовават на библиотека, която прилага техниката на обектно-релационното картографиране.

ORM библиотеката е обикновена библиотека, написана на избрания от вас език, която капсулира кода, необходим за манипулиране на данните, така че вече не използвате SQL, а взаимодействате директно с обект на същия език, който използвате. [10]

* **Entity Framework**

Entity Framework [11] е обектно-релационен картограф (O / RM), който позволява на .NET разработчиците да работят с база данни, използвайки .NET обекти. Той елиминира нуждата от повечето от кода за достъп до данни, който обикновено трябва да напишат програмистите.

Фиг.5 Мястото на Entity framework в приложенито

На изображението(Фиг. 5) Entity Framework се вписва между бизнес множествата(entities) и базата данни. Той записва данни, съхранявани в свойствата на множествата(entities), също така извлича данни от базата данни и автоматично ги превръща в обекти на множествата.

Инстанциите на класовете на множествата се извличат от базата данни с помощта на LINQ(Language Integrated Query). Всяка инстанция на контектния обект(DbContext) съдържа атрибут наречен ChangeTracker, който отговаря за проследяването на промените, които трябва да бъдат записани в базата данни. Докато се правят промени в екземплярите на класовете на субектите(entities), тези промени се записват в ChangeTracker атрибута и след това се записват в базата данни, когато извикате метода за запазване SaveChanges. Entity Framework отговаря за превеждането на промените в специфични за базата данни операции( например INSERT, UPDATE и DELETE).

### Microsoft SQLServer



Microsoft SQL Server [12] е система за управление на релационни бази данни, разработена от Microsoft. В тази си роля (на система за управление) SQL Server има основната функция да съхранява и извлича данни, по заявки на други софтуерни приложения, които могат да вървят на същия или друг компютър в дадена мрежа (в т.ч. и Интернет). MS SQL Server има над дузина различни издания, които са подходящи за различни аудитории и различни натоварвания: от приложения за отделни персонални компютри, до масивни приложения работещи постояно в онлайн режим и обхващащи огромно количество компютри.

### GitHub



GitHub [13] e уеб базирана услуга за разполагане на софтуерни проекти и техни съвместни разработки върху отдалечен интернет сървър в т.нар. хранилище (software repository). Базира се на Git системите за контрол и управление на версиите.

### MahApps.Metro

MahApps.Metro [14] е библиотека с отворен код, която предоставя на WPF приложенията Metro-стила, типичен за модерните приложения за платформата Windows. MahApps.Metro заменя в изгледа стандартните контроли, със свои собствени, базирани на концепции от приложения за Windows Phone и Windows 8. Огромно предимство при използване на тази библиотека е, че спестява време на програмистите при стилизиране на приложението.

# Глава 4. Разработка на програмна система за фитнес тренировки

## Изисквания и прототипи на потребителския интерфейс

### Изисквания

Изискванията към потребителския интерфейс са следните:

* Да има един прозорец който съдържа:
  + Меню за създаване на: нов клиент, нова тренировкаи ново упражнение
  + Меню за редактиране на: мускулни групи, оборудване
  + Навигационно меню с три раздела: за клиенти, за тренировки и за упражнения
  + Основно поле, където да се изобразяват детайлните изгледи за: клиент, тренировка, упражнение, мускулни групи и оборудване
* Бутони от тип “раздели”(tab control), предоставящи възможност за навигиране между отделните детайлни изгледи
* Подходящи съобщения за грешка
* Интерактивен дизайн
* Използване на стандартните за съответната операционна система контроли за падащи менюта, диалози, избор на дата и час и др.

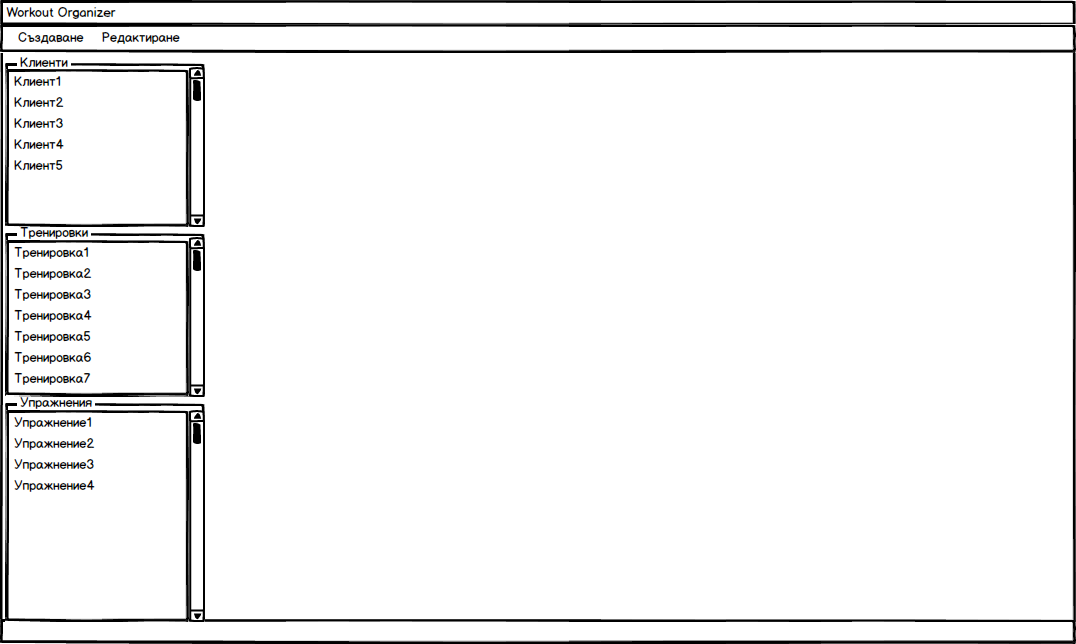
### Use-Case

### Прототипи

* **Начален прозорец**

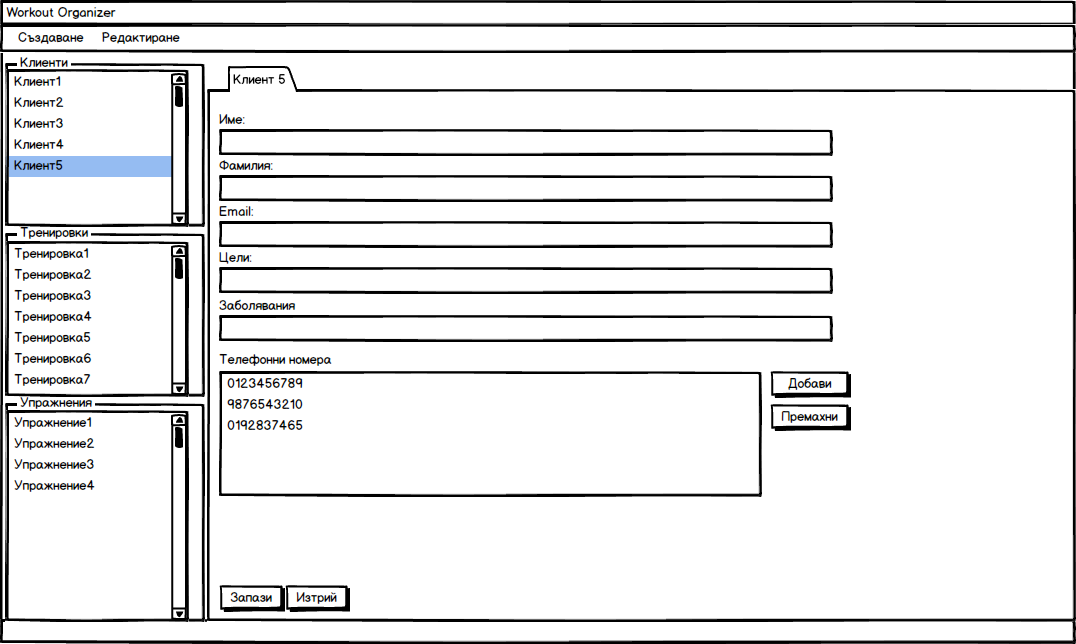
Изгледът при начално зареждане на приложението е показан на фиг.17 Може да се види навигационното меню, както и лентата с менюта за създаване и редактиране

Фиг.17 Изглед на началният екран при зареждане на приложението



* **Детайлен изглед с информация за клиент**

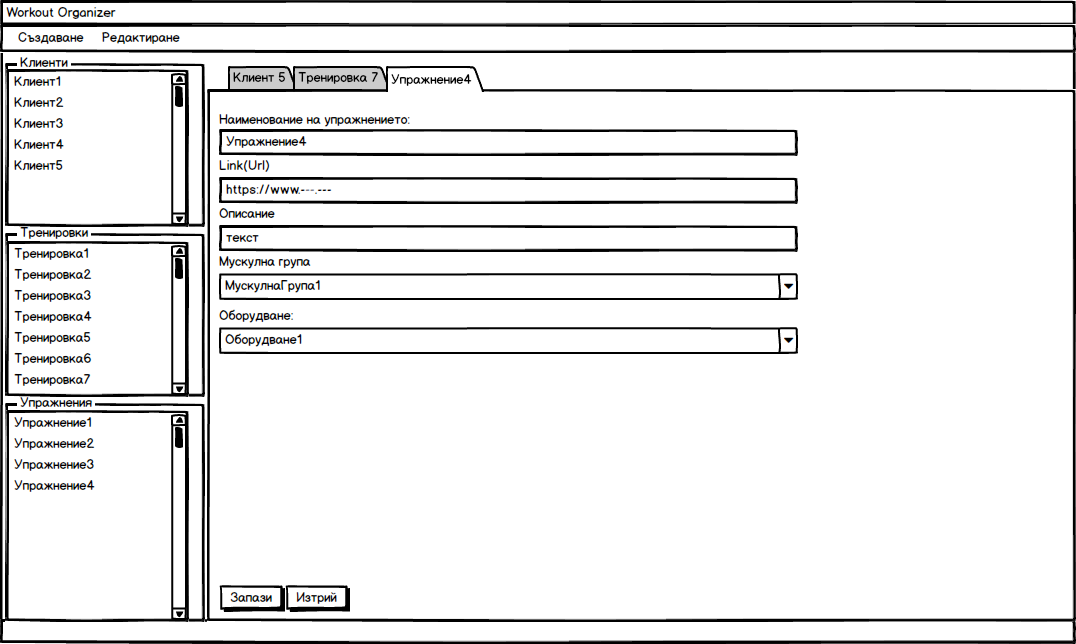
Детайлният изгледът за клиент е показан на фиг.18



Фиг.18 Изглед с детайли за клиент

* **Детайлен изглед с информация за упражнение**

Детайлният изгледът за упражнение е показан на фиг.19

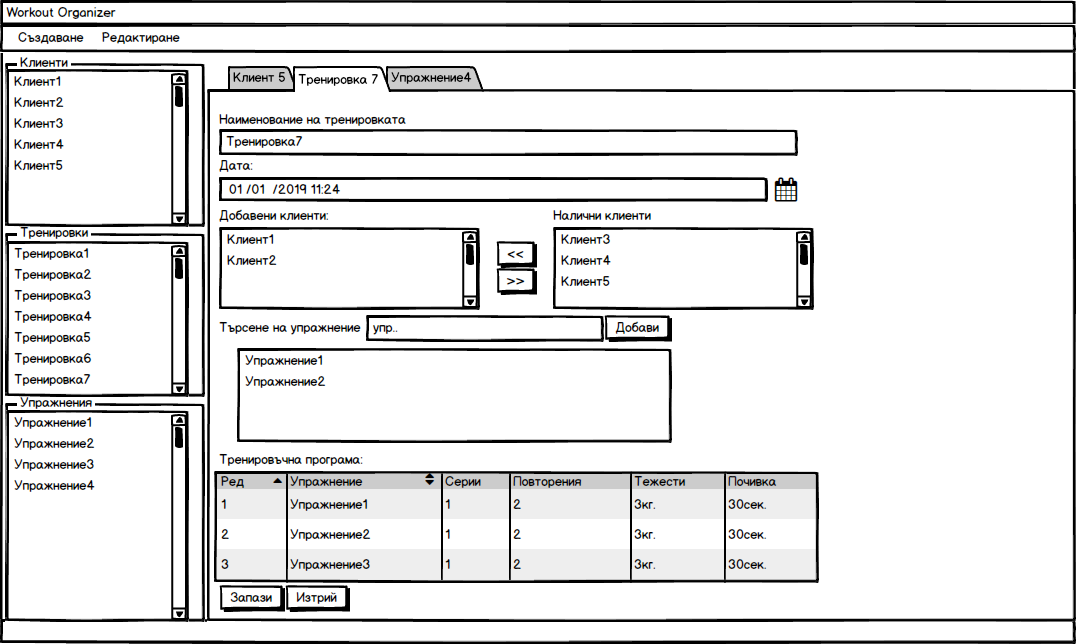


Фиг.19 Изглед с детайли за клиент

* **Детайлен изглед с информация за тренировка**

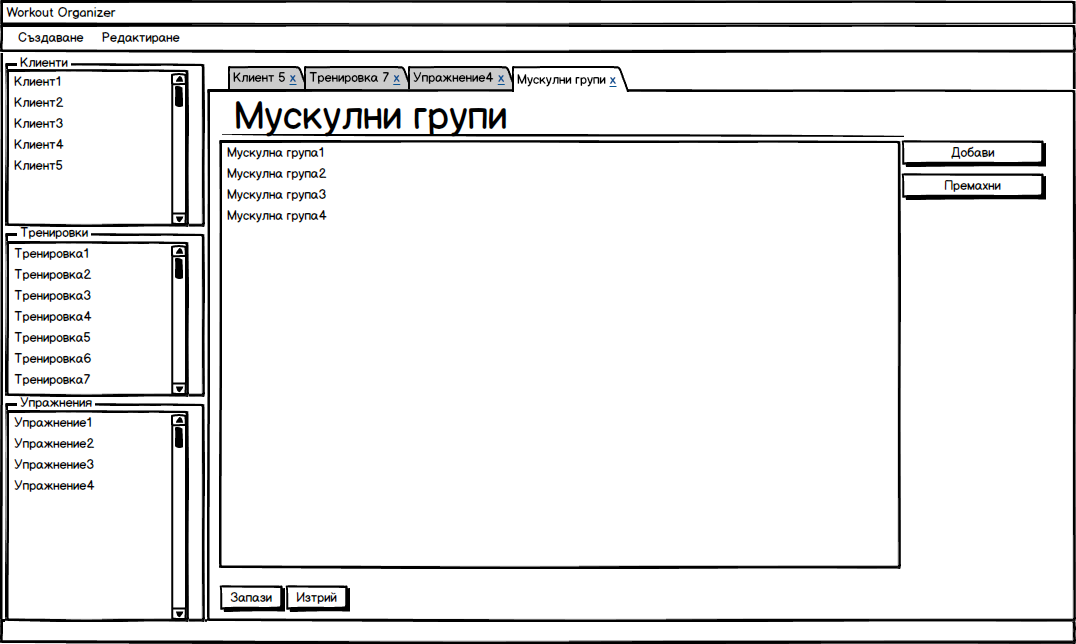
Детайлният изгледът за тренировка е показан на фиг.20

Фиг.20 Изглед с детайли за тренировка



* **Детайлен изглед с мускулни групи**

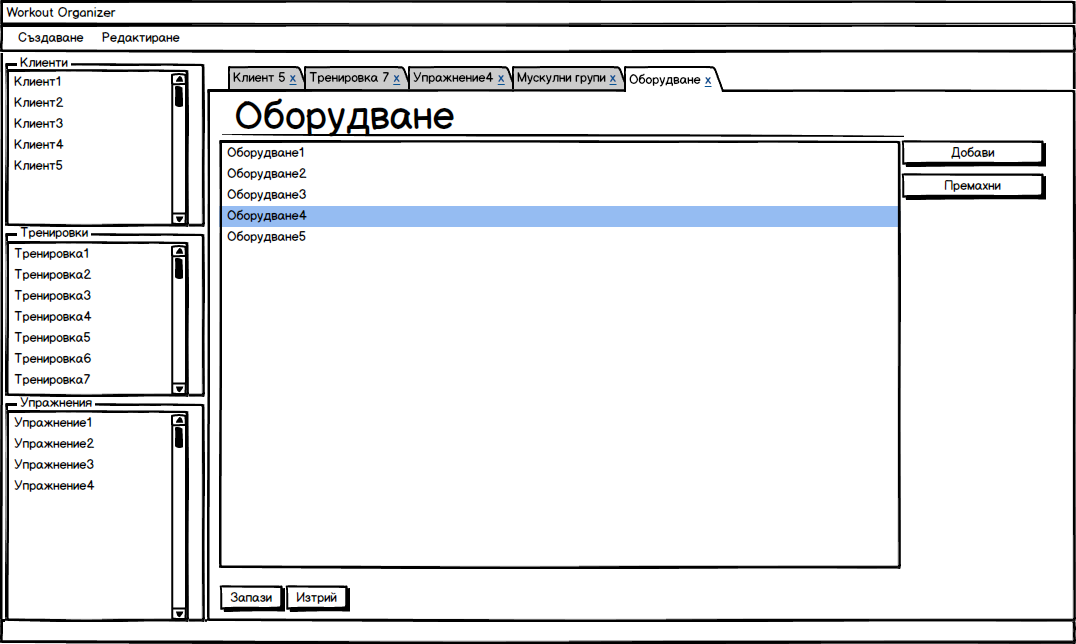
Детайлният изгледът с мускулни групи е показан на фиг.21



Фиг.21 Изглед с детайли за мускулни групи

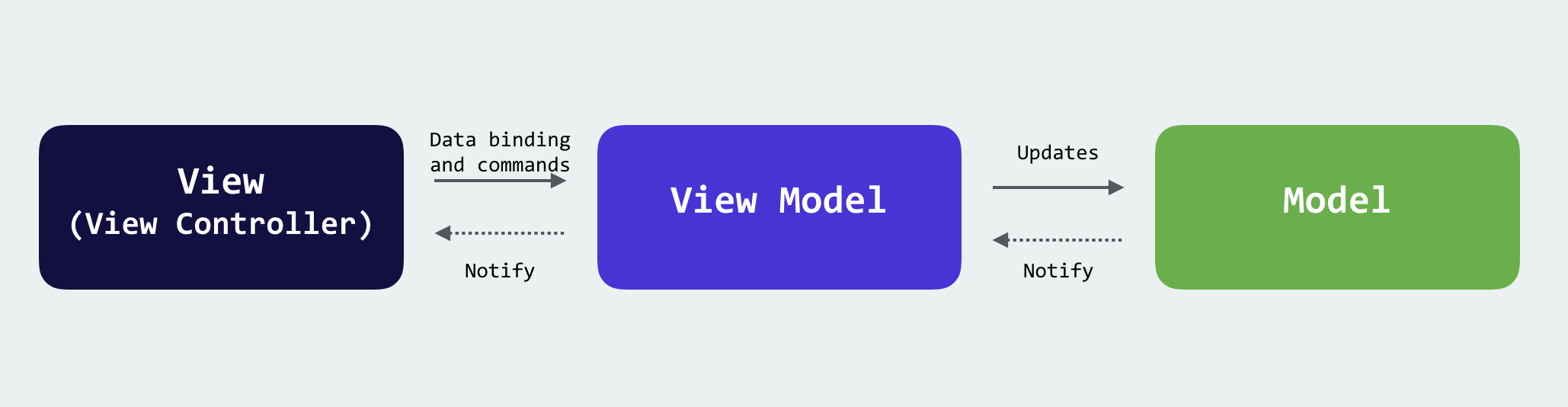
* **Детайлен изглед с информация за оборудване**

Детайлният изгледът с оборудване е показан на фиг.22



Фиг.22 Изглед с детайли за оборудване

## MVVM (Model-View-ViewModel)

Model-View-ViewModel (MVVM) [15] е софтуерен шаблон, който може да се използва при всички XAML платформи. Целта му е да осигури разделяне на връзките между контролите от потребителския интерфейс и тяхната логика. В MVVM има три оновни компонента: model(модел), view(изглед) и view model(изглед-модел). Всеки от тях изпълнява отделна и различна роля. Връзките между компонентите са показани на фиг.23

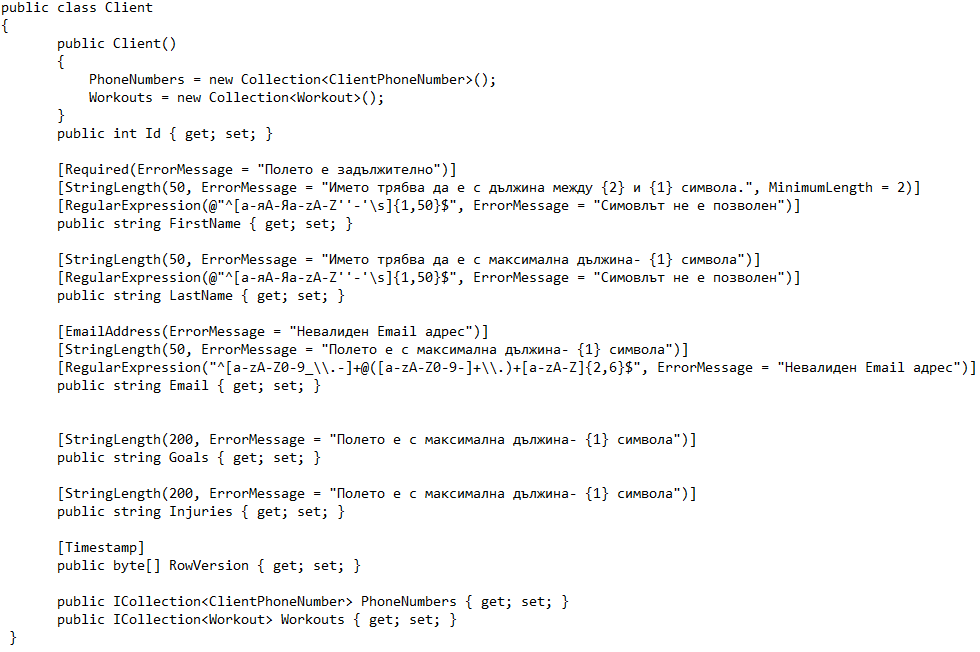
Фиг.23 Диаграма показваща връзките между отделните компоненти [16]

* Моделът(Model) в MVVM е имплементация на основния модел на приложението, което включва модел на данните, заедно с бизнес и валидационна логика
* Изгледът(View) е отговорен за създаването на подредбата, структурата и външния вид на това което вижда потребителя на екрана. View се дефинира изцяло с XAML, с ограничен code-behind, който не съдържа бизнес логика.
* Изглед-модел(ViewModel) работи като посредник между View и Model и е отговорен за обработката на изгледа(View). Обикновено ViewModel взаимодейства с Model като извиква методи от неговите класове. След това ViewModel предоставя данни от модела в подходящ за изгледа формат. Също така и предоставя имплементация на команди, които потребителя на приложението инициира в изгледа.

## Модели и достъп до базата данни

С Entity Framework достъпът до данни се осъществява с помощта на модел. Моделът е съставен от класове на множествата(entities) и контекстен обект(DbContext), който представлява сесия с базата данни, коeто позволява правенето на заявки и запазването на данни.

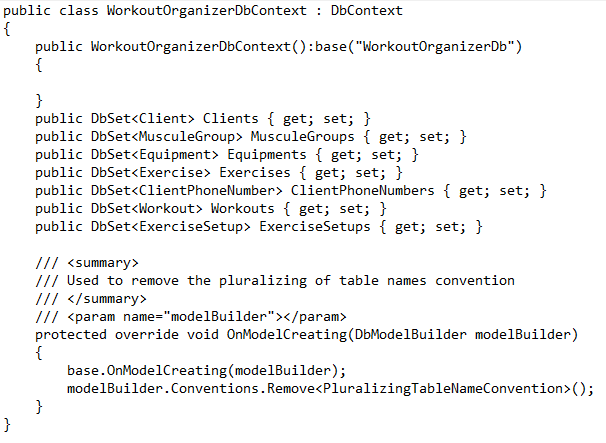
### Класове на множествата(entities)

Класовете на множествата представляват Plain Old CLR Object(POCO) обекти, които описват таблица от базата данни. Entity Framework използва набор от конвенции за създаване на модела базиран на формата на класовете на множествата (entity classes). Могат да се задават допълнителни конфигурации, които да допълнят и/или отменят това, което е открито чрез конвенция. Тези допълнителни конфигурации се задават чрез анотации(Data Annotations) към класовете и техните атрибути. Data Annotations служат и за част от валидационната логика. Пример за такъв клас е класът Client.cs(./WorkoutOrganizer/WorkoutOrganizer.Model/Client.cs) Фиг.24

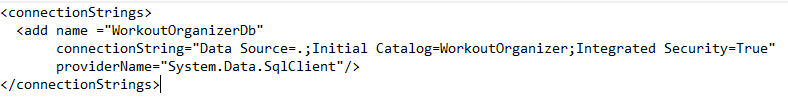
Фиг.24 Описание на класът Client.cs чрез Data annotations

Както се вижда на Фиг.4.4.1 анотациите са разположени над свойствата на класа и се записват в квадратни скоби. За валидация на полетата могат да се напишат персонализирани съобщенията за грешка, които се записват в ErrorMessage свойството на анотациите.

### DbContext

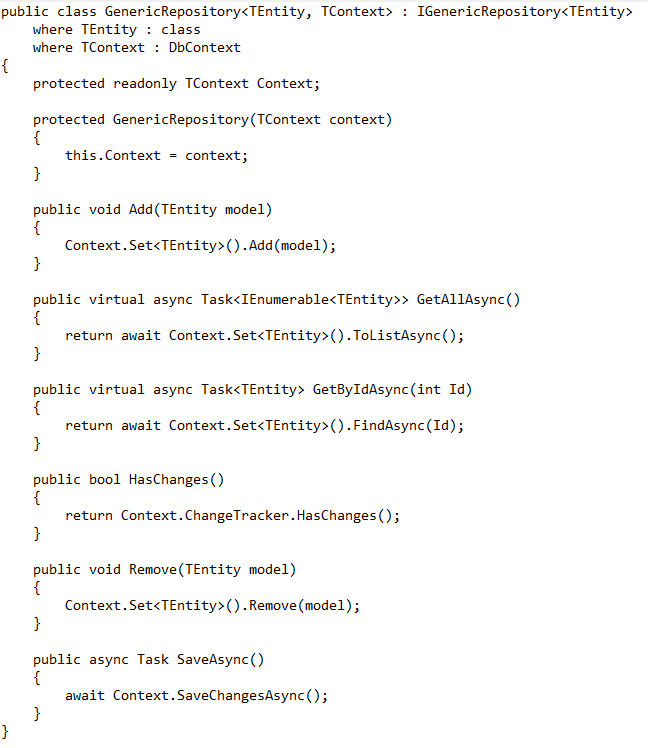
За да се използва DbContext в приложението, трябва да се създаде клас който да наследява DbContext, известен още като контекстен клас. Този контекстен клас обикновено включва в себе си DbSet<TEntity> атрибути, за всяко едно множество(entity) на модела. Връзката към базата данни се осъществява като се извика базовият конструкрур в който се подава името на базата данни. Имплементацията на класа може да се види на фиг.25 Тази връзка се описва и в App.config файлът Фиг.26

Фиг.25 Имплементация на контекстният клас WorkoutOrganizerDbContext.cs

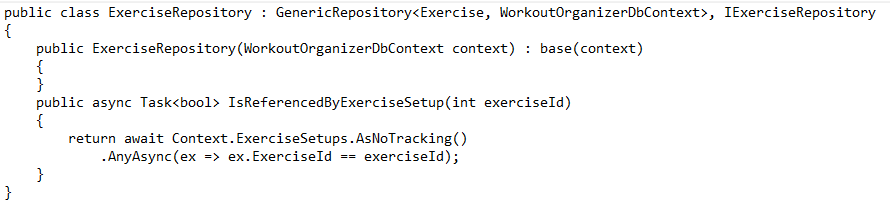


Фиг.26 Описание на връзката в App.config файла

### Хранилища(Repositories)

Хранилищата служат за изпълнение на стандартните CRUD(Create, Read, Update, Delete) операции с базата данни. В настоящата разработка са създадени хранилища за всеки един модел(entity) от базата данни. С цел да се избегне повтаряемост в кода, се създава шаблонно хранилище(Generic Repository), което съдържа основните операции с базата данни. Всяко отделно хранилище наследява базовия шаблонен клас GenericRepository<TEntity, TContext>, където TEntity е клас на множествата(entity), а TContext е контекстен клас. Имлементацията на шаблонният клас е показана на фиг.27. В отделните хранилища се имплементират и допълнителни, специфични за отделните множества операции. На фиг.28 е даден пример с хранилището ExerciseRepository, което се отнася за множестовто(entity) Exercise.

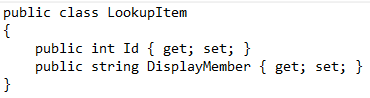
Фиг.27. Имплементация на шаблонния клас GenericRepository<TEntity,TContext>

Фиг.28 Имплементация на класа ExerciseRepository

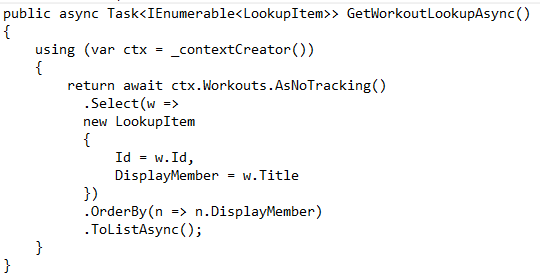
## Lookups

При запълване на контроли, от типа на combo box, list view и др., с информация, използването на целите обекти с всички техни полета, не е добра идея, защото това може да забави доста процесът на зареждане на даден изглед(View). С цел оптимизация, се изготвят т.нар. Lookups. Lookups са таблици с опростени обекти LookupItems, които съдържат в себе си два атрибута: Id и DisplayMember. Тези два атрибута представят идентификационния номер и текста за обекта, който искаме да представим в Lookup таблицата. Тези LookupItems обекти не се записват в базата данни.

За всеки модел(entity), който искаме да представим под формата на Lookup е създаден метод във класа LookupDataService. Този метод изпълнява заявка, която извлича от базата данни Id и полетата които искаме да изобразим в Lookup. След което ги записва в колекция от тип LookupItem.

Структурата на класа LookupItem е показана на Фиг.29, а имплементацията на метод за създаване на Lookup за тренировки(Workouts) в класа LookupDataService на Фиг.30

Фиг.29 Структура на класа LookupItem

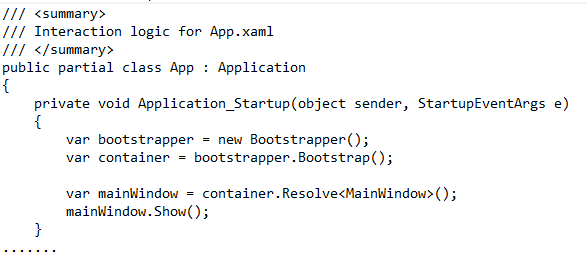


Фиг.30 Метод GetWorkoutLookupAsync(), който създава Lookup за тренировки.

## Dependency Injection

Dependency Injection [17] (инжектиране на зависимости) представлява шаблон за дизайн, който позволява да отстраним здравите връзки между класовете и да ги подменяме по време на компилация или по време на изпълнението на нашата програма. Главната причина за създаването на този шаблон е да могат да се променят конкретните имплементации, които се използват по време на изпълнение на програмата, а не при компилиране. Това позволява кодът, който пишем, да е много по модуларен и „разкачен“, което го прави в същото време и много по-тестваем.

За да реализациятата на Dependency Injection, се използва библиотеката Autofac [18]. Чрез нея се създават метод в класа Bootstrapper, наречен Bootstrap, който служи за създаване на т.нар. контейнер. В този контейнер се регистрират зависимостите и начинът по който да бъдат разрешавани, т.е. кой клас да се инстанцира, когато ни е необходима конкретна имплементация на даден интерфейс. Създаването на този контейнер се извършва при стартиране на приложението. Стартирането на приложението се извършва в метода Application\_Startup на code-behind класа на App.xaml. Фиг.31

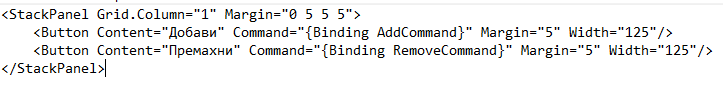
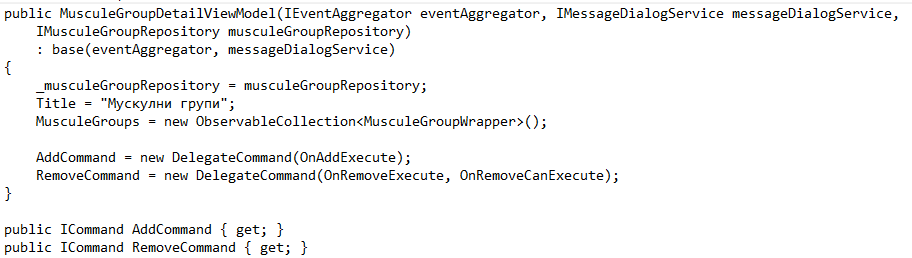
Фиг.31 Създаване на контейнер в Application\_Startup метода на класа App.xaml.cs

## Връзки между View и ViewModel

Едно View може да има собствен ViewModel, или да наследява този на своя родител. Всяко View получава данните си от собствения ViewModel, чрез връзки(bindings), или чрез извикване на методи от ViewModel. По време на изпълнение на програмата View се променя, когато UI отговарят на ViewModel свойства, повдигащи събития за известяване за промяна(change notification events). [15]

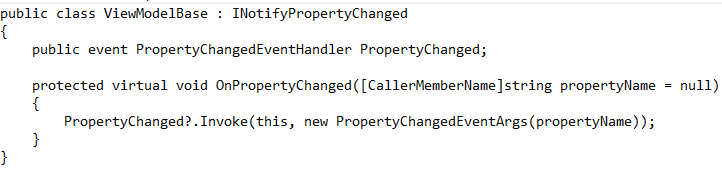
Има различни възможности за изпълнение на код във ViewModel, в отговор на взаимодейстията във View, като например натискане на бутон или избор на елемент. Ако контрола е източник на команда, свойството му Command може да бъде свързано с ICommand свойство на ViewModel. Когато се извика командата на контрола, се изпълнява кода във ViewModel.

Пример за такава връзка е даден на фиг.32 На фиг.32. са показани бутоните за добавяне и премахване на мускулна група, във XAML файлът на MusculeGroupDetailView.xaml, които са източници на команда. При натискане на някой от двата бутона, се свързват с ICommand свойството, със съответното име, в MusculeGroupDetailViewModel.cs. Задаването на метод, който да се изпълни при в извикване на командата, става при инициализацията на тази команда в конструктура. Фиг.33

Фиг.32 Бутони за добавяне и премахване на мускулна група, които са източници на команда.

Фиг.33 Свойствата за добавяне и премахване на мускулна група в класа MusculeGroupDetailViewModel инициализирането им неговия конструктур

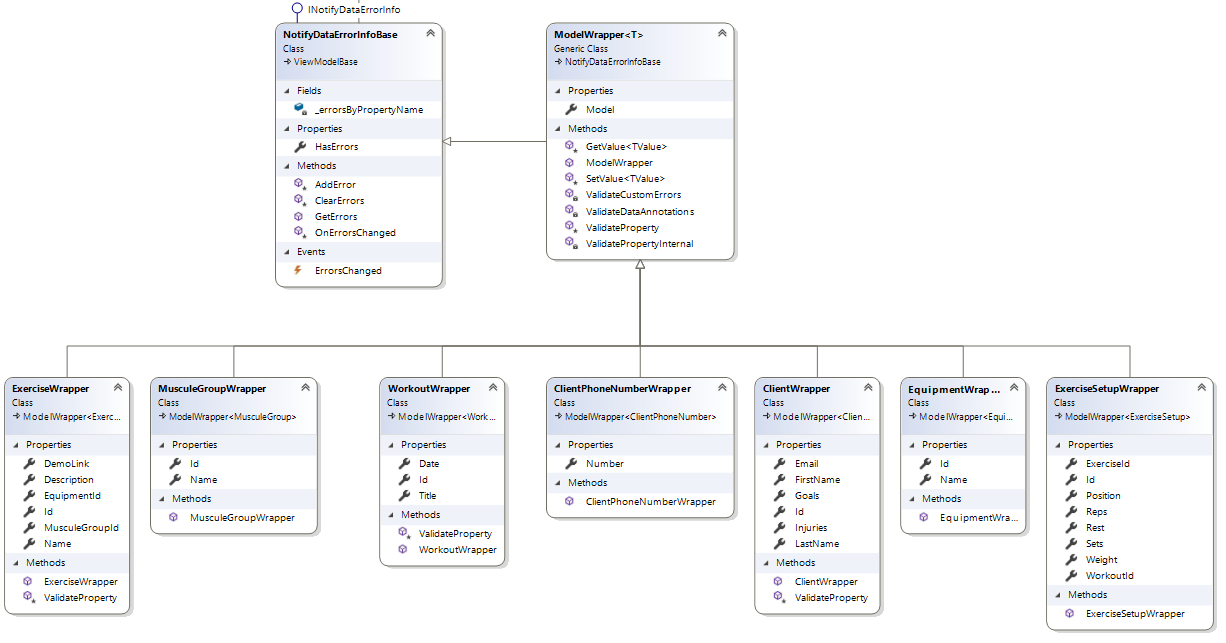
За да може ViewModel да участва в двупосочна връзка на данните(two-way data binding) с View, неговите свойства трябва да предизвикват събитието PropertyChanged. Класовете на ViewModel изпълняват изискването за имплементация на интерфейса INotifyPropertyChanged и предизването на събитието PropertyChanged при промяна на свойство.

За да се избегне повтаряемост на кода, имплементацията на интерфейса INotifyPropertyChanged е направена в базовия клас ViewModelBase Фиг.34. Всеки ViewModel, който участва в two-way data binding, наследява този клас.

Фиг.34 Имплементация на INotifyPropertyChanged в базовия клас ViewModelBase

## Валидация на полетата

Връзката на данни(data-binding) в WPF има вградена поддръжка на валидиране. Когато свойството на целта се записва обратно в свойството на източника, се появява валидация. За да се използва тази вградена функционалност, класовете на моделите(entities) трябва да имплементират интерфейса INotifyDataErrorInfo. Но за да се запазят класовете на моделите възможно най-опростени, е нужно тези класове да се „обвият“ с т.нар. Wrapper класове. Чрез Wrapper класовете, програмиста може да добави и допълнителна валидационна логика, отделна от тази, която предоставят анотациите на моделите(entities). Един Wrapper клас съдържа същите полета, каквито има и модела(entity).

На Фиг.35 е показана диаграма на класовете Wrappers и имплементацията им с интерфейсът INotifyDataErrorInfo. NotifyDataErrorInfoBase класът имплементира интерфейсът INotifyDataErrorInfo. ModelWrapper класът е шаблонен базов клас, който наследява NotifyDataErrorInfoBase и съдържа в себе си методите, нужни за валидиране на свойствата на отделните Wrapper класове. Всеки Wrapper клас наследява базовия клас ModelWrapper<T>, където T е модел(entity).

Фиг.35 Диаграма на класовете Wrappers, ModelWrapper и NotifyDataErrorInfoBase връзките помежду им

# Глава 5. Тестове на програмна система

Цялостната проверка на разработената програмна част и съответно, адекватно на проекта изпълнение на предвидената функционалност, е задължително условия за постигането на качествено и надеждно програмно осигуряване. Обекти на тестване могат да бъдат отделните програмни компоненти (модули, процедури и др.) и/или цялата програмната система. [19]

За тестване на отделните модули, в настоящата разработка, процедурите подлагани на тестване се разглеждат като „черна кутия“. Не се анализира процедурата на ниво оператори и не се следят междинните резултати. За провеждането на тест е необходимо:

* Определяне на входните данни за всеки модул и тяхното класифициране като правилни и неправилни. Правилни данни са тези , които отговарят на обявената функционалност и са в определения от проекта диапазон на изменение.
* Определяне на „чувствителни“ данни или комбинации, при които изпълнението на процедурата може да доведе до неочаквани резултати
* Изпълнение на тестовите последователности и анализ на получените резултати.

## Тест на модул за въвеждане на данни за нов клиент

**Определяне ограниченията на полетата:**

* Полето за име не може да е празно
* Полетата за име и фамилия не могат да бъдат по-дълги от 50 символа и по-къси от 1 символ
* Полетата за име и фамилия не могат да приемат специални символи
* Полето за и-мейл адрес не може да е по-дълго от 50 символа
* Полето за и-мейл да е валиден и-мейл адрес
* Полетата за цели и заболявания на клиента да не са по-дълги от 200 символа
* Телефонните номера на клиента не са по-дълги от 10 символа
* Телефонните номера да са валидни по българските стандарти мобилни телефонни номера. Първата цифра да е задължително 0, втората цифра да е 8 или 9, третата да е 8 или 9 или 7.

**Класифициране на входните данни на правилен и неправилен клас:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование на полето** | **Входно условие** | **Правилен клас** | **Неправилен клас** |
| 1. Име | 1. Не по-малко от 1 и не повече от 50 символа  2.Само букви  3.Без специални символи | а-я , А-Я, a-z,  A-Z  От 1 до 50 символа | <1 и >50  Специални символи  Празно поле |
| 2.Фамилия | 1. Не по-малко от едно и не повече от 50 символа  2.Само букви  3.Без специални символи | а-я , А-Я, a-z,  A-Z  От 1 до 50 символа | >50  Специални символи |
| 3.Email | 1. Не повече от 50 символа  2.Валиден e-mail адрес | < 50  Валиден e-mail | >50  Невалиден e-mail |
| 4.Цели | 1.Не повече от 200 символа | <200 | >200 |
| 5.Заболявания | 1.Не повече от 200 символа | <200 | >200 |
| 6.Телефонен номер | 1. Точно 10 символа  2.Първата цифра да е задължително 0, втората цифра да е 8 или 9, третата да е 8 или 9 или 7  3.Само цифри | 10 символа  Номера започващи с:  (0)(9|8)(8|9|7) | <10 и >10  Номера започващи с различно от:  (0)(9|8)(8|9|7)  Символи различни от цифри |

## Тест на модул за въвеждане на данни за ново упражнение

**Определяне ограниченията на полетата:**

* Полето за наименование на упражнението не може да е празно
* Полето за наименование на упражнението не може да бъде по-дълго от 70 символа и по-късо от 1 символ
* Полето за описание не може да бъде по-дълго от 120 символа
* Полето за Link(url) трябва да бъде валиден URL адрес
* Полето за Link(url) не може да бъде по-дълго от 2083 символа

**Класифициране на входните данни на правилен и неправилен клас:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование на полето** | **Входно условие** | **Правилен клас** | **Неправилен клас** |
| 1.Наименование | 1.Не по-малко от 1 и не повече от 50 символа | От 1 до 70 символа | <1 и >70 символа  Празно поле |
| 2.Link(Url) | 1.Не повече от 2083 символа  2.Валиден url | <2083 символа  Валиден url адрес | >2083  Невалиден url адрес |
| 3.Описание | 1.Не повече от 120 символа | <120 символа | >120 символа |

## Тест на модул за въвеждане на данни за нова тренировка

**Определяне ограниченията на полетата:**

* Полето за наименование на тренировката не може да е празно
* Полето за наименование на тренировката не може да бъде по-дълго от 50 символа и по-късо от 1 символ
* Полето за дата не може да приема дати, преди настоящата дата

|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование на полето** | **Входно условие** | **Правилен клас** | **Неправилен клас** |
| 1.Наименование | 1.Не по-малко от 1 и не повече от 50 символа | От 1 до 50 символа | >50 символа  Празно поле |
| 2.Дата | 2.Дати, които не са преди настоящата | Днешна дата или следваща дата | Предходни дати |

## Тест на модул за въвеждане на данни за начин на изпълнение на упражнението в дадена тренировка

**Определяне ограниченията на полетата:**

* Полето за серии не може да бъде по- дълго от 10 символа
* Полето за повторения не може да бъде по- дълго от 10 символа
* Полето за тежести не може да бъде по- дълго от 20 символа
* Полето за почивка не може да бъде по- дълго от 10 символа

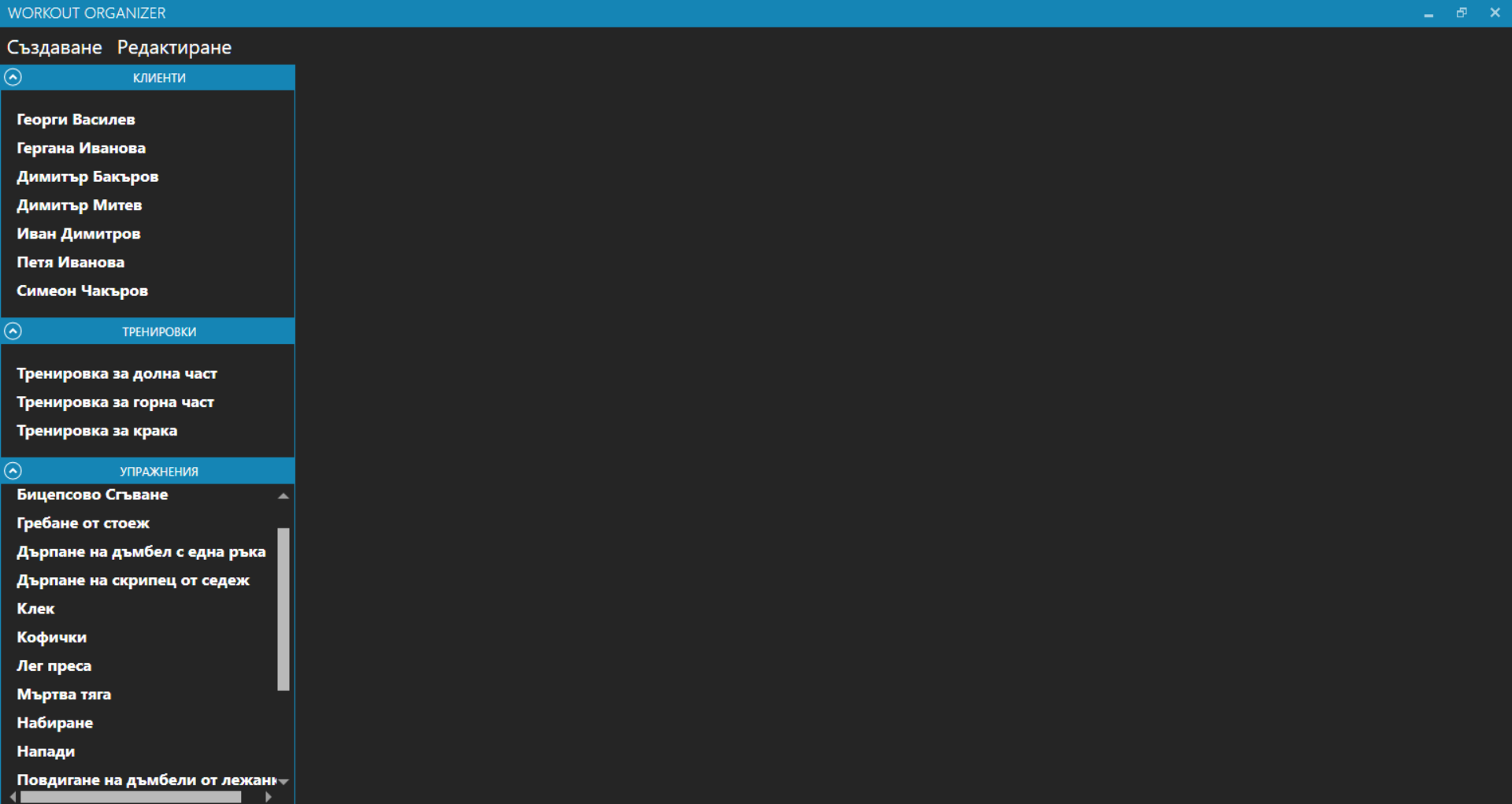
|  |  |  |  |
| --- | --- | --- | --- |
| **Наименование на полето** | **Входно условие** | **Правилен клас** | **Неправилен клас** |
| 1.Серии | Не повече от 10 символа | <10 | >10 |
| 2.Повторения | Не повече от 10 символа | <10 | >10 |
| 3.Тежести | Не повече от 20 символа | <20 | >20 |
| 4.Почивака | Не повече от 10 символа | <10 | >20 |

Протоколи и резултати от направените тестове са описани в Приложение1.

# Глава 6. Ръководство на потребителя

## Начален екран на приложението

На фиг. 36 е показан началният екран, при зареждане на приложението.

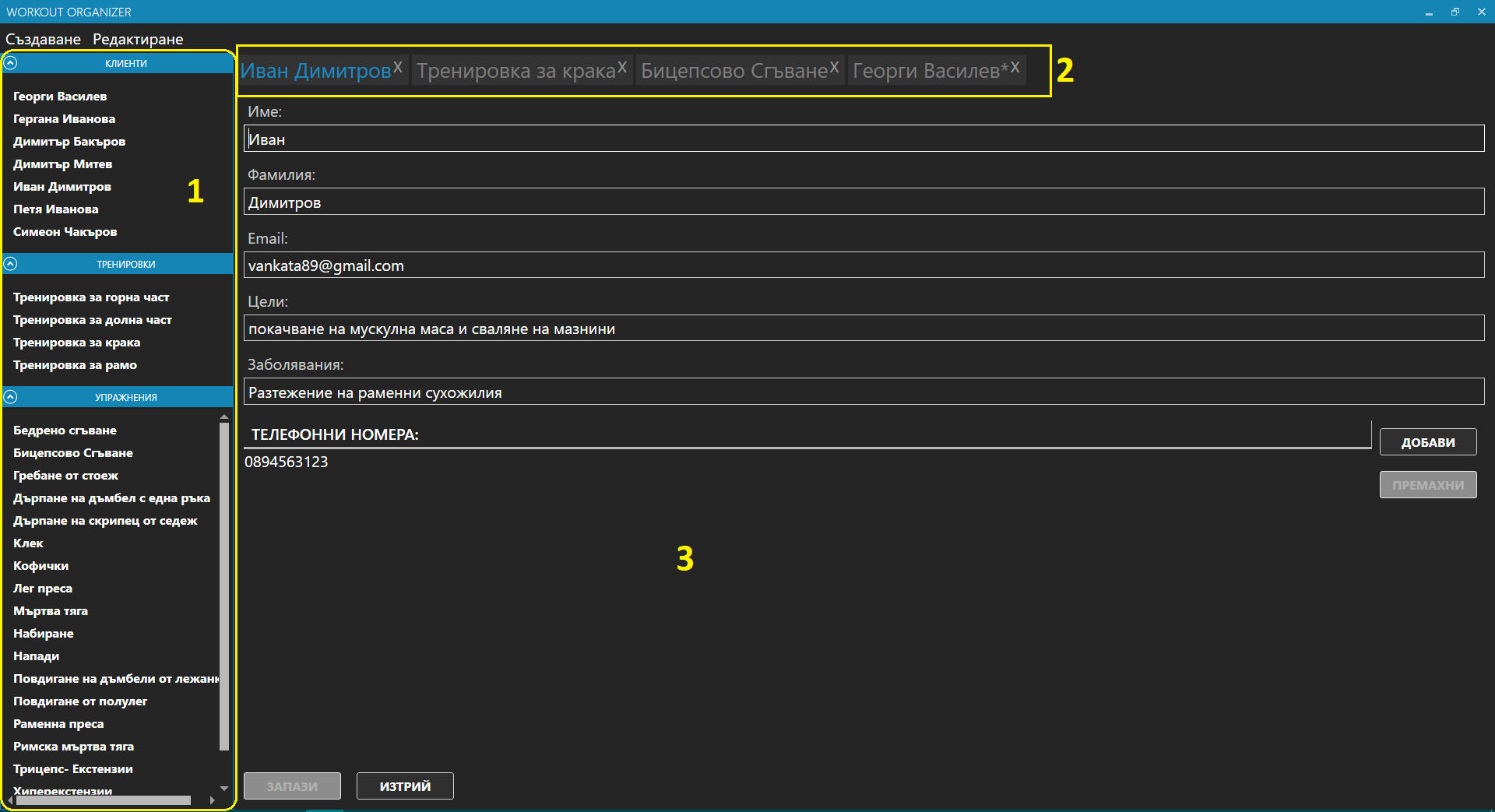


Фиг. 36 Начален екран при зареждане на приложениетоСтруктура на прозореца

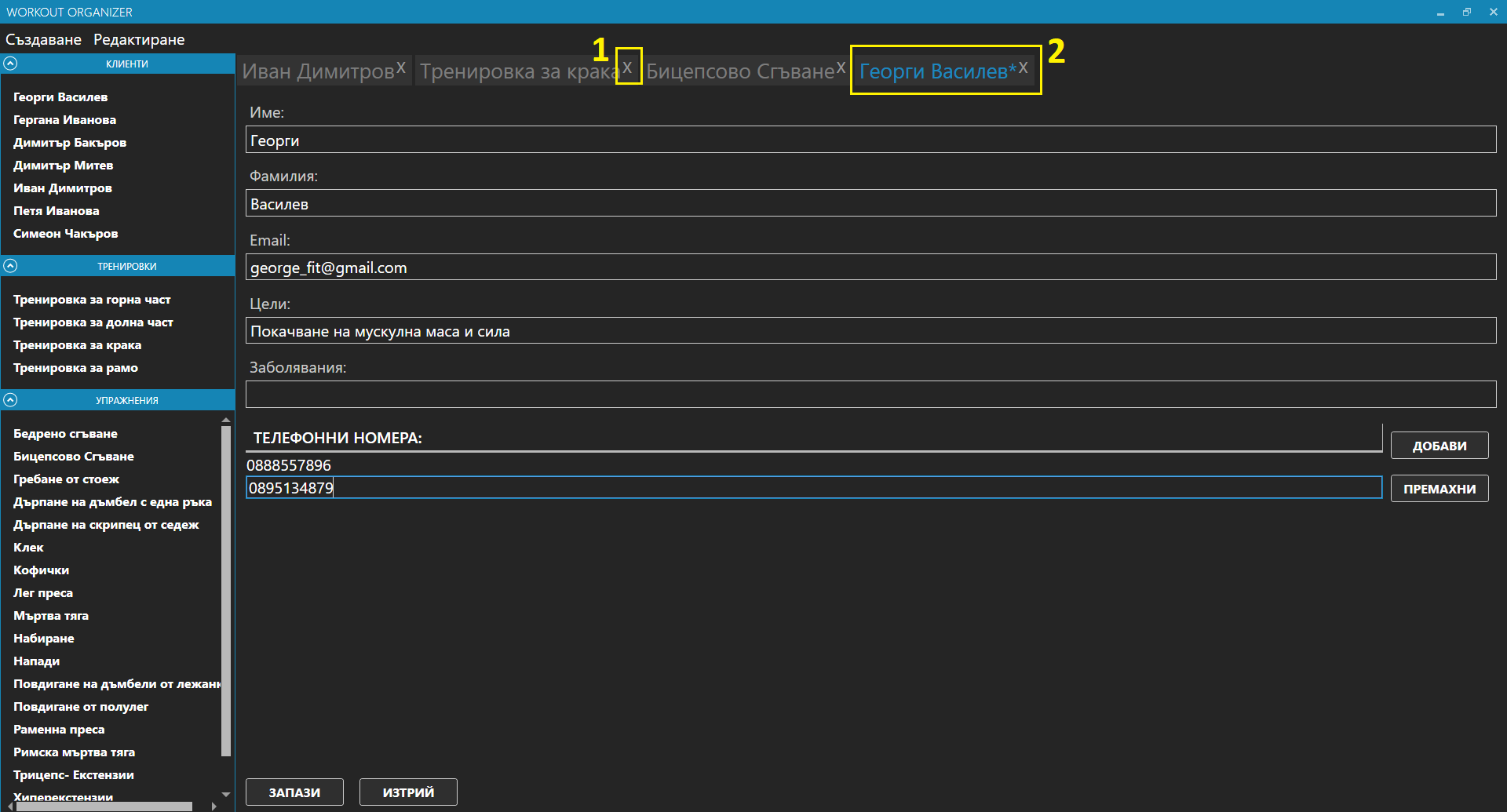
## Структура на прозореца

1. **Навигационно меню.** В навигационното има три раздела: Клиенти, Тренировки и Упражнения. Във всеки един има списъци със създадените съответно, клиенти тренировки и упражнения. Когато кликнете на елемент от този списък, информацията за съответния модел се показва в главното поле на изображението.
2. **Главно поле.** Главното поле служи за изобразяване на информацията на избраният от вас елемент, както и полета, в които можете да редактирате, добавяте или изтривате информация.
3. **Раздели(Tabs).** Разделите служат, за навигиране между отворените от вас елементи.

Фиг. 37 Структура на прозореца



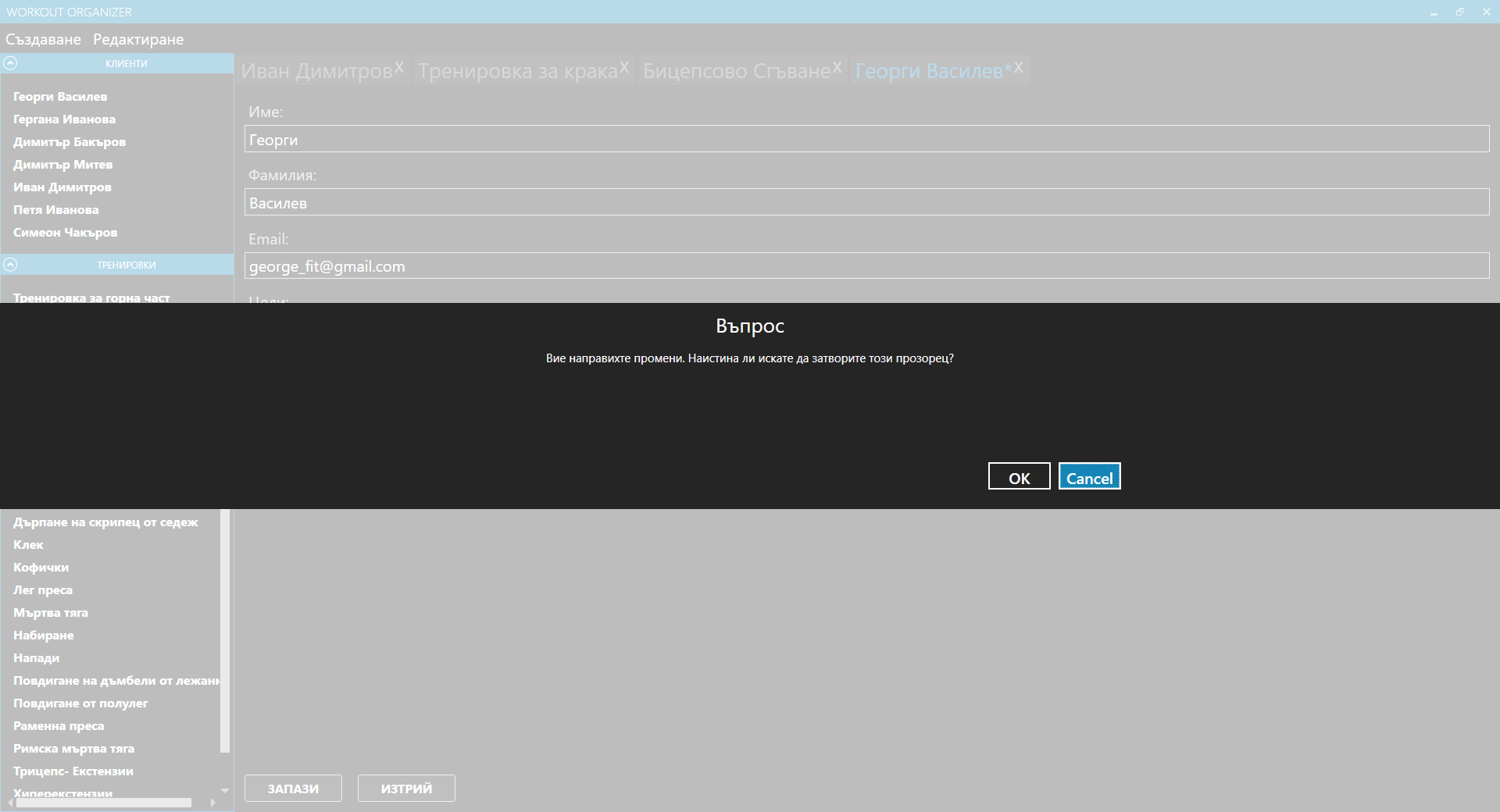
## Раздели(Tabs)



Фиг.38 Раздели(Tabs)

Ако желаете да затворите даден раздел(tab) натиснете бутон **1 „х“,** който се намира в след името на раздела(tab).

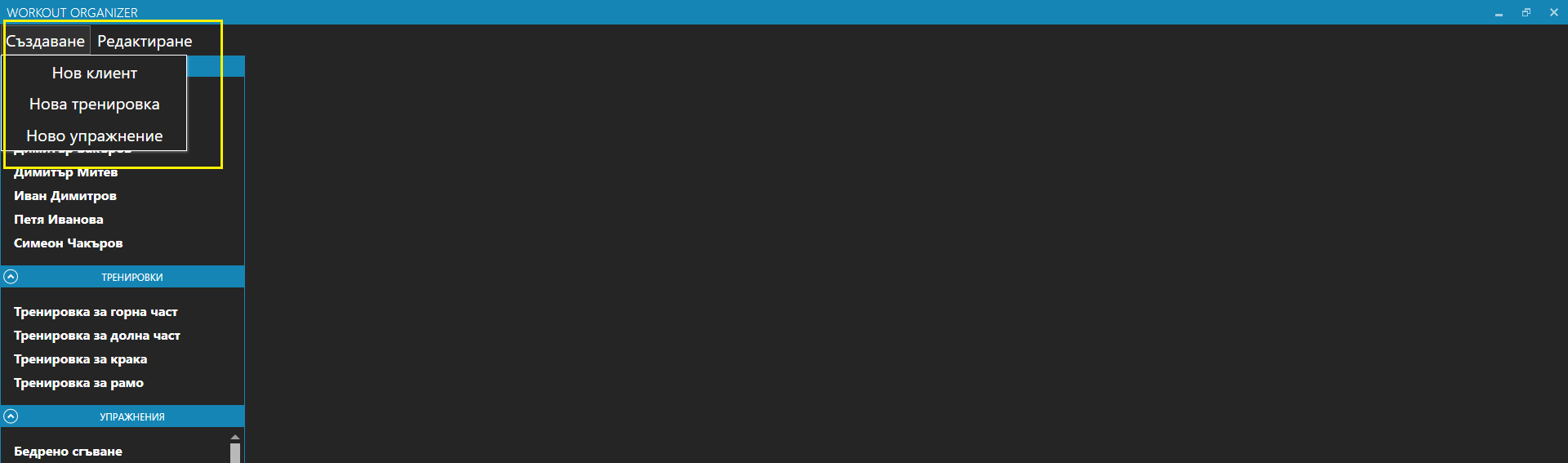
Ако сте направили промени в някое от полетата на даден раздел, и не сте ги запазили, след името на раздела(tab), се появява звезда.

Ако се опитате да затворите раздел, чиито промени не са запазени, в прозорецът изскача диалогова кутия Фиг. 39. „Вие направихте промени. Наистина ли искате да затворите този прозорец?“ и два бутона: „ОК“ и „Cancel“. При натискане на OK, системата затваря раздела и НЕ запазва промените. Ако натиснете Cancel, разделът не се затваря, но и промените остават незапазени.

Фиг. 39 Диалогова кутия за затваряне на незапазен раздел.

## Меню- Създаване

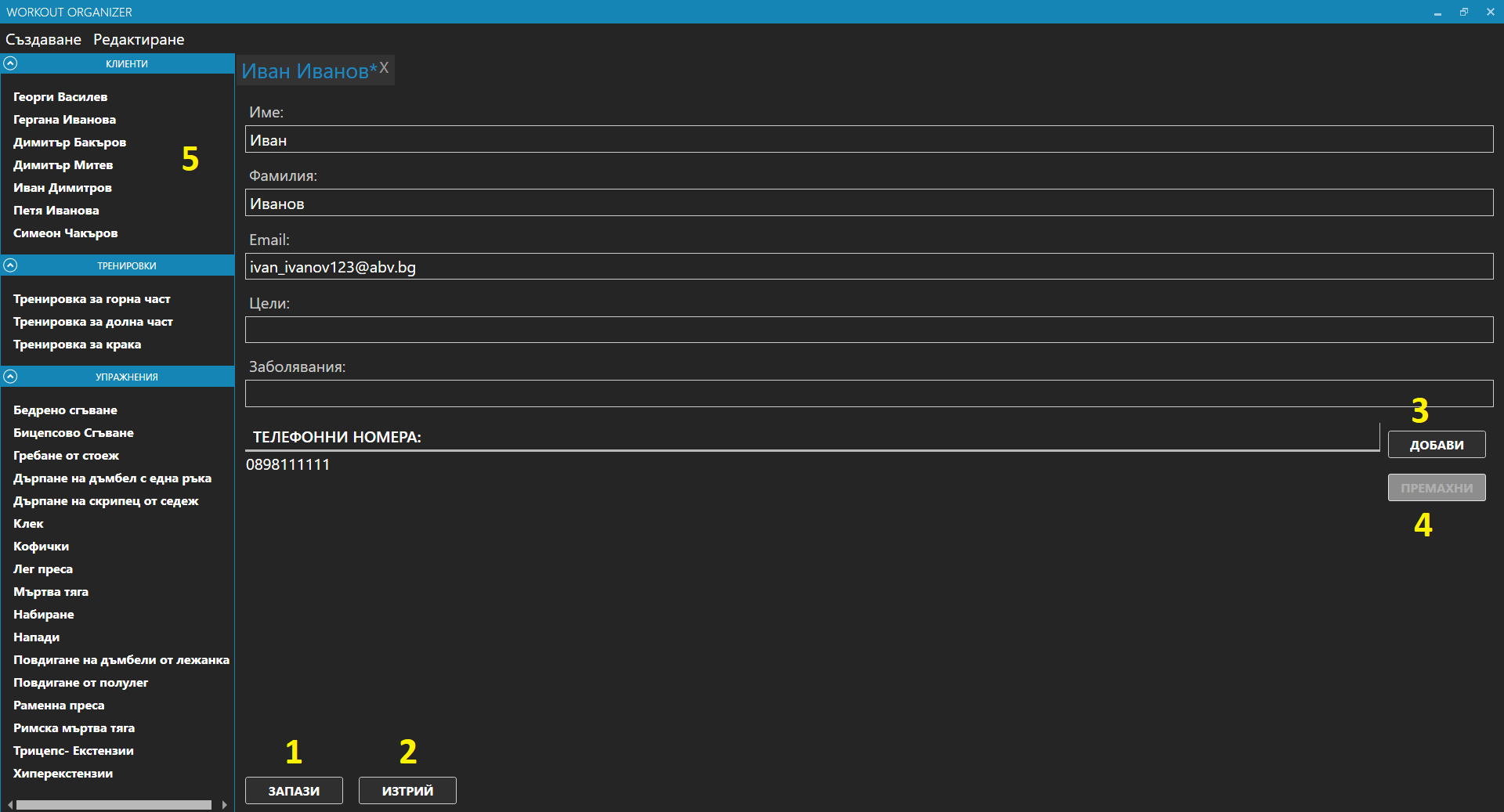
Ако искате да създадете нов клиент, нова тренировка или ново упражнение, изберете менюто Създаване. То се намира в горният ляв ъгъл на прозореца. След като кликнете върху менюто, се отваря падащо меню с подменюта. Фиг. 40.



Фиг. 40 Избор на меню- Създаване

## Създаване на нов клиент

За да създадете нов клиент, натиснете менюто Създаване, а след това от подменюто изберете „Нов клиент“. След като сте избрали това меню, в прозореца се появяват полета за въвеждане на информация за нов клиент. Фиг.41



Фиг. 41 Създаване на нов клиент

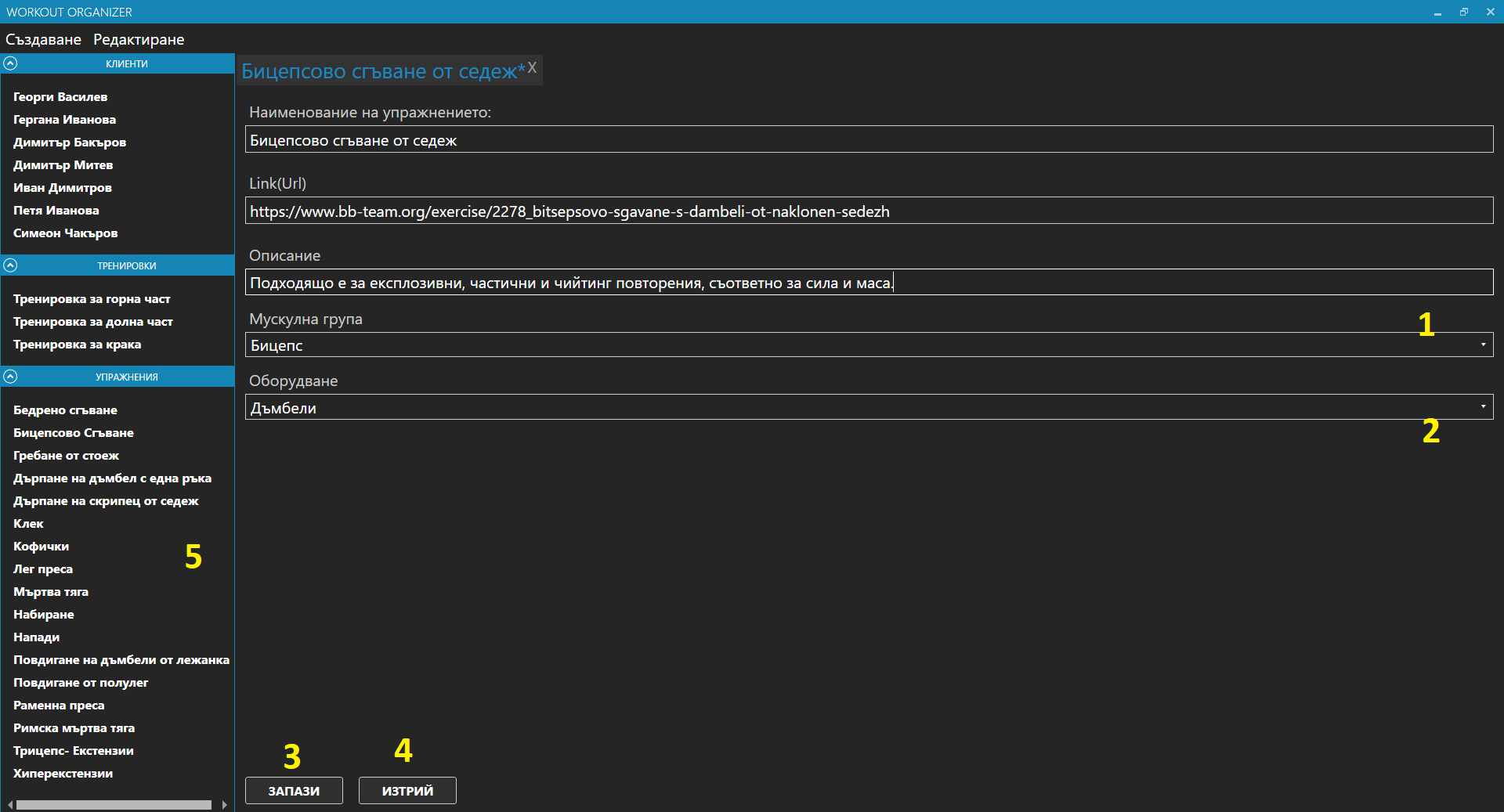
За да добавите телефонен номер на клиент, натиснете бутон **3** – „Добави“. Ако искате да премахнете телефонен номер, селектирайте номерът, който искате да премахнете и натиснете бутон **4**- „Премахни“. За да запазите промените натиснете бутон **1**-„Запази“. Ако сте запазили промените, имената на клиента ще бъдат добавени в навигационното меню **5**- „Клиенти“. Ако искате да изтриете създаденият от вас клиент, натиснете бутон **2**- „Изтрий“.

## Създаване на ново упражнение

За да създадете ново упражнение, натиснете менюто Създаване, а след това Ново упражнение. След като сте избрали това меню, в прозореца се появяват полета за въвеждане на информация за нов упражнение. Фиг.42

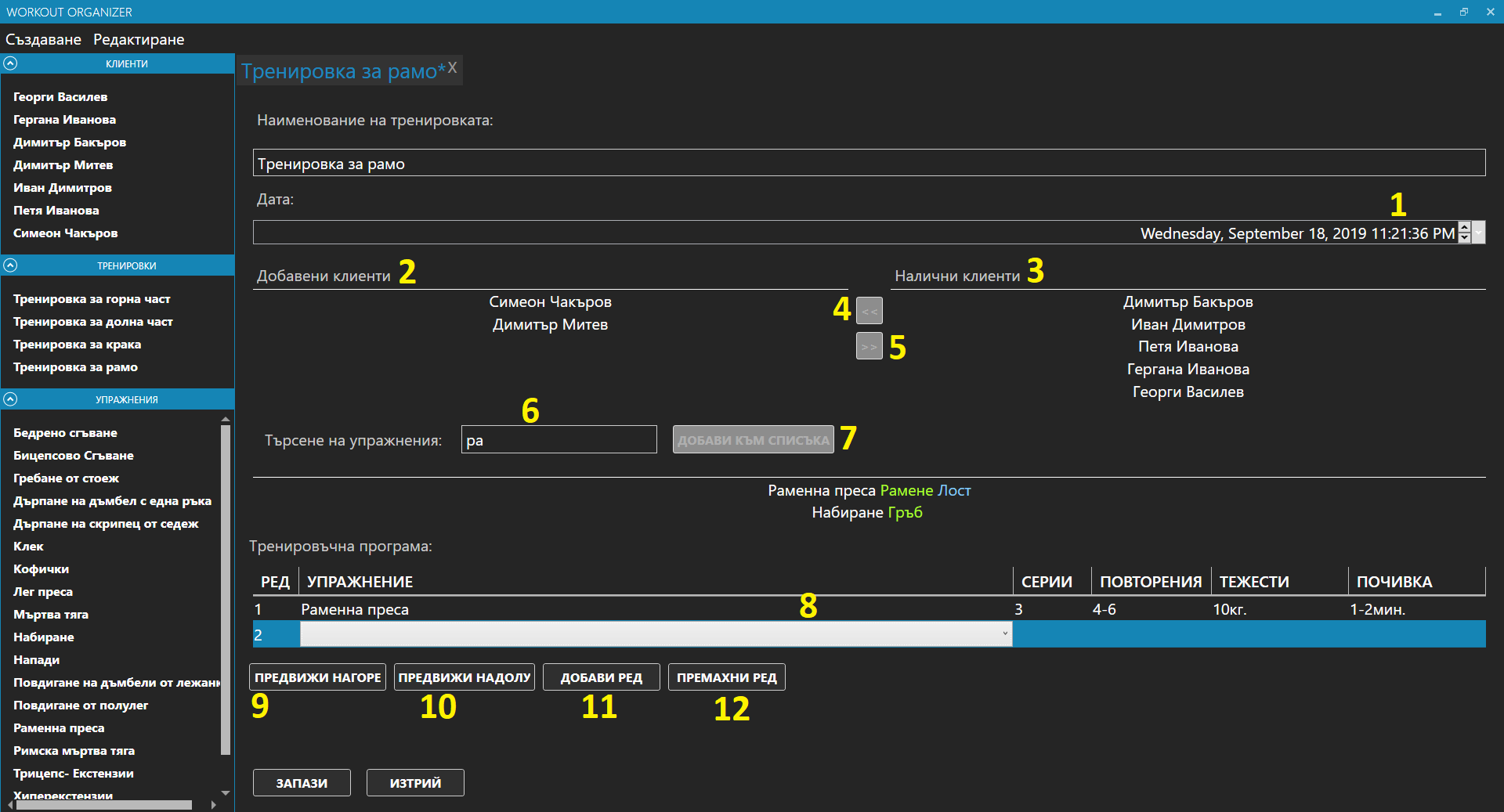
За да изберете мускулна група за даденото упражнение, кликнете върху падащото меню **1** и селектирайте избраната от вас мускулна група. Изборът на оборудване се извършва по същият начин, чрез кликане върху падащото меню **2.** За да запазите промените натиснете бутон **3**-„Запази“. Ако сте запазили промените, наименованието на упражнението ще бъде добавено в навигационното меню **5**- „Упражнения“. Ако искате да изтриете създаденото от вас упражнение, натиснете бутон **4**- „Изтрий“.

Фиг.42 Създаване на ново упражнение



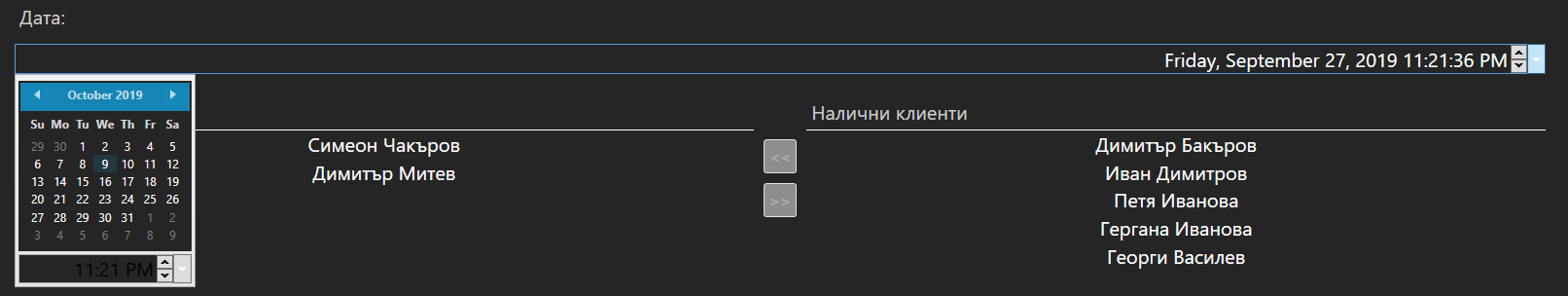
## Създаване на нова тренировка

За да създадете нова тренировка, натиснете менюто Създаване, а след това Нова тренировка. След като сте избрали това меню, в прозореца се появяват полета за въвеждане на информация за нова тренировка. Фиг.43

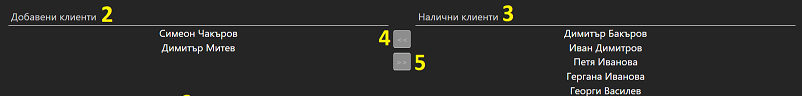


Фиг. 43 Създаване на нова тренировка

**Избор на дата и час на тренировка:**

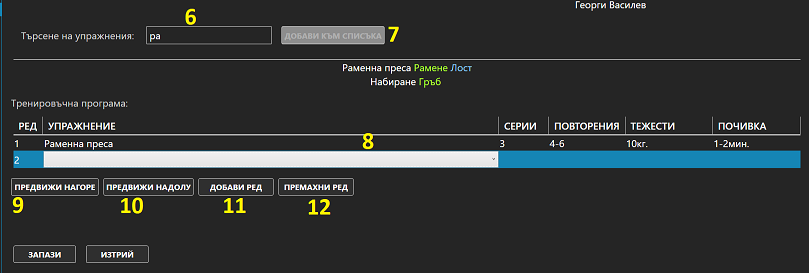
За да изберете дата и час на тренировката, кликнете върху падащото меню **1,** след което, на екрана се появява календар, от който може да изберете желаната от вас дата. Също така можете да използвате и стрелките(нагоре/надолу), за да изберете дата. Фиг. 44

Фиг. 44 Избор на дата и час на тренировка

**Добавяне/Премахване на клиенти от тренировка:**

Фиг.45 добавяне/премахване на клиенти от тренировка

За да добавите клиент към тренировката, селектирайте клиент от списъка с налични клиенти **3,** а след това натиснете бутонът **4** -„<<“. За да премахнете клиент, селектирайте клиента от списъка с добавени клиенти **2**, а след това натиснете бутон **5**- „>>“. Фиг. 45

**Съставяне на тренировъчна програма**

Фиг.46 Съставяне на тренировъчна програма

* Търсене и добавяне на упражнение към тренировъчната програма

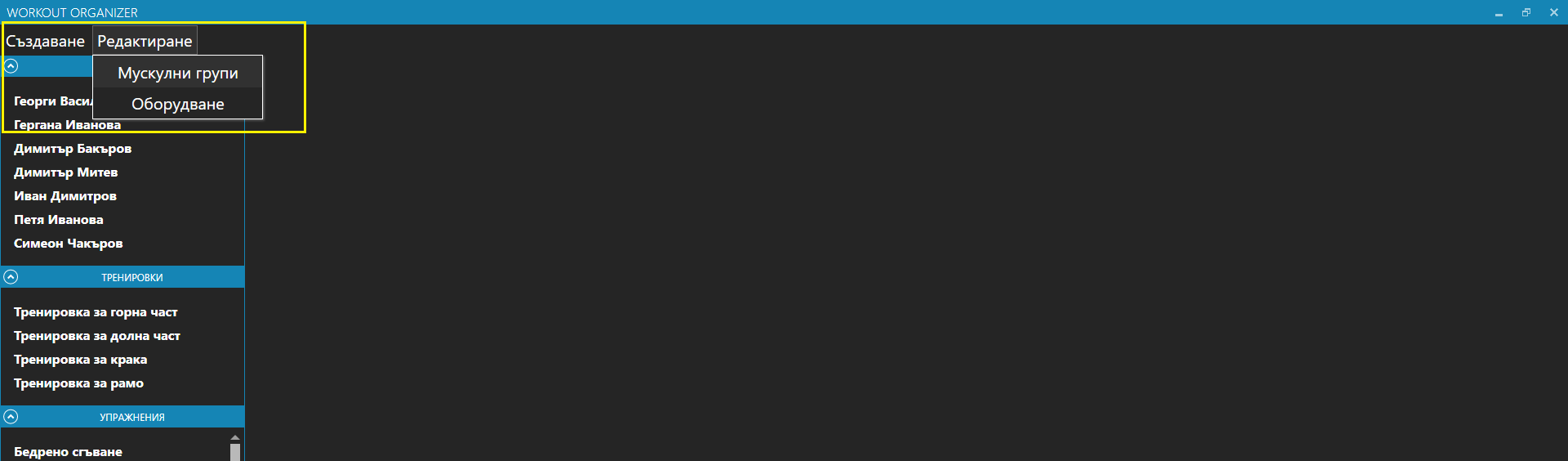
За да търсите упражнение по неговото име, започнете да изписвате името му в текстовата кутия **6**. Упражненията с имена, които съвпадат с изписаното от вас име, се появяват отдолу. За да добавите избраното от вас упражнение, го селектирайте, след което натиснете бутон **7**- „Добави към списъка“.

* Можете да добавите ред към тренировъчната програмата, като натиснете бутон **11- „Добави ред“**. След това от падащото меню **8** изберете упражнение.
* За да премахнете ред от тренировъчната програма, селектирайте избраният от вас ред и натиснете бутон **12** – „Премахни ред“
* Ако искате да промените подредбата на редовете в тренировъчната програма, селектирайте реда, който искате да предвижите и използвайте бутон **9** –„Предвижи нагоре“, за да го предвижите нагоре и съответно, бутон **10** – „Предвижи надолу“, за да го предвижите надолу.

За да запазите промените натиснете бутон „Запази“. Ако сте запазили промените, наименованието на тренировката ще бъде добавено в навигационното меню „Тренировки“. Ако искате да изтриете създаденото от вас упражнение, натиснете бутон „Изтрий“.

## Меню- Редактиране

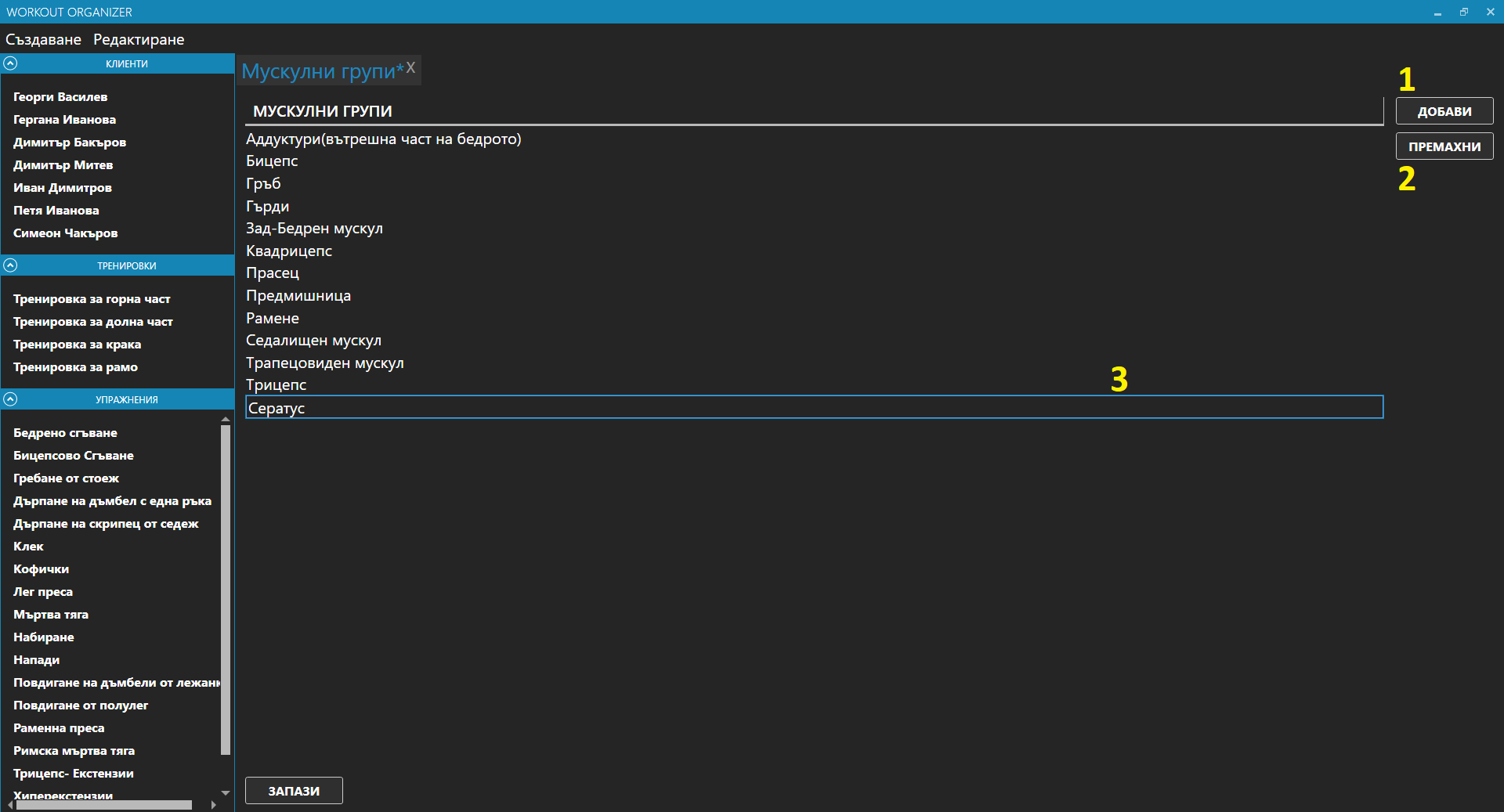
Ако искате да редактирате, добавите или премахнете мускулна група или оборудване, изберете менюто Редактиране. То се намира в горният ляв ъгъл на прозореца. След като кликнете върху менюто се отваря падащо меню с подменюта. Фиг. 47.



Фиг. 47 Меню- Редактиране

## Редактиране, добавяне и премахване на Мускулни групи

Ако искате да редактирате, добавите или премахнете мускулна група, изберете менюто *Редактиране* и след това изберете подменюто *Мускулни групи*(Фиг. 47). След като сте кликнаки върху подменюто, в главното поле на прозореца се показва списък с всички мускулни групи Фиг. 48.



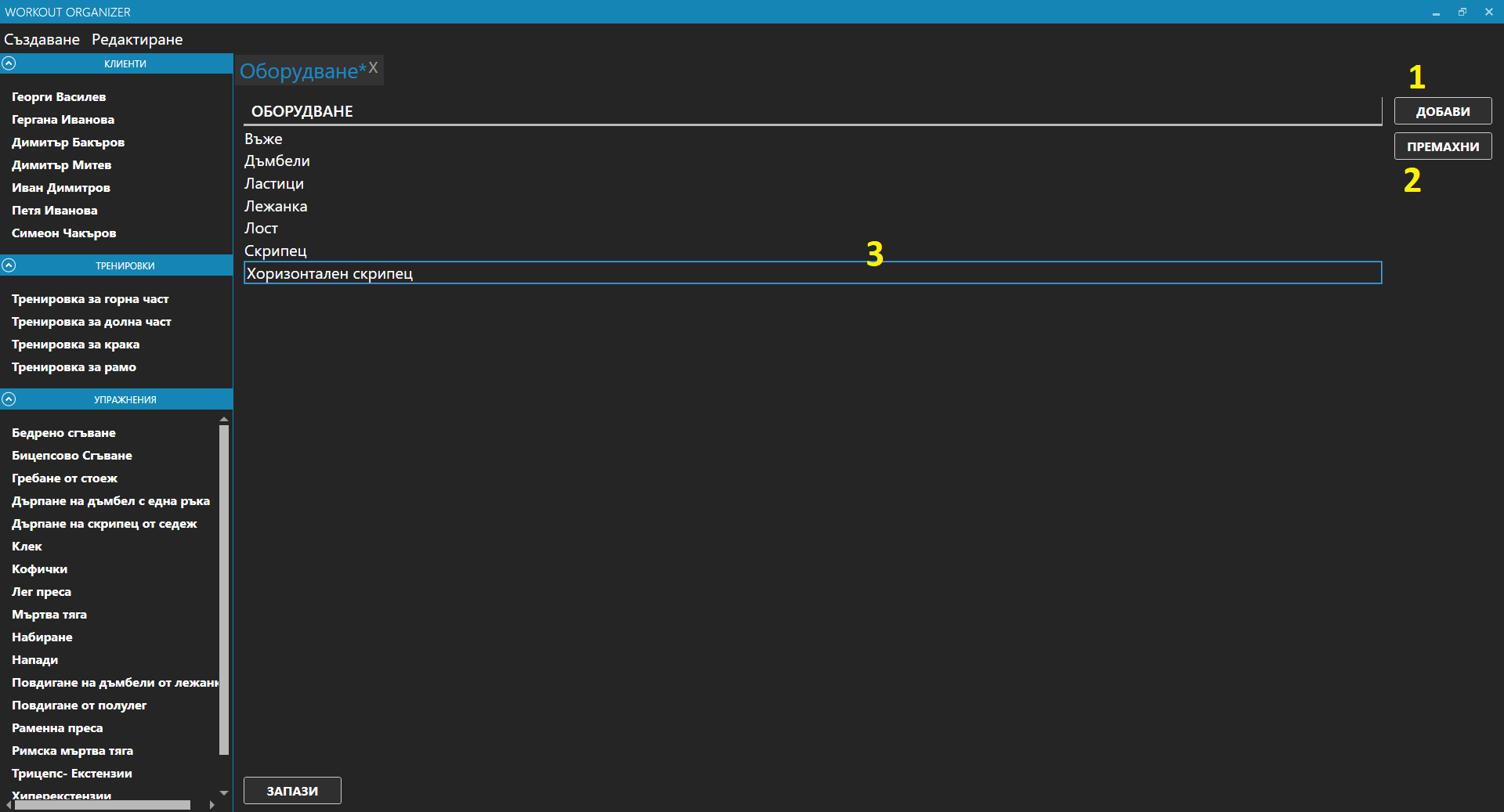
Фиг. 48 Мускулни групи

* Добавянето на мускулна група става чрез натискане на бутон **1- „Добави“.** След натискане на бутона се появява текстова кутия **3** , в която можете да въведете наименованието на мускулната група.
* За да премахнете мускулна група, селектирайте избраната от вас мускулна група от списъка, след което натиснете бутон **2-„Премахни“.**

За да запазите промените натиснете бутонът- **„Запази“**

## Редактиране, добавяне и премахване на Оборудване

Ако искате да редактирате, добавите или премахнете оборудване, изберете менюто *Редактиране* и след това изберете подменюто *Оборудване*(Фиг. 47). След като сте кликнаки върху подменюто, в главното поле на прозореца се показва списък с всички оборудвания Фиг. 49.



Фиг. 49 Оборудване

* Добавянето на оборудване става чрез натискане на бутон **1- „Добави“.** След натискане на бутона се появява текстова кутия **3** , в която можете да въведете наименованието на оборудването.
* За да премахнете оборудване, селектирайте избраното от вас оборудване от списъка, след което натиснете бутон **2-„Премахни“.**

За да запазите промените натиснете бутонът- **„Запази“**

# **Глава 7.Заключение и бъдещо развитие**

В настоящата дипломна се разработва обектно ориентирано десктоп приложение за организация на тренировки, клиенти и упражнения. Насочено е към треньорите като потребители, лесно и удобно да организират своите клиенти, да съставят и подходящи упражнения, спрямо техните потребности и желания. Системата е проектирана на български език.

Системата е създадена като са удовлетворени всички изисквания и целите са изпълнени.

В сравнение с подобни съществуващи решения, системата притежава по-голяма част от техните предимства, а именно:

* Опростен и добре структуриран интерфейс
* Създаване и управление на клиенти
* Възможност за добавяне на персонализирани упражнения
* Създаване и управление на тренировъчни планове

Системата има следните възможности за бъдещо развитие:

* Добавяне на допълнителни езици (напр. немски, руски, английски)
* Добавяне на модул за известяване на клиентите за предстояща тренировка
* Добавяне на потребители, потребителска регистрация и вход.
* Добавяне на календар с предстоящи събития и тренировки на потребителя.
* Добавяне на допълнителни данни за клиент(напр. снимки, телесни измерения и др.)
* Добавяне на възможност за следене на прогреса на даден клиент.
* Добавяне на отделен модул за потребителски вход на клиент.
* Създаване на мобилно приложение.

# Глава 8. Използвана литература

[1] Jefit- <https://www.jefit.com>

[2] MyWorkoutPlan- <https://play.google.com/store/apps/details?id=com.myworkoutplan.myworkoutplan&hl=bg>

[3] PTminder- <https://www.ptminder.com/>

[4] Същност на концептуалния модел на данните- <http://tuj.asenevtsi.com/DB2007/BD54.htm>

[5] Windows Presentation Foundation- <https://docs.microsoft.com/en-us/dotnet/framework/wpf/>

[6] XAML- <https://bg.wikipedia.org/wiki/XAML>

[7] Microsoft Visual Studio- <https://bg.wikipedia.org/wiki/Microsoft_Visual_Studio>

[8] Prism- <https://prismlibrary.github.io/index.html>

[9] Event Aggregator- <https://prismlibrary.github.io/docs/event-aggregator.html>

[10] ORM- <http://zultimate1.com/2671096>

[11] Entity Framework- <https://docs.microsoft.com/en-us/ef/>

[12] Microsoft SQL Server- <https://softuni.bg/blog/history-of-ms-sql-server>

[13] GitHub- <https://bg.wikipedia.org/wiki/GitHub>

[14] MahApps.Metro- <https://mahapps.com/>

[15] Model-View-ViewModel(MVVM)- <http://www.tasheva.info/PS/Upr3_PS_2016.pdf>

[16] An MVVM App- <https://www.appcoda.com/mvvm-vs-mvc/>

[17] Dependency Injection- <https://gatakka.eu/?p=1704>

[18] Autofac- <https://autofac.org/>

[19] Лабораторно упражнение №7- Тестване на програмни модули

# Обем на чертежите

## Приложение 1: Протоколи от проведените тестове

За всеки един обект първоначално се тестват произволно избрани данни от диапазона на правилния клас. След това данните от неправилните класове. Полученият резултат се записва в последната колона като се посочва и конкретния текст на съобщението, ако има такова.

**Резултати от тестове на модул за въвеждане на данни за нов клиент Таблица1.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование на тестваното поле** | **Правилен/неправилен клас** | **Тествана стойност** | **Получен резултат** |
| 1 | Наименование | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане. |
| 2 | неправилен | Опит за въвеждане на повече от 50 символа | След 50 символ следва съобщение „Името трябва да е с дължина между 1 и 50 символа.“ |
| 3 | неправилен | Опит за въвеждане на специален символ | Следва съобщение „Симовлът не е позволен“ |
| 4 | неправилен | Празно поле | Селдва съобщение „Полето е задължително“ |
| 5 | Фамилия | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 6 | неправилен | Опит за въвеждане на специален символ | Следва съобщение „Симовлът не е позволен“ |
| 7 | неправилен | Опит за въвеждане на повече от 50 символа | След 50 символ следва съобщение „Името трябва да е с максимална дължина- 50 символа“ |
| 8 | Email | правилен | 20 символа, валиден e-mail адрес | Изпълнява се по време на въвеждане |
| 9 | неправилен | Опит за въвеждане на повече от 50 символа | След 50 символ следва съобщение „Полето е с максимална дължина- 50 символа“ |
| 10 | неправилен | Опит за въвеждане на невалиден e-mail адрес | Следва съобщение „Невалиден Email адрес“ |
| 11 | Цели | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 12 | неправилен | Опит за въвеждане на повече от 200 символа | Следва съобщение „Полето е с максимална дължина- 200 символа“ |
| 13 | Заболявания | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 14 | неправилен | Опит за въвеждане на повече от 200 символа | Следва съобщение „Полето е с максимална дължина- 200 символа“ |
| 15 | Телефонен номер | правилен | Въвеждане на номер който започва с 089 и съдържа точно 10 цифри | Изпълнява се по време на въвеждане |
| 16 | неправилен | Опит за въвеждане на повече от 10 символа | Следва съобщение „Невалиден телефонен номер“ |
| 17 | неправилен | Опит за въвеждане на по-малко от 10 символа | Следва съобщение „Невалиден телефонен номер“ |
|  | неправилен | Празно поле | Следва съобщение „Полето не може да бъде празно“ |

Таблица1

**Резултати от тестове на модул за въвеждане на данни за ново упражнение Таблица2.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование на тестваното поле** | **Правилен/неправилен клас** | **Тествана стойност** | **Получен резултат** |
| 1 | Наименование | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане. |
| 2 | неправилен | Опит за въвеждане на повече от 70 символа | След 70 символ следва съобщение „Полето е с максимална дължина- 70 символа.“ |
| 3 | неправилен | Празно поле | Селдва съобщение „Полето е задължително“ |
| 4 | Link(Url) | правилен | Въвеждане на валиден url адрес | Изпълнява се по време на въвеждане |
| 5 | неправилен | Опит за въвеждане на невалиден url адрес | Следва съобщение „Невалиден адрес.“ |
| 6 | неправилен | Опит за въвеждане на валиден url адрес с повече от 2083 символа | След 2083 символ следва съобщение „Полето е с максимална дължина- 2083 символа“ |
| 7 | Описание | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 8 | неправилен | Опит за въвеждане на повече от 120 символа | След 120 символ следва съобщение „Полето е с максимална дължина- 120 символа“ |

Таблица2

**Резултати от тестове на модул за въвеждане на данни за нова тренировка Таблица3.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование на тестваното поле** | **Правилен/неправилен клас** | **Тествана стойност** | **Получен резултат** |
| 1 | Наименование | правилен | 20 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане. |
| 2 | неправилен | Опит за въвеждане на повече от 50 символа | След 50 символ следва съобщение „Полето е с максимална дължина- 50 символа.“ |
| 3 | неправилен | Празно поле | Селдва съобщение „Полето е задължително“ |
| 5 | Дата | правилен | Избор на датата 30/12/2019 (произволно избрана дата) | Изпълнява се по време на въвеждане |
| 6 | неправилен | Избор на предходна дата | Следва съобщение „Не може да избирате предходни дати.“ |

Таблица3

**Резултати от тестове на модул за въвеждане на данни за начин на изпълнение на упражнението в дадена тренировка Таблица4.**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **№** | **Наименование на тестваното поле** | **Правилен/неправилен клас** | **Тествана стойност** | **Получен резултат** |
| 1 | Серии | правилен | 9 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане. |
| 2 | неправилен | Опит за въвеждане на повече от 10 символа | След 10 символ следва съобщение „Полето е с максимална дължина- 10 символа.“ |
| 3 | Повторения | правилен | 9 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 4 | неправилен | Опит за въвеждане на повече от 10 символа | След 10 символ следва съобщение „Полето е с максимална дължина- 10 символа.“ |
| 5 | Тежести | правилен | 15 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 6 | неправилен | Опит за въвеждане на повече от 20 символа | След 20 символ следва съобщение „Полето е с максимална дължина- 10 символа.“ |
| 7 | Почивка | правилен | 9 символа (произволно избран брой символи) | Изпълнява се по време на въвеждане |
| 8 | неправилен | Опит за въвеждане на повече от 10 символа | След 10 символ следва съобщение „Полето е с максимална дължина- 10 символа.“ |

Таблица4

## Приложение 2: Текст на програмата

**WorkoutOrganizerDbContext.cs**

namespace WorkoutOrganizer.DataAccess

{

public class WorkoutOrganizerDbContext : DbContext

{

public WorkoutOrganizerDbContext():base("WorkoutOrganizerDb")

{

}

public DbSet<Client> Clients { get; set; }

public DbSet<MusculeGroup> MusculeGroups { get; set; }

public DbSet<Equipment> Equipments { get; set; }

public DbSet<Exercise> Exercises { get; set; }

public DbSet<ClientPhoneNumber> ClientPhoneNumbers { get; set; }

public DbSet<Workout> Workouts { get; set; }

public DbSet<ExerciseSetup> ExerciseSetups { get; set; }

/// <summary>

/// Used to remove the pluralizing of table names convention

/// </summary>

/// <param name="modelBuilder"></param>

protected override void OnModelCreating(DbModelBuilder modelBuilder)

{

base.OnModelCreating(modelBuilder);

modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();

}

}

}

**Client.cs**

public class Client

{

public Client()

{

PhoneNumbers = new Collection<ClientPhoneNumber>();

Workouts = new Collection<Workout>();

}

public int Id { get; set; }

[Required(ErrorMessage = "Полето е задължително")]

[StringLength(50, ErrorMessage = "Името трябва да е с дължина между {2} и {1} символа.", MinimumLength = 2)]

[RegularExpression(@"^[а-яА-Яa-zA-Z''-'\s]{1,100}$", ErrorMessage = "Симовлът не е позволен")]

public string FirstName { get; set; }

[StringLength(50, ErrorMessage = "Името трябва да е с максимална дължина- {1} символа")]

[RegularExpression(@"^[а-яА-Яa-zA-Z''-'\s]{1,100}$", ErrorMessage = "Симовлът не е позволен")]

public string LastName { get; set; }

[EmailAddress(ErrorMessage = "Невалиден Email адрес")]

[StringLength(50, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

[RegularExpression("^[a-zA-Z0-9\_\\.-]+@([a-zA-Z0-9-]+\\.)+[a-zA-Z]{2,6}$", ErrorMessage = "Невалиден Email адрес")]

public string Email { get; set; }

[StringLength(200, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Goals { get; set; }

[StringLength(200, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Injuries { get; set; }

[Timestamp]

public byte[] RowVersion { get; set; }

public ICollection<ClientPhoneNumber> PhoneNumbers { get; set; }

public ICollection<Workout> Workouts { get; set; }

}

**ClientPhoneNumber.cs**

public class ClientPhoneNumber

{

public int Id { get; set; }

[Required(ErrorMessage = "Полето не може да бъде празно")]

[DataType(DataType.PhoneNumber)]

[RegularExpression(@"^\(?(0{1})([8|9{1}])([9|8|7]{1})([0-9]{7})(.\*)$", ErrorMessage = "Невалиден телефонен номер")]

[MaxLength(10 , ErrorMessage = "Дължината на полето не може да е повече от {1} символа")]

public string Number { get; set; }

public int ClientId { get; set; }

public Client Client { get; set; }

}

**Equipment.cs**

namespace WorkoutOrganizer.Model

{

public class Equipment

{

public int Id { get; set; }

[Required(ErrorMessage = "Полето е задължително")]

[StringLength(50, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

[Index(IsUnique = true)]

public string Name { get; set; }

}

}

**Exercise.cs**

namespace WorkoutOrganizer.Model

{

public class Exercise

{

public Exercise()

{

ExerciseSetups = new Collection<ExerciseSetup>();

}

public int Id { get; set; }

[Required(ErrorMessage = "Полето е задължително")]

[StringLength(70, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Name { get; set; }

[StringLength(120, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Description { get; set; }

[Url(ErrorMessage = "Невалиден адрес.")]

[StringLength(2083, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string DemoLink { get; set; }

[Timestamp]

public byte[] RowVersion { get; set; }

public int? MusculeGroupId { get; set; }

public MusculeGroup MusculeGroup { get; set; }

public int? EquipmentId { get; set; }

public Equipment Equipment { get; set; }

public ICollection<ExerciseSetup> ExerciseSetups { get; set; }

}

}

**ExerciseSetup.cs**

namespace WorkoutOrganizer.Model

{

public class ExerciseSetup

{

public int Id { get; set; }

[Required]

public int Position { get; set; }

[StringLength(10, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Sets { get; set; }

[StringLength(10, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Reps { get; set; }

[StringLength(20, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Weight { get; set; }

[StringLength(10, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Rest { get; set; }

public int ExerciseId { get; set; }

public Exercise Exercise { get; set; }

public int WorkoutId { get; set; }

public Workout Workout { get; set; }

}

}

**MusculeGroup.cs**

namespace WorkoutOrganizer.Model

{

public class MusculeGroup

{

public int Id { get; set; }

[Required(ErrorMessage = "Полето е задължително")]

[StringLength(50, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

[Index(IsUnique = true)]

public string Name { get; set; }

}

}

**Workout.cs**

namespace WorkoutOrganizer.Model

{

public class Workout

{

public Workout()

{

Clients = new Collection<Client>();

ExerciseSetups = new Collection<ExerciseSetup>();

}

public int Id { get; set; }

[Required(ErrorMessage ="Полето е задължително")]

[StringLength(50, ErrorMessage = "Полето е с максимална дължина- {1} символа")]

public string Title { get; set; }

public DateTime Date { get; set; }

[Timestamp]

public byte[] RowVersion { get; set; }

public ICollection<Client> Clients { get; set; }

public ICollection<ExerciseSetup> ExerciseSetups { get; set; }

}

}

**LookupItem.cs**

namespace WorkoutOrganizer.Model

{

public class LookupItem

{

public int Id { get; set; }

public string DisplayMember { get; set; }

}

public class NullLookupItem : LookupItem

{

public new int? Id { get { return null; } }

}

}

**IClientLookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public interface IClientLookupDataService

{

Task<IEnumerable<LookupItem>> GetClientLookupAsync();

}

}

**IEquipmentLookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public interface IEquipmentLookupDataService

{

Task<IEnumerable<LookupItem>> GetEquipmentLookupAsync();

}

}

**IExerciseLookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public interface IExerciseLookupDataService

{

Task<IEnumerable<LookupItem>> GetExerciseLookupAsync();

}

}

**IMusculeGroupLookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public interface IMusculeGroupLookupDataService

{

Task<IEnumerable<LookupItem>> GetMusculeGroupLookupAsync();

}

}

**IWorkoutLookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public interface IWorkoutLookupDataService

{

Task<IEnumerable<LookupItem>> GetWorkoutLookupAsync();

}

}

**LookupDataService.cs**

namespace WorkoutOrganizer.UI.Data.Lookups

{

public class LookupDataService : IClientLookupDataService, IWorkoutLookupDataService, IMusculeGroupLookupDataService, IExerciseLookupDataService, IEquipmentLookupDataService

{

private Func<WorkoutOrganizerDbContext> \_contextCreator;

public LookupDataService(Func<WorkoutOrganizerDbContext> contextCreator)

{

\_contextCreator = contextCreator;

}

public async Task<IEnumerable<LookupItem>> GetClientLookupAsync()

{

using (var ctx = \_contextCreator())

{

return await ctx.Clients.AsNoTracking()

.Select(f =>

new LookupItem

{

Id = f.Id,

DisplayMember = f.FirstName + " " + f.LastName

})

.OrderBy(n => n.DisplayMember)

.ToListAsync();

}

}

public async Task<IEnumerable<LookupItem>> GetWorkoutLookupAsync()

{

using (var ctx = \_contextCreator())

{

return await ctx.Workouts.AsNoTracking()

.Select(w =>

new LookupItem

{

Id = w.Id,

DisplayMember = w.Title

})

.OrderBy(n => n.DisplayMember)

.ToListAsync();

}

}

public async Task<IEnumerable<LookupItem>> GetMusculeGroupLookupAsync()

{

using (var ctx = \_contextCreator())

{

return await ctx.MusculeGroups.AsNoTracking()

.Select(f =>

new LookupItem

{

Id = f.Id,

DisplayMember = f.Name

})

.OrderBy(n => n.DisplayMember)

.ToListAsync();

}

}

public async Task<IEnumerable<LookupItem>> GetExerciseLookupAsync()

{

using (var ctx = \_contextCreator())

{

return await ctx.Exercises.AsNoTracking()

.Select(f =>

new LookupItem

{

Id = f.Id,

DisplayMember = f.Name

})

.OrderBy(n => n.DisplayMember)

.ToListAsync();

}

}

public async Task<IEnumerable<LookupItem>> GetEquipmentLookupAsync()

{

using (var ctx = \_contextCreator())

{

return await ctx.Equipments.AsNoTracking()

.Select(e =>

new LookupItem

{

Id = e.Id,

DisplayMember = e.Name

})

.OrderBy(n => n.DisplayMember)

.ToListAsync();

}

}

}

}

**ClientRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class ClientRepository : GenericRepository<Client,WorkoutOrganizerDbContext>, IClientRepository

{

public ClientRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public override async Task<Client> GetByIdAsync(int clientId)

{

return await Context.Clients.Include(c => c.PhoneNumbers).SingleAsync(c => c.Id == clientId);

}

public void RemovePhoneNumber(ClientPhoneNumber model)

{

Context.ClientPhoneNumbers.Remove(model);

}

public async Task<bool> HasWorkoutsAsync(int clientId)

{

return await Context.Workouts.AsNoTracking()

.Include(wc => wc.Clients)

.AnyAsync(w => w.Clients.Any(c => c.Id == clientId));

}

}

}

**EquipmentRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class EquipmentRepository : GenericRepository<Equipment, WorkoutOrganizerDbContext>, IEquipmentRepository

{

public EquipmentRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public async Task<bool> IsReferencedByExerciseAsync(int equipmentId)

{

return await Context.Exercises.AsNoTracking()

.AnyAsync(e => e.EquipmentId == equipmentId);

}

}

}

**ExerciseRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class ExerciseRepository : GenericRepository<Exercise, WorkoutOrganizerDbContext>, IExerciseRepository

{

public ExerciseRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public async Task<bool> IsReferencedByExerciseSetup(int exerciseId)

{

return await Context.ExerciseSetups.AsNoTracking()

.AnyAsync(ex => ex.ExerciseId == exerciseId);

}

}

}

**ExerciseSetupRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class ExerciseSetupRepository : GenericRepository<ExerciseSetup, WorkoutOrganizerDbContext>, IExerciseSetupRepository

{

public ExerciseSetupRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public async Task<IEnumerable<ExerciseSetup>> GetAllAsync(int workoutId)

{

return await Context.Set<ExerciseSetup>()

.Where(e => e.WorkoutId == workoutId)

.Include(ex => ex.Exercise)

.OrderBy(p => p.Position)

.ToListAsync();

}

}

}

**GenericRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class GenericRepository<TEntity, TContext> : IGenericRepository<TEntity>

where TEntity : class

where TContext : DbContext

{

protected readonly TContext Context;

protected GenericRepository(TContext context)

{

this.Context = context;

}

public void Add(TEntity model)

{

Context.Set<TEntity>().Add(model);

}

public virtual async Task<IEnumerable<TEntity>> GetAllAsync()

{

return await Context.Set<TEntity>().ToListAsync();

}

public virtual async Task<TEntity> GetByIdAsync(int Id)

{

return await Context.Set<TEntity>().FindAsync(Id);

}

public bool HasChanges()

{

return Context.ChangeTracker.HasChanges();

}

public void Remove(TEntity model)

{

Context.Set<TEntity>().Remove(model);

}

public async Task SaveAsync()

{

await Context.SaveChangesAsync();

}

}

}

**IClientRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IClientRepository : IGenericRepository<Client>

{

void RemovePhoneNumber(ClientPhoneNumber model);

Task<bool> HasWorkoutsAsync(int clientId);

}

}

**IEquipmentRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IEquipmentRepository : IGenericRepository<Equipment>

{

Task<bool> IsReferencedByExerciseAsync(int musculeGroupId);

}

}

**IExerciseRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IExerciseRepository : IGenericRepository<Exercise>

{

Task<bool> IsReferencedByExerciseSetup(int exerciseId);

}

}

**IExerciseSetupRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IExerciseSetupRepository : IGenericRepository<ExerciseSetup>

{

Task<IEnumerable<ExerciseSetup>> GetAllAsync(int workoutId);

}

}

**IGenericRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IGenericRepository<T>

{

Task<T> GetByIdAsync(int clientId);

Task<IEnumerable<T>> GetAllAsync();

Task SaveAsync();

bool HasChanges();

void Add(T model);

void Remove(T model);

}

}

**IMusculeGroupRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IMusculeGroupRepository : IGenericRepository<MusculeGroup>

{

Task<bool> IsReferencedByExerciseAsync(int musculeGroupId);

}

}

**IWorkoutRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public interface IWorkoutRepository : IGenericRepository<Workout>

{

Task<List<Client>> GetAllClientsAsync();

Task<List<Exercise>> GetAllExercisesAsync();

Task ReloadClientAsync(int clientId);

Task ReloadExerciseAsync(int exerciseId);

Task<List<ExerciseSetup>> GetAllWorkoutSetupsAsync(int workoutId);

}

}

**MusculeGroupRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class MusculeGroupRepository : GenericRepository<MusculeGroup, WorkoutOrganizerDbContext>,

IMusculeGroupRepository

{

public MusculeGroupRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public async Task<bool> IsReferencedByExerciseAsync(int musculeGroupId)

{

return await Context.Exercises.AsNoTracking()

.AnyAsync(e => e.MusculeGroupId == musculeGroupId);

}

}

}

**WorkoutRepository.cs**

namespace WorkoutOrganizer.UI.Data.Repositories

{

public class WorkoutRepository : GenericRepository<Workout, WorkoutOrganizerDbContext>, IWorkoutRepository

{

public WorkoutRepository(WorkoutOrganizerDbContext context) : base(context)

{

}

public async override Task<Workout> GetByIdAsync(int Id)

{

return await Context.Workouts

.Include(w => w.Clients)

.Include(e => e.ExerciseSetups)

.SingleAsync(w => w.Id == Id);

}

public async Task<List<Client>> GetAllClientsAsync()

{

return await Context.Set<Client>()

.ToListAsync();

}

public async Task<List<Exercise>> GetAllExercisesAsync()

{

return await Context.Set<Exercise>()

.Include(m => m.MusculeGroup)

.Include(m => m.Equipment)

.ToListAsync();

}

public async Task ReloadClientAsync(int clientId)

{

var dbEntityEntry = Context.ChangeTracker.Entries<Client>()

.SingleOrDefault(db => db.Entity.Id == clientId);

if(dbEntityEntry != null)

{

await dbEntityEntry.ReloadAsync();

}

}

public async Task ReloadExerciseAsync(int exerciseId)

{

var dbEntityEntry = Context.ChangeTracker.Entries<Exercise>()

.SingleOrDefault(db => db.Entity.Id == exerciseId);

if (dbEntityEntry != null)

{

await dbEntityEntry.ReloadAsync();

}

}

public async Task<List<ExerciseSetup>> GetAllWorkoutSetupsAsync(int workoutId)

{

return await Context.Set<ExerciseSetup>()

.Where(e => e.WorkoutId == workoutId)

.Include(ex => ex.Exercise)

.ToListAsync();

}

}

}

**AfterCollectionSavedEvent.cs**

namespace WorkoutOrganizer.UI.Event

{

public class AfterCollectionSavedEvent : PubSubEvent<AfterCollectionSavedEventArgs>

{

}

public class AfterCollectionSavedEventArgs

{

public string ViewModelName { get; set; }

}

}

**AfterDetailClosedEvent.cs**

namespace WorkoutOrganizer.UI.Event

{

public class AfterDetailClosedEvent : PubSubEvent<AfterDetailClosedEventArgs>

{

}

public class AfterDetailClosedEventArgs

{

public int Id { get; set; }

public string ViewModelName { get; set; }

}

}

**AfterDetailDeletedEvent.cs**

namespace WorkoutOrganizer.UI.Event

{

public class AfterDetailDeletedEvent : PubSubEvent<AfterDetailDeletedEventArgs>

{

}

public class AfterDetailDeletedEventArgs

{

public int Id { get; set; }

public string ViewModelName { get; set; }

}

}

**AfterDetailSavedEvent.cs**

namespace WorkoutOrganizer.UI.Event

{

public class AfterDetailSavedEvent : PubSubEvent<AfterDetailSavedEventArgs>

{

}

public class AfterDetailSavedEventArgs

{

public int Id { get; set; }

public string DisplayMember { get; set; }

public string ViewModelName { get; set; }

}

}

**OpenDetailViewEvent.cs**

namespace WorkoutOrganizer.UI.Event

{

public class OpenDetailViewEvent : PubSubEvent<OpenDetailViewEventArgs>

{

}

public class OpenDetailViewEventArgs

{

public int Id { get; set; }

public string ViewModelName { get; set; }

}

}

**Bootstrapper.cs**

namespace WorkoutOrganizer.UI.Startup

{

public class Bootstrapper

{

public IContainer Bootstrap()

{

var builder = new ContainerBuilder();

builder.RegisterType<WorkoutOrganizerDbContext>().AsSelf();

builder.RegisterType<EventAggregator>().As<IEventAggregator>().SingleInstance();

builder.RegisterType<MessageDialogService>().As<IMessageDialogService>();

builder.RegisterType<MainWindow>().AsSelf();

builder.RegisterType<MainViewModel>().AsSelf();

builder.RegisterType<NavigationViewModel>().As<INavigatioViewModel>();

builder.RegisterType<ClientDetailViewModel>()

.Keyed<IDetailViewModel>(nameof(ClientDetailViewModel));

builder.RegisterType<WorkoutDetailViewModel>()

.Keyed<IDetailViewModel>(nameof(WorkoutDetailViewModel));

builder.RegisterType<MusculeGroupDetailViewModel>()

.Keyed<IDetailViewModel>(nameof(MusculeGroupDetailViewModel));

builder.RegisterType<EquipmentDetailViewModel>()

.Keyed<IDetailViewModel>(nameof(EquipmentDetailViewModel));

builder.RegisterType<ExerciseDetailViewModel>()

.Keyed<IDetailViewModel>(nameof(ExerciseDetailViewModel));

builder.RegisterType<LookupDataService>().AsImplementedInterfaces();

builder.RegisterType<ClientRepository>().As<IClientRepository>();

builder.RegisterType<WorkoutRepository>().As<IWorkoutRepository>();

builder.RegisterType<MusculeGroupRepository>().As<IMusculeGroupRepository>();

builder.RegisterType<ExerciseRepository>().As<IExerciseRepository>();

builder.RegisterType<EquipmentRepository>().As<IEquipmentRepository>();

builder.RegisterType<ExerciseSetupRepository>().As<IExerciseSetupRepository>();

return builder.Build();

}

}

}

**MessageDialogService.cs**

namespace WorkoutOrganizer.UI.View.Services

{

public class MessageDialogService : IMessageDialogService

{

private MetroWindow MetroWindow => (MetroWindow)App.Current.MainWindow;

public async Task<MessageDialogResult> ShowOkCancelDialogAsync(string text, string title)

{

var result = await MetroWindow.ShowMessageAsync(title, text, MessageDialogStyle.AffirmativeAndNegative);

return result == MahApps.Metro.Controls.Dialogs.MessageDialogResult.Affirmative

? MessageDialogResult.OK

: MessageDialogResult.Cancel;

}

public async Task ShowInfoDialogAsync(string text)

{

await MetroWindow.ShowMessageAsync("Информация", text);

}

}

public enum MessageDialogResult

{

OK,

Cancel

}

}

**IMessageDialogService.cs**

namespace WorkoutOrganizer.UI.View.Services

{

public interface IMessageDialogService

{

Task<MessageDialogResult> ShowOkCancelDialogAsync(string text, string title);

Task ShowInfoDialogAsync(string text);

}

}

**ClientDetailView.xaml**

<UserControl x:Class="WorkoutOrganizer.UI.View.ClientDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="Име:" Margin="10 10 10 0"/>

<TextBox Grid.Row="1" Text="{Binding Client.FirstName,

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="2" Content="Фамилия:" Margin="10 0"/>

<TextBox Grid.Row="3" Text="{Binding Client.LastName,

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="4" Content="Email:" Margin="10 0"/>

<TextBox Grid.Row="5" Text="{Binding Client.Email, TargetNullValue='',

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="6" Content="Цели:" Margin="10 0"/>

<TextBox Grid.Row="7" Text="{Binding Client.Goals, TargetNullValue='',

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="8" Content="Заболявания:" Margin="10 0"/>

<TextBox Grid.Row="9" Text="{Binding Client.Injuries, TargetNullValue='',

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<DockPanel Grid.Row="10" Margin="10">

<StackPanel DockPanel.Dock="Right">

<Button Content="Добави" Width="125" Margin="10"

Command="{Binding AddPhoneNumberCommand}"/>

<Button Content="Премахни" Width="125" Margin="10"

Command="{Binding RemovePhoneNumberCommand}"/>

</StackPanel>

<DataGrid ItemsSource="{Binding PhoneNumbers}"

SelectedItem="{Binding SelectedPhoneNumber,Mode=TwoWay}"

AutoGenerateColumns="False" RowHeaderWidth="0">

<DataGrid.Columns>

<DataGridTextColumn Header="Телефонни номера:" Width="\*"

Binding="{Binding Number,Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"/>

</DataGrid.Columns>

</DataGrid>

</DockPanel>

<StackPanel Grid.Row="12" Orientation="Horizontal">

<Button Content="Запази" Width="125" Margin="10"

Command="{Binding SaveCommand}"/>

<Button Content="Изтрий" Width="125" Margin="10"

Command="{Binding DeleteCommand}"/>

</StackPanel>

</Grid>

</UserControl>

**EquipmentDetailView.xaml**

<UserControl x:Class="WorkoutOrganizer.UI.View.EquipmentDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<DataGrid ItemsSource="{Binding Equipments}"

SelectedItem="{Binding SelectedEquipment, Mode=TwoWay}"

AutoGenerateColumns="False" RowHeaderWidth="0" Margin="10">

<DataGrid.Columns>

<DataGridTextColumn Header="Оборудване" Width="\*"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Name, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

</DataGrid.Columns>

</DataGrid>

<StackPanel Grid.Column="1" Margin="0 5 5 5">

<Button Content="Добави" Command="{Binding AddCommand}" Margin="5" Width="125"/>

<Button Content="Премахни" Command="{Binding RemoveCommand}" Margin="5" Width="125"/>

</StackPanel>

<StackPanel Grid.Row="1" Orientation="Horizontal">

<Button Content="Запази" Width="125" Margin="10" Command="{Binding SaveCommand}"/>

</StackPanel>

</Grid>

</UserControl>

**ExerciseDetailView.cs**

<UserControl x:Class="WorkoutOrganizer.UI.View.ExerciseDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

mc:Ignorable="d"

d:DesignHeight="300" d:DesignWidth="300">

<Grid Grid.Row="1">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="Наименование на упражнението:" Margin="10 10 10 0"/>

<TextBox Grid.Row="1" Text="{Binding Exercise.Name,

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="2" Content="Link(Url)" Margin="10 10 10 0"/>

<TextBox Grid.Row="3" Text="{Binding Exercise.DemoLink,

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10">

</TextBox>

<Label Grid.Row="4" Content="Описание" Margin="10 10 10 0"/>

<TextBox Grid.Row="5" Text="{Binding Exercise.Description,

Mode=TwoWay,UpdateSourceTrigger=PropertyChanged}" Margin="10 0 10 10"/>

<Label Grid.Row="6" Content="Мускулна група" Margin="10 0"/>

<ComboBox Grid.Row="7" ItemsSource="{Binding MusculeGroups}"

DisplayMemberPath="DisplayMember"

SelectedValuePath="Id"

SelectedValue="{Binding Exercise.MusculeGroupId,Mode=TwoWay}"

Margin="10 0 10 10"/>

<Label Grid.Row="8" Content="Оборудване" Margin="10 0"/>

<ComboBox Grid.Row="9" ItemsSource="{Binding Equipments}"

DisplayMemberPath="DisplayMember"

SelectedValuePath="Id"

SelectedValue="{Binding Exercise.EquipmentId,Mode=TwoWay}"

Margin="10 0 10 10"/>

<StackPanel Grid.Row="11" Orientation="Horizontal">

<Button Content="Запази" Width="125" Margin="10" Command="{Binding SaveCommand}"/>

<Button Content="Изтрий" Width="125" Margin="10" Command="{Binding DeleteCommand}"/>

</StackPanel>

</Grid>

</UserControl>

**MusculeGroupDetailView.xaml**

<UserControl x:Class="WorkoutOrganizer.UI.View.MusculeGroupDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

mc:Ignorable="d"

d:DesignHeight="450" d:DesignWidth="800">

<Grid>

<Grid.RowDefinitions>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="Auto"/>

</Grid.ColumnDefinitions>

<DataGrid ItemsSource="{Binding MusculeGroups}"

SelectedItem="{Binding SelectedMusculeGroup, Mode=TwoWay}"

AutoGenerateColumns="False" RowHeaderWidth="0" Margin="10">

<DataGrid.Columns>

<DataGridTextColumn Header="Мускулни групи" Width="\*"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Name, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

</DataGrid.Columns>

</DataGrid>

<StackPanel Grid.Column="1" Margin="0 5 5 5">

<Button Content="Добави" Command="{Binding AddCommand}" Margin="5" Width="125"/>

<Button Content="Премахни" Command="{Binding RemoveCommand}" Margin="5" Width="125"/>

</StackPanel>

<StackPanel Grid.Row="1" Orientation="Horizontal">

<Button Content="Запази" Width="125" Margin="10" Command="{Binding SaveCommand}"/>

</StackPanel>

</Grid>

</UserControl>

**NavigationView.xaml**

<UserControl x:Class="WorkoutOrganizer.UI.View.NavigationView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

mc:Ignorable="d"

d:DesignHeight="999" d:DesignWidth="500">

<UserControl.Resources>

<Style x:Key="NaviItemContainerStyle" TargetType="ContentPresenter">

<Setter Property="HorizontalAlignment" Value="Left"/>

<Setter Property="Margin" Value="2"/>

</Style>

<DataTemplate x:Key="NaviItemTemplate">

<Button Content="{Binding DisplayMember}"

Command="{Binding OpenDetailViewCommand}"

FontSize="16">

<Button.Template>

<ControlTemplate TargetType="Button">

<Grid x:Name="grid">

<ContentPresenter HorizontalAlignment="{TemplateBinding HorizontalContentAlignment}"

VerticalAlignment="{TemplateBinding VerticalContentAlignment}"/>

</Grid>

<ControlTemplate.Triggers>

<Trigger Property="IsMouseOver" Value="True">

<Setter Property="Cursor" Value="Hand"/>

<Setter Property="FontWeight" Value="Bold"/>

</Trigger>

<Trigger Property="IsPressed" Value="True">

<Setter Property="FontWeight" Value="Normal"/>

</Trigger>

</ControlTemplate.Triggers>

</ControlTemplate>

</Button.Template>

</Button>

</DataTemplate>

</UserControl.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Expander Header="Клиенти" MaxHeight="330" IsExpanded="True">

<ScrollViewer VerticalScrollBarVisibility="Auto"

HorizontalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding Clients}"

ItemContainerStyle="{StaticResource NaviItemContainerStyle}"

ItemTemplate="{StaticResource NaviItemTemplate}"

Margin="10"/>

</ScrollViewer>

</Expander>

<Expander Header="Тренировки" MaxHeight="330" IsExpanded="True" Grid.Row="1">

<ScrollViewer VerticalScrollBarVisibility="Auto"

HorizontalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding Workouts}"

ItemContainerStyle="{StaticResource NaviItemContainerStyle}"

ItemTemplate="{StaticResource NaviItemTemplate}"

Margin="10"/>

</ScrollViewer>

</Expander>

<Expander Header="Упражнения" Grid.Row="2" IsExpanded="True">

<ScrollViewer VerticalScrollBarVisibility="Auto"

HorizontalScrollBarVisibility="Auto">

<ItemsControl ItemsSource="{Binding Exercises}"

ItemContainerStyle="{StaticResource NaviItemContainerStyle}"

ItemTemplate="{StaticResource NaviItemTemplate}"

Margin="10"/>

</ScrollViewer>

</Expander>

</Grid>

</UserControl>

**WorkoutDetailView.cs**

<UserControl x:Class="WorkoutOrganizer.UI.View.WorkoutDetailView"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:local="clr-namespace:WorkoutOrganizer.UI.View"

xmlns:viewModel="clr-namespace:WorkoutOrganizer.UI.ViewModel"

xmlns:view="clr-namespace:WorkoutOrganizer.UI.View"

xmlns:sys="clr-namespace:System;assembly=mscorlib"

xmlns:model="clr-namespace:WorkoutOrganizer.Model;assembly=WorkoutOrganizer.Model"

xmlns:toolkit="http://schemas.xceed.com/wpf/xaml/toolkit"

mc:Ignorable="d"

d:DesignHeight="800" d:DesignWidth="800">

<ScrollViewer VerticalScrollBarVisibility="Auto">

<Grid Margin="10">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Label Content="Наименование на тренировката:" Margin="10 10 10 10"/>

<TextBox Grid.Row="1" Margin="10" Text="{Binding Workout.Title, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<Label Grid.Row="2" Content="Дата:" Margin="10 0"/>

<toolkit:DateTimePicker Margin="10" Grid.Row="3" Value="{Binding Workout.Date, Mode=TwoWay}" />

<Grid Grid.Row="4" Margin="10">

<Grid.Resources>

<DataTemplate DataType="{x:Type model:Client}">

<TextBlock><Run Text="{Binding FirstName}"/> <Run Text="{Binding LastName}"/></TextBlock>

</DataTemplate>

</Grid.Resources>

<Grid.ColumnDefinitions>

<ColumnDefinition/>

<ColumnDefinition Width="Auto"/>

<ColumnDefinition/>

</Grid.ColumnDefinitions>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Label Content="Добавени клиенти"/>

<ListView ScrollViewer.VerticalScrollBarVisibility="Auto" MaxHeight="200" Grid.Row="1" ItemsSource="{Binding AddedClients}"

SelectedItem="{Binding SelectedAddedClient}"/>

<StackPanel Grid.Row="1" Grid.Column="1">

<Button Content="&lt;&lt;" Margin="10" Command="{Binding AddClientCommand}"/>

<Button Content="&gt;&gt;" Margin="10 0 10 0" Command="{Binding RemoveClientCommand}"/>

</StackPanel>

<Label Content="Налични клиенти" Grid.Column="2"/>

<ListView ScrollViewer.VerticalScrollBarVisibility="Auto" MaxHeight="200" Grid.Row="1" ItemsSource="{Binding AvailableClients}"

SelectedItem="{Binding SelectedAvailableClient}" Grid.Column="2"/>

</Grid>

<Grid Grid.Row="5">

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

<RowDefinition/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<Grid>

<Grid.Resources>

<DataTemplate DataType="{x:Type model:Exercise}">

<TextBlock><Run Text="{Binding Name}"/> <Run Text="{Binding MusculeGroup.Name}" Foreground="GreenYellow"/>

<Run Text="{Binding Equipment.Name}" Foreground="LightSkyBlue"/></TextBlock>

</DataTemplate>

</Grid.Resources>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition Height="Auto"/>

</Grid.RowDefinitions>

<StackPanel Orientation="Horizontal" Margin="10">

<Label Margin="10" VerticalContentAlignment="Center" Content="Търсене на упражнения:"/>

<TextBox Margin="10" VerticalContentAlignment="Center" Width="250" Text="{Binding SearchText, UpdateSourceTrigger=PropertyChanged}"/>

<Button Content="Добави към списъка" Width="Auto" Margin="10" Command="{Binding AddToExerciseSetups}"/>

</StackPanel>

<ListView Grid.Row="1"

ScrollViewer.VerticalScrollBarVisibility="Auto"

MaxHeight="200"

Margin="10"

ItemsSource="{Binding FilteredExercises}"

SelectedItem="{Binding SelectedFilteredExercise}"/>

</Grid>

<Label Grid.Row="1" Content="Тренировъчна програма:"/>

<DataGrid Grid.Row="2" ItemsSource="{Binding ExerciseSetups}"

Width="Auto"

Margin="10"

SelectedItem="{Binding SelectedExerciseSetup, Mode=TwoWay}"

AutoGenerateColumns="False">

<DataGrid.Resources>

<CollectionViewSource x:Key="Exercises" Source="{Binding Exercises}"/>

</DataGrid.Resources>

<DataGrid.Columns>

<DataGridTextColumn Header="Ред" Width="Auto" IsReadOnly="True"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Position, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<DataGridComboBoxColumn Header="Упражнение" Width="\*" MinWidth="200"

ItemsSource="{Binding Source={StaticResource Exercises}}"

DisplayMemberPath="DisplayMember"

SelectedValuePath="Id"

SelectedValueBinding="{Binding ExerciseId, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<DataGridTextColumn Header="Серии" Width="Auto" MinWidth="100"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Sets, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<DataGridTextColumn Header="Повторения" Width="Auto" MinWidth="150"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Reps, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<DataGridTextColumn Header="Тежести" Width="Auto" MinWidth="175"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Weight, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

<DataGridTextColumn Header="Почивка" Width="Auto" MinWidth="175"

ElementStyle="{StaticResource DataGridColumnStyle}"

EditingElementStyle="{StaticResource DataGridEditingColumnStyle}"

Binding="{Binding Rest, Mode=TwoWay, UpdateSourceTrigger=PropertyChanged}"/>

</DataGrid.Columns>

</DataGrid>

<StackPanel Grid.Row="3" Margin="0 5 5 5" Orientation="Horizontal">

<Button Content="Предвижи нагоре" Command="{Binding MoveUp}" Margin="5" Width="175"/>

<Button Content="Предвижи надолу" Command="{Binding MoveDown}" Margin="5" Width="180"/>

<Button Content="Добави ред" Command="{Binding AddExerciseSetupCommand}" Margin="5" Width="150"/>

<Button Content="Премахни ред" Command="{Binding RemoveExerciseSetupCommand}" Margin="5" Width="150"/>

</StackPanel>

</Grid>

<StackPanel Grid.Row="7" Orientation="Horizontal">

<Button Content="Запази" Width="125" Margin="10" Command="{Binding SaveCommand}"/>

<Button Content="Изтрий" Width="125" Margin="10" Command="{Binding DeleteCommand}"/>

</StackPanel>

</Grid>

</ScrollViewer>

</UserControl>

**ModelWrapper.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel.DataAnnotations;

using System.Runtime.CompilerServices;

namespace WorkoutOrganizer.UI.Wrapper

{

public class ModelWrapper<T> : NotifyDataErrorInfoBase

{

public ModelWrapper(T model)

{

Model = model;

}

public T Model { get; }

protected virtual TValue GetValue<TValue>([CallerMemberName]string propertyName = null)

{

return (TValue)typeof(T).GetProperty(propertyName).GetValue(Model);

}

protected virtual void SetValue<TValue>(TValue value,

[CallerMemberName]string propertyName = null)

{

typeof(T).GetProperty(propertyName).SetValue(Model,value);

OnPropertyChanged(propertyName);

ValidatePropertyInternal(propertyName, value);

}

private void ValidatePropertyInternal(string propertyName, object currentValue)

{

ClearErrors(propertyName);

ValidateDataAnnotations(propertyName, currentValue);

ValidateCustomErrors(propertyName);

}

private void ValidateDataAnnotations(string propertyName, object currentValue)

{

var context = new ValidationContext(Model) { MemberName = propertyName };

var results = new List<ValidationResult>();

Validator.TryValidateProperty(currentValue, context, results);

foreach (var result in results)

{

AddError(propertyName, result.ErrorMessage);

}

}

private void ValidateCustomErrors(string propertyName)

{

var errors = ValidateProperty(propertyName);

if (errors != null)

{

foreach (var error in errors)

{

AddError(propertyName, error);

}

}

}

protected virtual IEnumerable<string> ValidateProperty(string propertyName)

{

return null;

}

}

}

**NotifyDataErrorInfoBase.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class NotifyDataErrorInfoBase : ViewModelBase, INotifyDataErrorInfo

{

private Dictionary<string, List<string>> \_errorsByPropertyName =

new Dictionary<string, List<string>>();

public bool HasErrors => \_errorsByPropertyName.Any();

public event EventHandler<DataErrorsChangedEventArgs> ErrorsChanged;

public IEnumerable GetErrors(string propertyName)

{

return \_errorsByPropertyName.ContainsKey(propertyName)

? \_errorsByPropertyName[propertyName]

: null;

}

protected virtual void OnErrorsChanged(string propertyName)

{

ErrorsChanged?.Invoke(this, new DataErrorsChangedEventArgs(propertyName));

base.OnPropertyChanged(nameof(HasErrors));

}

protected void AddError(string propertyName, string error)

{

if (!\_errorsByPropertyName.ContainsKey(propertyName))

{

\_errorsByPropertyName[propertyName] = new List<string>();

}

if (!\_errorsByPropertyName[propertyName].Contains(error))

{

\_errorsByPropertyName[propertyName].Add(error);

OnErrorsChanged(propertyName);

}

}

protected void ClearErrors(string propertyName)

{

if (\_errorsByPropertyName.ContainsKey(propertyName))

{

\_errorsByPropertyName.Remove(propertyName);

OnErrorsChanged(propertyName);

}

}

}

}

**ClientPhoneNumberWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class ClientPhoneNumberWrapper : ModelWrapper<ClientPhoneNumber>

{

public ClientPhoneNumberWrapper(ClientPhoneNumber model) : base(model)

{

}

public string Number

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

}

}

**ClientWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class ClientWrapper : ModelWrapper<Client>

{

public ClientWrapper(Client model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public string FirstName

{

get { return GetValue<string>(); }

set

{

SetValue(value);

}

}

public string LastName

{

get { return GetValue<string>(); }

set

{

SetValue(value);

}

}

public string Email

{

get { return GetValue<string>(); }

set

{

SetValue(value);

}

}

public string Goals

{

get { return GetValue<string>(); }

set

{

SetValue(value);

}

}

public string Injuries

{

get { return GetValue<string>(); }

set

{

SetValue(value);

}

}

protected override IEnumerable<string> ValidateProperty(string propertyName)

{

switch (propertyName)

{

case nameof(FirstName):

if (string.Equals(FirstName, "Robot", StringComparison.OrdinalIgnoreCase))

{

yield return "Robots are not valid clients";

}

break;

}

}

}

}

**EquipmentWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class EquipmentWrapper : ModelWrapper<Equipment>

{

public EquipmentWrapper(Equipment model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public string Name

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

}

}

**ExerciseSetupWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class ExerciseSetupWrapper : ModelWrapper<ExerciseSetup>

{

public ExerciseSetupWrapper(ExerciseSetup model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public int Position

{

get { return GetValue<int>(); }

set { SetValue(value); }

}

public string Sets

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public string Reps

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public string Weight

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public string Rest

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public int? WorkoutId

{

get { return GetValue<int?>(); }

set { SetValue(value); }

}

public int? ExerciseId

{

get { return GetValue<int?>(); }

set { SetValue(value); }

}

}

}

**ExerciseWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class ExerciseWrapper : ModelWrapper<Exercise>

{

public ExerciseWrapper(Exercise model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public string Name

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public string Description

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public string DemoLink

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public int? MusculeGroupId

{

get { return GetValue<int?>(); }

set { SetValue(value); }

}

public int? EquipmentId

{

get { return GetValue<int?>(); }

set { SetValue(value); }

}

protected override IEnumerable<string> ValidateProperty(string propertyName)

{

switch (propertyName)

{

case nameof(DemoLink):

Uri uriResult;

bool result = Uri.TryCreate(DemoLink, UriKind.Absolute, out uriResult)

&& (uriResult.Scheme == Uri.UriSchemeHttp || uriResult.Scheme == Uri.UriSchemeHttps);

if (!result)

{

yield return "Url е невалиден.";

}

break;

}

}

}

}

**MusculeGroupWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class MusculeGroupWrapper : ModelWrapper<MusculeGroup>

{

public MusculeGroupWrapper(MusculeGroup model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public string Name

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

}

}

**WorkoutWrapper.cs**

namespace WorkoutOrganizer.UI.Wrapper

{

public class WorkoutWrapper : ModelWrapper<Workout>

{

public WorkoutWrapper(Workout model) : base(model)

{

}

public int Id { get { return Model.Id; } }

public string Title

{

get { return GetValue<string>(); }

set { SetValue(value); }

}

public DateTime Date

{

get { return GetValue<DateTime>(); }

set { SetValue(value); }

}

protected override IEnumerable<string> ValidateProperty(string propertyName)

{

switch (propertyName)

{

case nameof(Date):

if (Date < DateTime.Today)

{

yield return "Не може да избирате предходни дати.";

}

break;

}

}

}

}

**ViewModelBase.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class ViewModelBase : INotifyPropertyChanged

{

public event PropertyChangedEventHandler PropertyChanged;

protected virtual void OnPropertyChanged([CallerMemberName]string propertyName = null)

{

PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));

}

}

}

**ClientDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class ClientDetailViewModel : DetailViewModelBase, IClientDetailViewModel

{

private IClientRepository \_clientRepository;

private ClientWrapper \_client;

private ClientPhoneNumberWrapper \_selectedPhoneNumber;

public ClientDetailViewModel(IClientRepository clientRepository,

IEventAggregator eventAggregator,

IMessageDialogService messageDialogService)

:base(eventAggregator, messageDialogService)

{

\_clientRepository = clientRepository;

AddPhoneNumberCommand = new DelegateCommand(OnAddPhoneNumberExecute);

RemovePhoneNumberCommand = new DelegateCommand(OnRemovePhoneNumberExecute, OnRemovePhoneNumberCanExecute);

PhoneNumbers = new ObservableCollection<ClientPhoneNumberWrapper>();

}

public override async Task LoadAsync(int clientId)

{

var client = clientId > 0

? await \_clientRepository.GetByIdAsync(clientId)

: CreateNewClient();

Id = clientId;

InitializeClient(client);

InitializeClientPhoneNumbers(client.PhoneNumbers);

}

private void InitializeClientPhoneNumbers(ICollection<ClientPhoneNumber> phoneNumbers)

{

foreach (var wrapper in PhoneNumbers)

{

wrapper.PropertyChanged -= ClientPhoneNumberWrapper\_PropertyChanged;

}

PhoneNumbers.Clear();

foreach (var clientPhoneNumber in phoneNumbers)

{

var wrapper = new ClientPhoneNumberWrapper(clientPhoneNumber);

PhoneNumbers.Add(wrapper);

wrapper.PropertyChanged += ClientPhoneNumberWrapper\_PropertyChanged;

}

}

private void ClientPhoneNumberWrapper\_PropertyChanged(object sender, PropertyChangedEventArgs e)

{

if (!HasChanges)

{

HasChanges = \_clientRepository.HasChanges();

}

if (e.PropertyName == nameof(ClientPhoneNumberWrapper.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

}

public ClientWrapper Client

{

get { return \_client; }

set

{

\_client = value;

OnPropertyChanged();

}

}

public ClientPhoneNumberWrapper SelectedPhoneNumber

{

get { return \_selectedPhoneNumber; }

set

{

\_selectedPhoneNumber = value;

OnPropertyChanged();

((DelegateCommand)RemovePhoneNumberCommand).RaiseCanExecuteChanged();

}

}

public ICommand AddPhoneNumberCommand { get; }

public ICommand RemovePhoneNumberCommand { get; }

public ObservableCollection<ClientPhoneNumberWrapper> PhoneNumbers { get; set; }

protected override async void OnSaveExecute()

{

await SaveWithOptimisticConcurrencyAsync(\_clientRepository.SaveAsync,

() =>

{

HasChanges = \_clientRepository.HasChanges();

Id = Client.Id;

RaiseDetailSavedEvent(Client.Id, $"{Client.FirstName} {Client.LastName}");

});

}

protected override bool OnSaveCanExecute()

{

return Client != null

&& !Client.HasErrors

&& PhoneNumbers.All(pn => !pn.HasErrors)

&& HasChanges;

}

protected override async void OnDeleteExecute()

{

if(await \_clientRepository.HasWorkoutsAsync(Client.Id))

{

await MessageDialogService.ShowInfoDialogAsync($"{Client.FirstName} {Client.LastName} не може да бъде изтрит," +

$" понеже е част от поне една тренировка.");

return;

}

var result = await MessageDialogService.ShowOkCancelDialogAsync("Наистина ли искате да изтриете този клиент?", "Въпрос");

if (result == MessageDialogResult.OK)

{

\_clientRepository.Remove(Client.Model);

await \_clientRepository.SaveAsync();

RaiseDetailDeletedEvent(Client.Id);

}

}

private Client CreateNewClient()

{

var client = new Client();

\_clientRepository.Add(client);

return client;

}

private void InitializeClient(Client client)

{

Client = new ClientWrapper(client);

Client.PropertyChanged += (s, e) =>

{

if (!HasChanges)

{

HasChanges = \_clientRepository.HasChanges();

}

if (e.PropertyName == nameof(Client.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

if(e.PropertyName == nameof(Client.FirstName) || e.PropertyName == nameof(Client.LastName))

{

SetTitle();

}

};

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

if (Client.Id == 0)

{

//To trigger the validation

Client.FirstName = "";

}

SetTitle();

}

private void SetTitle()

{

Title = $"{Client.FirstName} {Client.LastName}";

}

private void OnAddPhoneNumberExecute()

{

var newNumber = new ClientPhoneNumberWrapper(new ClientPhoneNumber());

newNumber.PropertyChanged += ClientPhoneNumberWrapper\_PropertyChanged;

PhoneNumbers.Add(newNumber);

Client.Model.PhoneNumbers.Add(newNumber.Model);

newNumber.Number = ""; // Trigger validation :-)

}

private void OnRemovePhoneNumberExecute()

{

SelectedPhoneNumber.PropertyChanged -= ClientPhoneNumberWrapper\_PropertyChanged;

\_clientRepository.RemovePhoneNumber(SelectedPhoneNumber.Model);

PhoneNumbers.Remove(SelectedPhoneNumber);

SelectedPhoneNumber = null;

HasChanges = \_clientRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private bool OnRemovePhoneNumberCanExecute()

{

return SelectedPhoneNumber != null;

}

}

}

**DetailViewModelBase.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public abstract class DetailViewModelBase : ViewModelBase, IDetailViewModel

{

private bool \_hasChanges;

private int \_id;

private string \_title;

protected readonly IMessageDialogService MessageDialogService;

protected readonly IEventAggregator EventAggregator;

public DetailViewModelBase(IEventAggregator eventAggregator,

IMessageDialogService messageDialogService)

{

MessageDialogService = messageDialogService;

EventAggregator = eventAggregator;

SaveCommand = new DelegateCommand(OnSaveExecute, OnSaveCanExecute);

DeleteCommand = new DelegateCommand(OnDeleteExecute);

CloseDetailViewCommand = new DelegateCommand(OnCloseDetailViewExecute);

}

public int Id

{

get { return \_id; }

protected set { \_id = value; }

}

public string Title

{

get { return \_title; }

protected set {

\_title = value;

OnPropertyChanged();

}

}

public abstract Task LoadAsync(int id);

public ICommand SaveCommand { get; private set; }

public ICommand DeleteCommand { get; private set; }

public ICommand CloseDetailViewCommand { get;}

public bool HasChanges

{

get { return \_hasChanges; }

set

{

if (\_hasChanges != value)

{

\_hasChanges = value;

OnPropertyChanged();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

}

}

protected abstract void OnDeleteExecute();

protected abstract bool OnSaveCanExecute();

protected abstract void OnSaveExecute();

protected virtual void RaiseDetailDeletedEvent(int modelId)

{

EventAggregator.GetEvent<AfterDetailDeletedEvent>().Publish(new

AfterDetailDeletedEventArgs

{

Id = modelId,

ViewModelName = this.GetType().Name

});

}

protected virtual void RaiseDetailSavedEvent(int modelId, string displayMember)

{

EventAggregator.GetEvent<AfterDetailSavedEvent>().Publish(new AfterDetailSavedEventArgs

{

Id = modelId,

DisplayMember = displayMember,

ViewModelName = this.GetType().Name

});

}

protected virtual void RaiseCollectionSavedEvent()

{

EventAggregator.GetEvent<AfterCollectionSavedEvent>()

.Publish(new AfterCollectionSavedEventArgs

{

ViewModelName = this.GetType().Name

});

}

protected async virtual void OnCloseDetailViewExecute()

{

if (HasChanges)

{

var result = await MessageDialogService.ShowOkCancelDialogAsync(

"Вие направихте промени. Наистина ли искате да затворите този прозорец?",

"Въпрос");

if(result == MessageDialogResult.Cancel)

{

return;

}

}

EventAggregator.GetEvent<AfterDetailClosedEvent>()

.Publish(new AfterDetailClosedEventArgs

{

Id = this.Id,

ViewModelName = this.GetType().Name

});

}

protected async Task SaveWithOptimisticConcurrencyAsync(Func<Task> saveFunc, Action afterSaveAction)

{

try

{

await saveFunc();

}

catch (DbUpdateConcurrencyException ex) //handle the concurrency exception

{

var databaseValues = ex.Entries.Single().GetDatabaseValues();

if (databaseValues == null)

{

await MessageDialogService.ShowInfoDialogAsync("Записът вече е изтрит от друг потребител.");

RaiseDetailDeletedEvent(Id);

return;

}

var result = await MessageDialogService.ShowOkCancelDialogAsync("Записът вече е променен от някой друг. Натиснете OK, за да запазите вашите промени"

+ ", натиснете Cancel за да презаредите записа от базата данни.", "Въпрос");

if (result == MessageDialogResult.OK)

{

//Update the original values with database-values

var entry = ex.Entries.Single();

entry.OriginalValues.SetValues(entry.GetDatabaseValues());

await saveFunc();

}

else

{

//Reload entity from the database

await ex.Entries.Single().ReloadAsync();

await LoadAsync(Id);

}

};

afterSaveAction();

}

}

}

**EquipmentDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class EquipmentDetailViewModel : DetailViewModelBase

{

private IEquipmentRepository \_equipmentRepository;

private EquipmentWrapper \_selectedEquipment;

public EquipmentDetailViewModel(IEventAggregator eventAggregator, IMessageDialogService messageDialogService,

IEquipmentRepository equipmentRepository)

: base(eventAggregator, messageDialogService)

{

\_equipmentRepository = equipmentRepository;

Title = "Оборудване";

Equipments = new ObservableCollection<EquipmentWrapper>();

AddCommand = new DelegateCommand(OnAddExecute);

RemoveCommand = new DelegateCommand(OnRemoveExecute, OnRemoveCanExecute);

}

public ObservableCollection<EquipmentWrapper> Equipments { get; }

public ICommand AddCommand { get; }

public ICommand RemoveCommand { get; }

public EquipmentWrapper SelectedEquipment

{

get { return \_selectedEquipment; }

set

{

\_selectedEquipment = value;

OnPropertyChanged();

((DelegateCommand)RemoveCommand).RaiseCanExecuteChanged();

}

}

public async override Task LoadAsync(int id)

{

Id = id;

foreach (var wrapper in Equipments)

{

wrapper.PropertyChanged -= Wrapper\_PropertyChanged;

}

Equipments.Clear();

var equipments = await \_equipmentRepository.GetAllAsync();

foreach (var model in equipments)

{

var wrapper = new EquipmentWrapper(model);

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

Equipments.Add(wrapper);

}

}

protected override void OnDeleteExecute()

{

throw new NotImplementedException();

}

protected override bool OnSaveCanExecute()

{

return HasChanges && Equipments.All(m => !m.HasErrors);

}

protected async override void OnSaveExecute()

{

try

{

await \_equipmentRepository.SaveAsync();

HasChanges = \_equipmentRepository.HasChanges();

RaiseCollectionSavedEvent();

}

catch (Exception ex)

{

while (ex.InnerException != null)

{

ex = ex.InnerException;

}

await MessageDialogService.ShowInfoDialogAsync("Грешка при запазването. Информацията ще бъде презаредена. Детайли: " + ex.Message);

await LoadAsync(Id);

}

}

private void Wrapper\_PropertyChanged(object sender, PropertyChangedEventArgs e)

{

if (!HasChanges)

{

HasChanges = \_equipmentRepository.HasChanges();

}

if (e.PropertyName == nameof(EquipmentWrapper.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

}

private bool OnRemoveCanExecute()

{

return SelectedEquipment != null;

}

private async void OnRemoveExecute()

{

var isReferenced = await \_equipmentRepository.IsReferencedByExerciseAsync(SelectedEquipment.Id);

if (isReferenced)

{

await MessageDialogService.ShowInfoDialogAsync($"{SelectedEquipment.Name} не може да бъде премахнато" +

$", понеже се съдържа в поне едно упражнение.");

return;

}

SelectedEquipment.PropertyChanged -= Wrapper\_PropertyChanged;

\_equipmentRepository.Remove(SelectedEquipment.Model);

Equipments.Remove(SelectedEquipment);

SelectedEquipment = null;

HasChanges = \_equipmentRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private void OnAddExecute()

{

var wrapper = new EquipmentWrapper(new Equipment());

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

\_equipmentRepository.Add(wrapper.Model);

Equipments.Add(wrapper);

//trigger the validation

wrapper.Name = "";

}

}

}

**ExerciseDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class ExerciseDetailViewModel : DetailViewModelBase, IExerciseDetailViewModel

{

private ExerciseWrapper \_exercise;

private IExerciseRepository \_exerciseRepository;

private IMusculeGroupLookupDataService \_musculeGroupLookupDataService;

private IEquipmentLookupDataService \_equipmentLookupDataService;

public ExerciseDetailViewModel(IEventAggregator eventAggregator,

IMessageDialogService messageDialogService,

IExerciseRepository exerciseRepository,

IMusculeGroupLookupDataService musculeGroupLookupDataService,

IEquipmentLookupDataService equipmentLookupDataService

)

: base(eventAggregator, messageDialogService)

{

\_exerciseRepository = exerciseRepository;

\_musculeGroupLookupDataService = musculeGroupLookupDataService;

\_equipmentLookupDataService = equipmentLookupDataService;

eventAggregator.GetEvent<AfterCollectionSavedEvent>()

.Subscribe(AfterCollectionSaved);

OpenUrlCommand = new DelegateCommand(OpenUrl);

MusculeGroups = new ObservableCollection<LookupItem>();

Equipments = new ObservableCollection<LookupItem>();

}

public ICommand OpenUrlCommand { get; }

public async override Task LoadAsync(int exerciseId)

{

var exercise = exerciseId > 0

? await \_exerciseRepository.GetByIdAsync(exerciseId)

: CreateNewExercise();

Id = exerciseId;

InitializeExercise(exercise);

await LoadMusculeGroupsLookupAsync();

await LoadEquipmentsLookupAsync();

}

public ObservableCollection<LookupItem> MusculeGroups { get; }

public ObservableCollection<LookupItem> Equipments { get; }

public ExerciseWrapper Exercise

{

get { return \_exercise; }

set

{

\_exercise = value;

OnPropertyChanged();

}

}

protected async override void OnDeleteExecute()

{

var result = await MessageDialogService.ShowOkCancelDialogAsync($"Наистина ли искате да изтриете {Exercise.Name}?", "Въпрос");

if (result == MessageDialogResult.OK)

{

if (await \_exerciseRepository.IsReferencedByExerciseSetup(Exercise.Id))

{

result = await MessageDialogService.ShowOkCancelDialogAsync($"Упражнението {Exercise.Name} е част от поне една тренировъчна програма." +

$" Ако го изтриете, ще бъде премахнато от всички тренировъчни програми." +

$" Натиснете ОК за да го изтриете въпреки това. Натисете Cancel за да отмените изтриването.", "Информация");

if (result == MessageDialogResult.OK)

{

\_exerciseRepository.Remove(Exercise.Model);

await \_exerciseRepository.SaveAsync();

RaiseDetailDeletedEvent(Exercise.Id);

}

else

{

return;

}

}

else

{

\_exerciseRepository.Remove(Exercise.Model);

await \_exerciseRepository.SaveAsync();

RaiseDetailDeletedEvent(Exercise.Id);

}

}

}

protected override bool OnSaveCanExecute()

{

return Exercise != null

&& !Exercise.HasErrors

&& HasChanges;

}

protected async override void OnSaveExecute()

{

await SaveWithOptimisticConcurrencyAsync(\_exerciseRepository.SaveAsync,

() =>

{

HasChanges = \_exerciseRepository.HasChanges();

Id = Exercise.Id;

RaiseDetailSavedEvent(Exercise.Id, Exercise.Name);

});

}

private Exercise CreateNewExercise()

{

var exercise = new Exercise();

\_exerciseRepository.Add(exercise);

return exercise;

}

private void InitializeExercise(Exercise exercise)

{

Exercise = new ExerciseWrapper(exercise);

Exercise.PropertyChanged += (s, e) =>

{

if (!HasChanges)

{

HasChanges = \_exerciseRepository.HasChanges();

}

if (e.PropertyName == nameof(Exercise.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

if (e.PropertyName == nameof(Exercise.Name))

{

SetTitle();

}

};

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

if (Exercise.Id == 0)

{

// Little trick to trigger the validation

Exercise.Name = "";

}

SetTitle();

}

private void SetTitle()

{

Title = Exercise.Name;

}

private async Task LoadMusculeGroupsLookupAsync()

{

MusculeGroups.Clear();

//MusculeGroups.Add(new NullLookupItem { DisplayMember = " - " });

var lookup = await \_musculeGroupLookupDataService.GetMusculeGroupLookupAsync();

foreach (var lookupItem in lookup)

{

MusculeGroups.Add(lookupItem);

}

}

private async Task LoadEquipmentsLookupAsync()

{

Equipments.Clear();

//Equipments.Add(new NullLookupItem { DisplayMember = " - " });

var lookup = await \_equipmentLookupDataService.GetEquipmentLookupAsync();

foreach (var lookupItem in lookup)

{

Equipments.Add(lookupItem);

}

}

private async void AfterCollectionSaved(AfterCollectionSavedEventArgs args)

{

if (args.ViewModelName == nameof(MusculeGroupDetailViewModel))

{

await LoadMusculeGroupsLookupAsync();

}

if (args.ViewModelName == nameof(EquipmentDetailViewModel))

{

await LoadEquipmentsLookupAsync();

}

}

private void OpenUrl()

{

Process.Start(Exercise.DemoLink);

}

}

}

**IClientDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public interface IClientDetailViewModel : IDetailViewModel

{

}

}

**IDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public interface IDetailViewModel

{

Task LoadAsync(int Id);

bool HasChanges { get; }

int Id { get; }

}

}

**IExerciseDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public interface IExerciseDetailViewModel : IDetailViewModel

{

}

}

**INavigatioViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public interface INavigatioViewModel

{

Task LoadAsync();

}

}

**IWorkoutDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public interface IWorkoutDetailViewModel : IDetailViewModel

{

}

}

**MainViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

// Observable Collection- notifies the data binding when the collection has changed

public class MainViewModel : ViewModelBase

{

private IEventAggregator \_eventAggregator;

private IMessageDialogService \_messageDialogService;

private IIndex<string, IDetailViewModel> \_detailViewModelCreator;

private IDetailViewModel \_selectedDetailViewModel;

public MainViewModel(INavigatioViewModel navigatioViewModel,

IIndex<string, IDetailViewModel> detailViewModelCreator,

IEventAggregator eventAggregator,

IMessageDialogService messageDialogService)

{

\_detailViewModelCreator = detailViewModelCreator;

\_eventAggregator = eventAggregator;

\_messageDialogService = messageDialogService;

DetailViewModels = new ObservableCollection<IDetailViewModel>();

\_eventAggregator.GetEvent<OpenDetailViewEvent>()

.Subscribe(OnOpenDetailView);

\_eventAggregator.GetEvent<AfterDetailDeletedEvent>()

.Subscribe(AfterDetailDeleted);

\_eventAggregator.GetEvent<AfterDetailClosedEvent>()

.Subscribe(AfterDetailClosed);

CreateNewDetailCommand = new DelegateCommand<Type>(OnCreateNewDetailExecute);

OpenSingleDetailViewCommand = new DelegateCommand<Type>(OnOpenSingleDetailViewExecute);

NavigationViewModel = navigatioViewModel;

}

/// <summary>

/// Loads the data from the data service

/// </summary>

public async Task LoadAsync()

{

await NavigationViewModel.LoadAsync();

}

public ICommand CreateNewDetailCommand { get; }

public ICommand OpenSingleDetailViewCommand { get; }

public INavigatioViewModel NavigationViewModel { get; }

public ObservableCollection<IDetailViewModel> DetailViewModels { get; }

public IDetailViewModel SelectedDetailViewModel

{

get { return \_selectedDetailViewModel; }

set

{

\_selectedDetailViewModel = value;

OnPropertyChanged();

}

}

private async void OnOpenDetailView(OpenDetailViewEventArgs args)

{

var detailViewModel = DetailViewModels

.SingleOrDefault(vm => vm.Id == args.Id

&& vm.GetType().Name == args.ViewModelName);

if(detailViewModel == null)

{

detailViewModel = \_detailViewModelCreator[args.ViewModelName];

try

{

await detailViewModel.LoadAsync(args.Id);

}

catch

{

await \_messageDialogService.ShowInfoDialogAsync("Записът не може да бъде зареден." +

" Може би вече е изтрит от друг потребител. Наигационното меню ще бъде презаредено.");

await NavigationViewModel.LoadAsync();

return;

}

DetailViewModels.Add(detailViewModel);

}

SelectedDetailViewModel = detailViewModel;

}

private int nextNewItemId = 0;

private void OnCreateNewDetailExecute(Type viewModelType)

{

OnOpenDetailView(new OpenDetailViewEventArgs { Id = nextNewItemId--, ViewModelName = viewModelType.Name });

}

private void OnOpenSingleDetailViewExecute(Type viewModelType)

{

OnOpenDetailView(new OpenDetailViewEventArgs { Id = -1, ViewModelName = viewModelType.Name });

}

private void AfterDetailDeleted(AfterDetailDeletedEventArgs args)

{

RemoveDetailViewModel(args.Id, args.ViewModelName);

}

private void AfterDetailClosed(AfterDetailClosedEventArgs args)

{

RemoveDetailViewModel(args.Id, args.ViewModelName);

}

private void RemoveDetailViewModel(int id, string viewModelName)

{

var detailViewModel = DetailViewModels

.SingleOrDefault(vm => vm.Id == id

&& vm.GetType().Name == viewModelName);

if (detailViewModel != null)

{

DetailViewModels.Remove(detailViewModel);

}

}

}

}

**MusculeGroupDetailViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class MusculeGroupDetailViewModel : DetailViewModelBase

{

private IMusculeGroupRepository \_musculeGroupRepository;

private MusculeGroupWrapper \_selectedMusculeGroup;

public MusculeGroupDetailViewModel(IEventAggregator eventAggregator, IMessageDialogService messageDialogService,

IMusculeGroupRepository musculeGroupRepository)

: base(eventAggregator, messageDialogService)

{

\_musculeGroupRepository = musculeGroupRepository;

Title = "Мускулни групи";

MusculeGroups = new ObservableCollection<MusculeGroupWrapper>();

AddCommand = new DelegateCommand(OnAddExecute);

RemoveCommand = new DelegateCommand(OnRemoveExecute, OnRemoveCanExecute);

}

public ICommand AddCommand { get; }

public ICommand RemoveCommand { get; }

public ObservableCollection<MusculeGroupWrapper> MusculeGroups { get; }

public MusculeGroupWrapper SelectedMusculeGroup

{

get { return \_selectedMusculeGroup; }

set {

\_selectedMusculeGroup = value;

OnPropertyChanged();

((DelegateCommand)RemoveCommand).RaiseCanExecuteChanged();

}

}

public async override Task LoadAsync(int id)

{

Id = id;

foreach (var wrapper in MusculeGroups)

{

wrapper.PropertyChanged -= Wrapper\_PropertyChanged;

}

MusculeGroups.Clear();

var musculeGroups = await \_musculeGroupRepository.GetAllAsync();

foreach (var model in musculeGroups)

{

var wrapper = new MusculeGroupWrapper(model);

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

MusculeGroups.Add(wrapper);

}

}

protected override void OnDeleteExecute()

{

throw new NotImplementedException();

}

protected override bool OnSaveCanExecute()

{

return HasChanges && MusculeGroups.All(m => !m.HasErrors);

}

protected async override void OnSaveExecute()

{

try

{

await \_musculeGroupRepository.SaveAsync();

HasChanges = \_musculeGroupRepository.HasChanges();

RaiseCollectionSavedEvent();

}

catch(Exception ex)

{

while(ex.InnerException != null)

{

ex = ex.InnerException;

}

await MessageDialogService.ShowInfoDialogAsync("Грешка при запазването. Информацията ще бъде презаредена. Детайли: " + ex.Message);

await LoadAsync(Id);

}

}

private void Wrapper\_PropertyChanged(object sender, PropertyChangedEventArgs e)

{

if (!HasChanges)

{

HasChanges = \_musculeGroupRepository.HasChanges();

}

if (e.PropertyName == nameof(MusculeGroupWrapper.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

}

private bool OnRemoveCanExecute()

{

return SelectedMusculeGroup != null;

}

private async void OnRemoveExecute()

{

var isReferenced = await \_musculeGroupRepository.IsReferencedByExerciseAsync(SelectedMusculeGroup.Id);

if (isReferenced)

{

await MessageDialogService.ShowInfoDialogAsync($"Мускулната {SelectedMusculeGroup.Name} не може да бъде премахната" +

$", понеже се съдържа в поне едно упражнение.");

return;

}

SelectedMusculeGroup.PropertyChanged -= Wrapper\_PropertyChanged;

\_musculeGroupRepository.Remove(SelectedMusculeGroup.Model);

MusculeGroups.Remove(SelectedMusculeGroup);

SelectedMusculeGroup = null;

HasChanges = \_musculeGroupRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private void OnAddExecute()

{

var wrapper = new MusculeGroupWrapper(new MusculeGroup());

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

\_musculeGroupRepository.Add(wrapper.Model);

MusculeGroups.Add(wrapper);

//trigger the validation

wrapper.Name = "";

}

}

}

**WorkoutDetailViewModel.cs**

public class WorkoutDetailViewModel : DetailViewModelBase, IWorkoutDetailViewModel

{

private WorkoutWrapper \_workout;

private IWorkoutRepository \_workoutRepository;

private IExerciseSetupRepository \_exerciseSetupRepository;

private IExerciseLookupDataService \_exerciseLookupDataService;

private Client \_selectedAvailableClient;

private Client \_selectedAddedClient;

private string \_searchText;

private Exercise \_selectedFilteredExercise;

private List<Client> \_allClients;

private List<Exercise> \_allExercises;

private ExerciseSetupWrapper \_selectedExerciseSetup;

public WorkoutDetailViewModel(IEventAggregator eventAggregator,

IMessageDialogService messageDialogService,

IWorkoutRepository workoutRepository,

IExerciseSetupRepository exerciseSetupRepository,

IExerciseLookupDataService exerciseLookupDataService

) : base(eventAggregator, messageDialogService)

{

\_workoutRepository = workoutRepository;

\_exerciseSetupRepository = exerciseSetupRepository;

\_exerciseLookupDataService = exerciseLookupDataService;

eventAggregator.GetEvent<AfterDetailSavedEvent>().Subscribe(AfterDetailSaved);

eventAggregator.GetEvent<AfterDetailDeletedEvent>().Subscribe(AfterDetailDeleted);

AddedClients = new ObservableCollection<Client>();

AvailableClients = new ObservableCollection<Client>();

Exercises = new ObservableCollection<LookupItem>();

ExerciseSetups = new ObservableCollection<ExerciseSetupWrapper>();

AddToExerciseSetups = new DelegateCommand(OnAddToExerciseSetupsExecute, OnAddToExerciseSetupsCanExecute);

AddClientCommand = new DelegateCommand(OnAddClientExecute, OnAddClientCanExecute);

RemoveClientCommand = new DelegateCommand(OnRemoveClientExecute, OnRemoveClientCanExecute);

AddExerciseSetupCommand = new DelegateCommand(OnAddExerciseSetup);

RemoveExerciseSetupCommand = new DelegateCommand(OnRemoveExerciseSetupExecute, OnRemoveExerciseSetupCanExecute);

MoveUp = new DelegateCommand(MoveUpExecute, MoveUpCanExecute);

MoveDown = new DelegateCommand(MoveDownExecute, MoveDownCanExecute);

}

public WorkoutWrapper Workout

{

get { return \_workout; }

private set

{

\_workout = value;

OnPropertyChanged();

}

}

public ICommand AddClientCommand { get; }

public ICommand RemoveClientCommand { get; }

public ICommand AddExerciseSetupCommand { get; }

public ICommand RemoveExerciseSetupCommand { get; }

public ICommand AddToExerciseSetups { get; }

public ICommand MoveUp { get; }

public ICommand MoveDown { get; }

public ObservableCollection<Client> AddedClients { get; }

public ObservableCollection<Client> AvailableClients { get; }

public ObservableCollection<ExerciseSetupWrapper> ExerciseSetups { get; }

public ObservableCollection<LookupItem> Exercises { get; }

public IEnumerable<Exercise> FilteredExercises

{

get

{

if (SearchText == null) return \_allExercises;

return \_allExercises.Where(x => x.Name.ToUpper().Contains(SearchText.ToUpper()));

}

}

public Exercise SelectedFilteredExercise

{

get { return \_selectedFilteredExercise; }

set

{

\_selectedFilteredExercise = value;

OnPropertyChanged();

((DelegateCommand)AddToExerciseSetups).RaiseCanExecuteChanged();

}

}

public string SearchText

{

get { return \_searchText; }

set

{

\_searchText = value;

OnPropertyChanged("SearchText");

OnPropertyChanged("FilteredExercises");

}

}

public Client SelectedAvailableClient

{

get { return \_selectedAvailableClient; }

set

{

\_selectedAvailableClient = value;

OnPropertyChanged();

((DelegateCommand)AddClientCommand).RaiseCanExecuteChanged();

}

}

public Client SelectedAddedClient

{

get { return \_selectedAddedClient; }

set

{

\_selectedAddedClient = value;

OnPropertyChanged();

((DelegateCommand)RemoveClientCommand).RaiseCanExecuteChanged();

}

}

public ExerciseSetupWrapper SelectedExerciseSetup

{

get { return \_selectedExerciseSetup; }

set

{

\_selectedExerciseSetup = value;

OnPropertyChanged();

((DelegateCommand)RemoveExerciseSetupCommand).RaiseCanExecuteChanged();

((DelegateCommand)MoveUp).RaiseCanExecuteChanged();

((DelegateCommand)MoveDown).RaiseCanExecuteChanged();

}

}

public override async Task LoadAsync(int workoutId)

{

var workout = workoutId > 0

? await \_workoutRepository.GetByIdAsync(workoutId)

: CreateNewWorkout();

Id = workoutId;

InitializeWorkout(workout);

\_allClients = await \_workoutRepository.GetAllClientsAsync();

\_allExercises = await \_workoutRepository.GetAllExercisesAsync();

SetupPickList();

await InitializeExerciseSetups(workoutId);

}

protected async override void OnDeleteExecute()

{

var result = await MessageDialogService.ShowOkCancelDialogAsync($"Наистина ли искате да изтриете {Workout.Title}?", "Въпрос");

if (result == MessageDialogResult.OK)

{

\_workoutRepository.Remove(Workout.Model);

await \_workoutRepository.SaveAsync();

RaiseDetailDeletedEvent(Workout.Id);

}

}

protected override bool OnSaveCanExecute()

{

return Workout != null && !Workout.HasErrors && HasChanges && ExerciseSetups.All(ex => !ex.HasErrors);

}

protected override async void OnSaveExecute()

{

await SaveWithOptimisticConcurrencyAsync(\_workoutRepository.SaveAsync,

() =>

{

HasChanges = \_workoutRepository.HasChanges();

Id = Workout.Id;

RaiseDetailSavedEvent(Workout.Id, Workout.Title);

});

try

{

await \_exerciseSetupRepository.SaveAsync();

HasChanges = \_exerciseSetupRepository.HasChanges();

}

catch

{

return;

}

}

private void SetupPickList()

{

var wokoutClientIds = Workout.Model.Clients.Select(c => c.Id).ToList();

var addedClients = \_allClients.Where(c => wokoutClientIds.Contains(c.Id)).OrderBy(c => c.FirstName);

var availableClients = \_allClients.Except(addedClients).OrderBy(c => c.FirstName);

AddedClients.Clear();

AvailableClients.Clear();

foreach (var addedClient in addedClients)

{

AddedClients.Add(addedClient);

}

foreach (var availableClient in availableClients)

{

AvailableClients.Add(availableClient);

}

}

private Workout CreateNewWorkout()

{

var workout = new Workout

{

Date = DateTime.Now

};

\_workoutRepository.Add(workout);

return workout;

}

private void InitializeWorkout(Workout workout)

{

Workout = new WorkoutWrapper(workout);

Workout.PropertyChanged += (s, e) =>

{

if (!HasChanges)

{

HasChanges = \_workoutRepository.HasChanges();

}

if (e.PropertyName == nameof(Workout.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

if (e.PropertyName == nameof(Workout.Title))

{

SetTitle();

}

};

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

if (Workout.Id == 0)

{

Workout.Title = ""; //TO TRIGGER THE VALIDATION

}

SetTitle();

}

private void SetTitle()

{

Title = Workout.Title;

}

private async Task InitializeExerciseSetups(int workoutId)

{

foreach (var wrapper in ExerciseSetups)

{

wrapper.PropertyChanged -= Wrapper\_PropertyChanged;

}

ExerciseSetups.Clear();

var exerciseSetups = await \_exerciseSetupRepository.GetAllAsync(workoutId);

foreach (var model in exerciseSetups)

{

var wrapper = new ExerciseSetupWrapper(model);

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

ExerciseSetups.Add(wrapper);

}

await LoadExerciseLookupAsync();

}

private void Wrapper\_PropertyChanged(object sender, PropertyChangedEventArgs e)

{

if (!HasChanges)

{

HasChanges = \_exerciseSetupRepository.HasChanges();

}

if (e.PropertyName == nameof(ExerciseSetupWrapper.HasErrors))

{

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

}

private async Task LoadExerciseLookupAsync()

{

Exercises.Clear();

//Exercises.Add(new NullLookupItem { DisplayMember = " - " });

var lookup = await \_exerciseLookupDataService.GetExerciseLookupAsync();

foreach (var lookupItem in lookup)

{

Exercises.Add(lookupItem);

}

}

private bool OnRemoveClientCanExecute()

{

return SelectedAddedClient != null;

}

private void OnRemoveClientExecute()

{

var clientToRemove = SelectedAddedClient;

Workout.Model.Clients.Remove(clientToRemove);

AddedClients.Remove(clientToRemove);

AvailableClients.Add(clientToRemove);

HasChanges = \_workoutRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private void OnAddClientExecute()

{

var clientToAdd = SelectedAvailableClient;

Workout.Model.Clients.Add(clientToAdd);

AddedClients.Add(clientToAdd);

AvailableClients.Remove(clientToAdd);

HasChanges = \_workoutRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private bool OnAddClientCanExecute()

{

return SelectedAvailableClient != null;

}

private async void OnAddExerciseSetup()

{

if (Id > 0)

{

var position = ExerciseSetups.Count + 1;

var wrapper = new ExerciseSetupWrapper(new ExerciseSetup { WorkoutId = Id, Position = position });

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

\_exerciseSetupRepository.Add(wrapper.Model);

ExerciseSetups.Add(wrapper);

}

else

{

await MessageDialogService.ShowInfoDialogAsync("Моля натиснете бутона Запази преди да добавите нов ред.");

}

}

private void OnRemoveExerciseSetupExecute()

{

SelectedExerciseSetup.PropertyChanged -= Wrapper\_PropertyChanged;

\_exerciseSetupRepository.Remove(SelectedExerciseSetup.Model);

ExerciseSetups.Remove(SelectedExerciseSetup);

SelectedExerciseSetup = null;

for (int i = 1; i <= ExerciseSetups.Count; i++)

{

ExerciseSetups[i - 1].Position = i;

}

HasChanges = \_exerciseSetupRepository.HasChanges();

((DelegateCommand)SaveCommand).RaiseCanExecuteChanged();

}

private bool OnRemoveExerciseSetupCanExecute()

{

return SelectedExerciseSetup != null;

}

private async void OnAddToExerciseSetupsExecute()

{

if(Id <= 0) //if it's newly created

{

await MessageDialogService.ShowInfoDialogAsync("Натиснете бутона ЗАПАЗИ преди да добавите упражнение");

return;

}

else

{

var exerciseToAdd = SelectedFilteredExercise;

var position = ExerciseSetups.Count + 1;

var wrapper = new ExerciseSetupWrapper(new ExerciseSetup { WorkoutId = Id, Position = position, ExerciseId = exerciseToAdd.Id });

wrapper.PropertyChanged += Wrapper\_PropertyChanged;

\_exerciseSetupRepository.Add(wrapper.Model);

ExerciseSetups.Add(wrapper);

await \_exerciseSetupRepository.SaveAsync();

HasChanges = \_exerciseSetupRepository.HasChanges();

await \_workoutRepository.SaveAsync();

HasChanges = \_workoutRepository.HasChanges();

Id = Workout.Id;

RaiseDetailSavedEvent(Workout.Id, Workout.Title);

SelectedFilteredExercise = null;

}

}

private bool OnAddToExerciseSetupsCanExecute()

{

return SelectedFilteredExercise != null;

}

private bool MoveDownCanExecute()

{

return SelectedExerciseSetup != null && ExerciseSetups.Count != 1;

}

private bool MoveUpCanExecute()

{

return SelectedExerciseSetup != null && ExerciseSetups.Count != 1;

}

private void MoveUpExecute()

{

if (SelectedExerciseSetup == null)

{

return;

}

var selectedItemPosition = SelectedExerciseSetup.Position - 1;

if (selectedItemPosition == 0)

{

return;

}

var toSwap = ExerciseSetups[selectedItemPosition];

ExerciseSetups[selectedItemPosition] = ExerciseSetups[selectedItemPosition - 1];

ExerciseSetups[selectedItemPosition - 1] = toSwap;

ExerciseSetups[selectedItemPosition - 1].Position = selectedItemPosition;

ExerciseSetups[selectedItemPosition].Position = selectedItemPosition + 1;

SelectedExerciseSetup = ExerciseSetups[selectedItemPosition - 1];

}

private void MoveDownExecute()

{

if (SelectedExerciseSetup == null)

{

return;

}

var selectedItemPosition = SelectedExerciseSetup.Position - 1;

if (selectedItemPosition == ExerciseSetups.Count - 1)

{

return;

}

var toSwap = ExerciseSetups[selectedItemPosition];

ExerciseSetups[selectedItemPosition] = ExerciseSetups[selectedItemPosition + 1];

ExerciseSetups[selectedItemPosition + 1] = toSwap;

ExerciseSetups[selectedItemPosition + 1].Position = selectedItemPosition + 2;

ExerciseSetups[selectedItemPosition].Position = selectedItemPosition + 1;

SelectedExerciseSetup = ExerciseSetups[selectedItemPosition + 1];

}

private async void AfterDetailSaved(AfterDetailSavedEventArgs args)

{

if (args.ViewModelName == nameof(ClientDetailViewModel))

{

await \_workoutRepository.ReloadClientAsync(args.Id);

\_allClients = await \_workoutRepository.GetAllClientsAsync();

SetupPickList();

}

if (args.ViewModelName == nameof(ExerciseDetailViewModel))

{

await LoadAsync(Id);

}

}

private async void AfterDetailDeleted(AfterDetailDeletedEventArgs args)

{

if (args.ViewModelName == nameof(ClientDetailViewModel))

{

\_allClients = await \_workoutRepository.GetAllClientsAsync();

SetupPickList();

}

if (args.ViewModelName == nameof(ExerciseDetailViewModel))

{

\_allExercises = await \_workoutRepository.GetAllExercisesAsync();

await LoadAsync(Id);

var exerciseCount = ExerciseSetups.Count;

if (exerciseCount != 0)

{

for (int i = 1; i <= exerciseCount; i++)

{

ExerciseSetups[i - 1].Position = i;

}

}

await \_workoutRepository.SaveAsync();

HasChanges = \_workoutRepository.HasChanges();

Id = Workout.Id;

RaiseDetailSavedEvent(Workout.Id, Workout.Title);

}

}

}

}

**NavigationViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class NavigationViewModel : ViewModelBase, INavigatioViewModel

{

private IClientLookupDataService \_clientLookupService;

private IWorkoutLookupDataService \_workoutLookupService;

private IExerciseLookupDataService \_exerciseLookupDataService;

private IEventAggregator \_eventAggregator;

public NavigationViewModel(IClientLookupDataService clientLookupService,

IWorkoutLookupDataService workoutLookupService,

IExerciseLookupDataService exerciseLookupDataService,

IEventAggregator eventAggregator)

{

\_clientLookupService = clientLookupService;

\_workoutLookupService = workoutLookupService;

\_exerciseLookupDataService = exerciseLookupDataService;

\_eventAggregator = eventAggregator;

Clients = new ObservableCollection<NavigationItemViewModel>();

Workouts = new ObservableCollection<NavigationItemViewModel>();

Exercises = new ObservableCollection<NavigationItemViewModel>();

\_eventAggregator.GetEvent<AfterDetailSavedEvent>().Subscribe(AfterDetailSaved);

\_eventAggregator.GetEvent<AfterDetailDeletedEvent>().Subscribe(AfterDetailDeleted);

}

public async Task LoadAsync()

{

var lookup = await \_clientLookupService.GetClientLookupAsync();

Clients.Clear();

foreach (var item in lookup)

{

Clients.Add(new NavigationItemViewModel(item.Id, item.DisplayMember, \_eventAggregator, nameof(ClientDetailViewModel)));

}

lookup = await \_workoutLookupService.GetWorkoutLookupAsync();

Workouts.Clear();

foreach (var item in lookup)

{

Workouts.Add(new NavigationItemViewModel(item.Id, item.DisplayMember, \_eventAggregator, nameof(WorkoutDetailViewModel)));

}

lookup = await \_exerciseLookupDataService.GetExerciseLookupAsync();

Exercises.Clear();

foreach (var item in lookup)

{

Exercises.Add(new NavigationItemViewModel(item.Id, item.DisplayMember, \_eventAggregator, nameof(ExerciseDetailViewModel)));

}

}

public ObservableCollection<NavigationItemViewModel> Clients { get; }

public ObservableCollection<NavigationItemViewModel> Workouts { get; }

public ObservableCollection<NavigationItemViewModel> Exercises { get; }

private void AfterDetailDeleted(AfterDetailDeletedEventArgs args)

{

switch (args.ViewModelName)

{

case nameof(ClientDetailViewModel):

AfterDetailDeleted(Clients, args);

break;

case nameof(WorkoutDetailViewModel):

AfterDetailDeleted(Workouts, args);

break;

case nameof(ExerciseDetailViewModel):

AfterDetailDeleted(Exercises, args);

break;

}

}

private void AfterDetailDeleted(ObservableCollection<NavigationItemViewModel> items, AfterDetailDeletedEventArgs args)

{

var item = items.SingleOrDefault(c => c.Id == args.Id);

if (item != null)

{

items.Remove(item);

}

}

private void AfterDetailSaved(AfterDetailSavedEventArgs args)

{

switch (args.ViewModelName)

{

case nameof(ClientDetailViewModel):

AfterDetailSaved(Clients, args);

break;

case nameof(WorkoutDetailViewModel):

AfterDetailSaved(Workouts, args);

break;

case nameof(ExerciseDetailViewModel):

AfterDetailSaved(Exercises, args);

break;

}

}

private void AfterDetailSaved(ObservableCollection<NavigationItemViewModel> items, AfterDetailSavedEventArgs args)

{

var lookupItem = items.SingleOrDefault(l => l.Id == args.Id);

if (lookupItem == null)

{

items.Add(new NavigationItemViewModel(args.Id, args.DisplayMember, \_eventAggregator, args.ViewModelName));

}

else

{

lookupItem.DisplayMember = args.DisplayMember;

}

}

}

}

**NavigationItemViewModel.cs**

namespace WorkoutOrganizer.UI.ViewModel

{

public class NavigationItemViewModel : ViewModelBase

{

private string \_displayMember;

private IEventAggregator \_eventAggregator;

private string \_detailViewModelName;

public NavigationItemViewModel(int id, string displayMember, IEventAggregator eventAggregator, string detailViewModelName)

{

\_eventAggregator = eventAggregator;

\_detailViewModelName = detailViewModelName;

Id = id;

DisplayMember = displayMember;

OpenDetailViewCommand = new DelegateCommand(OnOpenDetailViewExecute);

}

public int Id { get; }

public string DisplayMember

{

get { return \_displayMember; }

set

{

\_displayMember = value;

OnPropertyChanged();

}

}

public ICommand OpenDetailViewCommand { get; }

private void OnOpenDetailViewExecute()

{

\_eventAggregator.GetEvent<OpenDetailViewEvent>()

.Publish(

new OpenDetailViewEventArgs

{

Id = Id,

ViewModelName = \_detailViewModelName

});

}

}

}

**MainWindow.xaml**

<Controls:MetroWindow x:Class="WorkoutOrganizer.UI.MainWindow"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:d="http://schemas.microsoft.com/expression/blend/2008"

xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"

xmlns:local="clr-namespace:WorkoutOrganizer.UI"

xmlns:view="clr-namespace:WorkoutOrganizer.UI.View"

xmlns:Controls="clr-namespace:MahApps.Metro.Controls;assembly=MahApps.Metro"

xmlns:viewModel="clr-namespace:WorkoutOrganizer.UI.ViewModel"

mc:Ignorable="d"

Title="Workout Organizer" Height="1000" Width="1200" FontSize="20"

WindowStartupLocation="CenterScreen"

WindowStyle="None"

WindowState="Maximized">

<Window.Resources>

<DataTemplate DataType="{x:Type viewModel:ClientDetailViewModel}">

<view:ClientDetailView/>

</DataTemplate>

<DataTemplate DataType="{x:Type viewModel:WorkoutDetailViewModel}">

<view:WorkoutDetailView/>

</DataTemplate>

<DataTemplate DataType="{x:Type viewModel:MusculeGroupDetailViewModel}">

<view:MusculeGroupDetailView/>

</DataTemplate>

<DataTemplate DataType="{x:Type viewModel:ExerciseDetailViewModel}">

<view:ExerciseDetailView/>

</DataTemplate>

<DataTemplate DataType="{x:Type viewModel:EquipmentDetailViewModel}">

<view:EquipmentDetailView/>

</DataTemplate>

</Window.Resources>

<Grid>

<Grid.RowDefinitions>

<RowDefinition Height="Auto"/>

<RowDefinition/>

</Grid.RowDefinitions>

<Grid.ColumnDefinitions>

<ColumnDefinition Width="300"/>

<ColumnDefinition />

</Grid.ColumnDefinitions>

<Menu Grid.ColumnSpan="2" FontSize="20">

<MenuItem Header="Създаване">

<MenuItem Header="Нов клиент" Command="{Binding CreateNewDetailCommand}"

CommandParameter="{x:Type viewModel:ClientDetailViewModel}"/>

<MenuItem Header="Нова тренировка" Command="{Binding CreateNewDetailCommand}"

CommandParameter="{x:Type viewModel:WorkoutDetailViewModel}"/>

<MenuItem Header="Ново упражнение" Command="{Binding CreateNewDetailCommand}"

CommandParameter="{x:Type viewModel:ExerciseDetailViewModel}"/>

</MenuItem>

<MenuItem Header="Редактиране">

<MenuItem Header="Мускулни групи" Command="{Binding OpenSingleDetailViewCommand}"

CommandParameter="{x:Type viewModel:MusculeGroupDetailViewModel}"/>

<MenuItem Header="Оборудване" Command="{Binding OpenSingleDetailViewCommand}"

CommandParameter="{x:Type viewModel:EquipmentDetailViewModel}"/>

</MenuItem>

</Menu>

<view:NavigationView Grid.Row ="1"

DataContext="{Binding NavigationViewModel}"/>

<TabControl Grid.Row ="1"

Grid.Column="1"

ItemsSource="{Binding DetailViewModels}"

SelectedItem="{Binding SelectedDetailViewModel, Mode=TwoWay}">

<TabControl.ItemContainerStyle>

<Style TargetType="TabItem"

BasedOn="{StaticResource {x:Type TabItem}}">

<Setter Property="Background" Value="#333333"/>

<Setter Property="BorderBrush" Value="Transparent"/>

<Setter Property="BorderThickness" Value="2"/>

</Style>

</TabControl.ItemContainerStyle>

<TabControl.ItemTemplate>

<DataTemplate>

<StackPanel Orientation="Horizontal">

<TextBlock Text="{Binding Title}"/>

<TextBlock Text="\*" Visibility="{Binding HasChanges,

Converter={StaticResource booleanToVisibilityConverter}}"/>

<Button Command="{Binding CloseDetailViewCommand}"

Style="{StaticResource ChromelessButtonStyle}"

VerticalContentAlignment="Top" Margin="2 0 2 2"

Content="X" FontSize="20"/>

</StackPanel>

</DataTemplate>

</TabControl.ItemTemplate>

</TabControl>

</Grid>

</Controls:MetroWindow>

**App.xaml**

<Application x:Class="WorkoutOrganizer.UI.App"

xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"

xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"

xmlns:local="clr-namespace:WorkoutOrganizer.UI"

xmlns:toolkit="http://schemas.xceed.com/wpf/xaml/toolkit"

Startup="Application\_Startup"

DispatcherUnhandledException="Application\_DispatcherUnhandledException">

<Application.Resources>

<ResourceDictionary>

<ResourceDictionary.MergedDictionaries>

<!-- MahApps.Metro resource dictionaries. Make sure that all file names are Case Sensitive! -->

<ResourceDictionary Source="pack://application:,,,/MahApps.Metro;component/Styles/Controls.xaml" />

<ResourceDictionary Source="pack://application:,,,/MahApps.Metro;component/Styles/Fonts.xaml" />

<ResourceDictionary Source="pack://application:,,,/MahApps.Metro;component/Styles/Colors.xaml" />

<!-- Accent and AppTheme setting -->

<ResourceDictionary Source="pack://application:,,,/MahApps.Metro;component/Styles/Accents/Blue.xaml" />

<ResourceDictionary Source="pack://application:,,,/MahApps.Metro;component/Styles/Accents/BaseDark.xaml" />

<ResourceDictionary>

<Style TargetType="Button" BasedOn="{StaticResource {x:Type Button}}">

<Setter Property="FontSize" Value="16"/>

</Style>

<Style TargetType="ComboBox" BasedOn="{StaticResource {x:Type ComboBox}}">

<Setter Property="FontSize" Value="20"/>

</Style>

<Style TargetType="Label" BasedOn="{StaticResource {x:Type Label}}">

<Setter Property="Foreground" Value="#CCCCCC"/>

</Style>

<Style TargetType="TextBlock" BasedOn="{StaticResource {x:Type TextBlock}}">

<Setter Property="HorizontalAlignment" Value="Center"/>

</Style>

<Style TargetType="toolkit:DateTimePicker" BasedOn="{StaticResource {x:Type toolkit:DateTimePicker}}">

<Setter Property="Foreground" Value="White"/>

<Setter Property="Validation.ErrorTemplate">

<Setter.Value>

<ControlTemplate>

<StackPanel>

<AdornedElementPlaceholder x:Name="placeholder"/>

<TextBlock FontSize="12" Foreground="White" Text="{Binding

ElementName=placeholder, Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"/>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<Style.Triggers>

<Trigger Property="Validation.HasError" Value="True">

<Setter Property="Background" Value="DarkRed"/>

<Setter Property="ToolTip" Value="{Binding

RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}"/>

</Trigger>

</Style.Triggers>

</Style>

<BooleanToVisibilityConverter x:Key="booleanToVisibilityConverter"/>

<Style TargetType="TextBox" BasedOn="{StaticResource {x:Type TextBox}}">

<Setter Property="FontSize" Value="20"/>

<Setter Property="Validation.ErrorTemplate">

<Setter.Value>

<ControlTemplate>

<StackPanel>

<AdornedElementPlaceholder x:Name="placeholder"/>

<TextBlock FontSize="12" Foreground="White" Text="{Binding

ElementName=placeholder, Path=AdornedElement.(Validation.Errors)[0].ErrorContent}"/>

</StackPanel>

</ControlTemplate>

</Setter.Value>

</Setter>

<Style.Triggers>

<Trigger Property="Validation.HasError" Value="True">

<Setter Property="Background" Value="DarkRed"/>

<Setter Property="ToolTip" Value="{Binding

RelativeSource={RelativeSource Self}, Path=(Validation.Errors)[0].ErrorContent}"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="DataGridColumnStyle" TargetType="TextBlock">

<Style.Triggers>

<Trigger Property="Validation.HasError" Value="True">

<Setter Property="Background" Value="DarkRed"/>

<Setter Property="ToolTip" Value="{Binding RelativeSource={RelativeSource Self},

Path=(Validation.Errors)[0].ErrorContent}"/>

</Trigger>

</Style.Triggers>

</Style>

<Style x:Key="DataGridEditingColumnStyle" TargetType="TextBox">

<Style.Triggers>

<Trigger Property="Validation.HasError" Value="True">

<Setter Property="Background" Value="DarkRed"/>

<Setter Property="ToolTip" Value="{Binding RelativeSource={RelativeSource Self},

Path=(Validation.Errors)[0].ErrorContent}"/>

</Trigger>

</Style.Triggers>

</Style>

</ResourceDictionary>

</ResourceDictionary.MergedDictionaries>

</ResourceDictionary>

</Application.Resources>

</Application>