Технически университет - София

ДОКУМЕНТАЦИЯ

на проект

по дисциплина

ПРОГРАМНИ СИСТЕМИ

Тема №3

Факултет: Компютърни системи и технологии

Специалност: Компютърно и софтуерно инженерство

Студент: *Самуил Валентинов Иванов*

Група: *43*

Фак. номер: *121220011*

Асистент*: ас. Петър Маринов*

СОФИЯ

2023г.

Съдържание

Задание

Без използване на готова компилирана или лицензирана библиотека чрез използване или надграждане на включените в Windows Forms и в WPF класове да се изгради:

Система за изменяемо визуализиране на списък от обекти и избор на един от обектите

Изисквания:

- ➔ Трябва да може да се посочи списък от обекти за визуализиране.
- ➔ Трябва да може динамично да се укаже кои полета/свойства/ от обектите ще бъдат визуализирани.
- ➔ При избор на ред/обект от визуализацията трябва да може да се върне целият съответстващ обект, независимо колко полета/свойства/ са били визуализирани.
- ➔ Трябва да може визуализацията и списъкът да бъдат изчиствани и да се посочва нов списък за визуализация.

Приложение за демонстрация:

- ➔ Системата трябва да се използва в .Net C# WPF приложение с връзка с БД чрез EF и с MVVM структура, както и в .Net C# Windows Forms приложение. (Приложенията не е нужно да са богато реализирани за целите на заданието.)
- ➔ Обектите за визуализация и избор идват от БД.
- ➔ Подаването на даден списък от обекти трябва да е помодулен начин, необвързан към конкретна инстанция на системата или към конкретен интерфейс.
- ➔ Оказване на полетата/свойствата/ за визуализация трябва да е помодулен начин, необвързан към конкретния тип обекти, списък от който се подава за визуализация.
- ➔ Да се демонстрира в рамките на поне 2 различни интерфейса (с различен view-model) посочване на списъци от 2 различни типа обекти.
- ➔ Да се демонстрира в рамките на един интерфейс посочването на два различни комплекта полета/свойства/.

Примери за използване (Application Scenario):

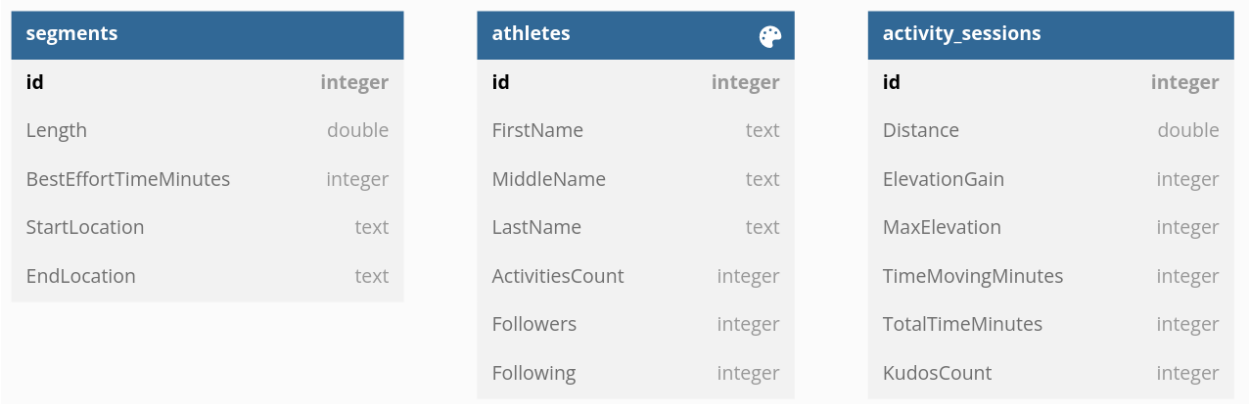
1. Визуализация на всякакъв списък от данни идващи от ДБ, от които потребителя трябва да си избере един, с който да продължи работа.

Въведение в приложението

Този проект представлява WPF приложение, изградено на базата на MVVM архитектурата за изграждане на многослойни приложения. Връзката с базата данни се осъществява посредством Entity Framework рамката. Това, което реализира приложението, предоставя на потребителите възможност за визуализиране на обекти от базата данни на базата на техните атрибути (полета/колони). Имплементирани са следните обекти: атлети (Athletes), сегменти (Segments) и тренировки (ActivitySessions). Изборът между отделните обекти и съответните атрибути, които притежават, се осъществява динамично в рамките на главния прозорец, който визуализира 3 екрана в процеса на използване на приложението. При избиране на конкретен обект в първият екран и натискане на бутон за визуализиране на атрибутите на избрания обект, атрибутите се визуализират с възможност за селектиране на желаните от нас с помощта на checkbox бутони. След посочване на желаните атрибути и натискане на бутон за визуализация, на екрана се визуализира таблица с данните за избрания обект, като таблицата съдържа само посочените полета. При двойно натискане на даден ред от таблицата се визуализира пълната информация за обекта от конкретния ред от базата данни.

Спецификация на базата данни

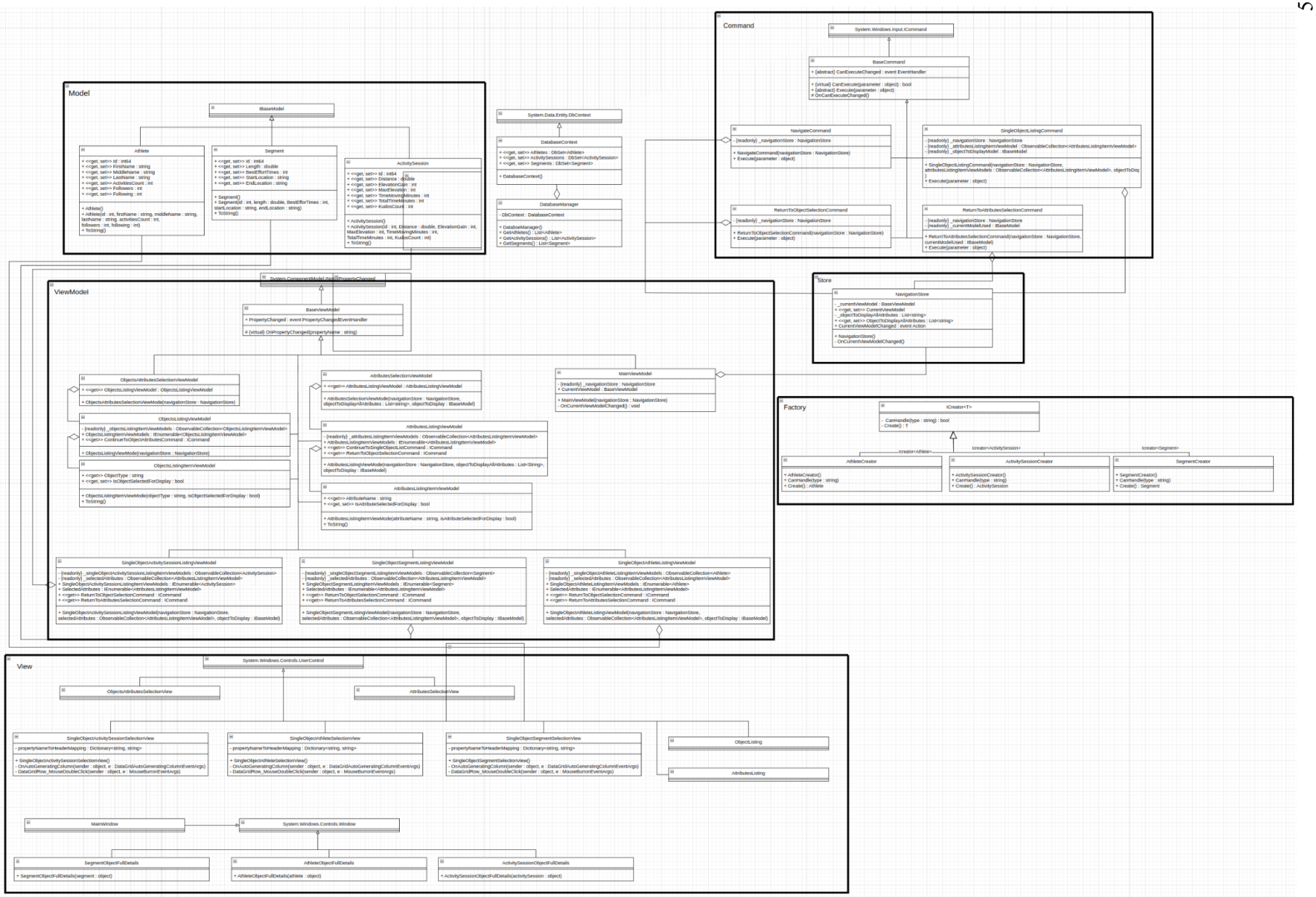
На фигура 1. долу е представена Entity Relationship (ER) диаграма на базата данни, която показва наличните таблици и връзки в базата от данни. Всяка таблица представлява обект, който може да бъде визуализиран в приложението.



Фигура 1. Entity Relationship диаграма

Спецификация на класовете

На фигура 2. на следващата страница е показана UML диаграма на класовете:


Model
IBaseModel
Athlete
+ <<get, set>> Id : int64
+ <<get, set>> FirstName : string
+ <<get, set>> MiddleName : string
+ <<get, set>> LastName : string
+ <<get, set>> ActivitiesCount : int
+ <<get, set>> Followers : int
+ <<get, set>> Following : int
+ Athlete()
+ Athlete(id : int, firstName : string, middleName : string, lastName : string, activitiesCount : int, followers : int, following : int)
+ ToString()
Segment
+ <<get, set>> Id : int64
+ <<get, set>> Length : double
+ <<get, set>> BestEffortTimes : int
+ <<get, set>> StartLocation : string
+ <<get, set>> EndLocation : string
+ Segment()
+ Segment(id : int, length : double, BestEffortTimes : int, startLocation : string, endLocation : string)
+ ToString()
ActivitySession
+ <<get, set>> Id : int64
+ <<get, set>> Distance : double
+ <<get, set>> ElevationGain : int
+ <<get, set>> MaxElevation : int
+ <<get, set>> TimeMovingMinutes : int
+ <<get, set>> TotalTimeMinutes : int
+ <<get, set>> KudosCount : int
+ ActivitySession()
+ ActivitySession(id : int, Distance : double, ElevationGain : int, MaxElevation : int, TimeMovingMinutes : int, TotalTimeMinutes : int, KudosCount : int)
+ ToString()
System.Data.Entity.DbContext
DatabaseContext
+ <<get, set>> Athletes : DbSet<Athlete>
+ <<get, set>> ActivitySessions : DbSet<ActivitySession>
+ <<get, set>> Segments : DbSet
+ DatabaseContext()
DatabaseManager
- DbContext : DatabaseContext
+ DatabaseManager()
+ GetAthletes() : List<Athlete>
+ GetActivitySessions() : List<ActivitySession>
+ GetSegments() : List
Command
System.Windows.Input.ICommand
BaseCommand
+ [abstract] CanExecuteChanged : event EventHandler
+ [virtual] CanExecute(parameter : object) : bool
+ [abstract] Execute(parameter : object)
# OnCanExecuteChanged()
NavigateCommand
- [readonly] _navigationStore : NavigationStore
+ NavigateCommand(navigationStore : NavigationStore)
+ Execute(parameter : object)
SingleObjectListingCommand
- [readonly] _navigationStore : NavigationStore
- [readonly] _attributesListingItemViewModel : ObservableCollection<AttributesListingItemViewModel>
- [readonly] _objectToDisplayModel : IBaseModel
+ SingleObjectListingCommand(navigationStore : NavigationStore, attributesListingItemViewModels : ObservableCollection<AttributesListingItemViewModel>, objectToDisplayModel : IBaseModel)
+ Execute(parameter : object)
ReturnToObjectSelectionCommand
- [readonly] _navigationStore : NavigationStore
+ ReturnToObjectSelectionCommand(navigationStore : NavigationStore)
+ Execute(parameter : object)
ReturnToAttributesSelectionCommand
- [readonly] _navigationStore : NavigationStore
- [readonly] _currentModelUsed : IBaseModel
+ ReturnToAttributesSelectionCommand(navigationStore : NavigationStore, currentModelUsed : IBaseModel)
+ Execute(parameter : object)
Store
NavigationStore
- _currentViewModel : BaseViewModel
+ <<get, set>> CurrentViewModel
- _objectToDisplayAllAttributes : List<string>
+ <<get, set>> ObjectToDisplayAllAttributes : List<string>
+ CurrentViewModelChanged : event Action
+ NavigationStore()
- OnCurrentViewModelChanged()
ViewModel
System.ComponentModel.INotifyPropertyChanged
BaseViewModel
+ PropertyChanged : event PropertyChangedEventHandler
# (virtual) OnPropertyChanged(propertyName : string)
ObjectsAttributesSelectionViewModel
+ <<get>> ObjectsListingViewModel : ObjectsListingViewModel
+ ObjectsAttributesSelectionViewModel(navigationStore : NavigationStore)
ObjectsListingViewModel
- [readonly] _objectsListingItemViewModels : ObservableCollection<ObjectsListingItemViewModel>
+ ObjectsListingItemViewModels : IEnumerable<ObjectsListingItemViewModel>
+ <<get>> ContinueToObjectAttributesCommand : ICommand
+ ObjectsListingViewMode(navigationStore : NavigationStore)
ObjectsListingItemViewModel
+ <<get>> ObjectType : string
+ <<get, set>> IsObjectSelectedForDisplay : bool
+ ObjectsListingItemViewMode(objectType : string, isObjectSelectedForDisplay : bool)
+ ToString()
AttributesSelectionViewModel
+ <<get>> AttributesListingViewModel : AttributesListingViewModel
+ AttributesSelectionViewMode(navigationStore : NavigationStore, objectToDisplayAllAttributes : List<string>, objectToDisplay : IBaseModel)
AttributesListingViewModel
- [readonly] _attributesListingItemViewModels : ObservableCollection<AttributesListingItemViewModel>
+ AttributesListingItemViewModels : IEnumerable<AttributesListingItemViewModel>
+ <<get>> ContinueToSingleObjectListCommand : ICommand
+ <<get>> ReturnToObjectSelectionCommand : ICommand
+ AttributesListingViewMode(navigationStore : NavigationStore, objectToDisplayAllAttributes : List<String>, objectToDisplay : IBaseModel)
AttributesListingItemViewModel
+ <<get>> AttributeName : string
+ <<get, set>> IsAttributeSelectedForDisplay : bool
+ AttributesListingItemViewMode(attributeName : string, isAttributeSelectedForDisplay : bool)
+ ToString()
MainViewModel
- [readonly] _navigationStore : NavigationStore
+ CurrentViewModel : BaseViewModel
+ MainViewModel(navigationStore : NavigationStore)
- OnCurrentViewModelChanged() : void
SingleObjectActivitySessionListingViewModel
- [readonly] _singleObjectActivitySessionListingItemViewModels : ObservableCollection<ActivitySession>
- [readonly] _selectedAttributes : ObservableCollection<AttributesListingItemViewModel>
+ SingleObjectActivitySessionListingItemViewModels : IEnumerable<ActivitySession>
+ SelectedAttributes : IEnumerable<AttributesListingItemViewModel>
+ <<get>> ReturnToObjectSelectionCommand : ICommand
+ <<get>> ReturnToAttributesSelectionCommand : ICommand
+ SingleObjectActivitySessionListingViewModel(navigationStore : NavigationStore, selectedAttributes : ObservableCollection<AttributesListingItemViewModel>, objectToDisplay : IBaseModel)
SingleObjectSegmentListingViewModel
- [readonly] _singleObjectSegmentListingItemViewModels : ObservableCollection
- [readonly] _selectedAttributes : ObservableCollection<AttributesListingItemViewModel>
+ SingleObjectSegmentListingItemViewModels : IEnumerable
+ SelectedAttributes : IEnumerable<AttributesListingItemViewModel>
+ <<get>> ReturnToObjectSelectionCommand : ICommand
+ <<get>> ReturnToAttributesSelectionCommand : ICommand
+ SingleObjectSegmentListingViewModel(navigationStore : NavigationStore, selectedAttributes : ObservableCollection<AttributesListingItemViewModel>, objectToDisplay : IBaseModel)
SingleObjectAthleteListingViewModel
- [readonly] _singleObjectAthleteListingItemViewModels : ObservableCollection<Athlete>
- [readonly] _selectedAttributes : ObservableCollection<AttributesListingItemViewModel>
+ SingleObjectAthleteListingItemViewModels : IEnumerable<Athlete>
+ SelectedAttributes : IEnumerable<AttributesListingItemViewModel>
+ <<get>> ReturnToObjectSelectionCommand : ICommand
+ <<get>> ReturnToAttributesSelectionCommand : ICommand
+ SingleObjectAthleteListingViewModel(navigationStore : NavigationStore, selectedAttributes : ObservableCollection<AttributesListingItemViewModel>, objectToDisplay : IBaseModel)
Factory
ICreator<T>
- CanHandle(type : string) : bool
- Create() : T
ICreator<Athlete>
ICreator<ActivitySession>
ICreator
AthleteCreator
+ AthleteCreator()
+ CanHandle(type : string)
+ Create() : Athlete
ActivitySessionCreator
+ ActivitySessionCreator()
+ CanHandle(type : string)
+ Create() : ActivitySession
SegmentCreator
+ SegmentCreator()
+ CanHandle(type : string)
+ Create() : Segment
View
System.Windows.Controls.UserControl
ObjectsAttributesSelectionView
AttributesSelectionView
SingleObjectActivitySessionSelectionView
- propertyNameToHeaderMapping : Dictionary<string, string>
+ SingleObjectActivitySessionSelectionView()
- OnAutoGeneratingColumn(sender : object, e : DataGridAutoGeneratingColumnEventArgs)
- DataGridRow_MouseDoubleClick(sender : object, e : MouseButtonEventArgs)
SingleObjectAthleteSelectionView
- propertyNameToHeaderMapping : Dictionary<string, string>
+ SingleObjectAthleteSelectionView()
- OnAutoGeneratingColumn(sender : object, e : DataGridAutoGeneratingColumnEventArgs)
- DataGridRow_MouseDoubleClick(sender : object, e : MouseButtonEventArgs)
SingleObjectSegmentSelectionView
- propertyNameToHeaderMapping : Dictionary<string, string>
+ SingleObjectSegmentSelectionView()
- OnAutoGeneratingColumn(sender : object, e : DataGridAutoGeneratingColumnEventArgs)
- DataGridRow_MouseDoubleClick(sender : object, e : MouseButtonEventArgs)
ObjectListing
AttributesListing
MainWindow
System.Windows.Controls.Window
SegmentObjectFullDetails
+ SegmentObjectFullDetails(segment : object)
AthleteObjectFullDetails
+ AthleteObjectFullDetails(athlete : object)
ActivitySessionObjectFullDetails
+ ActivitySessionObjectFullDetails(activitySession : object)

Model

Този компонент е основният модел на приложението. Включва в себе си модела на данните на обектите, Класовете тук са описани по-долу:

- IBaseModel - Интерфейс за обозначаване на всички типове модели (обекти), които мога да бъдат визуализирани.
- Athlete - модел на атлет. Полетата на този тип обект са следните:
    - Id : int64 - идентификационен номер на кортеж в базата данни;
    - FirstName : string - първо име;
    - MiddleName : string - бащино име;
    - LastName : string - фамилно име;
    - ActivitiesCount : int - брой тренировки;
    - Followers : int - брой последователи;
    - Following : int - брой атлети, които е последвал.
- ActivitySession - модел на тренировъчна сесия. Полетата на този тип обект са следните:
    - Id : int64 - идентификационен номер на кортеж в базата данни;
    - Distance : double - изминато разстояние, в метри;
    - ElevationGain : int - изкачена надморска височина, в метри;
    - MaxElevation : int - най-висока достигната надморска височина, в метри;
    - TimeMovingMinutes : int - време в активно движение, минути;
    - TotalTimeMinutes : int - тотално време на тренировката, минути;
    - KudosCount : int - брой получена поздравления за тренировката.
- Segment - модел на сегмент. Полетата на този тип обект са следните:
    - Id : int64 - идентификационен номер на кортеж в базата данни;
    - Length : double - дължина на сегмента, в метри;
    - BestEffortTimes : int - най-добро постигнато време, в минути;
    - StartLocation : string - начална точка на сегмента;
    - EndLocation : string - крайна точка на сегмента.
- DatabaseContext - контекст на базата данни. Наследява System.Data.Entity.DbContext, като съдържа в себе си свойства от тип DbSet за всеки един тип обект, който се поддържа за визуализиране. Връзката към базата данни се осъществява чрез Connection String в конструктора на класа.
- DatabaseManager - Имплементира функионалността за работа с базата данни. Към момента съдържа следните методи:
    - public List<Athlete> GetAthletes() - връща всички налични атлети от базата данни;
    - public List<ActivitySession> GetActivitySessions() - връща всички налични тренировъчни сесии от базата данни;
    - public List GetSegments() - връща всички налични сегменти от базата данни.

ViewModel

ViewModel е средният слой в архитектурната структура на приложението. Използва се за връзка между другите два слоя - View и Model. В този слой се използва логиката на Model и Store, след което се предоставят необходимите данни на съответния изглед (View), където се реализира самата визуализация. Класовете в този слой са описани по-долу:

- BaseViewModel - базов клас, който наследява INotifyPropertyChanged интерфейса. Съдържа поле от тип събитие (event PropertyChangedEventHandler) и виртуален метод, който служи за изпълняване на това събитие (virtual void OnPropertyChanged).
- MainViewModel - това е основният ViewModel, който се инициализира при стартирането на приложението в App.xaml.cs с параметър от тип ObjectsAttributesSelectionViewModel, и се подава като DataContext на основния прозорец на приложението MainWindow, където се осъществава навигацията между отделните екрани и проследяването на текущия заложен ViewModel (CurrentViewModel).
- ObjectsAttributesSelectionViewModel - този клас съдържа поле от тип ObjectListingViewModel. Целта на този клас е да бъде с междинна роля, с цел използването на компоненти при визуализацията на данните във View слоя. В конструктора този клас създава обект от тип ObjectsListingViewModel.
- ObjectsListingViewModel - съдържа ObservableCollection списък с типовете наличните типове обекти, които се поддържат за визуализация, както и команда ContinueToObjectAttributesCommand, която се използва за преминаване към следващия екран.
- ObjectsListingItemViewModel - клас, който съдържа данните, които са необходими при визуализацията на типовете обекти. Това са:
    - public string ObjectType { get; set; };
    - public bool IsObjectSelectedForDisplay { get; set; }.
- AttributesSelectionViewModel - този клас съдържа поле от тип AttributesListingViewModel. Целта на този клас е да бъде с междинна роля, с цел използването на компоненти при визуализацията на данните във View слоя. В конструктора този клас създава обект от тип AttributesListingViewModel.
- ListingViewModel - съдържа следните полета:
    - ObservableCollection - списък с атрибутите на избрания тип обект за визуализация;
    - ContinueToSingleObjectListCommand - команда, която се използва за преминаване към следващия екран;
    - ReturnToObjectSelectionCommand - команда, която се използва за връщане към предишния екран за преизбиране на друг тип обект при желание или объркване на потребителя.

- AttributesListingItemViewModel - клас, който съдържа данните, които са необходими при визуализацията на типовете обекти. Това са:
    - public string AttributeName { get; set; };
    - public bool IsAttributeSelectedForDisplay { get; set; }.
- SingleObjectSegmentListingViewModel - клас, който съдържа данните, които са необходими при визуализацията на всички редове от таблицата за обект от тип Segment. Това са:
    - ObservableCollection _signleObjectSegmentListingItemViewModels
    - ObservableCollection<AttributesLIstingItemViewModel>_selectedAttributes
    - ICommand ReturnToObjectSelectionCommand
    - ICommand ReturnToAttributesSelectionCommand
- SingleObjectAthleteListingViewModel - клас, който съдържа данните, които са необходими при визуализацията на всички редове от таблицата за обект от тип Athlete. Това са:
    - ObservableCollection<Athlete> _signleObjectSegmentListingItemViewModels
    - ObservableCollection<AttributesLIstingItemViewModel>_selectedAttributes
    - ICommand ReturnToObjectSelectionCommand
    - ICommand ReturnToAttributesSelectionCommand
- SingleObjectActivitySessionListingViewModel - клас, който съдържа данните, които са необходими при визуализацията на всички редове от таблицата за обект от тип ActivitySession. Това са:
    - ObservableCollection<ActivitySession> _signleObjectSegmentListingItemViewModels
    - ObservableCollection<AttributesLIstingItemViewModel>_selectedAttributes
    - ICommand ReturnToObjectSelectionCommand
    - ICommand ReturnToAttributesSelectionCommand

Command

Този компонент на приложението съдържа класовете за реализиране на различните команди за навигация в приложението, залагане на съответния текущ ViewModel с цел визуализация на правилния екран в потребителския интерфейс. Тук се съдържат следните класове:

- BaseCommand - наследява интерфейса ICommand. Съдържа поле от топ събитие (event EventHandler) и три метода:
    - public virtual bool CanExectute();
    - Public abstract void Execute(object parameter);
    - protected void OnCanExecuteChanged().
- NavigateCommand - наследява BaseCommand и имплементира Execute метода. Тази команда взима конкретния обект (Athlete, Segment или ActivitySession) на базата на

параметър, който се подава от изгледа и е със стойност един от тези типове обекти. След това се взимат всички атрибути на селектирания обект и се запазват в списък. Накрая стойността на текущия ViewModel се залага да бъде равна на обект от тип AttributesSelectionViewModel. Когато се случи промяната на текущия ViewModel се визуализира и съответния за него изглед. Съответствието между ViewModel и View е заложено в <DataTemplate> контролата в основния прозорец (MainWindow.xaml). В случая, когато потребителят не е избрал нито един от типовете обекти, се показва предупредително съобщение с необходимото пояснение към потребителя за избор на тип обект за визуализация.

- SingleObjectListingCommand - наследява BaseCommand и имплементира Execute метода. На базата на селектирания тип обект за визуализация, тази команда инициализира обект от тип (SingleObject**Segment**ListingViewModel, SingleObject**Athlete**ListingViewModel или SingleObject**ActivitySessionListing**ViewModel) и се обновява стойността на текущия ViewModel на новоинициализирания обект. В случая, когато потребителят не е избрал нито един от атрибутите на избрания тип обекти, се показва предупредително съобщение с необходимото пояснение към потребителя за избор на поне един атрибут за визуализация.
- ReturnToObjectSelectionCommand - наследява BaseCommand и имплементира Execute метода. Тази команда служи за връщане към изгледа за избор на тип обект за визуализация в случай, че потребителят е избрал грешен такъв и желае да поправи избора си.
- ReturnToAttributesSelectionCommand - наследява BaseCommand и имплементира Execute метода. Тази команда служи за връщане към изгледа за избор на атрибути от обекта избран за визуализация в случай, че потребителят е избрал грешни такива и желае да поправи избора си. Това се случва чрез използване на текущия заложен ViewModel и посредством Reflection се извличат имената на атрибутите от него.

Factory (InstantiationManager)

Този компонент на приложението е спомагателен. Използва се за съпоставяне на създаване на обект от конкретния селектиран такъв в началния изглед на системата. Съдържа следните класове:

- ICreator<T> - генеричен (generic) интерфейс
  - bool CanHandle(string type);
  - T Create().
- AthleteCreator : ICreator<Athlete> - този клас служи за създаване на обекти от тип Athlete, като имплементира двата метода на интерфейса майка

- ActivitySessionCreator : ICreator<ActivitySession> - този клас служи за създаване на обекти от тип ActivitySession, като имплементира двата метода на интерфейса майка
- SegmentCreator : ICreator - този клас служи за създаване на обекти от тип Segment, като имплементира двата метода на интерфейса майка
- InstanceMapperToModels - това е статичен клас, който имплементира следните два метода:
    - public static IBaseModel MapInstanceToMode(string objectType) - целта на този статичен метод е на базата на подадения като параметър тип обект да създаде обект от този тип. Това се реализира по следния начин. Първоначално се взимат всички класове, които са в namespace “DynamicObjectListing.InstantiationManager”. След това с помощта на цикъл се преминава през всеки един клас и се проверява дали е наследник на интерфекса ICreator, като по този начин се филтрират само Factory класовете. След това за текущия създаден creator клас (AthleteCreator, ActivitySessionCreator или SegmentCreator) се изпълнява метода CanHandle(objectType), с който се проверява дали текущият factory клас поддържа създаването на инстанции от тип objectType. Ако този метод върне разултат “true” се преминава към създаване на обект от подадения на функцията тип и се приключва изпълнението на метода. В противен случай се продължава към следващият creator factory клас, докато не се открие такъв, който поддържа създаването на инстанции от този тип или до приключване на цикъла;
    - public static List<string> GetAllCurrentlyAvailableObjectTypes() - целта на този статичен метод е да извлече всички типове класове, които могат да бъдат визуализирани. Функционалността е подобна на горният метод. Взимат се всички класове от namespace “DynamicObjectListing.InstantiationManager”. След това се преминава през всеки един от тях с помощта на цикъл и се проверява дали са наследници на ICreator интерфейса. Ако това е вярно, към списъка с поддържани типове обекти за визуализация се добавя текущия тип. Накрая се връща резултатния списък със всички поддържани типове обекти за визуализация.

Store

Този компонент на системата също е спомагателен. Съдържа само един клас - NavigationStore. Целта на този клас е да пази информация за текущо използвания ViewModel, на базата на който и се визуализира съответното правилно View. Този клас се използва от MainViewModel и от всички команди за залагане на текущо използвания ViewModel. Във MainViewModel е имплементирано събитие, което следи за промяната на стойността на текущия ViewModel в NavigationStore и на базата на него отразява

стойността и на собственото си поле navigationStore, спрямо което в основния прозорец на системата се визуализират различните изгледи с принадлежащите им ViewModels.

View

Този с слой служи за визуализирането на данните в потребителския интерфейс, предоставени от съответните ViewModel. Реализирани са с помощта на XAML и code-behind в малък брой от изгледите. Класовете в този слой са следните:

- MainWindow.xaml - това е основният прозорец на приложението. В него с помощта на <DataTemplate> и <ContentControl> контроли се задава връзката между изгледите и съответните ViewModel обекти, за да бъде възможна правилната визуализация на желаните данни.
- ObjectsAttributesSelectionView.xaml - този изглед служи за списъчно визуализиране на поддържаните типове обекти за визуализация. Изборът се осъществява чрез селектиране на конкретен ред и натискане на бутон в горната част на изгледа.
    - Ред 0, колона 0 - TextBlock заглавие на изгледа.
    - Ред 1, колона 0 - следващо ниво таблица с 1 колона, в която се внедрява ObjectListing компонентния изглед
- ObjectListing.xaml
    - Ред 0, колона 0 - Command атрибута на бутона е свързан към ContinueToObjectAttributesCommand командата;
    - Ред 1 - listView контрола за визуализиране на всички поддържани типове обекти, като ItemSource атрибута се свързва към ObjectsListingItemViewModels списъка чрез Binding. Всеки ред от ListView контролата се състои от TextBlock контрола, която съдържа името на типа обект за визуализация.
- AttributesSelectionView.xaml - този изглед служи за списъчно визуализиране на атрибутите на избрания обект за визуализация. Изборът се осъществява чрез селектиране на атрибутите с помощта на checkbox бутони и натискане на бутон в горната част на изгледа.
    - Ред 0, колона 0 - TextBlock заглавие на изгледа.
    - Ред 1, колона 0 - следващо ниво таблица с 1 колона, в която се внедрява AttributesListing компонентния изглед
- AttributesListing.xaml
    - Ред 0, колона 0 - бутон за връщане към изгледа за избора на тип обект за визуализация. Command атрибута на бутона е свързан към ReturnToObjectSelectionCommand командата;
    - Ред 0, колона 1 - бутон за визуализиране на следващия изглед. Command атрибута на бутона е свързан към ContinueToSingleObjectListCommand командата;

    - Ред 1 - listView контрола за визуализиране на всички атрибути на избрания обект, като ItemSource атрибута се свързва към AttributesListingItemViewModels списъка чрез Binding. Всеки ред от ListView контролата се състои от TextBlock контрола, която съдържа името на типа обект за визуализация и RibbonCheckbox контрола за избор на съответния атрибут.
- SingleObjectSegmentSelectionView.xaml
    - Ред 0, колона 0 - TextBlock заглавие на изгледа
    - Ред 1 - следващо ниво на таблицата
        - Ред 0, колона 0 - бутон за връщане към изгледа за избора на тип обект за визуализация. Command атрибута на бутона е свързан към ReturnToObjectSelectionCommand командата;
        - Ред 0, колона 1 - бутон за връщане към изгледа за избора на атрибути на селектирания тип обект. Command атрибута на бутона е свързан към ReturnToAtributesSelectionCommand командата;
        - Ред 1 - DataGrid контрола за визуализиране на избрания тип обект и конкретните атрибути, които са посочени. ItemSource атрибута е свързан към SingleObjectSegmentListingItemViewModels списъкът посредством Binding. OnAutoGeneratingColumn атрибута е свързан към OnAutoGenerating метода в code-behind на изгледа.
- SingleObjectSegmentSelectionView.xaml.cs
    - private Dictionary<string, string> propertyNameToHeaderMapping - тази променлива служи за залагане на лесно четими (Human Readable) имена на заглавията на колоните.
    - private void OnAutoGeneratingColumn(Object sender, DataGridAutoGeneratingColumnEventArgs e) - този метод изпълнява се при визуализацията на всеки ред от DataGrid контролата и се използва за скриване от визуализацията на атрибутите, които не са били избрани за визуализация от потребителя.
    - DataGridRow_MouseDoubleClick(object sender, MouseButtonEventArgs e) - този метод се използва за визуализацията на прозорец с пълната информация (данни за всичките атрибути на обекта, независимо какви атрибути сме селектирали първоначално) за реда от таблицата, който се избрали.
- SingleObjectAthleteSelectionView.xaml
    - Ред 0, колона 0 - TextBlock заглавие на изгледа
    - Ред 1 - следващо ниво на таблицата
        - Ред 0, колона 0 - бутон за връщане към изгледа за избора на тип обект за визуализация. Command атрибута на бутона е свързан към ReturnToObjectSelectionCommand командата;

        - Ред 0, колона 1 - бутон за връщане към изгледа за избора на атрибути на селектирания тип обект. Command атрибута на бутона е свързан към ReturnToAtributesSelectionCommand командата;
        - Ред 1 - DataGrid контрола за визуализиране на избрания тип обект и конкретните атрибути, които са посочени. ItemSource атрибута е свързан към SingleObjectAthleteListingItemViewModels списъкът посредством Binding. OnAutoGeneratingColumn атрибута е свързан към OnAutoGenerating метода в code-behind на изгледа.
- SingleObjectActivitySessionSelectionView.xaml.cs
    - private Dictionary<string, string> propertyNameToHeaderMapping - тази променлива служи за залагане на лесно четими (Human Readable) имена на заглавията на колоните.
    - private void OnAutoGeneratingColumn(Object sender, DataGridAutoGeneratingColumnEventArgs e) - този метод изпълнява се при визуализацията на всеки ред от DataGrid контролата и се използва за скриване от визуализацията на атрибутите, които не са били избрани за визуализация от потребителя.
    - DataGridRow_MouseDoubleClick(object sender, MouseButtonEventArgs e) - този метод се използва за визуализацията на прозорец с пълната информация (данни за всичките атрибути на обекта, независимо какви атрибути сме селектирали първоначално) за реда от таблицата, който се избрали.
- SingleObjectActivitySessionSelectionView.xaml
    - Ред 0, колона 0 - TextBlock заглавие на изгледа
    - Ред 1 - следващо ниво на таблицата
        - Ред 0, колона 0 - бутон за връщане към изгледа за избора на тип обект за визуализация. Command атрибута на бутона е свързан към ReturnToObjectSelectionCommand командата;
        - Ред 0, колона 1 - бутон за връщане към изгледа за избора на атрибути на селектирания тип обект. Command атрибута на бутона е свързан към ReturnToAtributesSelectionCommand командата;
        - Ред 1 - DataGrid контрола за визуализиране на избрания тип обект и конкретните атрибути, които са посочени. ItemSource атрибута е свързан към SingleObjectActivitySessionListingItemViewModels списъкът посредством Binding. OnAutoGeneratingColumn атрибута е свързан към OnAutoGenerating метода в code-behind на изгледа.
- SingleObjectActivitySessionSelectionView.xaml.cs
    - private Dictionary<string, string> propertyNameToHeaderMapping - тази променлива служи за залагане на лесно четими (Human Readable) имена на заглавията на колоните.

    - private void OnAutoGeneratingColumn(Object sender, DataGridAutoGeneratingColumnEventArgs e) - този метод изпълнява се при визуализацията на всеки ред от DataGrid контролата и се използва за скриване от визуализацията на атрибутите, които не са били избрани за визуализация от потребителя.
    - DataGridRow_MouseDoubleClick(object sender, MouseButtonEventArgs e) - този метод се използва за визуализацията на прозорец с пълната информация (данни за всичките атрибути на обекта, независимо какви атрибути сме селектирали първоначално) за реда от таблицата, който се избрали.
- SegmentObjectFullDetails.xaml
    - TextBlock - дължина на сегмента, в метри
    - TextBlock - най-добро постигнато време, в минути
    - TextBlock - начална точка на сегмента
    - TextBlock - крайна точка на сегмента
- AthleteObjectFullDetails.xaml
    - TextBlock - първо име
    - TextBlock - бащино име
    - TextBlock - фамилно име
    - TextBlock - брой тренировъчна сесии
    - TextBlock - брой последователи
    - TextBlock - брой последвани атлети
- AthleteObjectFullDetails.xaml
    - TextBlock - изминато разстояние, в метри
    - TextBlock - изкачена надморска височина, в метри
    - TextBlock - максимална достигната надморска височина, в метри
    - TextBlock - време в движение, в минути
    - TextBlock - тотално време, в минути
    - TextBlock - брой получени поздравления

Ръководство за програмиста

Сценарият за последващо развитие на приложение е следното: **добавяне на нов обект за визуализация**. Необходимо е да се използва среда за разработване с инсталиране Microsoft.NetCore.App 3.1.0, Microsoft.WindowsDesktop.App.WPF 3.1.0, също така и Nugget пакета Entity Framework 6.4.4.

Стъпките за по-нататъшна разработка стъпват върху Code-First подхода при създаване на таблица с обекти в базата данни и биват:

1. Добавяне на клас, описващ новият тип обект (Model). Класът трябва да наследява IBaseModel интерфейса, да има анотация [Table(“table_name”)], която указва името на таблицата, която ще бъде създадена в базата данни и анотация

[DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity), Key()] за Id полето на класа;

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.ComponentModel.DataAnnotations.Schema;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicObjectListing.Model
{
    [Table("session_analysis")]
    2 references
    public class SessionAnalysis : IBaseModel
    {
        [DatabaseGeneratedAttribute(DatabaseGeneratedOption.Identity), Key()]
        1 reference
        public Int64 Id { get; set; }
        1 reference
        public int HeartRateMin { get; set; }
        1 reference
        public int HeartRateMax { get; set; }
        1 reference
        public int HeartRateAvg { get; set; }
        1 reference
        public int PowerAvg { get; set; }
        0 references
        public SessionAnalysis() { }

        0 references
        public SessionAnalysis(Int64 id, int heartRateMin, int hearthRateMax, int hearthRateAvg, int powerAvg)
        {
            this.Id = id;
            this.HeartRateMin = heartRateMin;
            this.HeartRateMax = hearthRateMax;
            this.HeartRateAvg = hearthRateAvg;
            this.PowerAvg = powerAvg;
        }
    }
}
```

2. В namespace “**DynamicObjectListing.Model**”, в класа **DatabaseContext** да се добави публично свойство от тип DbSet<NewCreatedModel>. Конструкторът на DatabaseContext класа извиква конструктора на класа майка (DbContext), като се подава като параметър низ за свързване към базата от данни (Connection String);

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicObjectListing.Model
{
    7 references
    internal class DatabaseContext : DbContext
    {
        1 reference
        public DatabaseContext() : base("Data Source=DESKTOP-BUOKINV;Initial Catalog=ps_object_listing;Integrated Security=True") { }

        2 references
        public DbSet<Athlete> Athletes { get; set; }
        2 references
        public DbSet<ActivitySession> ActivitySessions { get; set; }
        2 references
        public DbSet<Segment> Segments { get; set; }
        0 references
        public DbSet<SessionAnalysis> SessionAnalyses { get; set; }
    }
}
```

3. В namespace “**DynamicObjectListing.Model**”, в класа **DatabaseManager** да се добави публичен метод, който да връща колекция от тип List<NewCreatedModel>, която съдържа всички записи от базата данни, които да бъдат предоставени за визуализация;

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicObjectListing.Model.Databse
{
    4 references
    public class DatabaseManager
    {
        DatabaseContext DbContext;

        3 references
        public DatabaseManager()
        {
            DbContext = new DatabaseContext();
        }

        1 reference
        public List<Athlete> GetAthletes()
        {
            return DbContext.Athletes.ToList();
        }
        1 reference
        public List<ActivitySession> GetActivitySessions()
        {
            return DbContext.ActivitySessions.ToList();
        }
        1 reference
        public List<Segment> GetSegments()
        {
            return DbContext.Segments.ToList();
        }
        0 references
        public List<SessionAnalysis> GetSessionAnalyses()
        {
            return DbContext.SessionAnalyses.ToList();
        }
    }
}
```

4. Обновяване на базата данни в съответствие с новосъздадения модел:
   4.1. Изпълняване на “Enable-Migrations” ? - за активиране на функционалност за създаване на миграции;
   4.2. Изпълняване на “Add-Migration <Migration_Name>” - за създаване на клас за миграция с подходящо име.
   4.3. Изпълняване на “Update-Database” - изпълнява последния файл за миграция, който е създаден от предхождащата команда, и прави същинското обновяване на базата данни спрямо инструкциите в този файл.

При правилно изпълнение на посочените 4 стъпки дотук, трябва да бъдат налични следните резултати:

- Създадена нова таблица в базата от данни, на базата на полетата в обекта NewCreatedModel;
- Връзка между приложението и базата данни;
- метод , който връща колекция от новия обект NewCreatedModel, съдържаща всички записи от базата данни в новата таблица.

```
namespace DynamicObjectListing.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    2 references
    public partial class adding_session_analysis_table : DbMigration
    {
        0 references
        public override void Up()
        {
            CreateTable(
                "dbo.session_analysis",
                c => new
                    {
                        Id = c.Long(nullable: false, identity: true),
                        HeartRateMin = c.Int(nullable: false),
                        HeartRateMax = c.Int(nullable: false),
                        HeartRateAvg = c.Int(nullable: false),
                        PowerAvg = c.Int(nullable: false),
                    })
                .PrimaryKey(t => t.Id);

        }

        0 references
        public override void Down()
        {
            DropTable("dbo.session_analysis");
        }
    }
}
```

5. В namespace "**DynamicObjectListing.InstantiationManager**" да се добави нов factory клас NewCreatedModelCreator : ICreator<NewCreatedModel>, който наследява ICreator интерфейса и да имплементира двете му функции:
   5.1. public CanHandle(string type) - трябва да връща "true", когато подадения тип обект е равен на NewCreatedMethod;
   5.2. public Create() - създава обект от тип NewCreatedModel.

```
using DynamicObjectListing.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace DynamicObjectListing.InstantiationManager
{
    1 reference
    public class SessionAnalysisCreator : ICreator<SessionAnalysis>
    {
        0 references
        public SessionAnalysisCreator() { }
        1 reference
        public bool CanHandle(string type) { return "SessionAnalysis" == type; }
        1 reference
        public SessionAnalysis Create() { return new SessionAnalysis(); }
    }
}
```

6. В namespace "**DynamicObjectListing.ViewModel**" да се добави клас SingleObjectNewCreatedModelListingViewModel, който да съдържа следните полета:
   6.1. private ObservableCollection<NewCreatedObject> _singleObjecNewCreatedModelListingItemViewModels;
   6.2. private ObservableCollection<AttributesLIstingItemViewModel> _selectedAttributes;

6.3. Public IEnumerable<NewCreatedObject> SingleObjecNewCreatedModelListingItemViewModels - колекция, съдържаща всички записи от таблицата за новия обект в базата данни;

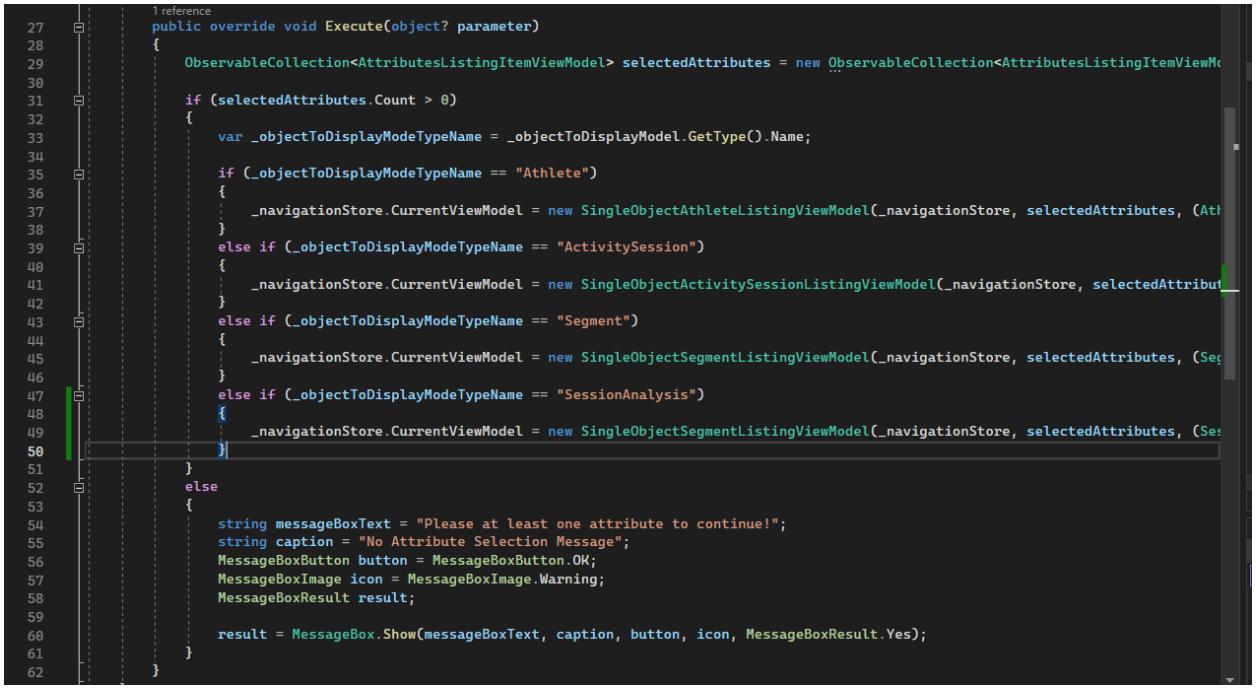
6.4. public IEnumerable<AttributesLIstingItemViewModel> SelectedAttributes - колекция, съдържаща атрибутите, които са избрани за визуализация;

6.5. public ICommand ReturnToObjectSelectionCommand;

6.6. public ICommand ReturnToAttributesSelectionCommand.

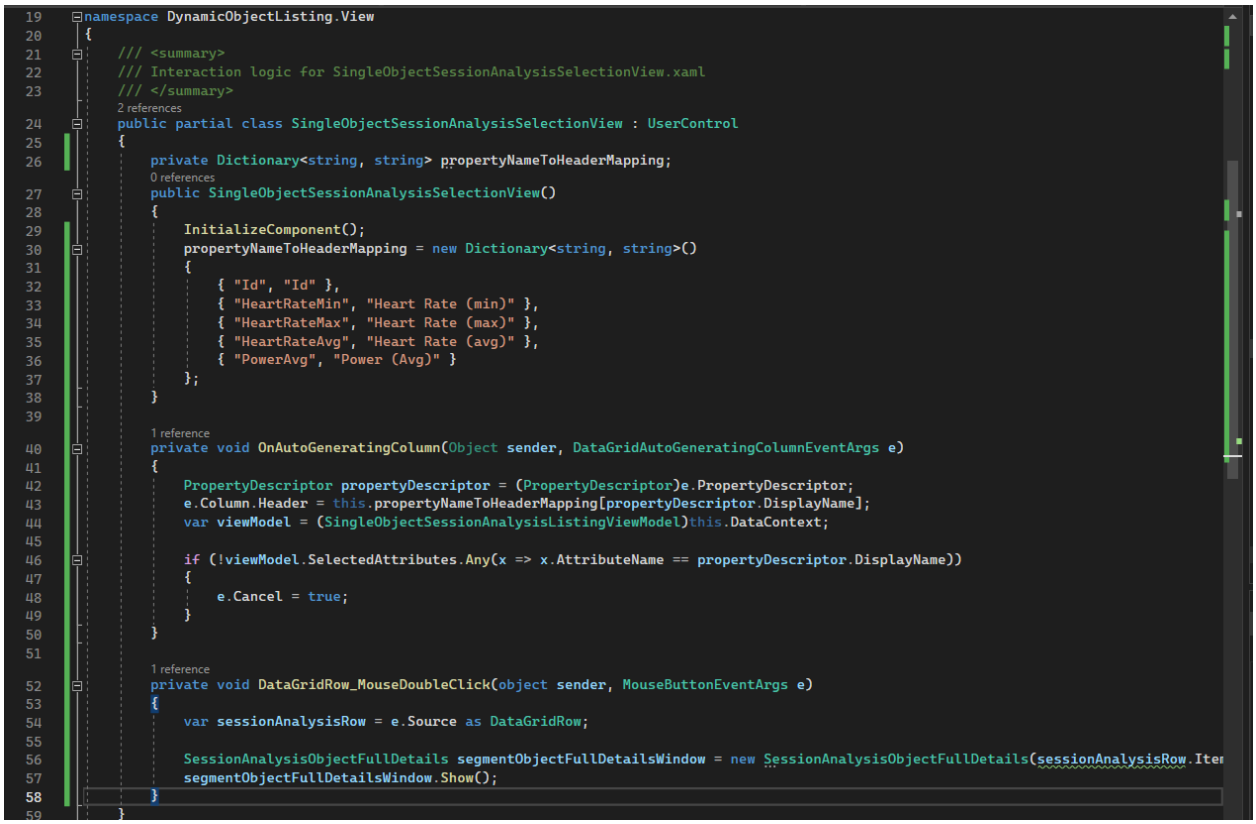
```
using DynamicObjectListing.Command;
using DynamicObjectListing.Model.Database;
using DynamicObjectListing.Model;
using DynamicObjectListing.Store;
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Input;

namespace DynamicObjectListing.ViewModel
{
    1 reference
    public class SingleObjectSessionAnalysisListingViewModel : BaseViewModel
    {
        private readonly ObservableCollection<SessionAnalysis> _singleObjectSegmentListingItemViewModels;
        private readonly ObservableCollection<AttributesListingItemViewModel> _selectedAttributes;

        0 references
        public IEnumerable<SessionAnalysis> SingleObjectSegmentListingItemViewModels => _singleObjectSegmentListingItemViewModels;
        0 references
        public IEnumerable<AttributesListingItemViewModel> SelectedAttributes => _selectedAttributes;
        1 reference
        public ICommand? ReturnToObjectSelectionCommand { get; }
        1 reference
        public ICommand? ReturnToAttributesSelectionCommand { get; }

        0 references
        public SingleObjectSessionAnalysisListingViewModel(NavigationStore navigationStore,
                                                           ObservableCollection<AttributesListingItemViewModel> selectedAttributes,
                                                           IBaseModel objectToDisplayModel)
        {
            _selectedAttributes = selectedAttributes;
            _singleObjectSegmentListingItemViewModels = new ObservableCollection<SessionAnalysis>(new DatabaseManager().GetSessionAnalyses

            ReturnToObjectSelectionCommand = new ReturnToObjectSelectionCommand(navigationStore);
            ReturnToAttributesSelectionCommand = new ReturnToAttributesSelectionCommand(navigationStore, objectToDisplayModel);
        }
    }
}
```

7. В namespace “**DynamicObjectListing.Command**”, в класа “**SingleObjectListingCommand**” да се добави нов else if блок, който да проверява за новият тип обект и да залага стойността на текущия ViewModel да бъде равна на нов обек от тип SingleObjectNewCreatedModelListingViewModel.



```
1 reference
public override void Execute(object? parameter)
{
    ObservableCollection<AttributesListingItemViewModel> selectedAttributes = new ObservableCollection<AttributesListingItemViewM

    if (selectedAttributes.Count > 0)
    {
        var _objectToDisplayModeTypeName = _objectToDisplayModel.GetType().Name;

        if (_objectToDisplayModeTypeName == "Athlete")
        {
            _navigationStore.CurrentViewModel = new SingleObjectAthleteListingViewModel(_navigationStore, selectedAttributes, (Ath
        }
        else if (_objectToDisplayModeTypeName == "ActivitySession")
        {
            _navigationStore.CurrentViewModel = new SingleObjectActivitySessionListingViewModel(_navigationStore, selectedAttribut
        }
        else if (_objectToDisplayModeTypeName == "Segment")
        {
            _navigationStore.CurrentViewModel = new SingleObjectSegmentListingViewModel(_navigationStore, selectedAttributes, (Seg
        }
        else if (_objectToDisplayModeTypeName == "SessionAnalysis")
        {
            _navigationStore.CurrentViewModel = new SingleObjectSegmentListingViewModel(_navigationStore, selectedAttributes, (Ses
        }
    }
    else
    {
        string messageBoxText = "Please at least one attribute to continue!";
        string caption = "No Attribute Selection Message";
        MessageBoxButton button = MessageBoxButton.OK;
        MessageBoxImage icon = MessageBoxImage.Warning;
        MessageBoxResult result;

        result = MessageBox.Show(messageBoxText, caption, button, icon, MessageBoxResult.Yes);
    }
}
```

8. В namespace “**DynamicObjectListing.View**” да се добави изглед (клас) тип UserControl с името **SingleObjectNewCreatedModelView.** ItemSource атрибута на DataGrid контролата чрез Binding се свързва към следното поле от новия ViewModel: SingleObjecNewCreatedModelListingItemViewModels. В code-behind на изгледа се добавят следните два метода:
   8.1. Private void OnAutoGenerating Column(object sender, DataGridAutoGeneratingColumnEventArgs e) - служи за проверка на колоните и визуализиране само на посочените такива;
   8.2. Private void DataGridRow_MouseDoubleClick(object sender, MouseButtonEventArgs e) - служи за отваряне на диалогов прозорец с пълната информация за посочения ред от таблицата.



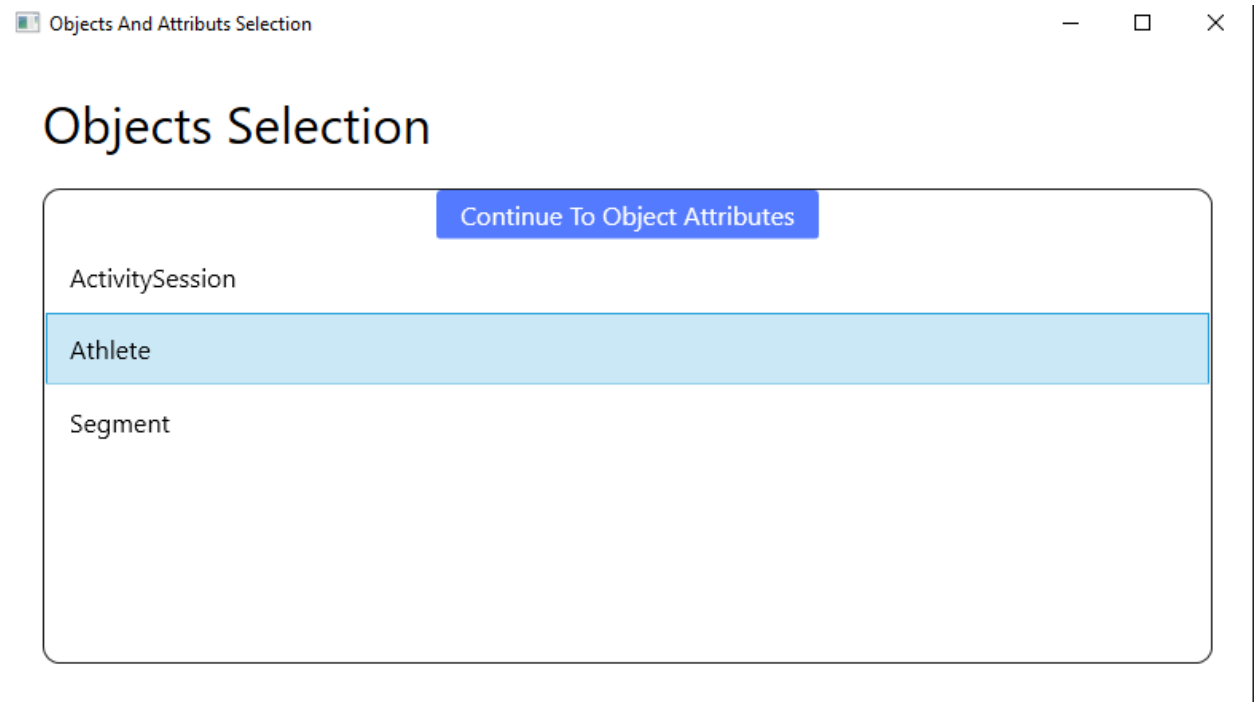
```
namespace DynamicObjectListing.View
{
    /// <summary>
    /// Interaction logic for SingleObjectSessionAnalysisSelectionView.xaml
    /// </summary>
    2 references
    public partial class SingleObjectSessionAnalysisSelectionView : UserControl
    {
        private Dictionary<string, string> propertyNameToHeaderMapping;
        0 references
        public SingleObjectSessionAnalysisSelectionView()
        {
            InitializeComponent();
            propertyNameToHeaderMapping = new Dictionary<string, string>()
            {
                { "Id", "Id" },
                { "HeartRateMin", "Heart Rate (min)" },
                { "HeartRateMax", "Heart Rate (max)" },
                { "HeartRateAvg", "Heart Rate (avg)" },
                { "PowerAvg", "Power (Avg)" }
            };
        }

        1 reference
        private void OnAutoGeneratingColumn(Object sender, DataGridAutoGeneratingColumnEventArgs e)
        {
            PropertyDescriptor propertyDescriptor = (PropertyDescriptor)e.PropertyDescriptor;
            e.Column.Header = this.propertyNameToHeaderMapping[propertyDescriptor.DisplayName];
            var viewModel = (SingleObjectSessionAnalysisListingViewModel)this.DataContext;

            if (!viewModel.SelectedAttributes.Any(x => x.AttributeName == propertyDescriptor.DisplayName))
            {
                e.Cancel = true;
            }
        }

        1 reference
        private void DataGridRow_MouseDoubleClick(object sender, MouseButtonEventArgs e)
        {
            var sessionAnalysisRow = e.Source as DataGridRow;

            SessionAnalysisObjectFullDetails segmentObjectFullDetailsWindow = new SessionAnalysisObjectFullDetails(sessionAnalysisRow.Ite
            segmentObjectFullDetailsWindow.Show();
        }
    }
```

9. В “**MainWindow.xaml**” да се добави DataTemplate контрола за новосъздадените ViewModel и View.

```
    <Grid Margin="25">
        <Grid.Resources>
            <DataTemplate DataType="{x:Type vms:ObjectsAttributesSelectionViewModel}">
                <view:ObjectsAttributesSelectionView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type vms:AttributesSelectionViewModel}">
                <view:AttributesSelectionView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type vms:SingleObjectSegmentListingViewModel}">
                <view:SingleObjectSegmentSelectionView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type vms:SingleObjectAthleteListingViewModel}">
                <view:SingleObjectAthleteSelectionView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type vms:SingleObjectActivitySessionListingViewModel}">
                <view:SingleObjectActivitySessionSelectionView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type vms:SingleObjectSessionAnalysisListingViewModel}">
                <view:SingleObjectSessionAnalysisSelectionView />
            </DataTemplate>
        </Grid.Resources>

        <ContentControl Content="{Binding CurrentViewModel}" />
    </Grid>
/Window>
```

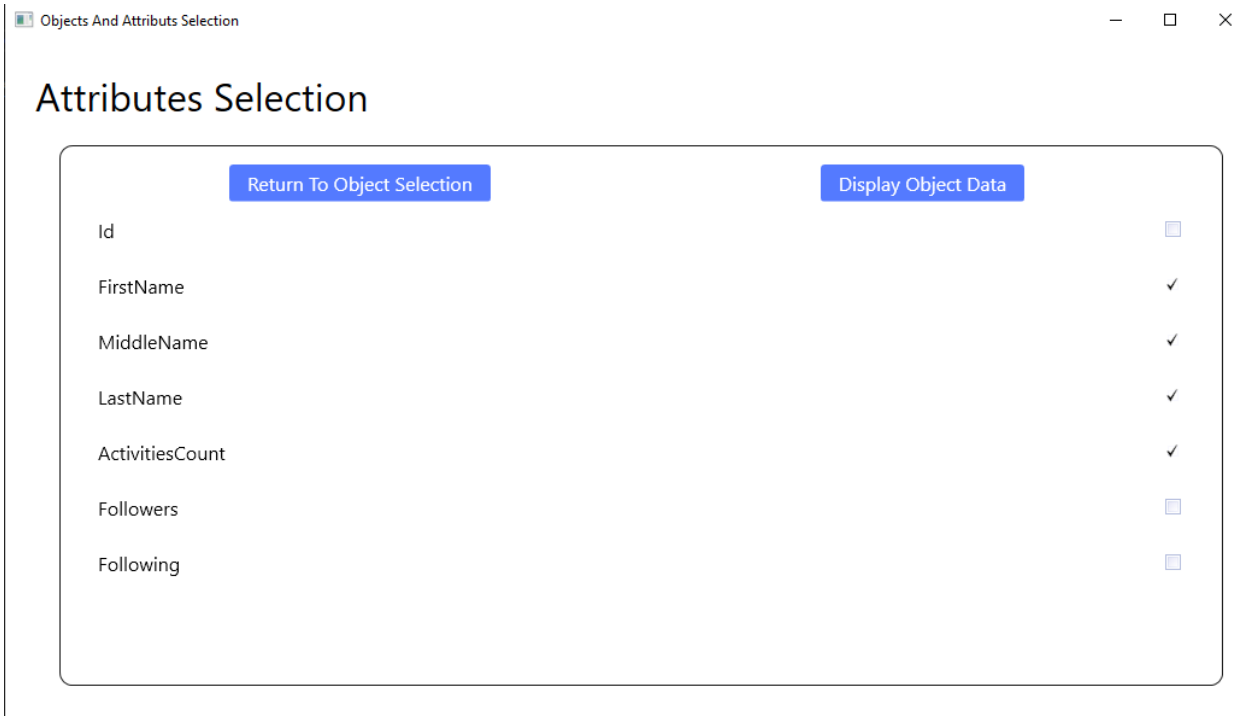
Демонстрация на работоспособността

На *фигура 1.* е показан първоначалният изглед за избор на тип обект за визуализация. Към момента приложението поддържа следните 3 типа обекта за визуализация: Athlete, ActivitySession, Segment. В този случай по време на демонстрацията се избира обект от тип Athlete за визуализация.



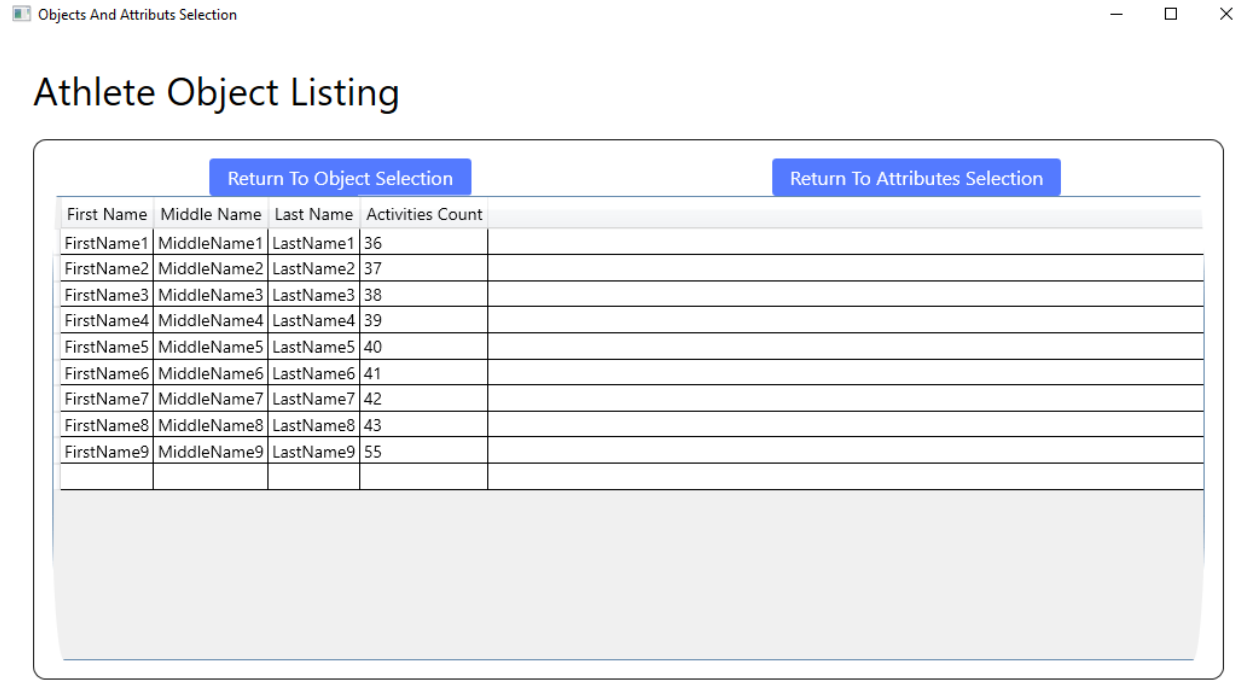
фигура 1. Главен (начален изглед на приложението)

На *фигура 2.* е представен изгледа за визуализиране на всички атрибути на предходно селектирания тип обект. Както се вижда, при избиране на обект от тип Athlete в следващия изглед се визуализират неговите атрибути. С цел демонстрация са избрани следните атрибути за визуализация: FirstName, MiddleName, LastName, ActivitiesCount.



фигура 2. Изглед за избиране на атрибути на предходно селектиран тип обект

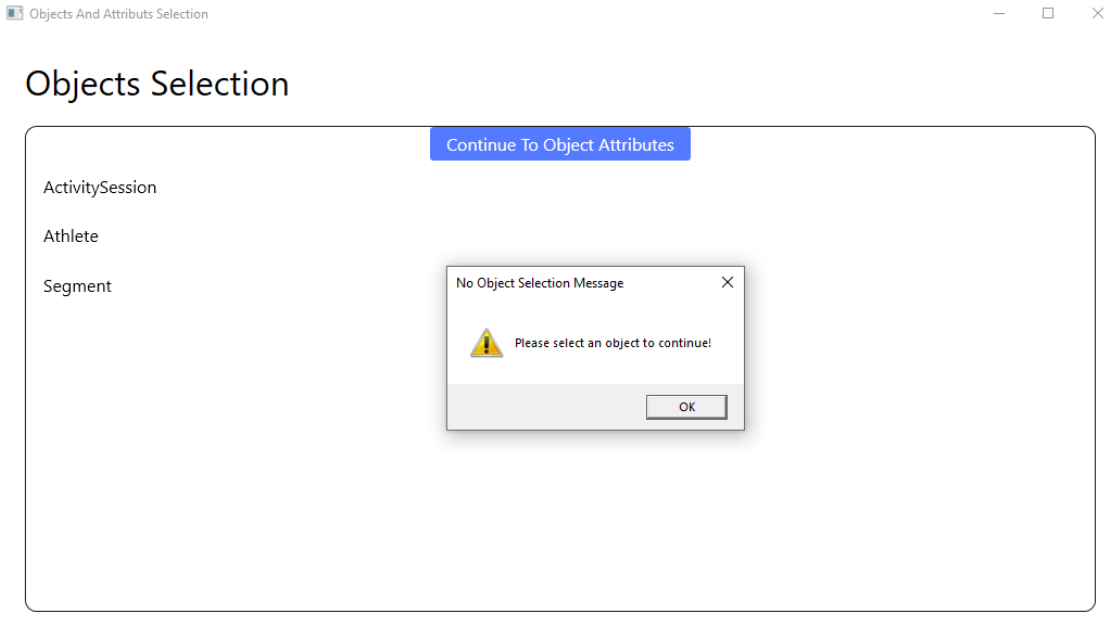
При натискането на бутон “Display Object Data” се визуализира изгледът, показан на *фигура 3.* Както се вижда, това са всички редове от таблицата в базата данни за този тип обект.



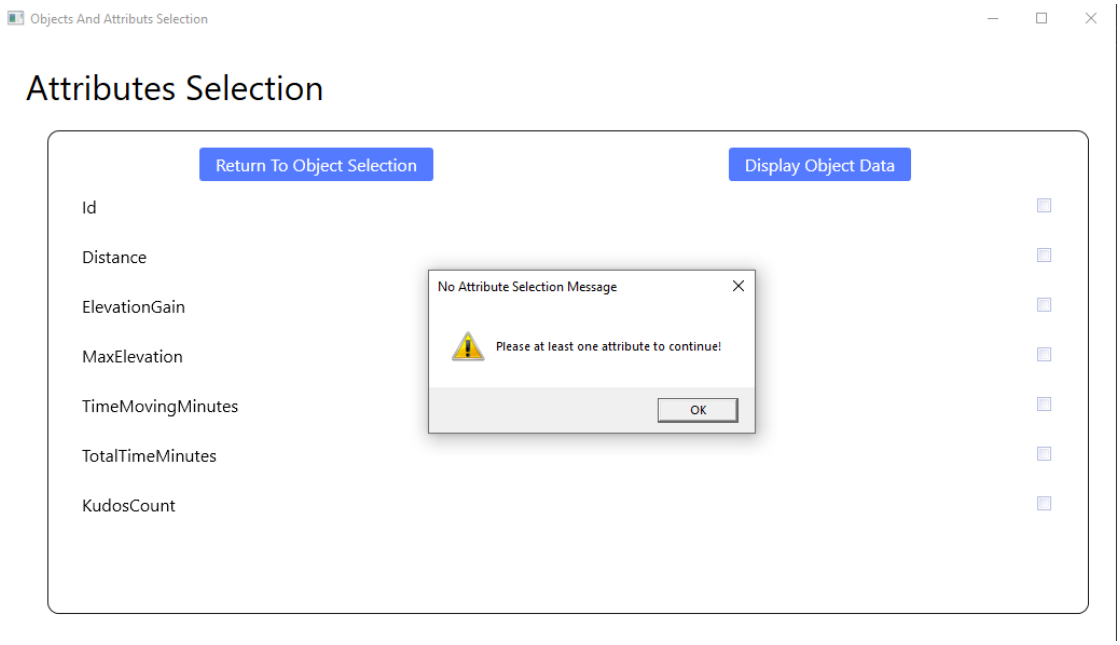
фигура 3. Таблично визуализиране на избраните атрибути на посочения обект

При натискане на бутон “Return To Attributes Selection” се визуализира изгледът със списък на всички атрибути на посочения тип обект. При натискане на бутон “Return To Object Selection” се визуализира изгледът със списък на всички налични типове обекти, които се поддържат.

На *фигури 4. и 5.* е показано поведението на приложението при неизбран тип обект за визуализация или неизбрани атрибути за визуализация.



фигури 4. Грешка при невъведен тип обект за визуализация



фигури 5. Грешка при невъведени атрибути за визуализация