**BACHELOR THESIS**

**Workout Journal – a mobile fitness-trainer application**

**Student**: **VLADIMIR ROSSENOV YANKOV Matriculation Nr**. 201210042

**Subject:** Informatics

Consultant of the bachelor thesis: Supervisor of the bachelor:

(Assoc. Prof. Doz. Virginia Dimitrova) (Assoc. Prof. Doz. Virginia Dimitrova)

Student: Vice Dean CST:

(Vladimir Yankov) (Prof. Dr. Ing. Aleksandar Tsenov)

Contents

[1 Introduction 3](#_Toc389518990)

[2 Requirements 5](#_Toc389518991)

[2.1 Customer requirements 5](#_Toc389518992)

[2.2 Architectural requirements 6](#_Toc389518995)

[2.3 Structural requirements 7](#_Toc389518996)

[2.4 Functional requirements 7](#_Toc389518998)

[2.5 Performance requirements 8](#_Toc389519002)

[2.6 Design requirements 8](#_Toc389519004)

[2.7 Behavioral requirements 8](#_Toc389519006)

[2.8 Use Case Diagram 9](#_Toc389519007)

[2.9 Requirements - Objectives Engineering Table 10](#_Toc389519008)

[3 Design and implementation 11](#_Toc389519009)

[3.1 Design 11](#_Toc389519010)

[3.2 Implementation 19](#_Toc389519014)

[4 Validation and verification 31](#_Toc389519020)

[4.1 Simulation 31](#_Toc389519021)

[4.2 Test results 32](#_Toc389519024)

[5 Demonstration 39](#_Toc389519029)

[5.1 Main view 39](#_Toc389519030)

[5.2 Workouts view 40](#_Toc389519031)

[5.3 Workout exercise view 41](#_Toc389519032)

[5.4 Play view 42](#_Toc389519033)

[5.5 Nutrition view 43](#_Toc389519034)

[5.6 Workout scheduler 44](#_Toc389519035)

[6 Summary 45](#_Toc389519036)

[7 Sources 48](#_Toc389519037)

[8 List of abbreviations 50](#_Toc389519038)

[9 Declaration of Authorship 51](#_Toc389519039)

# Introduction

The following thesis sets out with the development of a native mobile application for Windows Phone 8.0 intended as a personal fitness-trainer and guide. In this section I will briefly describe the reasons why such a solution is needed, its advantages and the offered functionality. Furthermore the used technologies will be presented and assessed for their current state.

Nowadays there are more and more reports for the raising level of people, who are affected from obesity and overweight. Meanwhile there is also a growing number of registered chronic diseases such as cardiovascular disease, diabetes, cancer and depression. These symptoms are influenced by many factors, however some of the most significant are the sedentary lifestyle and the unhealthy diet. The article from sportanddev[[1]](#footnote-1) makes a good case in point that sport and physical activity have a positive impact on health and are used as prevention for many health issues. Moreover the regular physical activity reduces stress levels.

Sport plays a crucial part in our life, so in order to optimize its impact we have to come up with a workout routine, which fits best our personal strengths, abilities and different background factors. This gives rise to the need of an instructor that can organize and improve a specific training according the personal needs. Of course one way to satisfy this need is to use the advice of a professional fitness instructor. However this can be quite expensive and time consuming. These advantages bring about the alternative use of a virtual trainer on the personal smartphones.

The advantages of having a mobile instructor are quite a few. On one hand it is completely free and available at any time. On the other hand it could be highly personalisable, so that it could target a larger audience. Meanwhile we could also mention its portability and mobility.

The presented solution for this problem in the following thesis serves as a personal workout journal and encompasses many of these benefits. The application comes with a variety of workout routines and exercises, which could be edited and in such a way personalized for the specific needs. Furthermore it gives the opportunity to add new items in both sections. The user can not only keep track of his favorite exercises but can also interact with them. The trainer offers a play mode, which simulates a real time workout and also gives advises about the needed rest times.

The implemented mobile instructor helps organizing the physical activity plan with the other tasks thanks to the built in scheduler. The appointed workouts are synced with the smartphone's calendar. This helps creating a personal fitness program. At last but not least the application helps the user to pay attention to his diet plan, pointing out healthy or avoidable foods.

The solution is built for the Windows Phone 8.0 platform. It is a completely native MVVM[[2]](#footnote-2) application, offering fast performance. It uses a local database structure in the isostore[[3]](#footnote-3) of the phone. In such a way it is independent of the mobile provider in means of internet traffic. The implemented ORM[[4]](#footnote-4) is LINQ to SQL[[5]](#footnote-5). The relational database is kept as a \*.sdf[[6]](#footnote-6) file. The application uses also the Windows Phone Toolkit. It is developed with Visual Studio 2013 Ultimate and tested on the built-in emulator and a Nokia Lumia 720 device.

For the time of creation of the application Windows Phone 8 is the newest OS[[7]](#footnote-7) of Microsoft. It provides different built-in and ready-to-use controls and features. It improves security and also gives the opportunity to connect and synchronize many devices using it. It is simultaneously a potential market niche for many developers.

To sum up the thesis has the goal to present and describe the different parts, methodologies and approaches for developing a highly personalisable and interactive fitness trainer as a native Windows Phone application.

Finally I wish to thank Doz. V. Dimitrova for being particularly helpful in guiding and supporting me toward the work. I also wish to thank the members of my committee for their support and patience.

# Requirements

This section focuses on the specific types of requirements, which can be elicitated on account of the target application, described in the introduction. At the end of the section the key requirements will be summarized, implementing a UML[[8]](#footnote-8) Use Case diagram and a table with the derived objectives and tasks.

## Customer requirements

### Targeted audience

The targeted group requirement could be also considered as an allocated requirement, which can be divided into multiple lower-level requirements considering a number of factors and background context. Nevertheless the developed application has the goal to reach a wide range of people. For that purpose the solution must be applicable to a variety of workout routines. Furthermore it also has to encompass many exercises, which can be altered respectively to the specific needs.

Another aspect is to provide not only descriptive interface, but to include also visual examples for better interpretation. The fitness trainer must not be offensive by any means and has to include a warning section that the given exercises are only a guidance and must be dealt with caution. Any forms of excessive physical activity must not be encouraged since they can invoke further health issues. At last the user must be informed thoroughly that he is using the software guidance at his own risk and is responsible for assessing the difficulty of the routines regarding his own strength.  
2.1.2 Availability

The availability requirement can be derived partly from the previous one. As the offered program has the goal to target a larger group of users it has to be offered freely on the market. Regarding the perceptive matter of the development it is better to provide multi-language interface. However writing the application in English (considered as an international language) and offering visual descriptive support is sufficient enough to reach the majority of people.

## Architectural requirements

As mentioned previously in the introduction, the chosen platform for the program is Windows Phone 8 since its growing popularity, the niche in the application's market and the highly developed technology. Subsequently this approach raises a number of architectural system constraints. In order to achieve the task, the developer is limited to use a Windows 8 64-bit OS. Regarding SDK[[9]](#footnote-9) it is advisable to use Visual Studio 2013, however using Windows Phone 8.0 SDK is completely sufficient.

[[10]](#footnote-10)The used development kit raises several system and network requirements regarding the ability to run a simulation on the built-in emulator. As for system - it must provide at least 4 GB or more RAM, the OS must be 64-bit version of Windows 8 Pro edition or higher and the following features must be supported in the BIOS[[11]](#footnote-11): Hardware-assisted virtualization, Second Level Address Translation (SLAT[[12]](#footnote-12)) and Hardware-based Data Execution Prevention (DEP[[13]](#footnote-13)). Further there are two Network requirements - DHCP[[14]](#footnote-14) and Automatically configured DNS and gateway settings. Finally the system must support Hyper-V[[15]](#footnote-15) and the following two must be fulfilled: In Windows, Hyper-V must be enabled and running; the user has to be a member of the local Hyper-V Administrators group.

In order to get the most in testing the application it is advisable that the developer has a Microsoft developer account. In this way he will be able to register a developer device and deploy and test the application directly on it without any submission. This option includes also exclusive access to the latest updates and fixes prior to their public release.

Regarding the best OOP[[16]](#footnote-16) practices, the application must follow an architectural design pattern and clear hierarchical and functional separation of the corresponding classes. The use of appropriate interfaces and abstractions is needed for better encapsulation, polymorphism and code reusability.

## Structural requirements

### Size of the application

The application must implement a database-structured way of preserving the information, so that the data could be quarried, presented and maintained more efficiently. At the same time the developed software's size must not exceed overly since it has to be run on a mobile device with a reasonable performance.

## Functional requirements

### Personalisation

The presented application must provide a variety of ways to personalize the individual workout experience as this is the initial idea for developing the software. For that purpose the corresponding model entities must implement the basic CRUD[[17]](#footnote-17) operations. These operations must be implemented in a variety of ways, offering different perspectives since the personalization requirements are essential for the logic of the program.

### Multimedia

The intended purpose for the developed application is to entertain and provide the user with clear instructions regarding the presented workout routines. Therefore the multimedia requirements encompass the presentation abilities of the software. Exercise entities must be supported by clear know-how multimedia and sufficient description. A further aspect of this section is the implementation of an interactive player to simulate a training routine. It must provide the user in real time with information regarding exercise repeats, rest times and performance details.

### Connectivity

One of the key aspects in nowadays mobile technologies is the diversity of connection potentials. The application offers a good chance to improve further these features, implementing a linked scheduler. The ability to create and manage workout appointments and to bind them with the tasks of the common calendar is very important for the overall impact of the developed software. This opens the possibility for better performance in personal trainings and brings about the need of such a connectivity requirement.

## Performance requirements

### Response time

The performance requirements are essential and play a crucial part for the overall success and popularity of the application. The developed software is intended for everyday use. It is also expected to deliver the acquired information with a glance in minimal time and to present it in a clear and unambiguous way. These requirements stem from the fact that people are able to spend a limited time for workout and interacting with the fitness instructor. Therefore the program is demanded to perform smoothly and without delays.

## Design requirements

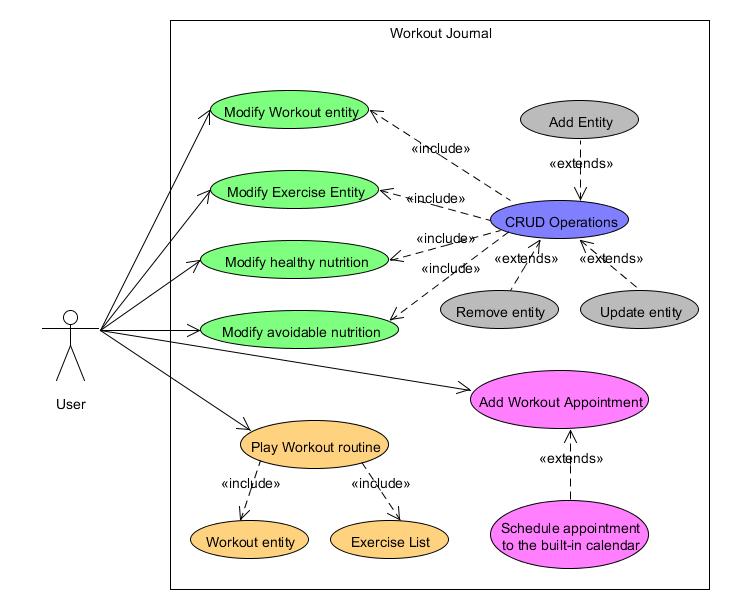
### Simplicity

As mentioned in the previous section the application must offer a user-friendly and resourceful interface. This expectation results in a demand for a productive navigation system, which has to offer quick references to the different views of the program. On the grounds that the software is intended as a mobile application the need for a clear and easily-readable design must be taken into consideration. The design simplicity is also a key factor defining the favors of interacting with the application. In order to satisfy the user needs, the software must provide an overall experience that its use deliver exactly the required content without the presence of redundant information.

## Behavioral requirements

Regarding the behavioral requirements the system must run flawlessly and respond adequate according to the input information. With other words any unwanted exceptions or errors must be avoided or handled in the corresponding code sections. The application must not crash or demand from the user to be restarted. The program must behave in such a way, so that it disposes the unneeded allocated resources and memory. The system has to be persistent and to avoid memory losses. Subsequently the existing or altered data must be stored coherent and cohesively regarding the raised modifications. In order to fulfill this task the application must preserve the changes, while it is not running and to restore the last system state when started again.

## Use Case Diagram



*1.1 UML Use Case Diagram for the application Workout Journal*

The presented Use Case UML Diagram offers a basic notion on the needed functionality, separating it into three main sections. The sections are supported by listing the basic operations, which they use.

## Requirements - Objectives Engineering Table

The following table aims to summarize the crucial requirements and the derived objectives, which the presented solution must fulfill.

|  |  |
| --- | --- |
| Original requirements | Derived Objectives |
| The user must be allowed to personalize the application's entities according his own needs and preferences | Implement basic CRUD operations for the Model classes |
| The user is intended to interact with the application by means of playing a specific workout routine | Include a workout routine play mode and add a real time indicators about workout's cycles, exercises and rest times |
| The application must store cohesive and coherent the presented or modified information units | Implement a relational data base system with the corresponding relationships |
| To the user must be presented the ability to organize the given workout routines with his own tasks | Add a basic workout scheduler interacting with the built-in calendar |
| The developed application must be running with a reasonable performance | Handle unwanted event occurrences and exceptions and dispose of the unneeded resources and memory |

*1.2 Requirement-Objective Table for the application Workout Journal*

# Design and implementation

This section has the goal to give a detailed representation of the project's features regarding the database model, the architectural design pattern and their components structure and functionality. The explanatory paragraph's logic will be supported by the implementation of graphs, diagrams and code fragments.

## Design

### Database Model

As mentioned earlier the developed application uses a local database to store the information. It is located in the app's local folder. Windows Phone apps use LINQ to SQL for all local database operations; LINQ to SQL is used to define the database schema, select data, and save changes to the underlying database file residing in the local folder.[[18]](#footnote-18) This approach has the following characteristics:

 A local database runs in the Windows Phone app’s process. Unlike a client-server database such as Microsoft SQL Server, it does not run continuously as a background service.

 A local database can be accessed only by the corresponding Windows Phone app. Because the database file resides in the local folder, no other apps can access that data.

 A local database can be accessed only with LINQ to SQL; Transact-SQL is not supported

#### LINQ to SQL runtime

LINQ to SQL provides object-relational mapping capabilities that enable the managed app to use Language Integrated Query (LINQ) to communicate with a relational database (that only “speaks” Transact-SQL). LINQ to SQL maps the object model to a relational database. When the app runs, LINQ to SQL translates language-integrated queries into Transact-SQL and then sends the queries to the database for execution. When the database returns the results, LINQ to SQL translates the results back to objects that are manageable in the specific programming language.

#### Architectural Overview

To store and retrieve data in a local database, a Windows Phone app uses LINQ to SQL. LINQ to SQL provides an object-oriented approach to working with data and comprises an object model and a runtime. The LINQ to SQL object model is made up primarily by the System.Data.Linq.DataContext object, which acts as a proxy for the local database. The LINQ to SQL runtime is responsible for bridging the world of objects (the DataContext object) with the world of data (the local database). This relationship is summarized in the following image.



#### Data context

The data context is a proxy, an object that represents the database. A data context contains Table objects, each of which represents a table in the database. Each Table object is made up of entities that correspond to rows of data in a database. The attributes on each entity determine the database table structure and define the mapping between the object model of the data and the schema of the database.

For each entity, mapping details are specified by using LINQ to SQL mapping attributes. These attributes define database-specific features such as tables, columns, primary keys, and indexes.

The following code shows the structure of the common DataContext class:

public class ObjectDataContext : DataContext

{

// Pass the connection string to the base class.

public ObjectDataContext(string connectionString) : base(connectionString)

{

}

// Specify tables for items.

public Table<Exercise> Exercises;

public Table<Workout> Workouts;

public Table<HealthyFood> HealthyFoods;

public Table<AvoidableFood> AvoidableFoods;

}

The following code snippet is an example for an entity class:  
[Table(Name = "Workouts")]

public class Workout : PropertyChangedBase, ITable

{

private int \_id;

[Column(IsPrimaryKey = true, IsDbGenerated = true, DbType = "Int NOT NULL IDENTITY", CanBeNull = false, AutoSync = AutoSync.OnInsert)]

public int Id

{

get

{

return \_id;

}

set

{

if (\_id != value)

{

NotifyPropertyChanging("Id");

\_id = value;

NotifyPropertyChanged("Id");

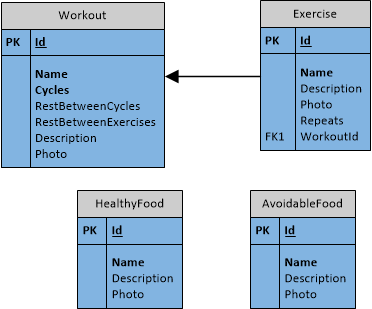
}

}

}

### Database diagram

The presented project uses the following database structure:

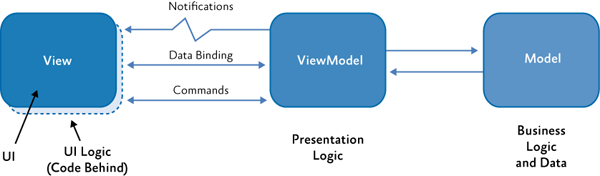


*1.3 Database diagram for the application Workout Journal*

### Model-View-ViewModel design pattern

As mentioned in the introduction the developed project follows the MVVM design pattern. This approach helps to separate the business and presentation logic from the user interface and also improves code re-use opportunities and makes the application much easier to test, maintain and evolve.

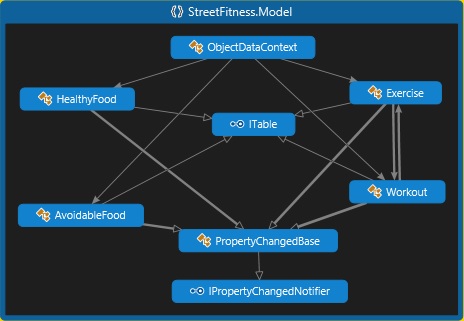
The pattern separates the project into three main sections: the view, which encapsulates the UI and UI logic; the view model, which encapsulates presentation logic and state; and the model, which encapsulates the application's business logic and data.

The following illustration shows the three MVVM classes and their interaction[[19]](#footnote-19):  


*1.4 MVVM working principle*

In the next subsections we will examine the structure and relationships of the main project's components using dependency graphs as representation. The visualization will be followed by a short description of the general role of the given objects.

#### Model



*1.5 Workout Journal's Model dependency graph*

##### Table objects

HealthyFood, AvoidableFood, Exercise and Workout represent the main data table objects. Each of these entity classes defines private fields, public properties, database columns and dependencies.

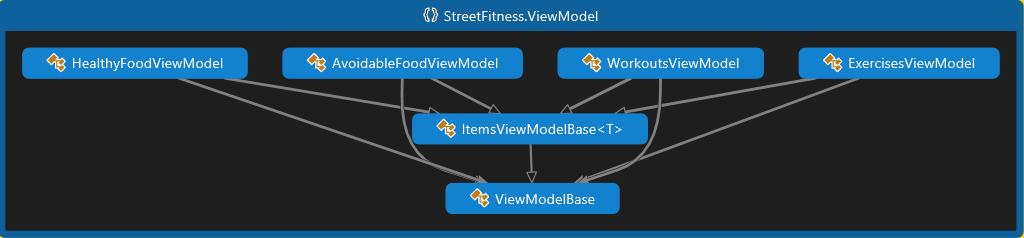
##### Interfaces

The entity classes implement three basic interfaces. ITable is used later in the ViewModel to define a specific table entity as a generic property in order to indicate the needed entity. The PropertyChagedBase includes two interfaces - INotifyPropertyChanged and INotifyPropertyChanging. They raise corresponding events when a property is about to change or is being changed, so that the changes could be reflected in the database.

##### ObjectDataContext

The table entities in the project share a common DataContext class : the ObjectDataContext. its purpose is to pass the connection string to the base class and to specify tables for the particular types in the underlying database.

#### ViewModel



*1.6 Workout Journal's ViewModel dependency graph*

##### ViewModelBase

The ViewModelBase abstract class implements the INotifyPropertyChanged interface and is inherited from all other classes in the ViewModel. It serves to acquire the DataContext from the Model. Further it defines a method submitting changes to the database. The class also defines two virtual methods - LoadData() and OnPropertyChanged(). The LoadData() method is overridden in each entity ViewModel class to get the items collection. The OnPropertyChanged() method is used in the derived ItemsViewModeBase class to indicate a change in a property from the items collection and to inform the system for that change.

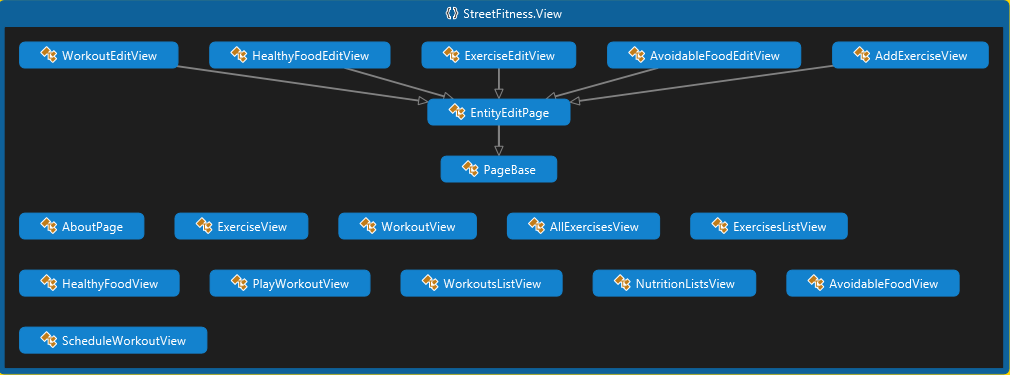
##### ItemsViewModelBase<T>

The ItemsViewModelBase<T> is also an abstract class, which is inherited from the entity ViewModels. Its purpose is to get the appropriate DataContext from the ViewModelBase class and to define an observable collection from the particular type. The class defines getters and setters for the items in the collection and specifies virtual methods for adding, removing or acquiring a specific item.

##### Entity ViewModels

Finally the Model namespace defines four entity ViewModel classes for the specific tables - HealthyFoodViewModel, AvoidableFoodViewModel, WorkoutsViewModel and ExercisesViewModel. Each of these classes overrides the virtual methods from the inherited abstract classes and defines a specific behavior for them.

#### View



*1.7 Workout Journal's View dependency graph*

##### View pages

The View namespace encompasses all the UI[[20]](#footnote-20) and UI logic. All elements from this section except for EntityEditPage and PageBase represent a page, which is visible and accessible for the user. Each element consists of two parts - a \*.xaml[[21]](#footnote-21) page and a code-behind file.

The XAML page is responsible for the creation of visible UI elements in the declarative XAML markup, such as buttons, labels and textboxes. In turn the code-behind page, in our case - a \*.xaml.cs file, since the used programming language is C#, separates the UI definition from the run-time logic. The connection between these two files is realized with partial class definitions.[[22]](#footnote-22)

Further each markup from the XAML page could be subscribed to a specific event, which respectively will be handled in the code-behind class. Another option is to set a data binding to the markup element and to define a DataContext for the given element. In this way the XAML directly represents the instantiation of objects in a specific set of backing types defined in assemblies.

##### PageBase

Each XAML page is normally derived from the PhoneApplicationPage, which is a predefined class in the Controls assembly from Microsoft. The same applies for the PageBase. It is inherited from all the Edit pages in the project and its purpose is to make sure that a certain entity is edited correctly. Basically it is used by updating or adding an element.

The PageBase defines two virtual methods - StoreState() and RestoreState(), which will be overridden in the derived classes. The class also overrides two methods - OnNavigatedTo() and OnNavigatedFrom(), where it defines under what condition the virtual methods must be called. The approach uses the PhoneApplicationPage.State[[23]](#footnote-23) property, which is a dictonary of type IDictionary<string, Object> and allows saving transient state data on the page.

This implementation has the following logic - if we are navigated to an edit page, then we are using the RestoreState() method, which possibly acquires the object's information, when we are trying to update an entity. Alternatively if we are navigated from an edit page and the navigation mode is not a back mode, then the StoreState() method is invoked, in order to preserve the edited information. The whole edit process logic is based only partly on the PageBase class. It's implementation is also supported by the next class - the EntityEditPage class.

##### EntityEditPage

The EntityEditPage class is a child of the PageBase class. It's task is to support further the edit process. Again it overrides the OnNavigatedTo() and OnNavigatedFrom() methods, however the object is to define whether a database rollback is required.

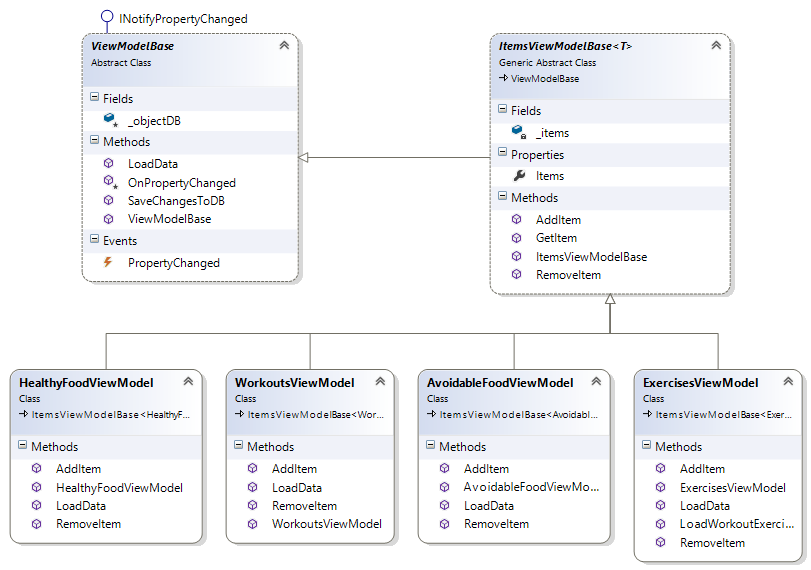
Briefly the class makes sure that the RollBack() method is called, when a database rollback is needed, i.e. when an entity has been changed and the change must be reflected in the database as well. The RollBack() method itself is defined as a helper method in the Utils namespace and more detailed explanation will follow in the Implementation section.

## Implementation

This part of the documentation will present to the reader the result of the development process - a summarizing notation of the implemented architecture and helper classes.

### ViewModel namespace

The following class diagram shows the developed result of the ViewModel namespace.

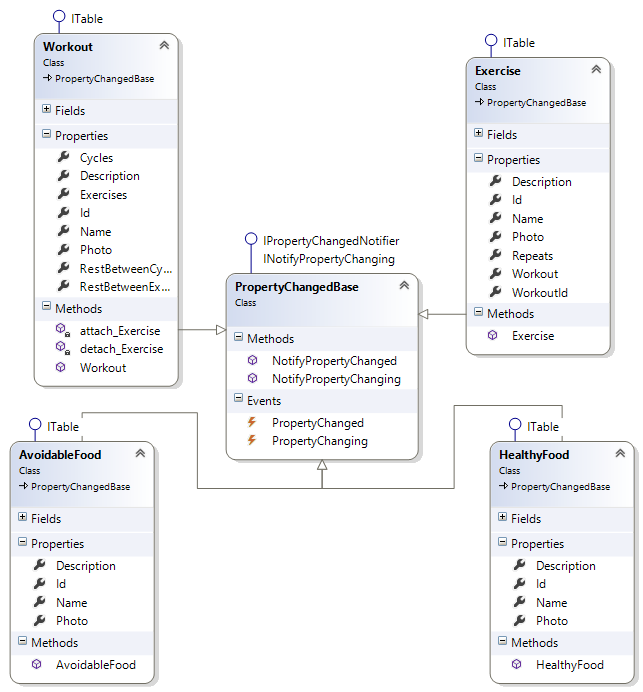


*1.8 Workout Journal's ViewModel class diagram*

To sum up - the first layer of this namespace gets the data context and defines the methods for interacting with the database. The second layer serves as a constructor for a collection of items of a certain type and specifies the needed methods for managing this collection. The final third layer contains a ViewModel class for each table object. Each class there sets a distinctive behavior of its methods according to the requested table object operations.

### Model namespace

The next class diagram is a representation of the developed model architecture blueprint



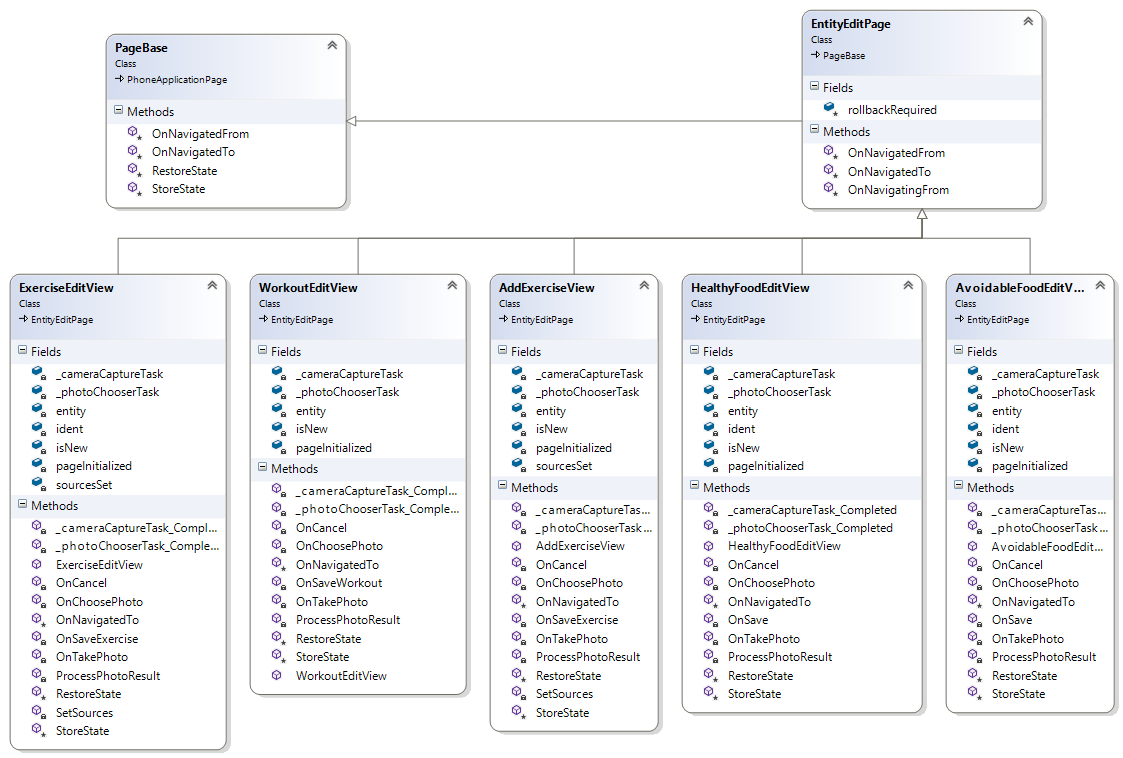
*1.9 Workout Journal's Model class diagram*

### View namespace

Since the application offers a variety of view classes, in this section will be presented only those of them, for which a class diagram could provide a better understanding about their working principle.

#### Editing Views

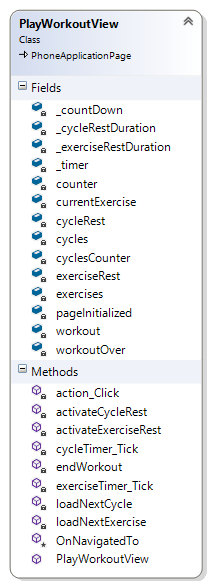
The class diagram below summarizes all view classes responsible for adding or editing information about object entity of a specific type



*1.10 Workout Journal's editing View classes*

#### Play workout class

One of the most important and interesting class of this namespace regarding functionality is the PlayWorkoutView. The role of this class is to implement the interactive approach to the application. In order to achieve this it uses a DispatcherTimer from the Threading namespace. Once a certain workout instance and its belonging exercise collection are loaded into the code-behind class, the class awaits and handles the user interaction by loading consecutive exercises and displaying information about the current progress. The implemented timer is used as a countdown to indicate the individual rest times. This is what the class's structure looks like:



*1.11 Workout Journal's PlayWorkoutView class*

#### Settings View

In the code-behind file of this view is located the logic, which allows the user to disable the locking feature of the device regarding the application's scope. This technique could come as an advantage in the interactive player mode, when some rest times could potentially require more time without interaction from the user side. In such a scenario the keep the screen alive option wouldn't require unlocking the screen in order to proceed with the workout. Here are the two events, responsible for managing the Idle Detection Mode:

private void toggleSwitch\_Checked(object sender, RoutedEventArgs e)

{

PhoneApplicationService.Current.UserIdleDetectionMode = IdleDetectionMode.Disabled;

Toggled = true;

}

private void toggleSwitch\_Unchecked(object sender, RoutedEventArgs e)

{

PhoneApplicationService.Current.UserIdleDetectionMode = IdleDetectionMode.Enabled;

Toggled = false;

}

#### Workout Scheduler

The application Workout Journal offers the user the ability to add an appointment about a particular workout routine to the tasks in the phone's built-in calendar. In this way the user could easily organize his individual training program and can become reminders for the upcoming exercises. Once the user fills the given fields in the scheduler view of the application, the built-in calendar is called to finalize the details about the appointment and to save it. Here is the event responsible for handling the requested operation:

private void sheduleWorkout\_Click(object sender, RoutedEventArgs e)

{

//define a task element

SaveAppointmentTask workoutSchedule = new SaveAppointmentTask();

//populate appointment's fields

workoutSchedule.StartTime = startTimePicker.Value;

workoutSchedule.EndTime = endTimePicker.Value;

selectedWorkout = (Workout)listWorkouts.SelectedItem;

workoutSchedule.Subject = selectedWorkout.Name.ToString();

workoutSchedule.Location = locationBox.Text;

workoutSchedule.Details = detailsBox.Text;

//call the built-in calendar to proceed saving the appointment

workoutSchedule.Show();

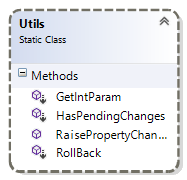
}

### Utils namespace

The developed project includes an additional namespace - the Utils namespace. It consists of helper classes, converters and methods. In this section will be presented their basic working logic.

#### Utils static class

The Utils static class has the following structure:



*1.12 Workout journal's Utils class*

##### GetIntParam()

This method is used from the navigation system to acquire an integer value, which indicates an Id of a table object. This approach is used when a specific entity must be loaded on a page and simultaneously the object is taken from a previous page according to the current context. This is how it works:  
public static int? GetIntParam(this NavigationContext context, string paramKey)

{

if (context.QueryString != null && context.QueryString.Keys.Contains(paramKey))

{

return int.Parse(context.QueryString[paramKey]);

}

return null;

}

##### RaisePropertyChanged()

The task of this method is to handle an event that a property has been changed and to notify the corresponding class for the change, initializing a new instance of the PropertyChangedEventArgs stating the property, which has been changed.

public static void RaisePropertyChanged(string propertyName, object sender, PropertyChangedEventHandler PropertyChanged)

{

Deployment.Current.Dispatcher.BeginInvoke(() =>

{

PropertyChangedEventHandler handler = PropertyChanged;

if (null != handler)

handler(sender, new PropertyChangedEventArgs(propertyName));

});

}

##### HasPendingChanges()

This method is used from an edit page to indicate whether a database rollback is required. The method returns a boolean variable dependent on the container of the current set, holding the changes. If any changes are being registered, then the method returns a true value, which means that a rollback is required.

public static bool HasPendingChanges(this DataContext dataContext)

{

ChangeSet pendingChanges = dataContext.GetChangeSet();

if (pendingChanges.Deletes.Count > 0 || pendingChanges.Inserts.Count > 0 ||

pendingChanges.Updates.Count > 0)

{

return true;

}

return false;

}

##### RollBack()

If the system comes to a state, when a database rollback is required, then this method is being invoked. The method acquires the pending changes from the current data context and depending on their type calls the necessary table methods to reflect the changes:

public static void RollBack(this DataContext dataContext)

{

ChangeSet pendingChanges = dataContext.GetChangeSet();

foreach (var insertion in pendingChanges.Inserts)

{

dataContext.GetTable(insertion.GetType()).InsertOnSubmit(insertion);

}

foreach (var deletion in pendingChanges.Deletes)

{

dataContext.GetTable(deletion.GetType()).DeleteOnSubmit(deletion);

}

foreach (var update in pendingChanges.Updates)

{

dataContext.Refresh(RefreshMode.OverwriteCurrentValues, update);

IPropertyChangedNotifier updateNotify = update as IPropertyChangedNotifier;

foreach (PropertyInfo propertyInfo in update.GetType().GetProperties())

{

updateNotify.NotifyPropertyChanged(propertyInfo.Name);

}

}

}

#### UriHelper static class

In order to achieve better and easier navigation between the separate pages the project implements a static class, where all the relative paths of the pages are stored as Uri fields. Further the class includes several methods, which allow the addition of extra elements in a Uri object. In this way we can pass an Id of an object to the navigation context and later retrieve the transferred Id with the GetIntParam() method. Here are two examples for field and method of this class, as well as its diagram.

//field  
public readonly static Uri WorkoutListViewUri = new Uri("/View/WorkoutsListView.xaml", UriKind.Relative);

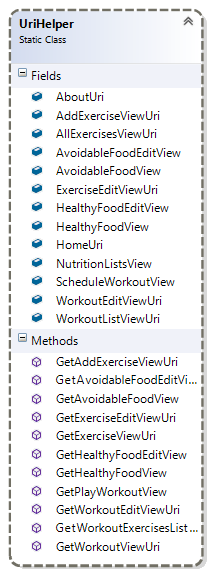
//method  
public static Uri GetWorkoutViewUri(Workout workout)

{

Console.WriteLine("uri helper id : {0}", workout.Id);

return new Uri(string.Format("/View/WorkoutView.xaml?id={0}", workout.Id), UriKind.Relative);

}



*1.13 Workout Journal's UriHelper static class*

#### BinaryToBitmapImageConverter class

This class implements the IValueConverter interface. The class is used to associate a value converter with a binding. It implements the Convert() method of the interface, so that a given binary value can be converted to a bitmap image, which will be displayed with a data binding in the XAML page. This is needed, since the images are kept in the database table objects as binary fields. Here is how the method works:

public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture)

{

if (value == null)

{

return null;

}

var memoryStream = new MemoryStream((value as Binary).ToArray());

var bitmap = new BitmapImage();

bitmap.SetSource(memoryStream);

return bitmap;

}

### App class

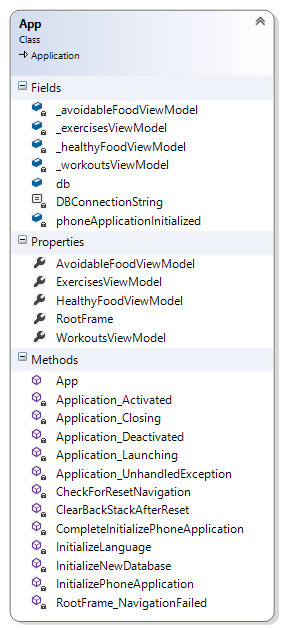
The final paragraph of the implementation section represents the App class of the project, since this is the class, which encapsulates a Windows Phone application[[24]](#footnote-24). It provides the following services:

* Application Entry Point
* Application Lifetime
* Application Management
* Application-Scoped Resources
* Unhandled Exception Detection

The class is entry point in a Windows Phone application and derives from Application. When the application starts, metadata in the application package is used to instantiate the application class. At this point, the application's lifetime starts.

For the current project the application class defines a database connection string, static database data context and static view models for each of the table objects. When the application launches the connection string is passed to the data context and afterwards the view models are instanced according to the data context. At this stage the view models call the load data methods to acquire the items collections.

Prior to the instance of the view models there is a check whether the specified database exists. Is it the case that there is no such database yet, the database is created for the current context. Since this scenario represents the first launch of the application, after the creation of the database, the InitializeNewDatabase() method is called. Its purpose is to insert initial data into the database. After this the SubmitChanges() method is called to save the changes to the database. When the application closes the SubmitChanges() method is called again to reinsure if any changes are not being reflected and finally the Dispose() method is called to free the allocated resources. The following diagram gives a better notion on the application class.



*1.14 Workout Journal's App class diagram*

# Validation and verification

The purpose of this section is to check up to what point the developed application meets the stated requirements and to see how the application behaves in general. In order to fulfill the task and to give an overall conclusion about the workflow and achievements of the project a number of tests and simulations have been conducted. The first subsection of this part summarizes the used technologies and their specifications. The second subsection follows with a comparison between the applications behavior in different environments, accompanied with data tables.

## Simulation

### Emulators

For the main part of the development process the project has been tested on the built-in Windows Phone Emulators. Since the development was initialized to target the Windows Phone 8.0 platform, but was later upgraded, in order to be compatible with the new Windows 8.1 update, the list of the introduced emulators include virtual devices for both Windows Phone 8.0 and Windows Phone 8.1 OS:

* Emulator WVGA 512MB
* Emulator WVGA
* Emulator WXGA
* Emulator 720p
* Emulator 8.1 WVGA 4 inch 512MB
* Emulator 8.1 WVGA 4 inch
* Emulator 8.1 WXGA 4.7 inch
* Emulator 8.1 720p 4.7 inch
* Emulator 8.1 1080p 5.5 inch
* Emulator 8.1 1080p 6 inch

Such an approach offers the opportunity to check the application's behavior in different environments. However the project's goal is to target a wider range of devices. Therefore the most tests and simulations were conducted for the memory-constrained emulators with lower screen resolution - respectively those with operational memory of 512MB and WVGA resolution of 800x480.

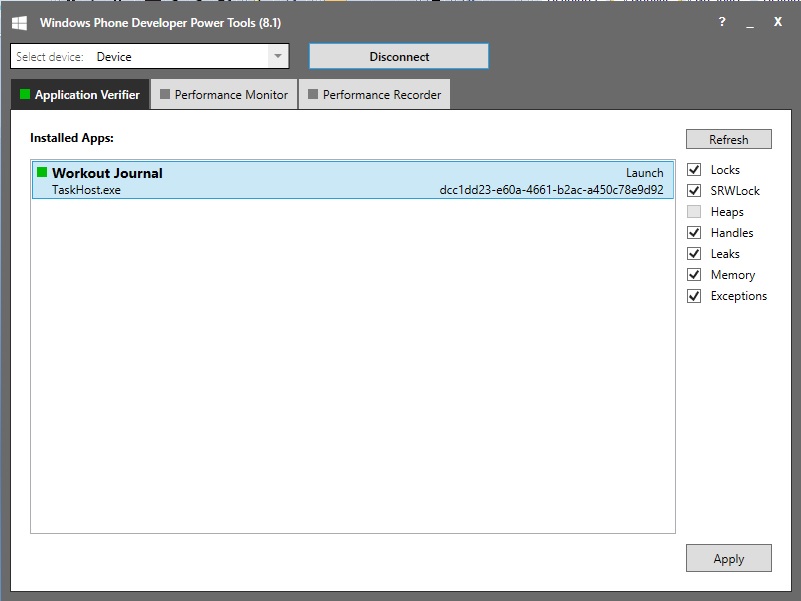
### Device

Apart from the mentioned emulators the application had been tested with a Nokia Lumia 720 Windows Phone 8.0 developer registered device, which was later upgraded to Windows Phone 8.1. The device has the following specifications:

* Display size: 4.3 ''
* Display resolution: WVGA (800 x 480)
* Battery capacity: 2000 mAh
* Processor name: Qualcomm Snapdragon™ S4
* Processor type: Dual-core 1 GHz
* RAM: 512 MB
* Mass memory: 8 GB
* Expandable memory card type: MicroSD

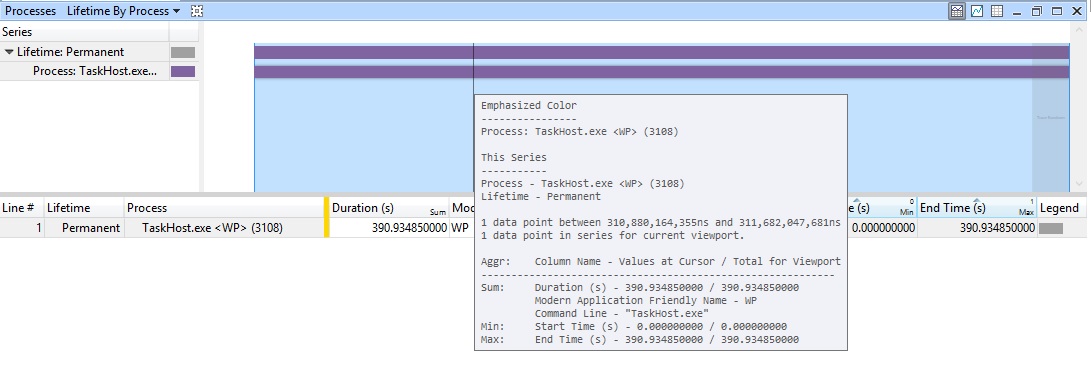
## Test results

The application's real time performance had been tested with the Windows Phone Developer Power Tools[[25]](#footnote-25). These are three powerful testing and debugging tools for app developers packaged into a single user interface. The purpose is to monitor your app's responsiveness and resource consumption and to debug its crashes. The presented data in this section had been collected with the Performance Recorder. The visualized screenshots of the collected results monitor the system behavior according two separate tests. Both tests were conducted in a period of about 5 minutes working with the developed application. Respectively one of the tests had been performed with the mentioned previously Nokia device, while the other represents an identical behavior, using the virtual emulator 8.1 720p 4.7 inch that fits best the device's specs.  
The tests emphasize on the process TaskHost.exe, which is the system's representation of the developed application.



*1.15 Workout Journal Application Verifier and process representation*The collected results focus on the following four aspects:

### Duration



*1.16 Emulator Test Duration*



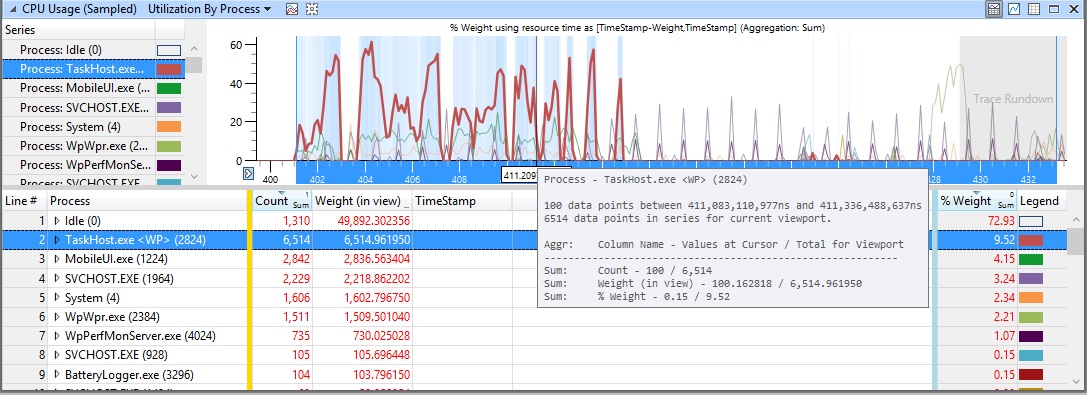
*1.17 Device Test Duration*

As we can see the process was running permanently in both cases. It's duration ranges from 390 to 433 seconds. In this period the application had been tested for its basic functionality - CRUD operations, I/O[[26]](#footnote-26) operations and the appointment scheduler. At both stages no errors were registered and the system was running without substantial delays.

### CPU Usage[[27]](#footnote-27)



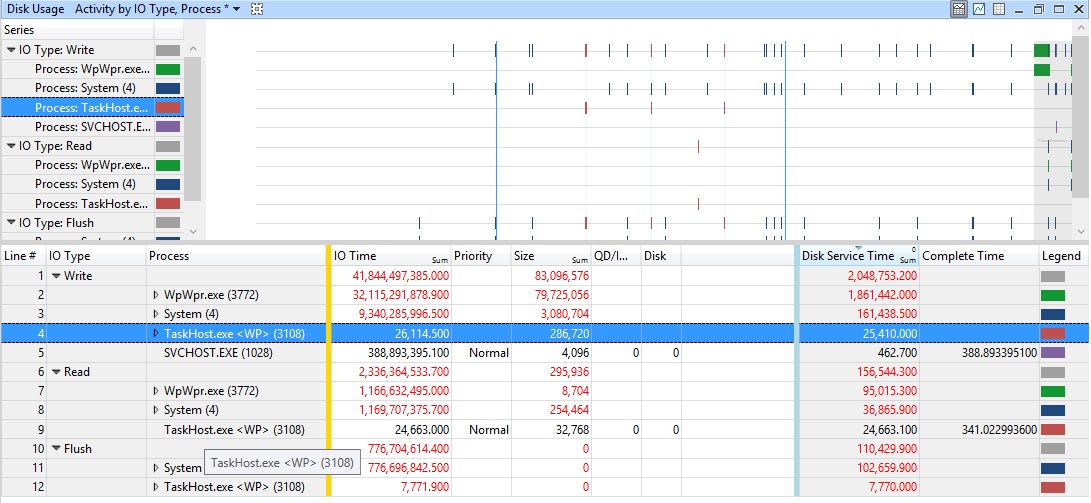
*1.18 Emulator Test CPU Usage*



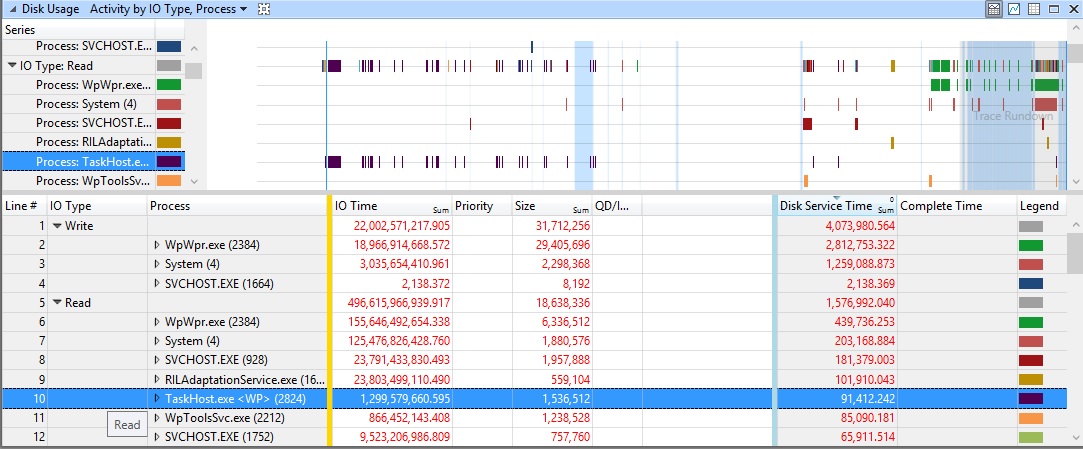
*1.19 Device Test CPU Usage*

The depicted results from both line charts describe the percentage of CPU Usage compared with the other running processes. Although the charts show reasonable differences at first sight, the results are almost similar after some detailed observation. The key factor in this context is the % Weight Sum column in the table below. This indicator expresses Weight as a percentage of total CPU time that is spent over the currently visible time range. In both tests this factor doesn't rise more than 10%. If we also pay attention to the weight of the MobileUI.exe, which is tightly connected with the representation of the observed process, in both cases the sum of the weight of the both factors is around 13%. Thus the fact that the emulator represents isolated environment, while the real device had many other applications and tasks running on the background explain further the reason for the observed changes.

### Disk Usage



*1.20 Emulator Test Disk Usage*



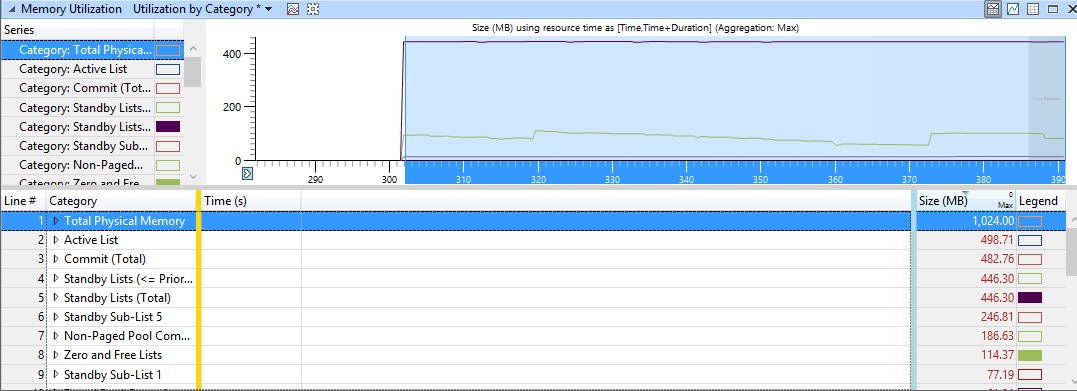
*1.21 Device Test Disk Usage*

This section represents actually the I/O activity of the process. The key factors from the screenshots, which determine the interpretation of the disk usage, are the following three table columns: IO Time, Size and Disk Service Time. The IO Time indicates the time from initiation to completion, including queuing time. Size is the size of the manipulated data. Disk Service Time is the estimated time for the operation based upon returning IRP timestamps. Since the performed operations for both tests differ from each other, it is difficult to make comparisons between the environments. Therefore we will follow the approach to handle each one individually.

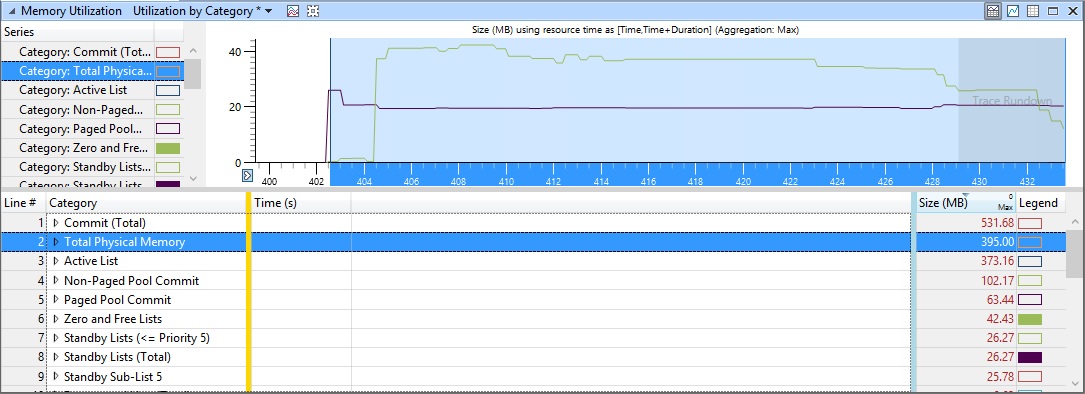
The emulator's test shows that both read and write operations take only small amount of size and I/O time compared with the other running processes. As a result the disk service time is also minimal.

The behavior of the application is similar in the device test, when we compare the bound data with this for the other processes. In this case the read operations are considerably more than they were by the emulator. However the IO and service time are rather low and what is very important to note - there are parallel processes, which have less or identical size, but at the same time require much more I/O time and disk service time to process this data.

### Memory Utilization



*1.22 Emulator Test Memory Utilization*



*1.23 Device Test Memory Utilization*

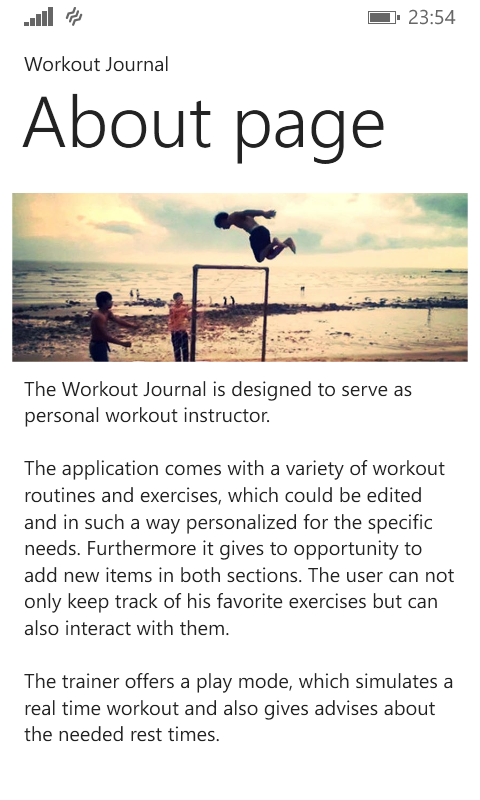
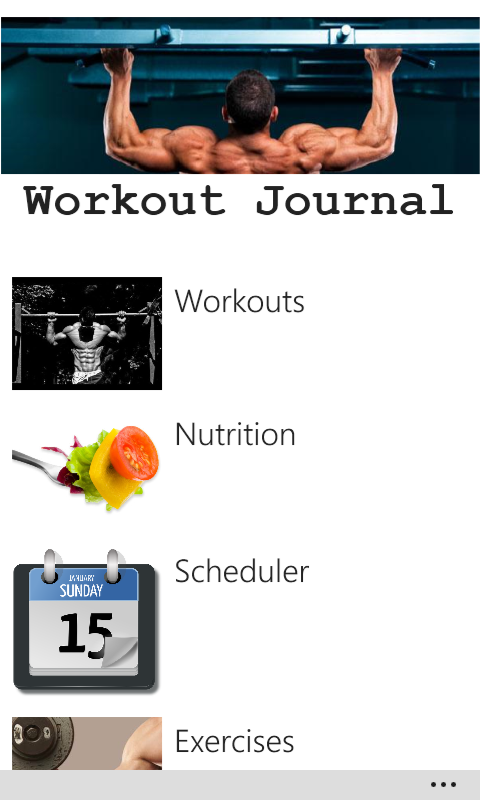
The result of this last section has the goal to compare the memory utilization in both environments. Substantial factor here is the difference in total physical memory. The virtual emulator represents a device with 1024MB sized memory. The real device has twice less memory - 512MB. This is also another reason for some of the discrepancies stated in the previous chapters. Although this factor causes some irregularities in tests, the amount of zero and free lists is proportional to the size of the overall memory in both situations and as a result we can conclude that the tested application shows similar behavior patterns among the different environments.

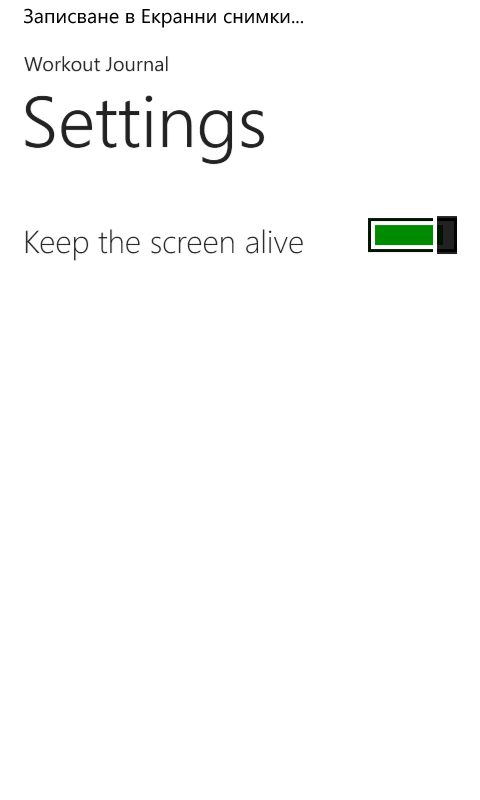
To sum up - the developed application had been tested on a number of different emulators and a physical device. The goal was to draw conclusions about its overall behavior. During this validation process no errors or crashes were indicated and the system was functioning according its standard behavior. This statement is supported by the collected test results. In summary, the project meets the staged requirements and is ready for use. Consequently the next chapter has this purpose.

# Demonstration

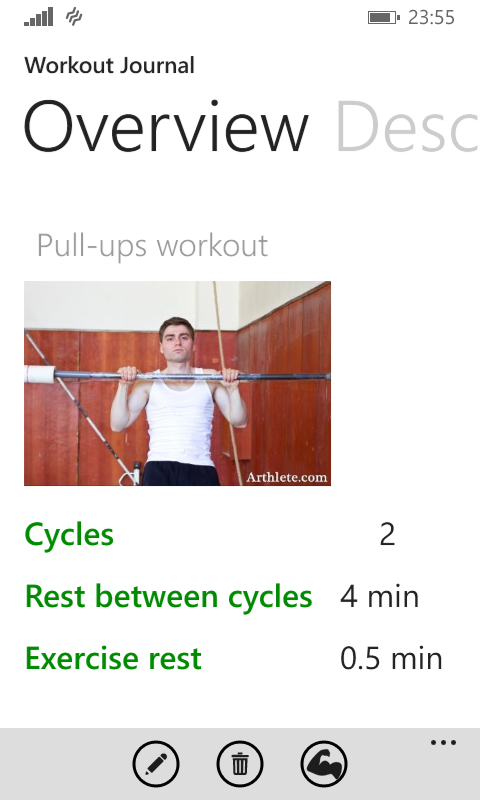
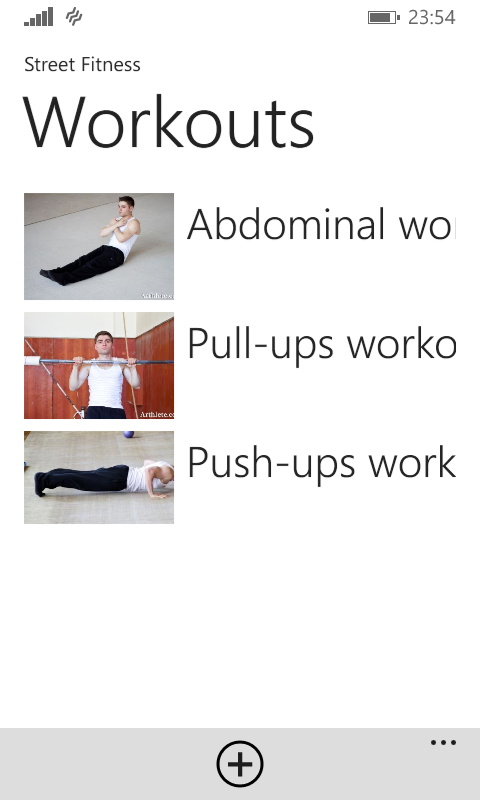
This section demonstrates the application's functionalities including a set of screenshots

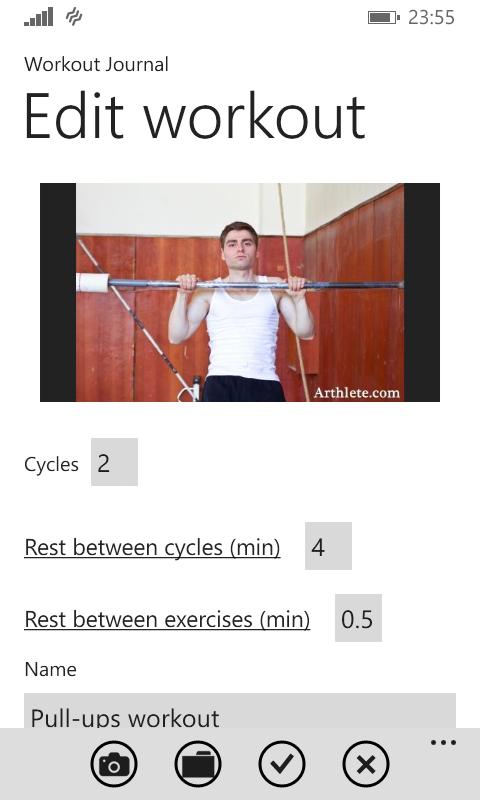
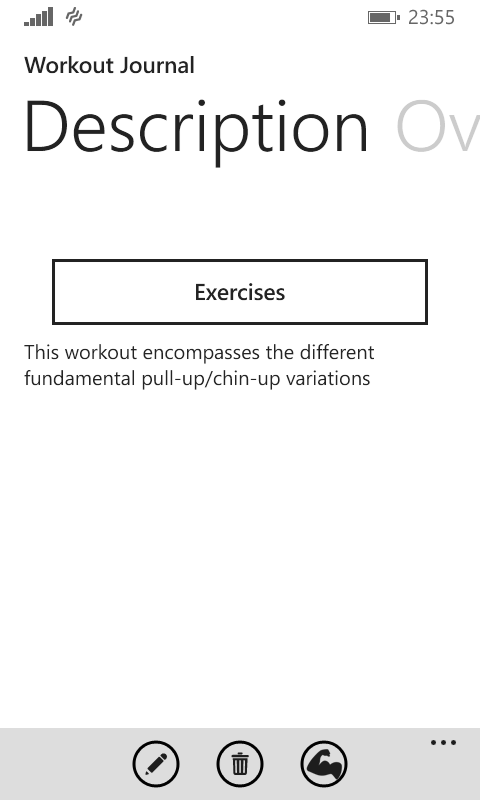
## Main view



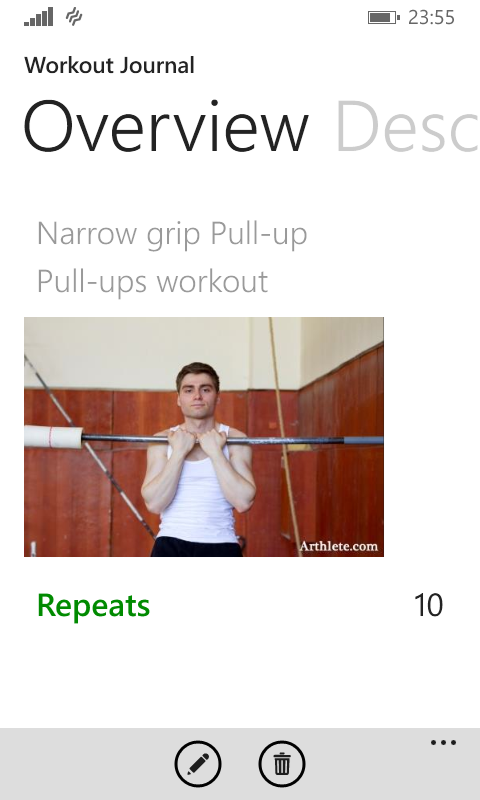
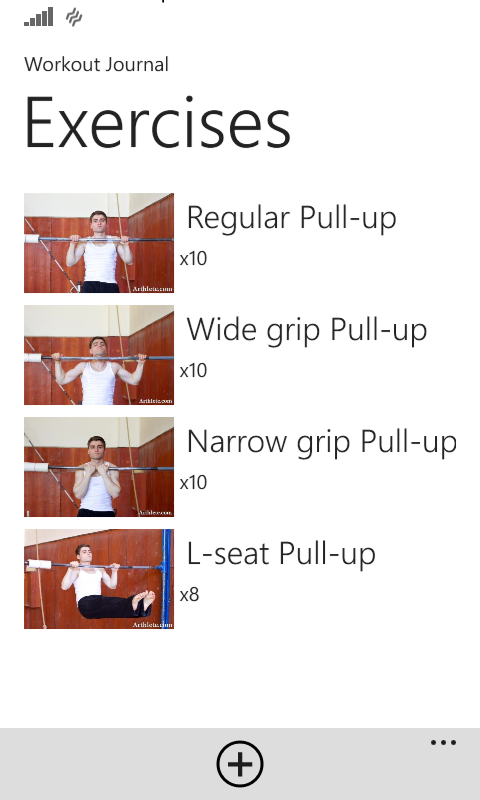


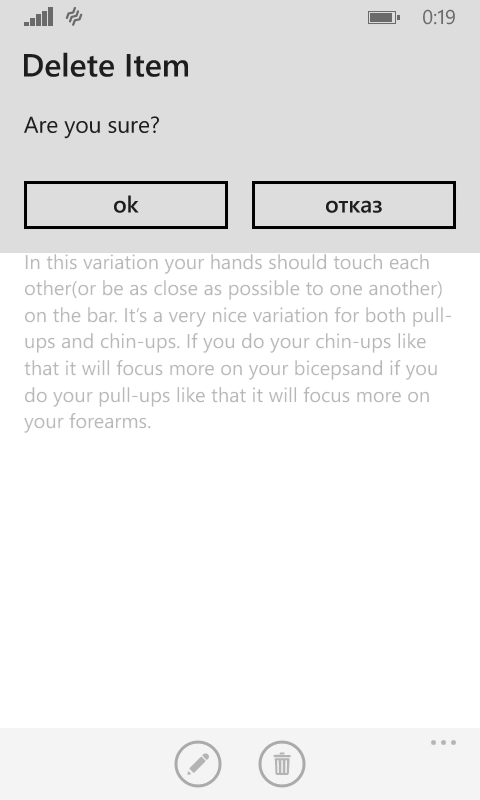
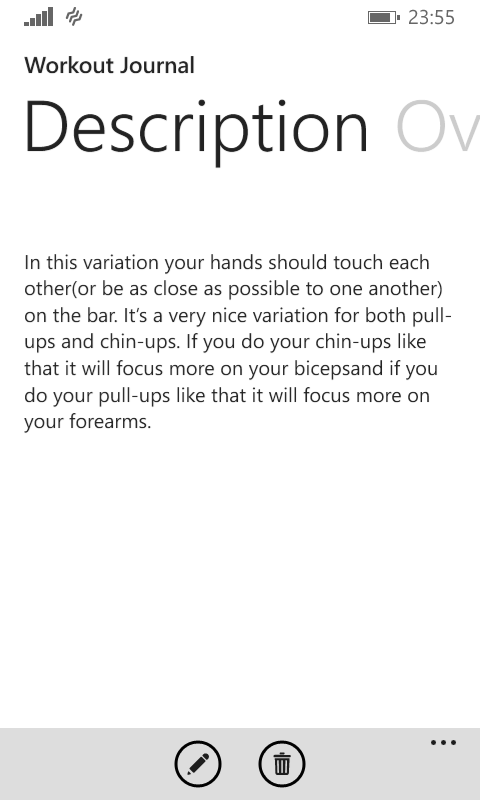
## Workouts view



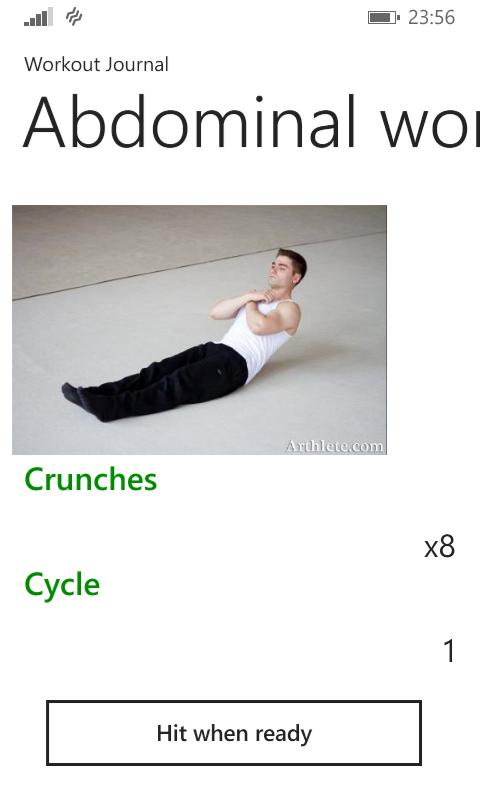


## Workout exercise view

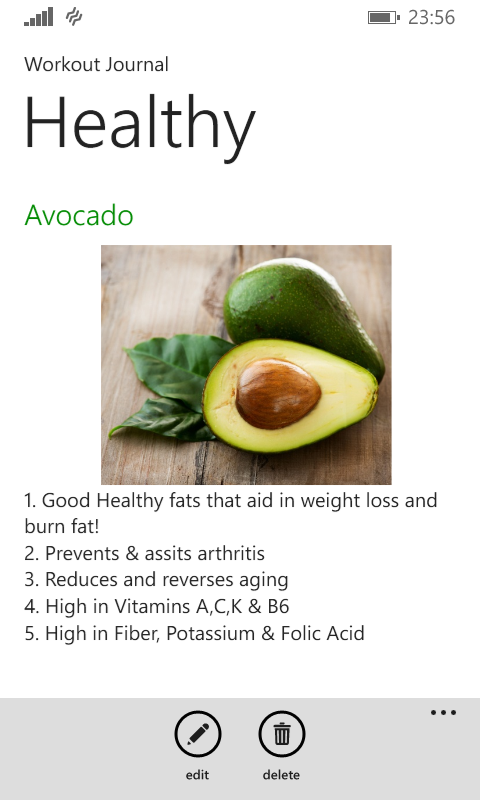


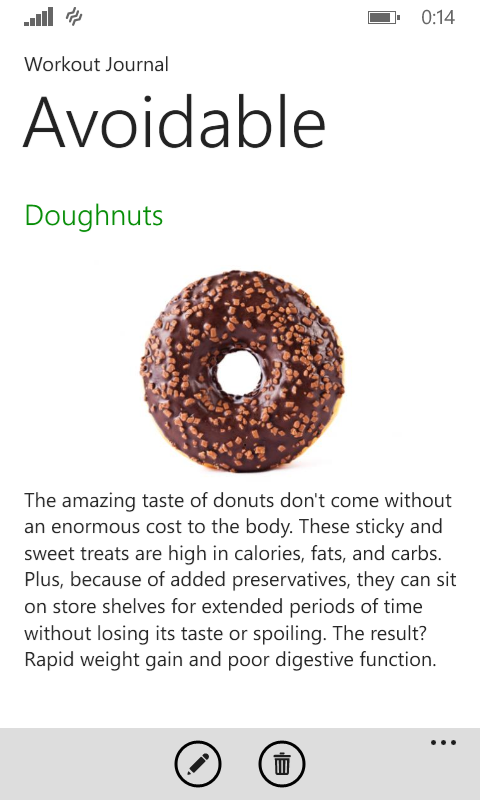
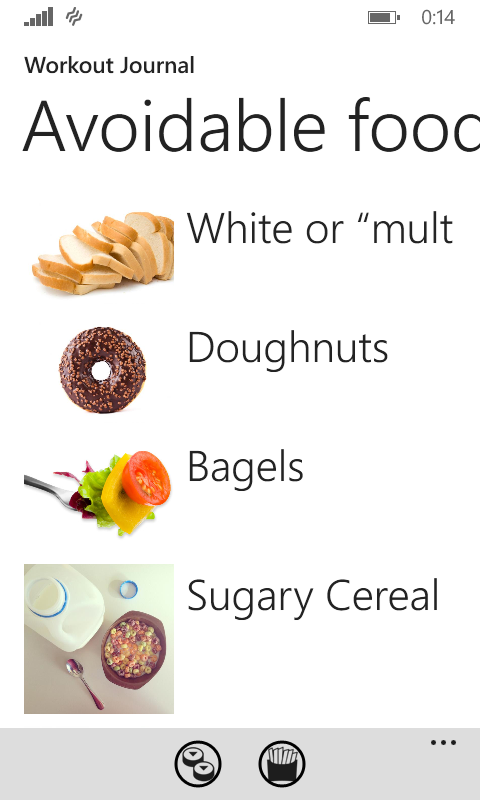


## Play view

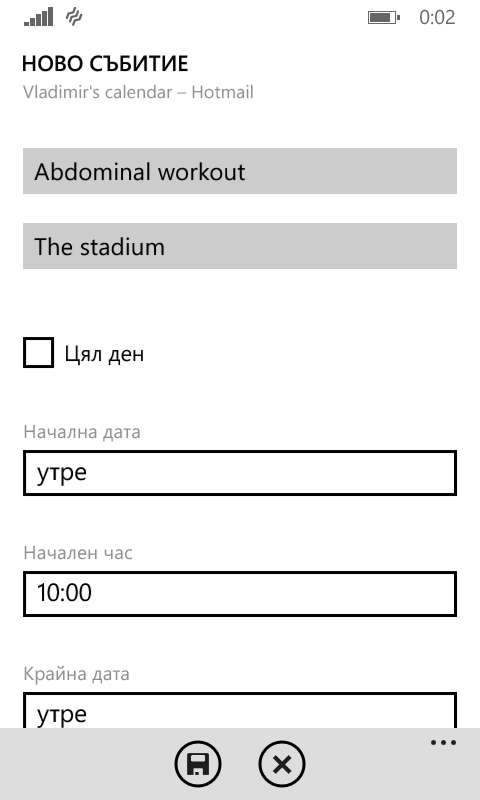
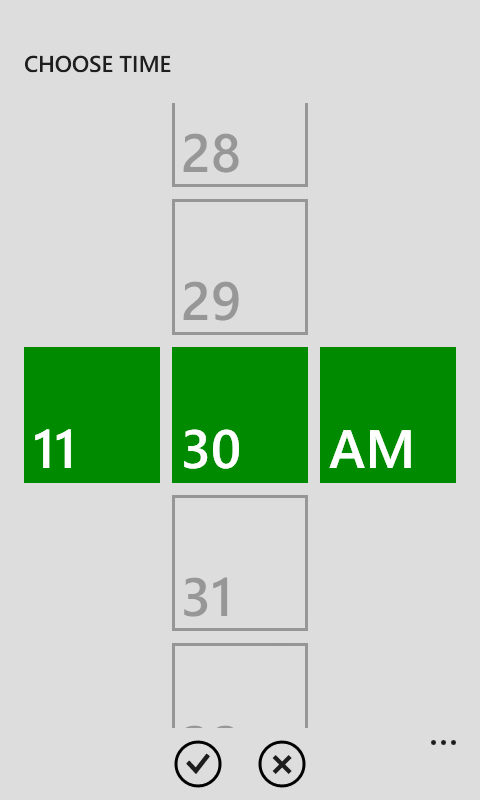
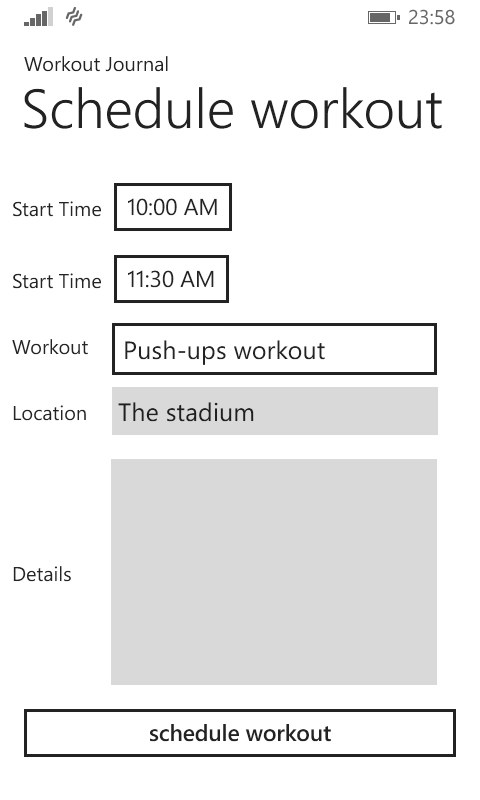


## Nutrition view





## Workout scheduler



# Summary

This section has the goal to analyze the whole development process and to draw conclusions about the collected results.

To recapitulate - the assigned task is to create a native mobile application, intended as a personal fitness-trainer and guide. The application must offer different workouts and exercises and in the same time the ability to highly personalize the individual training experience by editing them. Among the main requirements to the application are the additional organizer and player opportunities and the supportive nutrition advices. Nevertheless the most important factor about this project is to develop a solution, which is easy and convenient to work with and achieving reasonable performance. The derived objectives are summarized in section 2 on pages 7 and 8.

The given task has led to an architecture design, which is based on a relational database located in the isostore of the device. The chosen platform and its features caused the need to use an object-relational mapping technique. The details about this approach are to be found in section 3, starting on page 9.

In order to follow the best software practices regarding the selected platform, the project's design pattern follows the MVVM pattern. The implemented dependency graphs in the design chapter, pages 13 to 15, give a notion about the basic separation of the project into three main namespaces.

The real implementation of the project follows the chosen design pattern and includes some additional code fragments, such as the Utils namespace, containing helper classes and methods. More general description about the implementation is to be found in the second part of the third chapter, supported by class diagrams.

After the implementation process there were a number of tests that had been conducted in different environments to check the outcome behavior. This verification and validation process is presented in the fourth chapter of the project, starting on page 28. The conclusions, which can be made owing to the test results are that the developed project fulfills the general requirements. The application shows lower CPU usage. In addition the I/O operations don't take much time for processing and require less system resources. Taking into consideration the fact that the application had been tested on a real device as well and that it showed no breaks or deadlocks during the testing process, it is possible to summarize that the stated performance expectations had been met.

The last chapter represents the developed UI .It has the purpose to demonstrate that the application offers a simple, easy-to-use and user-friendly interface. The implementation of some of the platform-offered controls contributes to the interactive layout objectives.

The represented original findings in the submitted work include:

* Requirements elicitation
* Use-case diagram and requirements-objective table
* Relational database structure
* Namespace dependency graphs creation
* Validation and verification test graphs and tables
* Interface design

In this summarizing chapter I would like to list also some generic approaches, examples and information, which I used to develop this project. Links to the sources will follow in the sources section.

* Workout and exercise advices, tutorials and images from Nikolay Kolev's blog arthlete under his personal consent and approval
* Nutrition tips, taken from articles about healthy lifestyle from internet
* Advices and examples from sample projects from the MSDN Code Gallery about structuring and implementing the project correctly and following the best software practices

In summary the developed application corresponds to the stated expectations. As it is functioning in absence of faults it could be committed for approval from Microsoft in order to be shared with other people on the market. However in this last section I would like to share my vision about the future perspectives regarding this particular project.

In terms of connectivity the application could be extended with a feature allowing users to share their personal experience with the software. With the implementation of a web server people could share workout routines, exercises or general advices on their nutrition program. As the application already offers the ability to modify each entity, this extension would further improve the opportunities for individual approach towards the personal training.

Another option in expanding the applications functional scope is to implement a voice guidance while practicing a particular workout routine. In this way people would take advantage of using training time more effectively. However this approach suggests either the development of a automated voice generator, which would be capable of focusing on only a limited set of commands, or to include the ability to record orders and guidance from the user side. Either way the key factor that has to be taken under consideration for this scenario is the strategy of preserving the recorded samples.

A final way to extend the application's functionality would be the addition of more interactive media elements. Since the main goal of the software is to serve as a guidance and instructor, it would be very convenient to add animations, video clips or animated photos. In this way the user can become more detailed overview on how to perform a particular exercise. Again one of the most important factors would be storing the data. A possible scenario could overcome this challenge by preserving information on a cloud server and to provide it via links, but in this way all the amount of internet traffic would increase substantially.

To conclude - the developed application offers core functionality regarding a highly extendible matter. The implemented features are consistent with the conditioned requirements and offer a sufficient opportunities to improve the individual training approach. Nevertheless a possible solution, which includes some of the extras mentioned in the closing section, would potentially achieve more success among the user community.

# Sources

Local database for Windows Phone 8, May 10; 2014  
http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202860%28v=vs.105%29.aspx

Implementing the MVVM Pattern  
http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx

Application class; May 11, 2014  
http://msdn.microsoft.com/en-us/library/windowsphone/develop/System.Windows.Application%28v=vs.105%29.aspx

L.Bugnion; Inheriting from custom class instead of Window; March 2, 2007  
http://geekswithblogs.net/lbugnion/archive/2007/03/02/107747.aspx

Test and troubleshoot apps with the Windows Phone Developer Power Tools  
http://msdn.microsoft.com/en-us/library/windows/apps/dn629255.aspx

System requirements for the emulator for Windows Phone 8; May 10, 2014  
http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626524(v=vs.105).aspx

FrameworkElement.DataContext Property; May 11, 2014  
http://msdn.microsoft.com/en-US/library/windowsphone/develop/system.windows.frameworkelement.datacontext%28v=vs.105%29.aspx

Data binding for Windows Phone 8; May 10, 2014  
http://msdn.microsoft.com/en-us/library/windowsphone/develop/cc278072%28v=vs.105%29.aspx

N. Kolev; Workout tutorials  
http://www.arthlete.com/category/tutorials/

The Health Benefits of Sport and Physical Activity  
http://www.sportanddev.org/en/learnmore/sport\_and\_health/the\_health\_benefits\_of\_sport\_and\_physical\_activity/

14 Foods to Kick Out of the Kitchen Forever; J. Beckman;   
http://www.mensfitness.com/nutrition/what-to-eat/14-foods-to-kick-out-of-the-kitchen-forever

The 8 Absolute Worst Foods You Can Pump Into Your Body; HUMANFITPROJECT  
http://www.mensfitness.com/nutrition/what-to-eat/healthy-resolution-the-8-absolute-worst-foods-you-can-pump-into-your-body/slide/3

10 Foods That'll Hit Your Newsfeed in 2014; T. Gagnon  
http://www.mensfitness.com/nutrition/10-foods-thatll-hit-your-newsfeed-in-2014/slide/7

9 Foods That Boost Metabolism Naturally; Health magazine  
http://www.health.com/health/gallery/0,,20553780,00.html

Local Database Sample; Windows platform; 5/4/2013  
http://code.msdn.microsoft.com/Local-Database-Sample-57b1614c

Windows Phone 7.1 Local Database - Updating the Schema; M. Kruszynski; 8/1/2011  
http://code.msdn.microsoft.com/Windows-Phone-71-Local-da2fdf80

Windows Phone 8 JumpStart: Contoso Cookbook Step 11, Page Transitions; A. Wigley; 12/20/2012  
http://code.msdn.microsoft.com/Windows-Phone-8-JumpStart-e3c411c7

XAML Overview (WPF)  
http://msdn.microsoft.com/en-us/library/ms752059%28v=vs.110%29.aspx

PhoneApplicationPage.State Property; May 11, 2014  
http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.controls.phoneapplicationpage.state%28v=vs.105%29.aspx

Hyper-V; Wikipedia;   
http://en.wikipedia.org/wiki/Hyper-V

Using the New Launchers Supported in Windows Phone 8; V. Patel; April 15, 2013  
http://www.developer.com/ws/using-the-new-launchers-supported-in-windows-phone-8.html

Value Converters; C. Moser; 2010-02-19  
http://wpftutorial.net/ValueConverters.html

# List of abbreviations

* MVVM - Model View ViewModel
* isostore - isolated storage
* ORM - Object-relational mapping
* LINQ to SQl - Language Integrated Query to Structured Query Language
* sdf - SQL Server Compact Edition Database File
* OS - operating system
* UML - Unified Modeling Language
* SDK - Software Development Kit
* DHCP - Dynamic Host Configuration Protocol
* Hyper-V - Windows Server Virtualization
* OOP - Object Oriented Programming
* CRUD - Create, Remove, Update, Delete
* UI- User Interface
* XAML - Extensible Application Markup Language
* I/O - Input/Output
* CPU - Central processing unit
* SLAT - Second Level Address Translation
* BIOS - Basic Input/Output System
* DEP - Data Execution Prevention

# Declaration of Authorship

Declaration of Authorship

Last Name:

Yankov

First Name:

Vladimir

I declare that the work presented here is, to the best of my knowledge and belief, orig-

inal and the result of my own investigations, except as acknowledged, and has not been

submitted, either in part or whole, for a degree at this or any other University.

Formulations and ideas taken from other sources are cited as such. This work has not

been published.

Location, Date

Signature

1. http://www.sportanddev.org/en/learnmore/sport\_and\_health/the\_health\_benefits\_of\_sport\_and\_physical\_activity/ [↑](#footnote-ref-1)
2. MVVM - Model View ViewModel [↑](#footnote-ref-2)
3. isostore - isolated storage [↑](#footnote-ref-3)
4. ORM - Object-relational mapping [↑](#footnote-ref-4)
5. LINQ to SQl - Language Integrated Query to Structured Query Language [↑](#footnote-ref-5)
6. sdf - SQL Server Compact Edition Database File [↑](#footnote-ref-6)
7. OS - operating system [↑](#footnote-ref-7)
8. UML - Unified Modeling Language [↑](#footnote-ref-8)
9. SDK - Software Development Kit [↑](#footnote-ref-9)
10. http://msdn.microsoft.com/en-us/library/windowsphone/develop/ff626524%28v=vs.105%29.aspx [↑](#footnote-ref-10)
11. BIOS - Basic Input/Output System [↑](#footnote-ref-11)
12. SLAT - Second Level Address Translation [↑](#footnote-ref-12)
13. DEP - Data Execution Prevention [↑](#footnote-ref-13)
14. DHCP - Dynamic Host Configuration Protocol [↑](#footnote-ref-14)
15. Hyper-V - Windows Server Virtualization [↑](#footnote-ref-15)
16. OOP - Object Oriented Programming [↑](#footnote-ref-16)
17. CRUD - Create, Remove, Update, Delete [↑](#footnote-ref-17)
18. http://msdn.microsoft.com/en-us/library/windowsphone/develop/hh202860%28v=vs.105%29.aspx [↑](#footnote-ref-18)
19. http://msdn.microsoft.com/en-us/library/gg405484%28v=pandp.40%29.aspx [↑](#footnote-ref-19)
20. UI- User Interface [↑](#footnote-ref-20)
21. XAML - Extensible Application Markup Language [↑](#footnote-ref-21)
22. http://msdn.microsoft.com/en-us/library/ms752059%28v=vs.110%29.aspx [↑](#footnote-ref-22)
23. http://msdn.microsoft.com/en-US/library/windowsphone/develop/microsoft.phone.controls.phoneapplicationpage.state%28v=vs.105%29.aspx [↑](#footnote-ref-23)
24. http://msdn.microsoft.com/en-us/library/windowsphone/develop/System.Windows.Application%28v=vs.105%29.aspx [↑](#footnote-ref-24)
25. http://msdn.microsoft.com/en-us/library/windows/apps/dn629255.aspx [↑](#footnote-ref-25)
26. I/O - Input/Output [↑](#footnote-ref-26)
27. CPU - Central processing unit [↑](#footnote-ref-27)