Report for TDA367

Adam Andersson Max Fransson Anders Magnusson
Albin Söderberg

28 October 2018

Contents

1 Introduction **1**
1.1 Definitions, acronyms, and abbreviations 1

2 Requirements **2**
2.1 User stories 2
2.2 User interface 7
2.2.1 First sketch 7
2.2.2 Prototype screenshots 9

3 Domain model **17**
3.1 Class responsibilities 17

4 System architecture **18**
4.1 Model Package 19
4.2 View Package 20
4.3 View model Package 20
4.4 Service Package 21
4.5 Design decisions 21

5 Persistent data management **22**

6 Access control and security **23**

7 Peer review **23**
7.1 General Design 23
7.1.1 Do the design and implementation follow design principles? 23
7.1.2 Does the project use a consistent coding style? 23
7.1.3 Is the code reusable? 23
7.1.4 Is it easy to maintain? 24
7.1.5 Can we easily add/remove functionality? 24
7.1.6 Are design patterns used? 24
7.2 Is the code documented? 24

7.3 Are proper names used? 24
7.4 Is the design modular? Are there any unnecessary dependencies? 25
7.4.1 Is it easy to maintain?Does the code use proper abstractions? 25
7.5 Is the code well tested? 25
7.6 Are there any security problems, are there any performance issues? 25
7.7 Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts? 26
7.8 Can the design or code be improved? Are there better solutions? 26

1 Introduction

All the workout applications out there have one thing in common: they are unnecessarily complicated. You don't get very motivated to go work out when the app is hard to use, you have to learn a bunch of new things and you have no idea where to start.

The purpose of this application is to motivate people to exercise more by being easy to use and by providing pre-made workouts complete with instructions on how to perform them. To make users keep exercising, constant achievements rewarding regularity is added accompanied by challenges where the user can set personal records for popular and well known exercises. The application will also save statistics of the user's progress.

1.1 Definitions, acronyms, and abbreviations

MVVM - Model View ViewModel is a design pattern similar to Model View Controller (MVC)
Exercise - Exercises are for example: bench press, deadlift, push-ups
Workout - A workout contains a collection of exercises (contained by workout blocks) that are meant to be performed together in a session when working out.
Workout block - A block holds at least one exercise and has a multiplier number associated with it that represents the amount of sets to perform that/those exercise(s).

2 Requirements

2.1 User stories

User story 1

Story Identifier: 01
Story Name: Divided interface

Description

As an app user I want the interface to be divided into different sections, to more easily be able to focus on the chosen task.

Confirmation

Functional:
- Can I switch between tabs with the help of a navigation bar?
- Is the currently selected tab highlighted in the navigation bar?
- Can I click on the back button to go back to the previous page?

User story 2

Story Identifier: 02
Story Name: List of workouts

Description

As a non-experienced exerciser I want to see a list of pre-made workouts so that I can decide what type of workout I would like to do without knowing any in my head.

Confirmation

Functional:
- Can I see a full list of pre-made workouts?

- Can I click on a workout to see what it contains?

User story 3

Story Identifier: 03
Story Name: List of exercises

Description

As a non-experienced exerciser I want to see a list of exercises so that I can decide what exercises to do without knowing any in my head.

Confirmation

Functional:
- Can I see a full list of common exercises?
- Can I sort the exercises to find what I'm looking for?
- Can I click on an exercise to see more information about it?

User story 4

Story Identifier: 04
Story Name: Start workout

Description

As an exerciser I want to be able to start a workout so that I can get aid on what to do while focusing on doing the exercises.

Confirmation

Functional:
- Can I start a workout by selecting it from a list?
- Can I see a timer to know how long the workout has lasted so far?
- Can I pause the workout?
- Can I navigate around the app after having started a workout and then

come back without canceling it?

User story 5

Story Identifier: 05
Story Name: Exercise instructions

Description

As an exerciser I want to see instructions for different exercises so that I know how to perform them correctly.

Confirmation

Functional:
- Can I see the instructions for an exercise when I click on it?
- Can I see the instructions for the exercise while I'm doing its challenge?

User story 6

Story Identifier: 06
Story Name: Create workouts

Description

As an exerciser I want to be able to create my own custom workouts so that the training suits me.

Confirmation

Functional:
- Can I choose from all the exercises in the app and add them to my workout?
- Can I choose amount of sets and reps for each exercise?
- Does the created workout get added to the full list of workouts?
- Can I edit my workout after it is created?
- Can I remove my workout after it is created?

User story 7

Story Identifier: 07
Story Name: Create exercises

Description

As an exerciser I want to be able to create my own exercises so that I can customize my own workouts for whatever I want to do.

Confirmation

Functional:
- Can I choose a custom name, unit, category etc.?
- Does the created exercise get added to the full list of exercises?
- Can I edit my exercise after it is created?
- Can I remove my exercise after it is created?

User story 8

Story Identifier: 08
Story Name: Undertake Challenges

Description

As an exerciser I want to be able to try out different challenges, in order to challenge myself to get better and see improvement.

Confirmation

Functional:
- Can I choose which challenge to do from a list?
- Can I see the instructions for the challenge before I start?
- Does my score get saved?
- Can I see my high score on each challenge?

User story 9

Story Identifier: 09
Story Name: Earn Achievements

Description

As a user I want to earn achievements for completing tasks in the app, so that I can get a sense of pride and accomplishment.

Confirmation

Functional:
- Can I see a list of all the completed achievements and achievements in progress?
- Can I see my current progress on each achievement?
- Do I get new achievements to shoot for after completing the ones I have?

User story 10

Story Identifier: 10
Story Name: Exercise statistics

Description

As an exerciser I want to show statistics about how my exercise history, so that I can clearly see my progression in different exercises.

Confirmation

Functional:
- Can I save the statistics after performing a workout?
- Can I display the statistics for a specific exercise?

2.2 User interface

2.2.1 First sketch

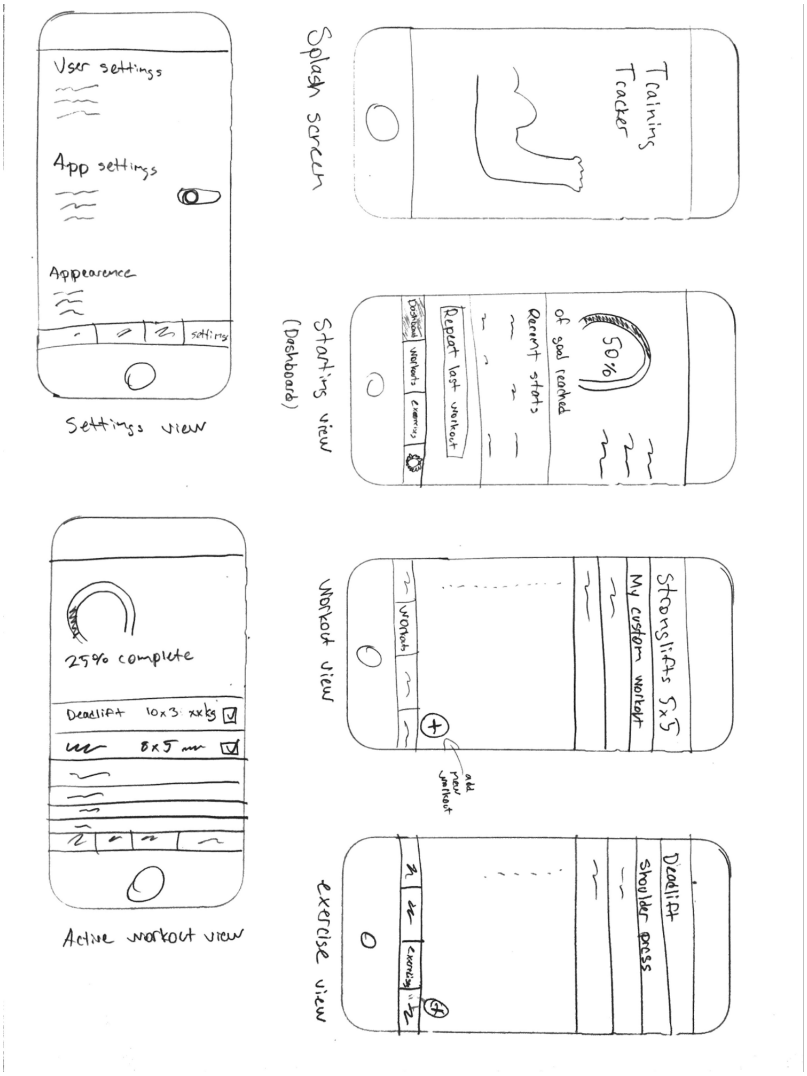

Figure 1: GUI-Sketches V1

The above are early sketches of the user interface. The main method of navigation through the interface was decided early, and can be seen in the sketches, a module tabs type of navigation. The user can navigate the app by selecting one of the main areas of the app (in this case “Dashboard”, ”Workouts”, “Exercises” and “Settings”. From within each tab the user can

navigate further. The Dashboard shows statistics and also acts as a start page. The Workout tab displays the list of all workouts, allows the user to create new workouts via the "+ button" in the downright corner, and leads to a more detailed view of a workout when one is selected from the list. The Exercise tab displays the list of all exercises, allows the user to create new exercises via the "+ button" in the downright corner, and leads to a more detailed view of an exercise when one is selected from the list. The Settings tab will let the user change the app settings.

2.2.2 Prototype screenshots

Figure 2: Home screen

The Home screen of the application [Figure 2] is what the User sees when the application is started. Below the logo and the icon a Bottom Navigation Bar is used, to let the User switch between the different views of the application. When a view is selected, the corresponding icon on the navigation bar turns purple to indicate which view is the active one.

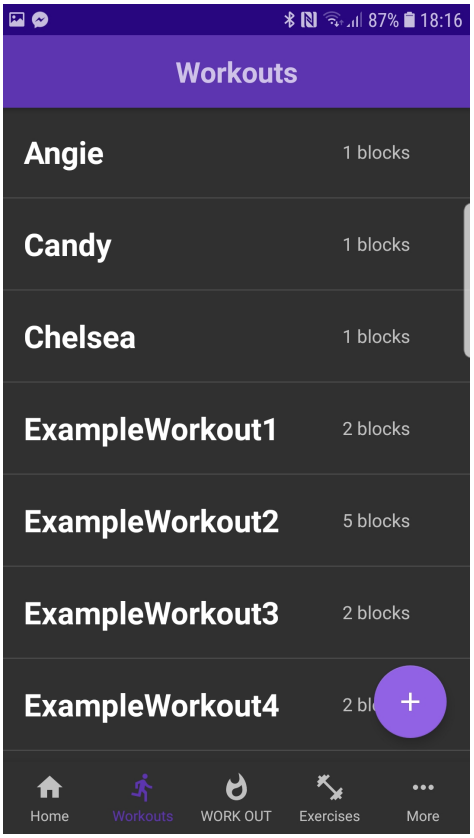

Figure 3: Workouts tab

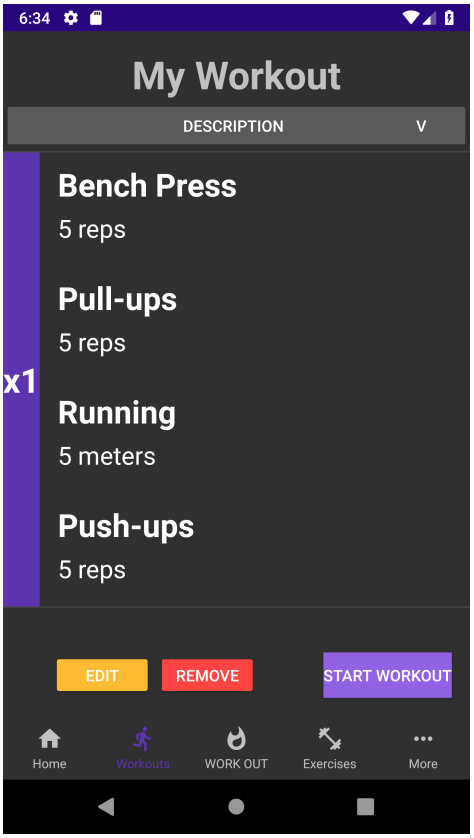

Figure 4: Workout details

The Workouts tab [Figure 3] lets the user scroll through available workouts, show more details about them and add new custom workouts. If the User selects a *custom* workout from the list, the user can choose to edit or remove it [Figure 4].

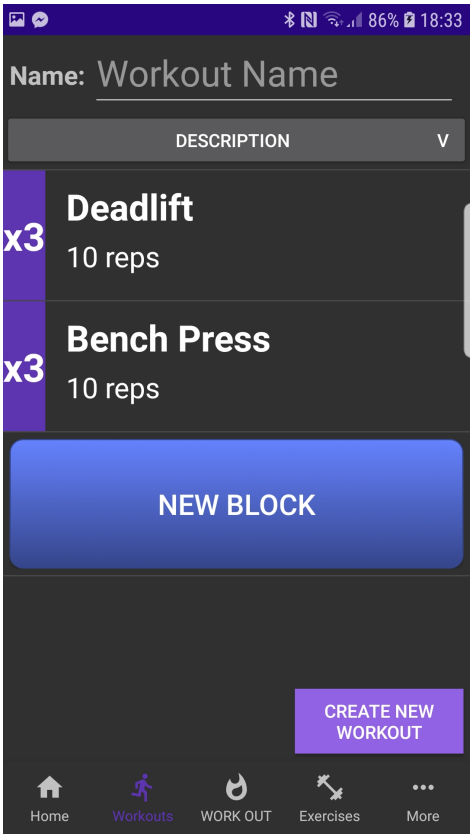

Figure 5: Workout creator

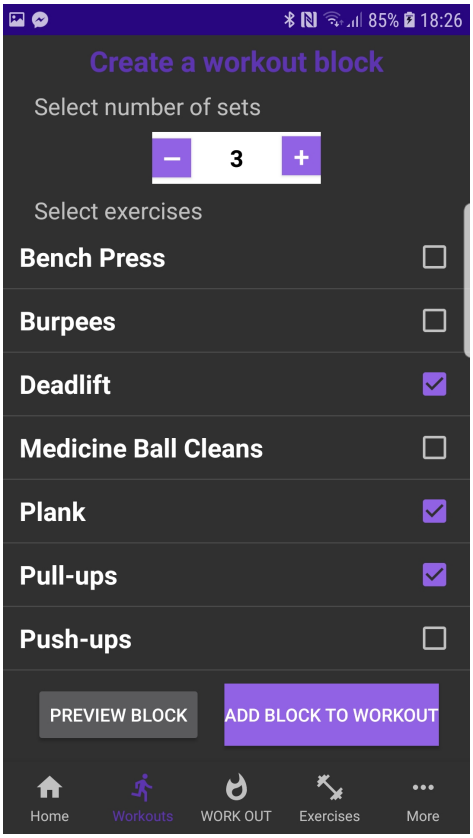

Figure 6: Workout block creator

The Workout creator [Figure 5] allows for the user to create a custom workout, filled with the base of any workout created in the Training Tracker application, the *Workout block*. When creating a *Workout block* the User selects the desired exercises and the amount of sets and reps for each exercise.

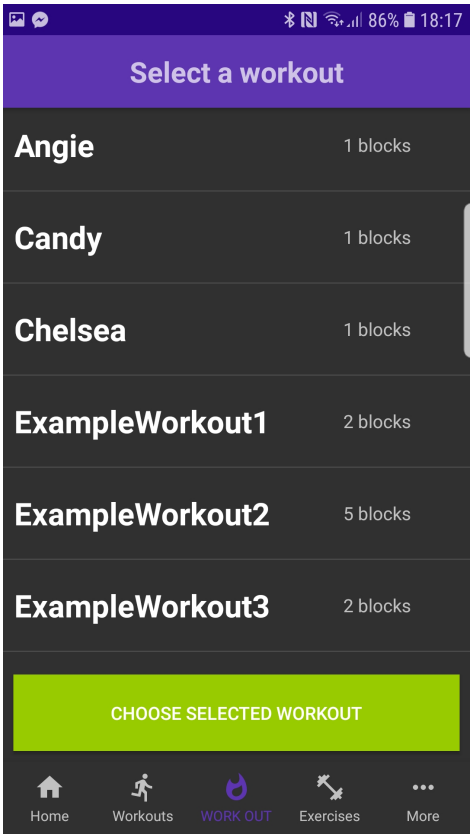

Figure 7: WORK OUT tab

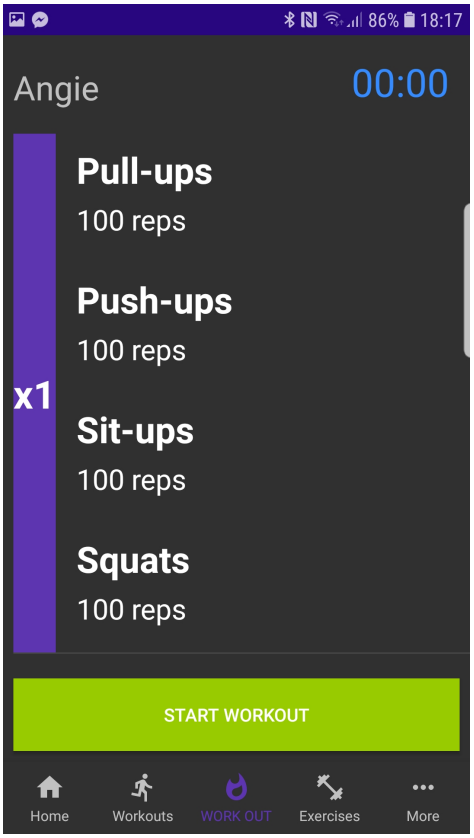

Figure 8: Active workout

When the WORK OUT view is selected [Figure 7], it displays a list of available workouts to choose from to start a new Active Workout. The Active Workout view displays the blocks of the selected workout, along with a stopwatch in order to let the User keep track of how long the User has spent on the workout. The User is also able to mark the workout as "finished" by holding the pause/finish workout button which shows after the workout is started.

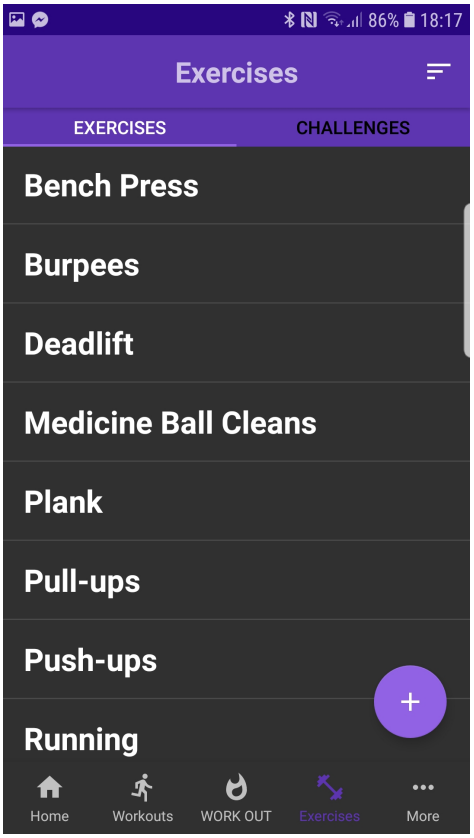

Figure 9: Exercises tab

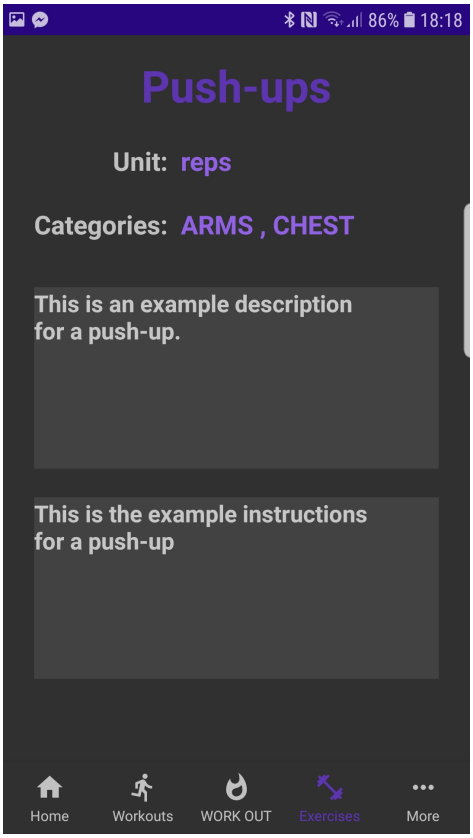

Figure 10: Exercise details

The exercises tab [Figure 9] lets the user scroll through available exercises. Clicking on an exercise takes you to a new page [Figure 10] to see more information about it. Clicking on the plus button takes the user to the exercise creator.

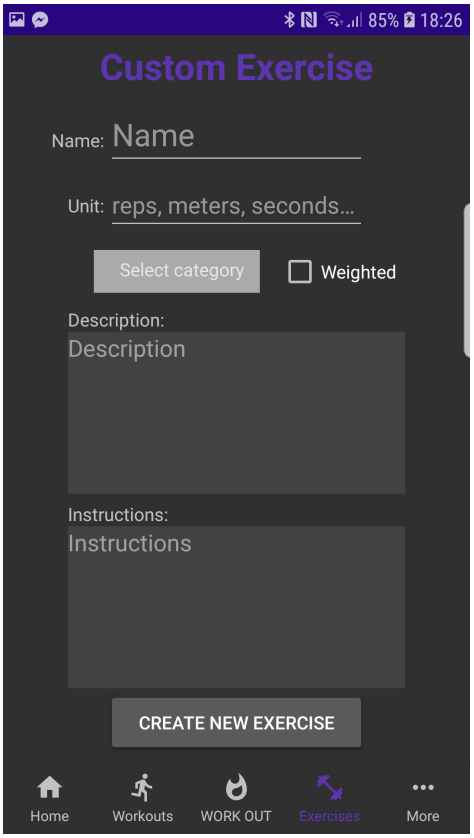

Figure 11: Exercise creator

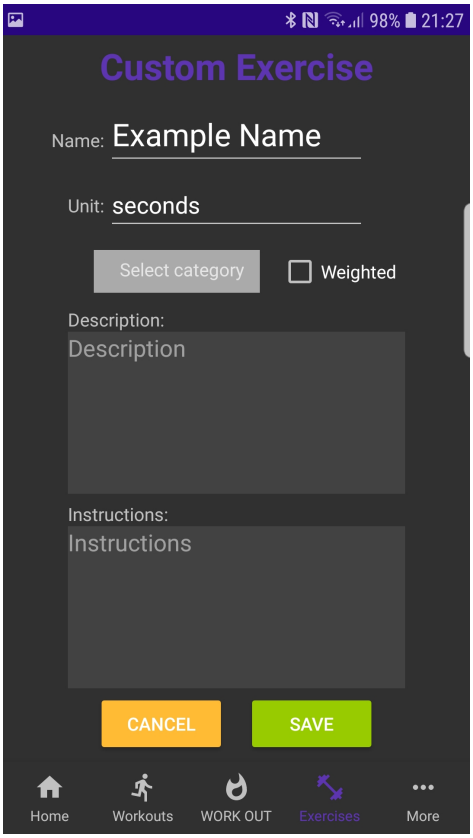

Figure 12: Exercise editor

In the exercise creator [Figure 11] the user can create their own custom exercise by filling in the fields. If the user clicks on "edit" in a *custom* exercise they get sent to the very similar exercise editor [Figure 12] where they can edit it.

In the challenges page [Figure 13] the user can choose to start a challenge which is basically an exercise where you can set a high score. When doing the challenge [Figure 14] the instructions are displayed and the user is prompted to put in the score they got.

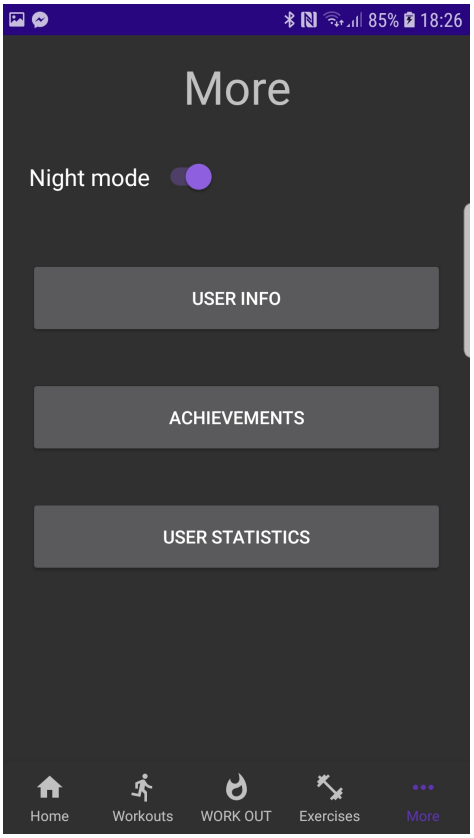

Figure 13: More tab

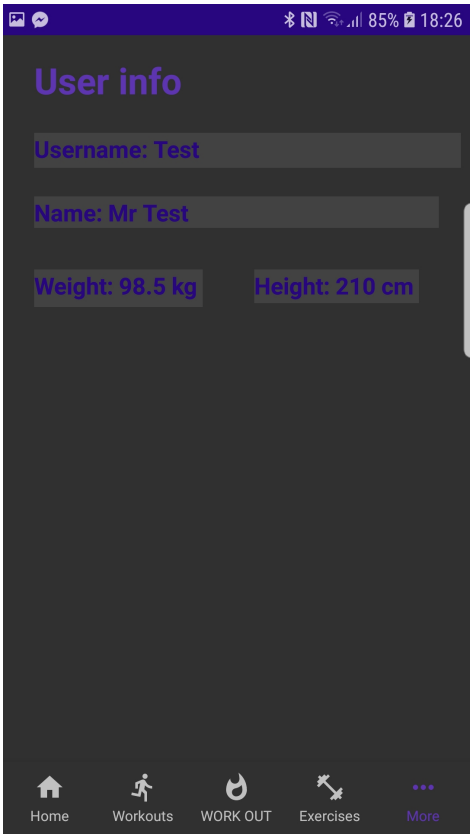

Figure 14: User info

The "More" tab contains functionality for displaying User info, User achievements and the User statistics [Figure 15]. There is also a switch for enabling the "Night mode" theme (currently shown).

The User info view shows the current information for the User of the application [Figure 16].

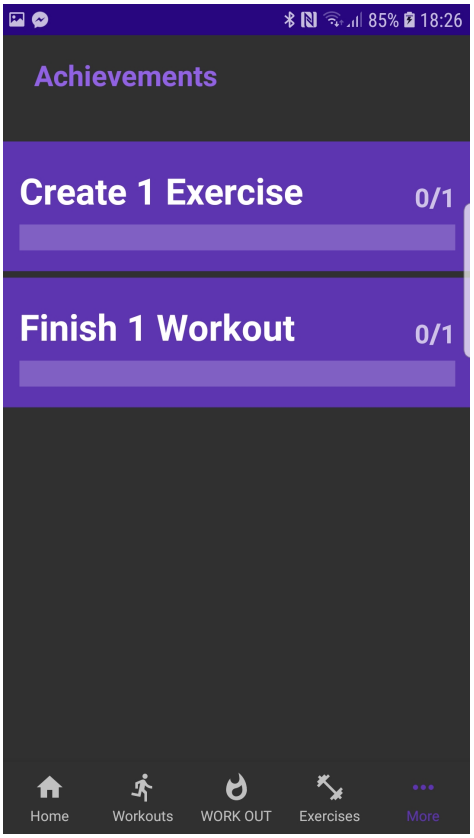

Figure 15: Achievements

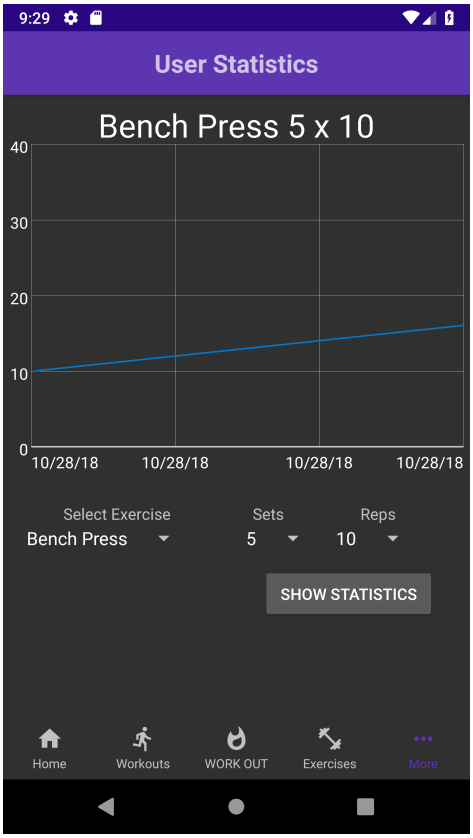

Figure 16: User statistics

The application uses Achievements to motivate the User to perform certain actions with the application. The User can track the progress via the Achievements page [Figure 17]

For the User to be able to keep track of how the workouts are going, the User can show exercise specific information [Figure 18]. The graph displays the weight increase for a selected exercise, and the combinations of sets and reps.

3 Domain model

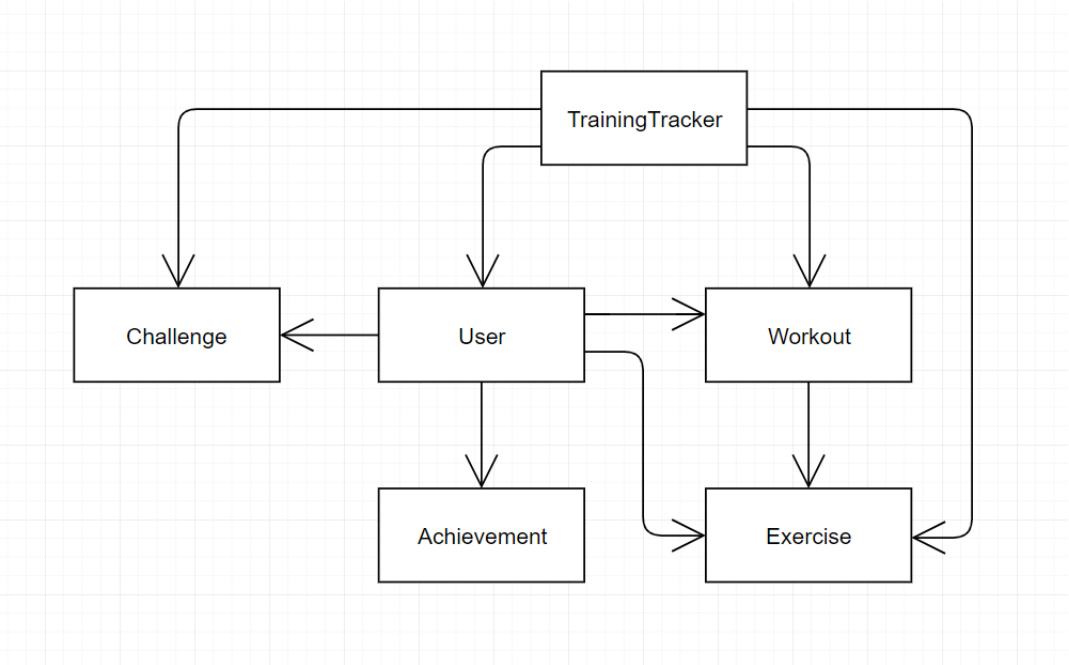

Figure 17: Domain model

3.1 Class responsibilities

The main class of the domain model is Training Tracker. Training Tracker is responsible for holding lists of the pre-made Workouts and Exercises in the app. It also holds the user-information, and the available challenges. The User of the app is responsible for holding the custom-made Workouts and exercises which are unique to the individual user. The User also stores information about personal challenges and achievements. Exercises contain the name and description of an exercise as well as the training category it belongs to. Workouts contain a name and description and also a list of exercises to perform.

4 System architecture

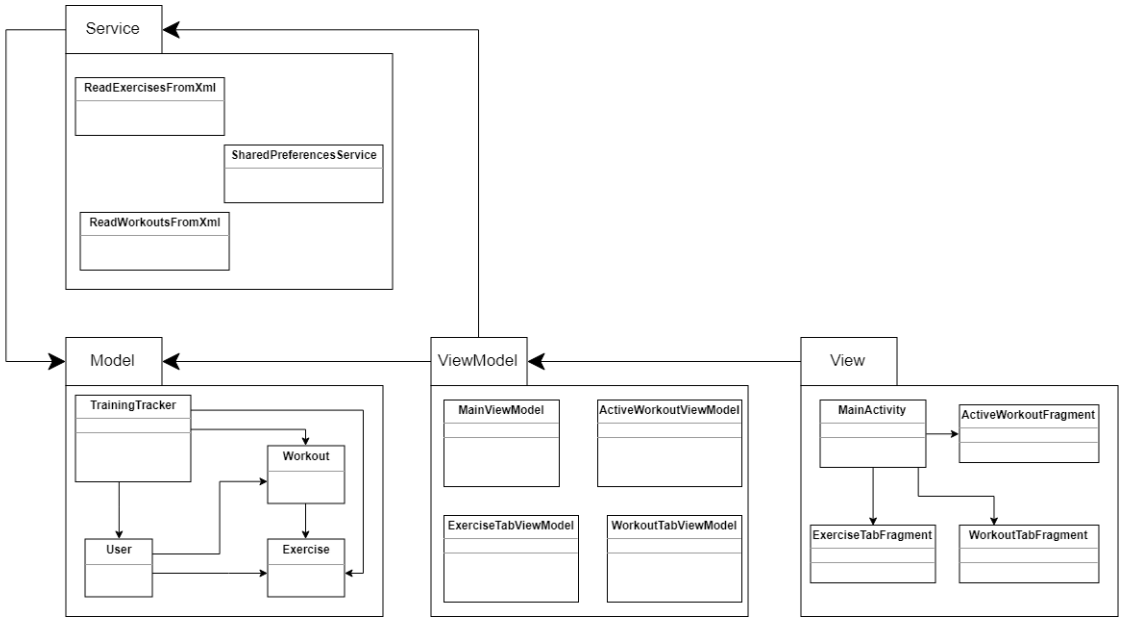

Figure 18: Packages

The above UML diagram shows the different packages within the application and the dependencies between them. The application has been built based on the (MVC-like) design pattern MVVM (model, view, view model).

4.1 Model Package

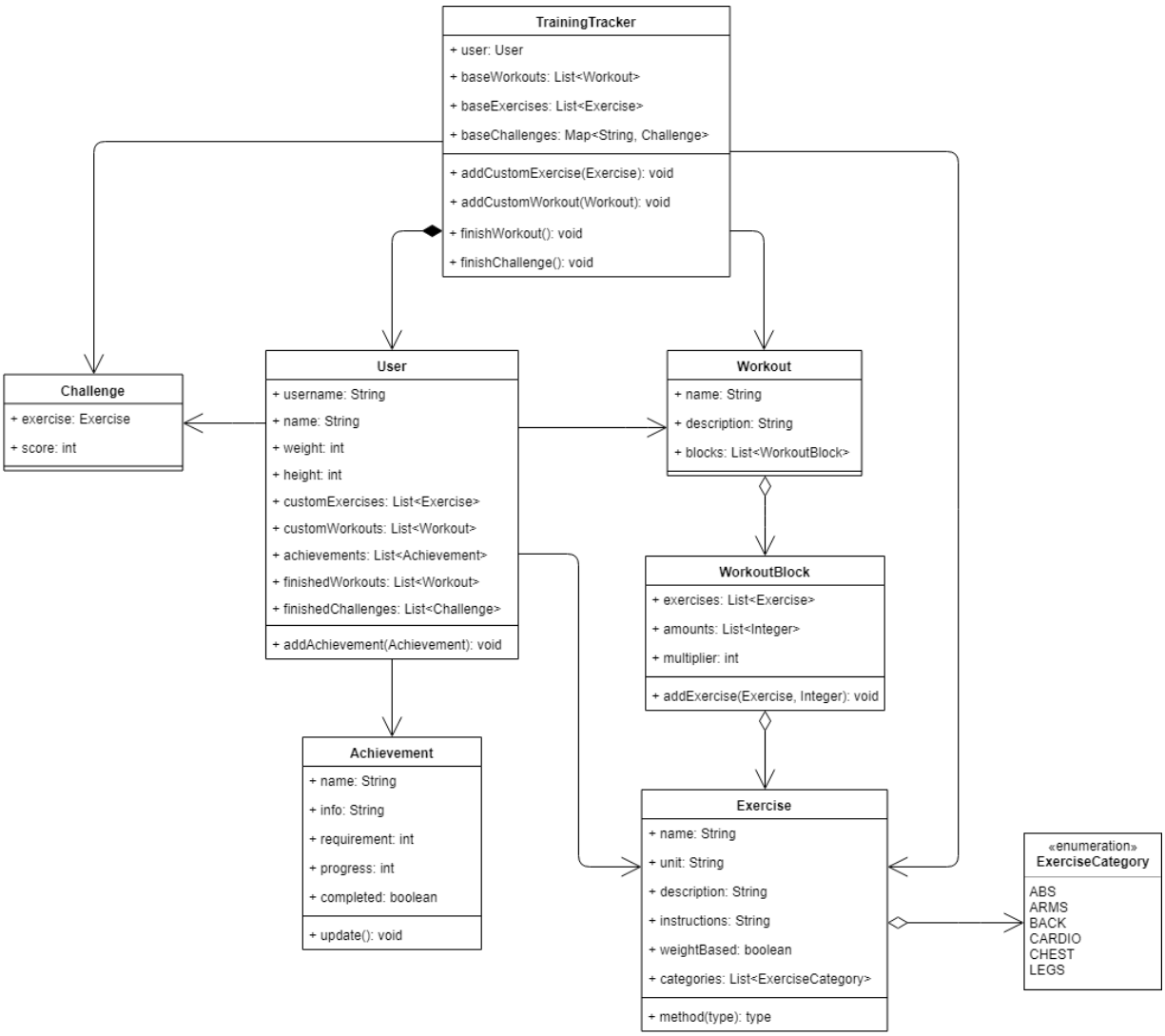

Figure 19: Design model

The above is the design model, showing the classes within the model package(not including interfaces). This is the backbone of the application where all the data, logic and rules of the app is. TrainingTracker is the main class, containing the data used during runtime. It has a reference to the current User and its data, and holds lists of the Workouts, Exercises and Challenges in the app. Through the reference to TrainingTracker, the rest of the model

can be accessed and modified.

4.2 View Package

The "View package" contains the visual part of the application, the GUI. It is responsible for displaying the data in the model to the user, while also letting the user modify the data through interaction with the interface.

The main class of this package is the MainActivity, which instantiates the View and the MainViewModel. It implements the interface NavigationManager which is responsible for all navigation between fragments. When a fragments needs to switch over to another fragment it does so via the NavigationManager.

The view in this application is designed in such a way that each separate screen of the interface is represented by a Fragment. A Fragment can be displayed as a part of the screen or cover the entire screen. It holds the information of how it is to be displayed, and what components it is made of. Each fragment also has a corresponding .xml file which defines the visual elements of the fragment, these elements are then used and manipulated in the fragments class. Each fragment also has a reference to a ViewModel, which it can use to populate its components with the data from the model. When the app switches from one fragment to another it does so either through the bottom navigation bar (and its listener in the MainActivity class), or through the reference to the MainActivity in the current Fragment.

4.3 View model Package

The "ViewModel Package" is responsible for handling the communication between the View and the Model, providing controller functions and reducing the direct dependency on the model from the view, making it more modular. Whenever the MainActivity or any of the fragments in the View package

needs to access or modify data in the Model, it uses the reference to its respective ViewModel object. Having a different ViewModel for each fragment in need of accessing or modifying the model helps to distribute functionality and keep each ViewModel specific to its usage. This ViewModel, in turn, holds an instantiation of the TrainingTracker from the Model package in order to access data and logic in the Model, and contains methods to access and modify the model. Also when the view would need to get notified of changes in the model it is done via subscribing to its view model, listening for changes.

4.4 Service Package

The "Service Package" contains various functionality for importing pre-created exercises and workouts, and functionality for serializing and de-serializing application data, in order to save information between app-sessions.

4.5 Design decisions

MVVM design pattern is a way to separate the different responsibilities of packages within an application(with a GUI). Designing with this pattern in mind will therefore help you adhere to the Single Responsibility Principle. The ViewModel-package has several ViewModels. Each to be connected to one/several fragments in the view. Having several ViewModels helps to separate the responsibilities throughout the package instead of having one big MainViewModel.

"Law of Demeter" design pattern is also used in this application. The classes in the view-package need to access and change data in the model and do so through their view-models. They mostly do not need to directly be in contact with the objects in the view-model to do that. If a view needs the name of an exercise, it does not need to receive the exercise and call getName() on the object. The view-model can take care of that, and only pass along the name

representing the String. That way the responsibilities will be separated and dependencies reduced.

Observer pattern is another pattern being used in the application. It is used during list filtering, in order to update the displayed list when a filter is applied. By using an observer there is no need to have a strong dependency go both ways.

Interfaces are used on all classes in the model, except for enum and abstract classes. Since all classes in the ViewModel-package and the ListView-adapters(in the View-package) use only references to interface-objects, the concrete dependencies on the model are greatly reduced because classes of the model can be independently changed as long as they fulfill their respective interfaces.

5 Persistent data management

The pre-made exercises and workouts are stored in easily readable xml format in the res/raw/exercises.xml and res/raw/workouts.xml files respectively, and are loaded during startup by their respective service classes. The challenges are loaded similarly, but instead of a structured xml file it is done through the res/raw/challenges.txt file which just contains the names of each exercise to create the challenge for. Reading that text file is done with the help of the ReadLinesFromFileService (in the service package) which is not reliant on this app at all and can be re-used in any other project.

The user specific data (user variables, custom exercises/workouts, challenges and achievements) are stored using Json, utilizing Jackson, Java's previously used standard library for Json usage.

Icons used in the application are stored in Vector format in the res/drawable directory, native to the android application.

6 Access control and security

The application currently does not include different roles for usage. The only available role is the user which has access to all the functionality that the user interface provides. Each user has the same options as every other user. The app does not support multiple uses on a single instance of the app but the thing that may differ between two users running the app on two different units are the individual exercises and workouts that the users themselves create as well as the individual user data.

7 Peer review

7.1 General Design

7.1.1 Do the design and implementation follow design principles?

Many of the view models break Single Responsibility Principle by handling both view related things as well as managing logic and operating on the model.

7.1.2 Does the project use a consistent coding style?

The coding style and convention is inconsistent throughout the program, for example OnClickListeners initiated in different ways in MenuVM, MainActivityVM and AddHabitVM.

7.1.3 Is the code reusable?

The view models are not very reusable as they highly depend on the system (android) and the model at the same time.

7.1.4 Is it easy to maintain?

Some view models are very big and have multiple responsibilities (e.g. MainActivityVM, AddHabitVM) so it's hard to get an overview or know what gets affected when changing something.

7.1.5 Can we easily add/remove functionality?

The way the achievements work with abstraction and the AchievementFactory allows for easily adding more achievement types.

7.1.6 Are design patterns used?

There are many design patterns used. Some examples that can be found are: MVVM, Singleton, Observer, Factory, Adapter.

7.2 Is the code documented?

There are good comments in some places but many classes are left without any documentation. There is no class documentation which would make the purpose of each class more clear.

7.3 Are proper names used?

Sometimes too general names such as "String string" and not always clear use of english (typos?). Variable names often do not include the type of object it holds, for example buttons often only contain a single word "save", "start" etc (instead of saveButton/startButton). Abbreviations for component names is used inconsistently, for example monCxb and mondayCheckbox or SaveButton, SaveBtn and Save are used interchangeably.

7.4 Is the design modular? Are there any unnecessary dependencies?

Model and view model are well divided into their own packages, and the model doesn't have any bad dependencies to anything outside. But within the ViewModel package there could be more modularity like grouping everything that has to do with logging in into one package.

7.4.1 Is it easy to maintain?Does the code use proper abstractions?

There are many interfaces/abstract classes used in the model package, for example the achievement system has a good setup with an abstract Achievement class and subclasses that have different implementations.

7.5 Is the code well tested?

Tests are often not very thorough, several of them only test a constructor and then checks for a null reference.

7.6 Are there any security problems, are there any performance issues?

HubbaModel being a singleton could potentially give access to things where it shouldn't be accessed.

Habits are marked as done using the String name variable. If there are several habits with the same name(for example brush teeth in the morning and brush teeth in the evening) both will be marked as done when one of them is marked as done. When using mainActivityVM.getHabit(title)it might return the wrong habit, if there are several with the same name.

Habit class contains odd functionality, public upStreak(Habit habit) increases the streak variable of the passed habit, but there is no reason to pass a habit to another habit just to use this functionality. Functions void isDone and void setDone perform the same functionality but in different ways.

7.7 Is the code easy to understand? Does it have an MVC structure, and is the model isolated from the other parts?

The model is well isolated.

There are major blocks of code that are not commented and the functionality is obfuscated due to the many lines of code.

Presumably the layout resource folder with the xml files is seen as the view part but the view models (controllers) also do many view related things. There are also some classes that perhaps would fit better in a Service package(for example Notification Receiver).

7.8 Can the design or code be improved? Are there better solutions?

Overall it is a very nice application but there is a lot of room for improvement.

To reduce size and responsibility of some view models you can make activities part of the view instead and create separate view models (that know nothing about the interface) that they can bind to. This would also make the view models a lot easier to test.

Instead of having HubbaModel as a singleton, instantiate it once and then pass that reference around to whoever needs it, for example with the help of a view model.