Developing a Gamified Fitness Application for Truck Drivers

Bachelor's thesis in Computer Science and Engineering

ANTON ERICSON
MARCUS KAREGREN

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

DEGREE PROJECT REPORT

Developing a Gamified Fitness Application for Truck Drivers

ANTON ERICSON
MARCUS KAREGREN

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2019

Developing a Gamified Fitness Application for Truck Drivers
ANTON ERICSON
MARCUS KAREGREN

Supervisor: Nick Smallbone, Functional Programming division, Department of Computer Science and Engineering, Chalmers University of Technology
Examiner: Jonas Duregård, Department of Computer Science and Engineering, Chalmers University of Technology

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 (0)31-772 1000

Cover picture: "Award, cup, leader, prize, sport, trophy, winner icon"[1] by Laura Reen, licenced under CC BY-NC 3.0[2]

Department Of Computer Science and Engineering
Gothenburg, Sweden 2019

[1] https://www.iconfinder.com/icons/2064041/award_cup_leader_prize_sport_trophy_winner_icon

[2] https://creativecommons.org/licenses/by-nc/3.0/

Abstract

A fitness application for Volvo Trucks' Android based infotainment system has been developed. The purpose of the project is to increase truck drivers motivation for physical activity, by using gamification. The gamification aspects of our application are a point and level system, where the user is rewarded with points upon completing a workout. These point are then used to determine the user's level. This means that to level up, the user has to perform workouts.

The application has been developed with a focus on following the development techniques recommended by Volvo Trucks and Google, to ensure that Volvo Trucks have the opportunity to continue the development of the application. This has dictated the choices we have made when developing the application. The focus has not been to implement a large number of features, but to implement a small number of features in a correct way. This meant that a large part of the project had to be spent on learning and then implementing the different design patterns and techniques correctly.
This resulted in an application that we are pleased with. The application follows the guidelines given by Volvo Trucks and Google and there are many opportunities to expand the application with new features or more workouts.

Keywords: android, application, gamification, fitness

Acknowledgements

First of all we would like to thank Jonas Vikentorp and Richard Kahl, Sigma Embedded Engineering, for the opportunity to do this project at Sigma together with Volvo Trucks and providing us with the tools and support necessary to complete the project.

We would also like to thank everyone at Volvo Trucks, for the support during the project and the opportunity to learn about and develop an application for their platform.

We would like to thank our supervisor, Nick Smallbone, Computer Science and Engineering department at Chalmers, for his support, ideas, and inspiration during the project.

We would like to express our gratitude towards everyone that has been involved in this project in any way. It has been a great opportunity for us to learn more about software development.

Thanks to David Frisk for providing the Overleaf LaTeX template used to write this thesis.

Anton Ericson & Marcus Karegren, Gothenburg, June 2019

Contents

List of Figures

Listings

1

Introduction

1.1 Background

A sedentary lifestyle can increase the risk of both cardiovascular disease and type 2 diabetes [1]. At particular risk are truck drivers who spend up to 9 hours a day sitting down [2]. This can have a negative impact on the driver's health if not complemented with physical activity. Volvo Trucks want to increase their drivers' well-being by motivating them to perform some physical activity during their breaks.

The purpose of this project is to help the drivers to feel more motivated to perform these physical activities. This can be achieved in many different ways; this project will develop an application for the truck's built-in infotainment system that can help motivate the drivers. To increase motivation, the application will feature some aspects of gamification.

The project will be carried out at Sigma Embedded Engineering on behalf of Volvo Trucks. Sigma is a consulting company and the majority of the work will be done at their office. Volvo Trucks will provide contact to a development team, to discuss the development of the application. The result of this project is owned by Volvo Trucks.

1.2 Scope

This project will consist of investigating gamification and developing an Android application for the Volvo Trucks Android based infotainment system.

The application will be built according to Volvo Trucks' recommendations so that it can be further developed by Volvo Trucks. The application will have limited functionality and a basic user interface. The goal of the application is that the user can browse through different workouts, select a workout, perform the workout, get rewarded and see their level progression. The user will also be able to view their profile and see their level progression. The level system will be explained in the thesis.

Gamification is a broad term and this thesis will only cover some aspects of it. Not all gamification aspects covered in the thesis will be implemented in the application. The gamification aspect featured in the application will be a point- and leveling-system.

The application will feature a limited amount of exercises available in different workout sessions. The exercises will be implemented in a modular way, thus making it easier to add more exercises in the future. Each exercise will be accompanied by a text and an animation explaining the exercise. The animations will be provided by Volvo Trucks.

2

Theory

2.1 Motivation

Motivating people to exercise is a hard task and there are many different approaches to it. There are two main types of motivation, extrinsic and intrinsic. Extrinsic motivation is motivation coming from outside reward or punishment, for example your salary. Intrinsic motivation comes from the individual itself, for example an athlete is motivated to push themselves because they want to become the best at their sport [3].

Since intrinsic motivation comes from the individual, the form of these motivators vary greatly. Therefore it is important that you cover the entire spectrum of motivators to be able to appeal to the majority of people. Using gamification in the application is a way to make it possible to appeal to the majority of people and what intrinsically motivates them. The motivators in the application combined with the health benefits of exercising will hopefully make the user continue to use the application and exercise, since exercising increases overall happiness [4].

2.1.1 Gamification

K. Werbach and D. Hunter [3] defines gamification as: "The use of game elements and game-design in non-game contexts". Gamification can be found in many popular fitness applications. Some gamification aspects commonly found in fitness applications are leaderboards, achievements, and a level system. Different gamification aspects will have different effects on people. That is why it is important to implement several different types of gamification to try to motivate the majority of users.

2.1.2 Self-Determination Theory

Self-Determination Theory(SDT) is a psychological theory that explains how people are motivated. There are a lot of gamification elements that can be explained with the SDT. K. Webach and D. Hunter [3] explains SDT as: "SDT focuses on what humans beings need to allow their innate growth and well-being tendencies to flourish".

There are three different categories in SDT; Competence, autonomy and relatedness. Competence refers to feeling that your skills are being tested, for example, the ability to compete against someone or increase the difficulty of exercises they complete. Autonomy refers to feeling that your control your own life and your choices, for example, allowing the user the ability to choose what kind of workouts they want to complete. Relatedness refers to feeling that you connect with other people and have social interactions, for example, being notified when your friend performs a workout, in some kind of social feed.

2.2 Development

The application is developed according to Volvo Trucks' recommendations to ensure that they can continue the development. The different techniques and design patterns recommended by Volvo Trucks will be explained in this section.

2.2.1 Android Studio

Android Studio is used to develop the application and access its built-in Android emulator that is used to simulate the infotainment system found in the Volvo trucks. Android Studio is Google's official integrated development environment (IDE) for developing Android applications. Android Studio was chosen for this project due to both Volvo Trucks' and Google's recommendations.

2.2.2 Kotlin

The application is developed in Kotlin. Kotlin is a programming language developed by JetBrains, released in 2016 [5]. Kotlin is similar to Java and fixes some issues that Java suffers from. For example, Kotlin fixes null reference issues [6], by controlling null references with a type system. Kotlin's type system "distinguishes between references that can hold null (nullable references) and those that can not (non-null references)" [7].

2.2.3 Model-View-ViewModel

Model-View-ViewModel (MVVM) is a software architectural pattern that is popular in the development of Android applications. Model refers to the business logic and data for the application and is completely independent from the User Interface(UI). It handles the processing of the problem domain and stores the state of the application [8]. The view is responsible for the UI. This means it only handles the structure and layout of what the user sees on their screen [9]. The ViewModel acts as a intermediary between the model and the view [9]. The View has no direct communication with the business logic and data found in the Model. Therefore it needs to observe the ViewModel while it receives and sends data to the Model (See figure 2.1) and then make changes to the UI when necessary.

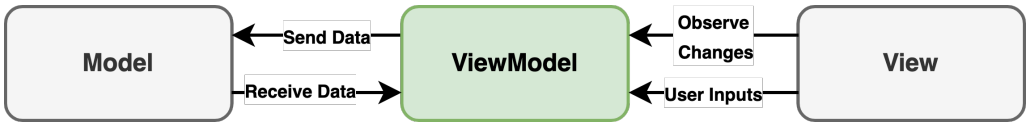

Figure 2.1: Illustration of MVVM communication.

2.2.4 Android Jetpack

Android Jetpack is a collection of different software components for Android. The purpose of Jetpack is to make development of Android applications easier [10]. Jetpack has three main advantages compared to how Android applications were previously developed. Firstly, it accelerates development by taking advantage of Kotlin and by making the different components work individually as well as together [10]. Secondly, it eliminates a lot of boilerplate code by assisting and managing navigation through the application and other background tasks [10]. Lastly, it takes advantage of modern design techniques such as separation of concerns and testability [11].

2.2.5 Fragments

Fragments are a UI component of Android Jetpack. They are used to create a modular and flexible UI for the user to interact with. A window in an application is represented by an Activity, which allows you to customize the UI of the window. The advantage of using Fragments within an Activity is that you can have multiple fragments inside one Activity window and reuse Fragments between different Activity windows [12]. Each fragment has its own lifecycle, which means all interactions with them are individual and they do not affect the other Fragments in the Activity. This allows a Fragment to be replaced with another one inside the same Activity (see figure 2.2) without affecting the other Fragments in the window.

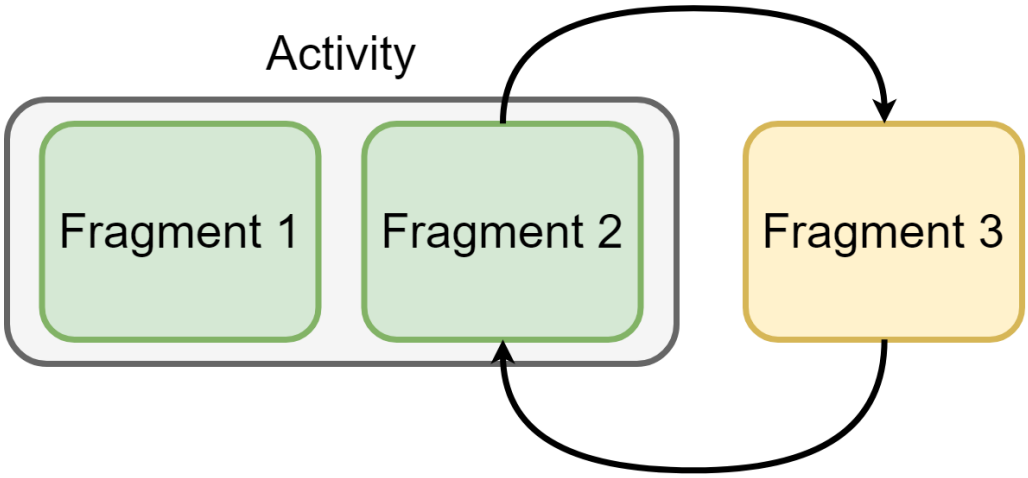

Figure 2.2: Illustration of an Activity with replaceable Fragments inside it.

2.2.6 Room Database

The "Room Persistence Library" is an abstraction layer over the SQLite [13] database management system. Room is used in the application to create and pre-populate a local database. The application will communicate with the database through a repository (see figure 2.3). This will separate the database from the rest of the application and make it possible to use a remote database later, without changing the application.

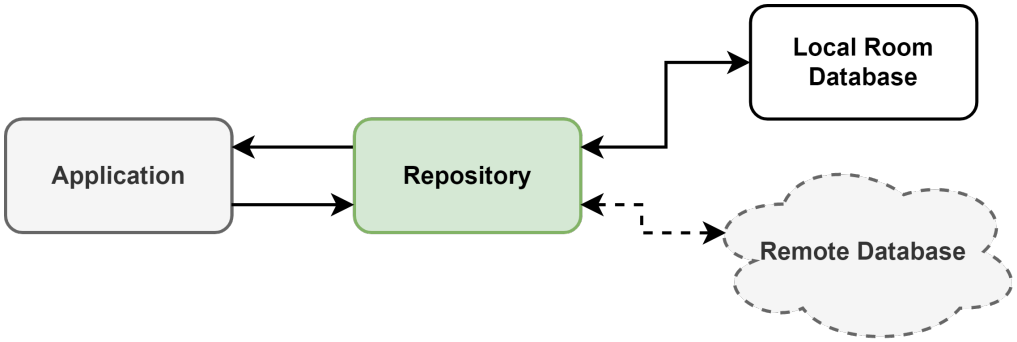

Figure 2.3: Illustration of communication between application and database.

2.2.7 Dependency Injection

Dependency injection is a programming technique used to reduce dependencies between classes [14]. This means the classes are not directly dependent on each other but instead they depend on an injector. The injector is a separate class that passes objects to other objects that need access to them.

This makes isolated unit tests easier since the classes does not depend on each other. It also reduces the amount of "boilerplate code" by not having to instantiate each object that is needed in a class. This project will use Koin for dependency injection due to recommendations from Volvo Trucks. Koin is a lightweight dependency injection framework for Kotlin [15]. Koin has built-in support for Android Architecture ViewModels, which makes it a good choice for this application.

3

Method

3.1 Research

The first phase of the project will consist of researching gamification, motivation, and exercises.

3.1.1 Gamification

To decide on what aspects of gamification to implement in the application we will research gamification and how the aspects motivate different people. It is important that the aspects cover a wide range of intrinsic motivators, to ensure that a majority of people will find the application motivating to use. To understand what motivates people we will also research motivation to get a better understanding of how different people can feel motivated.

The research will consist of reading literature and online sources regarding gamification and motivation. We will examine popular fitness applications for inspiration and ideas on how to implement the gamification aspects. When the research is done our findings will be evaluated to decide on which aspects to implement. A gamification aspect will be evaluated based on the ability to implement it within our time frame, as well as the ability to expand the aspect into something bigger.

3.1.2 Exercises

The exercises that will be featured in the application are selected by us. We decided to focus on stretching exercises due to them being relatively simple to perform in the limited space of a truck cabin. Since the exercises are not the main focus of the project, a limited amount of time will be spent on researching exercises. The research will consist of reading different online sources explaining stretching workouts and exercises. The only requirements for an exercise are that it can be performed within the limited space of a truck cabin and that it can be performed within a 7 minute time frame.

3.2 Development

When developing the application the focus will be to fulfil our goals and follow Volvo Trucks' recommendations. This is to make sure they can continue the development of the application. The goal, in terms of functionality, is to implement the user flow illustrated in figure 3.1. The user flow illustrates how a user will be able to navigate through the application when selecting and performing a workout.

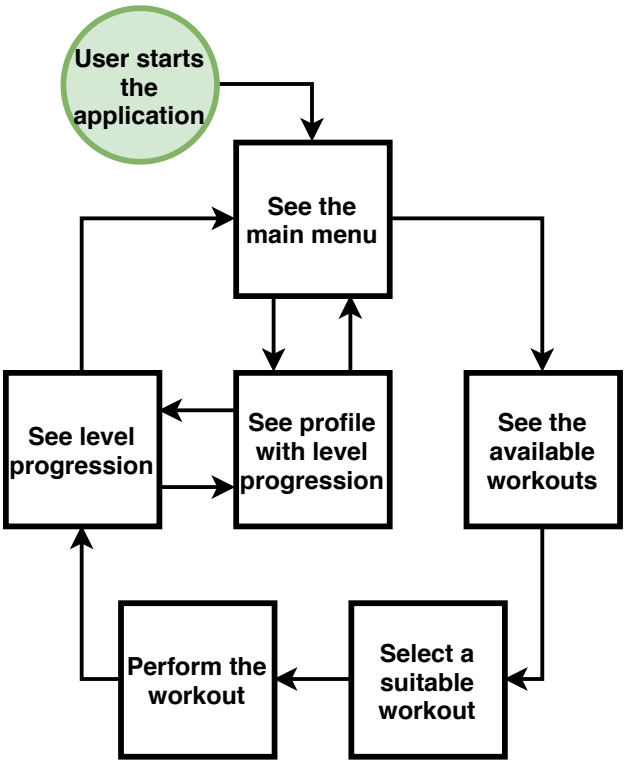

Figure 3.1: Illustration of intended user flow.

3.2.1 Workflow

Git and GitHub will be used for version control. A feature-branch system will be used. This means that when development on a new feature starts, a local branch will be created. The feature will then be fully developed in that branch and upon completion pushed as a branch to GitHub and a "pull request" will be created. When a pull request is created, GitHub performs an automatic integration test to check if the branch can be integrated in the master branch without conflicts. The pull request system also allows us to inspect the commits in the specific branch along with all the code that was added, removed, or changed. When the code has been reviewed and approved GitHub will merge the feature branch into the master branch.

The development of the application will be split up between us to increase efficiency. As this could lead to a lower general understanding of the application, we will review each others code and make sure that we understand it. We will comment the code to increase understandability, ease the review process and ensure that Volvo can continue the development. The review process is done to increase our general understanding of the code and ensure that the design patterns and techniques are used correctly. During the project we will have continuous meetings with both Volvo and our supervisor at Chalmers to ensure that the project is moving in the right direction.

3.3 Milestones

To give structure to the development, some milestones have been created. The milestones consist of important features of the application that need to be implemented. When developing the application we will work towards these milestones, which will help us reach our goals.

Room Database	A database needs to be set up. The database will contain workouts, exercises and users.
Main menu	The main menu should feature two buttons, one to see your profile and one to go to the list of workouts.
List of workouts	The list of workouts should display all available workouts with an explanation when clicked.
Workout session	When performing a workout, a fragment will be used to show each step of the workout session will be used.
Profile screen	The profile screen will show information about the user.

Workout finished screen When a workout is finished, a screen showing the users progress is shown.

Visual aid for exercises Each exercise will have some sort of visual aid, provided by Volvo.

3.4 Success Criteria

We have set a couple of criteria to be able to determine if the project has been successfully completed. They are connected either to the gamification of the application or the development of the application itself.

3.4.1 Gamification

For the implementation of gamification to be considered a success, a point and level system has to be implemented. The application should give you points for completing exercises. The level system should be based on the points that the user earns.

3.4.2 Development

For the development of the application to be considered a success, Volvo's guidelines have to be followed. We need to enable the possibility of further additions of gamification aspects. This means that the application has to be developed in Kotlin using a Model-View-ViewModel pattern. The application should take advantage of components from Android Jetpack and use Koin for its dependency injection. If all milestones have been reached and the application is designed like the intended user flow chart found in figure 3.1, then we can say that the application has been successfully developed.

4

Implementation

4.1 Workouts & Exercises

The application features workouts and exercises. A workout is a collection of exercises. The user can browse through the workouts, see the exercises they contain and then choose a workout to perform.

In the application there are two different workouts available, containing three and four exercises respectively. They are stretching exercises and can all be performed while sitting down. Since the selection of exercises was not the focus of the project a collection of arbitrary exercises was selected based purely on the fact that they could be performed while sitting down. The exercises have not been tested in a truck cabin.

4.2 Gamification

To motivate the user to perform workouts a points and level system is implemented. These aspects were chosen based on the reasearch we did in the begining of the project. The motivators behind a points and level system falls under the "Competence" category of the Self-Determination Theory. The reasoning behind choosing these features are that they allow us to add new features that are based on them. The user receives points upon completing a workout, and as the user collects more points their level increases. To determine what level a user is, the application uses the Level class, which contains information about the amount of points that are required for each level.

When an user has completed a workout they will be presented with a screen displaying their current level progress and the amount of points they have collected (see figure 4.1).

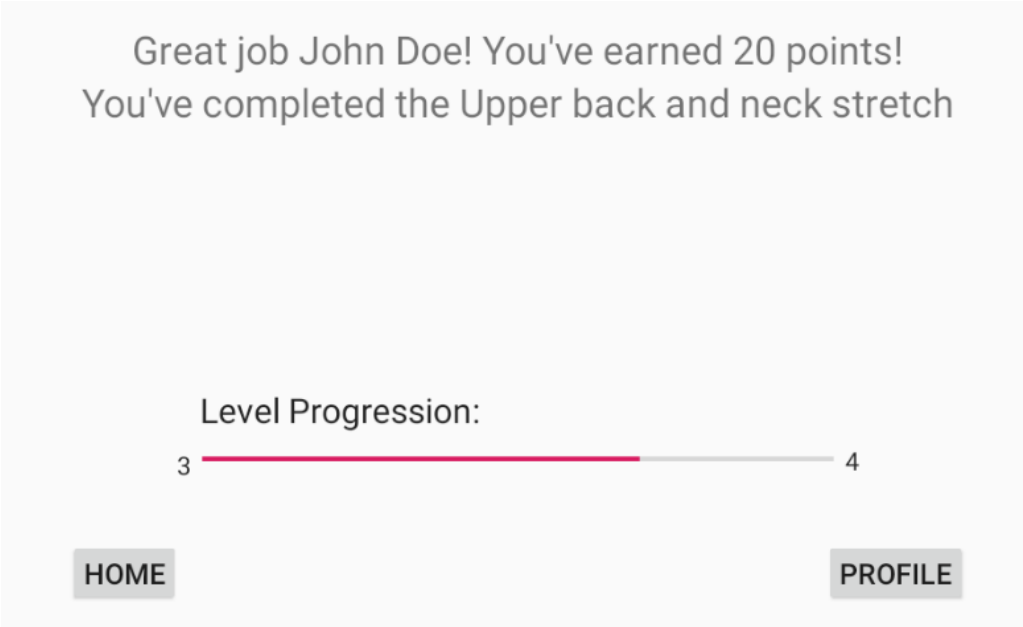

Figure 4.1: The progression screen displayed upon completing a workout.

The points received are based on the number of exercises the workout consists of. In figure 4.1 the user has completed the workout called ”Upper back and neck stretch” and has earned 20 points. This is meant to motivate the user to collect more points to be able to level up. The user can also go to their profile screen in the application to see their progression (see figure 4.2).

Figure 4.2: The information displayed in the profile screen.

The profile screen displays the users profile information. The user can see their collected points, level progression, and how many points they need to level up. This screen is accessible from the main menu and also from the ”workout completed” screen.

4.3 Model

The model part of the application is represented by the database and classes used to communicate with the database. These classes will be explained further in this section.

4.3.1 Database

To store the workouts and exercises the application needs a database. This application uses a local Room database to store the workouts, exercises, and profile information. The code for the database consists of three types of classes.

- Database class
- Entity classes
- DAO classes

The database class handles the initialisation of the database itself and prepopulates itself upon initialisation. Since there should only ever exist one instance of the database it uses the singleton design pattern. This means the database class has a ”getInstance” function that returns the database instance (see code in listing 4.1).

```
1 getInstance()
2     if database exists
3         return database
4     else
5         create new database
6         return database
```

Listing 4.1: getInstance pseudocode.

Entities & Data Access Objects

The data in the database is stored in Entities containing different variables. The application uses three different Entities (see figure 4.3). For a more detailed explanation of the entities see appendix A.

The data from the entities can be accessed by using their respective “Data Access Objects” (DAOs). These DAOs are classes wherein the database interactions are defined (see code in listing 4.2).

```
1 @Query("SELECT * FROM workouts")
2 fun getAll(): LiveData<Array<Workout>>
```

Listing 4.2: Database interaction sample code.

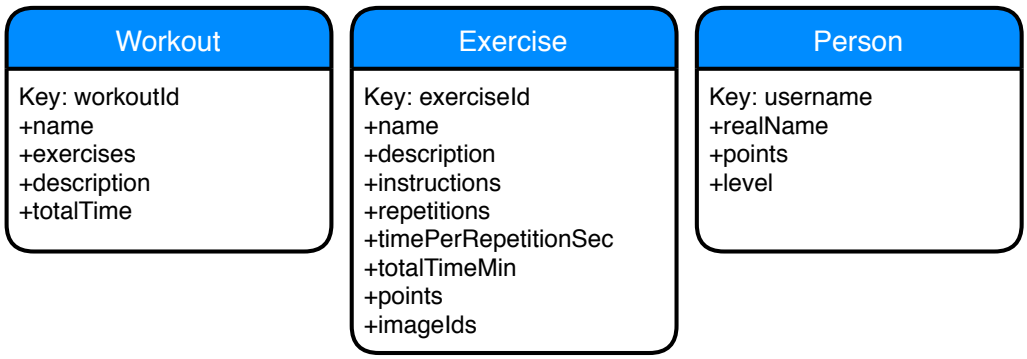

Figure 4.3: The entities contained in the database.

The database interactions are defined by the SQL query on line 1; in this case you send a query to the database to get everything (SELECT *) from (FROM) the table workouts (workouts). This retrieves all the entries from the ”workouts” table in the database. Line 2 defines the function that can be called from the application to run the aforementioned SQL query. The function takes no parameters and returns data in the form of a LiveData object containing an Array of all the Workouts in the database.

Repositories

To separate the database from the rest of the application and to make it easier to add new data sources (API's or remote databases) in the future, a “Repository Pattern” is used. The Repositories consist of one repository-class per Entity, and are used to call the respective DAOs functions to retrieve information from the database. The repositories use two main methods of providing data to the application.

- LiveData.
- Suspend functions.

To retrieve data from the database using LiveData, a LiveData object is created in the repository. This LiveData object will be updated automatically when the database changes. LiveData is used in the application when retrieving information that will be used in a View.

The other method used to retrieve data from the database is suspend functions. These functions are called from a separate thread so that they run in the background. This ensures that the application does not stop when it is accessing the database.

Pre-populating the database

The workouts, exercises, and profile information are inserted in the database upon the first launch of the application. This information is imported using a helper class ”DatabasePopulator”. This class contains functions that reads the included json files and converts the text to objects. These functions returns a list of objects of the specified type. The functions are called in the Database class using a separate thread and the lists of objects that they return are then inserted in the database using the objects respective DAO's.

4.4 ViewModel

The ViewModel acts as an intermediary between the Model and the View. A ViewModel has two different ways to expose data to the View related to it. It could either retrieve data from the Model through a suspend function or LiveData. When you want to retrieve data from a suspend function, a Coroutine is used.

4.4.1 Coroutines

Coroutines is a library found in Kotlin, which allows you to asynchronously call suspend functions. Coroutines are used in every ViewModel of the application, either when initialising the ViewModel with data from the database or to update an entity. When the initialisation of the ViewModel requires something from the database to proceed, the main thread has to be blocked and wait for the data to be retrieved from the database (see code in listing 4.3).

```
runBlocking {
    //Launch background thread
    val getPersonJob = scope.launch(Dispatchers.IO) {
        //Get the person
        person = personRepo.getPerson("john_doe")
    }
    //Wait for the background thread to finish
    getPersonJob.join()
}
```

Listing 4.3: Asynchronous call of suspended function.

RunBlocking is a Coroutine function that blocks the current thread by creating a new coroutine that runs until it has been completed. This allows us to block the main thread until our background thread has retrieved the necessary data through the repository.

However when communication with the database is not time-critical, the main thread should not be blocked. This is the case when you want to update the database with new data (see code in listing 4.4).

```
fun updatePoints(points : Int){
    //Launch background thread
    scope.launch(Dispatchers.IO) {
        personRepo.addPoints("john_doe", points)
    }
}
```

Listing 4.4: Updating the total points of an user.

4.4.2 LiveData

LiveData is a part of Android Jetpack, and is used to retrieve data from the database that is not time-critical. The ViewModel uses LiveData from the repository to expose data from the Model. You are not able to write to a LiveData object, it only changes when the its source is updated.

```
val currentPoints: LiveData<Int> = personRepo.getPoints("
    john_doe")
```

Listing 4.5: LiveData exposing the current amount of points gathered.

Listing 4.5 shows a variable that contains LiveData representing the user's current number of points. Whenever the function from Listing 4.4 is called, the LiveData in currentPoints will be updated and the View observing it will be notified.

For data to be observable it does not have to come from the Model. When you want the View to observe data directly from the ViewModel, MutableLiveData is used. Compared to LiveData, MutableLiveData is mutable and allows us to write directly to it, to make changes. This means that we can update the MutableLiveData object directly from the ViewModel.

```
1 private val privateCurrentExercise = MutableLiveData<
    Exercise>().apply{ value = null}
2 val currentExercise : LiveData<Exercise>
3     get() = privateCurrentExercise
```

Listing 4.6: MutableLiveData representing the current exercise.

Listing 4.6 shows how a MutableLiveData object is set up. The data from the MutableLiveData object is of the type "Exercise" and the data in it is set to null initially, until it is set to a value in the code (see code in Listing 4.7).

```
1 privateCurrentExercise.value = Instance of Exercise
```

Listing 4.7: Pseudocode of assigning a value to the MutableLiveData.

The object is private to ensure that only the ViewModel can update it. To allow a View to observe the data, a LiveData variable is used. This variable contains the data from the MutableLiveData object and is automatically updated when the MutableLiveData is updated.

4.5 View

The view takes care of everything that is related to UI and interactions from the user. The window that the user can interact with is handled by the Activity class, which is an Android class that controls what is displayed on the screen. The application's UI uses a "Single Activity architecture" to control the UI. This means that the application only has one Activity that is always active. Inside it there are Fragments which can be swapped in and out depending on what you want to display.

4.5.1 Navigation

The Navigation class controls the exchange of Fragments in the Activity. Navigation is a part of Android Jetpack and allows you to design how the user navigates through the application. The Navigation component has three key parts that make the navigation of the application work [16]:

Nav graph — XML resource that gathers all information related to the navigation in one place. Inside it there are destinations which are the Fragments you can navigate to and what paths you can take to the Fragments.

NavHost — Is a container used for displaying destinations.

NavController — Controls the navigation through the application and what Fragment that is meant to be displayed in the container of the NavHost.

An advantage of using Android Studio is their graphical tool that allows you to easily draw lines between Fragments that symbolises how you can navigate throughout the application (see figure 4.4). The lines are then translated into paths and destinations, then inserted into the XML code.

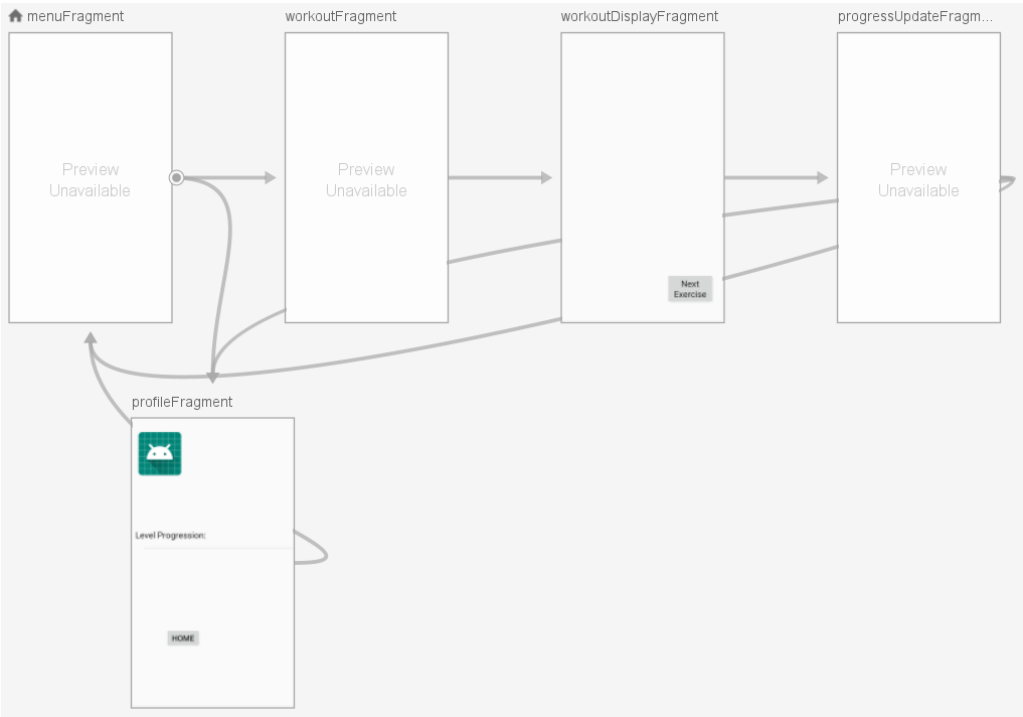

Figure 4.4: Picture of Nav graph displaying the fragments and the paths to them.

4.5.2 Retrieve data from ViewModel

The View only handles the UI and needs the ViewModel to handle the business logic for it. Each Fragment has its own ViewModel and whenever you navigate to a Fragment, a reference to its ViewModel is retrieved. When an user input requires the Fragment to update something in the business logic or the database, the Fragment will send the input to the ViewModel and the change is made (bottom arrow in figure 4.5). However, since the ViewModel has no knowledge about the Fragment itself, it can not forward any changes to the business logic or database directly back to the Fragment that sent the input.

There are two ways for the Fragment to retrieve data from the ViewModel. Either by a simple getter function or by observing LiveData inside the referenced ViewModel (upper arrow in figure 4.5). LiveData is used when you want the Fragment to be notified when the state of an object has changed in the ViewModel. This means that the Fragment can react directly to changes and make the necessary updates to the UI. A getter is used when the Fragment does not need updated data, but only need to do some sort of calculation.

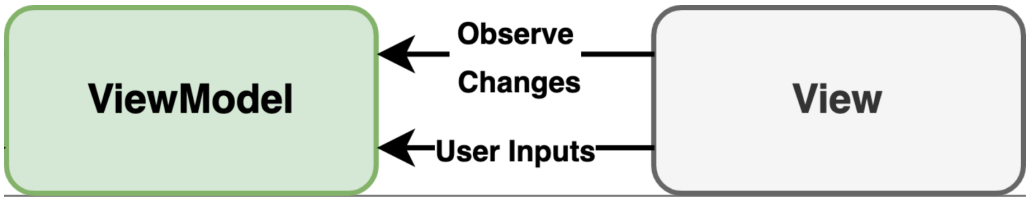

Figure 4.5: Interaction between View and ViewModel.

4.5.3 Data Binding

The UI is represented by an XML resource file. Usually you would have to write a lot of boilerplate code in the Fragment to change a text or the function of a button in the UI. For example, if the UI has some text in it that you want to update with some data from the ViewModel, the usual way of doing that would look something like the code in listing 4.8.

```
1 findViewById<TextView>(R.id.some_text).apply {
2     text = viewModel.updatedText
3 }
```

Listing 4.8: Setting TextView without Data Binding sample code.

If you have more than one or two components in the UI, the Fragment will become cluttered very quickly.

This can be avoided by using Data Binding, which is a part of Android Jetpack. As the name suggests, data binding binds a data source to a component of the UI. Instead of writing something similar to listing 4.8 for every component, we declare a variable in the XML file that represents some object inside the resource file and then use that variable to access the data. Listing 4.9 is the same example as the previous, but we use data binding to bind the TextView to a data source. Row 1 to 4 is the declaration of the viewModel variable. Row 6 to 8 is the represents getting data from the variable "updatedText", found in the viewModel.

```
1 <data>
2     <variable name="viewModel"
3     type="com.application.data.ViewModel" />
4 </data>
5
6 <TextView
7     android:text="@{viewModel.updatedText}"
8 />
```

Listing 4.9: Data Binding example.

In the example above, the data source is a variable from the ViewModel, but it could just as easily be a function from any of our own classes or a library from Android. This is the case throughout the application and means that our Fragments contain fewer lines of code and are kept cleaner.

4.6 Dependency Injection with Koin

Koin is used to handle dependency injection. To use Koin for dependency injection we must create a module, which includes declarations of objects that can be injected. Listing 4.10 shows a declaration of a ViewModel and its parameter. In this case the repository is a singleton object and the ViewModel takes a repository as parameter. The “get()” function automatically chooses the correct parameter to initialise the ViewModel with.

```
1 val exampleModule = module {
2     single{ ExampleRepository(AppDatabase.getInstance(
    androidContext()).ExampleDao()) }
3     viewModel { ExampleViewModel(get()) }
4 }
```

Listing 4.10: Koin module sample code.

This means that all the classes that needs a ViewModel uses the Koin module to retrieve them. Instead of initialising them directly. As mentioned in the Theory chapter this removes the dependency between classes. Listing 4.11 shows an example of how a Fragment uses Koin to inject a ViewModel.

```
1 class ExampleFragment : Fragment() {
2     private val exampleViewModel: ExampleViewModel by
    viewModel()
3     ...
4 }
```

Listing 4.11: Koin ViewModel inject sample code.

Koin is also used to inject the Level class. Since Level is not a ViewModel this injection is done slightly differently. Listing 4.12 shows how the Level class is injected. The difference between how the Level and the Repositories are initialised is that the repositories use the "single" function, this means that there will only exist one instance of the repository. The "factory" function gives a new instance every time it is called.

```
1 //Code from the Koin module
2 val exampleModule = module {
3     factory{ Level() }
4 }
5
6 //Code used in class that injects Level
7 private val levels : Level by inject()
```

Listing 4.12: Koin Level inject sample code.

5

Results

The project resulted in a functioning application for the Volvo Trucks' Android based infotainment system. The application features gamification aspects selected from gamification and motivation research.

5.1 Application

The user flow illustrated in section 3.2 (figure 3.1) have been implemented. The different steps illustrated in the user flow are represented by the following screens in the application.

See the main menu

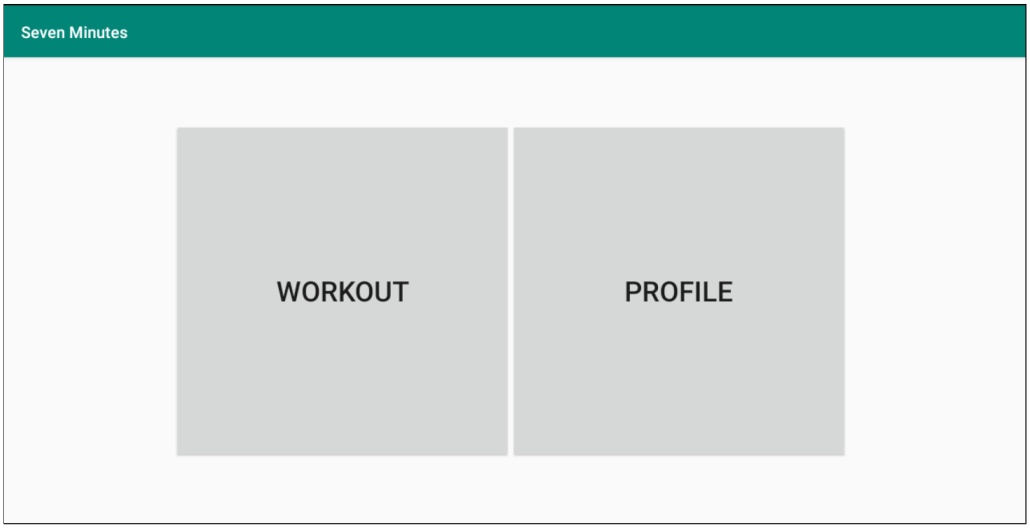

Figure 5.1: Main menu screen.

See the available workouts & Select a suitable workout

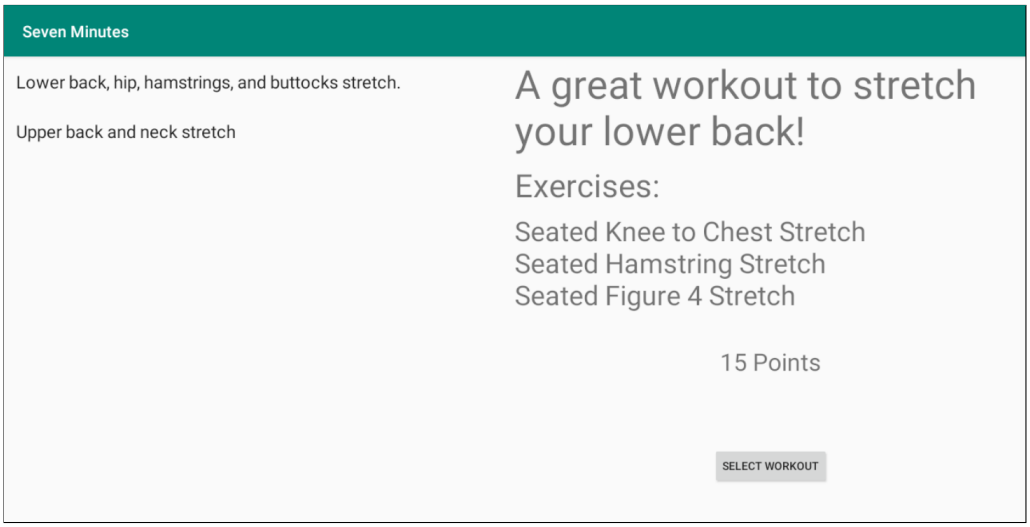

Figure 5.2: Select workout screen.

Perform the workout

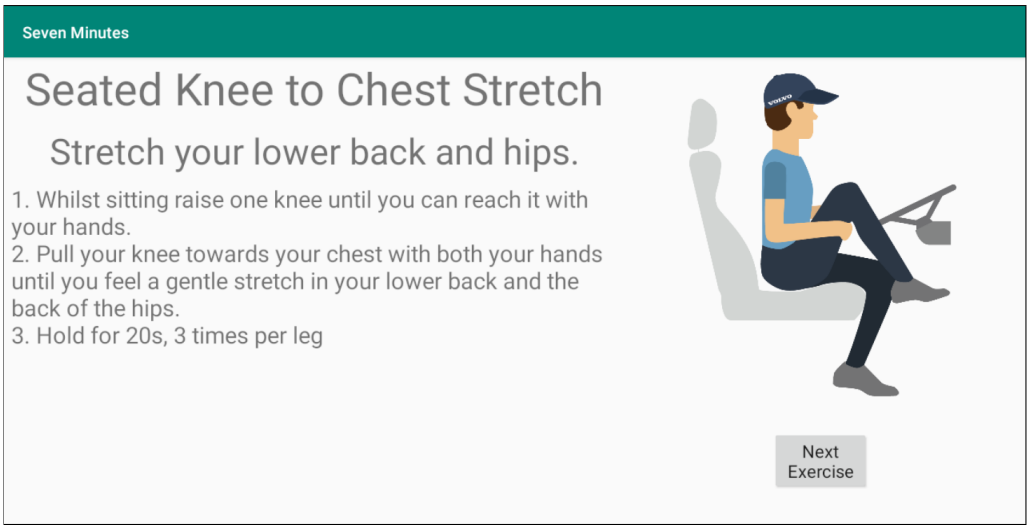

Figure 5.3: Workout session screen.

See level progression

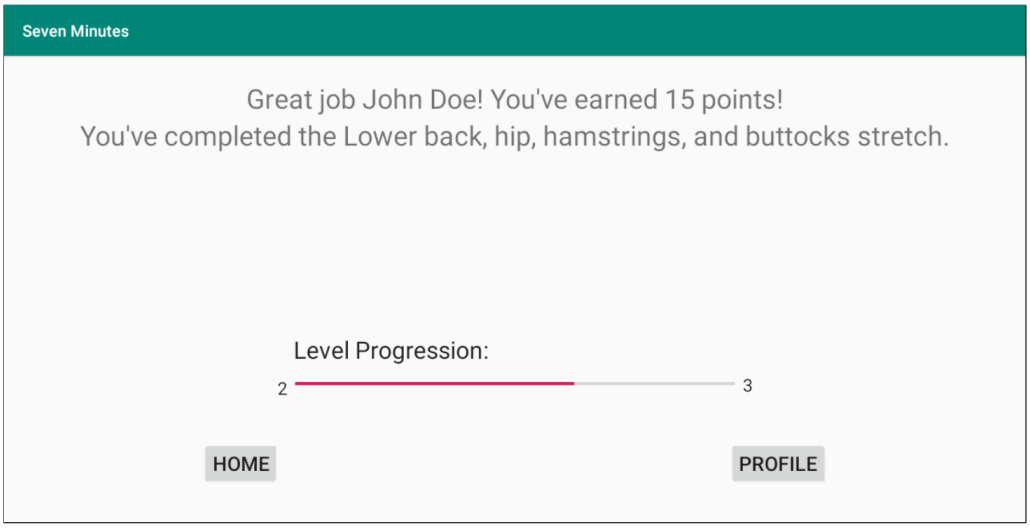

Figure 5.4: Progression screen.

See profile with level progression

Figure 5.5: Profile screen.

There were some goals set up for the development of the application. The main goal was to make it possible for Volvo Trucks to continue the development of the application. This meant that we had to follow Volvo Trucks' and Google's guidelines for developing an Android application. We believe this has been achieved by following the MVVM design pattern, using Koin for dependency injection, and following the Android Jetpack guidelines. However, we have not been able to confirm this with the developers at Volvo Trucks.

A lot of time was spent on developing the database of the application. Since we wanted the application to be easily expandable we wanted to implement a database that could be swapped out without any changes to the application. This was achieved by using a repository to communicate with the database. This means that, in theory, the database can be switched out to a remote database without any changes to the application. We also made sure that exercises and workouts could easily be inserted into the application. This was achieved by using Json files to pre-populate the database, so that to add exercises you can just add them to the json file and rebuild the database and application.

The goals that were set up for the workouts and exercises were also reached. The main goal was to implement exercises in a modular way and have each exercise be accompanied with a text and an animation. This has been partially achieved. The exercises are implemented in a modular way as explained above, and every exercise is accompanied with a text that explains it. However, instead of animations, there are two cycling images explaining the exercise.

5.1.1 Gamification

The application is meant to increase the user's motivation for physical activity. This is done by implementing two gamification aspects, points and levels, explained in section 4.2. These gamification aspects are the same that were presented in section 1.2. Since we successfully implemented these gamification aspects we have reached our goals regarding gamification. There have been no tests done to confirm the effectiveness of the application. The gamification aspects have been chosen based on our research of gamification and motivation. Investigating gamification was also a part of the scope, this was done, and as mentioned the findings was used to choose relevant gamification aspects to implement.

6

Conclusion

6.1 Discussion

As a whole we are very pleased with how the project turned out. We were introduced to a lot of new techniques and patterns, and learning them was very time consuming. But in the end we developed an application we are proud of. We put a lot of thought and effort into ensuring that the application followed Volvo Trucks' guidelines. This led to the application having a solid ground to stand on and allows it to be further developed. However, this was at the cost of only having a basic UI.

At an early stage of the project we had a lot of ideas on what gamification aspects to implement. But once we started to actually plan the milestones of the project, we realised that the amount of features would need to be kept to a minimum. However, we did accomplish what we set as our scope and we are happy with the features we decided to implement. We feel like there are a lot of new features that can be built on top of the current ones.

At the end of the project, all the effort we put in to ensure that the application followed the guidelines paid off. At the last week of development, Volvo Trucks handed us the images for the exercises, which meant we had to put them in the application with little time left. Normally this would not be the easiest thing to do, since each exercise had to be paired with two pictures that had to be cycled. But since we had put so much time and effort into learning the components of Android Jetpack, this took no more than an hour.

An issue we have discovered with the application, is that there is no way to confirm if the user has actually completed a workout. This means that a user could very easily open a workout and finish it without actually performing it. Since the application has a points and level system, we think this could be exploited by users. This would especially become a problem if there were ways to compare yourself with other users inside the application. This issue would need to be solved before the application is installed in Volvo Trucks' infotainment systems.

6.2 Further development

The application is built with the possibility of further development in mind. That is why the majority of the time was spent on building a solid ground to make further development easier. There are many ways the application could be improved, and in this section we will discuss our thoughts on some aspects that can be improved.

6.2.1 Solution to potential exploit

A potential solution to the exploit that we identified in the discussion, would be to use the accelerometer found in phones. The idea is that the user would connect their phone to the infotainment system and hold the phone in their hand while performing the workout. The data from the accelerometer could then verify that the user has performed the exercise. A problem with this solution, is that the user could shake their phone to trick the application into believing they are performing the exercise.

6.2.2 User Interface

The user interface is currently very basic as it has not been the focus of the project. There are many changes that could be made to it to improve the feel of the application. The most noticeable UI weakness according to us is the lack of consistency. To have the entire application follow the same style guide would improve the look of the application and make it feel more complete.

6.2.3 Gamification aspects

When deciding on what gamification aspects to implement in the application, the possibility for further development was kept in mind.

When levelling up, new features could be unlocked. This would make levelling up feel more rewarding. A new feature could be allowing the user to create their own workouts. This would fall under the "Autonomy" category of the SDT, which is lacking in the application right now.

Achievements could be implemented. For example, an achievement could be to do a certain workout three days in a row. When you complete an achievement, a badge would be added to your collection and you could display it in your profile. This would be categorised as "Competence" according to the SDT. Achievements would be a way to feel competent without having to compete with other people.

A leaderboard would be a good addition to the application. The leaderboard would be based on the players total points. There could be multiple leaderboards, for example one global leaderboard and an exclusive one for a truck fleet. A leaderboard would fall under "Competence" and compared to Achievements, it would be more attractive to competitive users that want to compare themselves with others.

A social feed could be implemented. The social feed would show you what people in your truck fleet have been up to. For example, if you complete an achievement you could share it to a social feed and other users could interact with you. This would be categorised as "Relatedness" according to the SDT. Which is something that is lacking in the application right now.

6.2.4 Exercises

The exercises implemented in the application are very basic and have been selected solely because they are possible to perform whilst sitting down. The application would benefit greatly from additions of both exercises and workouts. The addition of more exercises and workouts would make the application feel more complete and give the user a broader selection of workouts to choose from. This would give the user a greater feeling of autonomy which in theory would increase motivation as mentioned in chapter 2.

6.2.5 Implement a remote database

The implementation of a remote database would benefit the application, as it would simplify the addition of workouts and exercises. The workouts and exercises would be uploaded to the remote database. The users would then automatically get the available workouts and exercises from the database. As mentioned in section 4.3.1, the application uses a repository to retrieve information from the database, which in theory would make it possible to implement a remote database without changing the application.

Bibliography

[1] N. Bucciarelli Pedersen and L. Eisenberg, "If sitting is the new smoking, what does this mean for employers? A look at potential workers' compensation claims in the sedentary workplace", *Lewis & Clark Law Review*, vol. 22, no. 3, pp. 965–990, 2018.

[2] Transportstyrelsen, *Regler om kör- och vilotider*. [Online]. Available: https://transportstyrelsen.se/sv/vagtrafik/Yrkestrafik/Kor--och-vilotider/regler-om-kor--och-vilotider/.

[3] K. Werbach and D. Hunter, *For the win: How game thinking can revolutionize your business*. Wharton Digital Press, 2012.

[4] G. Reynolds, *A little exercise might bring cheer*, English, University of Michigan; Copyright New York Times Company May 8, 2018; Last updated - 2018-11-12, May 2018. [Online]. Available: http://proxy.lib.chalmers.se/login?url=https://search-proquest-com.proxy.lib.chalmers.se/docview/2035556507?accountid=10041.

[5] Kotlin Team, *Kotlin FAQ*, [Online; accessed 3-May-2019]. [Online]. Available: https://kotlinlang.org/docs/reference/faq.html.

[6] ——, *Comparison to Java programming language*, [Online; accessed 3-May-2019]. [Online]. Available: https://kotlinlang.org/docs/reference/comparison-to-java.html.

[7] ——, *Nullable types and non-null types*, [Online; accessed 3-May-2019]. [Online]. Available: https://kotlinlang.org/docs/reference/null-safety.html.

[8] J. Gossman, *Introduction to model/view/viewmodel pattern for building WPF apps*, [Online; accessed 25-April-2019]. [Online]. Available: https://blogs.msdn.microsoft.com/johngossman/2005/10/08/introduction-to-modelviewviewmodel-pattern-for-building-wpf-apps/.

[9] Microsoft Developers, *The MVVM pattern*, [Online; accessed 25-April-2019]. [Online]. Available: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/hh848246(v=pandp.10).

[10] Android Developers, *Android Jetpack*, [Online; accessed 25-April-2019]. [Online]. Available: https://developer.android.com/jetpack.

[11] C. Sells, B. Poiesz, and K. Ng, *Use Android Jetpack to accelerate your app development*,

[Online; accessed 3-May-2019]. [Online]. Available: https://android-developers.googleblog.com/2018/05/use-android-jetpack-to-accelerate-your.html.

[12] Android Developers, *Fragments*, [Online; accessed 3-May-2019]. [Online]. Available: https://developer.android.com/guide/components/fragments.

[13] ——, *Room persistence library*, [Online; accessed 25-April-2019]. [Online]. Available: https://developer.android.com/topic/libraries/architecture/room.

[14] T. Janssen, *Design patterns explained – dependency injection with code examples*, [Online; accessed 25-April-2019]. [Online]. Available: https://stackify.com/dependency-injection/.

[15] Koin Developers, *Koin webpage*, [Online; accessed 4-May-2019]. [Online]. Available: https://insert-koin.io/.

[16] Android Developers, *Android navigation*, [Online; accessed 29-May-2019]. [Online]. Available: https://developer.android.com/guide/navigation/.

A

Database Entities Explanation

A.1 Workout

workoutId The id of the workout.

name The name of the workout.

exercises A list of id's for the exercises contained in the workout.

description A description of the workout.

totalTime The estimated total time of the workout.

A.2 Exercise

exerciseId The id of the exercise.

name The name of the exercise.

instructions A set of instructions on how to perform the exercise.

repetitions The number of repetitions the exercise should be performed.

timePerRepetitionsSec The approximated time a repetition takes, in seconds.

totalTimeMin The approximated time the exercise takes, in minutes.

points The amount of points the user receives upon completion of the exercise.

imageIds A list of Strings representing the file names of the images corresponding to this exercise.

A.3 Person

username The username of the user.

realName The real name of the user.

points The amount of points the user has collected.

level The level the user has reached.