LiftMate – Documentation v 1.0



Marco Teran Gutmanowitz – SUM200

# The App Idea

An application for logging workouts with a focus on simplicity, app performance and user usability. The app targets the gym-goer, a person who goes to the gym to perform exercise activities.

# Why?

Applications today are more complex than ever before. Fancy User Interface-effects, the ability share your latest Duolingo score with an unnecessary social media function, you get the picture. Many applications today are overloaded with unnecessary functions that often are left unused.

# How?

The workout application that will be described in this document will lay a heavy focus on the following over-arching criterias:

* Simplicity i.e no cloud databases, no unnecessary functions or effects. Saving of data will be done through a local SQLite database.
* App performance – the app should start quickly and responds with minimal delay for the user
* Small storage space footprint – the app should take up as little storage space as possible
* A focus on delivering the important features in an app in a resource saving, simple way without removing from the functionality of the app.

**More Concretely put:**

* Use of a SQLite database to store/retrieve data
* Usage of algorithms that are as efficient as possible
* A “no frills”-methodology to the app
* An API to display motivational quotes on the main page of the app
* A Calorie Tracker function that utilizes the accelerometer of the smartphone in order to calculate calories burned.

# User Stories

* As a **fitness minded person** I want to track my progression in the gym in order to keep track of my workouts and exercises
* As an **athlete** I want to log the amount of weight I lift each exercise
* As a **jogger** I want to track how many calories I burn each jogging session
* As a **person with an old smartphone** i want an exercise app that doesn’t affect the performance of my smartphone
* As a **user with varying types of workouts**, I want to be able to create profiles that track my progress in each and one of my different workout plans.
* As a **weightlifter**, I want to visualize my progress over time through graphs and charts that display my lifting performance, so I can identify trends and areas for improvement.
* As a **fitness** **enthusiast**, I want to create personalized workout routines by selected exercises from a list, specifying sets, reps and other relevant information, so I can follow a structured plan during my gym sessions.

# Flow chart

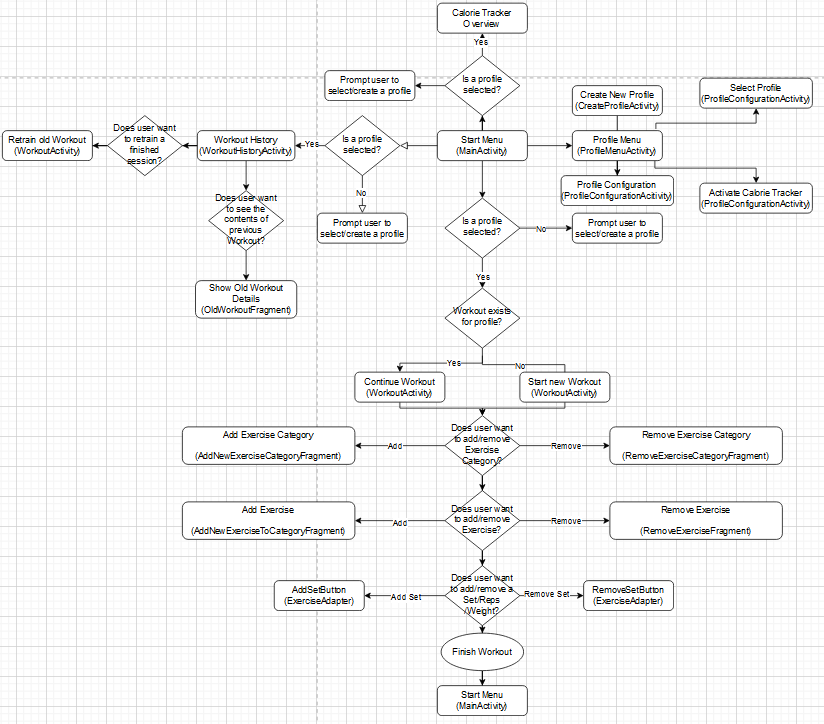


Figure - Flow Chart for LiftMate

# Discussion about navigation Pattern

Considering the purpose of the application, which is to log workouts, it's important to design the navigation in a way that caters to the intermittent and potentially distracted usage patterns of our users. Given this context, a stack-based navigation pattern has been chosen for the app. For this reason, a stack-based navigation pattern has been selected with a primary focus on the user. Users may not always have their full attention on the app, and they may interact with it sporadically during their workouts. This navigation pattern guides users through the app's sequences with simplicity, ensuring that each step is clear and intuitive. It minimizes the risk of user errors and aligns with the user's expectations.

The advantage of stack-based navigation simplifies complex tasks by breaking them down into manageable steps. It ensures a linear and guided user experience. While other navigation patterns exist, such as tab-based or drawer navigation, the stack-based approach was chosen for its suitability in our context. Unlike tab-based navigation, which often requires users to choose between multiple destinations simultaneously, the stack-based pattern provides a linear and guided user experience, which is essential for a that may not always have full attention on the app.

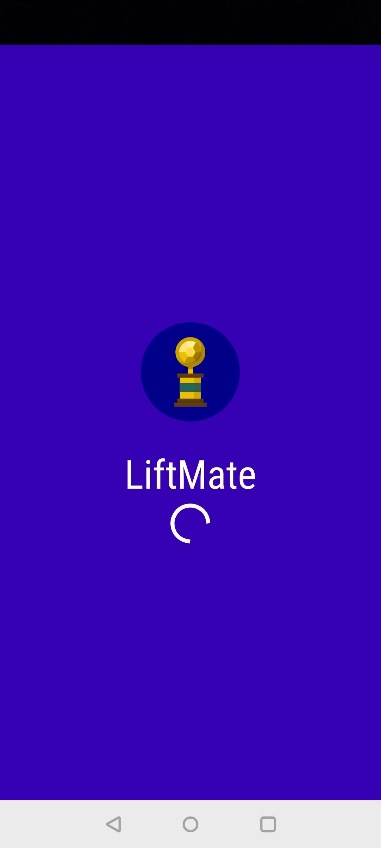
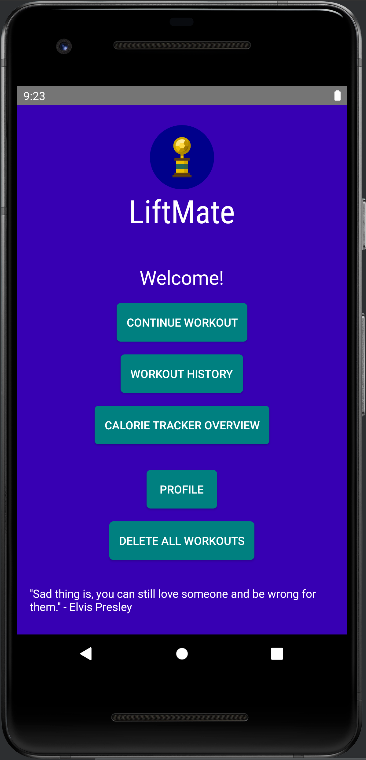
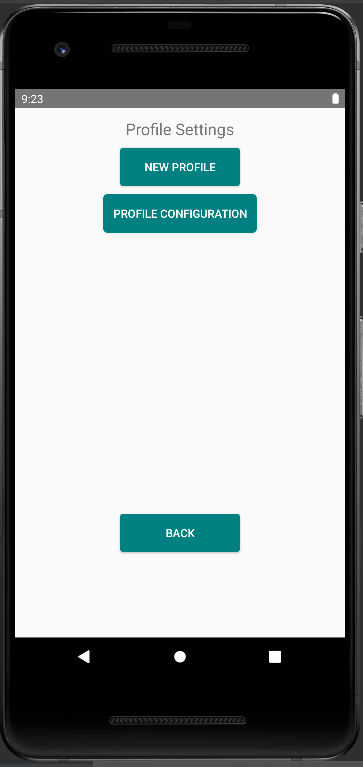
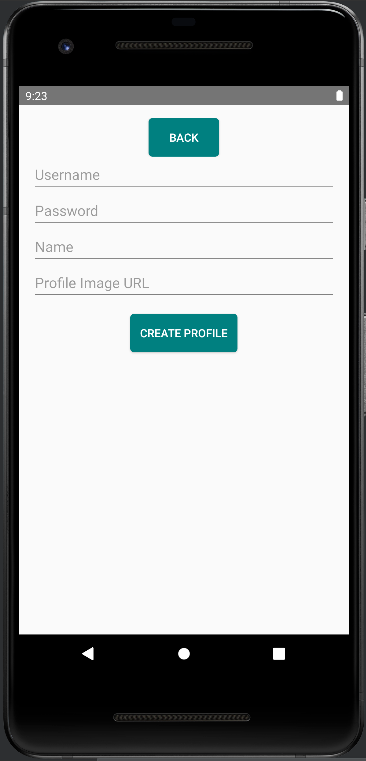
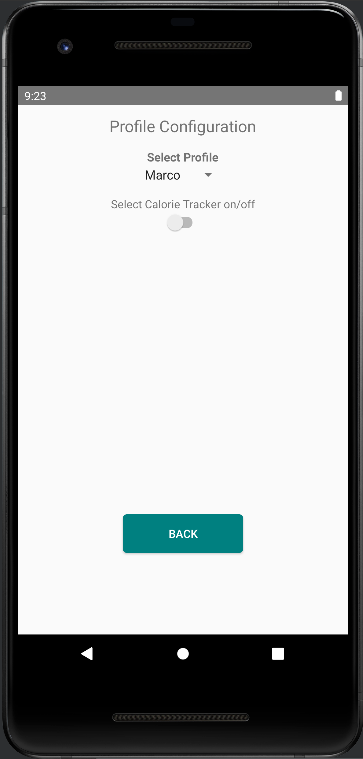
Implementing a stack-based navigation also ensures consistency and predictability across the app. Each sequence of the app will have a button to move forward or backward through a given sequence of screens, at an appropriate place for each screen, in order to not bring confusion as to where the user will be in the application at any point in time, and there should be minimal risk for user error, i.e there should not be any clickables that are positioned in such a way that the user click on the wrong button.

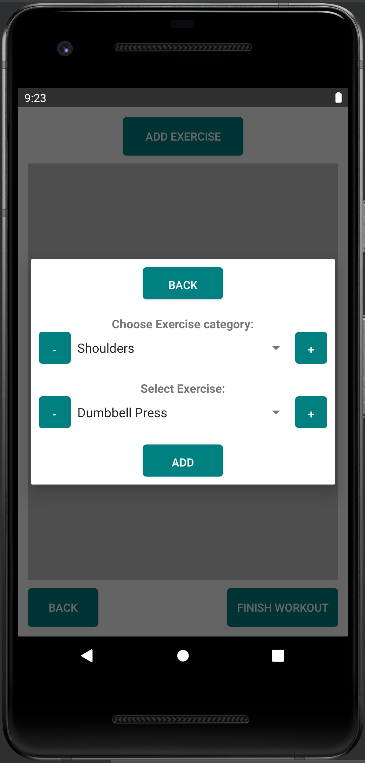
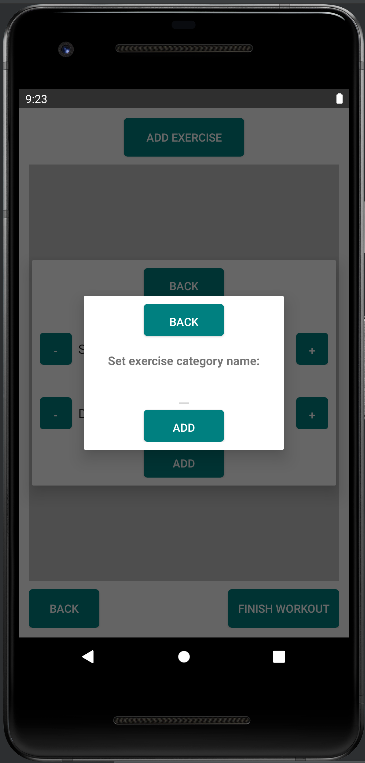
# Graphic Design

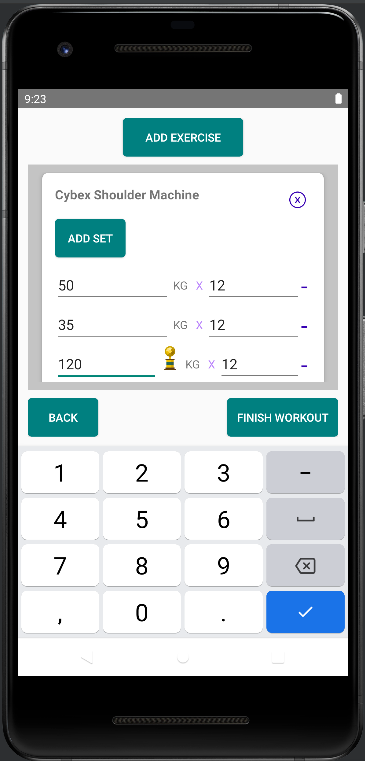
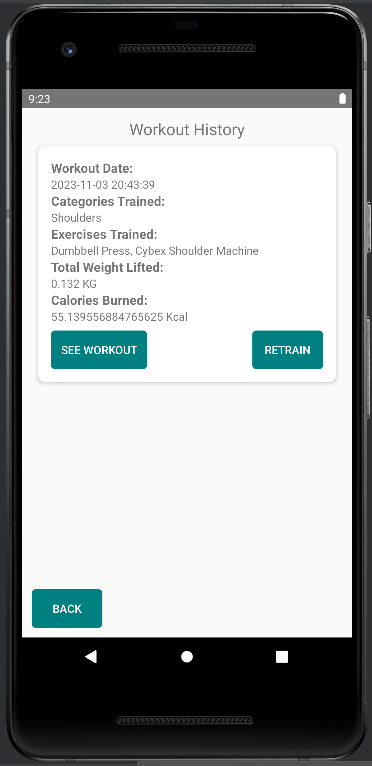
In conformity with the overall theme of simplicity and ease of use, the following colors have been selected as the color palette for the application:

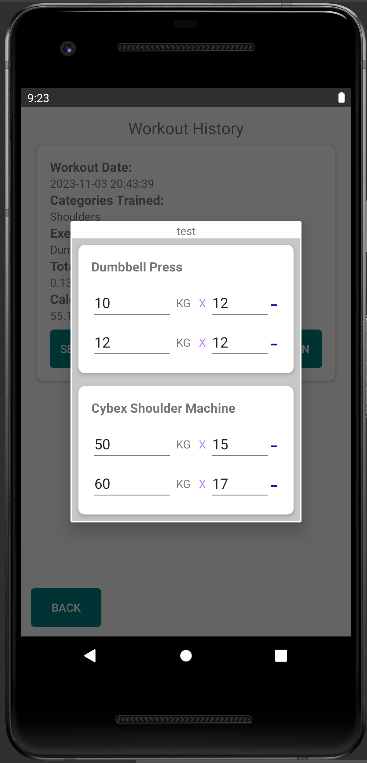
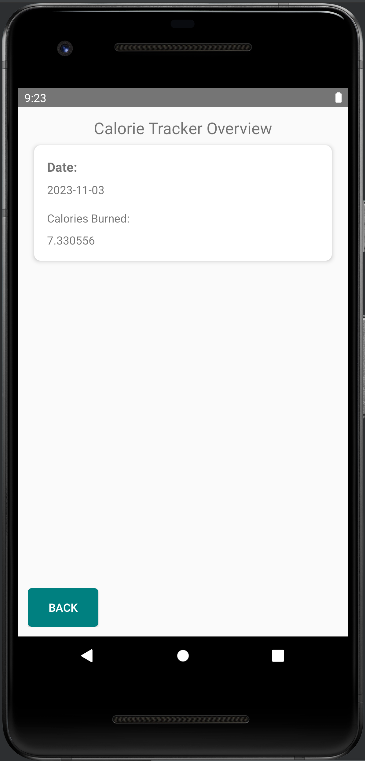
* White (#FFFFFFFF) – background color, text color 2
* Teal (#008080) – button color
* Purple (#FF3700B3) – Main page (MainActivity) background color
* Red (#FF0000) – warning color
* Black (#FF000000) - text color 1

# Application Showcase

# Functionality of the App:

The application allows the user to

* Create a profile
* Select a profile
* Start a workout
* Add and Delete exercise categories (For example Shoulders)
* Add and Delete exercises (For example Dumbbell press in the category Shoulders)
* Add and Remove Sets to an Exercise in an ongoing workout
* Modify the Weight and Reps value of Sets
* Display the exercises in a workout through a RecyclerView.
* See all previously finished workouts in fine detail.
* Enable/disable Calorie Tracker
* See how many calories were burned during the workout.
* Start a new workout with all previous exercises, sets, reps and weight from the old one.
* Get an overview of the calories burned for each day.

## Calorie Tracker

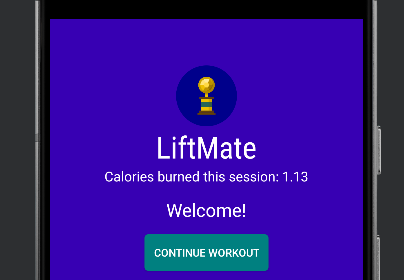
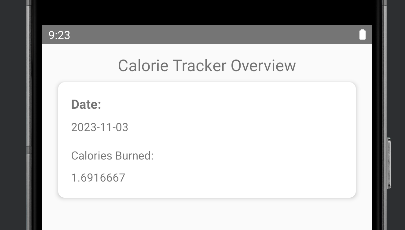
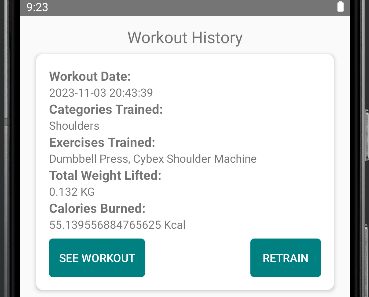
  

Figure - Calorie Tracker Activated

The Calorie Tracker can be activated in the Profile Configuration menu. When activated, the calorie tracker runs in the background, until the user turns it off. The calorie tracker utilizes the Accelerometer of the smartphone in order to calculate the calories burned during running, for example. The tumbling effect that the smartphone is exposed to can be utilized to calculate how many calories the user is burning. The function works by getting sensor data every three seconds, and adding the data to a variable. Every 10 counts or 30 seconds, the data is sent to the database and added into the CaloriesTracker table, and the variable is set to 0 again.

The data stored in the CaloriesTracker table is then utilized in order to calculate calories burned during workouts (Workout History), or during the day (Calorie Tracker Overview).

Another thing to note is that when the Calorie Tracker is activated, a notification channel is created, and a notification is persistent in the notifications window of the smartphone. This is a requirement from Google when creating functions that run in the background of an Android smarthphone.

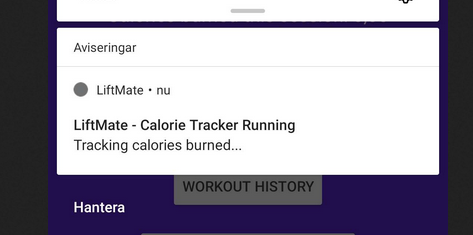


Figure - Notification when CalorieTracker is running

## Trophies

Small functions that contribute to the overall usability and user experience of the app have been added, such as a tiny trophy that appears next to a set if the user score a new personal best for a given exercise.

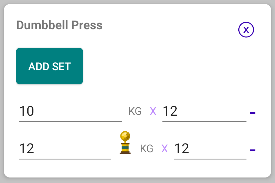


Figure - Trophy displayed next to PR

## DialogFragments

The Usage of DialogFragments has been utilized in order to make the usage of the app quicker. DialogFragments are used when the user wants to add an Exercise. From that point the user may opt to add or remove Exercises and Exercise Categories.

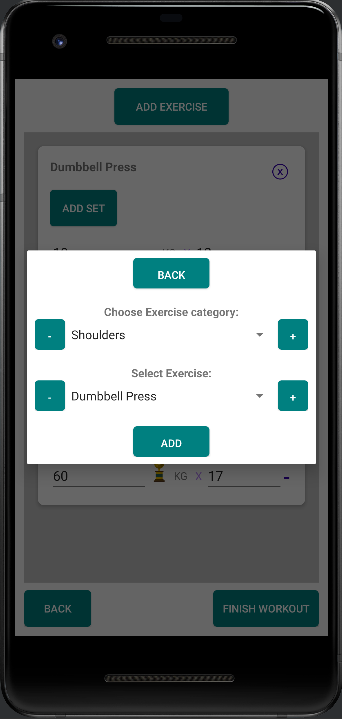
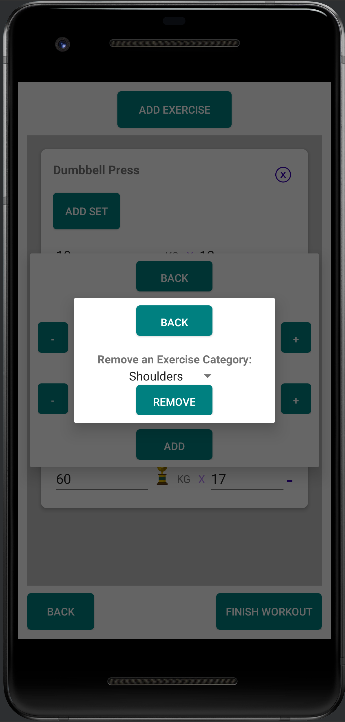
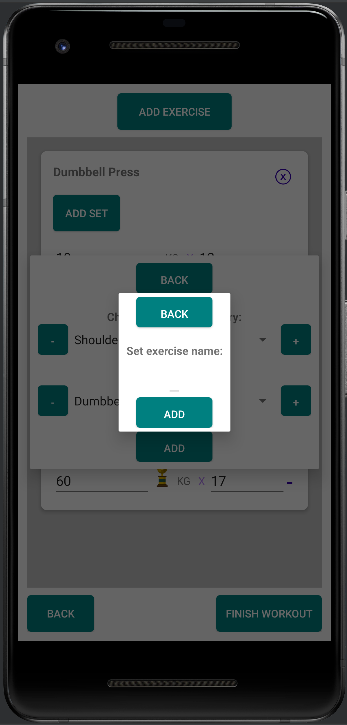
  

Figure - DialogFragment examples

## User Constraints and Exception Handling

Care has been taken to ensure that all functions have adequate exception handling and user constraints. For example, there are whitelists for the symbols allowed for naming Exercises and Exercise Categories. A user may only enter numerals into the weight EditText of an Exercise in the RecyclerView, as shown below.

## 

Figure - User constraints displayed in the workout Recyclerview

Likewise, as many as possible of the crucial methods in the project contain try/catch blocks, or if/else statements to catch the most common errors, in order to avoid the app crashing.

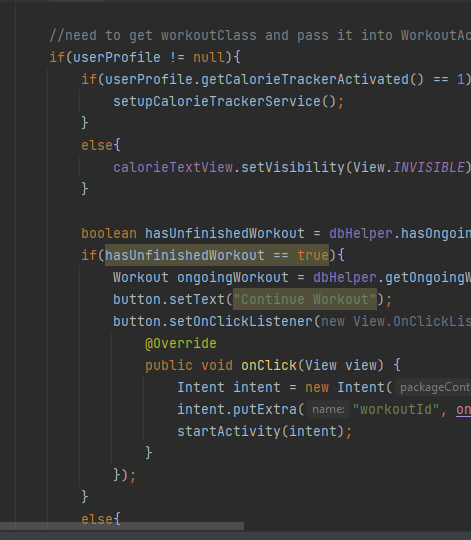


Figure - Exception handling in Code

# App Classes

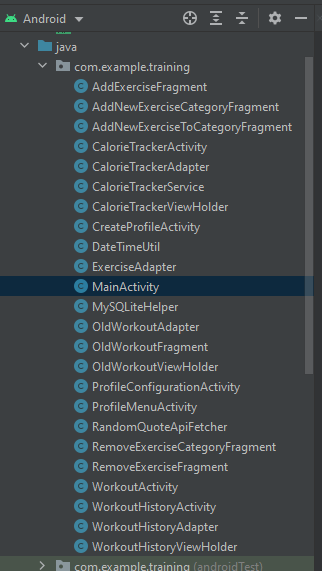


Figure - Classes found in the Liftmate Project

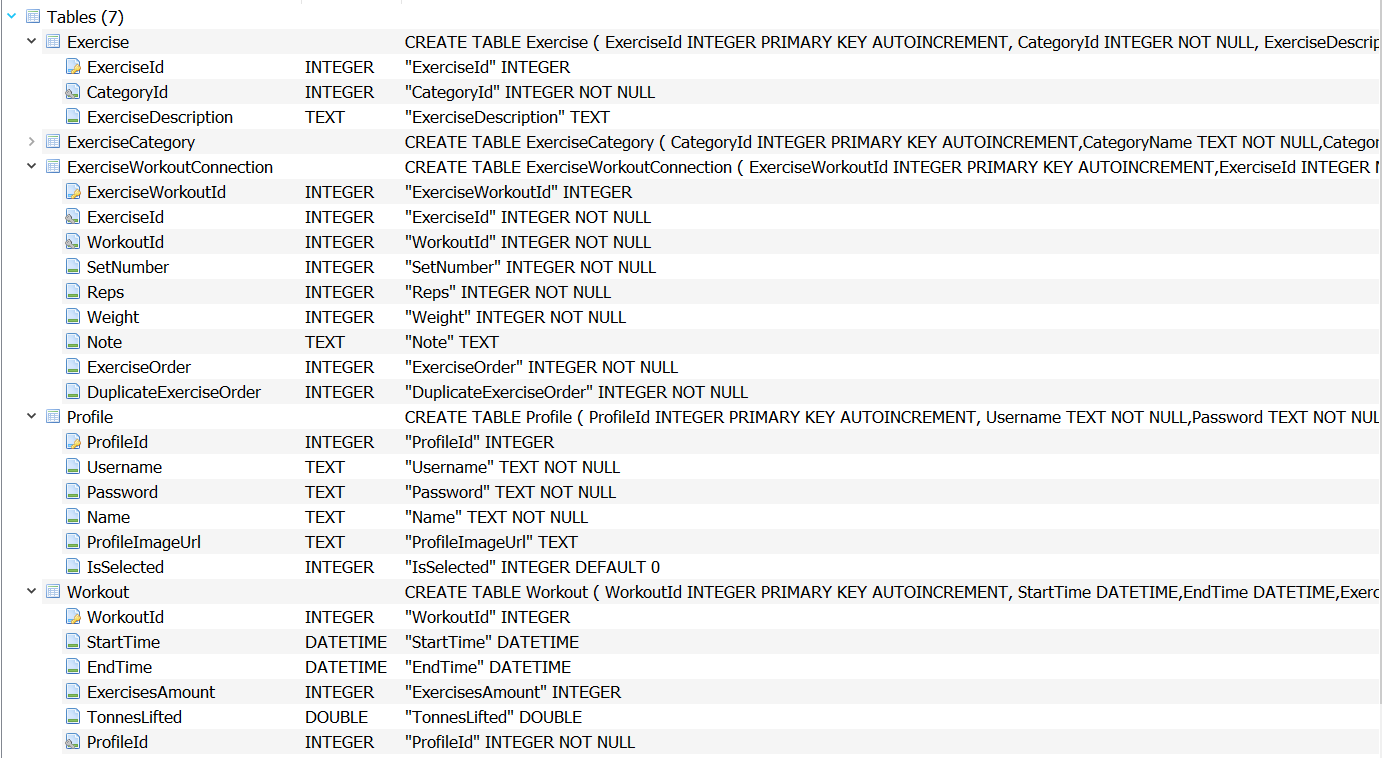
In total, the project consists of 24 separate classes. 7 of these are classes necessary for properly using RecyclerViews, these have the suffix -Adapter and -ViewHolder.

Here is a short overview of the integral classes and what they do:

* MySQLiteHelper
  + Contains the definition of the SQLite Database and all database methods
* MainActivity
  + The main page of the App.
* CalorieTrackerActivity
  + The Calorie Tracker Overview page of the App.
* CalorieTrackerService
  + This contains the logic for calculating the calories burned, if the option is enabled for the profile.
* WorkoutActivity
  + The page that displays an ongoing workout through a RecyclerView
* WorkoutHistoryActivity
  + This page displays all previous finished workouts. Contains logic such as starting a new workout with all exercises, sets and reps from a previous workout. The user can also analyze a previous workout in detail, with the use of the OldWorkoutFragment.
* ExerciseAdapter
  + Contains the logic for the RecyclerView that is present inside the WorkoutActivity.
* RandomQuoteApiFetcher
  + Contains the logic for getting and displaying the motivational quote that is displayed on the MainActivity
* ProfileConfigurationActivity
  + Contains the logic for selecting a profile. It is also here where the user can enable the calorie tracker on/off.
* CreateProfileActivity
  + Self explanatory – the page where one can create a profile.

The rest of the classes in the list are DialogFragment specific classes, which contribute to the functionality of Activity pages, but are not stand-alone.

**Database:**

****