Slutrapport

Erik Bock, Marcus Svensson, Daniel Olsson,
Augustas Eidikis, Alexander Bergsten

2018-10-20
version 0.5

Table of Content

1 Introduction

Gym Companion is an application suited for helping both advanced and beginner users to keep track of their progress in the gym. The application offers a way to use both existing and customized routines and scheduling them through a calendar. Users can view their progress both on-the-go and after the workout through an intuitive statistics page, including an interactive graph and lifetime stats.

1.1 Definitions, acronyms, and abbreviations

- **Routine**: a workout routine, training routine
- **Exercise**: a part of a routine
- **Schedule**: a lists of intended routines and times
- **GUI**: Graphical User Interface (what the user sees and interacts with)
- **MVVM**: Model-View-ViewModel is a design pattern that helps us separate the logic from the user interface.
- **Design Pattern**: A general reusable solution to a commonly occurring software development issue.
- **Law of Demeter**: A design guideline saying that a class should only call methods on objects closely related to itself and avoid method calls through other objects.
- **Model**: Also known as the domain model. Contains all logic
- **View**: A class that is connected to GUI and gets fed information to display
- **ViewModel**: A class that converts data into something that can be shown on a View
- **User Story**: A user story is a short description in common language of what a user wants to achieve.
- **Run-time**: Used to describe the state of the app when it's running
- **Subclass**: The classes under a specified class in a hierarchy
- **Super-class**: The classes above a specified class in a hierarchy
- **JSON**: JavaScript Object Notation, a data format for structuring objects and their properties.
- **Gson**: A Java serialization/deserialization library for parsing JSON-files into objects and vice versa.

2 Requirements

2.1 User Stories

There are in total 16 User Stories for the application. They each have a set priority depending on how important they are for the functionality and purpose of the application.

2.1.1 M01

Story Name Gym Scheduler/Calendar

Description As a very busy gym-goer, I need a quick and easy way to schedule my workout routines so that I have time for other things.

Functional Confirmation

- Can I create a weekly schedule?
- Can I schedule routines?

Non-functional Confirmation

- Can I schedule my routine any time of the day?

Priority 2

2.1.2 M02

Story Name Avoid harmful mistakes

Description As a novice gym-goer, I need help understanding how to perform certain exercises because I dont want to make potentially harmful mistakes.

Functional Confirmation

- Is there a written guide to aid me in understanding?
- Are there video guides/pictures to aid me in understanding?
- Is there a description for how to best avoid mistakes that can be harmful?

Non-functional Confirmation

- Can I access the app 24/7?

Priority 4

2.1.3 M03

Story Name Workout Reminder

Description As a forgetful person, I would like to be reminded when my appointed training time is about to start so that I dont miss it.

Functional Confirmation

- Is there a way to get reminded a while before the workout is scheduled?
- Will I be able to choose when and how I get reminded?

Non-functional Confirmation

- Will the reminders happen no matter the time of day?

Priority 5

2.1.4 M04

Story Name Save completed exercise information

Description As a gym-goer, I need a way of logging my exercises so that my performance is saved.

Functional Confirmation

- When I am done with an exercise will it be stored/saved?
- Will I be able to see what I have completed/not completed?

Non-functional Confirmation

- N/A

Priority 3

2.1.5 M05

Story Name Browse Routines and Exercises

Description As a non-creative gym practitioner I want to be able to browse for routines and exercises so that I can exercise effectively.

Functional Confirmation

- Can I find routines and exercises by name?
- Can I browse for routines or exercises specifically?
- Can I navigate by difficulty?
- Is it possible to sort by muscle group?
- Can I save things I find when I browse?

Non-functional Confirmation

- N/A

Priority 2

2.1.6 M06

Story Name Creating a workout routine

Description As an advanced gym-goer, I want to be able to customize my own workout routine so I can focus on specific body parts.

Functional Confirmation

- Will I be able to create my own workout routine?
- Can I create more than one customizable routine?
- Will I be able to customize the workout routine with different exercises?
- Can I edit an already existing workout routine?

Non-functional Confirmation

- N/A

Priority 1

2.1.7 M07

Story Name Friends

Description Since I often work out with a group of friends, I want to be able to get connected to them through the app so that I can keep in touch.

Functional Confirmation

- Can I add a friend to my contact list?
- Can I remove a friend from my contact list?
- Will I be able to see my friends profile?

Non-functional Confirmation

- Is there a way to confirm another user as a friend?

Priority 5

2.1.8 M08

Account

Description As a user of multiple platforms, I need to be able to view my data on many different devices so that I have the same data everywhere.

Functional Confirmation

- Can I access my information?
- If i log into my account, will I be able to access information from any device I use?
- Will I be able to create a complicated password? (Not only letters/numbers)

Non-functional Confirmation

- Can I access my information any time of the day?
- Will my password be hashed?

Priority 3

2.1.9 M09

Story Name Profile

Description As a gym-goer, I need to be able to store my personal and training information so that I am not forced to enter it every time I start the app.

Functional Confirmation

- Can I later alter or remove previously entered information?
- Will I only have to enter information once?
- Is this information available for use by the app in all applicable scenarios?
- Can I set a profile picture?
- Can I set my experience level?

Non-functional Confirmation

- Can I set my profile to private/public?

Priority 3

2.1.10 M10

Story Name On-the-go edits

Description As an indecisive person, I need to be able to change my current workout routine on the go if I change my mind.

Functional Confirmation

- Can I modify the current routine while it is in progress?
- Can I modify the amount of repetitions and sets of individual exercises?

Non-functional Confirmation

- Can I change it from the beginning of the workout routine until the very end of it?

Priority 3

2.1.11 M11

First time at the gym

Description As a novice gym-goer, I need the app to introduce me to beginner level routines and exercises that fit me.

Functional Confirmation

- Can I get recommended routines for beginners?
- Can I get recommended exercises for beginners?

Non-functional Confirmation

- N/A

Priority 4

2.1.12 M12

Story Name Workout statistics

Description As a gym-goer, I need a way of seeing my development over time so that I can decide how to train.

Functional Confirmation

- Is this information presented to me?
- Are there graphs showing development over time?
- Can I see a history of performed exercises?

Non-functional Confirmation

- N/A

Priority 2

2.1.13 M13

Story Name Share Workout routines

Description As a personal trainer, I want to be able to share custom-made workout routines so that others can access it.

Functional Confirmation

- Will I be able to share my workout routines?
- Will I be able to add additional information like description and muscle group for the workout routine once I share it?

Non-functional Confirmation

- N/A

Priority 5

2.1.14 M14

Story Name Track current workout routine

Description As a gym-goer, I want to keep track of what I am doing during my workout so that I do not forget about anything.

Functional Confirmation

- Can I see what I have done?
- Can I see what I have yet to do?
- Is this displayed to me in a intuitive way?
- Can I leave exercises unfinished but still finish the routine?
- Can I quit if I do not want to finish the routine?

Non-functional Confirmation

- N/A

Priority 1

2.1.15 M15

Story Name Begin and end training

Description As a gym-goer, I need to be able to start a workout routine and get satisfactory finishing results upon completing the workout, so that my performance is documented.

Functional Confirmation

- Can I start a workout routine?
- Is the correct workout routine for the day chosen as default when I start?
- Can I choose which workout routine to perform when starting?
- Can I end a workout routine?

Non-functional Confirmation

- N/A

Priority 1

2.1.16 M16

Story Name Sorting and Filtering

Description As a picky person, I need to be able to sort and filter when I search and browse so that I will easily find things that fit my preference.

Functional Confirmation

- Can I filter my searches?
- Can I filter what I browse?
- Can I sort my search results in specific ways?
- Can I sort what I browse?

Non-functional Confirmation

- N/A

Priority 3

2.2 User interface

There are in total 5 main pages that are atop of their own page-hierarchy. The names of these pages are: **Home**, **Browse/Search**, **Schedule**, **My Routines**, and **Statistics**.

2.2.1 Common elements of all pages

All the pages mentioned above have a common element, the navigation bar. This bar allows the user to move between all the Start Pages that are at the top of every page hierarchy. This element remains constant throughout the app, so that the user can always navigate back to one of the start pages when necessary. The only exception is the **Progress Page**, where the user has to either complete their current routine or exit out of it.

2.2.2 Home Page

Start Page The home page is the main page of the application. This is where the user ends up when first starting the app. The first thing a user will see is a big button that leads to the apps main functionality, tracking a users workout. Other than that, the page simply contains a text that tells the user which routine will be used for the days workout.

Progress Page Pressing the button leads the user to the second page, where they're able to see which exercises they have to do and what they've already done. Here the user can check off exercises as they finish them. They are also able to press a button to finish a workout, and are then sent to the next page.

Finished Page When the user finishes their workout they're sent to the final page of the home page hierarchy. Here they can see their stats from the workout they just finished, for example exercises completed. There is also a back to home button, so that the user can return to the start page.

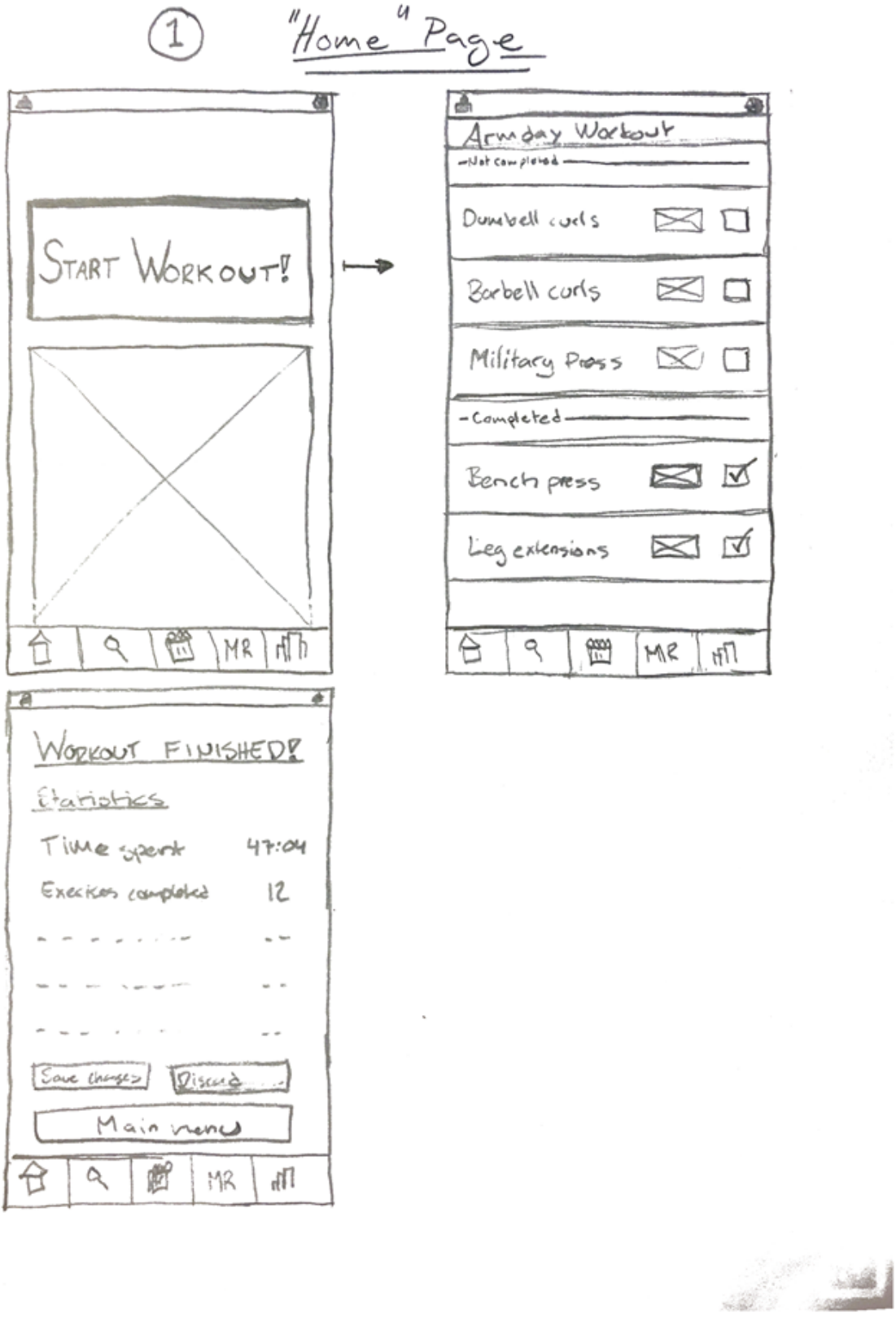

Figure 1: The Home Page Hierarchy

2.2.3 Browse/Search Page

Start Page The first page the user sees when they go to search/browse is a page that offers the users a choice between searching directly or picking categories to browse. Search simply shows all exercises and routines that fits the users search on the result page. If entering an empty string into the search bar, the user will be shown everything. Browse on the other hand has an extra step. First the user picks if they want to filter by muscle groups or recommended filters. Then they are sent to the next page.

Muscle Group Selection Page This page is only shown to the user if they picked muscle groups on the previous page. Here the user is prompted to choose a muscle group, and when they do they are sent to the Result page.

Recommended Selection Page This page is only shown to the user if they picked recommended on the last page. Here the user can pick which recommended filter they want to apply. For example there is a beginner filter that only leaves a handful of easy exercises for the user to browse. When something is selected they are directed to the next page.

Result Page This page shows the user the result of either their search or what the category they have chosen contains. They are also able to sort the results in a variety of ways and the option to see routines, exercises or both is also there. From here the user is able to directly add routines to the ***My Routines Page*** list or add exercises to a new or already existing routine in the ***My Routines Page***.

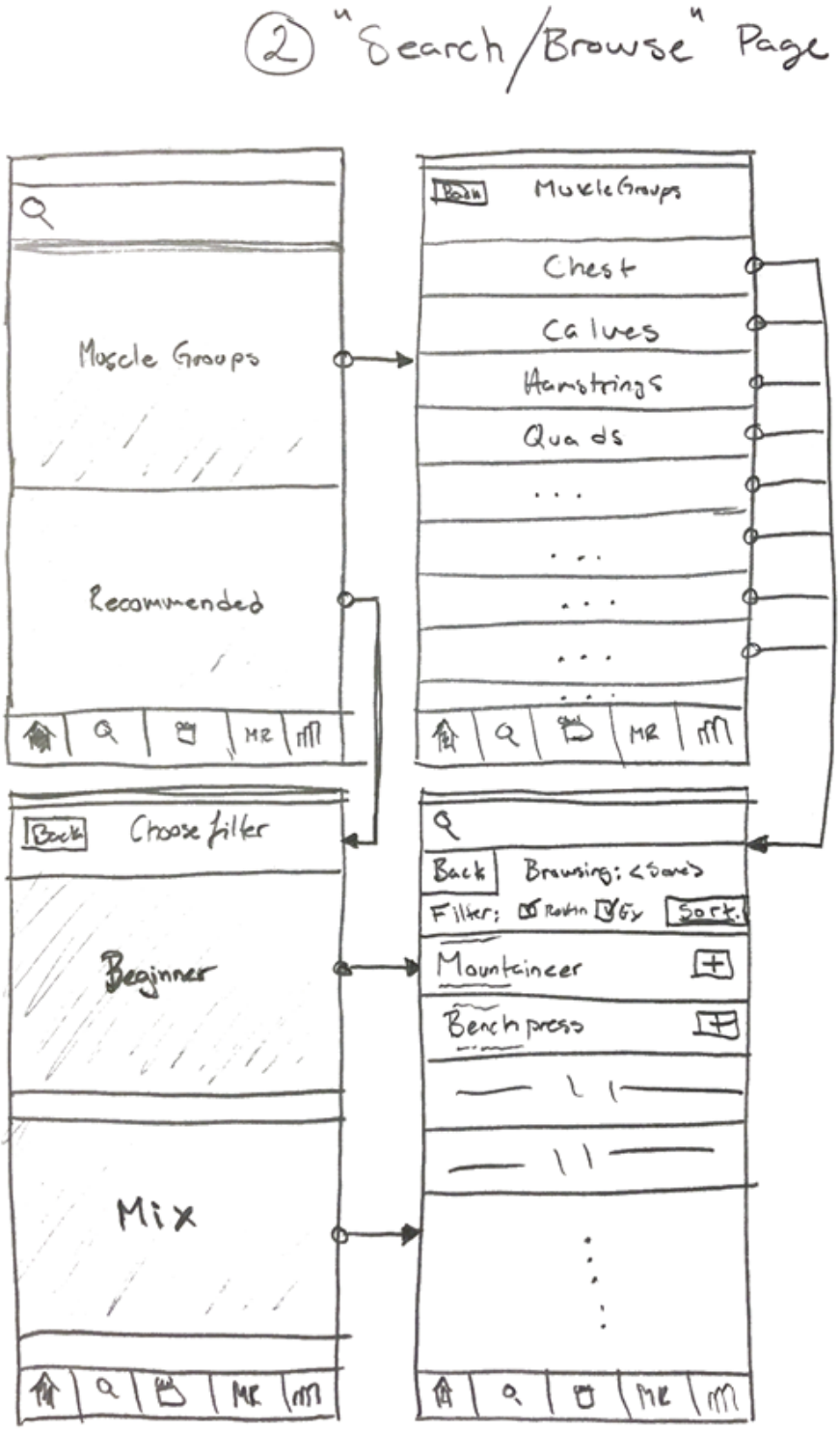

Figure 2: The Browse/Search Page Hierarchy

2.2.4 Schedule Page

Start Page The first page the user sees when navigating to the Schedule page hierarchy displays a calendar. The user is able to schedule routines to specific dates. When a day contains a routine the user will be able to see the name of the routine. When the user wants to add a routine to a certain day, they can select that day and press the Book Routine-button. They are then sent to the page for picking a routine. If the user wishes to change the scheduled routine on a day that is already scheduled it would need to press the Change Routine-button. This action would also lead the user to the Pick Routine page.

Pick Routine Page This page displays a list of all routines in the ***My Routines Page***. The user selects a routine by pressing on it. They will then be directed to the start page of Schedule Page and the selected routine will then be scheduled.

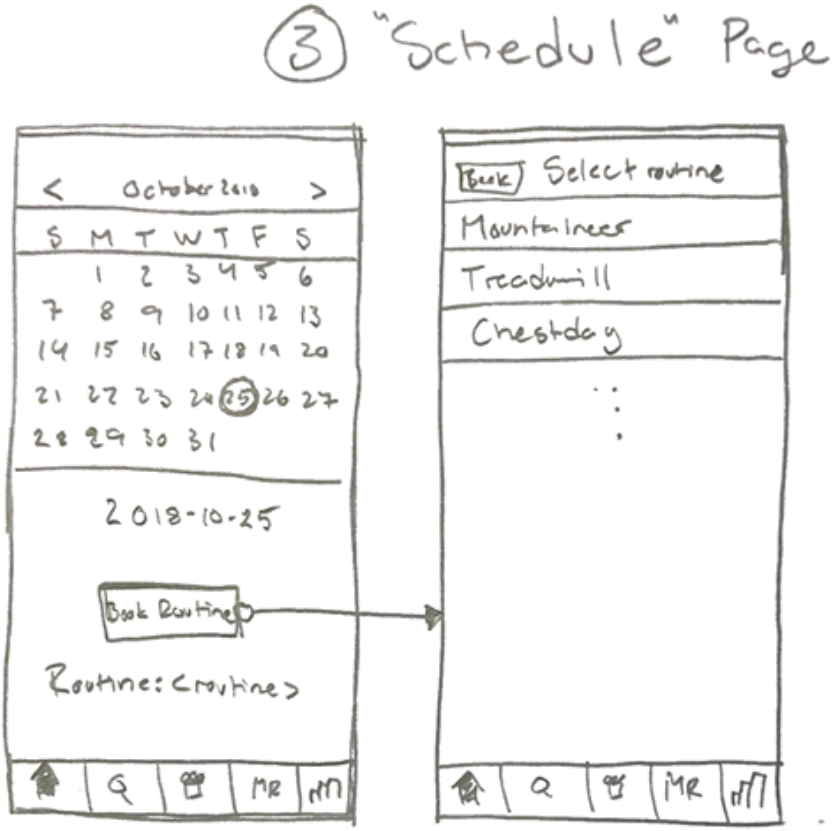

Figure 3: The Schedule Page Hierarchy

2.2.5 My Routines Page

Start Page The first page that the user sees when they go to My Routines is a list of all their current routines and a button that allows the user to create new ones. Pressing a routine in the list takes the user to a page detailing that routine and the exercises it contains. Pressing the button to create a new one does the same thing, but the page that opens is an empty template of a routine.

Routine Info Page This page displays info about the selected routine. For example the name, the amount of exercises and which exercises it contains. For a newly created routine most of these would be a default value or empty. The user is able to modify all of this information by simply pressing it. There is also a button that allows the user to add exercises which sends the user to a page similar to the browse page with exercises selected. If the user on the other hand selects an exercise that is already in the list, they are sent to a page with more information about that exercise.

Exercise Info Page This page display info about the selected exercise and allows the user to modify the version that is in their routine. For example they can change the amount of sets included and the amount of reps and weight that every set contains.

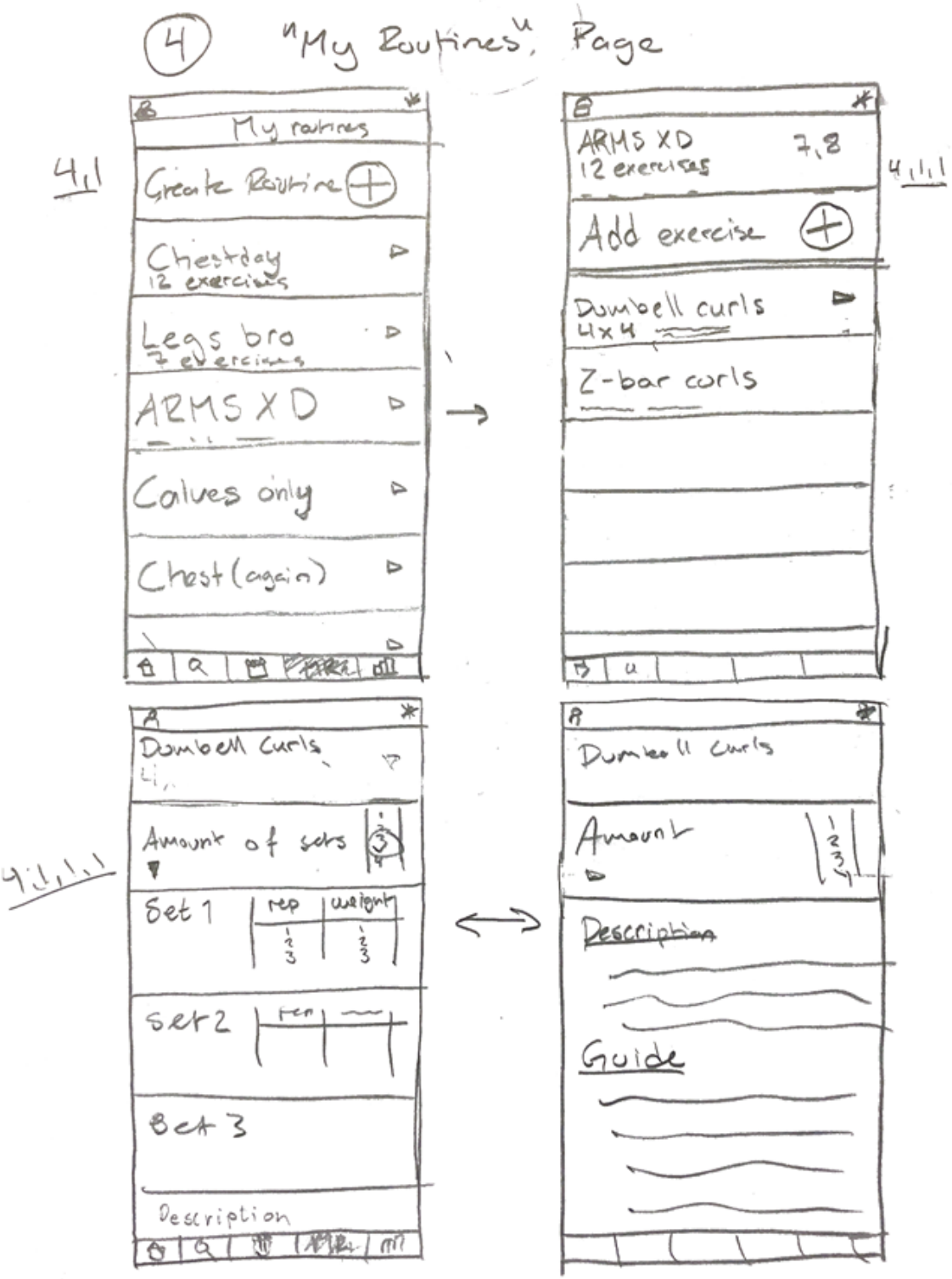

Figure 4: The My Routines Page Hierarchy

2.2.6 Statistics Page

Start Page The first page that the user sees when they go to the Statistics Page shows a graph and a couple of buttons that lead the user to either the Lifetime Stats page or the History page. The graph shows the routine scores of performed routines in the current week.

Lifetime Stats Page This page contains all the information that has been gathered by the user since they began using the app. For example the user can see how many routines that has been performed since they first started using the app. Other than that there is also a button that allows them to go back to the Start Page.

History Page This page shows the user a list of performed routines and the corresponding date for these is also displayed. If the user presses on some item in the list it will be directed to the History Routine Details page.

History Routine Details Page This page contains a list of all exercises for the corresponding routine pressed in the History page. The user can see whether or not they managed to perform an individual exercise by looking at the check box next to the exercise name. If an exercise was performed successfully the check box will be checked, else it will be unchecked.

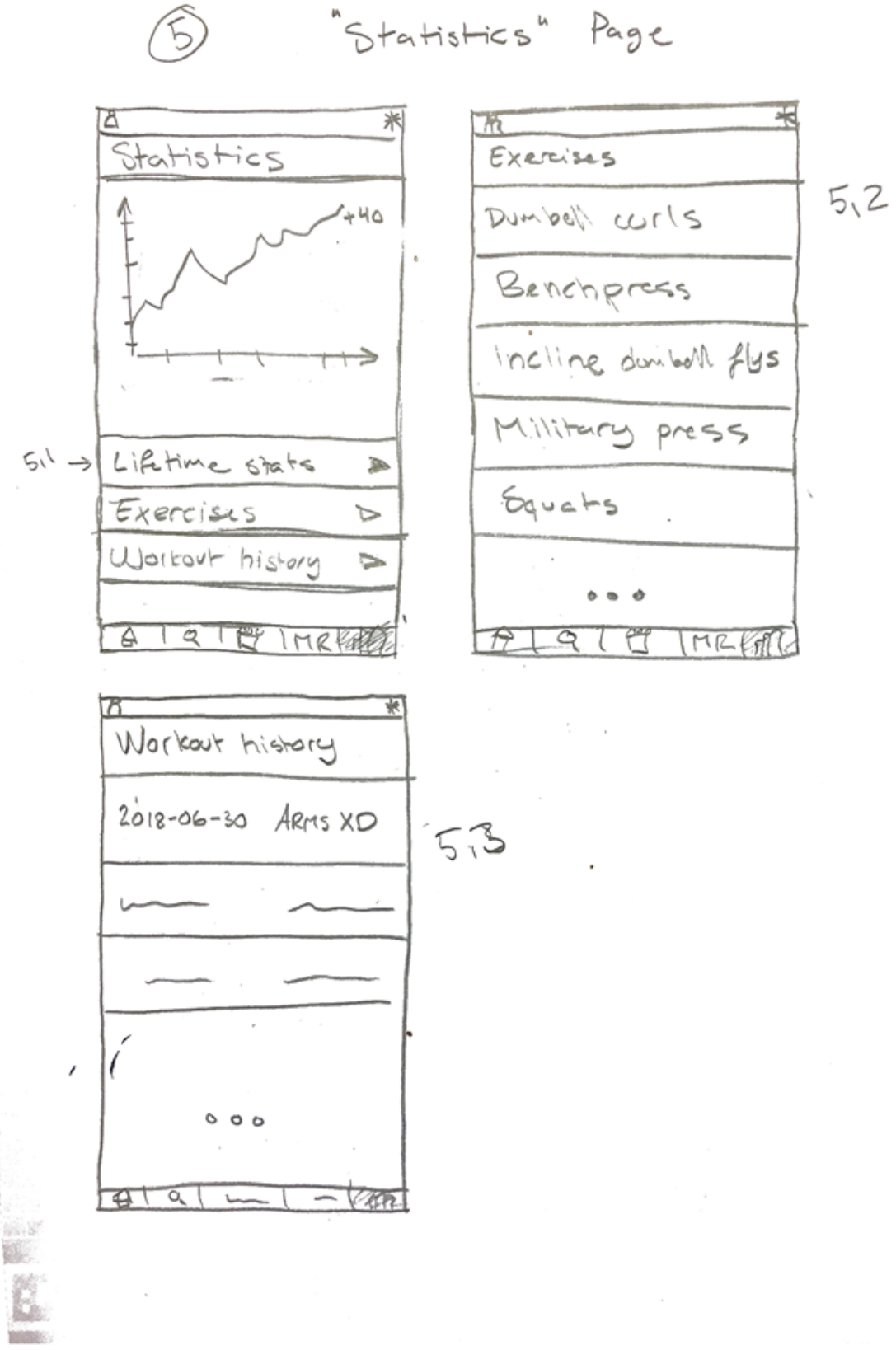

Figure 5: The Statistics Page Hierarchy

3 Domain model

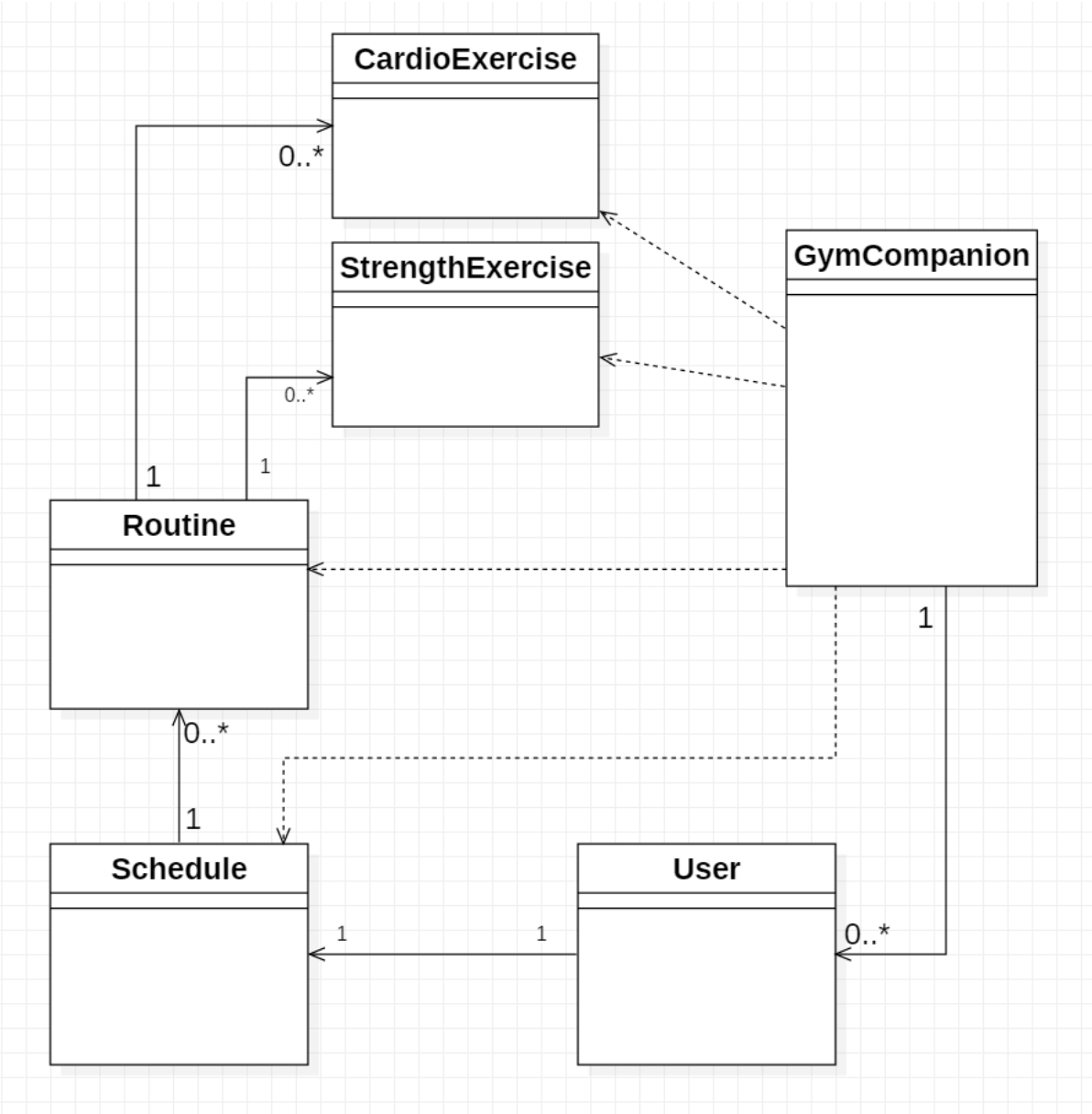

Figure 6: A high level overview of the application using a UML diagram

3.1 Class responsibilities

Explanation of responsibilities of classes in diagram.

3.1.1 GymCompanion

This module represents the app. It has knowledge of all other modules.

3.1.2 Exercise

This module is an entity that represents an exercise. It knows details about a particular exercise such as how it is to be performed.

3.1.3 Routine

This module is an entity for an exercise routine. It consists of a collection of Exercises that are performed in some sequence. A Routine knows the order in which Exercises are to be performed.

3.1.4 Schedule

This module is an entity for a schedule. It puts Routines in a calendar so that the User knows when a particular Routine is to be performed.

3.1.5 User

This module has a Schedule. It represents the user of the app. It only knows what the user needs to know, i.e. when the user should exercise and which Routine that is to be performed at those moments.

4 System architecture

The application runs on a single local mobile device. It is divided into packages, each containing related classes and files. Gym Companion implements the MVVM pattern, which is a form of MVC, where the ViewModel handles communication between the View and the Model. These three application elements are split into three different packages with the respective responsibility area. This separates the logic from the user interface.

4.1 Quality and Testing

Unit tests for each package are located in the **'src/test/java/se.chalmers.group22.gymcompanion'** folder. **LocalDatabaseTest** and **ParserTest** need an Application context in order to run and access local files, therefore they are located in the **'src/androidTest/java/se.chalmers.group22.gymcompanion'** folder. Unit tests reach a coverage of 71% lines of code in the Model package, excluding the Android tests.

The project consists of one single component: the application. The application's subsystem is divided into the following components:

4.2 GymCompanion

This package contains the main class of the application, **GymCompanion**. **GymCompanion** is the class that is used to represent the model for the viewmodels. The user state is read and modified through this class. The great number of methods in the class is a consequence of following the Law of Demeter principle. Following the law means that every variable that should be accessible from outside the model should have it's own getter method. Method calls get redirected to their responsible classes if **GymCompanion** can't handle the call by itself. There is only one instance of **GymCompanion** that is shared between all viewmodels.

4.2.1 Diagrams

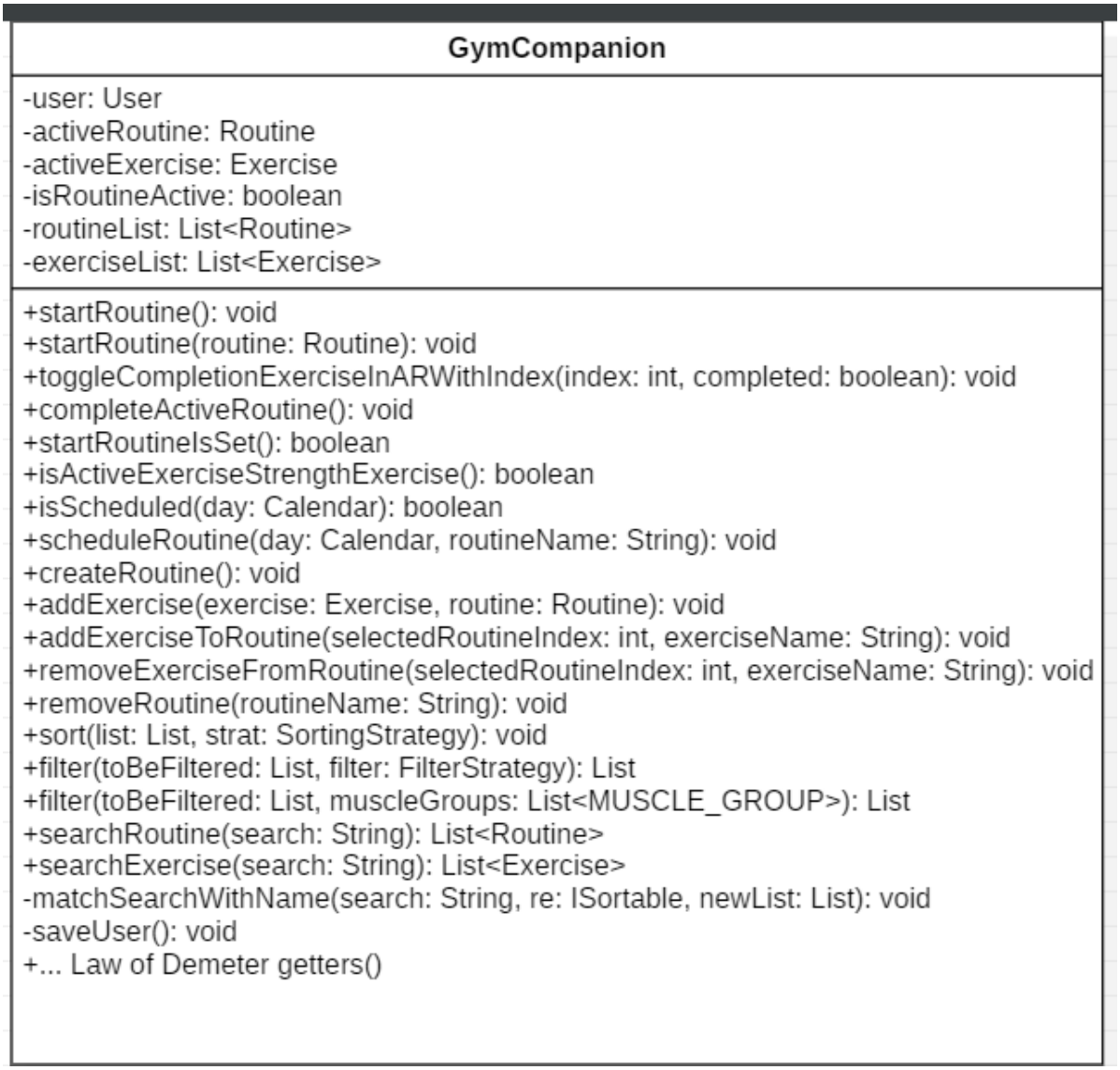

Figure 7: GymCompanion Package UML

4.3 Schedule

This package contains a schedule model. The schedule model is responsible for maintaining and scheduling user routines. This is done by connecting a day to a routine, represented by a HashMap in the code, which simulates a bookable calendar for the user.

4.3.1 Diagrams

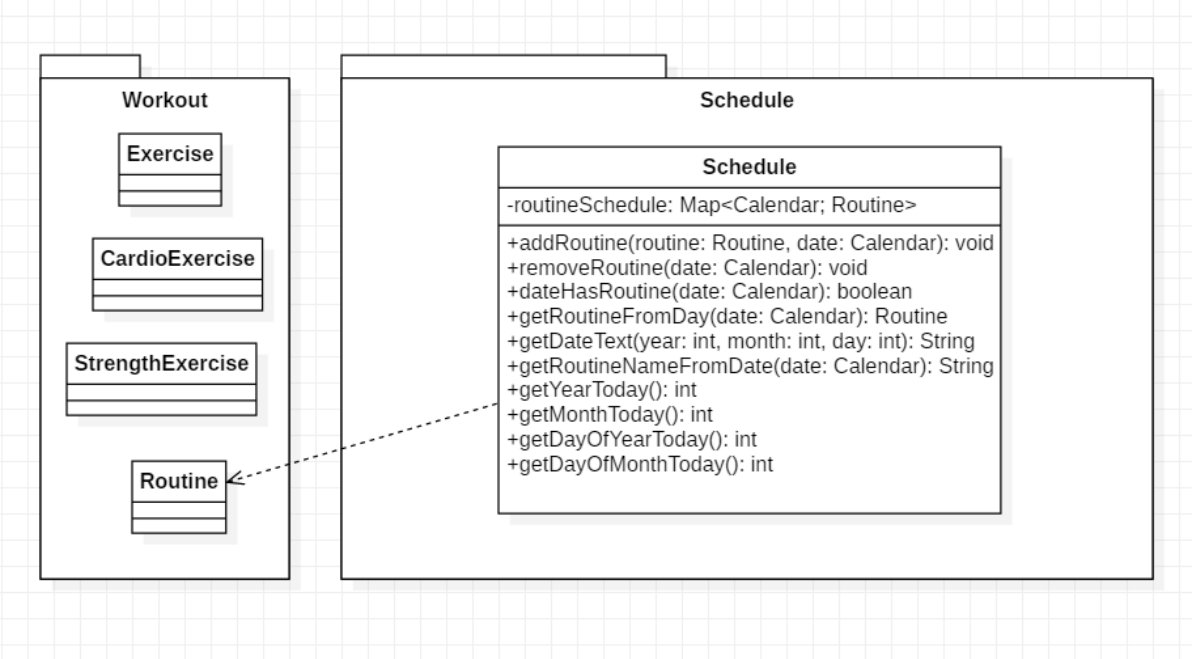

Figure 8: Schedule Package UML

4.4 Statistics

This package contains a statistics model. Its responsibility is to calculate statistics and prepare the data for plotting graphs. It takes routines and days from the schedule and with them creates data points for a graph. The days are returned as x-values, so that they can be used to show change over time. The routines are transformed into scores, which differ depending on which exercises they contain and how difficult they were, and are returned as y-values.

4.4.1 Diagrams

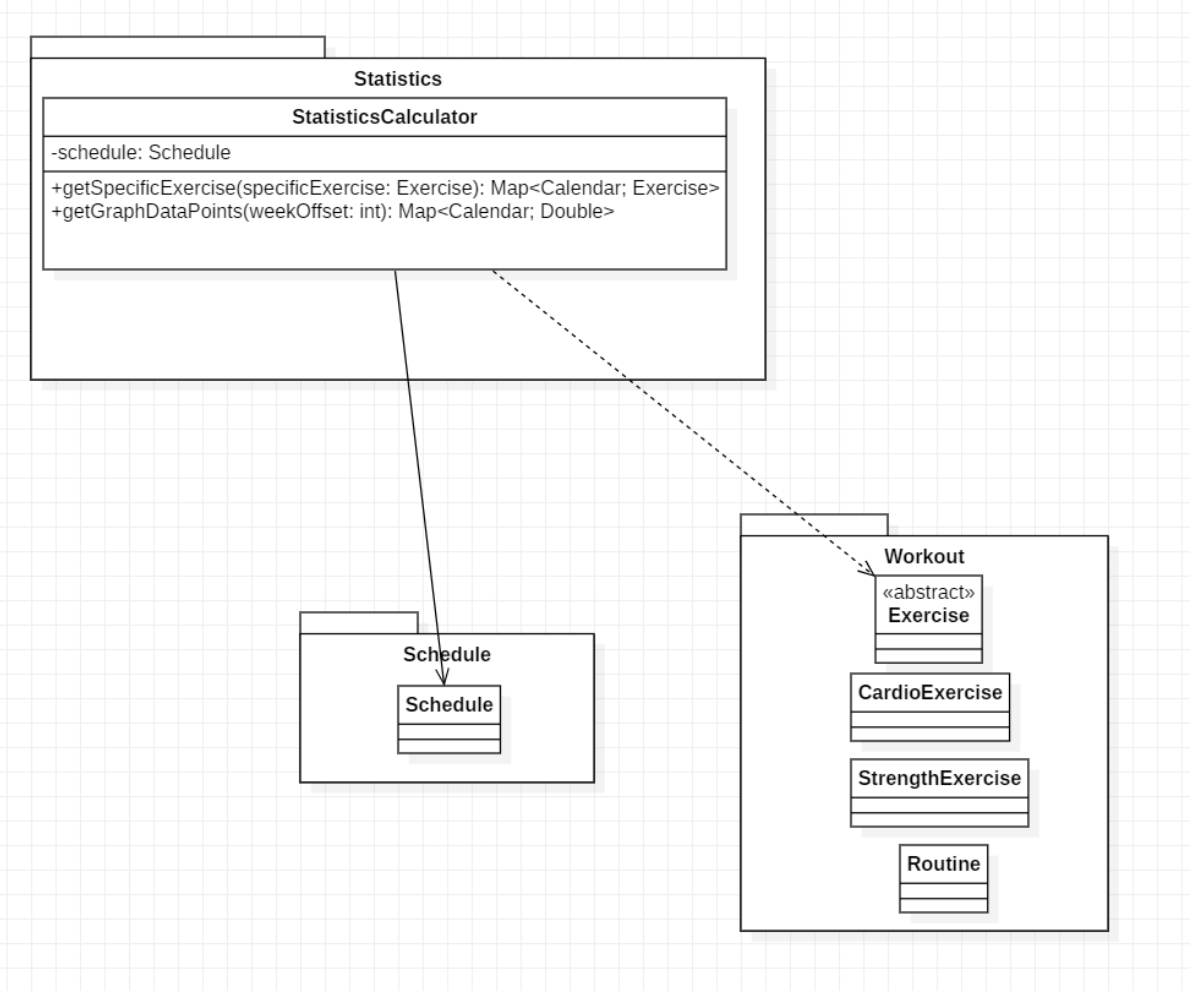

Figure 9: Statistics Package UML

4.5 Strategies

This package contains logic for sorting algorithms. The classes in the package define different strategies as part of the Strategy design pattern. The strategy pattern makes it possible to choose an algorithm at run-time, while also making it easy to create and implement new algorithms. In the app the strategies are divided into two categories, **FilterStrategy** and **SortingStrategy**, which are both represented by interfaces that their respective type of strategy can implement. There are two different filter strategies and four different sorting strategies.

4.5.1 Filter Strategies

The filter strategies are **BeginnerFilter** and **MixedFilter**. Their task is to filter a list in a specific way.

- **BeginnerFilter** is used if the user wants to filter out everything except the easiest Routine/Exercise from every muscle group.
- **MixedFilter** on the other hand is a completely random filter that returns a randomised set of Routines/Exercises.

4.5.2 Sorting Strategies

The sorting strategies are **AscendingAlphabetic**, **AscendingDifficulty**, **DescendingAlphabetic** and **DescendingDifficulty**. Their task is to sort a list in a specific way.

- **AscendingAlphabetic** sorts a list of Routines/Exercises according to their names in alphabetical order starting from the first letter of the alphabet.
- **DescendingAlphabetic** sorts a list of Routines/Exercises according to their names in reverse alphabetical order starting from the last letter of the alphabet.
- **AscendingDifficulty** sorts a list of Routines/Exercises according to their difficulty from lowest to highest.
- **DescendingDifficulty** sorts a list of Routines/Exercises according to their difficulty from highest to lowest.

4.5.3 Diagrams

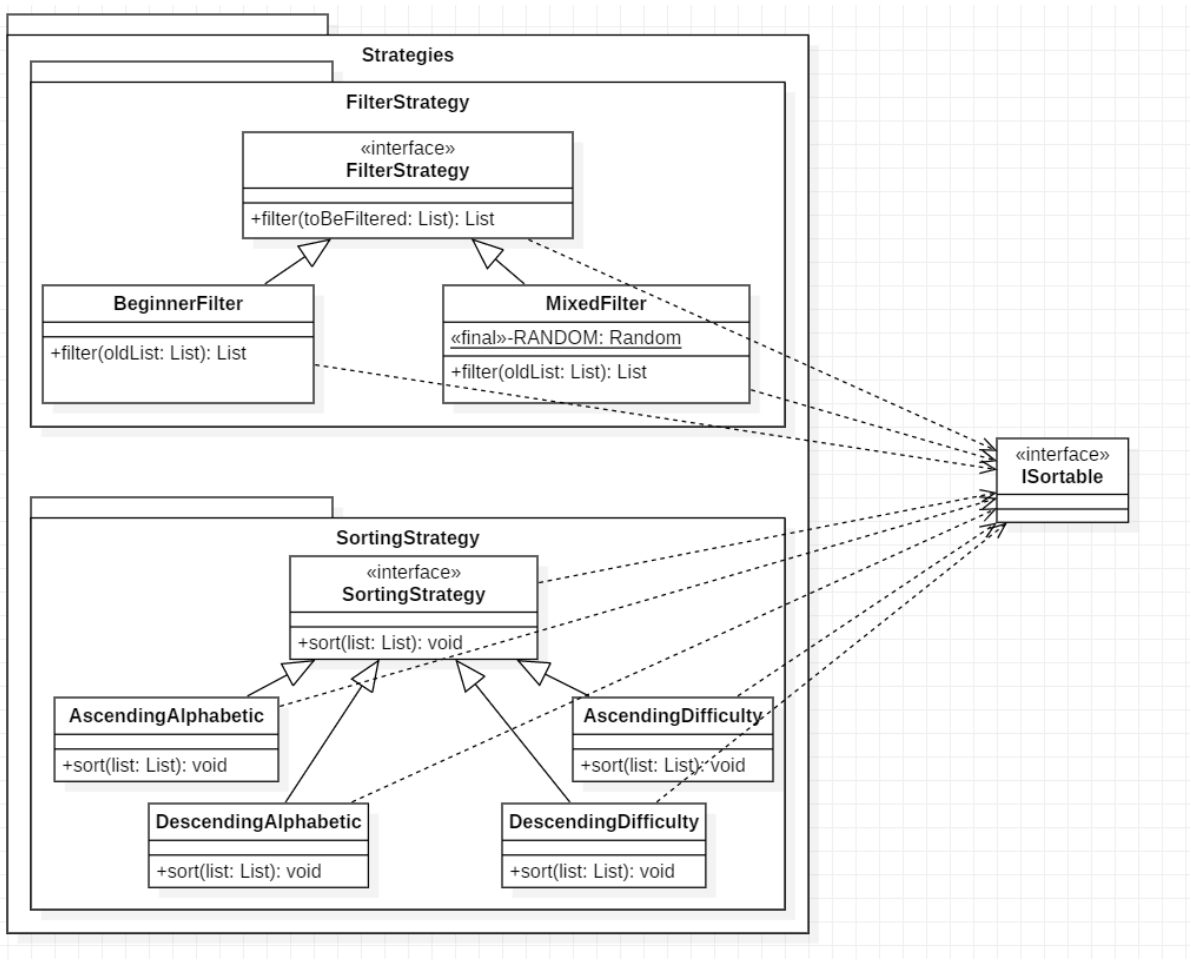

Figure 10: Strategies Package UML

4.6 User

This package contains a single class responsible for handling user related actions. The user class contains all the information about the user, their schedule, their own routines and their completed routines. This information is saved locally, so that it is still there after the user exits the app. Just as in the class **GymCompanion** the User class is packed with methods to comply with Law of Demeter. The User class holds instances of **Schedule** and **StatisticsCalculator**. Methods regarding scheduling actions are directed to Schedule and same for statistics actions with **StatisticsCalculator**. A User also has a list of **Routine** objects as well as a list of completed routines.

4.6.1 Diagrams

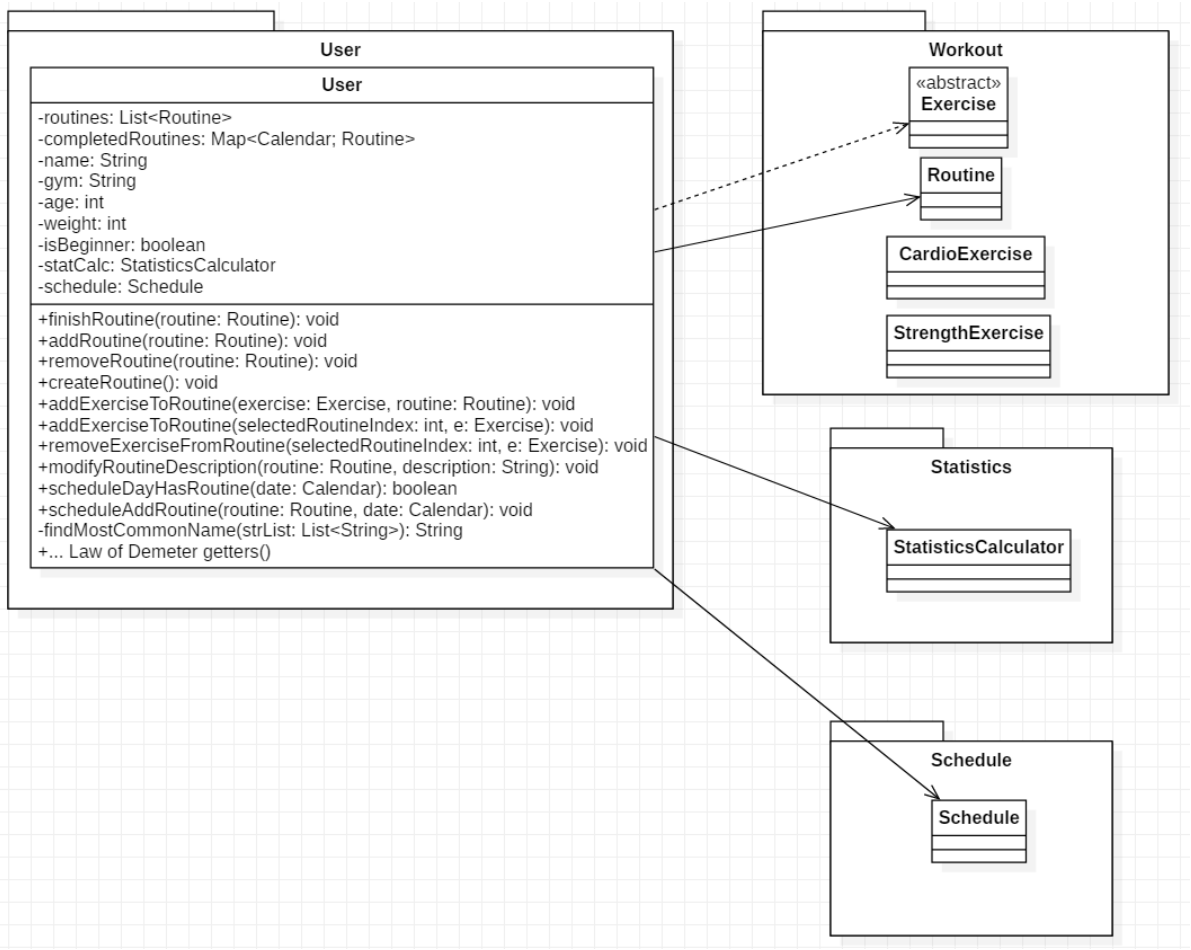

Figure 11: User Package UML

4.7 Workout

This package contains classes for modelling a workout routine and its exercises. The exercise model is divided in an exercise hierarchy.

4.7.1 Routine

Routine is a class that models a workout routine. Its primary functionality is to house a list of different exercises. The user is able to add and remove exercises from a routine.

4.7.2 Exercise

Exercise is a class that models a training exercise. Exercise is represented by a hierarchy with an abstract **Exercise** class at the top and two implementations **StrengthExercise** and **CardioExercise** below it. The abstract **Exercise** class makes use of the template method design pattern by defining abstract ”skeleton” methods **calculateScore** and **clone**, and then letting its subclasses define concrete implementations.

4.7.3 Diagrams

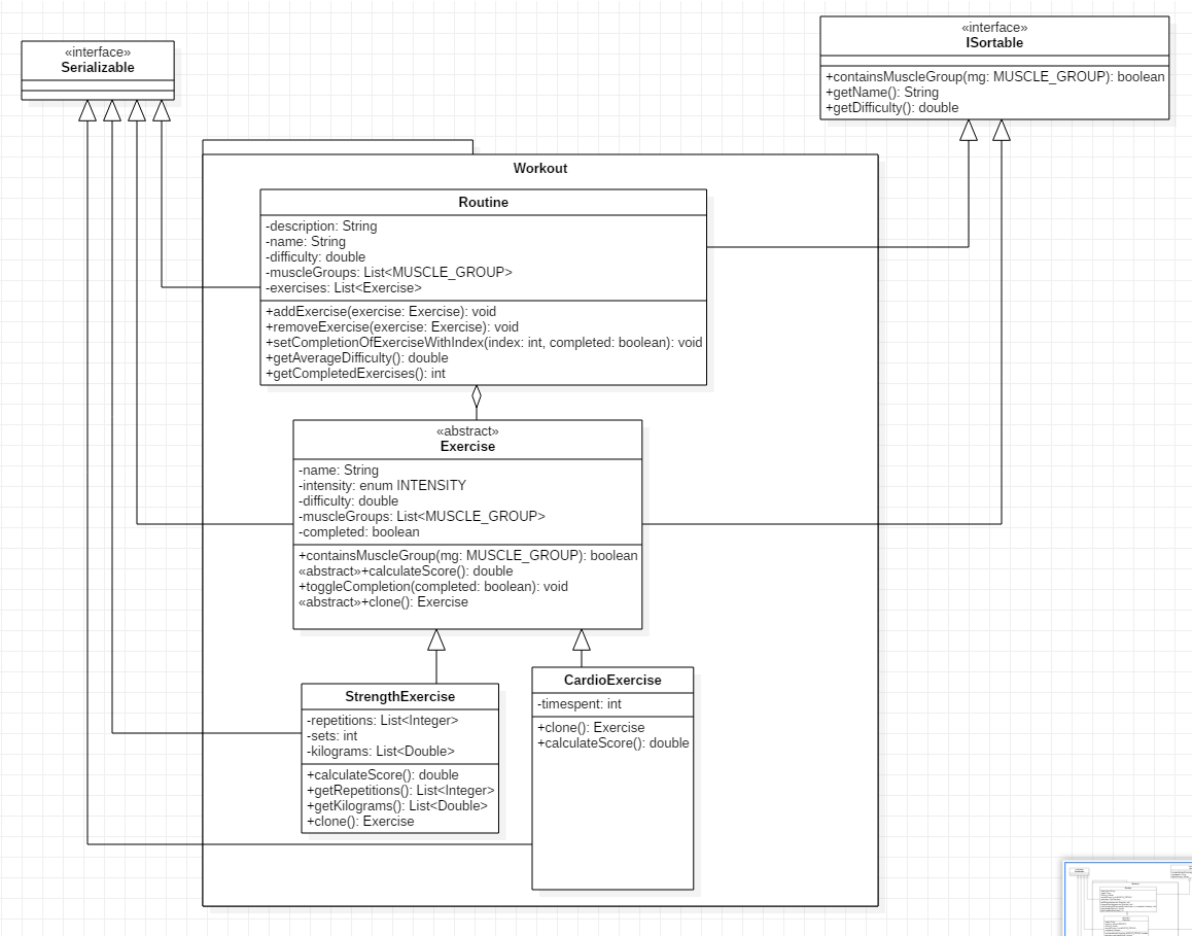

Figure 12: Workout Package UML

4.8 DataStorage

This package contains logic for parsing, loading and saving user data. The package consists of two classes, LocalDatabase and Parser. LocalDatabase handles serializing

and deserializing user objects. Parser translates premade routines and exercises from JSON-data into objects during runtime.

4.8.1 LocalDatabase

The LocalDatabase class loads and saves a User object from a local file by using the built-in Serialized java interface. The application occasionally saves the current user, and LocalDatabase is the class that manages the save action. The user is loaded into the application in the beginning of the application lifecycle.

4.8.2 Parser

The Parser class parses JSON-data stored in files located in the 'res/raw' folder. Each JSON-object and its properties is sent to their respective class constructor and creates an object with matching fields. New premade routines and exercises can be created by adding new JSON-objects in these files.

4.8.3 Diagrams

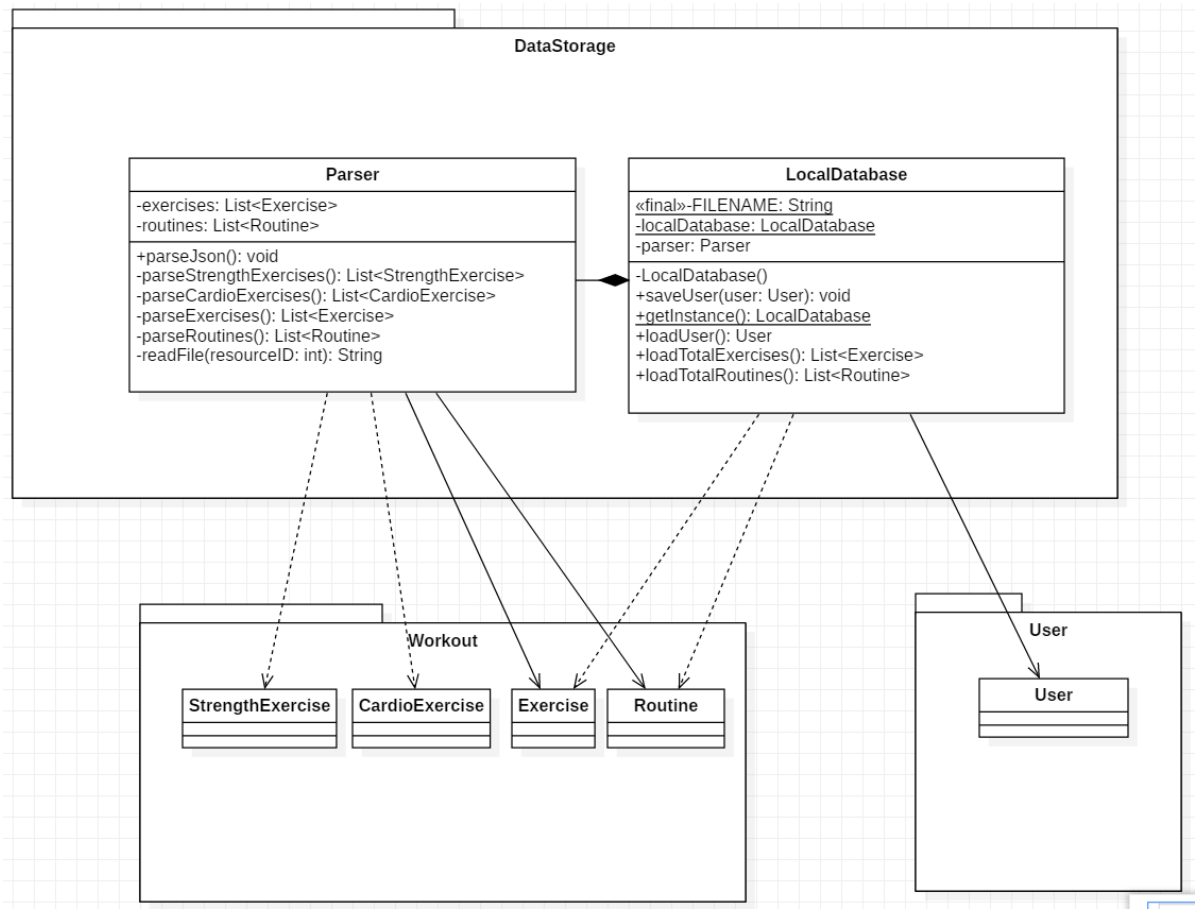

Figure 13: DataStorage Package UML

5 Persistent data management

The application stores user data by serializing a User object into a sequence of bytes in a local file. The object is then deserialized from the file into an object that can be used during runtime. Here the built-in Serialized interface is used within User and its fields to enable such data management. The saved user is loaded when the application starts and thus initiates the state of the user. The user is saved on occasions when user data is modified. The class LocalDatabase is responsible for saving and loading User objects, giving other classes the ability to access the user data.

The premade routines and exercises used by the application are stored in files with JSON-structure. The files are divided in ”routines.json”, ”strength_exercises.json” and ”cardio_exercises.json”, each containing the respective content for each class. Each JSON-object consists of properties that are then sent in to an object constructor, with the use of the library ’gson’. The class responsible for parsing these files is called Parser.

All resource files, such as images, string values, layouts and user files, are located in the ”res” folder inside the project's ”main” folder. An essential part of the application's design consists of layout files in the folder ”res/layout”. These XML-files are named after their use in different application pages and they are used to create the intended layout design of a page.

6 Access control and security

Access control and security is not applicable to the current application, and the following sections explains the motivations as to why that is.

6.1 Access Control

The application does not have different roles, since the need of an admin is not necessary. It's not possible to cheat in the application, since the users themselves decide how they want to use the app and what they do affect no other users. There is no user interaction either, so there is no need for anyone the moderate how a user uses the application.

6.2 Security

Since accounts were never implemented, the applications security is entirely nonexistent. No log in is needed after the application is started, and the user is put directly at the home page of the application. Information about the user is stored directly on their device, and almost none of that information could be considered sensitive and in need of encrypting.

7 Peer-review

The following is the peer-review of Group 23s PaintIT application.

7.1 Design Principles/Design Patterns

The following sections contain information about the design patterns that were implemented fully or to an extent in the program.

7.1.1 Factory Pattern

In the class **ImageServiceFactory** there is an implementation of *Factory Pattern*. This class has a static method which takes a File as a parameter to decide the file type, and returns a (new) **JPGImage** or **PNGImage** depending on the parameter. The factory

is used in the **PaintITController** class, and effectively removes the dependency from **PaintITController** to **PNGImage/JPGImage** and replaces it with a dependency to the factory.

7.1.2 Bridge Pattern

The class **LineTool** implements a *Bridge Pattern* by composing a **LineTool** object with another object implementing a draw method. It is however not an implementation of the *Strategy Pattern*, as the name **DesignStrategy** might suggest, because the behaviour of the implemented method can't be changed at runtime.

7.1.3 Template Method Pattern

In the class **ColorTool**, which is abstract, there are 3 methods (**onMousePressed**, **onMouseDragged**, **onMouseReleased**) that acts as templates for the subclasses to override. Although, in two out of the three classes which extends **ColorTool** the method **onMouseReleased** is completely empty. This means that those two classes are forced to implement a method which they are never going to use. Furthermore, the **onMousePressed** method is used in the exact same way in each of the subclasses, which would suggest that this method should be moved to the superclass. Therefore, the only correct implementation of *Template Method Pattern* is the **onMouseDragged** method, which is used by every subclass but with distinctive functionality.

7.2 Documentation

The methods are well-documented with a purpose as well as a thorough description of the parameters. All classes have a documentation, however, they do not present all the necessary information. Authors and date of creation are missing. The documentation, which describes the classes dependencies, is not present.

7.3 Name Usage

Proper names are used for both methods and classes. It is easy to decipher what the methods do, and the names are short and follows the format of camel case. It is a little bit misleading with a package that is named **View** (after the *MVC-pattern*) that only contains classed with a name ending with Controller. These are completely different things in the *MVC-pattern*.

7.4 Modular Design

The classes are divided into packages depending on their responsibilities. thus it is easier to exchange or reuse these modules for other purposes. The **Memory** package has no dependencies, and **Tools** has only a dependency to **Memory**.

7.5 Abstractions

ColorTool is an abstract class that abstracts functionality from its subclasses **LineTool**, **ColorPipette** and **FillTool**. This makes it possible for the classes to create their own version of a certain method that all subclasses share. The classes **Position**, **Pixel**, **PixelImage** and **ModelImage** use composition which is preferred over inheritance. A **ModelImage** has a **PixelImage**, a **PixelImage** a matrix of **Pixels** and a **Pixel** a **Position**, which seems logical.

7.6 Tests

All tests pass. the class coverage is 100%, the method coverage is 71%, and the line coverage is 77%. The coverage is adequate.

7.7 MVC Implementation

The enumerable **SelectedTool** is placed inside the **Controller** package, which leads to a dependency from the Model package to the Controller package. A quick solution is to move the enumerable outside the package so no dependency occurs. There is also a circular dependency between **PaintITController** and **MainFxmlController**, which should be avoidable. All other controllers (views) also have a circular dependency to **MainFxmlController**.

7.8 Understandability

As mentioned previously, the names of the classes and methods are descriptive and precise. The program overall is easy to understand since most of it is thoroughly commented.

It is clear what the icons in the program do, given that you have used a similar program before. A needed addition would be tool-tips for the icons so that beginners get to know what different buttons do.