Cor Sanum

Principles of Software Development
Android Project Report

Rocio Lopez Perez, Maciej Żurad
{rocio.nightwater@gmail.com, maciej.zurad@gmail.com}

1 Introduction

In this report, we present our Android application for the course **Principles of Software Development**. Our application named **Cor Sanum** (en. Healthy Heart) is designed for people who want to keep a healthy lifestyle. It's goal is to allow people create exercises, which then can be followed. Our exercises are composed of activities, each activity can be for example "Running", "Running fast", "Walking", "Walking fast", "Stretching", etc. Where each activity has a starting and ending position and a duration for the case of "Stretching" activity. New exercises can be created, while existing ones can be modified. We also allow for a *Free route* mode, which does not constraint the user to any activity and allows to freely do any exercise, giving him feedback about his position, step count and speed.

2 Design overview

We wanted to focus also on material that was not present in the class. That's why we are using *Dynamic Fragments*, *Recyclers Views*, *Fragment Dialogs*, *Swipe Layouts*, *Scene Transitions*, *Android Animations*, *Google Directions API* and *Google Fit API*. They will be explained in detail in places they are used.

We decided to opt in for as much code re-usability as possible in order to employ good software development practices. Hence, the use of Android Fragments. We also reuse classes, which will be seen later on as well. In general, application was designed with great care using DRY[1] and KISS[2] principles. That's also the reason we decided to use some third party libraries, in order not to reinvent the wheel.

3 Data Model

Our data model is quite simple. It's an observable list of Exercises, where each Exercise represents meta-data about an Exercise. An Exercise on the other hand, has an observable list of activities, named Actions. Every Action as stated before, has a starting and ending position, a type (e.g. Walking) and a duration in case of Stretching action. This model gets modified, when the user interacts with our application. For example, the user can add new exercise, or modify the current one or he can simply restore factory settings. Therefore, we decided to put our model in persistence layer by serializing this list of Exercises into JSON, and then simply putting it in Shared Preferences in a private mode. It's a good solution in our opinion, because our data is comparatively small and using a Content Provider would be an overkill, resource- and implementation-wise. We decided to use a very well-known library to perform such serialization called GSON [3].

4 Android Activities

4.1 Base Activity

Base Activity is an abstract class, that all activities except the Main Screen Activity inherit from. This activity is responsible for loading the model of our data into memory in *onCreate* and *onResume* callbacks, as well as saving model it in *onPause* callback. Activities inheriting from Base Activity have therefore access to the model in an easy way, and we are sure that the model will be correctly saved and restored from persistence layer.

4.2 Main Screen Activity

Main Screen Activity is the entry point of the application. In that activity the user can choose between going to a *Free route* mode, browsing his Exercises or restoring to factory settings. Figure 1 shows the UI of this activity.

[1] DRY - Don't Repeat Yourself
[2] KISS - Keep it simple stupid
[3] GSON is a JAVA–to–JSON serialization library `https://github.com/google/gson`

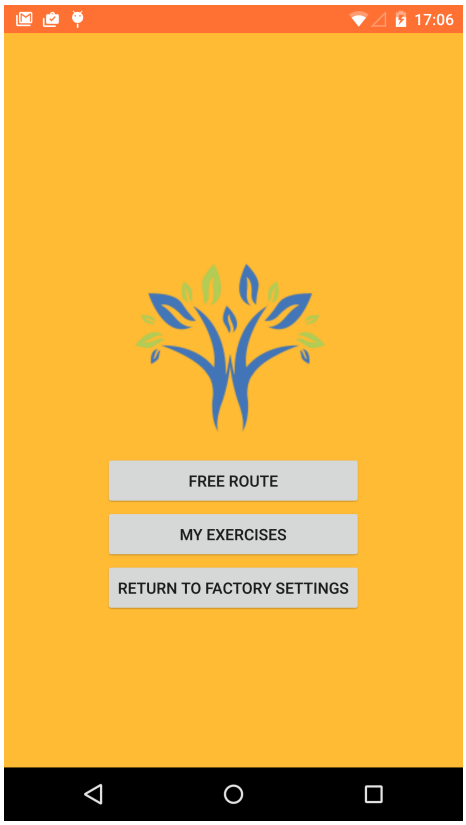


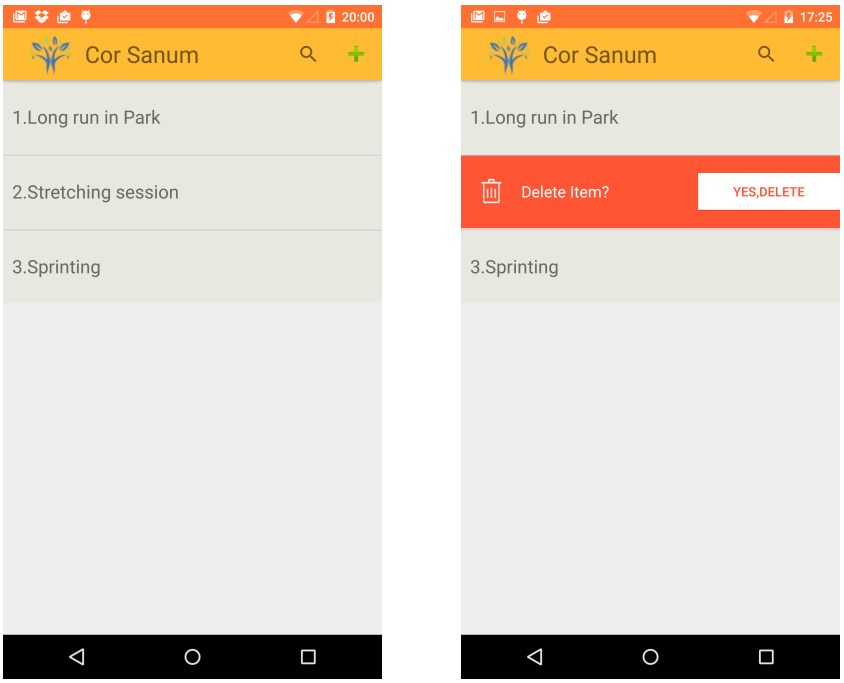
Figure 1: Main Screen Activity (App's entry point)

4.3 My Exercises Activity

My Exercises Activity is invoked, when the user clicks on the button *My Exercises* in Main Screen Activity. This activity allows user to browse current exercises, delete exercises and add new exercises.

This activity is using a *RecyclerView*, which is a new Android component also referred to as *ListView 2.0*. RecyclerView is much more efficient than normal ListView, because it does not keep in memory all other views, but rather always keeps the same amount, usually 7 items and hot-swaps items as they become invisible. It also allows for implementing nicer effects, such as swipe to delete functionality as presented in figure 2. User can swipe any item from the RecyclerView from right to left, and a new layout will overlay existing one, showing an animated trashcan icon and a button that upon clicking deletes this exercise.

Another interesting thing in this activity is the Tool Bar, which is at the top of the screen. The plus button allows to add new activities. It launches a custom fragment dialog. Shown in figure 3, left picture displays scenario, what happens when the user didn't type anything into the text edit field. This is implemented using a Text Watcher. After a user types in a valid name, he can then create a new exercise. When this happen, the user is taken to a new activity called *Exercise Detail Activity*, which allows to finish creating the new



(a) Before Swipe (b) After Swipe

Figure 2: My Exercise Activity - Swipe to delete functionality

exercise. This process is described later on.

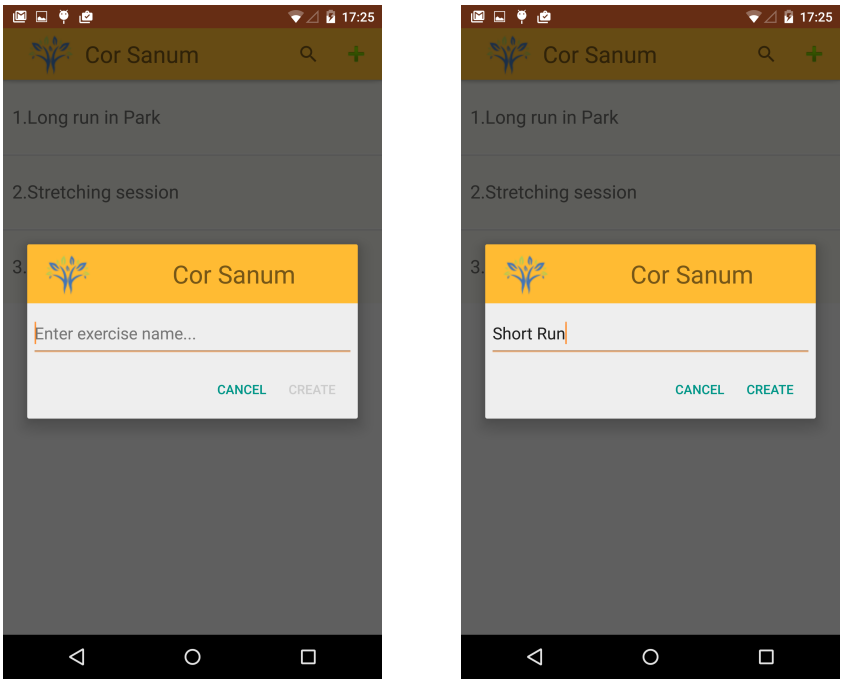
4.4 Exercise Detail Activity

This is a major activity that allows for modifying existing exercises. We support complete modifications, meaning: changing type of an action, changing starting or ending position of action with visual feedback, as well as adding duration in case type of an action was changed to, or remains "Stretching". We can also delete any action, as well as add new actions to our exercise.

This activity is composed of Android Fragments, which are dynamically bound at run–time to the activity. Depending on which action is performed at any time, different fragments can be bound. If we enter to this activity by simply browsing and choosing an exercise from "My Exercises", then we get *Exercise Detail Header Fragment* at the top and a Map Fragment at the bottom. However, if we click the plus button to add a new exercise in "My Exercises" activity, then there are no actions in a newly created exercise. Therefore, we start off by showing *Edit Action Fragment* at the top, since we should add new actions to this exercise. Each fragment is explained more in detail below.

4.4.1 Exercise Detail Header Fragment

This fragment as stated before is bound by default, when user enters to this activity by selecting one of the exercises in "My Exercises", or if the user comes



(a) Empty field not possible

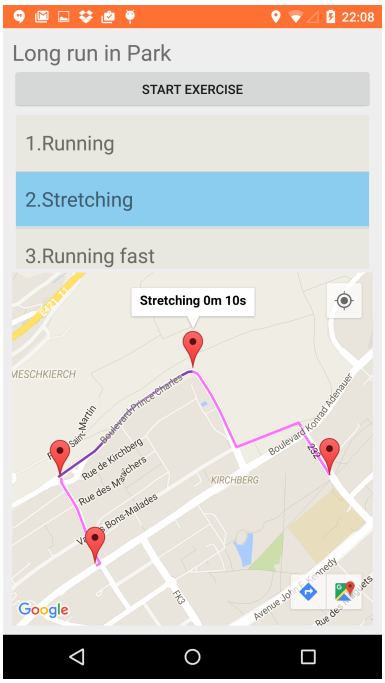
(b) Non-empty field accepted

Figure 3: My Exercise Activity - Create new exercise dialog

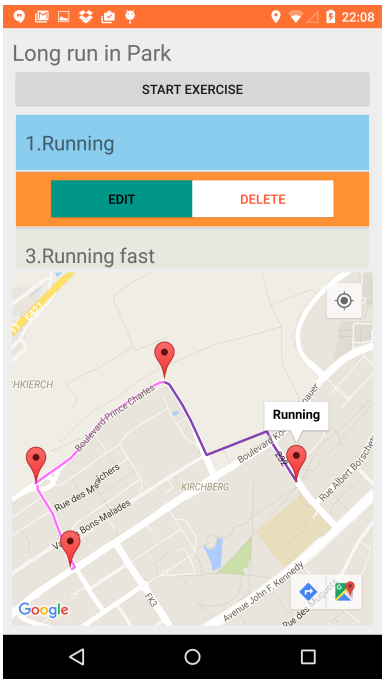
back from editing by clicking "Save" button. Figure 4 shows some basic functionality of this fragment. This fragment occupies only top part of the screen, bottom part is occupied by a Map Fragment. However, these fragments communicate with each other through an activity. We can see another RecyclerView, which as before allows for swiping to discover two buttons hidden under. First one, allows for editing. It replaces this fragment with the *Edit Action Fragment*. Delete obviously deletes current action. We also have a "Start Exercise" action button, that launches "Exercise Activity", which holds main logic for controlling the exercise.

4.4.2 Edit Action Fragment

This fragment is launched either if the user decided to create a new exercise or if "Edit" action was chosen from the swipe layout in the previously described fragment. Changing from the previous fragment to this one happens in a transaction using an Android Animation, that is stored in a appropriate XML resource file under *res/animator/*.xml*. Figure 5 depicts scenario, when a user wants to add or modify an action. Once, we are in Edit Action Fragment, we can move markers of the currently edited route which will cause rerouting to happen, we can also change the type of our Action. On top of that we can click "Add" to place a new route. Once the user finished editing or adding a new action, he can click "Save". This will cause a reverse fragment swap transaction, and the user will return to viewing his current Exercise, where he can than add/modify



(a) Highlighting a route



(b) Swiping action item from right (Delete/Edit)

Figure 4: Exercise Detail Activity - Route select and swipe to edit/delete

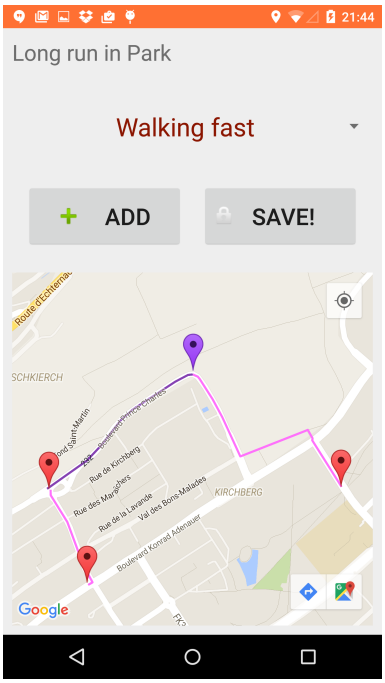
more actions or start the exercise.

4.4.3 Map Controller

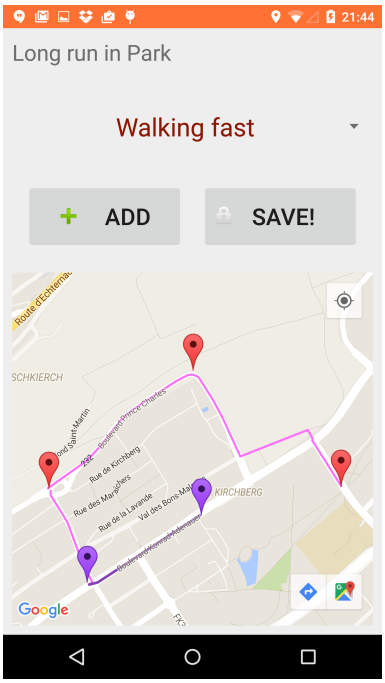
Map Controller is not a fragment, but it's a class that is responsible for managing the content of the Map Fragment. Showing the routes from marker A to marker B, markers themselves happens in this class. It's a middle layer between actual model data and visual representation. When the user is editing an action he can simply drag markers, he can also see which markers are draggable, because the route and the color changes as shown in figure 5(a).

4.5 Exercise Activity

Exercise Activity is the main activity holding logic for the actual exercise execution. It's composed of the same Map Fragment, which holds Map Controller and a Control Exercise Fragment. It's worth noting also that since we wanted to reuse as much code as possible, this activity is used also for "Free route" mode. The difference between "Free route" and a "Predefined path" / "User-created path" is that for "Free route" the path is empty. Depending on the received Intent, different logic is executed for this activity. This activity is also responsible for receiving data from Google Fit Service, which will be described later on. Figure 6(b) shows how user is requested to authorize Google Fit to receive sensor data.



(a) Editing a route



(b) After click "Add", new route was placed

Figure 5: Edit Action Fragment - Adding new action

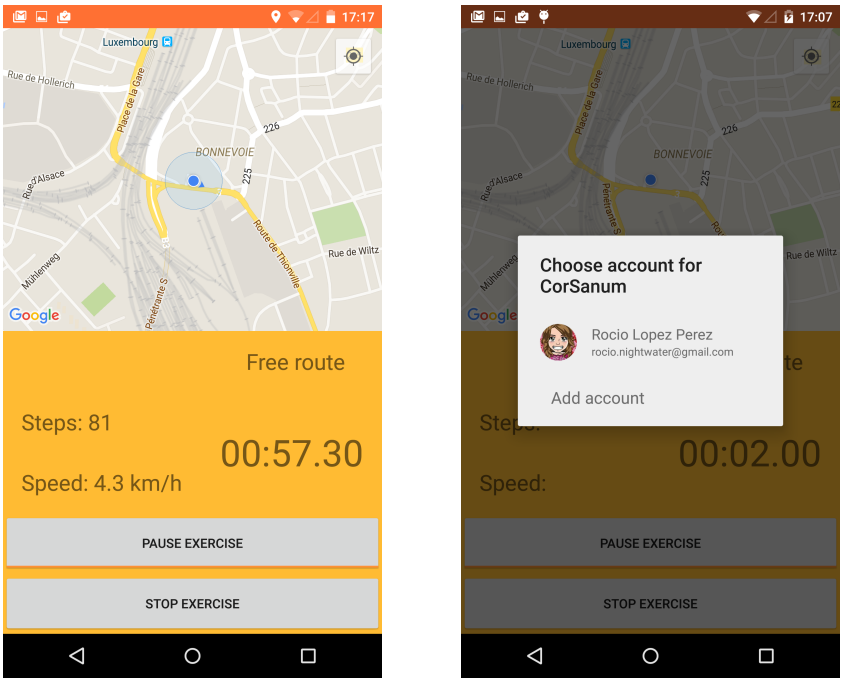
4.5.1 Control Exercise Fragment

Control Exercise Fragment occupies the bottom part of the screen in Exercise Activity. User can start, pause/resume and stop an exercise. Starting an exercise forces a connect to Google Fit Service, which will display the received data. There are two data types supported. Step count since the beginning of the exercise and instantaneous speed in km/h.

Depending on the selected mode, the fragment does different things.

- **Free route mode** - Since in "Free route" there is no path to follow. Starting exercise, simply starts a timer, which user can pause/resume and stop. Therefore the user can see, for how long he was exercising. As well as read the data from Google Fit.
- **Exercise following mode** - In this mode the user has to follow path, from waypoint to waypoint. Each segment represents an action and the user sees, whether he should "Run" / "Run fast" / "Walk" / "Stretch" / etc. In case of stretching, not only reaching the waypoint is necessary, but waiting until the stretching duration is finished is mandatory as well. Upon entering to a "Stretching" segment, the user will see a count down timer, which will inform the user about the progress, presented in figure 7(a). Once the timer finishes, the user will see a notification to go to the next waypoint, if he hasn't reached it yet.

Originally all markers are in red color, except for the next marker which will be in yellow. This shows the progress to the user. Once the user finishes a segment. A corresponding marker will turn to green color. Exercises is finished, when the user reached the finish line. This is depicted in figure 7(b).



(a) Free route (b) Google Fit Sign-in

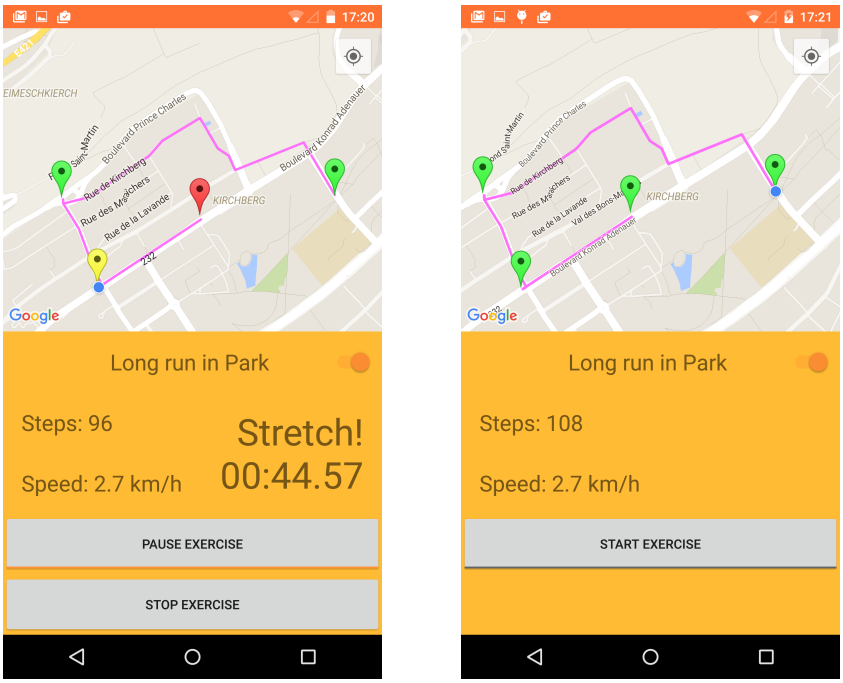
Figure 6: Control Exercise Fragment - Free route / Google Sign-in

5 Android Service and Broadcast Receivers

This sections describes the Android Service and Broadcast Receivers used in this application, which were necessary in order to use Google API Client.

5.1 Google Fit Service

Google Fit Service is a custom class, that implements an Android Service. It handles the connection of the Google API Client to Google Fit. It builds the client and attempts to connect to it. However, since Internet connection can be unavailable or the user might not agree to some permissions, it has to resolve conflict. Another problem is that, the information about conflict is accessible only in the Google Fit Service class, and we have to notify the UI to perform specific action to resolve the conflict. Hence, the use of Broadcast Receivers, which are explained more in detail in their sub-section.



(a) Stretching timer (b) Exercise completed

Figure 7: Control Exercise Fragment - Exercise following mode

Once client connects to Google, it registers for the requested data sources, which are cumulative step count as well as instantaneous speed. Upon getting the data sources, it then registers specific listeners. This listeners upon getting the callback with information, perform parsing and data manipulation and send information through Broadcast Receivers.

5.2 Broadcast Receivers

Since the data from Google Fit is received in a Android Service, we have to be able to send it to the running activity, such that it can update the UI. Broadcast Receivers are very easy in implementation and are widely used Android components. We use them to notify the Exercise Activity about Google Fit connection problem as well as we send regularly step count/speed data. Upon receive of the Intent, we update Text Views or start a conflict resolving method. Example of such conflict resolving is displayed in figure 6(b).

6 Testing with a Mock

In order to test our application. We decided to create a mock, which will mock our current position by setting a Location Source on Google Map. This mock is fed directly with the data describing our current exercise. This includes all actions as well as routes for these actions. However, the routes are not very good for displaying a smooth position transition on the map, because they consist of

points, which are relatively far away from each other. That's why, we interpolate between these points keeping the speed to the one that is set in mock. In figure 7(a), we can see the mock as a Android switch in the upper right corner of the lower fragment. Mock can be paused at any time and resumed to mimic the user taking a break. Mock also has the information about the actions, therefore, he will stop and wait, once he reaches the end position of a stretching segment if the stretch timer is still running.

It's important to note, that our mock by any means does **not** change the state of the application, except from mocking current location. It is meant as a testing framework, therefore only data about the exercise is provided, but nothing else.