Eötvös Loránd University

Faculty of Informatics

Department of Software Technology and Methodology

Meaningful Goal Tracker

Supervisor:

Dr. Rudolf Szendrei

Assistant Professor

Author:

Maksim Toropygin

Computer Science

Budapest, 2020

Contents

1 Introduction

The topic of productivity tends to be more and more popular every year. A lot of applications have been made [1] to fulfil the desire of people to be productive, to control and track their goals and tasks.

There is one unique characteristic of Meaningful Goal Tracker that makes it stand out. Like the title says, it puts the meaning first. This is one thing that is almost always missed in any productivity content, regardless of the form.

Productivity gurus convince people that they should be productive without explaining why to be productive in the first place. [2] And usually people are convinced without proper explanation. They start the race without knowing why they are running, and this meaninglessness of their life can be discovered suddenly.

The reason why this application was developed is to avoid such incidents. It helps to define the reason why user is being productive. It gives people the opportunity to actually decide what they value in life. And only after defining things that are important, the user can start building goals and compose tasks to achieve it.

In essence, the program should have several important functionalities:

- User can define what they value in life;
- User can create specific goals, based on one of the values;
- User can create a schedule. Every task in the schedule is based on one of the goals;
- User can track the progress related to the tasks of the schedule;
- As an additional functionality, user can use a to-do list, each item of which is based on one of the goals.

2 User Documentation

2.1 System requirements

This program is an Android application, which can run on Android devices.

The minimal version is Android 6.0 (API level 23), recommended version is Android 10 or higher (API level 29 or higher).

Minimal storage space is 18.27MB.

2.2 Solved problem

Meaningful Goal Tracker is an Android application that makes it possible for the user to be productive, but also forces the user to know what to be productive for.

The list of functionalities to be implemented can be found in the previous page. (See Introduction)

2.3 Navigation map

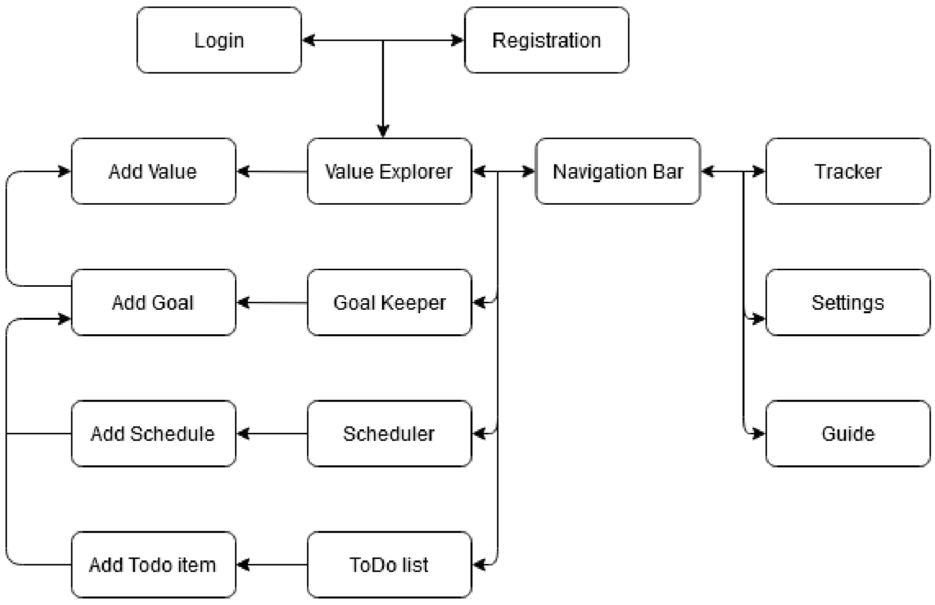


Figure 1 (navigation map)

In Figure 1 (navigation map) the navigation map of the application is described. When the application is first opened, user can move between "Login" and "Registration" pages.

When user is logged in, it is possible to move between different pages using the navigation bar. Apart from that, from the main 4-four pages user can go to the corresponding add/edit page.

2.4 Summary of the used methods and tools, how to use the program

2.4.1 Core

The program consists of different navigation pages, each of which solves one particular problem. All the pages are interconnected, and the workflow of the program will be explained here, the examples will be given.

If the user is not registered, then it should be done. After registration, user can log in into the system.

The first page that the user is looking at is the page of Value Explorer. On this page user can see the list of values. Each value has its title and description. User can add a new value, edit or delete existing values.

The second page depends on the previous one. This is the page for goals. The page has the same structure with one difference. Apart from the title and description, goals should also have associated values. That prompts the user to actually understand what is the deeper meaning of the goal that is being pursued.

The third page depends on goals. This is the schedule page. There is an idea of ‘Scheduler activity’ in the program. Scheduler activity has a title, description, associated goal, and also the schedule itself, where the user can define on what days and at what time the activity should be performed. User also can decide which particular scheduled event was missed or completed.

Sometimes scheduling is not enough, and there should be the tasks to complete without particular timing. For that we made the fourth page, which is to-do list. User can add any other task related to the goals here.

The last page related to the main problem is the tracker. Tracker is showing statistics for 3-three periods - last week, last month, and the last year. Statistics for every period is showing how many scheduled events were missed and completed.

2.4.2 Network

The application does not store data inside the phone; therefore, the user needs to stay online to access functionality of the application.

Every user has its personal account. If user is not logged in, the new account can be registered. The login for the account is the email of the user. Apart from that, user also can define the name, which can be changed. The page "Profile" is used for account data manipulations.

User can log out of the account in order to register the new account or log in into different existing account.

2.4.3 Basic guide

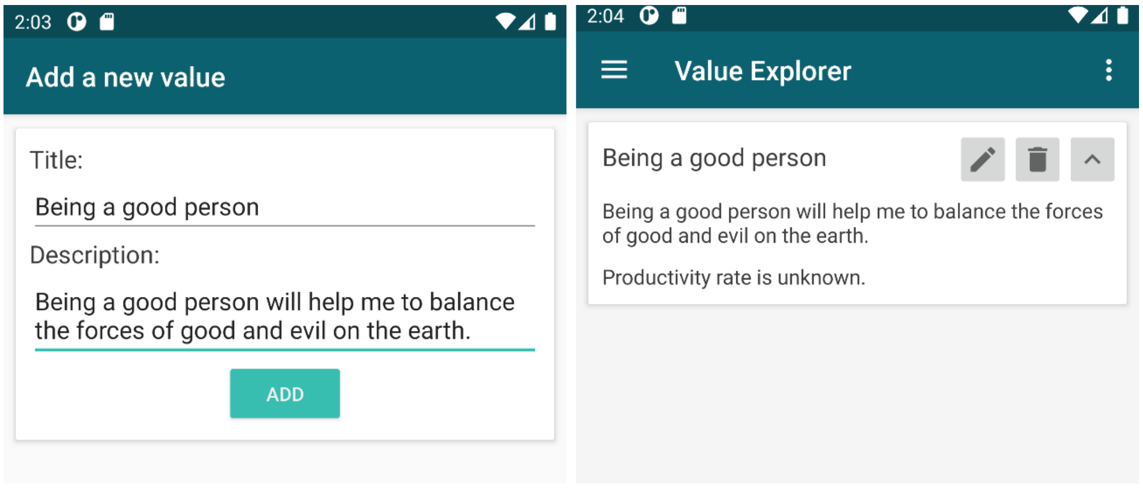
As it was described, the program parts are interconnected (see Core). By default, it is recommended for the user to start from the top page and define the values. After that to define goals, and then to make a schedule. To-do items can be defined when needed.

Apart from that, it is probable that user has some activities in life already. For any activity it should be defined what goals the user is trying to achieve. And for every goal that user has, it is recommended to also think about it for a while and decide what deeper meaning and value corresponds to the goal.

2.4.4 Example of using the application (default usage)

In this example, we have to look at the process of using the application from the perspective of a user:

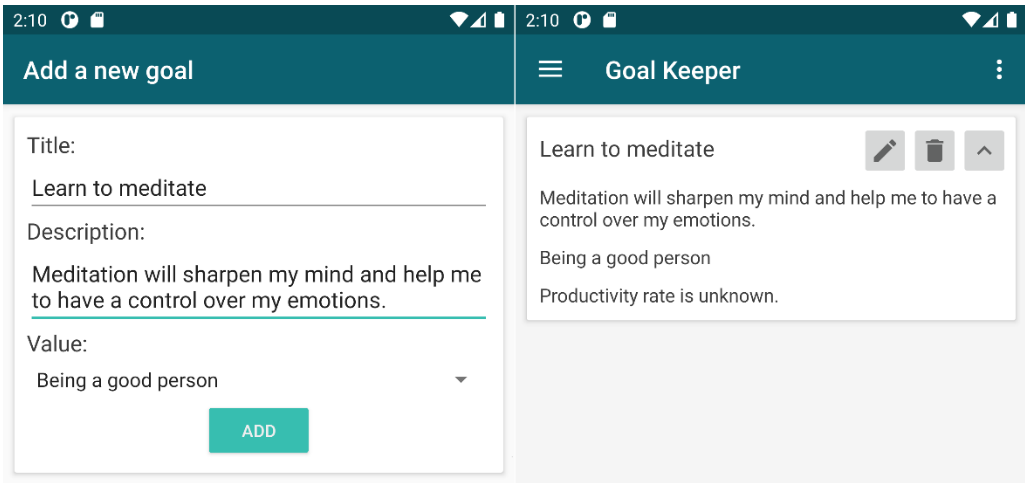
1. First, we have to define the value. Value should be the meaningful thing that we want to achieve. E.g., we want to become a good person. We can specify it so, as it is shown in Figure 2 (creating a new value).


2:03
Add a new value
Title:
Being a good person
Description:
Being a good person will help me to balance the forces of good and evil on the earth.
ADD

2:04
Value Explorer
Being a good person
Being a good person will help me to balance the forces of good and evil on the earth.
Productivity rate is unknown.

Figure 2 (creating a new value)

2. We have to specify the goal that will help us to become a good person. We can say for example that the goal is to learn how to meditate. This time we can do it as it is shown in Figure 3 (creating a new goal).

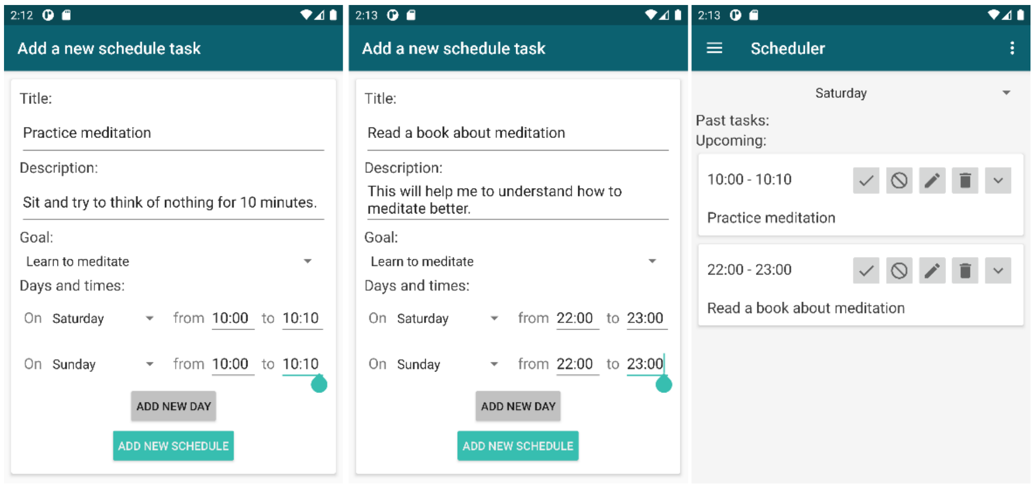

2:10
Add a new goal
Title:
Learn to meditate
Description:
Meditation will sharpen my mind and help me to have a control over my emotions.
Value:
Being a good person
ADD

2:10
Goal Keeper
Learn to meditate
Meditation will sharpen my mind and help me to have a control over my emotions.
Being a good person
Productivity rate is unknown.

Figure 3 (creating a new goal)

3. Now we should define the tasks that we can put into our schedule in order to learn how to meditate. These are 2-two possible examples:
   - Reading a book about meditation on weekends at 10 pm;
   - Practising meditation on weekends at 10 am.

   We shall do it as it is shown in Figure 4 (creating a new schedule task).


2:12

Add a new schedule task

Title:
Practice meditation

Description:
Sit and try to think of nothing for 10 minutes.

Goal:
Learn to meditate

Days and times:
On Saturday from 10:00 to 10:10
On Sunday from 10:00 to 10:10

ADD NEW DAY
ADD NEW SCHEDULE

2:13

Add a new schedule task

Title:
Read a book about meditation

Description:
This will help me to understand how to meditate better.

Goal:
Learn to meditate

Days and times:
On Saturday from 22:00 to 23:00
On Sunday from 22:00 to 23:00

ADD NEW DAY
ADD NEW SCHEDULE

2:13

Scheduler

Saturday

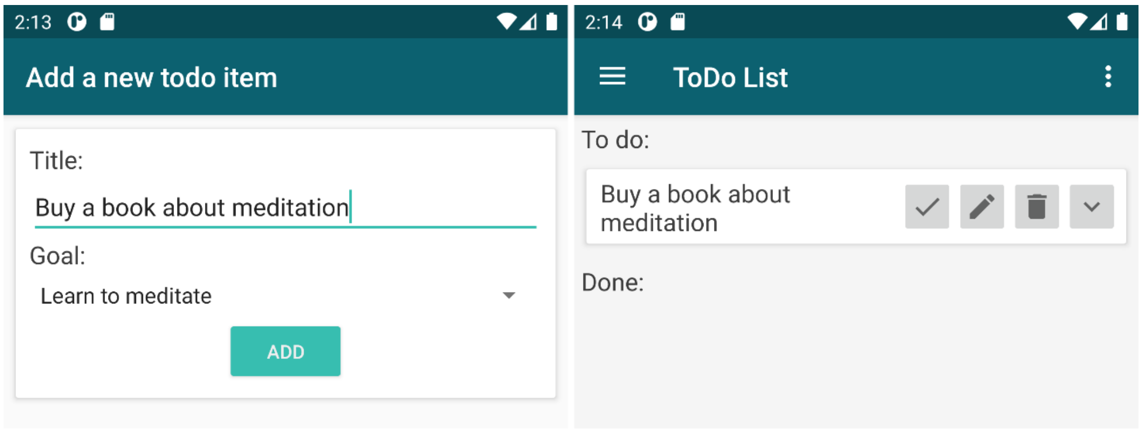
Past tasks:
Upcoming:

10:00 - 10:10
Practice meditation

22:00 - 23:00
Read a book about meditation

Figure 4 (creating a new schedule task)

4. Sudden events can force us to define the task without particular timing. E.g., we need to buy a book first, in order start reading it on weekends. Then we should add new to-do item under the name "Buy a meditation book", as it is shown in Figure 5 (creating a new to-do item).


2:13

Add a new todo item

Title:
Buy a book about meditation

Goal:
Learn to meditate

ADD

2:14

ToDo List

To do:
Buy a book about meditation

Done:

Figure 5 (creating a new to-do item)

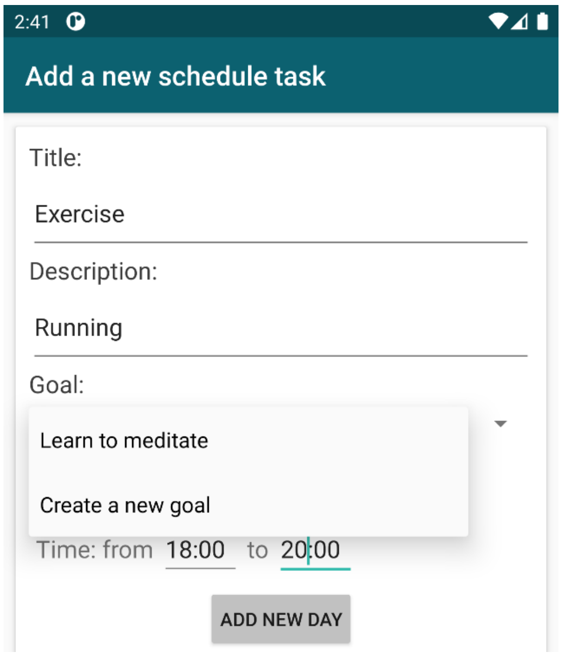
Now we can proceed with the created tasks.

2.4.5 Example of using the application (backward usage)

It might be that user already has a particular schedule task, or a to-do item that needs to be added to the application. But it might happen that user doesn’t have a particular goal for this item. In this case we can see another example of using the application, but here we are defining tasks backwards.

As it was done previously (see Example of using the application (default usage)), we will look at the usage of the application from the perspective of user:

1. There is a schedule activity that we have. E.g., we exercise on Saturday from 6 pm to 8 pm. We should add this activity to our application. As it can be seen in Figure 6 (creating a new schedule task), we don't have an appropriate goal for this task. In this case, we shall define a new one by using "Create a new goal" option.


2:41

Add a new schedule task

Title: Exercise

Description: Running

Goal:

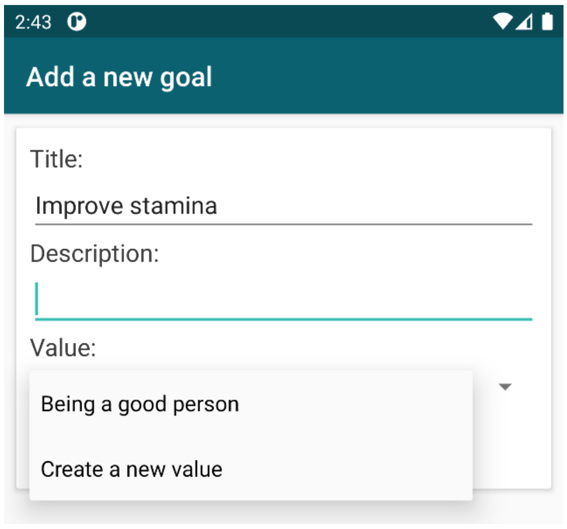
Learn to meditate

Create a new goal

Time: from 18:00 to 20:00

Figure 6 (creating a new schedule task)

2. We filled out fields for our goal, but we still have a problem. We don't have a proper value for it. Let's click on "Create a new value", as it is shown in Figure 7 (creating a new goal).


2:43

Add a new goal

Title:

Improve stamina

Description:

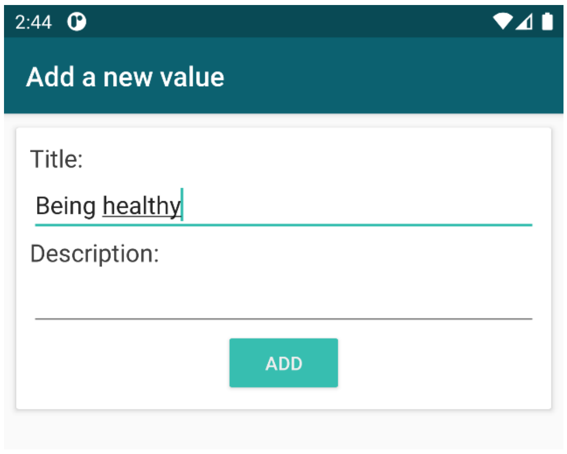
Value:

Being a good person

Create a new value

Figure 7 (creating a new goal)

3. Now we can actually specify the value we need, and then save it, as it is shown in Figure 8 (creating a new value).


2:44

Add a new value

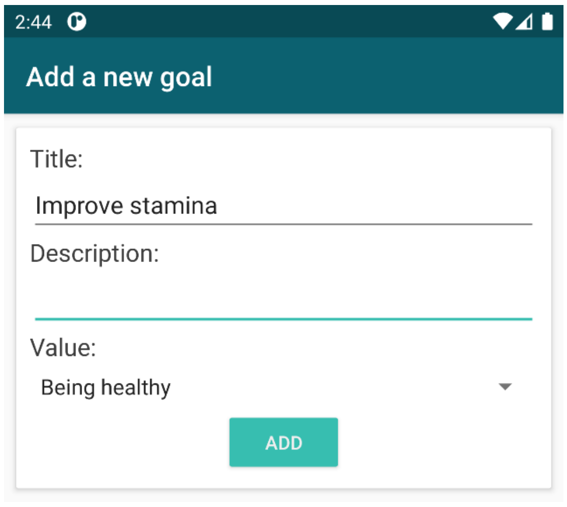
Title:

Being healthy

Description:

Figure 8 (creating a new value)

4. As it can be seen in Figure 9 (creating a new goal now is possible), the new value is in the list. We shall save the goal now.


2:44

Add a new goal

Title:

Improve stamina

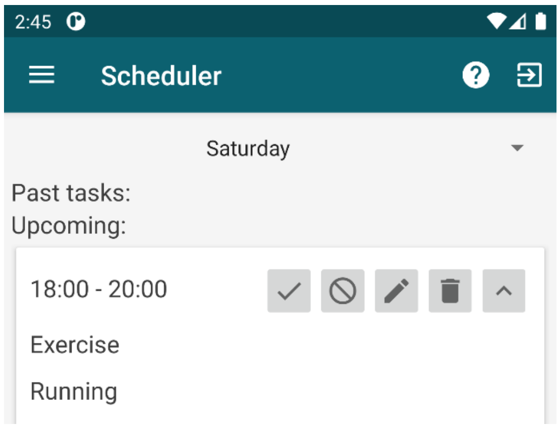
Description:

Value:

Being healthy

Figure 9 (creating a new goal now is possible)

5. We are able to add the schedule with the new goal. The result can be seen in Figure 10 (new schedule task is created).


2:45

Scheduler

Saturday

Past tasks:

Upcoming:

18:00 - 20:00

Exercise

Running

Figure 10 (new schedule task is created)

3 Developer Documentation

3.1 Required development tools

The application can be developed using Android Studio IDE. All the other tools can be downloaded inside Android Studio using Android Studio SDK. The list of tools to be installed:

- SDK Platform Android 11.0 (R) (API level 30);
- Android SDK Build-Tools 30.*;
- Android Emulator 30.* (this tool is used for testing purposes, instead of testing the application on real device);
- Android SDK Platform-Tools 30.*.

The important part is that we need our tools for Android API of version 30. Recommended subversion of any tool is the latest one. (e.g., Android SDK Build-Tools 30.0.2)

3.2 The design plan

The application was decided to be an android application. Considering that, it was decided to use Android Studio [3] to implement it. Android Studio has the advantages related to using Google services, and developing the application in Android Studio is an important aspect in choosing the tools that I will need to use.

The main thing which should be decided is where the data should be stored. Data of our application is complex and interconnected. Goals depend on values; schedule activities depend on goals; to-do items depend on goals. The database was needed to store this complex data.

It would not make sense to store the data on the phone, because devices are changing rapidly, and the user can have a new device every year. That means that it is useful to store the data on the internet.

As it was discussed in the beginning of the chapter, development was done in Android Studio. Android Studio can be easily connected to Firebase, because Firebase is also the product of Google. [4] The Firebase is the most popular choice for storing data in android application, and it means that in case any problem appears, usage of the search engine will provide a quick answer to it.

Other reasons to choose Firebase include:

- Support of real-time operations, instant synchronization of the data in the application;
- No need for a backend server;
- Analytics tools;
- Free to use for the purposes of our project.

Firebase also has different options for databases. My decision was to use Firestore - real-time database. [5] Real-time database is useful, because the application should update data elements in the real time as the information changes.

Using Firestore, data are kept as documents in different collections. There is one collection for values, one for goals, and one for the to-do list. Additionally, there are two collections for scheduling data (information about schedule task itself and information about particular events related to the schedules).

Information about users are also saved in the database.

More detailed description of the database will come later, when the architecture will be discussed in details. (See Database structure and Figure 17)

For the pages of values and goals I decided to use the same structure. User should see a list of items, each can be deleted or edited. User also should be able to add a new item to the list. (How the pages actually look like can be found in Figure 2 and Figure 3)

Page of scheduling (see Figure 4) is the most complicated page to implement. First, user should be able to choose between different days of the week. Second, every page of the week should contain the tasks. Every task should have the information if it was completed, missed, or neither of it (implemented via second database collection). One complicated part is related to creating a new schedule. The main part is similar to previous pages, but then we need to define days and times of tasks.

Page of to-do list is similar to the pages of values and goals. The only addition to that is that user, as in the schedule page, can define if the particular to-do item was completed. (See Figure 5)

Tracker page is taking an advantage of the second collection for scheduling (schedule events). Using algorithms, I extract information from the database and convert it to statistics.

For statistics, I decided to use the library with the possibility to draw graphs. For this application, I used only one graph - PieChart. [6]

Apart from that, account editing should be available. User should be able to register, log in, log out, and change the name.

As it has been discussed before (in the section Network), our database is a real-time database. When user makes any changes, the algorithm should be able to recognize that the data in the database was updated, and that it needs to be refreshed in the application.

Firestore is not very smooth and has some problems which are hard to avoid. They are usually related to the fact that the database is working in real-time, and that data extraction is asynchronous. That is the hard problem for the algorithms which will include nested loops in order to extract the data from the database.

3.3 Wireframe

The wireframe was created in order to show how the pages should look like. When user opens the application for the first time, the starting pages should be available. They are shown in the Figure 11. The main four pages are shown in Figure 12-15. Other pages can be seen in Figure 16.

3.3.1 Login and registration

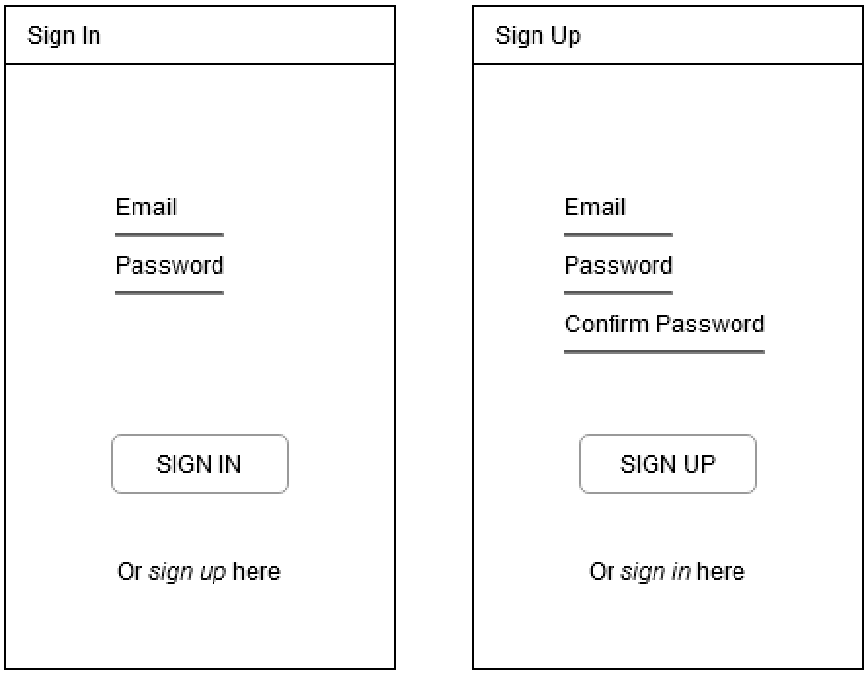


Figure 11 (login and registration wireframe)

3.3.2 Value Explorer pages

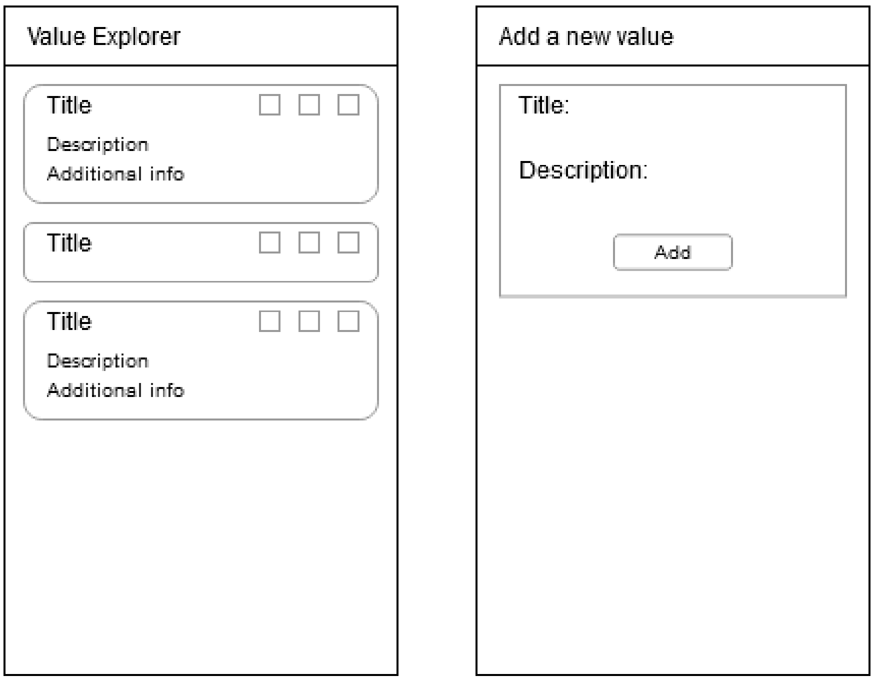


Figure 12 (value explorer wireframe)

3.3.3 Goal Keeper pages

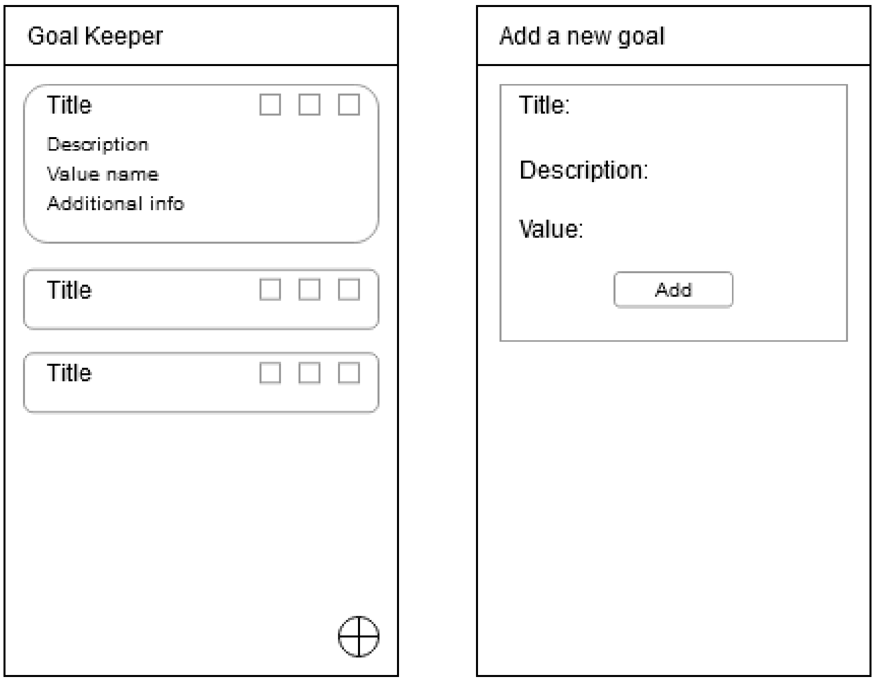


Figure 13 (goal keeper wireframe)

3.3.4 Scheduler pages

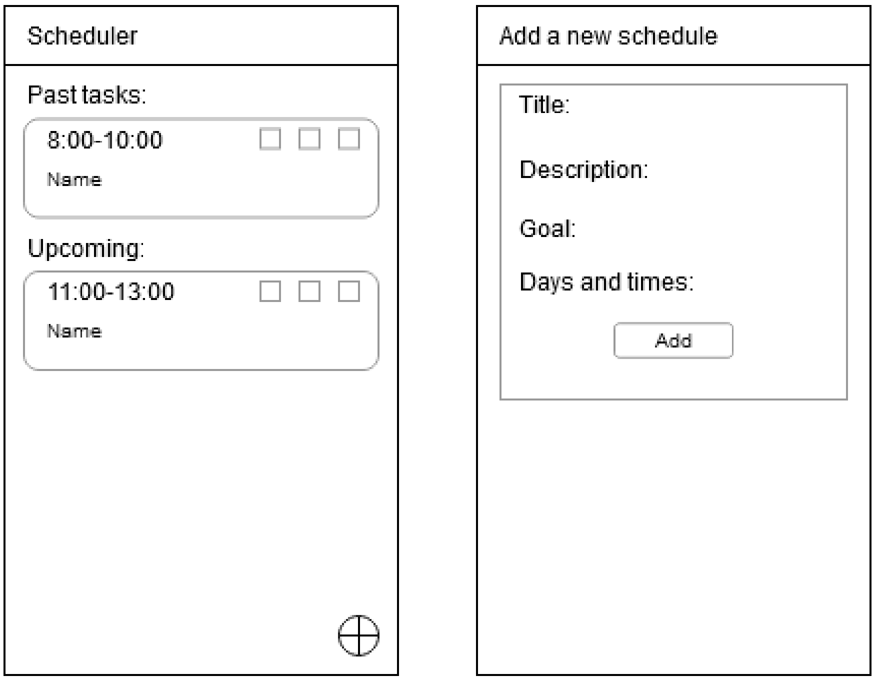


Figure 14 (scheduler wireframe)

3.3.5 ToDo List pages

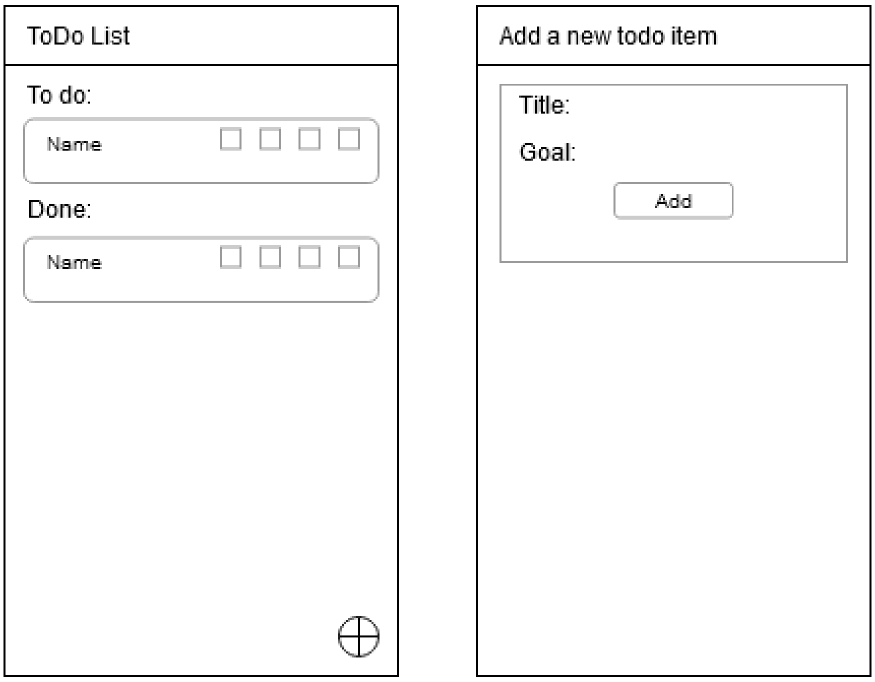
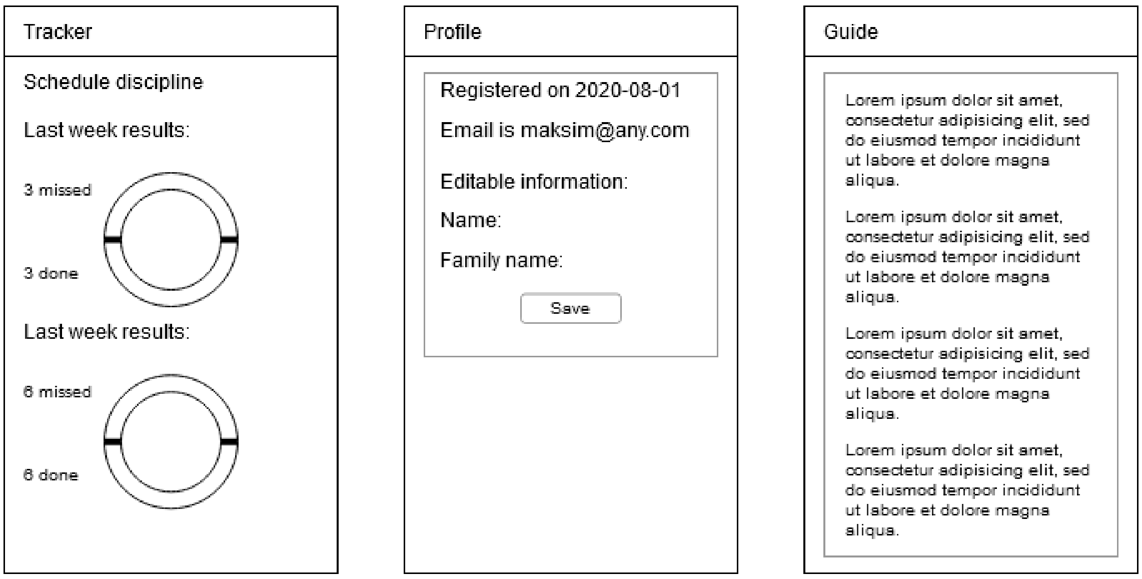


Figure 15 (todo list wireframe)

3.3.6 Other pages



Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Lorem ipsum dolor sit amet, consectetur adipisicing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.

Figure 16 (wireframe of other pages)

3.4 The detailed description of the used methods, the definitions of the used notions

3.4.1 Database

First, I had to decide where to store the database. The solution was to use Firebase. The solution was justified by the fact that Android Studio and Firebase are connected to each other, and developers made it possible to connect particular projects easily. More reasons on why I made this choice can be found in section The design plan.

Firebase had 2-two database choices: Cloud Firestore and Realtime Database. Cloud Firestore is a new version of Realtime Database. On the official website of Firebase, it can be found that Cloud Firestore used all the successful strategies of Realtime Database, but also added new features and a new more intuitive data model was developed. [7]

Realtime Database will become obsolete at some point of time, and the Cloud Firestore is a new standard. That is why I chose to use it.

3.4.2 Data manipulation

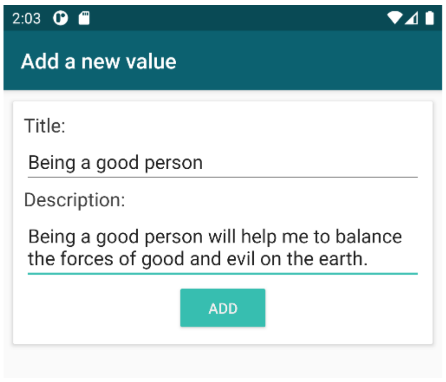
There are 4-four relevant pages to discuss here: values, goals, schedules, to-do list. All of them have their differences, but at the same time they share the same goal - to manipulate data. User should be able to delete, edit, add new, or list existing data.

Describing the pages one by one:

3.4.2.1 Values

When this page was created, the most important aspects of data manipulation had to be solved. All the other pages would only inherit this model of solving the problem, and then modify it.

The example of adding and listing can be seen in Figure 2 (creating a new value):


2:03

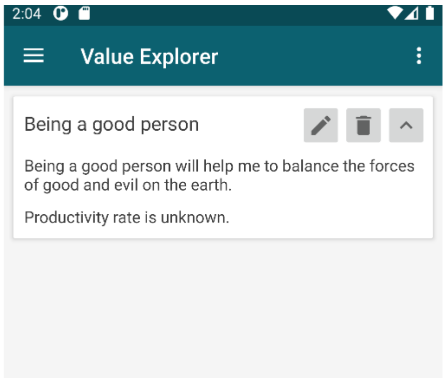
Add a new value

Title:

Being a good person

Description:

Being a good person will help me to balance the forces of good and evil on the earth.


2:04

Value Explorer

Being a good person

Being a good person will help me to balance the forces of good and evil on the earth.

Productivity rate is unknown.

Adding the data

I decided to use the "plus" button in the right bottom part of the screen for that. It is a default solution for android applications, and there was no reason to change it.

When the user clicks on that button, the new intent is being created. Intent is used to run new activity. This is the moment when the second page of the application opens, and it persists there until user closes it (by adding data).

When user opens the new activity, 2-two fields and one button appear: fields for title and description, and the button "Save". When user clicks on the button, data are sent to the Firestore, and then the activity closes.

Data visualization

If data are in database, visualization should be available for user. For that purpose, I added a new element in the values page - RecyclerView. [8]

RecyclerView is essentially a list element. It has its own custom format (the developer needs to define it). We can assign the ArrayList (of the particular data model that fits the format) to the RecyclerView. RecyclerView will be able to display the list to the user.

To define the first RecyclerView, I created a file recycler_value_explorer.xml. Inside this file, I, using standard XML guidelines, added particular elements to the RecyclerView. In essence, this file will define how one particular element of the list will look like. In this case, every element will have Title, Description, and 2-two buttons: Edit and Delete.

Connecting data to the RecyclerView

We get the data from our database, then store it inside an ArrayList. When we have a new type of data, we have to decide what class ArrayList should use. For values, I have defined a new class, which is stored under /Adapters/Data/ directory and has the name Value.java. This class stores the id of the value, userId, title, and description attributes of the value.

Using this data model, I created a new ArrayList, populated it with data from the database, and assigned it to a RecyclerView.

Deleting and editing

RecyclerView also has an adapter. Inside this adapter we can manipulate data and elements of the RecyclerView. For example, we can assign a ClickListener to our buttons.

In case of deleting, everything is straightforward, because we already know id of the value. The algorithm is following: by clicking the Delete button, we just search for the value with this id, and remove it from the database.

Editing is a bit more complicated. One of the solutions would be to create a new activity for editing. When we start it, populate it with existing data, and then make it possible to edit it.

I decided to use similar approach, but I don't create a new activity. We already have an activity for adding the value, which has the same fields. My algorithm for editing is following: when we click on Edit button, AddOrEdit activity is opened, which will be populated with existing data, and the button text will be changed from "Add" to "Edit".

The activity itself will be more complicated. It will receive information from intent about the origin of action and the id of the value. Activity will execute one of 2-two different things based on this information. If the action is for adding, it will add a new document to the collection. If the action is for editing, it will update the existing value using its id.

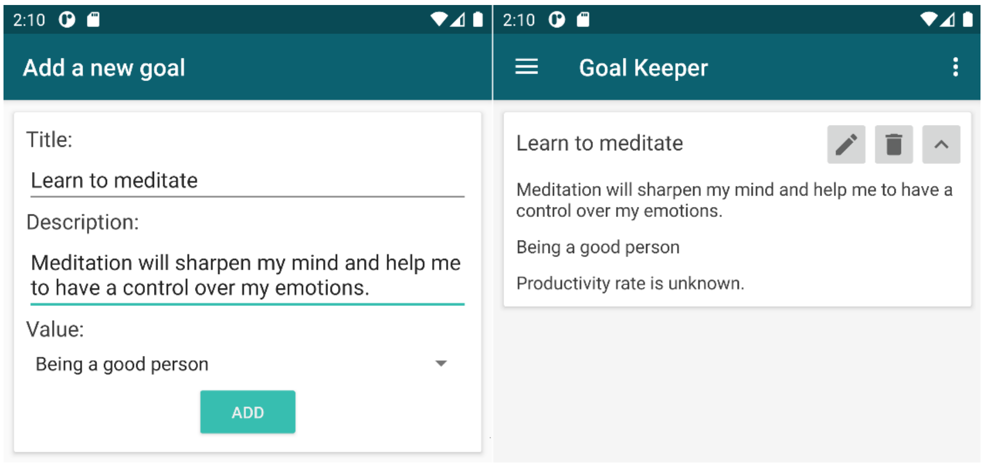
Summary

It was described how the system of data manipulation works. From now on, I will just reference the information that is already known. Information about recycler views and their adapters, array lists and corresponding data models, add/edit activities.

3.4.2.2 Goals

The Goal Keeper page has similar structure. The only difference is that the goal data model has additional field - value.

The example of listing and adding data can be seen in Figure 3:


2:10

Add a new goal

Title:

Learn to meditate

Description:

Meditation will sharpen my mind and help me to have a control over my emotions.

Value:

Being a good person

ADD

2:10

Goal Keeper

Learn to meditate

Meditation will sharpen my mind and help me to have a control over my emotions.

Being a good person

Productivity rate is unknown.

First, in the definition of data model I added an additional parameter - valueId. The same is added to the structure of the database collection for goals.

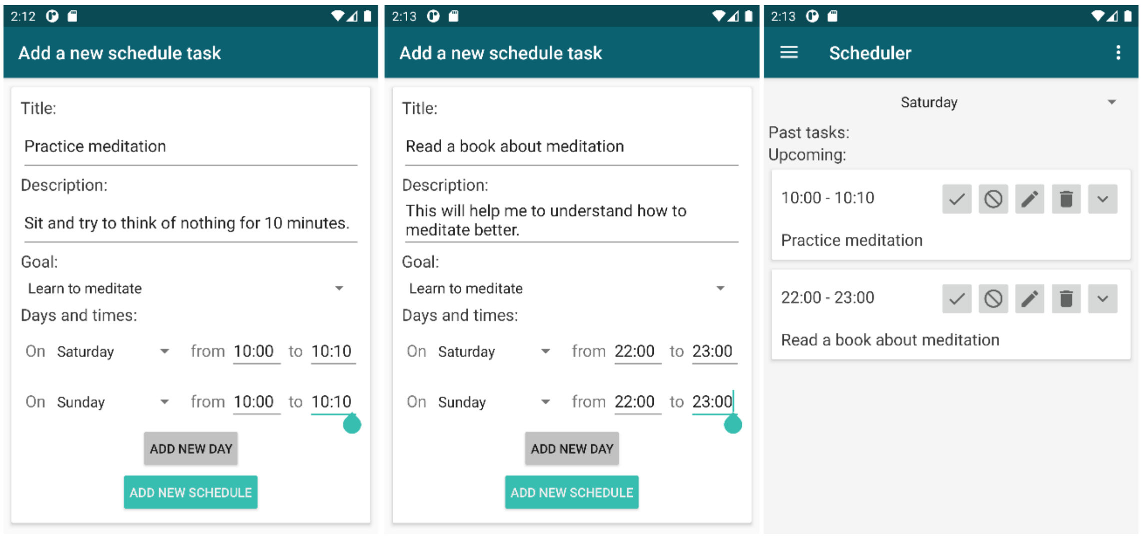
Inside the add/edit activity user should choose the value for the goal. I decided to use the dropdown menu for it.

My particular choice was Spinner android element. [9] Spinner element should not have any custom adapter in our case, because it takes only the string values. The collection of values is used to populate a new ArrayList. This ArrayList will be assigned to the Spinner. When the user picks any element from Spinner, we know the index of this element. Using index, the id of the value can be found.

3.4.2.3 Schedules

As it was discussed before, the Scheduler page is the most complicated page in our application. Using the same structure as value or goal pages, it adds few features which are hard to implement.

The example of adding and listing data can be seen in Figure 4:


Add a new schedule task

Title: Practice meditation

Description: Sit and try to think of nothing for 10 minutes.

Goal: Learn to meditate

Days and times:

On Saturday from 10:00 to 10:10

On Sunday from 10:00 to 10:10

Add a new schedule task

Title: Read a book about meditation

Description: This will help me to understand how to meditate better.

Goal: Learn to meditate

Days and times:

On Saturday from 22:00 to 23:00

On Sunday from 22:00 to 23:00

Scheduler

Saturday

Past tasks:

Upcoming:

10:00 - 10:10 Practice meditation

22:00 - 23:00 Read a book about meditation

Add/edit page

Apart from defining title, description, and the goal of the schedule task, we have to define days and times of the task.

If one task corresponded to only one day, that would be easy. I would just create 2-two EditText fields: finish and start time, and Spinner to choose the exact day.

But this is not how it should work. User should be able to manipulate the *list*. User should be able to add a new day into the list with the same 3 fields: finish, start, day. RecyclerView can be used again. And we use it exactly how we used it before. The new adapter will be called DaytimeAdapter, which will use Daytime data model. I decided to add a button "Add new day" to the layout, which will append a new default day into the list that the user can edit.

After we defined everything, we have to collect all of our data and send it to Firestore.

The data we send is complex. It has the arrays of daysOfTheWeek (of integer), startTime (of strings in the format "hh:mm"), and finishTime (of strings in the format "hh:mm"). For this purpose, I break down the Array of Daytime elements into the 3 arrays, and then send these arrays to the database.

Editing is a bit harder, because now we have to receive this complex data. When we receive the data, we will get (apart from title, description, and goal) 3 arrays of days, starting times, and finish times. To deal with the data, I convert the elements of these 3 arrays into one array of Daytime elements, assign this array to our RecyclerView. After that the editing should be possible.

Picking the other day of the schedule

This is the week schedule. User should be able to switch between different days of the week and see different tasks.

Dropdown menu in the form of Spinner element is used again. It will have 7 days to choose from, and user will be able to access the information about chosen day.

When we get the data about scheduled tasks from the database, we keep only the ones which have the corresponding day. After we pick a different day, the information should be updated.

Past and future tasks

There is nothing hard about listing the data here. We just need to use RecyclerView with title and time. But there is one small feature I decided to add that will make the life of the user easier: past and future tasks.

Every element of our recycler view has the start time and finish time. I made 2-two lists of elements: past tasks (finish time before the current time), and future tasks (start time after the current time).

Now the user will be able to understand which tasks are already in the past, and which are in the future.

Completing or missing the task

We should track our schedule activities. We should know where to get info from. For that I made a new database collection - scheduleActivityEvents.

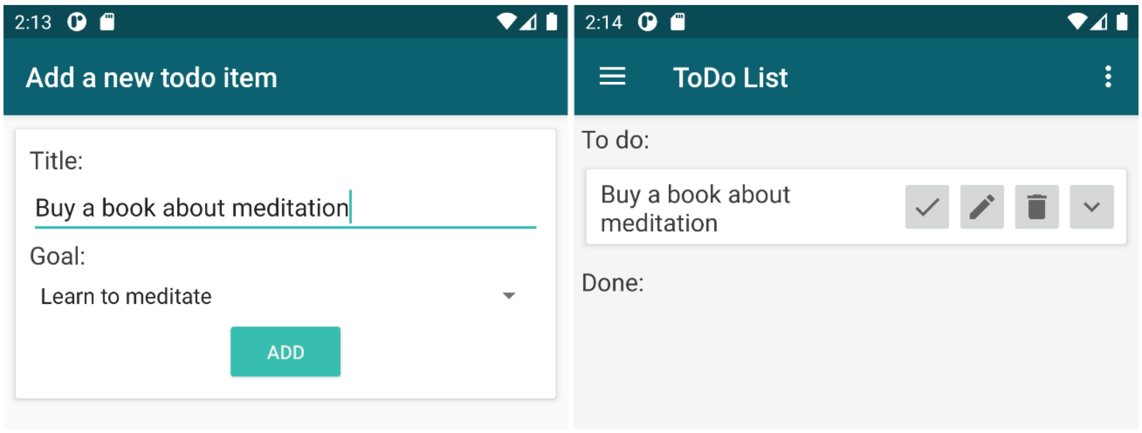
Every task can be set to be 'done' or 'missed'. If user clicks, for example, on 'done', then a new document is added to this collection: the document will collect basic information about task, date of completion, and new status.

When we list our tasks on the page, we also go through this collection. For every task we check if there is a document with corresponding date and corresponding taskId. If yes, then we will remove the buttons (done, missed) and just display the status of the task.

3.4.2.4 To-do

To-do page is very much like goal page, though there is a difference: if goal page had value in its data model, then to-do page has a goal instead of it.

Example can be seen in Figure 5:



Title:

Buy a book about meditation

Goal:

Learn to meditate

To do:

Buy a book about meditation

Done:

Apart from that, the goal can't be completed, but to-do item can be. For that to-do item also has a status: pending or completed.

I created 2-two recycler lists here: first for the to-do items which hasn't been completed yet, and the second one for completed ones.

For the items in the first list, we have a button "Done". When user clicks on this button, the status is changed to 'done', and the element goes to the second list.

3.4.2.5 Tracker

Tracker has one simple task: it should show to the user the percentage of completed schedule events on the periods of:

- Last week,
- Last month,
- Last year.

In order to do that, we go through all the existing events of the current user, and save them. Then we check the date of events. Based on that we make a statistic.

The hard part here is not the statistics itself. The hard part is showing these values. The plain text will not work. It is much better to use graphs to visualize the data.

In order to do it, I used the library called MPAndroidChart. [6] This library has the ability to use very different graphs, but my choice fell to PieChart.

In the tracker layout we can create the element of PieChart, and then in the activity we just assign data to that graph. I used this approach to show the statistics about 3-three periods.

3.4.2.6 User management

User is able to register a new account and log in into the existing account.

Apart from that, we have a page for profile settings. Here we can change the name of the user.

Also, when we open the left side menu, we can see the name and email of the current user.

3.4.3 The description of the logical and physical structure of the software

The project consists of 2-two parts: java files and XML layout files. Every activity is using a particular XML layout.

Java files include the data structure files, activities, and fragments.

The main activity, called MainActivity, is using fragments in order to switch pages.

Fragments are connected to the database in order to manipulate data. Usually, we get data from a database and put it into the arrays of particular data models. Apart from that, fragments can call a new activity, in order to edit or add a new data.

This section will consist of a description of the main structure elements:

3.4.3.1 Database structure

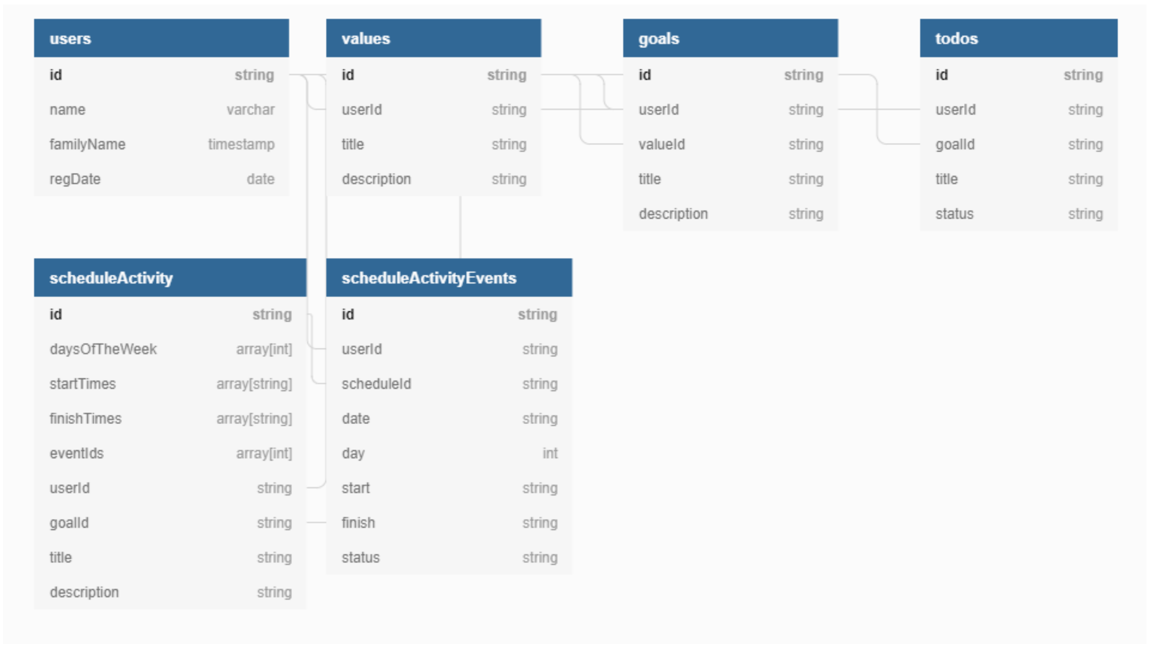


Figure 17 (database structure)

As you can see in Figure 17 (database structure), the main collection here is users. It is connected to every other collection. It was done in order to speed up the retrieving of the data for a particular user, because Firebase is not SQL database and does not support merge tables.

Pages of values, goals and to-do elements have their own collections. The page for scheduling has 2-two collections: for storing schedules themselves and registering events (set status to "done" or "missed" on particular schedule task).

The ID of the user is the email address.

3.4.3.2 Data models

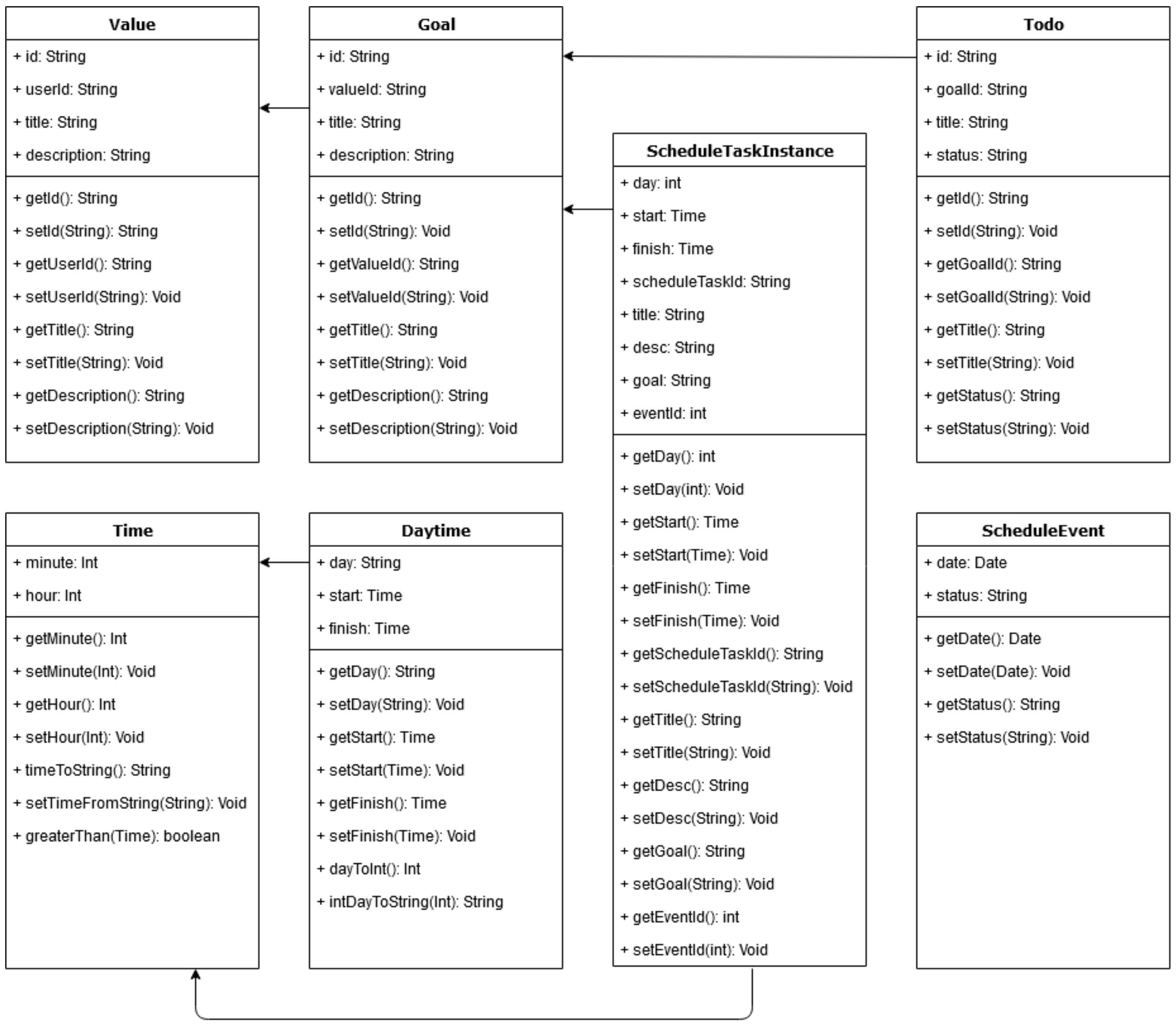
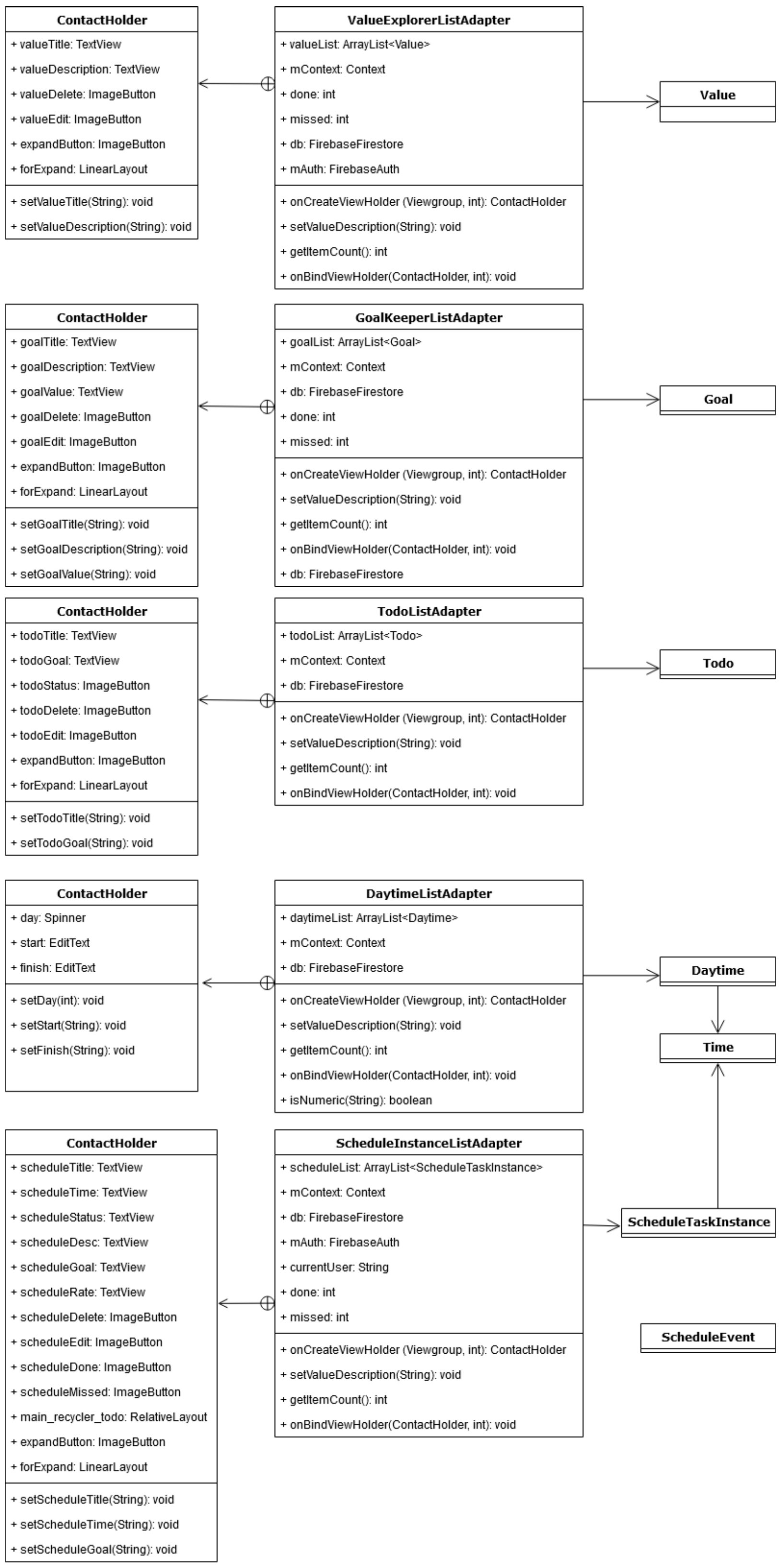


Figure 18 (the data models diagram)

In Figure 18 (the data models diagram) detailed description of all data models that are used in the project is described. They are not directly related to each other, except the fact that Daytime and ScheduleTaskInstance are using Time class.

Though, there are undirected relations among them. For example, class Goal has valueId parameter. This valueId shows which value is connected to the goal. The same kind of indirect relations can be seen in other data models too.

3.4.3.3 Adapters with data models



ContactHolder
+ valueTitle: TextView
+ valueDescription: TextView
+ valueDelete: ImageButton
+ valueEdit: ImageButton
+ expandButton: ImageButton
+ forExpand: LinearLayout
+ setValueTitle(String): void
+ setValueDescription(String): void
ValueExplorerListAdapter
+ valueList: ArrayList<Value>
+ mContext: Context
+ done: int
+ missed: int
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ onCreateViewHolder (Viewgroup, int): ContactHolder
+ setValueDescription(String): void
+ getItemCount(): int
+ onBindViewHolder(ContactHolder, int): void
Value
ContactHolder
+ goalTitle: TextView
+ goalDescription: TextView
+ goalValue: TextView
+ goalDelete: ImageButton
+ goalEdit: ImageButton
+ expandButton: ImageButton
+ forExpand: LinearLayout
+ setGoalTitle(String): void
+ setGoalDescription(String): void
+ setGoalValue(String): void
GoalKeeperListAdapter
+ goalList: ArrayList<Goal>
+ mContext: Context
+ db: FirebaseFirestore
+ done: int
+ missed: int
+ onCreateViewHolder (Viewgroup, int): ContactHolder
+ setValueDescription(String): void
+ getItemCount(): int
+ onBindViewHolder(ContactHolder, int): void
+ db: FirebaseFirestore
Goal
ContactHolder
+ todoTitle: TextView
+ todoGoal: TextView
+ todoStatus: ImageButton
+ todoDelete: ImageButton
+ todoEdit: ImageButton
+ expandButton: ImageButton
+ forExpand: LinearLayout
+ setTodoTitle(String): void
+ setTodoGoal(String): void
TodoListAdapter
+ todoList: ArrayList<Todo>
+ mContext: Context
+ db: FirebaseFirestore
+ onCreateViewHolder (Viewgroup, int): ContactHolder
+ setValueDescription(String): void
+ getItemCount(): int
+ onBindViewHolder(ContactHolder, int): void
Todo
ContactHolder
+ day: Spinner
+ start: EditText
+ finish: EditText
+ setDay(int): void
+ setStart(String): void
+ setFinish(String): void
DaytimeListAdapter
+ daytimeList: ArrayList<Daytime>
+ mContext: Context
+ db: FirebaseFirestore
+ onCreateViewHolder (Viewgroup, int): ContactHolder
+ setValueDescription(String): void
+ getItemCount(): int
+ onBindViewHolder(ContactHolder, int): void
+ isNumeric(String): boolean
Daytime
Time
ContactHolder
+ scheduleTitle: TextView
+ scheduleTime: TextView
+ scheduleStatus: TextView
+ scheduleDesc: TextView
+ scheduleGoal: TextView
+ scheduleRate: TextView
+ scheduleDelete: ImageButton
+ scheduleEdit: ImageButton
+ scheduleDone: ImageButton
+ scheduleMissed: ImageButton
+ main_recycler_todo: RelativeLayout
+ expandButton: ImageButton
+ forExpand: LinearLayout
+ setScheduleTitle(String): void
+ setScheduleTime(String): void
+ setScheduleGoal(String): void
ScheduleInstanceListAdapter
+ scheduleList: ArrayList<ScheduleTaskInstance>
+ mContext: Context
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ done: int
+ missed: int
+ onCreateViewHolder (Viewgroup, int): ContactHolder
+ setValueDescription(String): void
+ getItemCount(): int
+ onBindViewHolder(ContactHolder, int): void
ScheduleTaskInstance
ScheduleEvent

Figure 19 (adapters with data models)

One more separate diagram can be seen in Figure 19 (adapters with data models). The same data models are used here (as in Figure 18 (the data models diagram)), but without description, as the descriptions were given already in the previous diagram.

This diagram focuses more on the adapters, and how they are connected to data models. Adapters are responsible for specifying the behaviour of recycler views.

Adapters also have inner classes called ContactHolder. Contact holders are responsible for holding important elements from the layout of recycler views, and giving the ability to the user to set or get values of the elements.

3.4.3.4 Fragments with adapters

In Figure 20 (fragments with adapters) we can see the fragments, alongside with data models and adapters. Again, the latter elements were already described in details above (see Figure 19), and in the next diagram we omit the details.

Every fragment is using separate adapter for recycler view.

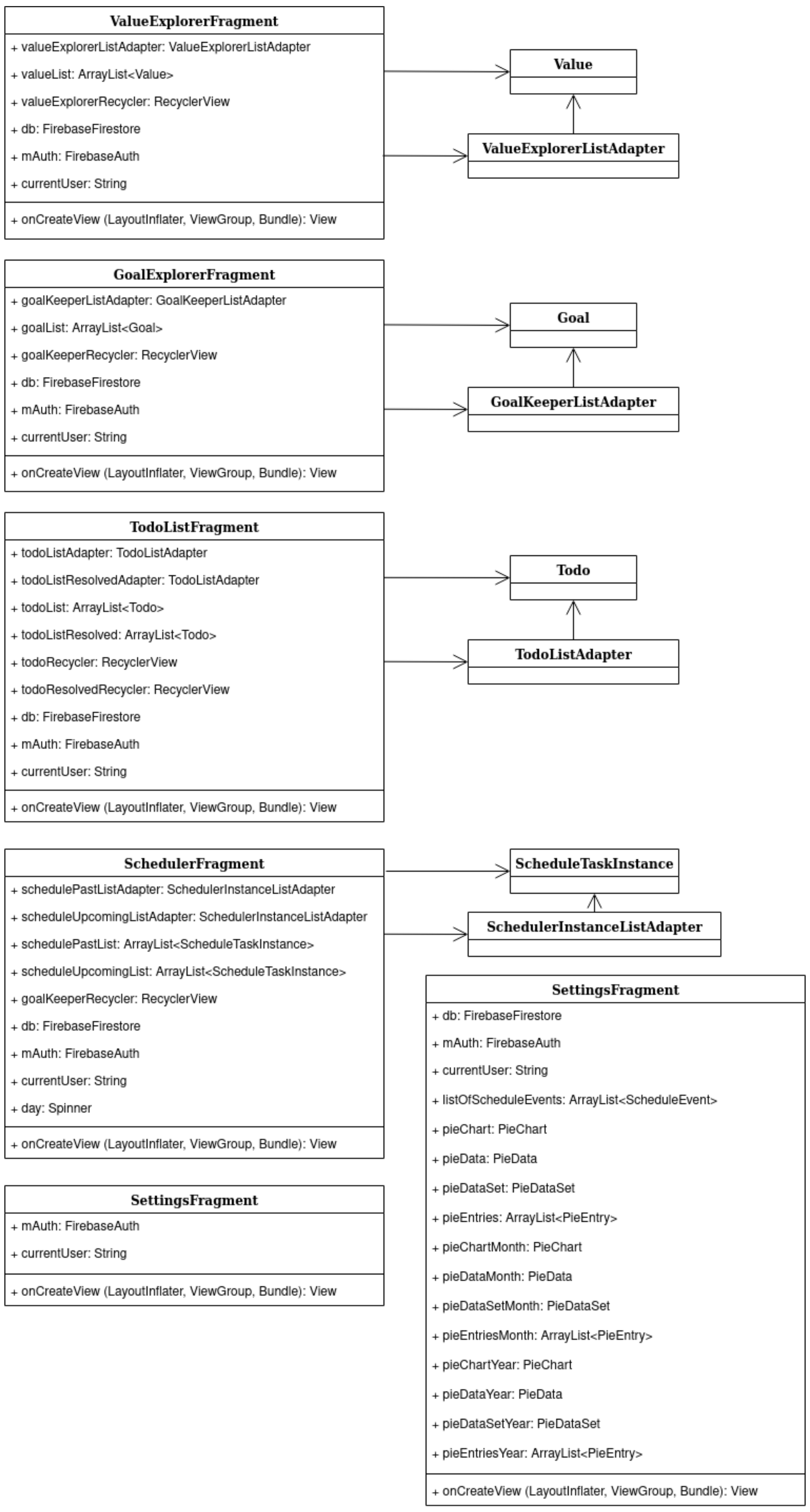

ValueExplorerFragment
+ valueExplorerListAdapter: ValueExplorerListAdapter
+ valueList: ArrayList<Value>
+ valueExplorerRecycler: RecyclerView
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View
Value
ValueExplorerListAdapter
GoalExplorerFragment
+ goalKeeperListAdapter: GoalKeeperListAdapter
+ goalList: ArrayList<Goal>
+ goalKeeperRecycler: RecyclerView
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View
Goal
GoalKeeperListAdapter
TodoListFragment
+ todoListAdapter: TodoListAdapter
+ todoListResolvedAdapter: TodoListAdapter
+ todoList: ArrayList<Todo>
+ todoListResolved: ArrayList<Todo>
+ todoRecycler: RecyclerView
+ todoResolvedRecycler: RecyclerView
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View
Todo
TodoListAdapter
SchedulerFragment
+ schedulePastListAdapter: SchedulerInstanceListAdapter
+ scheduleUpcomingListAdapter: SchedulerInstanceListAdapter
+ schedulePastList: ArrayList<ScheduleTaskInstance>
+ scheduleUpcomingList: ArrayList<ScheduleTaskInstance>
+ goalKeeperRecycler: RecyclerView
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ day: Spinner
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View
ScheduleTaskInstance
SchedulerInstanceListAdapter
SettingsFragment
+ mAuth: FirebaseAuth
+ currentUser: String
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View
SettingsFragment
+ db: FirebaseFirestore
+ mAuth: FirebaseAuth
+ currentUser: String
+ listOfScheduleEvents: ArrayList<ScheduleEvent>
+ pieChart: PieChart
+ pieData: PieData
+ pieDataSet: PieDataSet
+ pieEntries: ArrayList<PieEntry>
+ pieChartMonth: PieChart
+ pieDataMonth: PieData
+ pieDataSetMonth: PieDataSet
+ pieEntriesMonth: ArrayList<PieEntry>
+ pieChartYear: PieChart
+ pieDataYear: PieData
+ pieDataSetYear: PieDataSet
+ pieEntriesYear: ArrayList<PieEntry>
+ onCreateView (LayoutInflater, ViewGroup, Bundle): View

Figure 20 (fragments with adapters)

3.4.3.5 AddEdit pages

As it can be seen in Figure 21 (add/edit pages), user can start a new AddEdit activity from any of 4-four main fragments by either attempting to add a new element, or to edit the existing element. Pages are using EditText and Spinner fields in order to indicate the input for adding or editing.

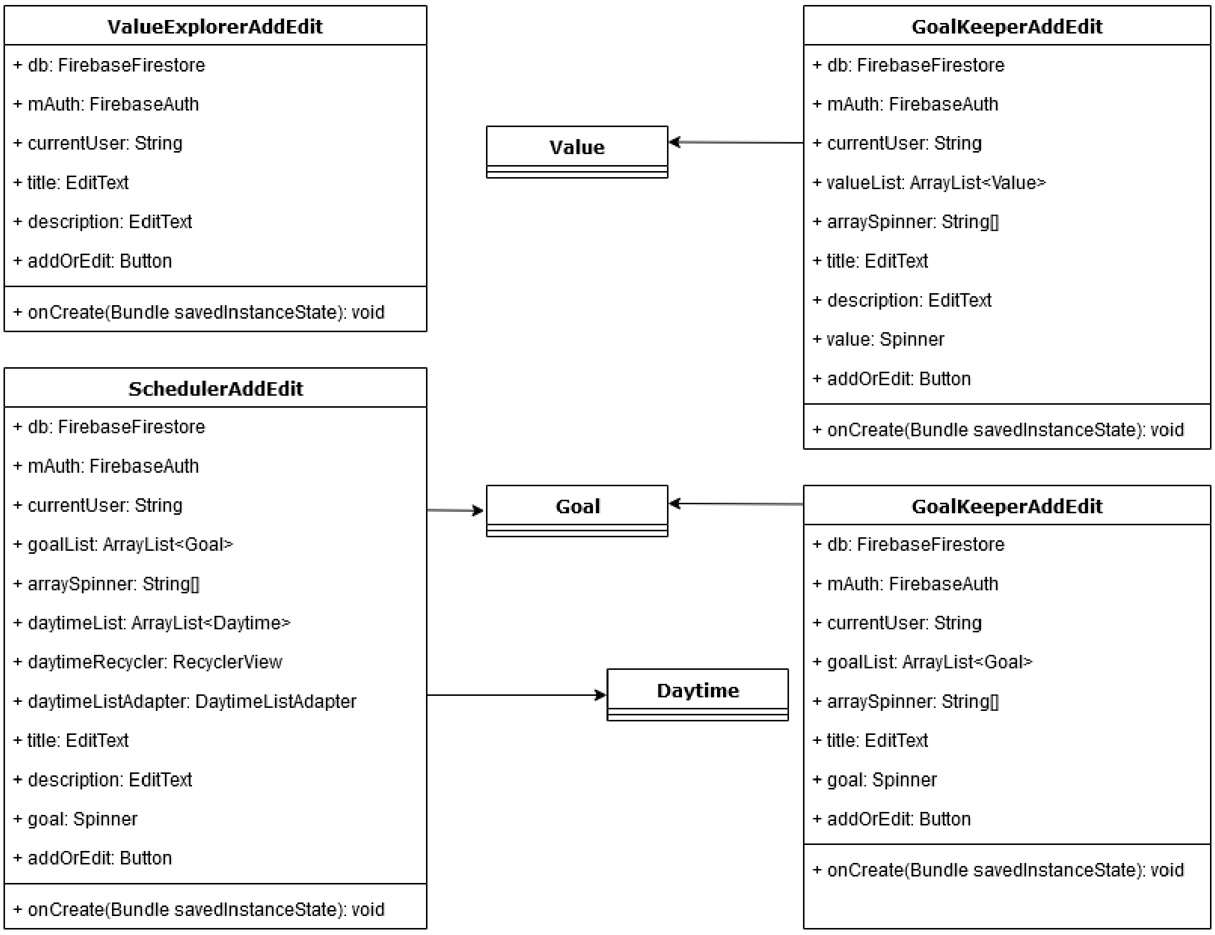


Figure 21 (add/edit pages)

3.4.3.6 Layout (main part)

All the layout files are located in the layout folder, they have XML extension. We can see the main layout structure in Figure 22 (main layout). MainActivity.java is using activity_main as the layout. Activity_main is using app_bar_main in order to show the AppBarLayout and content_main file. Content_main layout file is used in order to show different fragments.

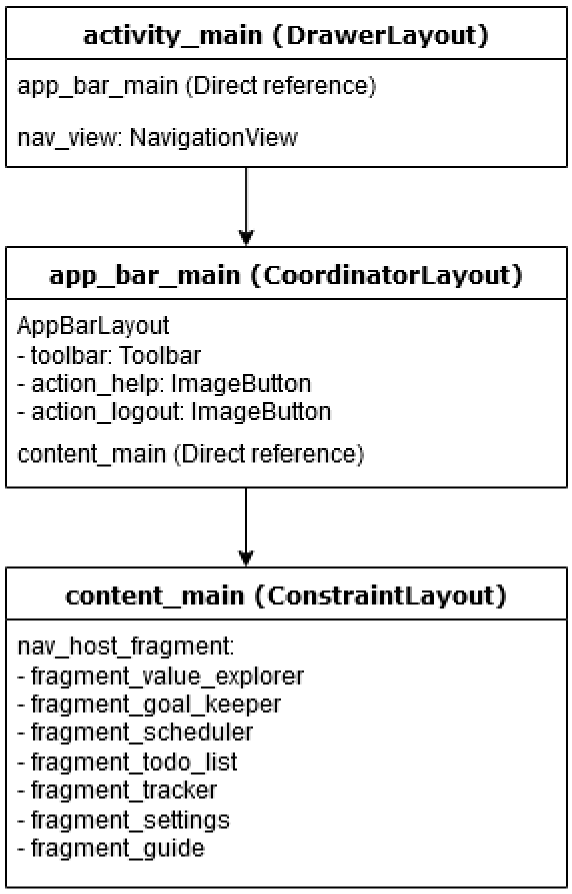

activity_main (DrawerLayout)
app_bar_main (Direct reference)
nav_view: NavigationView
app_bar_main (CoordinatorLayout)
AppBarLayout
- toolbar: Toolbar
- action_help: ImageButton
- action_logout: ImageButton
content_main (Direct reference)
content_main (ConstraintLayout)
nav_host_fragment:
- fragment_value_explorer
- fragment_goal_keeper
- fragment_scheduler
- fragment_todo_list
- fragment_tracker
- fragment_settings
- fragment_guide

Figure 22 (main layout)

3.4.3.7 Layout (fragments)

In Figure 23 (fragments layout) we can see details of the structure of fragment layouts. Usually, our fragment layouts contain recycler view layouts. And recycler view layouts are managed by recycler adapters, which are used by fragment classes.

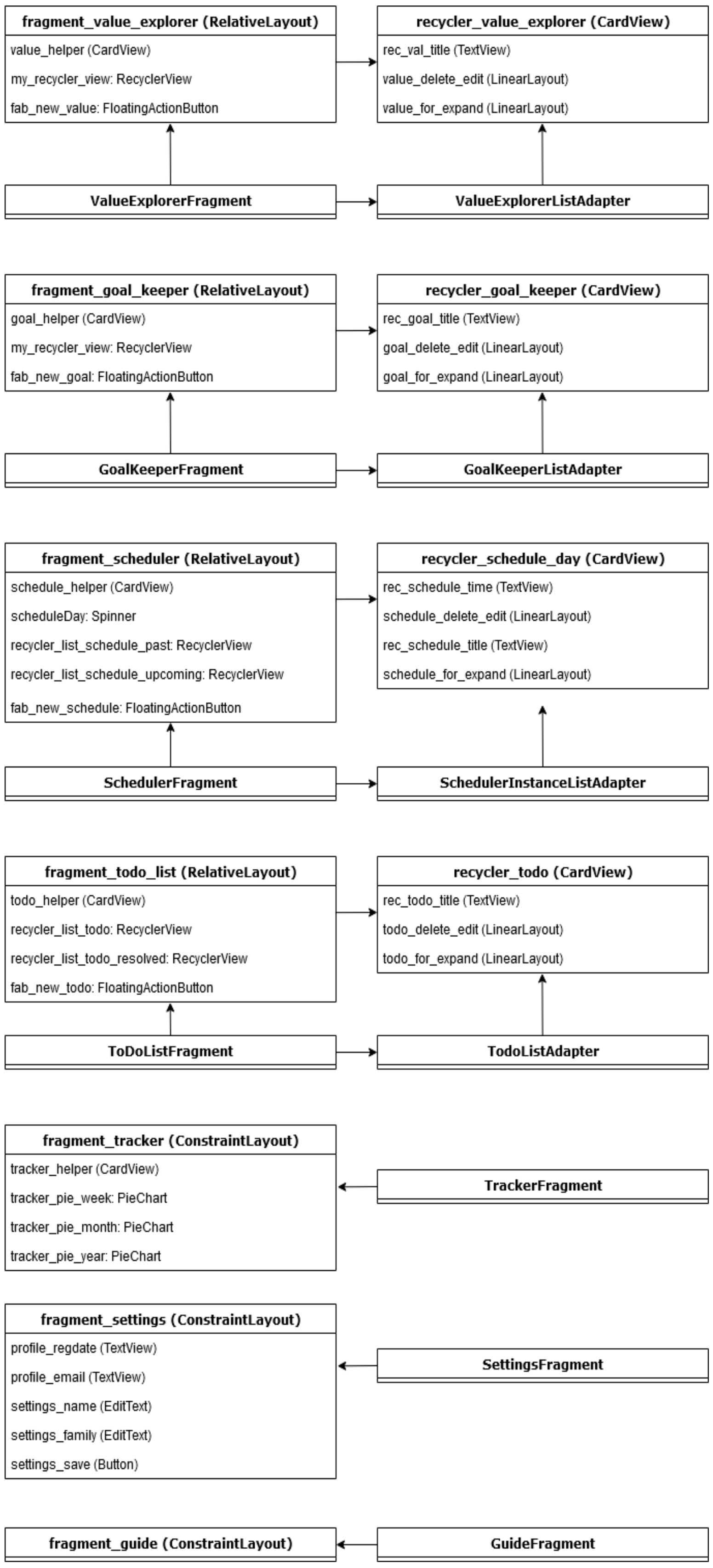

fragment_value_explorer (RelativeLayout)
value_helper (CardView)
my_recycler_view: RecyclerView
fab_new_value: FloatingActionButton
recycler_value_explorer (CardView)
rec_val_title (TextView)
value_delete_edit (LinearLayout)
value_for_expand (LinearLayout)
ValueExplorerFragment
ValueExplorerListAdapter
fragment_goal_keeper (RelativeLayout)
goal_helper (CardView)
my_recycler_view: RecyclerView
fab_new_goal: FloatingActionButton
recycler_goal_keeper (CardView)
rec_goal_title (TextView)
goal_delete_edit (LinearLayout)
goal_for_expand (LinearLayout)
GoalKeeperFragment
GoalKeeperListAdapter
fragment_scheduler (RelativeLayout)
schedule_helper (CardView)
scheduleDay: Spinner
recycler_list_schedule_past: RecyclerView
recycler_list_schedule_upcoming: RecyclerView
fab_new_schedule: FloatingActionButton
recycler_schedule_day (CardView)
rec_schedule_time (TextView)
schedule_delete_edit (LinearLayout)
rec_schedule_title (TextView)
schedule_for_expand (LinearLayout)
SchedulerFragment
SchedulerInstanceListAdapter
fragment_todo_list (RelativeLayout)
todo_helper (CardView)
recycler_list_todo: RecyclerView
recycler_list_todo_resolved: RecyclerView
fab_new_todo: FloatingActionButton
recycler_todo (CardView)
rec_todo_title (TextView)
todo_delete_edit (LinearLayout)
todo_for_expand (LinearLayout)
ToDoListFragment
TodoListAdapter
fragment_tracker (ConstraintLayout)
tracker_helper (CardView)
tracker_pie_week: PieChart
tracker_pie_month: PieChart
tracker_pie_year: PieChart
TrackerFragment
fragment_settings (ConstraintLayout)
profile_regdate (TextView)
profile_email (TextView)
settings_name (EditText)
settings_family (EditText)
settings_save (Button)
SettingsFragment
fragment_guide (ConstraintLayout)
GuideFragment

Figure 23 (fragments layout)

3.4.3.8 Layout (AddEdit)

The similar structure, as with the adapter layouts. Though, as it can be seen in Figure 24 (add/edit layout), the structure of this layout is simpler because most of the AddEdit pages don't use recycler views.

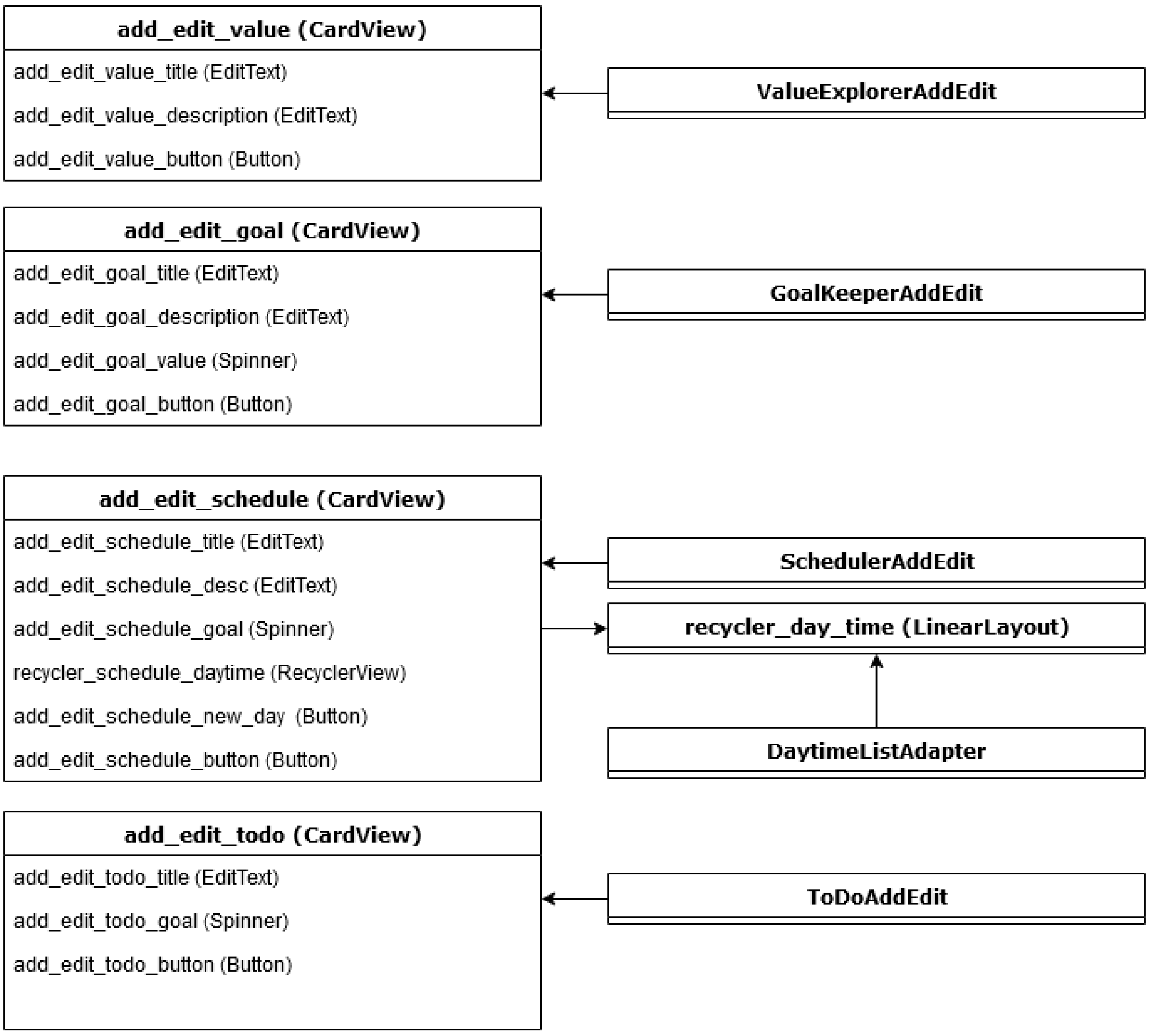

add_edit_value (CardView)
add_edit_value_title (EditText)
add_edit_value_description (EditText)
add_edit_value_button (Button)
ValueExplorerAddEdit
add_edit_goal (CardView)
add_edit_goal_title (EditText)
add_edit_goal_description (EditText)
add_edit_goal_value (Spinner)
add_edit_goal_button (Button)
GoalKeeperAddEdit
add_edit_schedule (CardView)
add_edit_schedule_title (EditText)
add_edit_schedule_desc (EditText)
add_edit_schedule_goal (Spinner)
recycler_schedule_daytime (RecyclerView)
add_edit_schedule_new_day (Button)
add_edit_schedule_button (Button)
SchedulerAddEdit
recycler_day_time (LinearLayout)
DaytimeListAdapter
add_edit_todo (CardView)
add_edit_todo_title (EditText)
add_edit_todo_goal (Spinner)
add_edit_todo_button (Button)
ToDoAddEdit

Figure 24 (add/edit layout)

3.5 Testing plan

The testing in our case will consist of 2-two parts: unit and manual testing.

3.5.1 Unit testing

Unit testing will cover the most complicated methods:

- Daytime objects manipulation (dayToInt(), intDayToString()),
- Time object manipulation (timeToString(), setTimeFromString(), greaterThan()),
- DaytimeListAdapter (isNumeric()).

3.5.2 Manual testing

Manual testing will consist of 2-two parts: independent testing and dependent testing.

3.5.2.1 Independent testing

Here we will have few independent test cases. Each of the test cases are not strongly related to the other ones.

3.5.2.2 Dependent testing

In this section we will have more complicated test cases, which will show the interconnection between different pages. E.g., creating a value and then navigating to Goal Keeper page in order to create the goal with the previously created value.

3.6 Testing execution

3.6.1 Unit testing

3.6.1.1 Daytime object manipulation

Method to test: dayToInt().

Location:	adapters/java/Daytime.java.
Description:	Class Daytime has a String variable called “day”. The method converts day from string to integer. E.g., from “Monday” to 0.

As a developer, I want to test the dayToInt() method.

Given:	object of the class Daytime with variable day that equals “Monday”,
When:	we call the dayToInt() method,
Then:	the method should return 0.

Given:	object of the class Daytime with variable day that equals “Wednesday”,
When:	we call the dayToInt() method,
Then:	the method should return 2.

Given:	object of the class Daytime with variable day that equals “Something else”,
When:	we call the dayToInt() method,
Then:	the method should return 7.

Method to test: intDayToString(int day).

Location:	adapters/java/Daytime.java.
Description:	reverse of dayToInt().

As a developer, I want to test intDayToString(int day) method.

Given:	object of the class Daytime with free variable **day** which equals 0,
When:	we call the intDayToString(day) method,
Then:	the method should return “Monday”.

Given:	object of the class Daytime with free variable **day** which equals 2,
When:	we call the intDayToString(day) method,
Then:	the method should return “Wednesday”.

Given:	object of the class Daytime with free variable **day** which equals 6,
When:	we call the intDayToString(day) method,
Then:	the method should return “Sunday”.

Given:	object of the class Daytime with free variable **day** which equals 123,
When:	we call the intDayToString(day) method,
Then:	the method should return “Error”.

3.6.1.2 Time objects manipulation

Method to test: timeToString().

Location:	adapters/java/Time.java.
Description:	Class Daytime has 2-two integer variables: hour and minute. The method converts them to string. E.g., 10 and 2 converts to “10:02”.

As a developer, I want to test timeToString () method.

Given:	object of the class Time with variables hour which equals 5, and minute which equals 10,
When:	we call the timeToString() method,
Then:	the method should return “05:10”.

Given:	object of the class Time with variables hour which equals 23, and minute which equals 0,
When:	we call the timeToString() method,
Then:	the method should return “23:00”.

Method to test: setTimeFromString(String time)

Location: adapters/java/Time.java.

Description: Class Daytime has 2-two integer variables: hour and minute. The method takes a string in a format of "hh:mm", converts it to 2-two integer values of hour and minute, and assign them to the object of the class.

As a developer, I want to test setTimeFromString(String time) method.

Given: object of the class Time with variables minute and hour,

When: we call the setTimeFromString("05:10") method,

Then: the object's variables minute and hour now should be equal to 5 and 10 respectively.

Given: object of the class Time with variables minute and hour,

When: we call the setTimeFromString("23:05") method,

Then: the object's variables minute and hour now should be equal to 23 and 5 respectively.

Method to test: greaterThan(Time time).

Location: adapters/java/Time.java.

Description: takes an object of type Time, and compares it to the initial object of Time.

As a developer, I want to test greaterThan(Time time) method.

Given: object of the class Time with variables hour and minute that are equal to 10 and 5 respectively,

When: we call the greaterThan(new Time(10, 6)) method,

Then:	the method should return false.

Given:	object of the class Time with variables hour and minute that are equal to 10 and 5 respectively,
When:	we call the greaterThan(new Time(10, 5)) method,
Then:	the method should return false.

Given:	object of the class Time with variables hour and minute that are equal to 10 and 5 respectively,
When:	we call the greaterThan(new Time(9, 6)) method,
Then:	the method should return true.

3.6.1.3 Manipulation of DaytimeListAdapter

Method to test: isNumeric (String str).

Location:	adapters/java/Time.java.
Description:	takes a string value and checks that it can be converted to integer value.

As a developer, I want to test isNumeric(String str) method.

Given:	the method isNumeric(String str),
When:	we call isNumeric (“10”),
Then:	the method should return true.

Given:	the method isNumeric(String str),
When:	we call isNumeric (“a”),
Then:	the method should return false.

3.6.2 Manual testing

3.6.2.1 Independent testing

As a user, I want to register and log in into created account.

Given: I am not logged in,

When: I execute the following actions:

1. Open the application
2. Click on "sign up"
3. Enter email as manual_testing@gmail.com, password as "123456", confirm the password, and click on "Sign up"
4. On the right top corner see the sign of logging out. Click on that.
5. Sign in page should appear. Enter the previous values and click "SIGN IN",

Then: all the action should be successful and I should be logged in.

As a user, I want to add a new value.

Given: I am logged in, the page of Value explorer is open,

When: I click on "+", specify title and description, click "Save",

Then: the new value should be created.

As a user, I want to edit the value.

Given: I am logged in, the page of Value explorer is open, there is existing value,

When: I click on edit button of the value, edit title and description to "Test value" and "Test description", click on button "Save",

Then: the window for editing should be closed. You should see the new title. If you expand the value, you should see the new description too.

As a user, I want to delete the value.

Given:	I am logged in, the page of Value explorer is open, there is existing value,
When:	I click on delete button of the value
Then:	value should be deleted.

As a user, I want to add a new goal.

Given:	I am logged in, the page of Goal keeper is open, at least one value exists,
When:	I click on "+", specify title, description, value, click "Save",
Then:	the new goal should be created.

Given:	I am logged in, the page of Goal keeper is open, there are no values
When:	I click on "+", specify title, description, *using spinner create a new value by clicking on the button "Create a new value",* click "Save",
Then:	the new goal should be created.

As a user, I want to edit the goal.

Given:	I am logged in, the page of Goal keeper is open, there is existing goal,
When:	I click on edit button of the goal, edit title, description and/or value, click on button "Save",
Then:	the window for editing should be closed. You should see the new title. If you expand the goal, you should see the new description and/or new value too.

As a user, I want to delete the goal.

Given:	I am logged in, the page of Goal keeper is open, there is existing goal,
When:	I click on delete button of the goal
Then:	goal should be deleted.

As a user, I want to add a new schedule.

Given: I am logged in, the page of Scheduler is open, at least one goal exists,

When: I click on “+”, specify title, description, goal, *days and times*, click “Save”,

Then: new schedule task should appear on days and times I specify.

Given: I am logged in, the page of Scheduler is open, no goals exist,

When: I click on “+”, specify title, description, *create a new goal by clicking “Create a new goal” button*, *days and times*, click “Save”,

Then: new schedule task should appear on days and times I specify.

As a user, I want to edit the schedule.

Given: I am logged in, the page of Scheduler is open, there is existing goal,

When: I navigate to a particular day, I click on edit button of the scheduler, edit title, description and/or goal, days and times, click on button “Save”,

Then: the window for editing should be closed. Updated tasks should appear in days and times that I specified.

As a user, I want to delete the schedule task.

Given: I am logged in, the page of Scheduler is open, there is existing schedule task,

When: I click on delete button of the task,

Then: the task should be deleted on the particular day and time, but other tasks of the same scheduler activity should persist.

As a user, I want to add a new to-do item.

Given: I am logged in, the page of To-do list is open, at least one goal exists, When: I click on "+", specify title, goal, click "Save",

Then: the new to-do item should be created.

Given: I am logged in, the page of To-do list is open, there are no goals,

When: I click on "+", specify title, *using spinner create a new goal by clicking on the button "Create a new goal",* click "Save",

Then: the new to-do item should be created.

As a user, I want to edit the to-do item.

Given: I am logged in, the page of To-do list is open, there is existing to-do item,

When: I click on edit button of the to-do item, edit title and/or goal, click on button "Save",

Then: the item should be updated.

As a user, I want to delete the to-do item.

Given: I am logged in, the page of To-do list is open, there is existing to-do item,

When: I click on delete button of the to-do item,

Then: item should be deleted.

As a user, I want to delete the to-do item.

Given: I am logged in, the profile page is open,

When: I edit name and family name, and restart the application,

Then: I should see the new user data in the menu.

3.6.2.2 Dependent testing

This test is performed in the form of one test suite with few test cases. All test cases depend on each other. If one test case fails, then the upcoming test cases are not expected to be done successfully.

Practically, the "Given" part for every test case is that the previous test case was completed.

All the test cases should be done in order. In the sense, we could merge all these test cases into one. But for the sake of logical separation, several test cases will be defined.

As a user, I want to register a new user.

Given: we are mentally prepared to start this big test suite,

When: I execute the following actions:

1. Open the application, click on the "Sign up";
2. Create a new user with the following credentials: email:dep_test@mail.com, password: 123456;
3. You should be logged in into the account automatically;
4. Log out using the appropriate button on the top right corner;
5. Enter the credentials and log in.

Then: Log in should be successful. Value Explorer should appear.

As a user, I want to create 2 values.

Given: we are logged in, and the Value Explorer page is open,

When: I execute the following actions:

1. Click on "+" button. Create one value with the title "Value one" and arbitrary description. Save the value.
2. Click on "+" button. Create the second value with the title "Value two" and arbitrary description. Save the value.

Then: values should be created.

As a user, I want to edit the values.

Given: we created 2 values in the previous test case,

When: I execute the following actions:

1. Edit the first value. Change the name to “Value 1”. Save it.
2. Edit the second value. Change the name to “Value 2”. Save it.

Then: values should be updated.

As a user, I want to create 2 goals.

Given: we have 2 values from previous test case,

When: I execute the following actions:

1. Navigate to the Goal Keeper page;
2. Create one goal with title “Goal 1” and assign it to the “Value 2”;
3. Create second goal with the title “Goal 2” and assign it to the “Value 1”.

Then: goals should be created.

As a user, I want to edit goals.

Given: we have 2 goals from previous test case,

When: I execute the following actions:

1. Edit the first goal, change the value of the first goal to “Value 1”. Save it.
2. Edit the second goal, change the value of the second goal to “Value 2”. Save it.

Then: goals should be updated.

As a user, I want to create scheduler tasks.

Given: we have 2 goals from the previous test case, When: I execute the following actions:

1. Navigate to the Scheduler page.
2. Create a new scheduler activity with the title “Schedule task 1”, assign it to the “Goal 1”, and for the days and times give Monday from 10:00 to 12:00, and Tuesday from 10:00 to 12:00. Save it.
3. Create a new scheduler activity with the title “Schedule task 2”, assign it to the “Goal 2”, and for the days and times give Monday from 13:00 to 15:00, and Tuesday from 13:00 to 15:00. Save it.

Then: the tasks are appearing on the proper days and times.

As a user, I want to edit the scheduler tasks.

Given: I have 2 scheduler tasks from the previous test case,

When: I navigate to Monday. Edit the “Schedule task 1”. Move days to Wednesday and Thursday instead of Monday and Tuesday. Save it.

Then: tasks should be updated.

As a user, I want to complete scheduler tasks.

Given: I have 2 scheduler tasks from the previous test case,

When: I execute the following actions:

1. Navigate to Monday. Make the “Schedule task 2” to be “Done”.
2. Navigate to Tuesday. Make the “Schedule task 2” to be “Missed”.
3. Navigate to Wednesday. Make the “Schedule task 1” to be “Done”.
4. Navigate to Thursday. Make the “Schedule task 1” to be “Missed”.

Then: tasks are marked as “Done” or “Missed” appropriately.

As a user, I want to check productivity rate of my goals.

Given: I completed 2 schedule tasks from the previous test case, When: I navigate to the Goal Keeper page and expand goals,

Then: both of them should have “Productivity rate is 50%. Done 1, missed 1”.

As a user, I want to check productivity rate in tracker.

Given: I completed 2 schedule tasks from the previous test case,

When: I navigate to the Tracker page,

Then: For each period the amount of done and missed tasks should be equal to 1.

4 Summary

As the application is finished, some quick summary can be written about it. It will describe how the expectations were met, and the general feeling about the work.

Speaking of the project description, which was written in May 2020, the application fully meets all the requirements. All the functionalities were developed.

Apart from the official description, there were some additional functionalities, and some details about the functionalities, that were on my mind from the beginning.

Considering the additional functionalities, the idea about to-do list was there from the beginning. It was implemented. Apart from that, there was an idea of making the goals of a particular person publicly available (and which can be hidden in settings). Unfortunately, there was not enough time to implement that.

Considering the details of implementation of particular pages, everything was mostly developed according to the expectations. There were some details that could not be implemented in time. One of them is the weekly schedule view in the Scheduler page, that I had in my mind. The other one is moving the to-do items, changing their orders, which was suggested by my supervisor.

Even though these details were not implemented, it does not interfere much with the work of the user. The functionality is full and complete.

The application itself is working quite fast, can be used on different android phones and with different screen sizes. The internet is required to use the app, and bad connection can hurt the experience a bit. Though, it is more of a Firestore problem.

Speaking of that, there were some choices that I made in the beginning phase, including usage of Firestore. Firestore has its cons and pros. It is still the easiest to use database for android applications, because everything was done by Android Studio developers to integrate smoothly the support for Firestore.

Unfortunately, Firestore is a "no sql" database. For some developers it is a good thing, but with my knowledge of SQL, it would be much easier to work with SQL. Particularly, Firestore does not support "merging" tables. Also, Firestore is asynchronous, and it does not help the problem.

Speaking more specifically, I needed to merge tables, and I could not do it, because the lack of this feature. Because of that I had to use workarounds, add fields to the tables that should not be there. In other cases, even workarounds were not able to solve the problem. Because of that, user has to wait for some operations to finish for one or two seconds.

Apart from that, the choices that I made were fine. And even the usage of the Firebase makes sense, considering the fact that I had to develop the application quickly, and Firebase support from Android Studio helped a lot.

Of course, there is a room for improvement. E.g., I could add new features. But in general, it gives all the needed functionality, and provides a good user experience.

5 Bibliography

[1] "Productivity - Android Apps on Google Play," [Online]. Available: https://play.google.com/store/apps/category/PRODUCTIVITY?hl=en&gl=US. [Accessed 15 December 2020].

[2] "The Toxic World of Self Help: Hustle Culture, Toxic Positivity, Addiction, and Fake Gurus.," [Online].

Available: https://www.youtube.com/watch?v=dmLTLkCBSN8. [Accessed 15 December 2020].

[3] "Android Studio on Wikipedia," [Online].

Available: https://en.wikipedia.org/wiki/Android_Studio. [Accessed 15 December 2020].

[4] "Firebase on Wikipedia," [Online].

Available: https://en.wikipedia.org/wiki/Firebase. [Accessed 15 December 2020].

[5] "Cloud Firestore," [Online].

Available: https://firebase.google.com/docs/firestore. [Accessed 15 December 2020].

[6] "MPAndroidChart on GitHub," [Online].

Available: https://github.com/PhilJay/MPAndroidChart. [Accessed 15 December 2020].

[7] "Cloud Firestore or Realtime Database," [Online].

Available: https://firebase.google.com/docs/database/rtdb-vs-firestore. [Accessed 15 December 2020].

[8] "Create dynamic lists with RecyclerView," [Online]. Available:

https://developer.android.com/guide/topics/ui/layout/recyclerview. [Accessed 15 December 2020].

[9] "Spinners," [Online].

Available: https://developer.android.com/guide/topics/ui/controls/spinner. [Accessed 15 December 2020].