Workout Planner Application Report

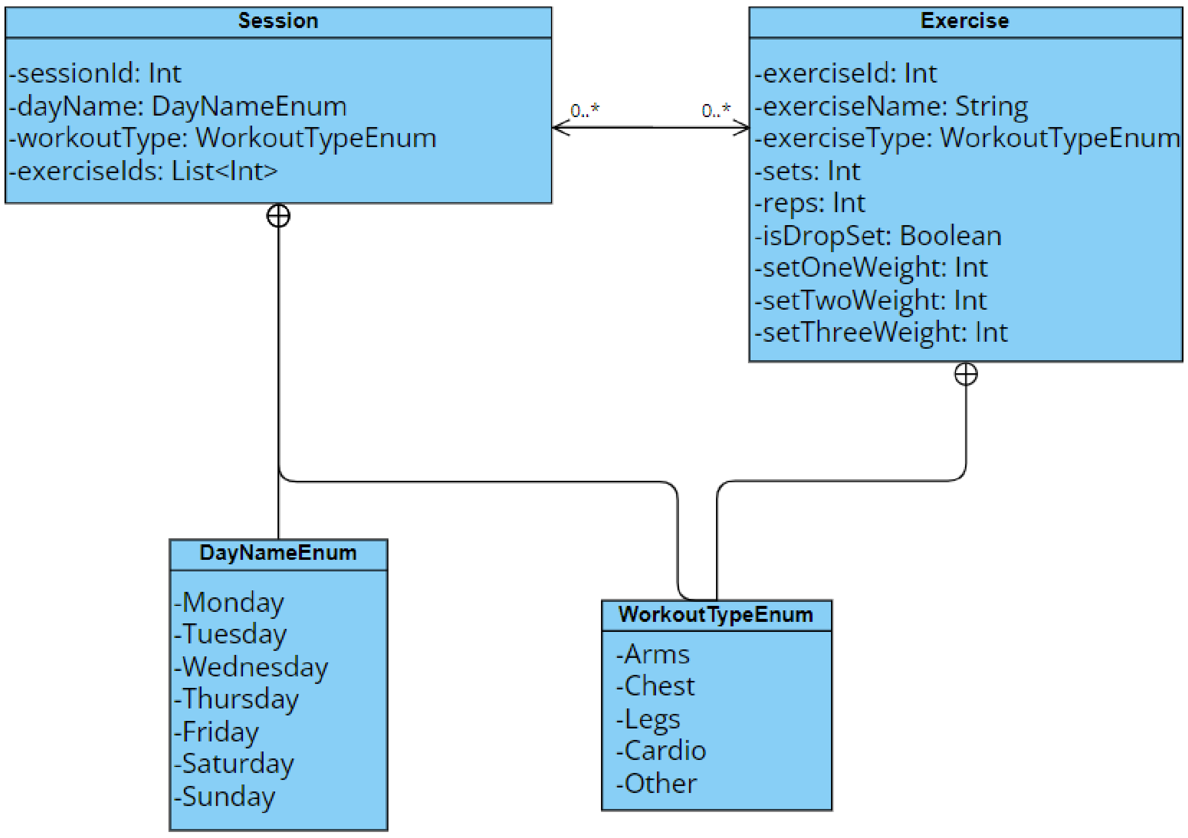
Kacper Prywata

[kap48]

Contents

Software Design

UML Diagram



Data classes

For this project I've decided to base the structure on the two main entities: Session and Exercise. A Session represents a workout session that happens on a given day of the week. The workout type variable takes one of the values from the Workout Type Enum. Each workout session holds zero or more exercises, whose ids are stored in the exerciseIDs variable

The Exercise class holds the name and type, which works analogously to the workoutType variable for a Session. For all exercises, it stores the number of sets and reps, and weight for the set, and the isDropSet variable which defines whether the exercise is treated as a regular exercise, or a drop set exercise. The drop set exercises make use of the setTwoWeight and setThreeWeight to store additional information.

I was debating the idea of implementing inheritance between the Exercise and Drop Set Exercise, with the latter being an extension adding the appropriate functionality, but I've decided that for the scope of this project this won't be necessary. The main drawback is the waste of the two additional weight fields for the regular exercises, but for the small scale of the project and the relatively small expected number of manually added exercises I kept it as just one class.

In the same file as the entities classes, I've defined the lists that are used to prepopulate the database.

Data relationship

I've identified the relationship between the Session and Exercise class as many-to-many relationship. Every Session can have zero or more Exercises in it, and each Exercise can be assigned to zero or more Sessions.

After researching the appropriate documentation, I concluded that implementing a many-to-many relationship with Room database would require a significant amount of work and would be out of scope for this project. Instead, I've decided to implement the relationship as a list of IDs of the Exercises for each Session.

That's not the ideal approach as it involves some consequences that will need to be worked around, but it's a solution that works and is relatively simple to implement.

Converting exercise ids into exercise objects

Because the Session objects store a list of exercise IDs, rather than Exercise objects and Room database doesn't support an equivalent data type I had to find a way to work around it. My solution was to implement a Type Converter, which would take the list of Exercise IDs, convert (serialize) them into its Json representation and store it as a string in the database. A reverse process (deserialization) happens when the list is retrieved from the database. For this operation I've used the Gson library.

When the Session object has retrieved and converted the list of Exercise IDs from the database, in order to get the list of Exercises matching the IDs in the list it has to query the *exercises* table. However, because Room uses the SQLite database under the hood, which has limited capabilities compared to other SQL databases, it's impossible to retrieve a list of rows based on a list of ids that contains duplicates (the row with the duplicated id is returned only once). To allow the user to assign the same exercise to a Session more than once I had to find a find a way to implement this.

What I've done is to implement additional logic in the Session View Model. First, I've created a Mutable State Flow object, (which is akin to Live Data, in the sense that it allows for asynchronous changes to its values, something that can be problematic with Mutable State). Then I've added a function called *loadSessionData* that makes the asynchronous queries to the database one by one for each exercise and assigns the result to the state flow object. The flow object is then *collected as state* in the Session screen composable

In order to avoid a loophole where the Exercise IDs are only added to the list held by the Session, but not removed once the Exercise is removed itself, I've added a cleaning phase to the function mentioned above. Each time the exercises are retrieved, a loop aggregates the list of indexes in the Exercise IDs list that haven't received an Exercise with the corresponding ID and removes them from the list.

The loadSessionData function has to be called before the invocation of the Top-Level Scaffold composable and whenever an Exercise is added or removed from a session to ensure the change is reflected in the UI. The *loadSessionData* function is also responsible for updating the estimated time of the session, which is also stored in a flow object, and works analogously to the one storing the Exercise list.

Navigation

The app uses a simple navigation graph which has reference to the two main screens. The user can switch between the screens by clicking the icons on the Navigation Bar.

User input validation

The place where the user can input text from the keyboard is the Exercise Details Dialog, which pops up whenever they attempt to create a new exercise or modify an existing one.

There are two stages to the input validation:

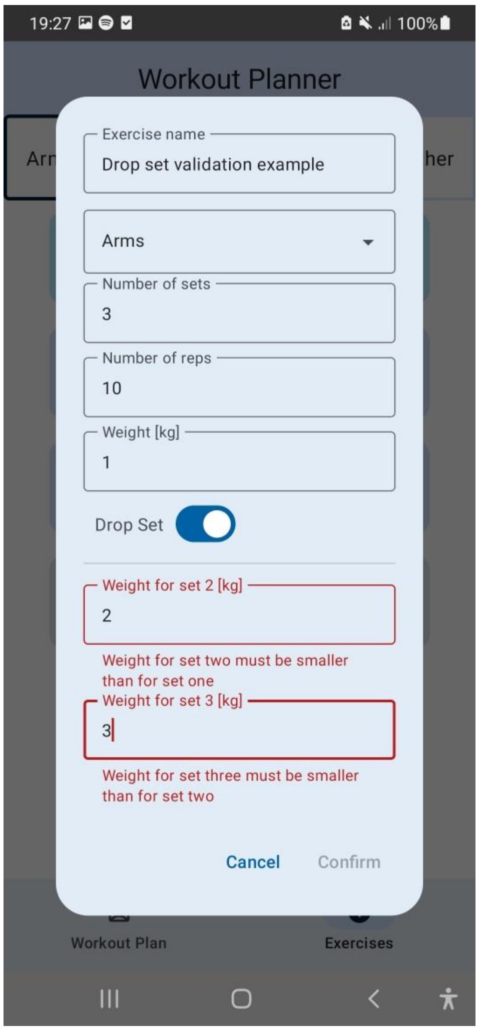
1. Disabling the *Confirm* button

The validation is performed by a simple logic that check that the input fields are not empty (and not zero for number fields). Additionally, when the drop set toggle is on, it checks that the fields for weight for the second and third set are not zero, and that the weight is decreased in each set.

When the input variables don't meet any of the criteria, the *Confirm* button is disabled and greyed out. The appropriate variables are passed directly to a logic statement whose result determines whether the Button composable is enabled. This ensures that recomposition happens each time any of the values changes, eliminating the need for an additional state variable to store the result of the validation.

2. Input field error message

If the drop set toggle is on, the fields for the weight for sets two and three perform and additional validation that checks that the weight for the first set is greater than the weight for the second set, and that the weight for the second set is greater than the weight for the third set. If the criteria aren't met, an error message is displayed under the text field:



Exercise name: Drop set validation example

Arms

Number of sets: 3

Number of reps: 10

Weight [kg]: 1

Drop Set: on

Weight for set 2 [kg]: 2

Weight for set two must be smaller than for set one

Weight for set 3 [kg]: 3

Weight for set three must be smaller than for set two

Cancel Confirm

User Interface

The UI makes heavy use of Jetpack Compose Cards, they are used as a container for the details of the workout sessions and the exercises.

For more elaborate user interactions, I've used Dialog boxes which allow the user to make choices such as selecting an Exercise to add to a Session and add or modify an existing Exercise.

Colour Scheme

The Colour Scheme for the app was generated following the Material Compose guidelines using the online tool provided by Google, and it was then imported to the project in the Color.kt file.

It is worth noting at this point that the Drop Set exercise cards are using the colour for tertiary container, rather than secondary. This decision was taken based purely on my preference for the appearance of the app – it was easier to distinguish from the primary container colour, which regular exercise cards are using.

Screens

The structure of both main screens in the app is similar: both extend the TopLevelScaffold composable, which contains the top app bar and bottom navigation bar.

As I've explained in the UI Prototype report, I didn't identify the need for a Navigation Drawer as according to Material Design 3, it's recommended for "Apps with 5 or more top-level destinations".

Exercise Screen

The Exercises screen then adds the Top Row Component, which can be used to filter the exercises by their types. The main content of the page then presents a list of existing exercises from the selected category. The user can:

- add new exercise – by clicking the button with the plus symbol at the bottom of the column, which will result in an Alert Dialog component opening, presenting to the user a form with exercise details to fill in.
- edit existing exercises – by clicking one of the existing exercises the user can open the Alert Dialog and edit the details of the exercise
- remove the exercise – by clicking the bin icon on the exercise card

Session Screen

The Session screen adds the Top Row Component, with the content of the cells being short forms of the names of days of the week (Mon, Tue, ...). This screen by default opens on the current day of the week on every app recreation event.

Day Card

The main content of the Session page contains the Card component stating the full name of the day, the type of the workout session, the image. On the right side of the card there's an icon, that when clicked opens a dropdown menu that allows the user to change the type of the session.

The day Card also contains the approximate time of the session based on the number of exercises, number of sets and reps in each exercise and whether it's a drop set. The total is calculated basing on an assumption that each in repetition of an exercise takes 3 seconds. According to the assignment brief, for drop sets, there's a 3-minute break between each cycle of the exercise and a 10 second break between repetitions of the drop sets. There's also a break of 2 minutes assumed between each exercise. All these values are defined in code as constants to allow easy change in case these assumptions are changed.

Exercise List

Below the Day Card, there's a list of exercises with regular and drop set exercises which are distinguished by the colour of the card as well as a label below the exercise icon. The drop set exercises cards display 3 values for the decreasing weight for each set whereas the regular exercises display just one value.

The user can remove the exercises from the Session by clicking the bin icon on the card and add exercises by clicking the card with the plus symbol at the bottom of the column. As a result, an Alert Dialog is opened, containing a list of all existing exercises. The user can add them to the session by simply clicking on one of the small exercise cards from the list.

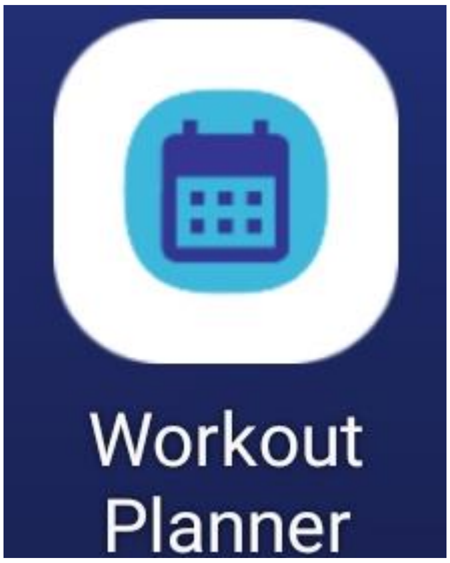
Use of Icons

The Day Card and Exercise Card components display an icon, which is selected based on the exercise or session type, which can take one of the values of the ExerciseTypesEnum.

The icons used were taken from the website *flaticon.com*, which allows unlimited use for non-commercial purposes. I added them using the Asset Studio and they're stored in the *drawable* folder of the app.

App Icon

I've added an icon for the app using the Asset Studio tool in Android Studio:



Testing

For testing, I've selected the testing table approach, which allows me to assess logic of the program and the User Interface at the same time. The table is based on the functional requirements I've identified in the problem definition in the assignment brief. The appropriate screenshots can be found in the Appendix.

Test Number	Test Description	Test Input	Expected output	Actual output	Pass/ Fail
FR 1	The user can see a typical workout week	The user opens the app	The user can see workout session	The app opens on the Workout Plan screen with the current day of the week selected	Pass
FR 2	The user can add workout sessions	The user adds a initializes the session by selecting its type and exercises	The change is represented in the UI	The user can modify the workout session – change the type and add/remove exercises	Pass
FR 3	The user can remove the workout session	The user tries to remove all exercises from a workout session.	The change is represented in the UI	The user can remove all the exercises from the workout session	Pass
FR 4	There is a separate way to view the exercises	Go to the Exercises screen.	A list of exercises is shown	A list of exercises is displayed on the Exercises screen	Pass
FR 5	The user can add exercises	Click the add exercise button, type in the details of the exercise.	A new exercise is added to the list	The exercise is added to the list	Pass
FR 6	The user can edit exercises	Click an exercise from the list and change its details.	The change is represented in the UI.	The user can change the details of the exercise.	Pass
FR 7	The user can remove exercises	The user clicks the bin icon.	The exercise is removed from the list	The exercise is removed from the list.	Pass

FR 8	The exercises can be shared across workout sessions	The user assigns an exercise to multiple workout sessions	The user can add an exercise to multiple sessions	The exercise is added to multiple sessions	Pass
FR 9	Each exercise has a title, image, number of sets, number of repetitions, weight in kilograms	N/A	The exercise details are represented in the UI.	The exercises have all the required details	Pass
FR 10	There is a drop set feature, which when activated, will allow the user to specify different weight for three consecutive sets of the exercise.	The user switches on the Drop Set toggle in the Exercise Details Dialog	The user can type in the additional details	The user can input the additional detail for drop set exercises	Pass

Discussion of a REST API

Assuming further development of the Workout Planner app, we might want to for example develop a website that would allow similar interactions as the native app. One of the ways to achieve this is a RESTful API running on a remote server that would allow the users on the website to access and modify the data.

For that, we would need to develop a set of API calls, using HTTP methods and clearly defined resource URIs that would allow us to make be analogous interactions to the CRUD operations that we perform on the local database when using the app.

One thing worth a brief discussion is how we want to represent the relations between the Session and Exercise entities we could do one of the two:

1. implement it in an analogous way to how it's done internally in the app and return the list of Exercise IDs for the given session, and rely on the client to make further requests for Exercises based on the list
2. return a Json representation of the list of Exercise objects with all their details

The final decision would largely depend on the implementation details, but in my opinion the first approach is more appropriate. It allows us to reduce the size of the payload that we send as a response therefore decreasing the processing time on the server side. It would force the client to make further request if the full Exercise information is required, but the overhead of that is negligeable.

Following the best practices, I would store the list of all the Exercises in the following Collection Resources: *https://workout-planner/sessions* and *https://workout-planner/exercises*.

The client could perform the following request:

```
GET /workout-planner.com/sessions/1
```
With the sending headers: Accept: application/json; q=1.0

And receive a response in the Json format:

```
{
  "sessionId": "1",
  "dayName": "Monday",
  "workoutType": "Arms",
  "exerciseIds": [1, 2, 3]
}
```

Along with the received headers: Content-Type: application/json
And the status code: 200 OK

In case where the client requests a session with an ID that isn't present on the server, it would return the 404 Not Found error code along with a short message in the response body.

After a successful request the client can select one of the Exercise Ids and perform another request to get the details of the exercise:

GET /workout-planner.com/exercises/3

With the sending headers: *Accept: application/json; q=1.0*

And receive the response in Json format:

```
{
  "exerciseID": 3,
  "exerciseName": "Lat raises drop set",
  "exerciseType": "Arms",
  "sets": 3,
  "reps": 10,
  "isDropSet": true,
  "setOneWeight": 30,
  "setTwoWeight": 20,
  "setThreeWeight": 10
}
```

With the received headers: Content-Type: application/json
And the response code: 200 OK

Reflection

Challenges and accomplishments

As I mentioned earlier, one of the main challenges for me was implementing the many to many relationship between the Session and Exercise entities. Finding a working solution required a large amount of research, by which I've expanded my understanding of how the datatypes and concurrency work in Kotlin.

Apart from that I didn't have any major challenges. I found implementing the user interface very satisfying, especially the aspect of developing the subsequent custom composables so that they match the prototype.

However, there still remains one problem that I wasn't able to solve – on the first attempt to run the application, it crashes because of a Null Pointer Exception on a Session object. My initial guess was that because the coroutines are running concurrently, the database is not yet populated and thus not ready to be used when the UI is generated. I tried to work around that by giving the state variable an initial value of a Session object with the default values, but that doesn't seem to work. The working solution is to simply try opening the app again.

Future improvements

I think that even though the app is simple, and all the possible user interactions are intuitive, the app could benefit from some kind of onboarding process that would familiarize the user with all the available actions. I've investigated the TapTargetView and AppIntro libraries and I'm planning to use one of them app to add a tutorial to the app.

In the future I will also try to add sorting the exercises not only on the Exercises screen, but also on the Pick Exercises Dialog on the Session Screen.

Takeaways for future projects

Using Jetpack Compose was my first time using a declarative programming paradigm for user interface. I've done some research, and it seems to be similar to React.js on the conceptual level so this project is definitely a good first step towards that direction.

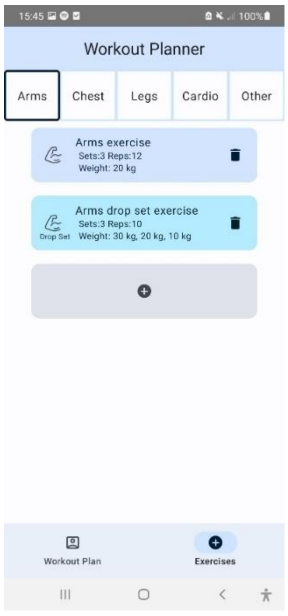
It was also my first practical experience using concurrency which I'm sure will be useful for my future career.

Mark estimation

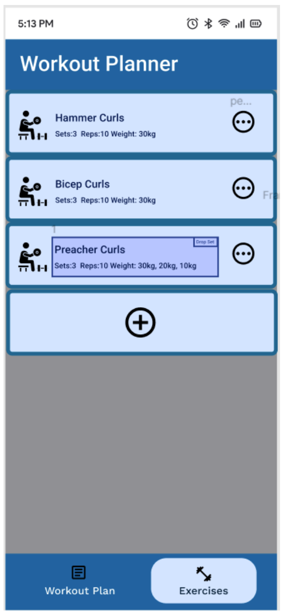
The application meets all of the functional requirements, in my opinion is visually accessible and responsive. The implementation makes use of the technologies presented in the lectures, both in terms of the visual design - adherence to Material Compose, and the backend – use of the Room Database, Kotlin coroutines, the Model–View–View Model etc. Considering this, the mark breakdown from the assignment brief and the mark I've received for part A of the assignment, I would predict the final mark to be somewhere around 75%.

Changes to the prototype

I've managed to stay relatively close to the initial UI prototype and applied most of the changes mentioned in the feedback. As an example, see the comparison of the Exercises screen attached below – screenshot of the app on the left and the UI prototype on the right:



Screenshot of the app


5:13 PM
Workout Planner
Hammer Curls
Sets:3 Reps:10 Weight: 30kg
Bicep Curls
Sets:3 Reps:10 Weight: 30kg
Preacher Curls
Drop Set
Sets:3 Reps:10 Weight: 30kg, 20kg, 10kg
Workout Plan
Exercises

App prototype

Addressing the feedback

Following the feedback from the UI prototype, I made sure that the app looks good in both light and dark mode. The screenshots can be seen throughout this report.

The exercises are displayed in the same order they were added to the Exercise list on the Exercises Screen, and in the same order they were assigned to the Workout Session in the Session Screen.

I decided to keep the card with the plus symbol at the bottom of the exercise list (instead of implementing a Floating Action Button) for adding an exercise both to the Exercise list and to the Session. This is only a personal preference, this approach seemed more intuitive to me, and I preferred how it looked over the FAB.

Editing and removing exercises

One of the things I've changed from the UI prototype is how the Exercises can be removed and edited in the Exercises screen. In the prototype, there was a clickable icon on the Exercise card, which would open an Alert Dialog, allowing the user to update or remove the Exercise.

What I've done instead was to enable opening the Alert Dialog window by clicking on the Exercise Card and only allow for editing the existing Exercise. Removing the Exercise could then be performed by clicking the Bin icon on the right.

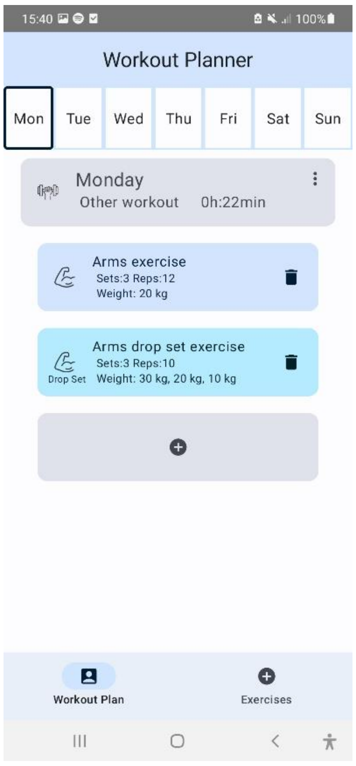
Image Selection

Another thing I've changed is how the images for exercises are chosen – instead of giving the user the option to select an image for a Session or an Exercise, it is now assigned automatically based on the workout type selected by the user. The relationship is hard-coded, and the images are stored in the *drawable* folder of the application.

Appendix

FR 1

The user can see a typical workout week


Workout Planner

Mon Tue Wed Thu Fri Sat Sun

Monday
Other workout 0h:22min

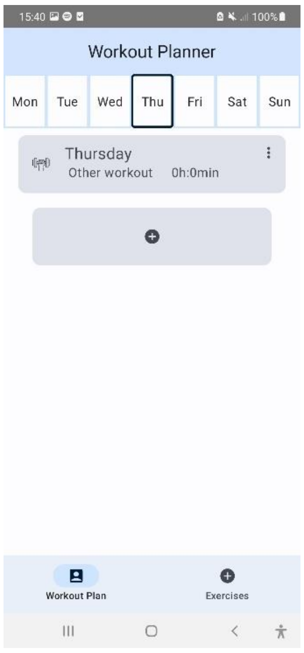
Arms exercise
Sets:3 Reps:12
Weight: 20 kg

Drop Set
Arms drop set exercise
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

The user can see the workout session

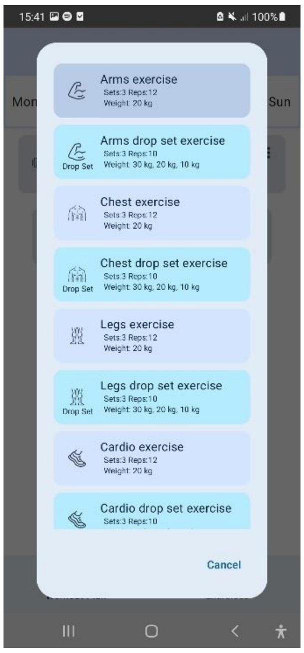
FR 2

The user can add workout sessions


Workout Planner

Mon Tue Wed Thu Fri Sat Sun

Thursday
Other workout 0h:0min


Arms exercise
Sets:3 Reps:12
Weight: 20 kg

Drop Set
Arms drop set exercise
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

Chest exercise
Sets:3 Reps:12
Weight: 20 kg

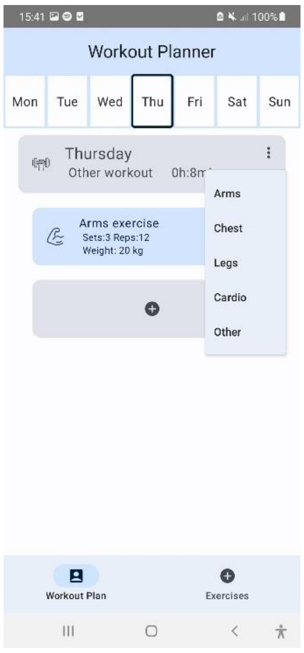
Drop Set
Chest drop set exercise
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

Legs exercise
Sets:3 Reps:12
Weight: 20 kg

Drop Set
Legs drop set exercise
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

Cardio exercise
Sets:3 Reps:12
Weight: 20 kg

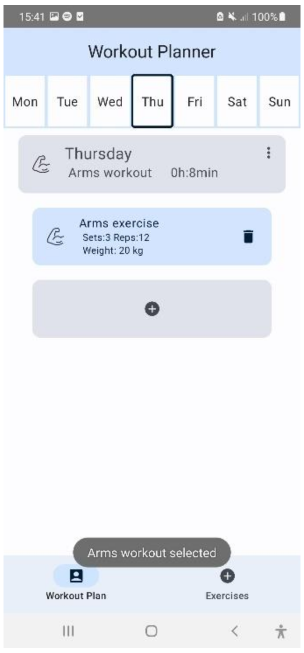
Cardio drop set exercise
Sets:3 Reps:10


Workout Planner

Mon Tue Wed Thu Fri Sat Sun

Thursday
Other workout 0h:8m

Arms exercise
Sets:3 Reps:12
Weight: 20 kg


Workout Planner

Mon Tue Wed Thu Fri Sat Sun

Thursday
Arms workout 0h:8min

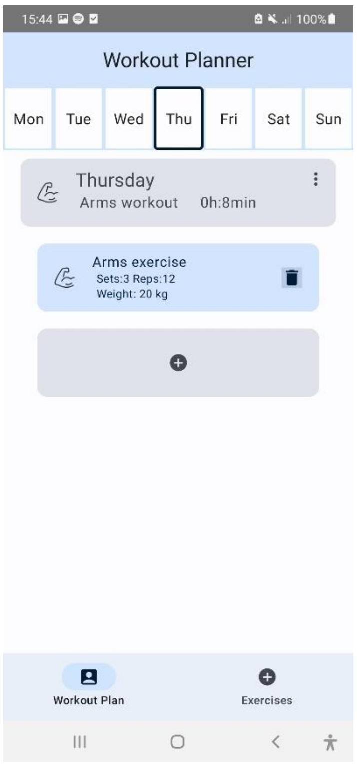
Arms exercise
Sets:3 Reps:12
Weight: 20 kg

Arms workout selected

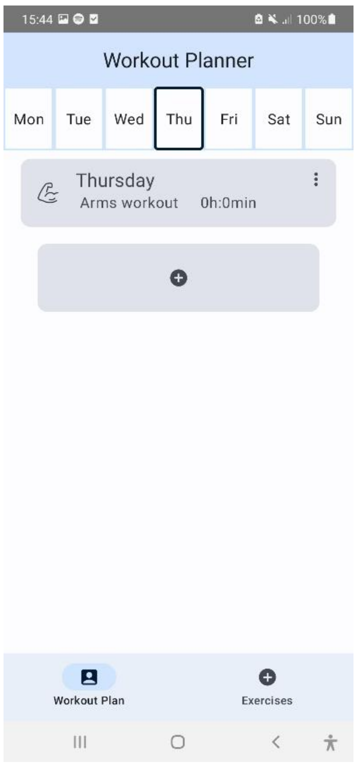
1. An empty workout session *2. The user adds an exercise* *3. The user selects session type* *4. The change is represented*

FR 3

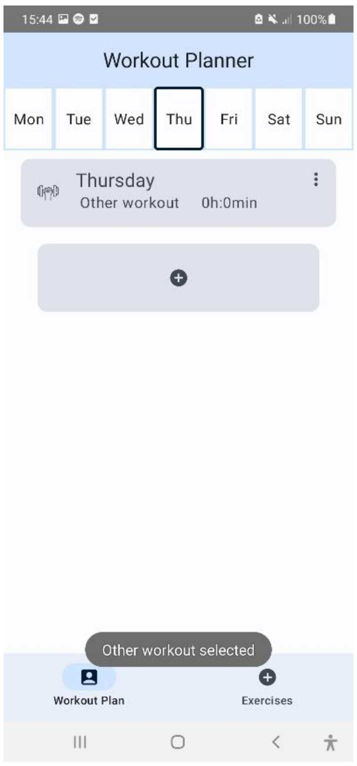
The user can remove the workout session



1. The user removes all exercises



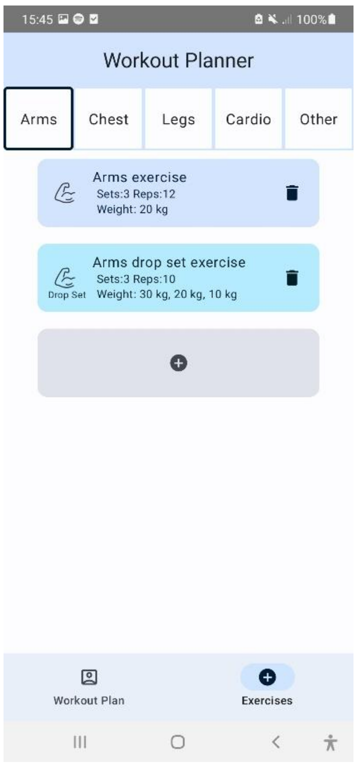
2. The user selects the default session type



3. The change is represented

FR 4

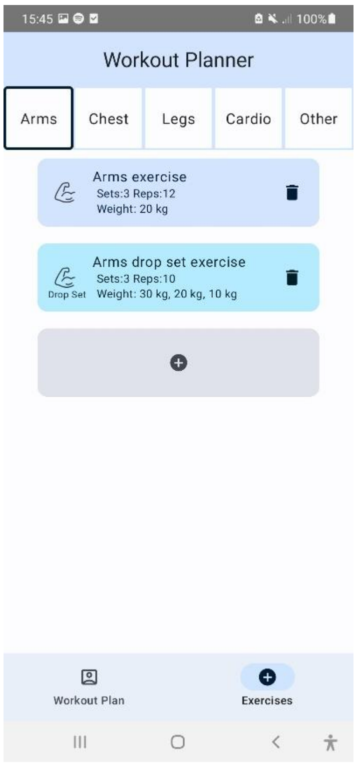
There is a separate way to view the exercises



The user can open the Exercises screen

FR 5

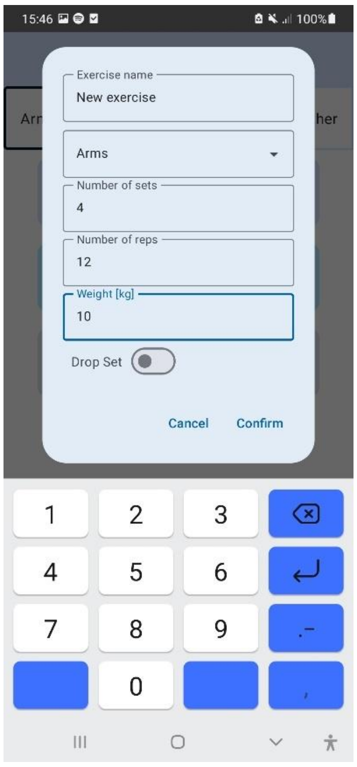
The user can add exercises



Arms exercise
Sets:3 Reps:12
Weight: 20 kg

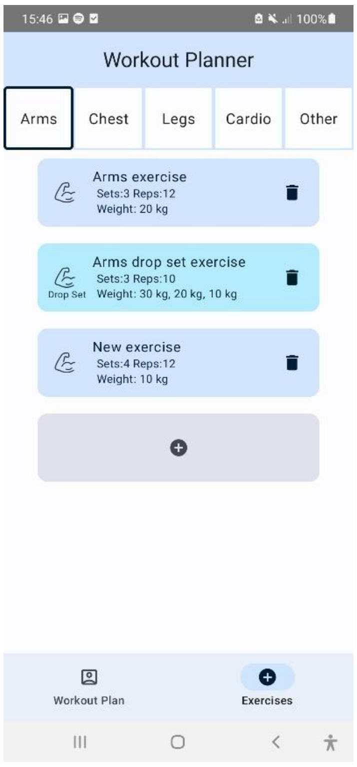
Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

1. The user clicks the add button



Exercise name: New exercise
Arms
Number of sets: 4
Number of reps: 12
Weight [kg]: 10
Drop Set

2. The user fills the exercise details



Arms exercise
Sets:3 Reps:12
Weight: 20 kg

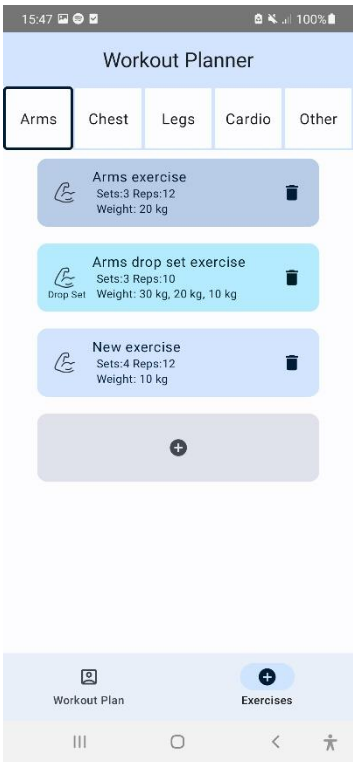
Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

New exercise
Sets:4 Reps:12
Weight: 10 kg

3. The change is represented

FR 6

The user can edit exercises

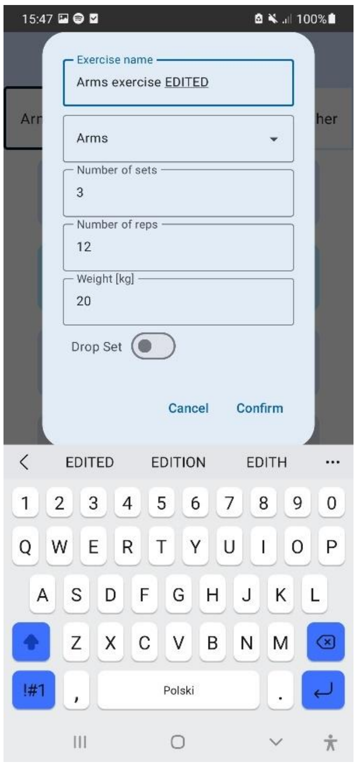


Arms exercise
Sets:3 Reps:12
Weight: 20 kg

Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

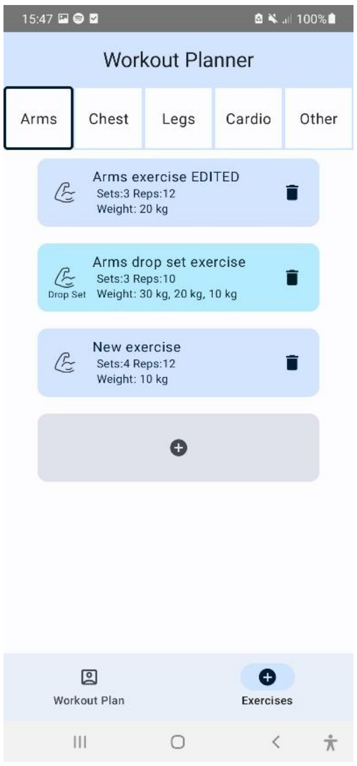
New exercise
Sets:4 Reps:12
Weight: 10 kg

1. The user clicks the exercise card



Exercise name: Arms exercise EDITED
Arms
Number of sets: 3
Number of reps: 12
Weight [kg]: 20
Drop Set

2. The user changes the exercise details



Arms exercise EDITED
Sets:3 Reps:12
Weight: 20 kg

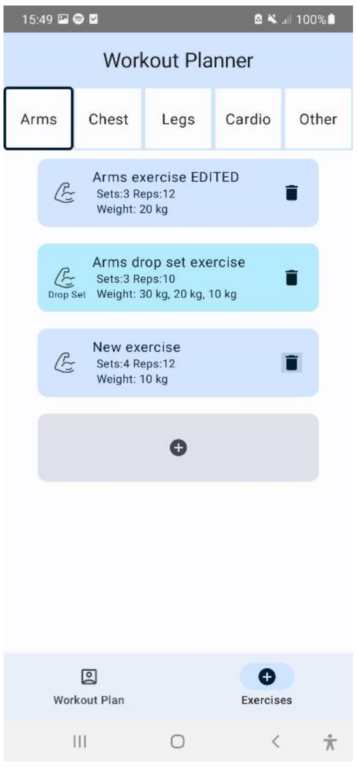
Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

New exercise
Sets:4 Reps:12
Weight: 10 kg

3. The change is represented

FR 7

The user can remove exercises


15:49 100%

Workout Planner

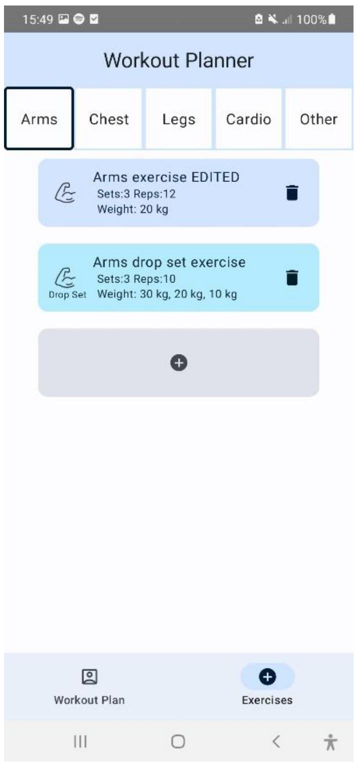
Arms | Chest | Legs | Cardio | Other

Arms exercise EDITED
Sets:3 Reps:12
Weight: 20 kg

Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

New exercise
Sets:4 Reps:12
Weight: 10 kg

1. The user clicks the bin icon


15:49 100%

Workout Planner

Arms | Chest | Legs | Cardio | Other

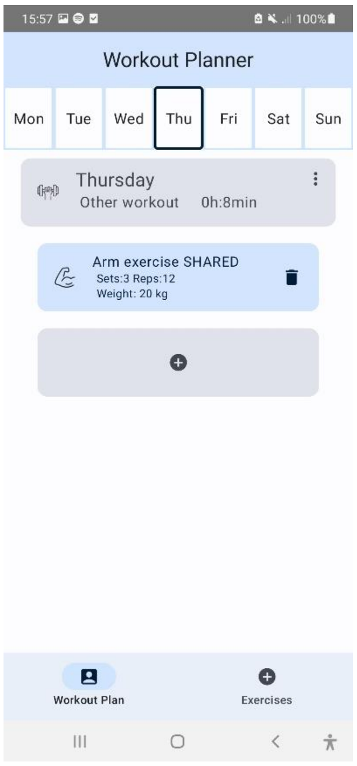
Arms exercise EDITED
Sets:3 Reps:12
Weight: 20 kg

Arms drop set exercise
Drop Set
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

2. The exercise is removed

FR 8

The exercises can be shared across workout sessions


15:57 100%

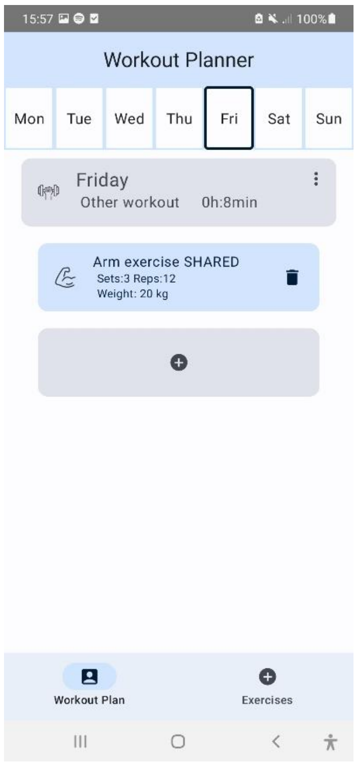
Workout Planner

Mon | Tue | Wed | Thu | Fri | Sat | Sun

Thursday
Other workout 0h:8min

Arm exercise SHARED
Sets:3 Reps:12
Weight: 20 kg

1. The exercise is added to a session


15:57 100%

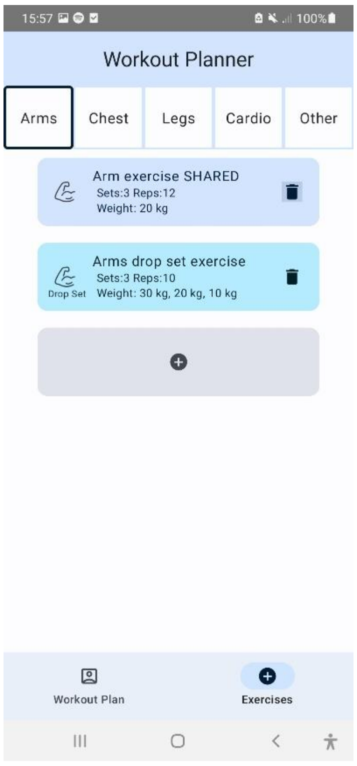
Workout Planner

Mon | Tue | Wed | Thu | Fri | Sat | Sun

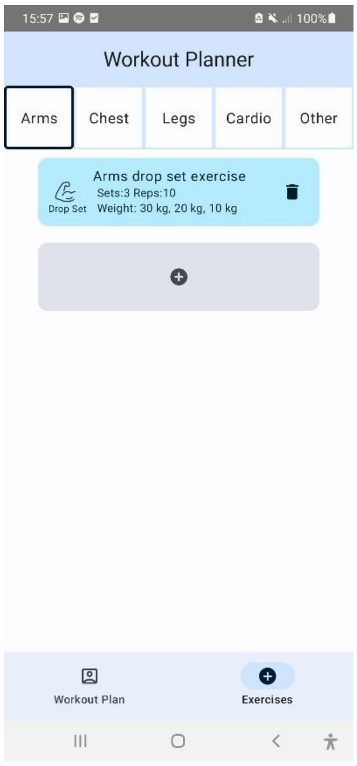
Friday
Other workout 0h:8min

Arm exercise SHARED
Sets:3 Reps:12
Weight: 20 kg

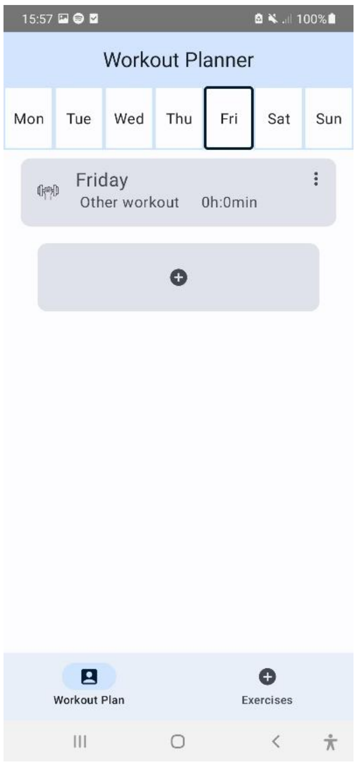
2. The exercise is added to another session



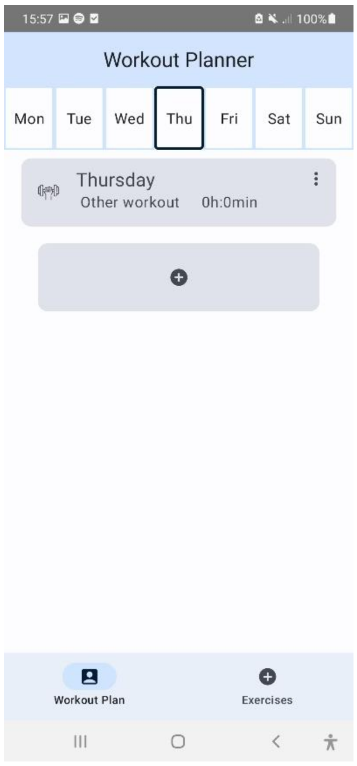
3. The exercise is removed from the exercise list



4. The change is represented



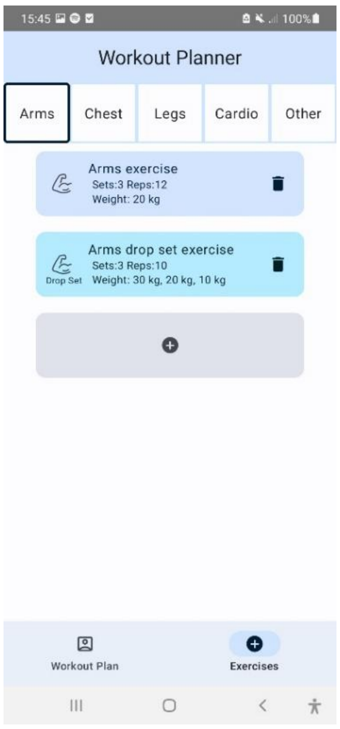
5. The change is automatically removed from the first session



6. The change is automatically removed from the first second

FR 9

Each exercise has a title, image, number of sets, number of repetitions, weight in kilograms



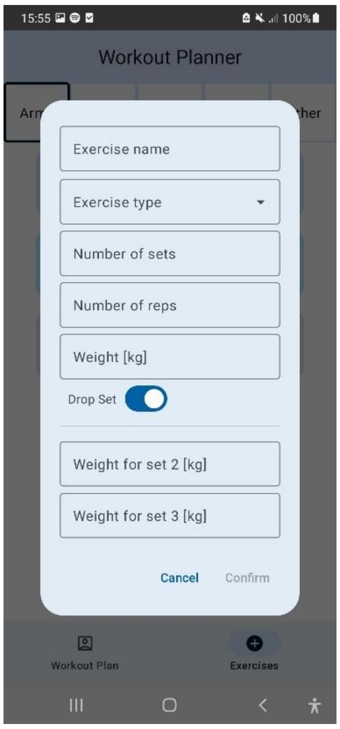
Arms exercise
Sets:3 Reps:12
Weight: 20 kg

Drop Set
Arms drop set exercise
Sets:3 Reps:10
Weight: 30 kg, 20 kg, 10 kg

The Exercise Card has the required details

FR 10

There is a drop set feature, which when activated, will allow the user to specify different weight for three consecutive sets of the exercise.



Exercise name
Exercise type
Number of sets
Number of reps
Weight [kg]
Drop Set
Weight for set 2 [kg]
Weight for set 3 [kg]

There is a Drop Set feature that allows the user to input the additional details.