**Systems structure – Server-side**

The server-side of the system manages the communication with the database and allows the client side to receive and save user’s information.

The system is implemented as a Web Service implementing the RESTful Web Services architecture.

For the implementation of the system, we chose the spring framework, utilizing Spring Web for the RESTful web server and Spring Data JPA alongside Hibernate to persist data in the database.

The server exposes multiple endpoints for incoming communication from the client.

The client communicates with the server over HTTP to execute the CRUD operations needed for the application to run.

**Server-side classes and relations**

Structure:

The server-side classes are split into 3 folders:

1. Entities
2. Repositories
3. Controllers

**Entities**

All the entities are in the entities folder.

These are mapped by the Hibernate JPA implementation to Tables in the database.

* Category

A workout category. Some example – “Legs”, “Chest”, “Biceps”.

* Exercise

A type of exercise. For example – “Bicep curls”, “Leg press”.

The application comes pre-loaded with a wide variety of exercises, and the users has an option of adding their own custom exercises as well.

* Set

The unit of work in a workout. A set contains information about the weight lifted, type of exercise, number of reps done, etc.

* User

A user in the system. Each user is only exposed to their own data.

* Workout

A saved workout template. The application comes with 3 basic workout templates to start with, and the user can add their own custom workout templets which can be executed.

* WorkoutLog – inherits from the Workout class.

A workout executed by the user. Contains information about the exercises preformed, starting/ending/updated time, and notes for the workout.

* Special entity – WGER

Used for getting the pre-loaded exercises. The data comes from a workout database publicly available, consumed by an API at https://wger.de/api/v2.

**Controllers**

Each entity is controlled by a matching controller for controlling the information in the database for the relevant entity.

The controllers expose API endpoints giving the client-side access to the data.

These APIs except one or more of the GET, POST, PUT and DELETE requests from the client-side and in return execute the necessary actions. When needed, the APIs return a response in the form of a JSON object for the client-side to process accordingly and update the applications frontend.

Some key examples:

* WorkoutLogController

[GET] “/logs” – returns a list of WorkoutLogs preformed by the user.

[POST] “/logs” – Persists a WorkoutLog in the database.

[DELETE] “/logs/{id}” – Deletes the WorkoutLog with the given id from the database.

* UserController

[GET] “/users” – returns the user according to the given user id parameter.

* WgerController

Special controller which gets the exercise data from the WGER public exercise database upon initiation of the application.

After initial import, the data is fetched on a weekly basis to keep up with any new exercises that might be available.

* SetController

[GET] “/sets” – Return a list of all the users sets.

**Repositories**

Interface classes used as repositories for interaction with the database.

Each entity has a repository for controlling its equivalent table in the database.

Spring DATA JPA comes with implementation for the basic actions a repository needs in the form of the JpaRepository interface.

We extend and add our own functions to meet our specific needs.

Some examples of our own specific functions:

* SetRepository

findByUser\_Id

findByUser\_IdAndNameIsOrderByWeightDesc

findFirstByUser\_idAndPrIsTrueAndNameIsOrderByWeightDesc

* WorkoutLogRepository

findByUser\_IdOrderByStartedAtDesc

* ExerciseRepository

findByCategory

findByUser\_Id

**Systems structure – Client-Side**

The client-side of the system shows the user the data that was retrieved by the server-side, and to trigger through the use of UI elements such as buttons in order to navigate between different screens and to call the server-side functions.

We built the client side using Android Studio.

The project is split up into 3 main packages: UI, Services, and Data.

**UI package:**

Consists of the classes that affect the UI of the app, such as The different Activities and fragments. Popup windows, and Recycler view adapters.

The App has 2 activities:

**SignInActivity**: This is the starting point of the app. The user signs in using Google.

After signing in, the system will use firebase authentication to create a UID for the user based on his google account, which is used to link the user’s data in the DB to the user.

**MainActivity**: After signing in, all of the user’s interactions with the app will be in this activity.

The Activity is split up into multiple fragments. Each serving as a screen for the user to interact with. All the fragments in the main activity are linked via a navigation graph and a bottom navigation bar, allowing the user to travel between the different fragments.

I will now go into details about the different fragments.

**WorkoutsFragment:**

A fragment showing the user a list of their workout routines. There are 3 built-in routines that the user can view and perform to show an example of what a workout routine could look like.

The user can create a new routine, start a new empty workout, edit an existing routine or start a workout from a new routine.

The list is utilized using a recycler view which uses the WorkoutsAdapter class as its recycler view adapter.

**ActiveWorkoutFragment:**

This fragment represents an active workout. It too uses a recycler view to display its elements. It uses the ActiveWorkoutAdapter class as its recycler view adapter.

The different elements the recycler view show are the following:

1. Timer and Stopwatch.
2. Header with text field for the workout’s name.
3. A list of exercises, implemented with a recycler view and the ActiveWorkoutExerciseAdapter.
4. Buttons for saving or canceling the Workout.

**WorkoutTemplateFragment:**

This fragment represents a workout routine. It is very similar to the ActiveWorkoutFragment, and uses the WorkoutTemplateAdapter as its recycler view adapter.

The different elements the recycler view show are the following:

1. Header with text field for the workout’s name.
2. A list of exercises, implemented with a recycler view and the WorkoutTemplateExercisesFragment.
3. Buttons for saving or canceling the Workout.

**ExercisePickerFirstFragment:**

When the user adds a new exercise, they are taken to this fragment. At the top of the screen is a RadioGroup for selecting between the built-in Exercises and the user’s custom exercises.

The list is utilized using a Recycler view, which uses both ExericsePickerCategoriesAdapter and ExercisePickerExerciseAdapter as its adapter, based on which Radio Button is selected.

The built-in exercises list is comprised of different categories. When the user presses a category, they are taken to the ExercisePickerSecondFragment.

**ExercisePickerSecondFragment:**

Shows the user a list of the different exercises that are under the same category. It too uses a recycler view with the ExercisePickerExerciseAdapter as its adapter.

**LogsFragment:**

Show the user a history of their performed workouts. It too uses a recycler view with the WorkoutLogsAdapter as its adapter.

**LogFragment:**

Show the user detailed information about a specific performed workout, such as the weights and number of reps performed in each set, and whether a personal record was broken at that workout.

It too uses a recycler view with the LogExercisesAdapter as its adapter.

**ExercisesFragment**:

Shows the user a list of their custom exercises. And gives them the option to add or remove exercises. Implemented with a recycler view and the CustomExercisesAdapter.

**ProfileFragment:**

Shows the user their registered email address and a button for logging out.

**Data package:**

This package holds classes that represent the data in the database. It has classes such as SetData, UserData and WorkoutData which have the same fields as server-side counterparts.

The different UI classes use the data classes and show their data to the user in a compact way.

**Services package:**

This package consists of Service classes and their listeners. The services are used to send HTTP requests to the server-side.

Different fragments use different services and pass themselves as a listener.

For example:

The LogsFragment implements the LogServiceListener interface and has a private LogService.

When the user opens the LogsFragment screen, It calls the LogService’s getAllLogs(uid) function.

The service performs an HTTP request, And when a response is received, The service calls its listener’s onGetAllLogsCompleted() function, which passes the response to the fragment, which then shows the data to the user.

All The services classes perform HTTP requests using Google’s Volley.