**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