DailyFit

Analysis and Design Document

# Elaboration – Iteration 1.1

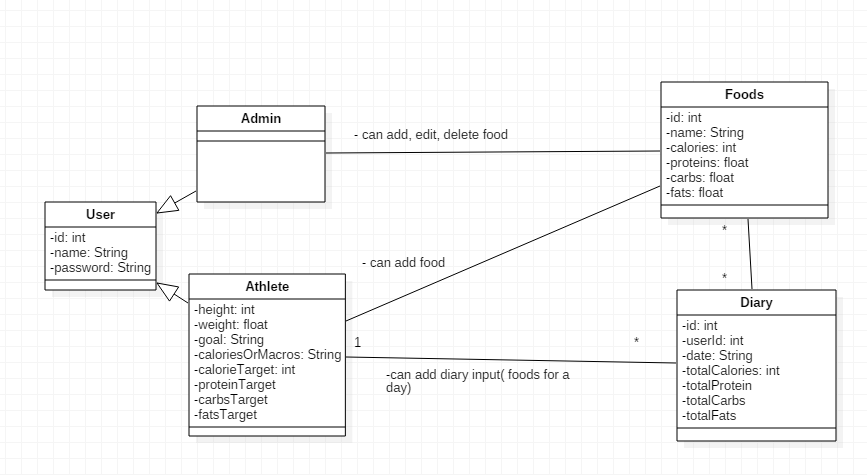
# Domain Model

My application has two types of users: admins and normal users, defined as athletes for simplicity. An athlete has attributes such as personal data (height, weight, goal, variables to track - caloriesOrMacros) and data that will be calculated according to the previous mentioned attributes (calorie target, macronutrients target – proteins, carbohydrates, fats). Admins have only general data that each user has: id, name and password.

Admins can add foods in the database, edit or delete them in order to maintain a clean database. Athletes can only add foods in the database.

Foods have an id, a name that will help the user to identify them and nutritional data for calculations (calories, proteins, carbs, fats). The data corresponds to a quantity of 5g.

A diary entry is basically a day. It contains an id, the id of the corresponding user, a data and the total number of calories and macros consumed in that day. There is a relationship of many to many between foods and diary that will require an additional table in the database to make the connections. A user eats more foods in a day and a food can be eaten in more that one specific day. Between athlete and diary exists a one to many relationship because an athlete can have many diary inputs (days) but a diary input corresponds to a single athlete.

**

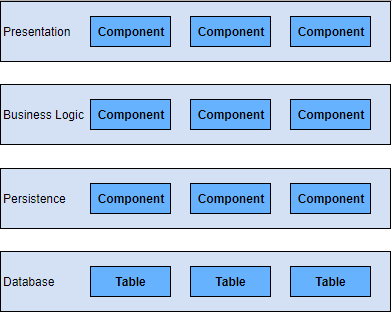
# Architectural Design

## Conceptual Architecture

In my application I will use a layered architecture. It will offer a more organized look and a clear definition of what each class is about because each layer has a specific role. It also makes it easier to replace certain parts of the application without affection the whole system. The architecture will contain the following layers in the given order: a presentation layer, a business layer, a persistence layer and the database layer.

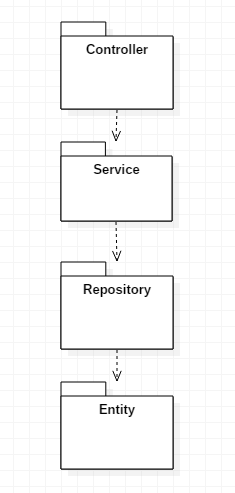
Dependencies should go from the top (presentation) to the bottom (database). Each layer can access only the layer below it, without jumping over intermediate layers. For example, the presentation layer should not depend directly to the database or the business layer to the presentation.

The database is not exactly a layer in the application, since it refers to the tables in the schema stored in MySQL. The persistence layer will contain a set of repositories for manipulation entities. The business logic layer will be represented by a set of service classes responsible for giving a set of available operations, actions.



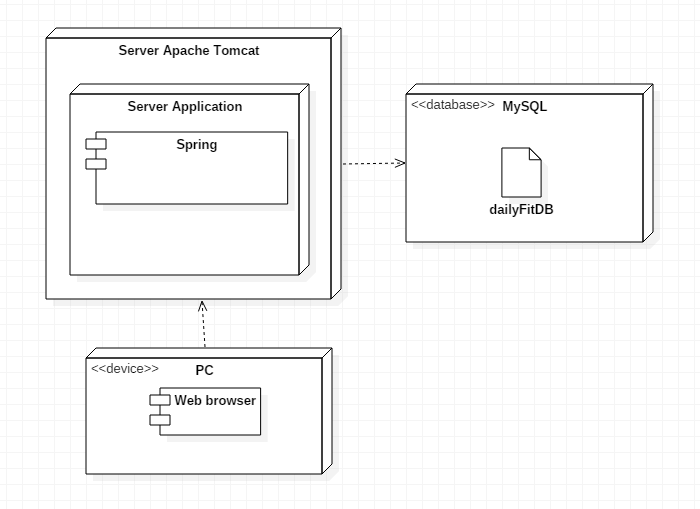
## Package Design

The application will contain the following packages: entity, repository, service and controller.

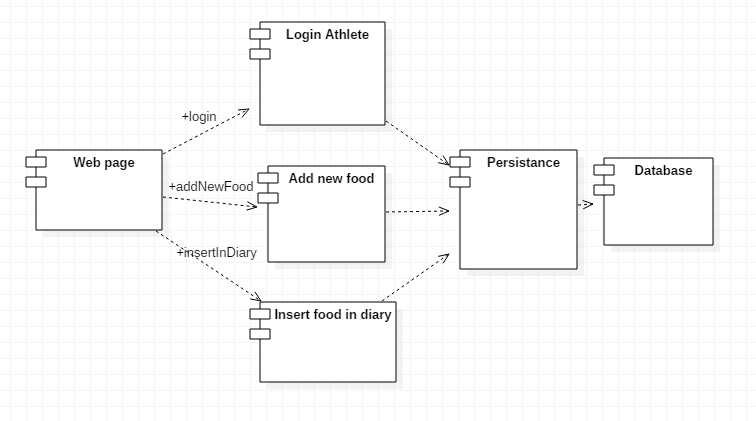
**

## Component and Deployment Diagrams

Deploymentdiagram



Component diagram



# Elaboration – Iteration 1.2

# Design Model

## Dynamic Behavior

*[Create the interaction diagrams (1 sequence, 1 communication diagrams) for 2 relevant scenarios]*

**Communication diagram** for the scenario: register a new user.

The user will be by default an athlete(the user of the application cannot make himself an admin). A boolean field isAdmin will be set by default on false on a new user. The user will give in the UI the username, password, height, current weight and goal(muscle gain, maintenance or fat loss). Data will be communicated to the Controller(GET method) and the corresponding method will handle the information. With a service method, the calories and macros will be calculated and the new user will be then inserted in the database table. The user will get a confirmation for the successful login or a warning if the given info is not valid.

*A close up of text on a white background

Description automatically generated*

*Communication Diagram – Register a new Athlete*

**Sequence diagram** for the scenario: add a food to diary.

The user selects the food and gives the quantity in grams in the UI. The data is sent to the controller and handled with a service method: if the current day already exists in the diary, a new food is added( a new entry in the table connecting the food table with the diary table) and the kcal and macro values are updated for the current day. Else a new diary entry is created.

*A close up of a map

Description automatically generated*

*Sequence Diagram – Add a food to diary*

## Class Design

*[Create the UML class diagram; apply GoF patterns and motivate your choice]*

*A close up of text on a white background

Description automatically generated*

# Data Model

*[Create the data model for the system.]*

A close up of text on a white background

Description automatically generated

There is a one to many relationship between user and diary (a user can have many diary entries, one for each day). The many to many relationship between Diary and Food is solved with a table containing the foreign keys for foodID and diaryID and also the quantity of the food. A diary entry (a day) can contain many foods and a food can be eaten in more than one day.

# Test Strategy

*[Present the used testing methods and the associated test case scenarios.]*

For testing I will use unit tests and the classic “try and see what happens” tests. My application will be loosely coupled, which will make it easier to create mocked data(implementations) for unit tests. The tests will be made for each layer of my application. The repository layer tests will verify the CRUD operations performed on the database. The next ones, the service layer tests will check the correct execution for the logic of the application and then the same for controller layer tests.

One test case scenario will be: an athlete wants to register. He gives his name, password, height, current weight and goal (some mocked data). The service method responsible for calculating the necessary calories and macros should return the expected values or else the test fails.

Another test case scenario will be: an athlete wants to add a new meal for the current day. He selects the food and gives the quantity. A service method responsible for updating the diary values for total calories and macros and creating a new meal will be called. The returned values will be compared with the expected ones for the given input data to establish if the test succeeded or not.

# Elaboration – Iteration 2

# Architectural Design Refinement

*[Refine the architectural design: conceptual architecture, package design (consider package design principles), component and deployment diagrams. Motivate the changes that have been made.]*

# Design Model Refinement

## *[Refine the UML class diagram by applying class design principles and GRASP; motivate your choices. Deliver the updated class diagrams.]*

# Construction and Transition

# System Testing

*[Describe how you applied integration testing and present the associated test case scenarios.]*

# Future improvements

*[Present future improvements for the system]*