FH University of Applied Sciences TECHNIKUM WIEN

Sports Exercise Battle (SEB)

This HTTP/REST-based server is built to be a platform for sports exercise tournaments of multiple users. Users hereby can challenge each other in the discipline of "pushups" or in other words the server tracks: "Who can do more push-ups in 2 minutes?".

- a **user** is a registered sportsman/sportswoman with **credentials** (unique username, password).
- a **user** can manage his/her **history** of push-ups which consists of the information "count" and "duration of exercise".
- the **history** is managed server-side.
- a **user** has multiple **push-up-record-entries** in his **library** and can add further ones after training.
- each entry starts a **tournament** of 2 minutes.
- The **user** with the most push-ups (in sum) entered in the 2min time-frame gets +2 ELO; other **users** that participated in the **tournament** get -1 ELO.
- in case of a draw in a tournament the users sharing the win get +1 ELO.
- persist the data into a **database**

Mandatory Requirements

- working REST-Interface over HTTP
- at least 10 unit-tests
- at least 5 automated integration tests work (provided curl || postman ||
- insomnia || custom-client)
- make the battles comprehensible by creating any kind of log information
- use the mandatory technologies (see: Mandatory Technologies)

In case the mandatory requirements are not met you will receive 0 points for the submission.

Game mechanics

Users can

- register and login to the server,
- check history,
- check user stats (= return ELO value and count of push-ups overall)
- check scoreboard (ELO and total count of entries per user),
- run tournaments

The data should be persisted in a postgreSQL database.

Further features:

- editable profile page
- security (check for the token that is retrieved at login on the server-side, so
- that user actions can only be performed by the corresponding user itself)
- persistence (be able to stop and start the server and have all scores/data in
- place as before (you don't need to consider pending calls))

HandIns

Create an application in Java or C# to spawn a REST-based (HTTP) server that acts as an API for possible frontends (WPF, JavaFX, Web, console). The frontend is not part of this project! You are not allowed to use any helper framework for the HTTP communication, but you are allowed to use nuget (JSON.NET) /mvn-packages (Jackson) for serialization of objects into strings (vice versa). Test your application with the provided curl script (or integration test) and add unit tests (10+) to verify your application code.

Add a **unique feature (mandatory)** to your solution.

Hand in the latest version of your source code as a zip in moodle (legal issue) with a README.pdf - file pointing to your git-repository.

Add a protocol as pdf - file with the following content:

- protocol about the technical steps you made (designs, failures and selected solutions)
- explain why these unit tests are chosen and why the tested code is critical
- track the time spent with the project
- consider that the git-history is part of the documentation (no need to copy it into the protocol)

The final presentation is done during a panel exam.

- present the working solution with all aspects
- show the integration-test (curl script or your adapted solution) and show
- adaptions you needed to make
- execute the integration tests and explain the results
- execute the unit tests

Please be aware that the curl-tests might be altered throughout the course (last change: one week before final presentation). In case of custom adjustments (depending on your implementation) in the curl scripts please adapt these final versions as well.

Mandatory Technologies

- C# / Java as a console application
- TCP
- HTTP
- JSON (nuget (JSON.NET) /mvn-packages (Jackson); in and out format)
- SQL (no OR-Mapper)
- PostgreSQL
- NUnit / JUnit

Grading

- 25: functional requirements
  - tournaments
    - start,
    - draw possible,
    - update ELOs
  - User Management (Register, ProfilePage)
  - Scoreboard and User Stats Management
  - mandatory unique feature
- 10: non-functional requirements
  - Token-based security
  - Persistence (DB)
  - Unit Tests
  - Integration Tests (provided curl or alternatively a custom app that works in an automated way)
- 05: protocol
  - design / lessons learned
  - unit test design
  - time spent
  - link to git