Party Playlist Battle (PPB)

This HTTP/REST-based server is built to be a platform for creating playlist on parties in a fun way. In addition, users can challenge each other in a game of rock-paper-scissors-lizard-spock to define the priority of your multi-media content.

- a **user** is a registered content-creator with **credentials** (unique username, password).
- a **user** can manage his **library**.
- the **multi-media-content** to play in a **playlist** consists of: a name and multiple attributes (file-type, file-size, title, artist, album, rating, genre, length, path,...).
- globally there is only 1 playlist to manage
- a **user** has multiple **multi-media content** in his **library** and can add and remove multi-media content there.
- a **user** can make a request to schedule his **multi-media-content** into the **playlist** (automatically added at the end of the playlist)
- only an **administrator** can reorder items in the playlist
- to become **administrator** you need to send a battle-request to the PPB-server which keeps you in the server for 15 sec (= a **tournament**) and sets you to be an **online-user**
- the **online-users** play rock-paper-scissors-lizard-spock in 5 **rounds** per **battle** against all other **online-users** to become **administrator** during the current **tournament**.
- the choice which action to use in the rock-paper-scissors-lizard-spock **battle** is defined in the battle-request-payload (exactly 5 actions) which are used in order
- for each won round you get +1 to round-score-counter (draws/loss don't change round-score-counter). The **user** with the highest round-score-counter wins the battle (round-score-counter will be reset after this calculation).
- The battle winner gets +1 to the battle-score-counter.
- in case of a draw in a battle the users will be locked for the current **tournament** and nobody becomes administrator
- the final administrator is the **online-user** with the highest battle-score-counter. In case of a draw nobody becomes administrator.
- after an administrator is chosen the tournament-score values in the scoreboard need to updated for the winning and losing users.
- the battle-score-counters are reset at the end of the tournament.
- 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,
- manage your library,
- manage the global playlist (only allowed as administrator),
- define a set of 5 rock-paper-scissors-lizard-spock actions,
- battle against each other and
- compare their stats in the scoreboard.

The data should be persisted in a postgreSQL database.

Further features:

- scoreboard (= sorted list of win statistics)
- editable profile page
- user stats
    - especially how often a user became administrator (+1 tournament-score-counter for win, -1 for loss, starting value: 100; draws don't change this value; higher sophisticated statistics system welcome)
- 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))

The Battle Logic

Actions:

- rock (R)
- paper (P)
- scissors (S)
- lizard (L)
- spock (V Vulcan sometimes Vulcanian)

Effectivenesses see:
https://en.wikipedia.org/wiki/File:Rock_paper_scissors_lizard_spock.png

There is no attacker or defender. All parties are equal in each round. Each round can be won, lost or there might be a draw. The sum of win/loss/draw of the rounds results in the win/loss/draw of a battle in the tournament. In case of a draw both parties are removed from the tournament and have no further chance to become the administrator in this tournament. As a result of the battle we want to return a log which describes the battle in great detail. Afterwards the player stats (see scoreboard) need to be updated (count of games played and calculate new score).

Example

Win A at battle A against B:

A (R,R,R,R,R) vs B (S,S,S,S,S) => winners: A,A,A,A,A => +5 for A => +1 battle point for A / 0 battle points for B

in case the tournament consists of only 2 players A wins the tournament and becomes Administrator. The tournament-score is updated for A with +1 in the scoreboard and

with -1 for B in the scoreboard.

Draw with 3 players A,B,C

First battle A,B

A (R,R,R,R,R) vs B (R,R,R,R,R) => no winner because 5 draws => both users are blocked for the rest of the tournament.

C wins automatically no matter which actions are defined

Tournament with players A,B,C,D,E,F (no draw)

Battle: A->B, A->C, A->D, A->E, A->F, B->C, B->D, B->E, B->F, C->D, C->E, C->F, D->E, D->F. E->F

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 (20+) 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.txt (or md or pdf)-file pointing to your git-repository and an explaining video.

Add a protocol as plain-text file (txt or md) or pdf 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 by handing in a video (10min)

- present your design protocol (with unit-test decisions)
- 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
    - Battle
        - play rounds,
        - draw possible,
        - clean log of a battle,
        - correct usage of score-counters
        - correct action handling (see effectivenesses)
    - 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