Media Ratings Platform (MRP) – Specification

Create a standalone application in Java or C# that implements a RESTful HTTP server that acts as an API for possible frontends (e.g., Mobile, Web, console). **The front end is not part of this project!**

Features

This HTTP/REST-based server provides a platform for users to manage media content (movies, series, and games), rates them, and receive content recommendations:

A **user**:

- *registers* and *logs in* with unique **credentials** (username, password)
- can *view* and *edit* a **profile** with personal statistics
- can *create, update, and delete* media entries
- can *rate* media entries from 1–5 stars and optionally *write a comment*
- can *edit* or *delete* their own ratings
- can *like* other users' ratings (1 like per rating)
- can *mark* media entries as favorites
- can *view* their own rating history and list of favorites
- *receives recommendations* based on previous rating behavior and content similarity

A **media entry**:

- represents either a **movie**, **series**, or **game**
- consists of **title**, **description**, **media type**, **release year**, **genre(s)**, and **age restriction**
- is created by a user and can only be edited or deleted by its **creator**
- includes a **list of ratings** and a calculated **average score**
- can be marked as favorite by other users

A **rating**:

- is tied to a specific media entry and a specific user
- contains: **star** value (1–5), optional **comment**, **timestamp**
- can be **liked by** other users
- can be edited or deleted by the **user who created it**
- requires **confirmation** by the creator before the comment becomes publicly visible (moderation feature)
    - comments are not publicly visible until *confirmed by the author*

Use-Cases:

- register and log in
- manage media entries (CRUD)
- rate and comment on media
- like other users' ratings
- search for media entries by title (partial matching)
- filter media by genre, media type, release year, age restriction, or rating
- sort results by title, year, or score
- mark and unmark media as favorites
- view a public leaderboard of the most active users
- receive recommendations based on:
    - genre similarity to previously highly rated media
    - content similarity (matching genres, media type, and age restriction)

Additional Features:

- display a leaderboard sorted by the number of ratings per user
- user profile includes personal statistics (e.g., total ratings, average score, favorite genre)
- favorite list for quickly accessing user-marked favorite entries

Implementation Requirements

- Build a REST server and implement the endpoints according to the HTTP specification. You may use a HTTP helper-framework for the pure-HTTP protocol stack like HttpListener (no ASP.Net, Spring or JSP/JSF allowed)
- You may use packages for object serialization (e.g., Jackson, Newtonsoft.JSON or else)
- Data must be persisted in a PostgreSQL database (which may run in Docker)
- Provide a Postman collection or a curl-script for integration testing
- Create at least 20 unit tests to validate core business logic

HTTP Specification

Return the correct HTTP response codes:

- 2XX → success
- 4XX → client-side error (e.g., invalid input, missing authentication)
- 5XX → server-side error (e.g., database unavailable)

Token-based Authorization

For authorization token-based security is used. Every incomming request (except for registration and login) must be pre-checked if the user is correctly logged in and the user has the right to access the resource. If not return the corresponding HTTP client-side error.

Sample for login:

```
HTTP POST http://localhost:8080/api/users/login
Content-Type: application/json

{ "Username":"mustermann", "Password":"max" }
```

This should return a token string, e.g. “mustermann-mrpToken”.

The token string has to be passed in all subsequent requests as HTTP Authentication Bearer Header Parameter, which has to be checked and handled accordingly (store the existing tokens and the matched user in your application).

Sample for the call to request the user’s profile information:

```
HTTP GET http://localhost:8080/api/users/mustermann/profile
Authentication: Bearer  mustermann-mrpToken
Accept: application/json
```

Will return the user’s profile JSON in the HTTP body.

Hand-Ins

The submission is done in two steps:

1. **Intermediate-Submission** (class 13): Includes HTTP server, user registration/login, and basic media management (CRUD)
2. **Final Submission** (class 22): adds full business logic, rating system, favorites, filtering, and recommendations

Hand in the latest version of your source code as a zip in Moodle (legal issue) that includes:

- Source code
- README.txt or README.md with link to GitHub repository
- Postman collection or curl-script demonstrating all relevant endpoints

- protocol.txt or protocol.md with a detailed development report

Add a protocol document with the following content:

- Description of technical steps and architecture decisions
- Explanation of unit test coverage and why specific logic was tested
- Notes on problems encountered and how they were solved
- Estimated time tracking for each major part of the project
- consider that the git-history is part of the documentation (no need to copy it into the protocol)

See the corresponding Checklist Excel sheets for the MUST-HAVES, Grading-Items and Grading-Points. Late submissions are not accepted! Missing hand-ins or missing MUST-HAVEs will automatically grade the submission with 0 points!

Final Presentation

For the final presentation in class 22-25, be prepared with your

- working solution already started on your machine.
- set up your environment so you can start the postman or curl tests directly.
- open your design (see protocol) to show your architecture / approach.

Optional Features

With optional features implemented you can compensate for possible errors in the implementation above. Nevertheless, it is not possible to exceed the maximum number of points (= 100%).