Software Development Project Protocol

Student Name: Alberta Hasi
Course: SWEN1

Date: *16.01.2026*

GitHub Repository:
https://github.com/hasii015/SEW_Projekt/

Inhalt

1. Project Summary

This project implements the backend of a Media Ratings Platform using C# and a pure HTTP server based on HttpListener. The system provides a REST API that allows users to register, log in using token-based authentication, and manage media entries through CRUD operations. Authenticated users can rate media, confirm rating comments, like other users' ratings, and manage favorite media entries. The backend also includes leaderboard and recommendation endpoints, as well as query support for searching and filtering media. All communication is handled via HTTP requests and JSON responses without using any web frameworks, and all data is stored in memory for this milestone.

2. System Architecture

The request flow follows a clear pipeline: HttpListener receives the request, HttpRestServer triggers the RequestReceived event, a HttpRestEventArgs object is created, and Handler.HandleEvent dispatches the request to the correct handler. Each feature is implemented in its own handler class, which improves modularity and makes it easy to add new endpoints without changing the server core.

A key architecture decision was ensuring handlers only check authentication after verifying the endpoint they handle. This prevents unrelated handlers from accidentally blocking public endpoints such as /api/users/register and /api/users/login.

3. Authentication & Session Handling

Authentication is implemented using Bearer tokens. Users log in via POST /api/users/login, and a random session token is generated and stored in an in-memory session dictionary. For protected endpoints, the client must send Authorization: Bearer <token> in the HTTP header. The server resolves the session centrally using HttpRestEventArgs.Session, which parses the token and calls Session.Get(token). If no session is found, the request is rejected with 401 Unauthorized. Sessions automatically expire after 30 minutes of inactivity.

4. REST Api Endpoints

The backend exposes several REST endpoints to manage users, authentication, media entries, ratings, favorites, recommendations, and leaderboard data. All protected endpoints require a valid Bearer token in the HTTP Authorization header.

4.1 User Registration – POST /api/users/register

Registers a new user in the system using a username and password.
If the registration is successful, the server returns a success confirmation.

4.2 Login – POST /api/users/login

Authenticates a user and creates a new session.
A random session token is generated and returned to the client. This token must be used for all further authenticated requests.

4.3 List All Media – GET /api/media

Returns a list of all stored media entries.
This endpoint requires authentication. If no token is provided or the token is invalid, the request is rejected with **401 Unauthorized**.

4.4 Create Media – POST /api/media

Creates a new media entry.
The authenticated user automatically becomes the creator of the media entry. The request body contains details such as title, description, media type, release year, genres, and age restriction.

4.5 Get Media by ID – GET /api/media/{id}

Returns a single media entry by its unique ID.
If the ID does not exist, the server responds with **404 Not Found**.

4.6 Update Media – PUT /api/media/{id}

Updates an existing media entry.
Only the original creator of the media entry is allowed to perform updates. If another user attempts to update the entry, the request is rejected with **403 Forbidden**.

4.7 Delete Media – DELETE /api/media/{id}

Deletes a media entry from the system.
Only the creator is allowed to delete the entry. After deletion, the entry is no longer available through the API.

Ratings

4.8 Rate Media – POST /api/media/{id}/rate

Creates a rating for a media entry.
The user submits a star rating and an optional comment. If the user already rated the same media entry before, the rating is updated instead of creating a second one.

4.9 Update Rating – PUT /api/ratings/{id}

Updates an existing rating (stars/comment).
Only the author of the rating can update it. After updating, the rating comment requires confirmation again.

4.10 Confirm Rating Comment – POST /api/ratings/{id}/confirm

Confirms a rating comment.
Only the author of the rating can confirm their comment. This mechanism ensures that comments are only visible after confirmation.

4.11 Like Rating – POST /api/ratings/{id}/like

Likes a rating from another user.
Users cannot like their own ratings, and duplicate likes are prevented.

Favorites

4.12 Mark Media as Favorite – POST /api/media/{id}/favorite

Adds a media entry to the authenticated user's list of favorites.

4.13 Unmark Media as Favorite – DELETE /api/media/{id}/favorite

Removes a media entry from the authenticated user's list of favorites.

4.14 List Favorites – GET /api/users/{id}/favorites

Returns a list of favorite media entries for the authenticated user.

User Profile & History

4.15 Get User Profile – GET /api/users/{id}/profile

Returns the profile information of the authenticated user (e.g., email and favorite genre).

4.16 Update User Profile – PUT /api/users/{id}/profile

Updates profile information of the authenticated user, such as email and favorite genre.

4.17 Get Rating History – GET /api/users/{id}/ratings

Returns a list of ratings created by the authenticated user.

Leaderboard

4.18 Get Leaderboard – GET /api/leaderboard

Returns ranking information based on user activity (e.g., rating count or engagement), depending on the implementation.

Recommendations

4.19 Get Recommendations by Genre – GET /api/users/{id}/recommendations?type=genre

Returns a list of recommended media entries based on the user's favorite genre.

4.20 Get Recommendations by Content – GET /api/users/{id}/recommendations?type=content

Returns recommendations based on similarity in rating behaviour and user interests (content-based approach).

Query Support (Filtering & Sorting)

4.21 Search & Filter Media – GET /api/media?title=...&genre=...&sortBy=...

Supports query parameters to search and filter media entries, for example by title, genre, media type, release year, age restriction, and sorting criteria.

5. SOLID Principles

The system applies multiple SOLID principles to ensure clean structure and maintainability. The Single Responsibility Principle is implemented by separating concerns between different components: HttpRestServer handles network communication, each handler class processes only its own endpoint group, and store classes manage in-memory persistence. For example, MediaHandler only manages media CRUD endpoints, while rating and favorite endpoints are handled by dedicated handlers.

The Open/Closed Principle is supported through the handler-based architecture. New functionality can be added by creating an additional handler class without modifying the core server logic or existing handlers. For example, endpoints such as leaderboard and recommendations were implemented as separate handlers and automatically integrated through reflection-based handler discovery.

6. Testing

Testing was performed using the provided Postman collection and PowerShell Invoke-RestMethod commands to validate the full request/response behaviour of the API. The tests focus on the most critical logic: authentication handling, permission checks, endpoint correctness, and the correct JSON response format.

The following areas were tested: user registration and login, token-based authorization for protected endpoints, media CRUD operations including creator-only update/delete rules, rating creation and updates, confirmation of rating comments by the author, rating likes by other users, favorite add/remove functionality, leaderboard aggregation, recommendation results for both genre-based and content-based modes, and query support for filtering and sorting media.

7. Problems Encountered & Solutions

During implementation, several issues were encountered and resolved. One major issue was that some handlers checked authentication too early and accidentally blocked public endpoints such as /api/users/register, returning “Missing or invalid token.” This was fixed by ensuring each handler verifies the exact endpoint first and only then requires authentication.

Another issue involved incorrect JSON handling for the genres field, since the specification expects an array but the internal model stored genres as a comma-separated string. This was solved by converting genres into a proper JsonArray in API responses while keeping internal storage simple.A third issue was endpoint overlap, where /api/media/{id}/rate and /api/media/{id}/favorite could be incorrectly handled by MediaHandler. This was fixed by letting MediaHandler skip these paths so the correct specialized handlers could process them.

8. Time Tracking

Task	Time
Project setup + server base	2h
User registration + login + sessions	4h
Media CRUD implementation	3h
Query support (filter/search/sort)	2h
Ratings (rate/update/confirm/like)	4h
Favorites	2h
User profile + history endpoints	2h
Leaderboard	1h
Recommendations	3h
Testing + debugging	6h
Documentation	2h
Total	**31h**