Software Development Project Protocol

Student Name: Alberta Hasi
Course: SWEN1

Date: *03.12.2025*

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 allows users to register, log in using token-based authentication, and manage media entries through a REST API. Authenticated users can create, view, update, and delete media entries. All communication is handled via HTTP requests and JSON responses, without using any web frameworks. For the intermediate milestone, all data is stored in memory and the focus lies on correct REST behaviour, authentication, and CRUD functionality.

2. System Architecture

The application is built using a handler-based REST architecture. At its core, the HttpRestServer class listens for incoming HTTP requests using HttpListener. Whenever a request is received, the RequestReceived event is triggered and passed to the static method Handler.HandleEvent, which is responsible for dispatching the request to the appropriate handler.

Each endpoint is implemented in its own handler class (UserHandler, SessionHandler, and MediaHandler). These handlers inherit from the abstract base class Handler and override the Handle method. The handlers decide whether they are responsible for a request based on the HTTP path and method. The system uses reflection to automatically discover all handlers, meaning no manual routing configuration is required.

The application logic is separated from data storage. Domain objects such as User, Session, MediaEntry, and Rating are located in the system layer, while static store classes such as MediaStore and the session dictionary inside Session are responsible for in-memory persistence. This clear separation ensures good maintainability and supports future extensions such as database integration.

3. Authentication & Session Handling

Authentication is implemented using Bearer tokens:

1. User logs in via POST /sessions.
2. A random session token is generated.
3. The token is stored in a static session dictionary.
4. For protected endpoints, the client must send:

   Authorization: Bearer <token>

5. The server resolves the session using Session.Get(token).
6. If no session is found, the request is rejected with 401 Unauthorized.
7. Sessions expire automatically after 30 minutes of inactivity.

4. REST Api Endpoints

The backend exposes several REST endpoints to manage users, sessions, and media. All protected endpoints require a valid Bearer token in the HTTP Authorization header.

4.1 User Registration – POST /users

Registers a new user in the system using a username, full name, email, and password.
If the registration is successful, a confirmation message is returned.

4.2 Login – POST /sessions

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 /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 /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 /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 /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 /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.

5. SOLID Principles

The system follows several key SOLID design principles to ensure clean structure and maintainability. The Single Responsibility Principle is applied by separating concerns across different components: the HttpRestServer handles network communication, handlers process specific request types, and store classes manage in-memory data persistence. Each class has exactly one well-defined responsibility.

The Open/Closed Principle is supported through the handler architecture. New functionality can be added by creating additional handler classes without modifying the existing server logic. Furthermore, the system demonstrates a clear Separation of Concerns between networking, business logic, and data storage, making the code base easy to extend and test.

6. Testing

6.1. User Registration – POST /users

```
C:\Users\Berta>curl -X POST "http://localhost:12000/users" -H "Content-Type:
 application/json" -d "{\"username\":\"alice\",\"fullname\":\"Alice\",\"emai
l\":\"alice@example.com\",\"password\":\"secret\"}"
{
  "success": true,
  "message": "User created."
}
```

6.2. Login – POST /sessions

```
C:\Users\Berta>curl -X POST "http://localhost:12000/sessions" -H "Content-Ty
pe: application/json" -d "{\"username\":\"alice\",\"password\":\"secret\"}"
{
  "success": true,
  "token": "MbDOQqEM8V0bpEzWYnFaexmH"
}
```

6.3. Authentication - Access Media Without Token

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media"
{
  "success": false,
  "reason": "Missing or invalid token."
}
```

6.4. List All Media – GET /media

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media" -H "Authorization:
 Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": true,
  "items": []
}
```

6.5. Create Media – POST /media

```
C:\Users\Berta>curl -X POST "http://localhost:12000/media" -H "Content-Type:
 application/json" -H "Authorization: Bearer MbDOQqEM8V0bpEzWYnFaexmH" -d "{
\"title\":\"Interstellar\",\"description\":\"Interstellar Description\",\"me
diatype\":\"movie\",\"releaseYear\":2014,\"genres\":\"Sci-Fi, Action\",\"age
Restriction\":16}"
{
  "success": true,
  "id": 2
}
```

6.6. Media List

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media" -H "Authorization:
 Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": true,
  "items": [
    {
      "Id": 2,
      "Title": "Interstellar",
      "Description": "Interstellar Description",
      "MediaType": "movie",
      "ReleaseYear": 2014,
      "Genres": "Sci-Fi, Action",
      "AgeRestriction": 16,
      "CreatedBy": "alice"
    }
  ]
}
```

6.7. Get Media by ID – GET /media/{id}

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media/2" -H "Authorizatio
n: Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "Id": 2,
  "Title": "Interstellar",
  "Description": "Interstellar Description",
  "MediaType": "movie",
  "ReleaseYear": 2014,
  "Genres": "Sci-Fi, Action",
  "AgeRestriction": 16,
  "CreatedBy": "alice"
}
```

6.8. (Error Case) Get Non-Existing Media

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media/3" -H "Authorizatio
n: Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": false,
  "reason": "Media not found."
}
```

6.9. Update Media – PUT /media/{id}

```
C:\Users\Berta>curl -X PUT "http://localhost:12000/media/2" -H "Content-Type
: application/json" -H "Authorization: Bearer MbDOQqEM8V0bpEzWYnFaexmH" -d "
{\"title\":\"Interstellar (Edit)\",\"description\":\"Edited description\"}"
{
  "success": true,
  "message": "Media updated."
}
```

6.10. Verify Update Worked

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media/2" -H "Authorizatio
n: Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "Id": 2,
  "Title": "Interstellar (Edit)",
  "Description": "Edited description",
  "MediaType": "movie",
  "ReleaseYear": 2014,
  "Genres": "Sci-Fi, Action",
  "AgeRestriction": 16,
  "CreatedBy": "alice"
}
```

6.11. Creator Protection Test

```
C:\Users\Berta>curl -X POST "http://localhost:12000/users" -H "Content-Type:
 application/json" -d "{\"username\":\"hasii015\",\"fullname\":\"Alberta Has
i\",\"email\":\"berta.hasi@gmail.com\",\"password\":\"secret111\"}"
{
  "success": true,
  "message": "User created."
}
C:\Users\Berta>curl -X POST "http://localhost:12000/sessions" -H "Content-Ty
pe: application/json" -d "{\"username\":\"hasii015\",\"password\":\"secret11
1\"}"
{
  "success": true,
  "token": "EyvRH12J0F6axuMsQ9nkqr5k"
}
C:\Users\Berta>curl -X PUT "http://localhost:12000/media/2" -H "Content-Type
: application/json" -H "Authorization: Bearer EyvRH12J0F6axuMsQ9nkqr5k" -d "
{\"title\":\"Hacked title\"}"
{
  "success": false,
  "reason": "Only creator may update this media."
}
```

6.12. Delete Media – DELETE /media/{id}

```
C:\Users\Berta>curl -X DELETE "http://localhost:12000/media/2" -H "Authoriza
tion: Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": true,
  "message": "Media deleted."
}
```

6.13. Verify Deletion

```
C:\Users\Berta>curl -X GET "http://localhost:12000/media/2" -H "Authorizatio
n: Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": false,
  "reason": "Media not found."
}
C:\Users\Berta>curl -X GET "http://localhost:12000/media" -H "Authorization:
 Bearer MbDOQqEM8V0bpEzWYnFaexmH"
{
  "success": true,
  "items": []
}
```