API Documentation: Movies API with JWT Authentication and Docker Compose.

Overview

The Movies API provides CRUD functionality for managing movies and allows users to authenticate using **JWT Bearer Tokens**. The API is containerized using **Docker** and **Docker Compose** to streamline deployment and ensure that the application runs in a consistent environment.

Technologies Used

- .NET Core 8.0
- Entity Framework Core (EF Core) Database-First
- SQL Server
- JWT Bearer Authentication
- Docker and Docker Compose

Prerequisites

To run this project locally, you will need:

- **Docker** and **Docker Compose** or Visual Studio installed.
- **Postman** or another API testing tool (for testing API endpoints)

Getting Started

A. Docker Compose Setup

The docker-compose.yml file orchestrates the API and the SQL Server database.

```yaml
networks:
    moviesapi:



services:
  moviesApidb:
    container_name: demo-movies-api-db
    image: mcr.microsoft.com/mssql/server:2022-latest
    ports:
      - 8002:1433
    environment:
      - ACCEPT_EULA=Y
      - MSSQL_SA_PASSWORD=Password@1234
    volumes:
      - db_data:/var/opt/mssql
    networks:
      - moviesapi
  moviesapi:
    container_name: demo-movies-api
    image: ${DOCKER_REGISTRY-}moviesapi
    build:
      context: .
      dockerfile: Movies API/Dockerfile
    ports:
      - 8001:8080
    depends_on:
      - moviesApidb
    environment:
      - DB_HOST=moviesApidb
      - DB_NAME=Movies
      - MSSQL_SA_PASSWORD=Password@1234
    networks:
      - moviesapi
volumes:
  db_data:
```

B. Running the Application

1. **Open the Solution in Visual Studio 2022**:
    - Make sure the project is set up with Docker support. If not, right-click on the project in **Solution Explorer** and select **Add > Docker Support**.
2. **Build and Run with Docker Compose**:
    - Open **Docker Compose** from the **Solution Explorer**.

- Right-click on the docker-compose project and select **Set as Startup Project**.
- Press **F5** to build and run the project with Docker Compose. Visual Studio will spin up the necessary containers (API and SQL Server).

Docker

1. **To run the project in docker:**
   - Search for **docker-compose.yml** file.
   - Make sure that the connection string of the **appSettings.json** is set to point to the database container.
   - Raise the service with the command **docker-compose up -d.**

API Endpoints

A. Authentication <

POST /api/users/login

- **Description**: Authenticate a user and return a JWT token.
- **Request Body:**

```
{
"email": "Jane@gmail.com",
"password": "qwerty"
}
```

- **Response**:

```
{
    "success": 1,
    "message": "All ok",
    "data": {
        "email": "Jane@gmail.com",
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
            eyJuYW1laWQiOlsiMyIsIkphbmVAZ21haWwuY29tIl0sIm5iZiI6MTcyODQyODMwMywiZXhwIjoxNzI4NDI4OTAzLCJpYXQiOjE3Mjg0MjgzMDN
            9.9BiP0Ql1-IncT5L3VqXLWVhw2aK7ubEiBBZMoakffWE"
    }
}
```

B. Movies

GET /api/movies

- **Description**: Retrieve a list of movies
- **Response**


```
{
    "success": 1,
    "message": null,
    "data": [
        {
            "id": 1,
            "name": "Alien: Romulus",
            "idGenre": 1,
            "genre": "Horror",
            "idAgeRating": 4,
            "rating": "R",
            "releaseDate": "2024-08-16"
        },
        {
            "id": 2,
            "name": "Dune: Part 2",
            "idGenre": 3,
            "genre": "Sci-fi",
            "idAgeRating": 3,
            "rating": "PG-13",
            "releaseDate": "2024-03-01"
        },
```


POST/api/movies

- **Description**: Add a movie to list of movies
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**


```
{
    "Name": "The Batman",
    "IdGenre": 4,
    "IdAgeRating": 3,
    "ReleaseDate": "2022-03-04"

}
```

- **Response**

```
{
    "success": 1,
    "message": "Item added",
    "data": null
}
```

PUT/api/movies

- **Description**: Updates a movie from list of movies
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**

```
{
    "id": 4,
    "name": "Aliens: the return",
    "idGenre": 1,
    "idAgeRating": 4,
    "releaseDate": "1986-07-14"
}
```

- **Response**

```
{
    "success": 1,
    "message": "Item updated",
    "data": null
}
```

DELETE/api/movies/id

- **Description**: Deletes a movie from the list of movies
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Response**

```
{
    "success": 1,
    "message": "Item deleted",
    "data": null
}
```

C. Genres

GET /api/genres

- **Description**: Retrieve a list of genres
- **Response**


```
{
    "success": 1,
    "message": "All ok",
    "data": [
        {
            "id": 1,
            "genre1": "Horror",
            "movies": []
        },
        {
            "id": 3,
            "genre1": "Sci-fi",
            "movies": []
        },
```


POST/api/genres

- **Description**: Add a genre to list of genres
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**


```
{
    "genre": "Western"
}
```


- **Response**


```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

PUT/api/genres

- **Description**: Updates a genre from the list of genres
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**

```
{
    "id": 8,
    "genre": "Drama"
}
```

- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

DELETE/api/genres/id

- **Description**: Deletes a genre from the list of genres
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

D. Movies Rating

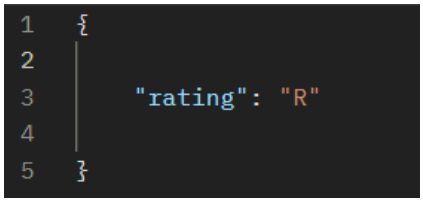
GET /api/moviesRating

- **Description**: Retrieve a list of movie ratings.
- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": [
        {
            "id": 1,
            "rating": "G",
            "movies": []
        },
        {
            "id": 2,
            "rating": "PG",
            "movies": []
        },
```

POST/api/moviesRating

- **Description**: Add a movie rating to a list of movie ratings.
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**



```
{

    "rating": "R"

}
```

- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

PUT/api/moviesRating

- **Description**: Updates a movie rating to a list of movie ratings.
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Request body**

```
{
    "id": 6,
    "rating": "RE"

}
```

- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

DELETE/api/ moviesRating/id

- **Description**: Deletes a movie rating to a list of movie ratings.
- **Headers**: Authorization: Bearer <JWT_TOKEN>
- **Response**

```
{
    "success": 1,
    "message": "All ok",
    "data": null
}
```

Testing with Postman

To test the API:

1. Use Postman to send a POST request to /api/users/login to retrieve a JWT token.
2. Use the token in the Authorization header for subsequent requests to secured endpoints (e.g POST/api/movies).

Example Authorization header: