Github:https://github.com/Nicuras/ReelReviewAPI

Introduction:

This project has been quite challenging for me. After spending around 15 hours troubleshooting my API, I concluded that the errors were due to circular references. Now that I have a working API, I will explain what I did to make it work and how I could have done it differently.

First, I would have approached my database scheme differently, especially with my junction tables. In SQL Server, we use Primary Key (PK) and Foreign Key (FK) to maintain relationship integrity. However, after scaffolding my API, it seemed to lose some of the logic. For instance, in my model classes, there was code that defined a navigation property as a collection of objects and initialize it to an empty list. This code added more to my requested body. Thus, I removed this code from every class model, and it worked.

Overview/Instructions:

I have developed a movie review API that allows users to search for reviews on movies they are interested in. This project was challenging, especially since it was my first time working on an API project independently. Making everything work together was not easy, and my API is not entirely flawless.

Instructions:

To create this API, you need to start by creating a new project under ASP.NET Web API. Open Visual Studio, create a new project, select "ASP.NET Web Application," give it a name, choose the "API" template, and click "Create." Visual Studio will create a new empty project waiting to be scaffolded. Once you scaffold your database to your web API, you can create your controllers that will do most of the heavy lifting in your web API. Finally, after you finish creating your controllers and everything seems to be routed correctly, you can test your API with Swagger.

Database:

Start by creating a new database with the name you want. Then, map out how you want your database to look and flow (this is not mandatory, but it helps).

Create your tables and populate them with columns. Once your columns are created, you can start with your relationship integrity.
Testing:
When testing with Swagger, you can test the API and each controller with the endpoint. Ensure that you change the requested body to request what you want for that endpoint. For example:

```
{
"actorId": 0,
"firstName": "string",
"lastName": "string",
"nationality": "string",
"birth": "string",
"movieActors": [
{
"actorId": 0,
"movieId": 0,
"movieActorId": 0,
"actor": "string",
"Movie":
```

ReelReview Programmer's Guide:

ReelReview is a web application that provides movie enthusiasts with a platform to review, rate, and discuss their favorite movies. This guide is for programmers who want to learn how to use ReelReview's API.

Actors:

GET: api/actors - Returns a list of all actors.
GET: api/actors/{id} - Returns a single actor based on the provided ID.
POST: api/actors - Creates a new actor based on the provided data in the request body.
PUT: api/actors/{id} - Updates an existing actor based on the provided ID and data in the request body.
DELETE: api/actors/{id} - Deletes an existing actor based on the provided ID.

Director:

GET: api/directors - Retrieves all directors from the database.
GET: api/directors/{id} - Retrieves a specific director by ID from the database.
PUT: api/directors/{id} - Updates an existing director in the database.
POST: api/directors - Adds a new director to the database.
Movies:
GET: api/movies - Retrieves a list of all movies from the database.
GET: api/movies/{id} - Retrieves a specific movie by ID from the database.
PUT: api/movies/{id} - Updates an existing movie in the database.
POST: api/movies - Adds a new movie to the database.
DELETE: api/movies/{id} - Deletes a movie from the database based on the provided ID.

Movie Reviews:

GET: api/moviereviews - Retrieves all movie reviews from the database.
GET: api/moviereviews/{id} - Retrieves a specific movie review by ID from the database.
POST: api/moviereviews - Adds a new movie review to the database.
PUT: api/moviereviews/{id} - Updates an existing movie review in the database.
DELETE: api/moviereviews/{id} - Deletes a movie review from the database based on the provided ID.

User:

GET: api/users - Retrieves all users from the database.
GET: api/users/{id} - Retrieves a specific user by ID from the database.
POST: api/users - Adds a new user to the database.
PUT: api/users/{id} - Updates an existing user in the database.
DELETE: api/users/{id} - Deletes a user from the database based on the provided ID.

Authentication:

POST: api/auth/register - Registers a new user in the database.
POST: api/auth/login - Authenticates a user and generates a token to be used for authentication in future requests.

Conclusion:

Overall, I learned the basics of creating and configuring a web API using ASP.NET Web API. I discovered multiple ways to scaffold a database and found that scaffolding controllers was the most efficient method. Although I initially created a couple of controllers from scratch, I later learned that scaffolding them was a simpler process. In the future, I would like to create some controllers manually and compare the success rates with those created through scaffolding. Additionally, I need to remember to bypass the SSL when attempting to scaffold. Going forward, I plan to retrieve the full request body to utilize the API to its maximum potential. Eventually, I hope to create a simple front-end layout that enables users to search for movies and view their reviews. Overall, I enjoyed working on this project and am thrilled to continue working on this API.