Movies Task

Clayton Mallia

Project overview

The project is split into 2 parts:

1. Backend - which is written with c# using .net 8
2. Frontend - written is written with React + typescript

Backend

Overview

API

The implementation of the Backend is of a Rest API in the **.net 8**. The project is making use of the controllers and not the minimal api. The architecture of the project is using N-tier but it can easily be changed to Clean or Onion Architecture. I am using the .net built in OutputCache to cache the genres endpoint.

Lambda

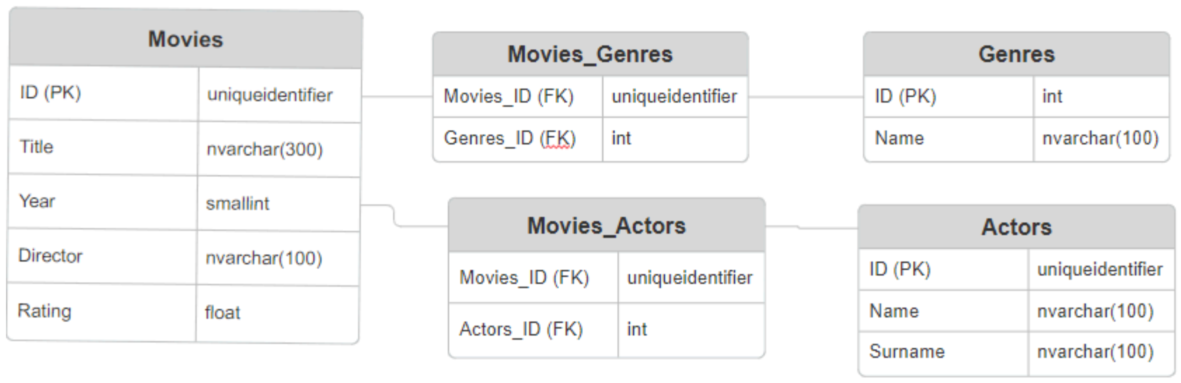
A Lambda project was also created but is not referenced by the frontend. The reason why this was created, in case I wanted to host it on the AWS infrastructure as mentioned in the section here.

Database

The project is not using any database as the data is being loaded from a list, but if it had to be implemented we would go for the option using either:

- Entity Framework (EF) Core - Code first
- Dapper - Code first

With what was implemented so far, it would be more favorable to implement for EF core. Below would be the ER Diagram to the database schema



Movies	
ID (PK)	uniqueidentifier
Title	nvarchar(300)
Year	smallint
Director	nvarchar(100)
Rating	float

Movies_Genres	
Movies_ID (FK)	uniqueidentifier
Genres_ID (FK)	int

Genres	
ID (PK)	int
Name	nvarchar(100)

Movies_Actors	
Movies_ID (FK)	uniqueidentifier
Actors_ID (FK)	int

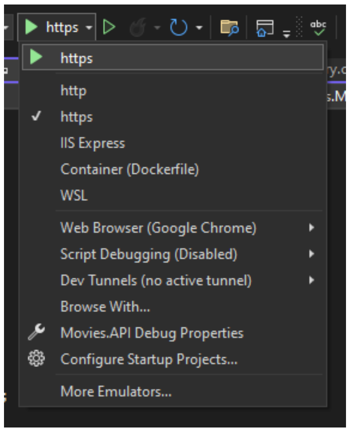
Actors	
ID (PK)	uniqueidentifier
Name	nvarchar(100)
Surname	nvarchar(100)

Running the backend Locally

Visual studio

API

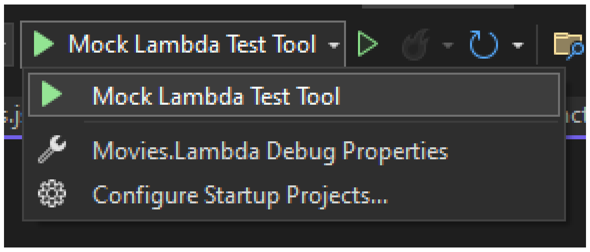
If you are running the project through visual studio you can use the https option.

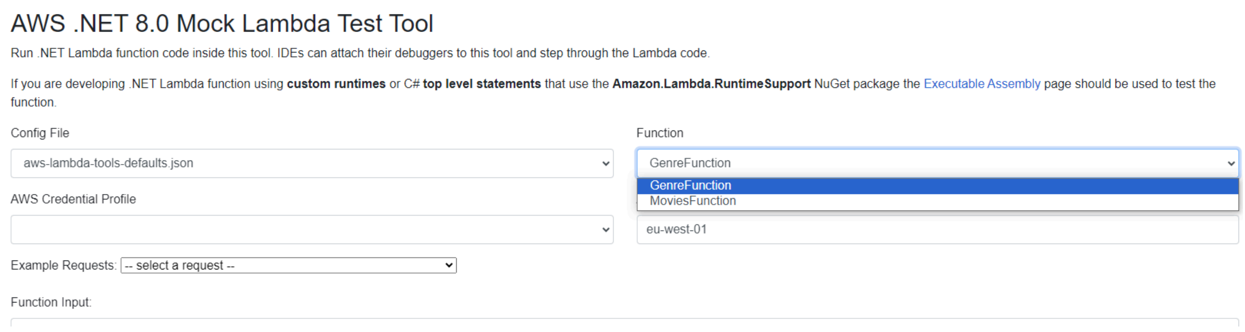

https
https
http
✓ https
IIS Express
Container (Dockerfile)
WSL
Web Browser (Google Chrome)
Script Debugging (Disabled)
Dev Tunnels (no active tunnel)
Browse With...
Movies.API Debug Properties
Configure Startup Projects...
More Emulators...

The url for the api would be https://localhost:7105/api/

Lambda

The project can run only through visual studio for local development which uses the **'Mock Lambda Test Tool'**, which will open AWS .NET 8.0 Mock Lambda Test Tool on web browser using http://localhost:5050/





AWS .NET 8.0 Mock Lambda Test Tool

Run .NET Lambda function code inside this tool. IDEs can attach their debuggers to this tool and step through the Lambda code.

If you are developing .NET Lambda function using **custom runtimes** or C# **top level statements** that use the **Amazon.Lambda.RuntimeSupport** NuGet package the Executable Assembly page should be used to test the function.

Config File: aws-lambda-tools-defaults.json

Function: GenreFunction

GenreFunction
MoviesFunction

AWS Credential Profile:

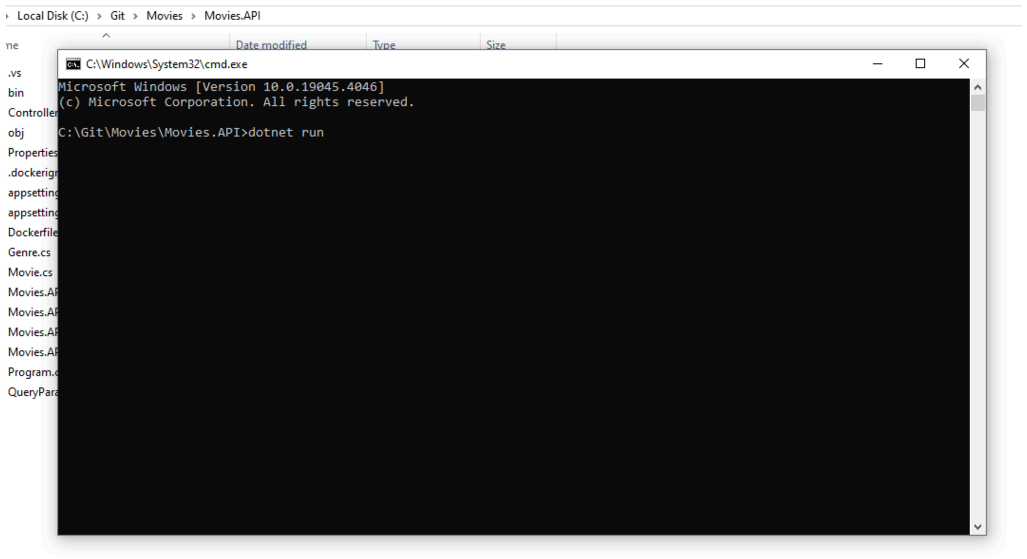
eu-west-01

Example Requests: -- select a request --

Function Input:

CMD

To run the project through the cmd, access the directory where the Movies.API is located and type **dotnet run**



```
Microsoft Windows [Version 10.0.19045.4046]
(c) Microsoft Corporation. All rights reserved.

C:\Git\Movies\Movies.API>dotnet run
```

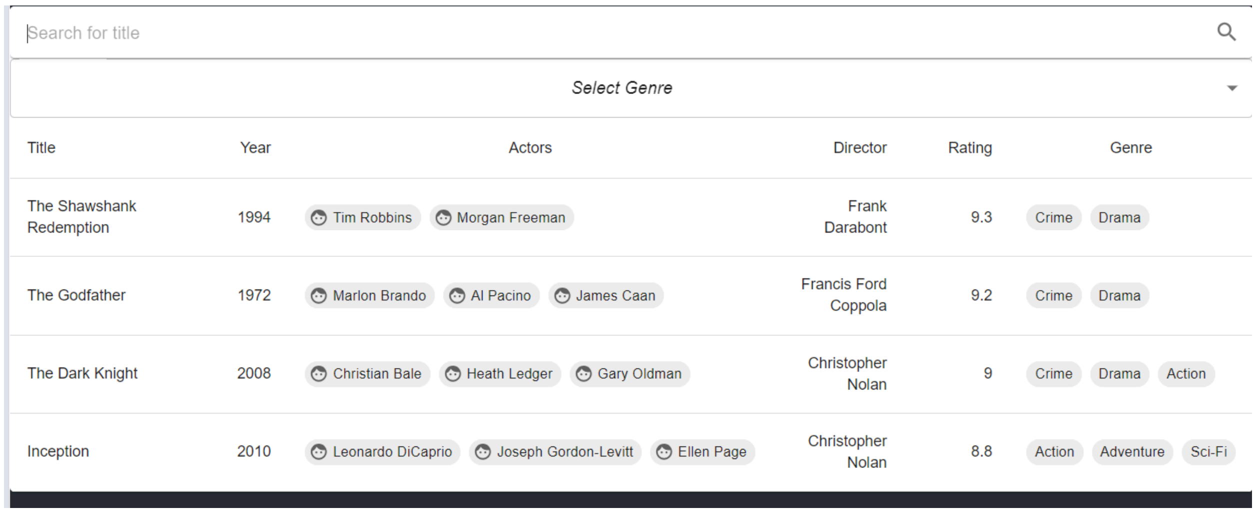
The url for the api would be http://localhost:5270

NB: Please make sure that whatever option you want to choose to run the rest api, update the baseUrl of the frontend which updated the Config.ts file (src > Utility > Config.ts)

Frontend

Overview

The implementation of the frontend is using react + typescript. The use of the Material UI components are using the in home page to display the data and allow the user to filter the results. The npm version used was 6.14.12 and the node version was v14.16.1



Title	Year	Actors	Director	Rating	Genre
The Shawshank Redemption	1994	Tim Robbins, Morgan Freeman	Frank Darabont	9.3	Crime, Drama
The Godfather	1972	Marlon Brando, Al Pacino, James Caan	Francis Ford Coppola	9.2	Crime, Drama
The Dark Knight	2008	Christian Bale, Heath Ledger, Gary Oldman	Christopher Nolan	9	Crime, Drama, Action
Inception	2010	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page	Christopher Nolan	8.8	Action, Adventure, Sci-Fi

Note: There are some minor UI fixes which I would have worked on in case I had more time. For future enhancements I would:

- use a loading component to show the user when data is being loaded through the api
- Introduce pagination (the rest api already supports this)

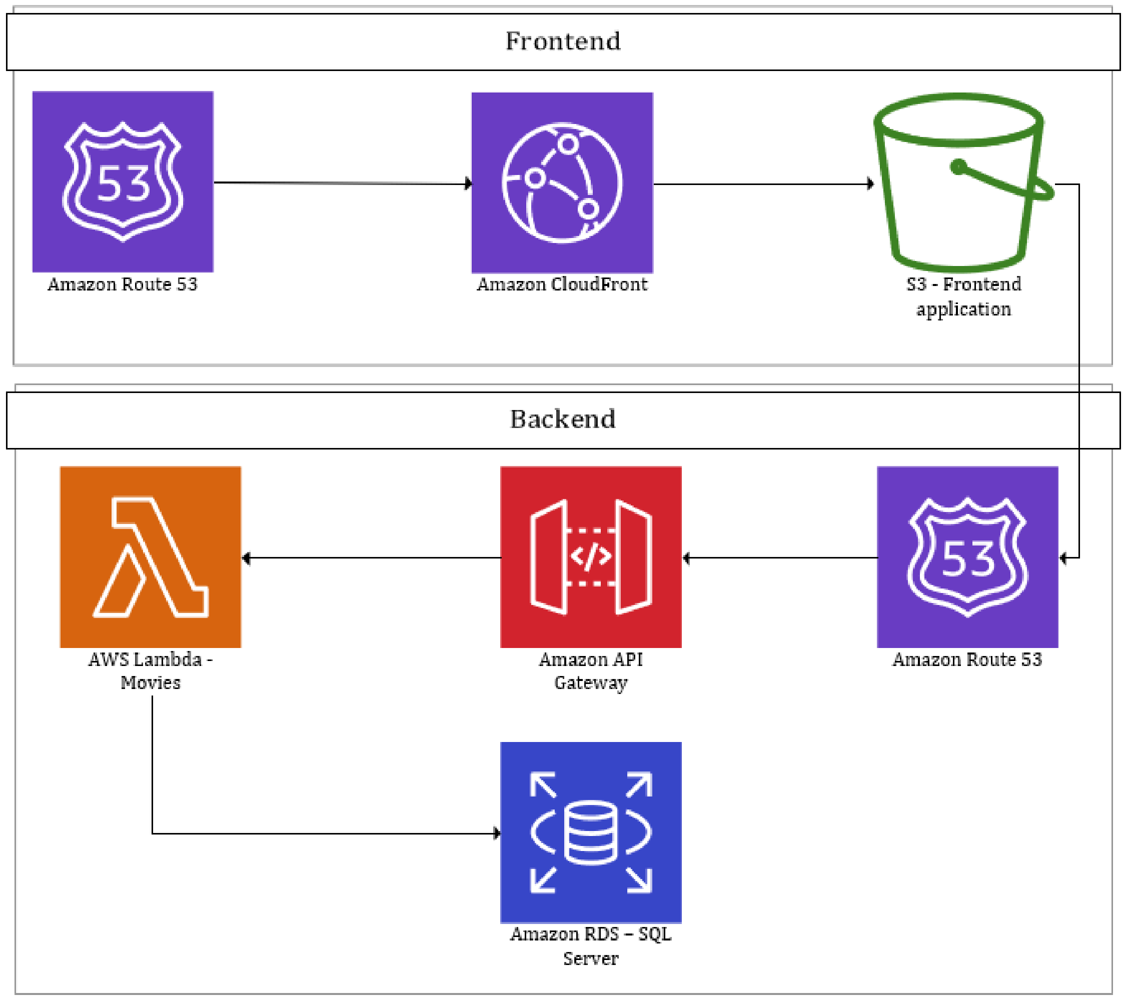
Running the Application

Before running the application make sure that:

1. the npm packages are installed by running **npm install**
2. The config.ts file is pointing to the correct api domain and port.

Then you can run the project by typing **npm start** in the Terminal

AWS Infrastructure



This solution is not deployed on AWS, but if I had to host it on AWS I would have setup the following AWS resources:

1. Route 53
    - To route traffic to Cloudfront CDN to access the site with a domain
    - To route traffic to the API Gateway to access the backend with a domain. This will eliminate the risk of having the frontend accessing directly the API gateway. In case the API gateway url changes, I wouldn't need to redeploy the frontend with a new api gateway url
2. CloudFront
    - Where I setup the CDN for the bucket so the site will be served from multiple regions
3. S3 Bucket
    - I would publish the react app using the CI/CD to the S3 bucket.
    - I would set the S3 bucket as Static Website hosting
    - Enable public access in the bucket policy.

4. API Gateway
    - This will be the communication gateway between the Frontend and the lambda function. Here I have control of what endpoints I want to expose to the client which will give me the facility to not expose all endpoints if needed
    - I can also turn on the API Gateway Cache here to cache the responses accordingly
5. AWS Lambda Function
    - The lambda function will serve the request from the api gateway and will get the data from the database and return the necessary response.
    - I would also use Cloudwatch to log different levels (such as info and errors).
6. Amazon RDS
    - In the RDS I would set a relational database (such as SQL or MYSQL). The database will store the data of Movies, Genres, Actors, etc...