**Webjet Movies Test Project**

The goal of the system is to fetching the information about movies from two different movie providers and show the information including price of the movies in a web application.

The providers have introduced REST APIs as the solution to integrate with.

**Considerations/Requirements**

* **Resiliency**  
  Resiliency is a key requirement. The web application must always display the most recently available movie’s data, even when the providers' APIs are temporarily unavailable (e.g., due to connection issues or maintenance). The system must cache and serve the latest successful data to ensure a seamless user experience.
* **Extensibility**  
  While the system currently integrates with two providers (Cinema World and Film World), it must be designed to easily accommodate additional providers in the future. Adding a new provider should be achievable with minimal effort and within a short timeframe.
* **Scalability**  
  The system must be able to scale horizontally to handle increased load, whether due to a high volume of requests or an increased number of providers. Services should be configured to automatically scale based on predefined metrics such as CPU or memory usage.
* **Security**  
  Although the web application does not require user authentication or authorization (i.e., it has no login functionality), backend services and APIs must still be secured. Measures must be in place to protect against common threats, such as denial-of-service (DoS) attacks, unauthorized access to APIs, and data tampering.
* **Easy Deployment**

The deployment process should be simple, reliable, and compatible with CI/CD pipelines. The system must support automated deployments, versioning, and easy rollbacks to ensure smooth releases and minimal downtime.

* **Testability**

The system must support automated testing, including unit tests and integration tests, to ensure reliability and maintain high code quality. A strong testing strategy will help catch issues early and support ongoing development and refactoring.

**System Architecture**

A circular logo with a circular arrow around it

AI-generated content may be incorrect.The system consists of 4 services

1. Cinema World Data Sync Service

This is an scheduled service that works in the background. At the moment it fetches the data every 5 minutes, using REST API provided by Cinema World and save the Movie’s information in a Redis Cache with a proper key.

**TODO**

Currently, the job scheduled to be run every 5 minutes, and it is hardcoded but ideally it should be configurable.

Also, in the real-world scenarios, I would used Azure Function or AKS with KEDA scaling.



1. Film World Data Sync Service

Another service which is a replication of Cinema World Service, it is set to fetch the movies from Film World Service using the REST Api provided by Film World. Similarly, it fetches the data and save them in the Redis using different key.



1. Web Application

The web application is an Angular App that call the backend REST API to fetch the movies and show the records in the browser.

**TODO**

A HTTP interceptor needs to be set to automatically add the API key to the header for all the http requests.



1. Backend API

This is the service that host an API. The Angular app integrates with the backend through this API.

**TODO**

The API is not secured yet, due to the lake of time. But in the real-world scenario, I would secure the API through Azure Entra with or at the least with API Kyes. Or Ideally, I would use Azure API M as a gateway to check the rate limiting and Authentications.

I have added a middleware just to show a simple solution to add the minimum security by checking the API keys sent in the header by the client. I have commented the line that adds the middleware since the front is not yet ready to send the API key.

How the components are integrated



A circular logo with a circular arrow around it

AI-generated content may be incorrect.

A circular logo with a circular arrow around it

AI-generated content may be incorrect.

A circular logo with a circular arrow around it

AI-generated content may be incorrect.

**Automation Test**

I have added tests for the RedisLibrary and CinemaWorldDataSync.

The tests I have created for CinemaWorldDataSync are integration tests. These tests runs the full process and Assert the outcome. Also, an integration test has been added that test the scenarios when the movie provider apis are down.

**TODO**

More Test need to be created. I have just added a few to show the skills of create tests and using mocking tools.

**How to run the application**

All the services are containerized and set in a docker compose file. So, for running the application you need to have docker installed on your machine.

1. First of all, you need to get the source code from this repository.

Open a git terminal, navigate to a folder you wish to put the source code and enter this command.

**Git clone https://github.com/faridkazemi/webjet-test.git**

1. Adding environment variables and secrets before running the application is necessary.

Go to the Webjet-test folder, create a file named. **.env --------------🡪 Webjet-test\.env**

And copy these values into the file and save it.

**RedisConfigOption\_\_ConnectionString=redis:6379,abortConnect=false,connectTimeout=10000,syncTimeout=10000**

**CinemaWorldConfigurationOption\_\_CinemaMovieHttpClientName=CinemaMovie**

**CinemaWorldConfigurationOption\_\_BaseUrl=https://webjetapitest.azurewebsites.net/api/cinemaworld/**

**FilmWorldConfigurationOption\_\_FilmMovieHttpClientName=FilmMovie**

**FilmWorldConfigurationOption\_\_BaseUrl=https://webjetapitest.azurewebsites.net/api/filmworld/**

1. Navigate to folder Webjet-Movies-Backend and create a folder called **secrets -------🡪 Webjet-test\Webjet-Movies-Backend**

Create the following files with the content.

**Webjet-test\Webjet-Movies-Backend\secrets\CinemaWorldConfigurationOption\_\_AccessToken.txt**

**Webjet-test\Webjet-Movies-Backend \secrets\FilmWorldConfigurationOption\_\_AccessToken.txt**

Due to security concerns, I don’t add the content of the files here. You just need to put the token you provided for this test, into these files. Since, the token is the same for the both APIs, the content of the files will be the same.

1. Now everything should be ready to run the app.

Open a terminal, navigate to the Webjet-test folder and run the command. **Docker compose up –build**

1. Open your browser and enter this url. **http://localhost:4200**
2. You should be able to see the list of movies from the both providers. Also, you can sort by price.

If your machine contains any proxy, Docker might not be able to access nuget packages during restoring packages. If you faced any similar issue you might need to fix the proxy issue firs.