# Coding test documentation

# Development environment

**Code solution is available at following 2 git repositories.**

<https://github.com/KavinduTeknika/FrontEnd.Movie>

<https://github.com/KavinduTeknika/BackEnd.Movie>

Front end component:

* Main technologies/programming languages used - > ReactJs, Redux, JavaScript
* FrontEnd.Movie -> Package.json contains the information about all the npm libraries and the versions of them

Backend component:

* Main technologies/programming languages used -> .Net Core 2.1, C#, Microsoft Azure blobs

# 2.0 Assumptions

* Third party api -> /api/cinamaworld/movies and /api/filmworld/movies are flaky and not reliable for retrieving the movie details real time by the api service is created in this solution.
* Movie prices are unlikely to update very frequently within a day and this solution assumes that the movie details are being updated once a day.

# 3.0 Solution

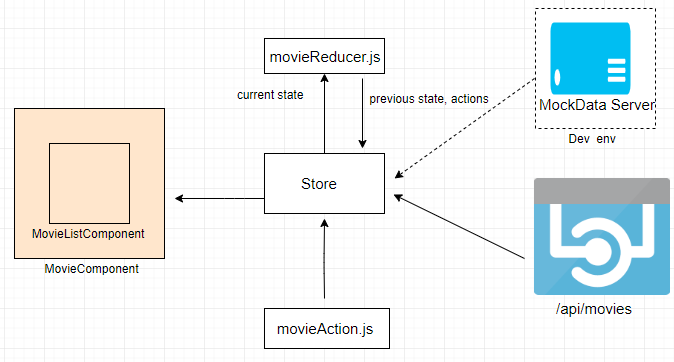


Fig: Design for the UI Application

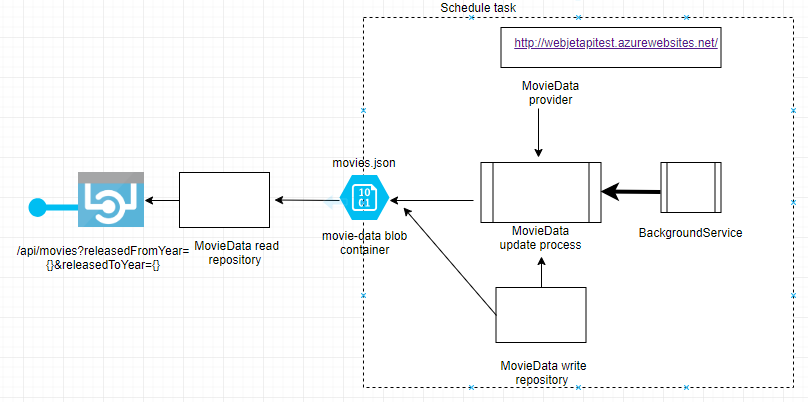


Fig: Design for the API service

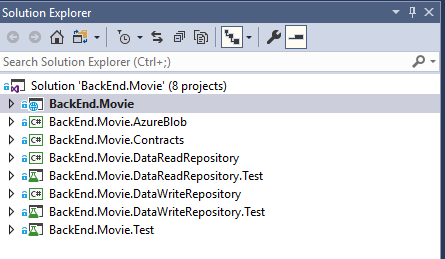


Fig: Package structure for BackEnd.Movie API service.

* New API end point: /api/movies?ReleasedFromYear={fromYear}&ReleasedToYear={toYear}

This REST API will accept the optional time range parameters: *ReleasedFromYear* and *ReleasedToYear* which specify the movie released year from/to and respond with the details of the movies are released within the specify time range.

* The new API service maintains a data storage (Azure blob) for the purpose of caching movie data as the movie data provider API (third party API) is not reliable.
* The cache/data storage is updated periodically by a schedule task.

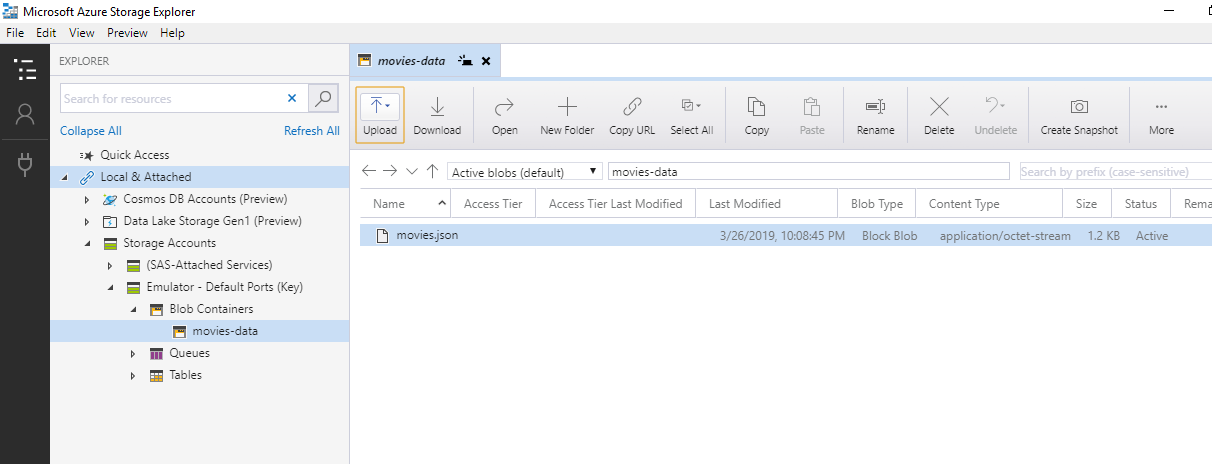


Fig: movies azure blob store movie data are fetched from third party api (WebJet api)

# 4.0 Future works

* Make the time interval for periodic cache/data updating configurable.
* Deploy BackEnd.Movie and FrontEnd.Movie to different app services/containers and utilize BackEnd.Movie service API within FrontEnd.Movie UI page.
* Implement the feature to retrieve movies are within the specified price range as well.
* API response should be sort by price in ascending order.

# 5.0 Testing

* The API service runs in local environment.
* Azure storage emulator is used to access the local azure storage.
* The UI application is run on <http://localhost/3000> and the mock data server which is a json server is run on <http://localhost/3001>

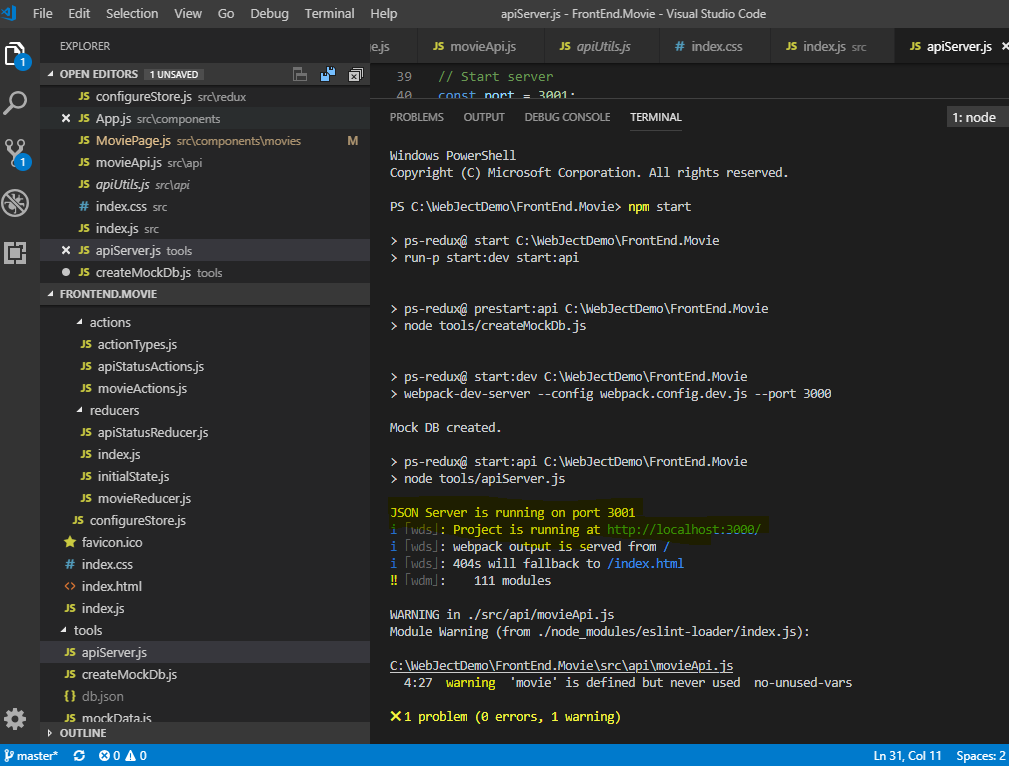
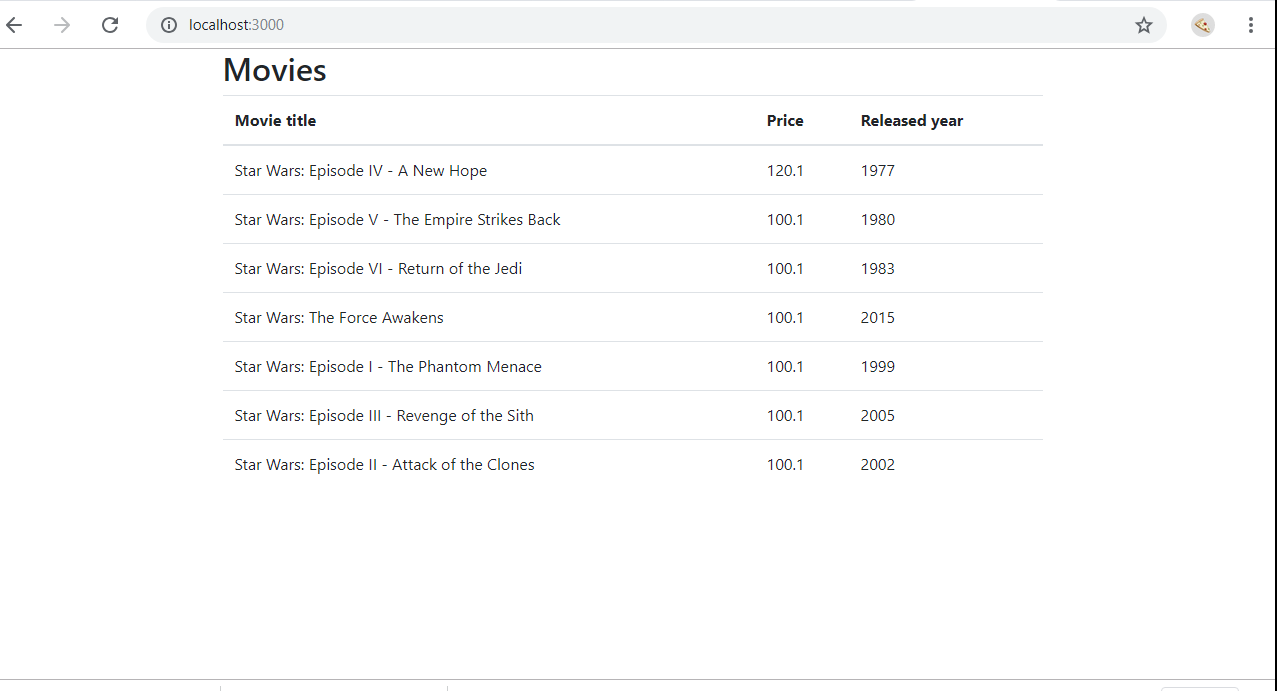
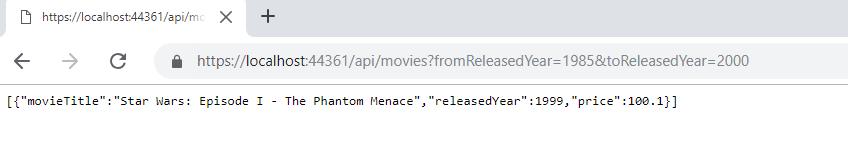


Fig: Execute UI app and the mock data server in the local environment.

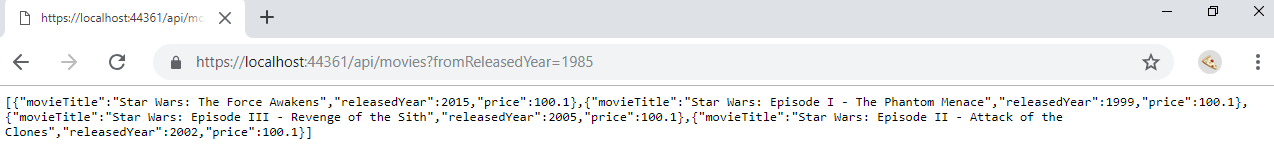
* Test 1: Display all the movies in the UI



* Test2: New API response to the request for the movies have been released within 1985 to 2000



* Test3: New API response to the request for the movies have been released within 1985 to this year 2019



* Test4: New API response to the request for the movies have been released within to this year 2019

