# Movie Service

## Overview

Service is built on webapi 2. This program uses in memory cache using concurrentlist. A better alternative would be using Redis. Application has been structured in a way that that can be switched for the existing cache by changing bindings on the service. Redis’s requirement for reverse index key and paucity of time on my side, made this alternative, not a feasible option for this project for me.

## Design decisions

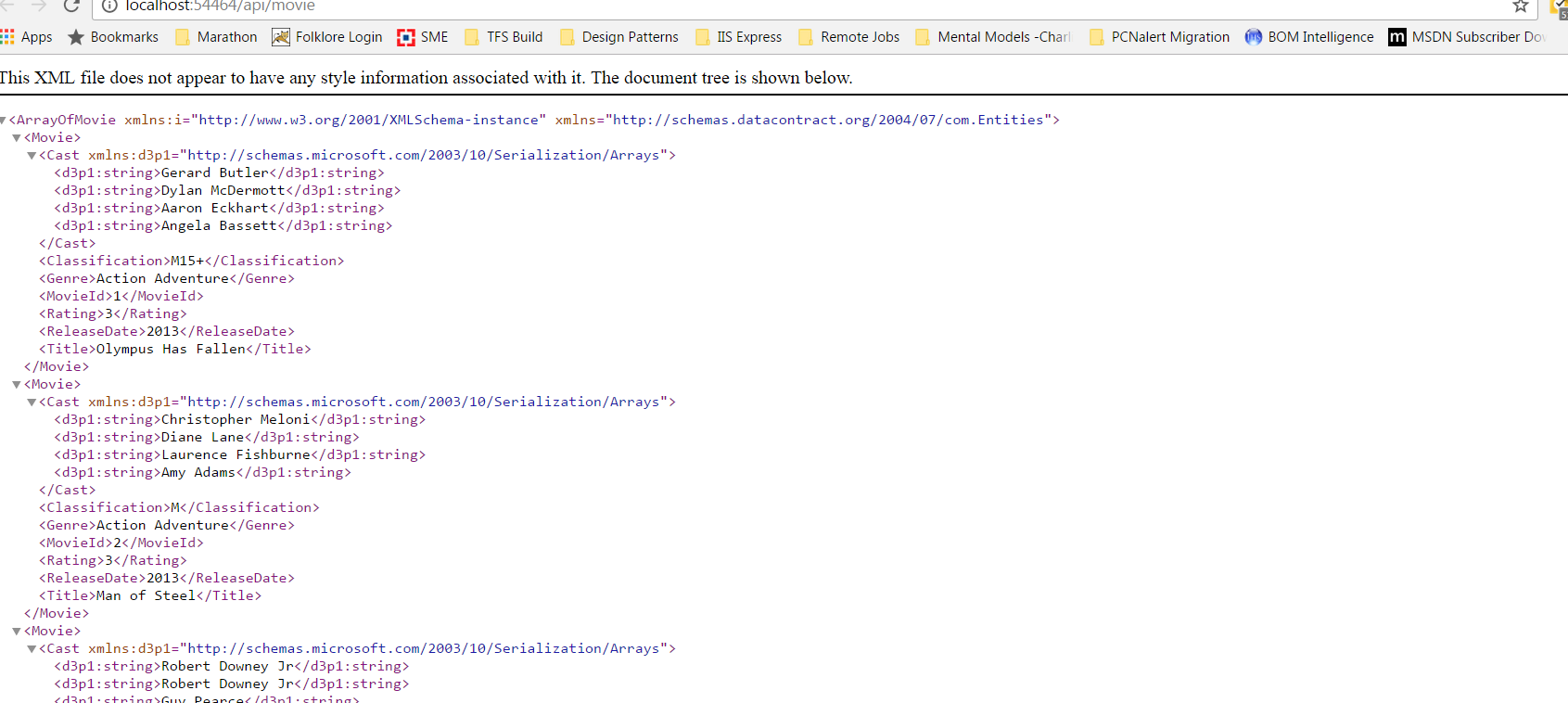
1. Store performs cache manipulation and cache as such is agnostic of other dependencies such as the MovieDataSource. Another cache (such as Redis) will be easy to incorporate.
2. I have used attribute routing instead of conventional webapi routing as it is my personal choice.
3. I have used dynamic linq (over linq to objects) as I find it is less time consuming during development.
4. I believe a service api has to do only the bare minimum of what is expected and should do it well. Therefore, I throw exceptions to the client to handle and tailor accordingly. In some cases, where .net might not send out appropriate messages to display, I have created threadbare custom exception classes.
5. All data is fetched from cache. Update and Create is made both in the cache and the moviedatasource. Syncing during a fresh from both data sources is simpler this way.
6. Admin controller exposes a url to invalidate and refresh the cache. A service can use this URI to perform the once in a day update of the movie library. [http://localhost:<port>/admin/refreshcache](http://localhost:%3cport%3e/admin/refreshcache).
7. Ninject bindings are set up in webapi service project. Ideally it should have been outside.

## List of service api

|  |  |  |
| --- | --- | --- |
| *Create* | Json string sent via Post request. Everything except Id goes in the string. Returns nothing except statuscode or exception. | api/Movie  formbody: (json)Movie  httpMethod: Post |
| *Update* | Json string with 2 arguments. Id of the movie to be updated and movie as json in the body of the request. Returns nothing except statuscode or exception. | api/movie/{id}  formbody: (json)Movie  httpMethod: Put |
| *Fetch all movies* | No arguments. Returns all movies or exception and status code. | api/Movie |
| *Fetch a single movie* | Returns a single movie if it exists, a status code or an exception. | api/Movie/{id:int?} |
| *Search* | Returns records | api/Movie/{Left}/{Oper}/{Right}  gt 🡺 >  lt 🡪 < |
| Sort | Returns records sorted in an order as per query | api/Movie/{query} |
| Exposed on Admin controller  Invalidate and Replenish cache | Post operation that invalidates cache and replenishes it. | Api/admin/refreshcache  Httpmethod: post |

Screenshots

Get all movies



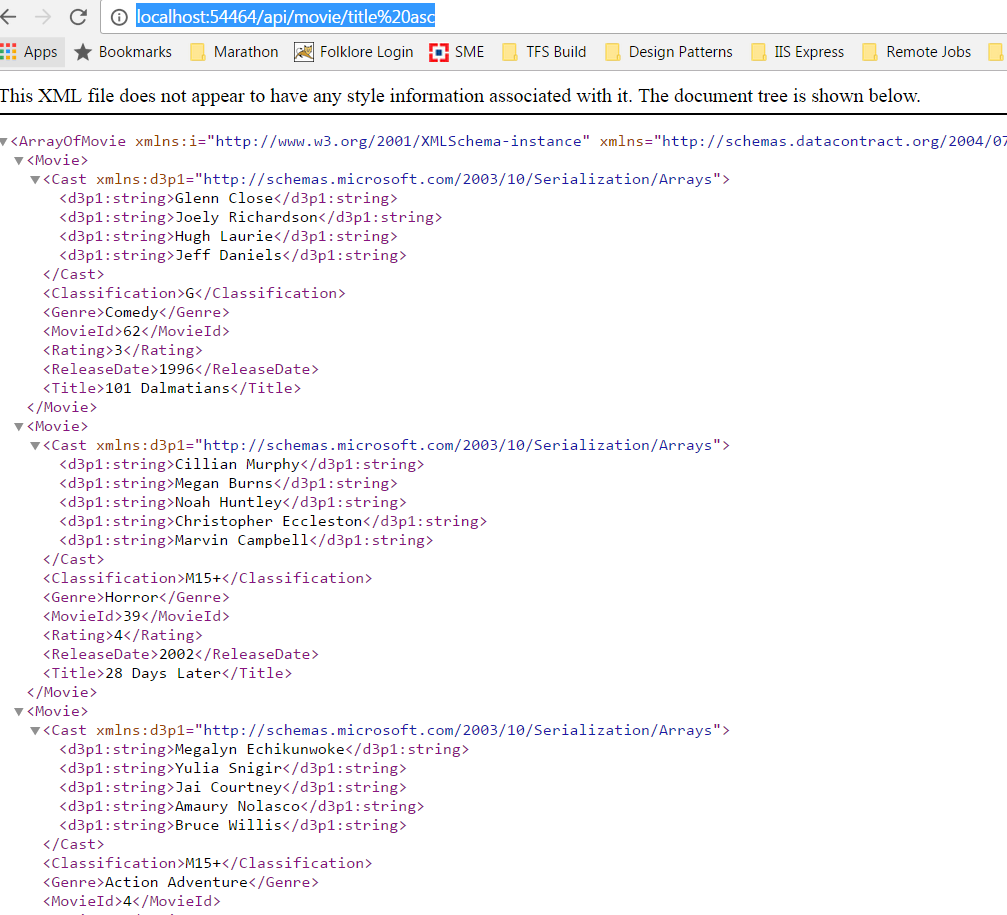
url: [http://localhost:<port>/api/movie](http://localhost:%3cport%3e/api/movie)

Get one movie:



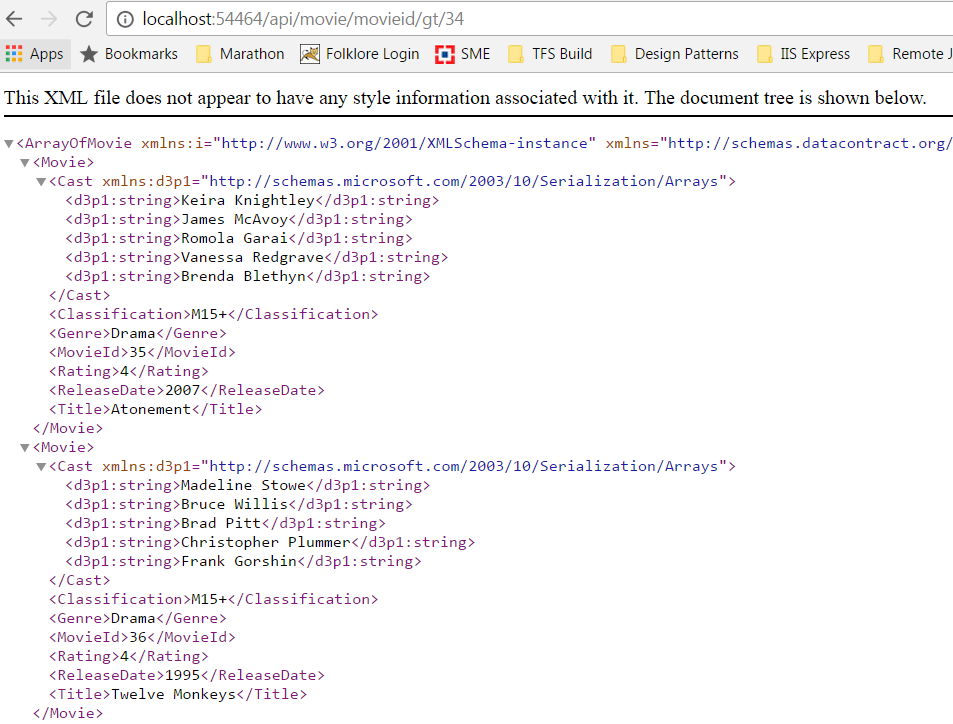
url: [http://localhost:<port>/api/movie/<movieid](http://localhost:%3cport%3e/api/movie/%3cmovieid)>

Sort:



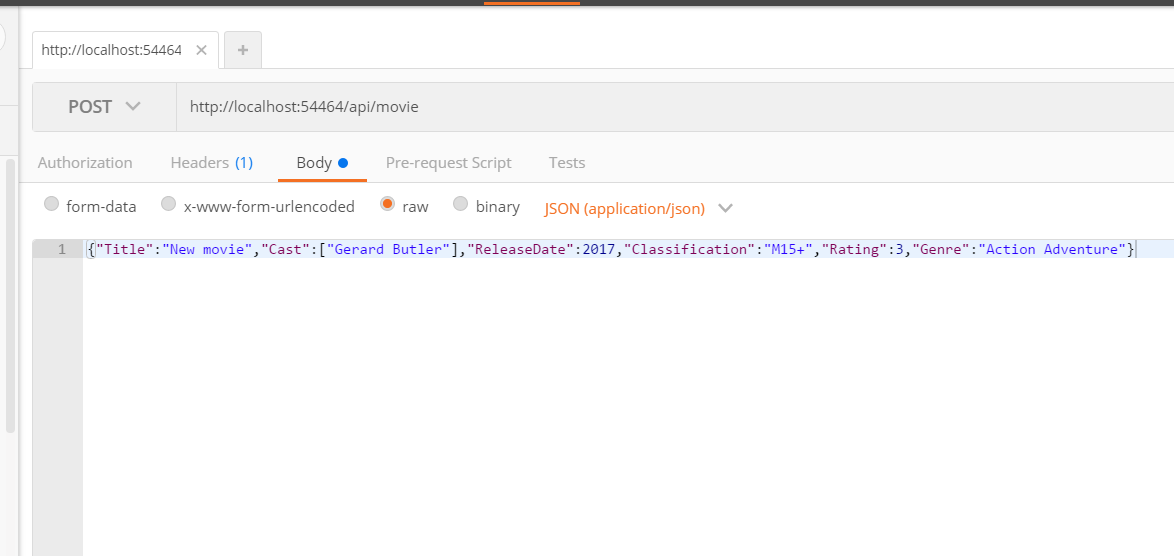
url: [http://localhost:<port>/api/movie/title%20asc](http://localhost:%3cport%3e/api/movie/title%20asc)

Search:



url: [http://localhost:54464/api/movie/**movieid/gt/34**](http://localhost:54464/api/movie/movieid/gt/34)

Create

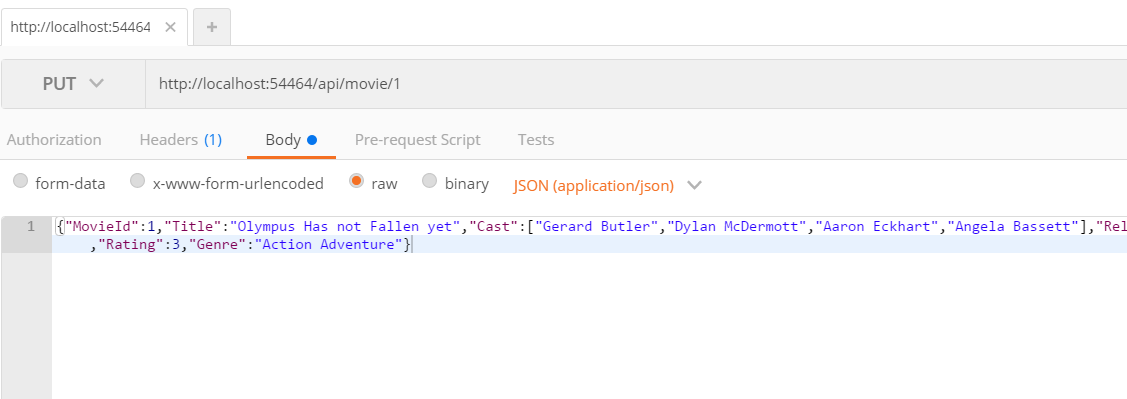
Screenshot from postman. 

After the post, browser can be used to see the newly added movie, as shown below.



Update:

Screenshot from postman.

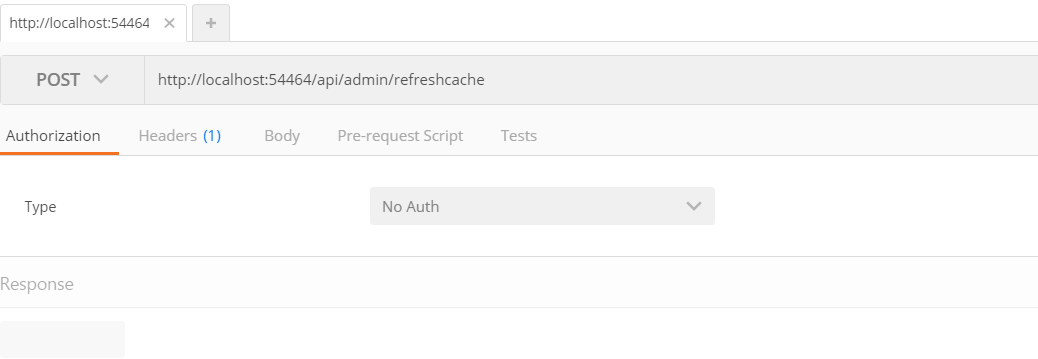


url: <http://localhost:54464/api/movie/1>

After update is done, browser can be used to see the updated movie.



Admin – refresh cache:



url: http://localhost:<port>/refreshcache