Contents

[Project Output 1](#_Toc472870773)

[Tools & LIbraries 2](#_Toc472870774)

[Project Structure 2](#_Toc472870775)

[Assumptions 3](#_Toc472870776)

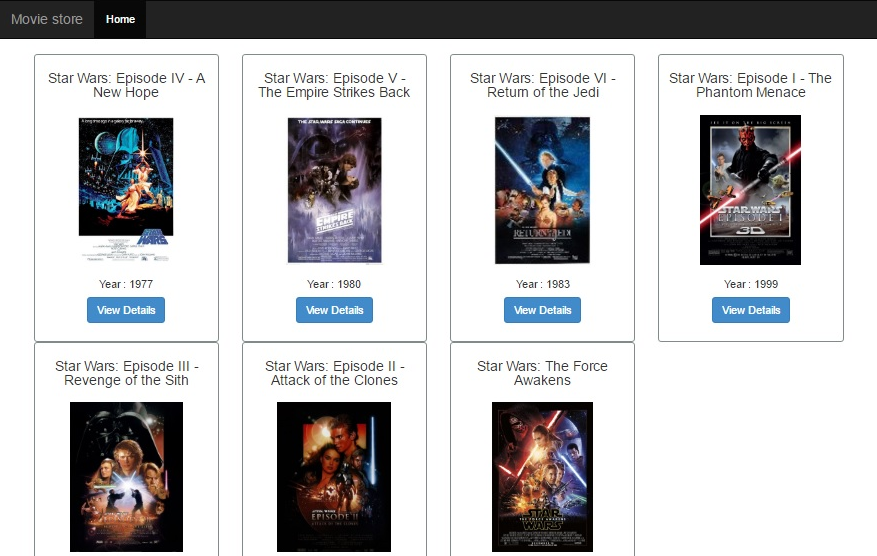
[Main Logic for comparison and loading of data 4](#_Toc472870777)

[**To Get the list of all movies in both db** 4](#_Toc472870778)

[**To Get details and cheaper price of a movie** 5](#_Toc472870779)

# Project Output

This is the main page that gets loaded with the list of movies from the source and rival movie database with a summary of film name , poster and year it was released.



To see detail of the movie with the indication of which movie db Cinemaworld or film world gives a cheaper price, you just have to select the view details button . The movie detail page appears which will give details of the movie. Also it will indicate in green whether Cinemaworld or film world gives a chepaer price with a thumbs up symbol just next to the cheaper one as below along with how much price difference each have.



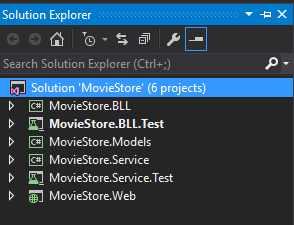
# Tools & LIbraries

These major libraries and tools are used in the sample appliction:

* AngularJS 1.2.6
* ASP.Net MVC 5
* Autofac IOC Container
* Bootstrap 3.1
* .NET Framwork 4.5
* Visual Studio 2015 with IIS Express
* Microsoft.Practices.EnterpriseLibrary.TransientFaultHandling
* Ms Test
* Jasmine & Karma

# Project Structure

The Project solution has been designed with the following Structure



In summary, the Web Layer talks to the BLL layer which calls the Service layer using the underlying models defined in the Models project.

It has been designed so that it is loosely coupled and each layer can be unit tested.

Solid principles have been used using IOC container Autofac.

|  |  |
| --- | --- |
| **Projects** | **Description** |
| MovieStore.Web | The Web project constitutes the web client part of the project. A front end interface has been designed using html, bootstrap and csss ,using angular js to making http calls to mvc controllers to fetch data. It includes a test folder for testing angular scripts controllers and services using jasmine and karma. |
| MovieStore.Models | All data models are defined in this project |
| MovieStore.BLL | This is where the Business logic layer split between the interfaces and implementations of the core logic of the program and calls the service layer. |
| MovieStore.Service | This where the service calls to the web apis are defined. Helper classes have been used to convert the httpresponsemessages into proper format with validations and error handlings in regards to web api defined as well. All web api urls and token info are stored in web.config for simplicity purpose. |
| MovieStore.BLL.Test | Includes all the unit testing for BLL layer using real and mock data |
| MovieStore.Service.Test | Includes all the unit test for Service layer mainly checking api data retrieval and functions | Includes all the testing for BLL layer using real and mock data |

# Assumptions

* Both filmworld and cinemaworld might not have the same movie in their database at all times but in case they have the same movie, their film titles would be identical along with other fields apart from price and ID ofcourse.
* Since we can’t use the id as join condition we will use the film title when comparing both movies as in real world the ids will never be the similar in both databases
* For demo and simplicity purppose, all fields apart from price have been used as string

# Main Logic for comparison and loading of data

Every movie has the following structure when retrieved from the api.

public class MovieBooking

{

public string ID { get; set; }

public string Title { get; set; }

public string Year { get; set; }

public string Type { get; set; }

public string Poster { get; set; }

public string Rated { get; set; }

public string Released { get; set; }

public string Runtime { get; set; }

public string Genre { get; set; }

public string Director { get; set; }

public string Writer { get; set; }

public string Actors { get; set; }

public string Plot { get; set; }

public string Language { get; set; }

public string Country { get; set; }

public string Metascore { get; set; }

public string Rating { get; set; }

public string Votes { get; set; }

public double Price { get; set; }

public string MovieDB { get; set; }

public string RivalID { get; set; }

public string RivalMovieDB { get; set; }

public double RivalPrice { get; set; }

public bool RivalPriceCheaper { get; set; }

}

## **To Get the list of all movies in both db**

Step 1 : First we get the sourcemovie data.(e.g cinemaworld) populating ID, Title, Year,Type, Poster fields. We Set The MovieDB Field to cinemaworld. This helps in comparison later.

Step 2 : Secondly, we get the Rivalmovie data.(e.g filmworld) populating ID, Title, Year,Type, Poster fields. We Set The MovieDB Field to filmworld. This helps in comparison later.

Step 3 : The we get all Common movies between both lists using a join.

If movie is present in both cinemaworld and filmworld, we set below fields for each

Movie. Since it is assumed all other fields are identical, date for those fields can be picked from any db.

ID: cinemaworld Movie ID

MovieDB: cinemaworld

RivalID : filmworld Movie id

RivalMovieDB : filmworld

Step 4 ; we get all movies present in Cinemaworld but not in FilmWorld.

Step 5 : We get all movies present in Filmworld but not in cinemaworld.

Step 6 we combine all data from step 3 , step 4, step 5 in one consolidated list.

## **To Get details and cheaper price of a movie**

Step 1 : Click on view details in one of the movies from the list

Step2: the movie record is passed as parameter with the below main fields

ID: cinemaworld Movie ID

MovieDB: cinemaworld

RivalID : filmworld Movie id

RivalMovieDB : filmworld

Step3:

Get Source data for that movie by Call api to get movie id by using movieDB from MovieDb field and ID field

ID: cinemaworld Movie ID

MovieDB: cinemaworld

http://webjetapitest.azurewebsites.net/api/ + MovieDB + "/movie/" + ID

when data is retrieved , update movie record Price : cinemaworld price

Step 4:

Get Rival data for that movie by Call api to get movie id by using movieDB from RivalMovieDB field and

RivalID : filmworld Movie id

RivalMovieDB : filmworld

http://webjetapitest.azurewebsites.net/api/ + RivalMovieDB + "/movie/" + RivalID

when data is retrieved , update same movie record RivalPrice: filmworld movie price

RivalPrice : filmworld movie price

Step 5:

An update to the movie record is done by below

if (SourceMovieData.Price >= RivalMovieData.Price)

{

Set RivalPriceCheaper = true;

}

Step 6: The data is pushed to the view where a comparison is done using the rivalcheaper price field and this is shown as cheaper in greeen with a thumbs up pic.