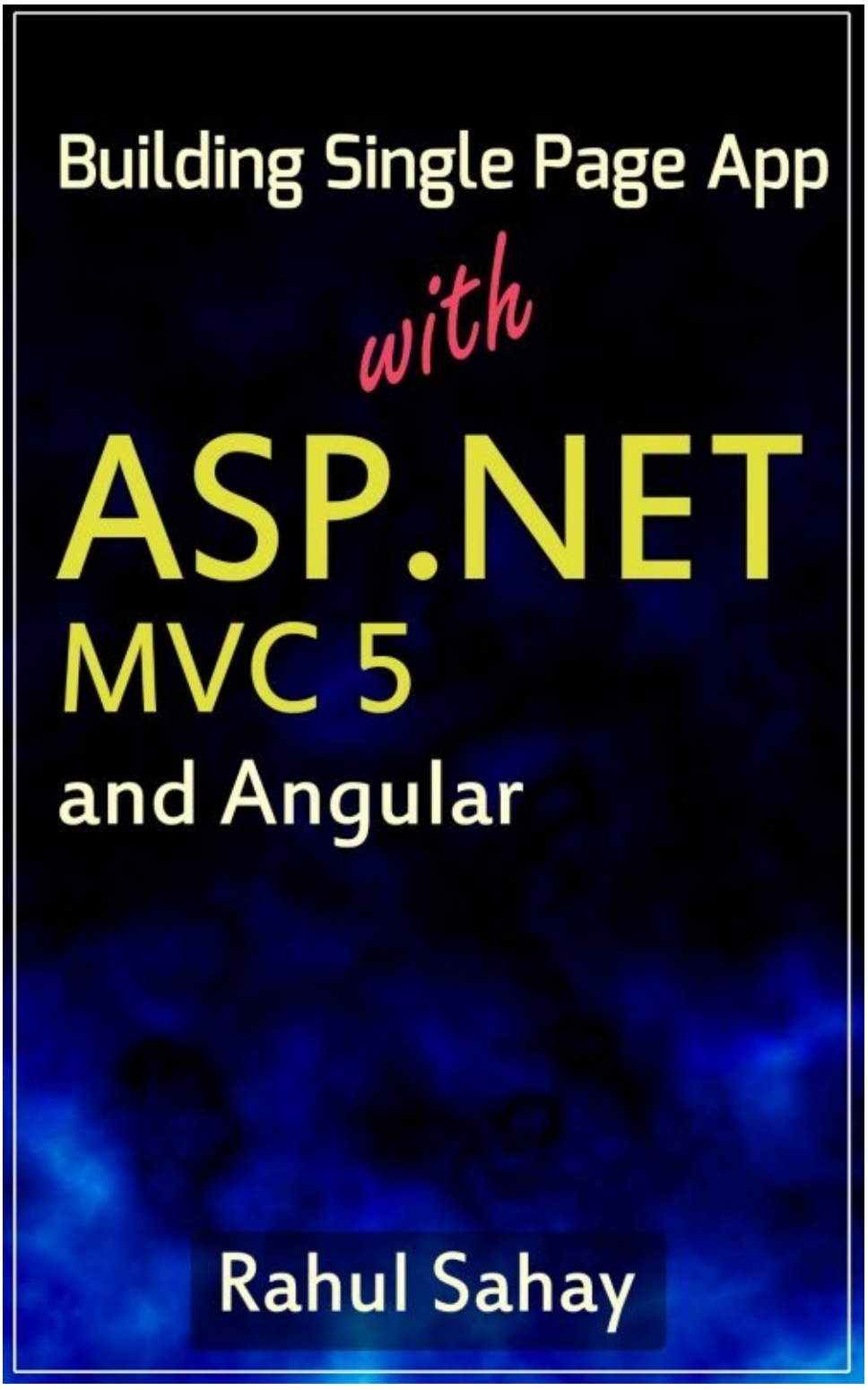
# Building Single Page App

*with*

# ASP.NET MVC 5

## and Angular

Rahul Sahay




# Contents

WHAT DO you find in this CHAPTER?

Angular JS Routing

Adding More Routes

Client side validation

Saving Data using Angular

Creating Angular JS Service

Creating Movie Edit Feature

Creating Reviews Workflow

Summary

**Chapter 6: Unit Testing**

WHAT DO you find in this CHAPTER?

Introduction

Creating Test Project

Installing Chutzpah Test Adapter

Writing 1st JavaScript Test

Writing Angular Test

Using $httpBackend Service

Writing Controller Tests

Code Coverage

Summary

**No of Pages: - 200**

# WHO SHOULD TAKE THIS BOOK

**Building SPA using MVC 5 and Angular** is designed to build the application right from the grass root level. This Book is actually targeted to those people who are comfortable with ASP.NET MVC and Angular as this needs basic knowledge of both the technology. Throughout this book my focus will be on teaching you making a full blown application rather than explaining basics. For basics you can check my other book **Hands on With ASP.NET MVC**. This book talks basics in great details with live demo in Azure. You can refer this book at this URL http://ow.ly/JetAi. I would recommend you to download the app from github URL shown below to help you while building https://github.com/rahulsahay19/MovieReview-Angular-Prod

# Chapter 1: Getting Started

## WHAT DO you find in this CHAPTER?

- Introduction
- What is SPA
- Technologies used to build SPA
- Glimpse of Movie Review App
- Summary

### Introduction:-

Hi my name is Rahul Sahay and I am going to introduce this whole new story of building Single Page Application right from the scratch. Here, in this context I am going to talk about bunch of different client/server side technologies and demonstrate how these small pieces marry together and creates a robust End to End application. So, without wasting time let’s get started.

### What is SPA:-

Single Page App is all about user experience. People will love your app if you give them nice user experience which not only fits nicely in your laptop or desktop rather it goes nicely with multitude of devices like tabs, phones etc without breaking any single functionality. As shown in the below diagram, these are basic requirements for building any SPA.


- Reliability: - People know that it's reliable and it's going to work. This kind of reliability only comes with positive experience.
- Responsiveness: - Responsiveness means it's going to work quickly for them. Quick is the key thing which any user expect to have in the app which he is using.
- Reach: - Reach is often substituted with mobility. Mobility is again one of the key ingredient which every user is looking for. They always want to have the data handy with irrespective of what device they are on.
- Available: - This thing is really important when it comes to the point at working offline. So, delivering a good user experience is must while building SPA.

"***So, in a nutshell a Single Page App is web application which fits in a single page providing a fluid UX by loading all the necessary data in a single load.***"

Now, apart from this there are many other attributes linked to SPA. They are:-

- Maintain History: - When you flip between pages, it maintains your history in the same order how you visited them. Actually, it's not going on different pages rather its loading different information's on the same page. But, it looks to user that it's presenting different pages to them.

- Persisting Information: - Persisting information is also very important aspect of the

SPA. It doesn’t mean that you need to save each and every thing at cache but you can store important things in the cache to improve the performance.

- Mostly loaded on Page Load: - Mostly loaded on the page load means majority of information user required to use gets loaded initially itself to avoid roundtrip back to the server.

- Dependent Elements: - As and when user requires to access different features of the application, app will go and download for the user.

## Technologies used to build SPA:-

Movie Review app is built using tons of different client side and server side technologies. Some of these I have listed below:-

Client Side Technologies:-

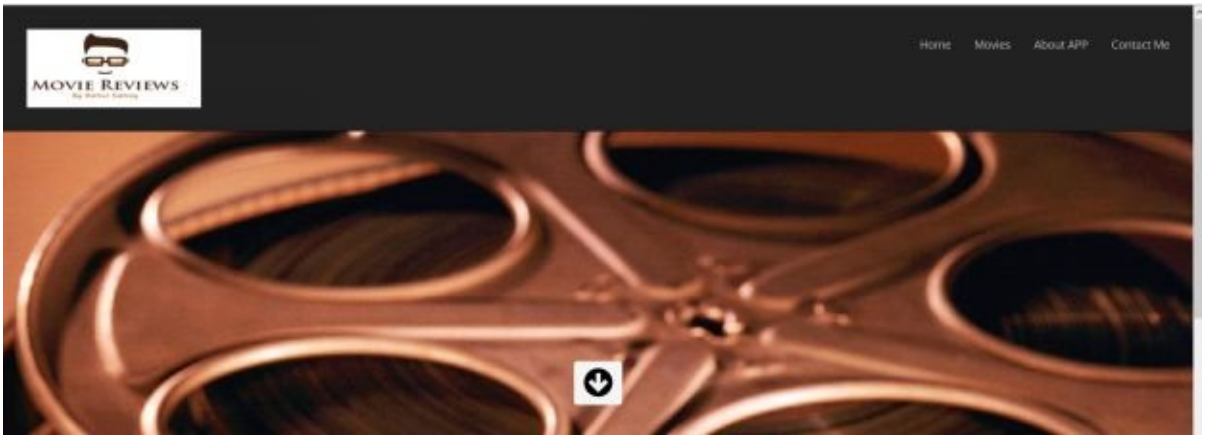
- HTML 5 & CSS
- Modernizer & LESS
- Media Queries
- Responsive Design
- Angular JS
- Toastr JS
- JQuery
- QUnit JS
- JQuery.MockJSON
- Change Tracking
- And many out of the box things

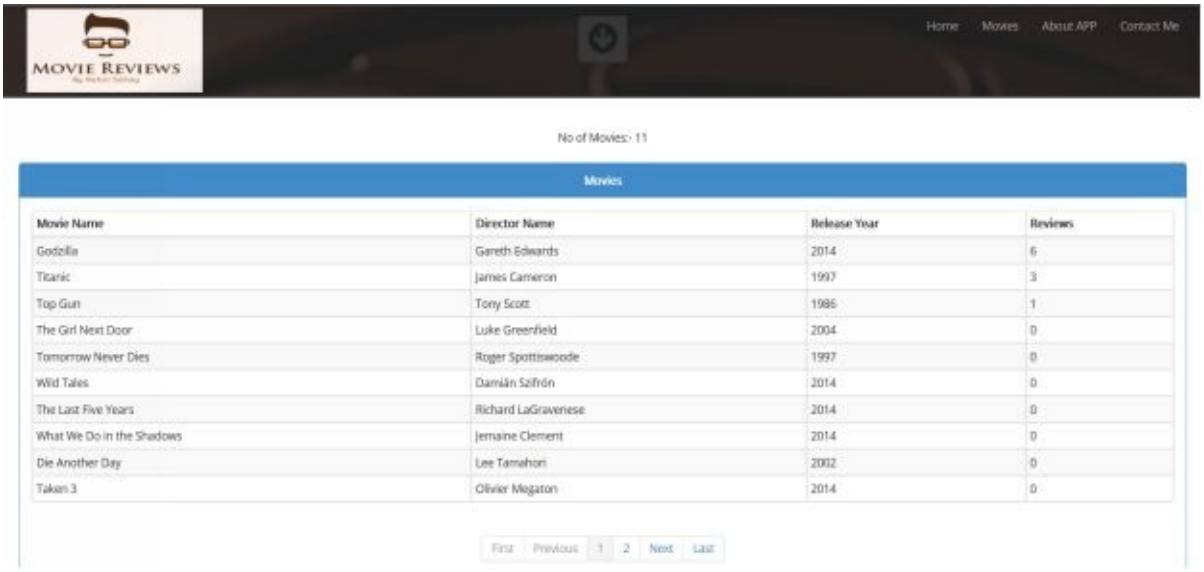
Server Side Technologies:-

- SQL Server
- Entity Framework – Code First Approach
- Repository Pattern
- Unit of Work Pattern
- Web API
- JSON & AJAX
- NuGet
- Ninject IOC
- POCO Models

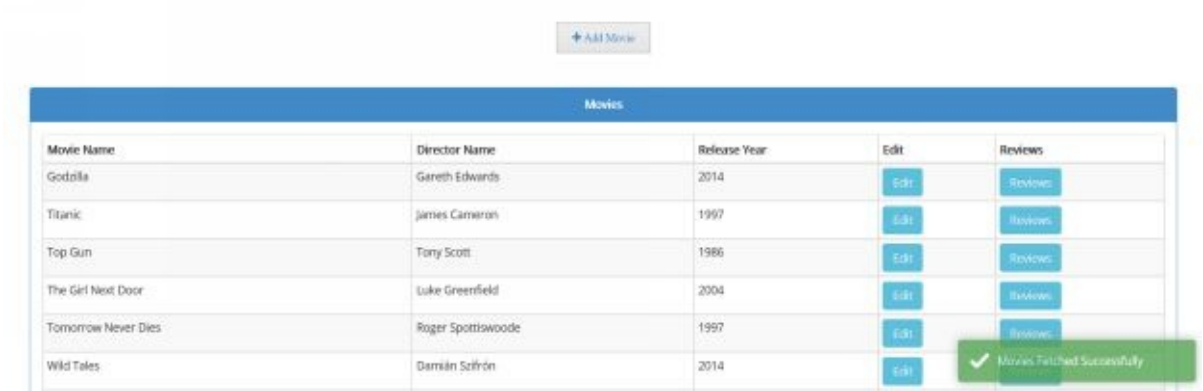
## Glimpse of Movie Review App:-

I think it would be good idea you to show you finished app before directly jump in creating the same. Here is the URL **http://rahulsahay19-001-site1.smarterasp.net/#/** where I have hosted my app. Now, when you click on this, you will land on the below shown page.



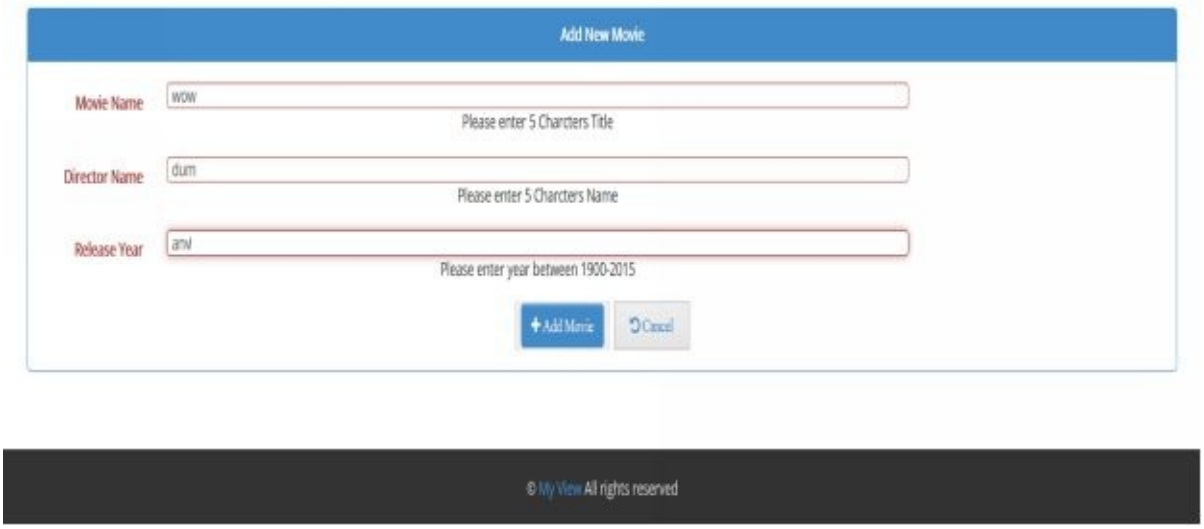




Above shown screen shot is the home page of the app. Now, when you click on the **Movies** link, it will take you to the below shown page.



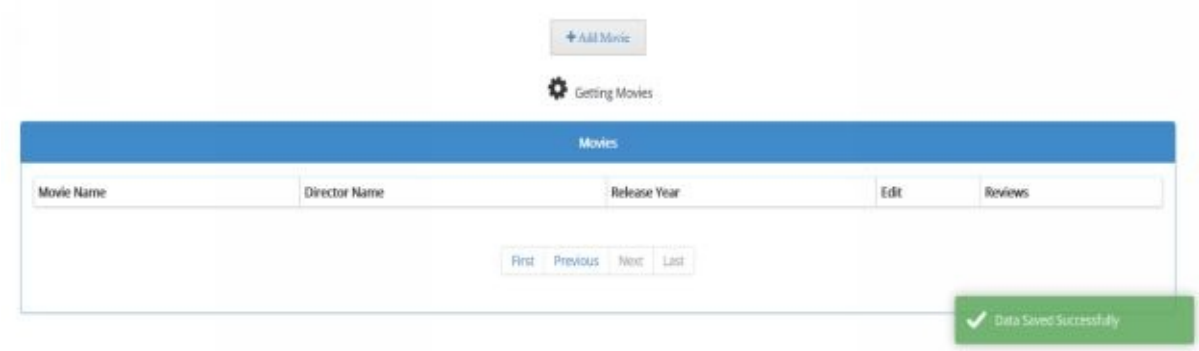
Once, page gets loaded, little toast message at the bottom right of the screen pops up saying **Movies Fetched Successfully**. Now, from this screen you can do all the CRUD Operation. Here the very 1st link is **Add Movie,** which will give user flexibility to go ahead and add any new movie as shown below.

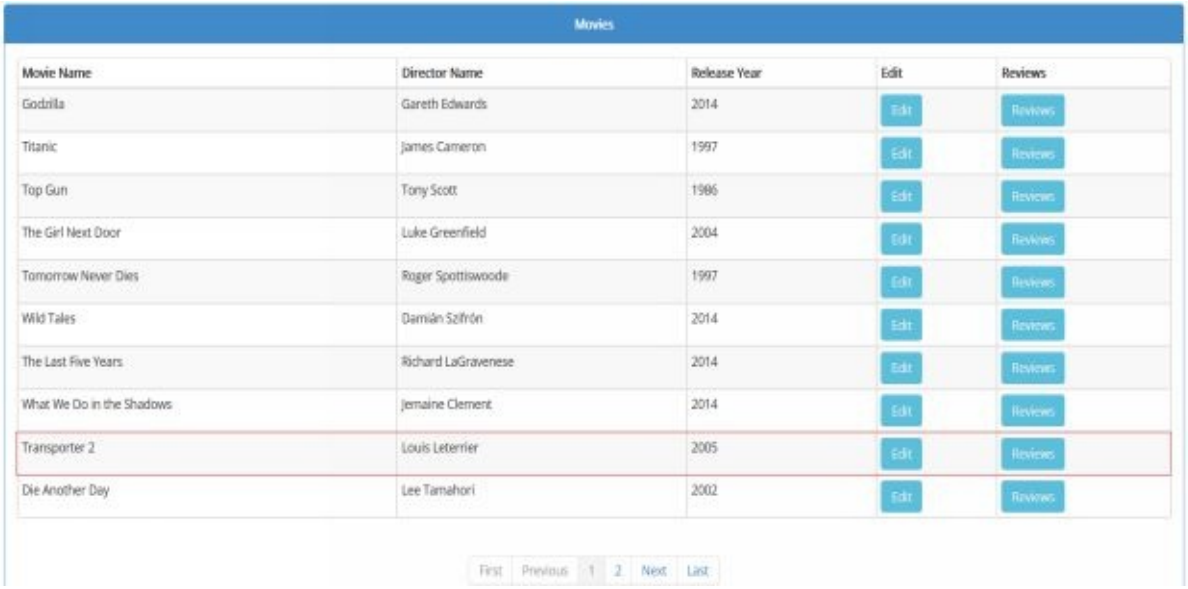

Now, let's suppose if we try to post the Form as it is blank, then it won't allow, because above fields are required as marked by its CSS color and star mark as well. Now, once I enter any information, different validation will get triggered



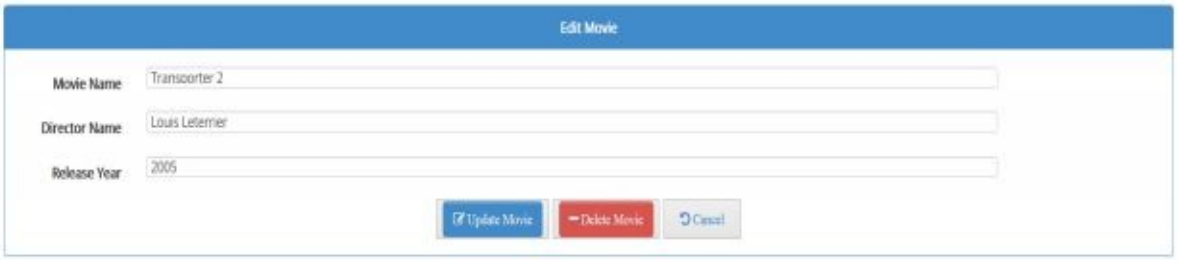

Even at this moment I cannot submit the form as the form is invalid. Once, I modify and enter valid details, then form error messages and its error color (Red) will disappear.

Now, at this instant I can go ahead and submit the movie. Once I click submit button; one toast message will appear saying **Data Saved Successfully** and will get redirected back to movies link.



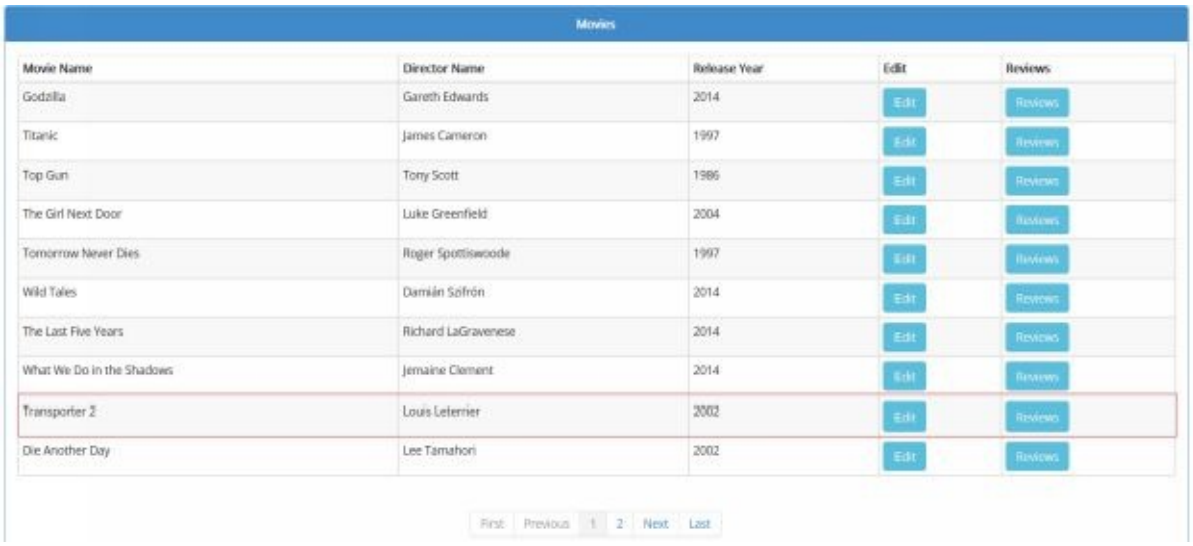




Next is the Edit link corresponding to the movie. When you click on this, it will present the below screen for editing the same.

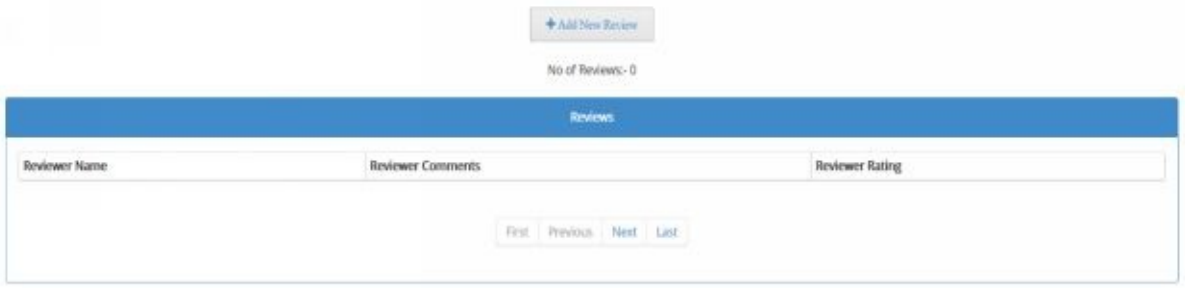



Here, also each and every validation will be there, what we have seen during creation. However, you can go ahead and edit anything over here, let’s say I change the year to 2002 and update.


Once I update the movie it will save the same in database and then get redirected to Movies link as shown below.

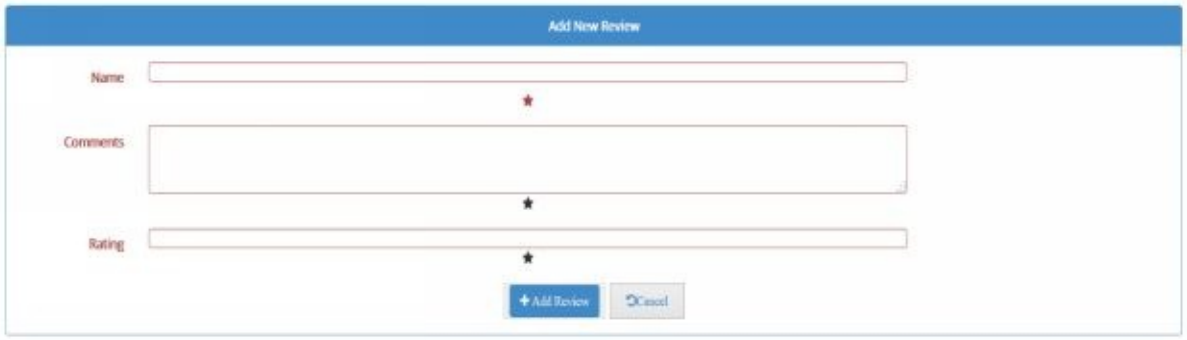



However, let me change the same back to the original one as it's not correct. Similarly, you can go ahead and delete the movie from the Edit link as well. Next to **Edit** link, there is link for **Reviews** as well. From this link you can go ahead and add new review as shown below.



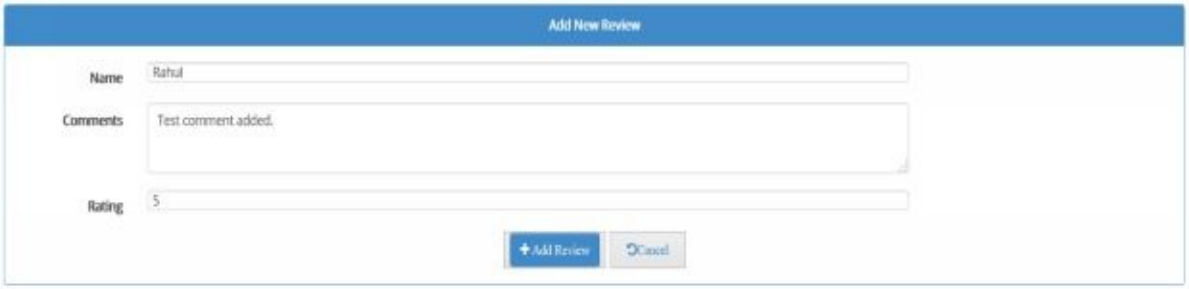

---

Here, when I click on **Add New Review**, it will take me to the below form.

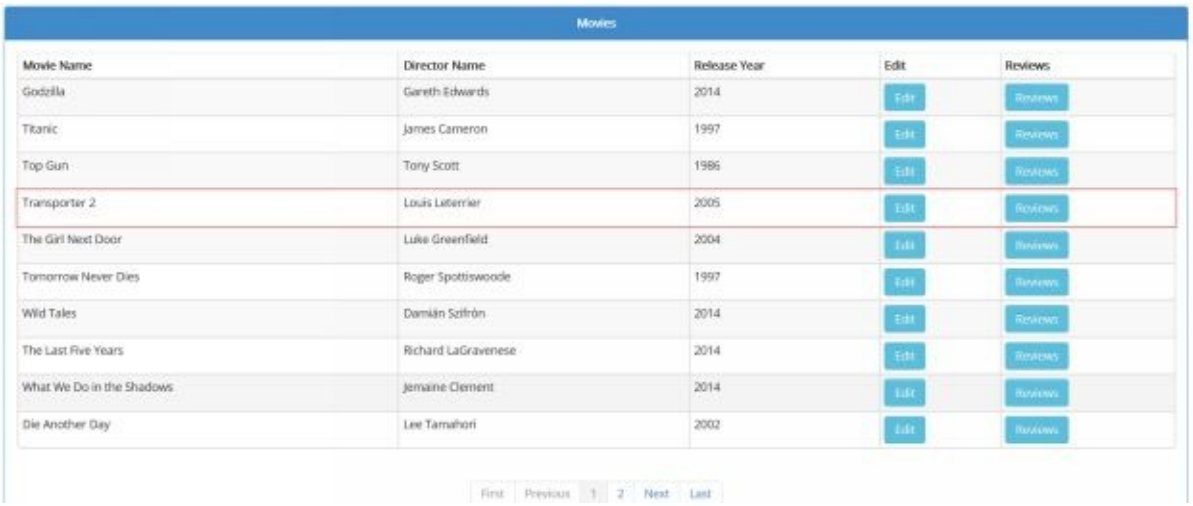



Above form has also got different set of validations as shown below.

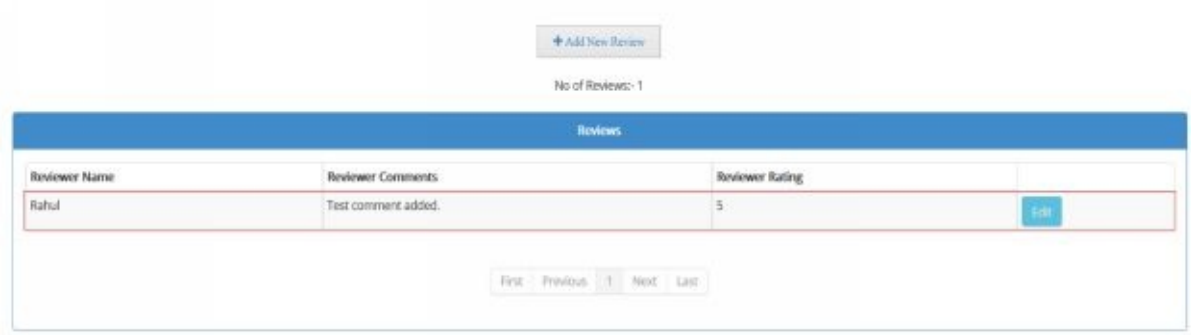

Once done all the corrections, it will be like this



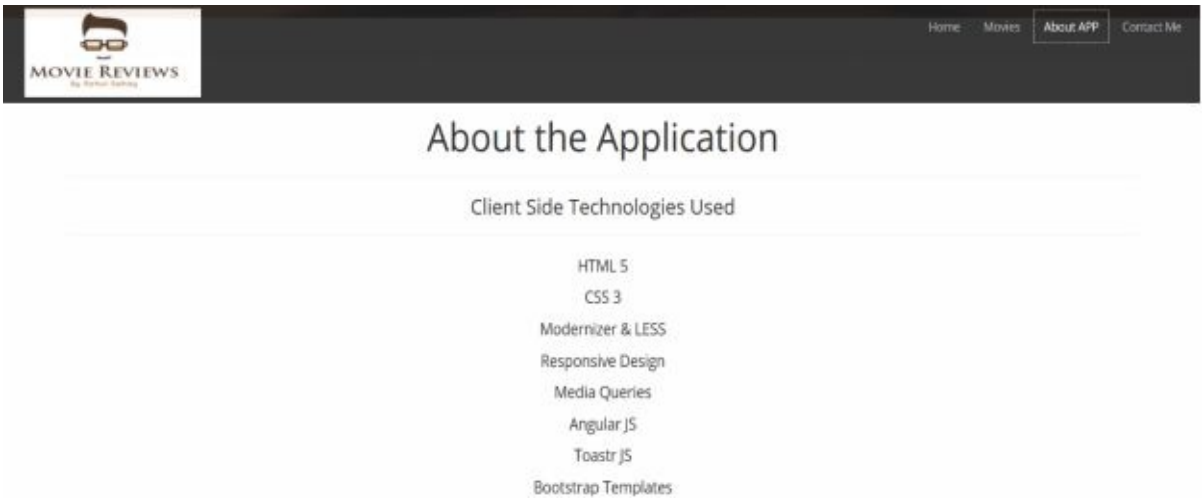

After successful, submission, it will redirect back to the **movies** link.

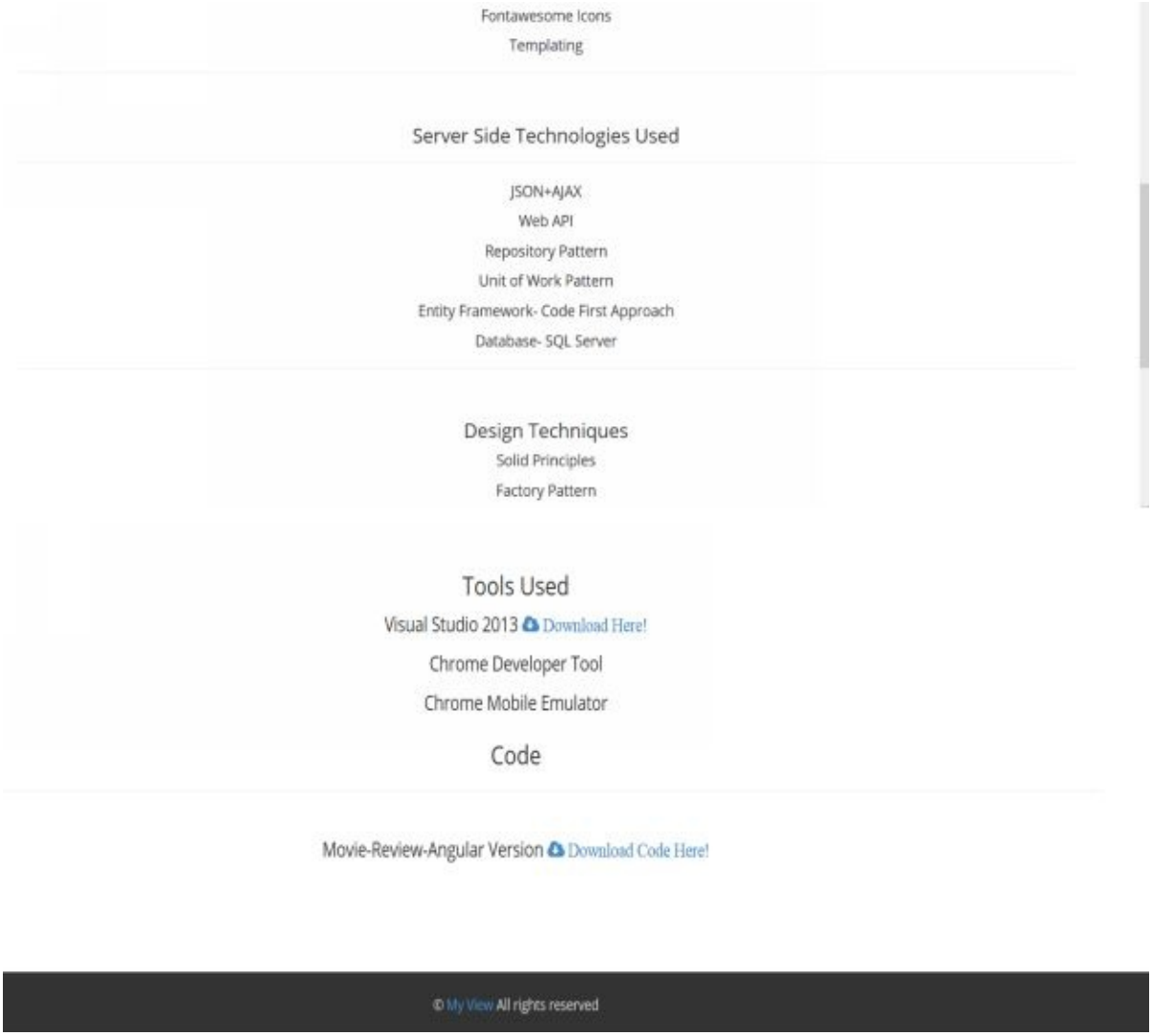


After adding Review, position of newly added movie moves up in the list as behind the scene order by clause is working on total no of reviews. Now, when I click on the Reviews link again, it will show me the review which I have added.



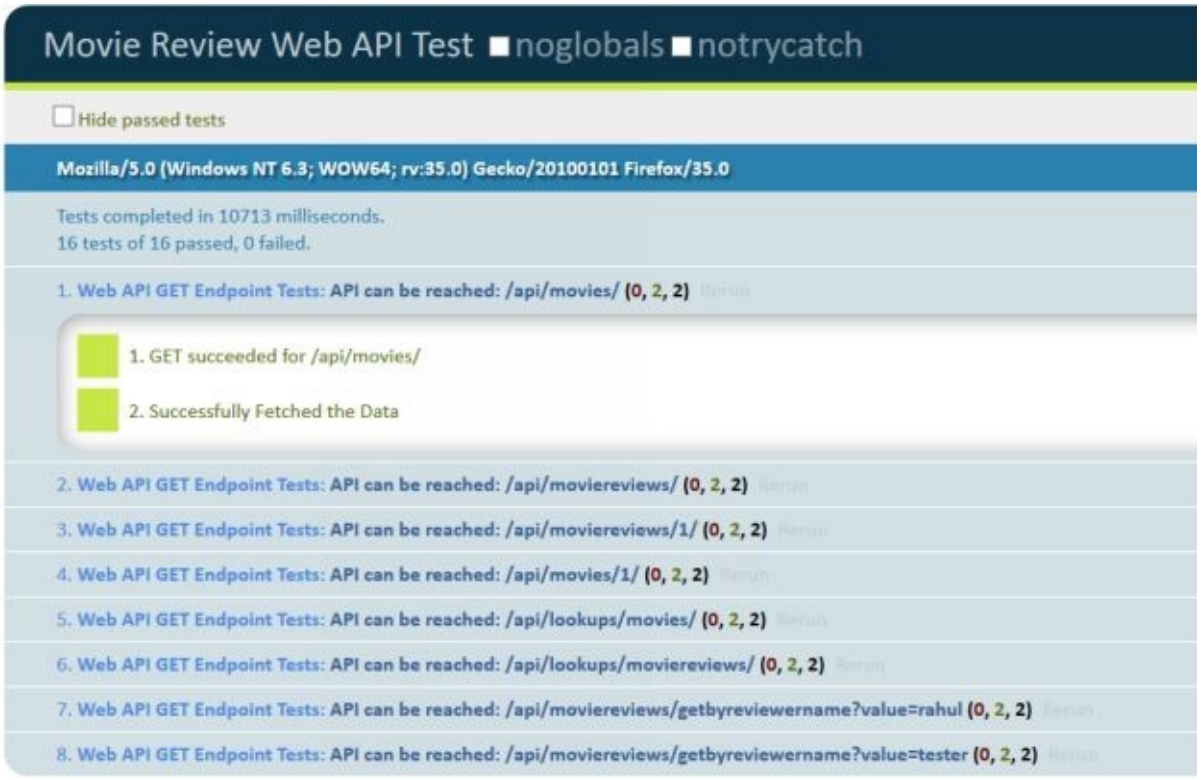

Here, corresponding to the Review, new Edit link also got enabled for editing or deleting the Review. This also works same what I explained above for Movie. **About App** link lists all the details of the application like what technologies used what tools you need, where to download the code etc….






I have also used **QUnit** Library to test my **Web APIs**. Below is the glimpse for the same.

When you click on any individual test, it will present you the detailed results as shown below in the screen shot.



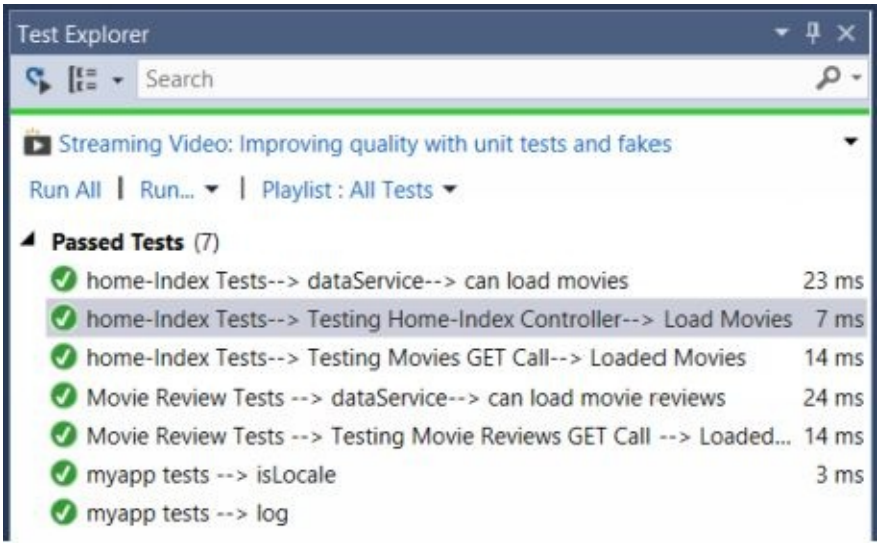

And if there is anything wrong with any end points, let's suppose that doesn't exist; easiest way to break the test, then it will be like


You can also verify these APIs like shown below

http://rahulsahay19-001-site1.smarterasp.net/api/movies

```
[
  {
      Id: 1,
      MovieName: "Godzilla",
      DirectorName: "Gareth Edwards",
      ReleaseYear: "2014",
      NoOfReviews: 6
  },
  {
      Id: 3,
      MovieName: "Titanic",
      DirectorName: "James Cameron",
      ReleaseYear: "1997",
      NoOfReviews: 3
  },
  {
      Id: 9,
      MovieName: "Top Gun",
      DirectorName: "Tony Scott",
      ReleaseYear: "1986",
      NoOfReviews: 1
  },
  {
      Id: 16,
      MovieName: "Transporter 2",
      DirectorName: "Louis Leterrier",
      ReleaseYear: "2005",
      NoOfReviews: 1
  },
  {
      Id: 10,
      MovieName: "The Girl Next Door",
      DirectorName: "Luke Greenfield",
      ReleaseYear: "2004",
      NoOfReviews: 0
  },
```

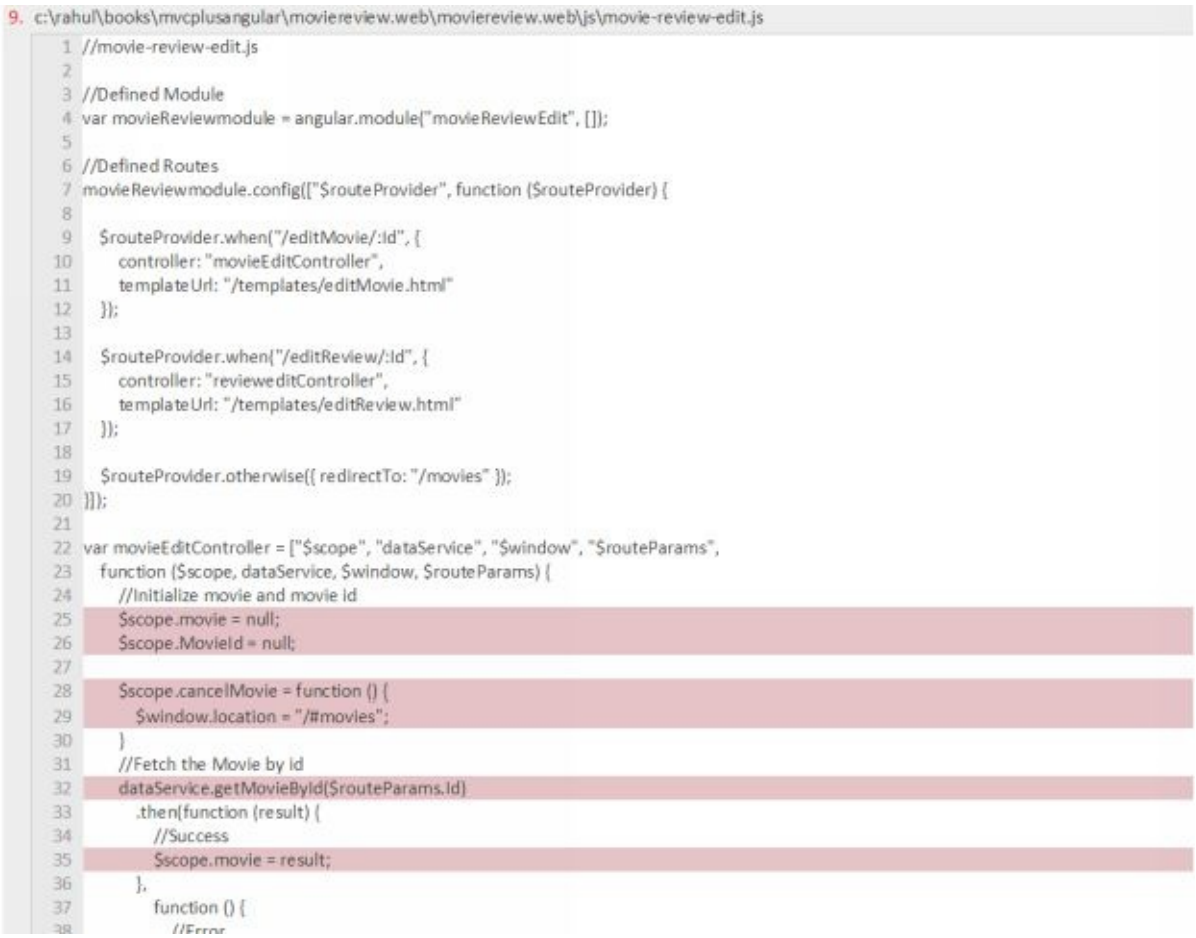

Similarly different endpoints can be tested. Apart from Web API tests. I have also used **Jasmine** to test my Angular code. You will also learn how to test client side JavaScript code with Visual Studio. Here, I have used **Chutzpah** to integrate **JavaScript Tests** with **Visual Studio**. Below is the glimpse for the same.



Now, when I check code coverage results for my tests, Chutzpah will open a new window in browser with the code coverage results for the client side as shown below in the screen shot.

| Chutzpah code coverage via blanket.js results | Covered/Total Smts. | Coverage (%) |
|---|---|---|
| 1. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\appjstest.js | 3/5 | 60 % |
| 2. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\myapptest.js | 5/5 | 100 % |
| 3. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular.min.js | 117/208 | 56.25 % |
| 4. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\ui-bootstrap-tpls.min.js | 2/2 | 100 % |
| 5. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular-route.min.js | 6/8 | 75 % |
| 6. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular-mocks.js | 219/558 | 39.25 % |
| 7. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\Scripts\jquery-1.10.2.min.js | 3/3 | 100 % |
| 8. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\homeindex.js | 43/148 | 29.05 % |
| 9. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\movie-review-edit.js | 7/51 | 13.73 % |
| 10. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\moviereviewtest.js | 21/21 | 100 % |
| 11. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\homeindextest.js | 30/30 | 100 % |
| 12. Total | 436/1039 | 41.89 % |

The ones which are highlighted in red are the ones which are not covered 100%, so when I click on any of this link, it will open the code in browser and show what is covered and what is not

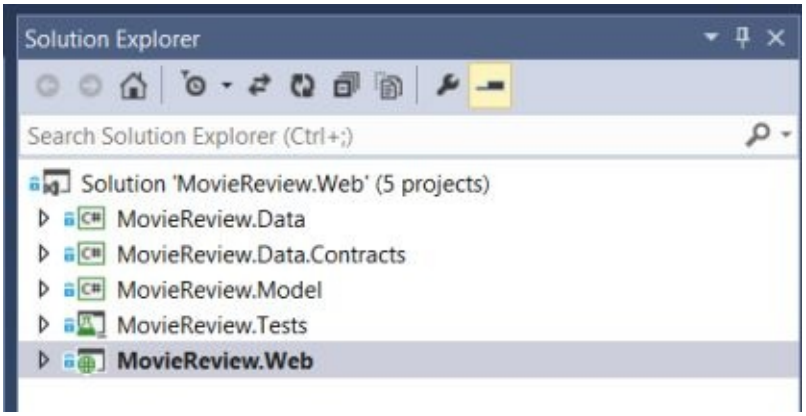


9. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\movie-review-edit.js

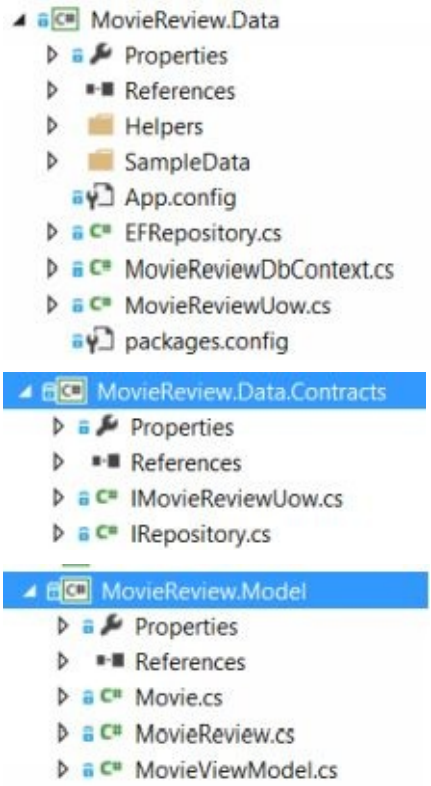
```
1  //movie-review-edit.js
2
3  //Defined Module
4  var movieReviewmodule = angular.module("movieReviewEdit", []);
5
6  //Defined Routes
7  movieReviewmodule.config(["$routeProvider", function ($routeProvider) {
8
9    $routeProvider.when("/editMovie/:Id", {
10     controller: "movieEditController",
11     templateUrl: "/templates/editMovie.html"
12   });
13
14   $routeProvider.when("/editReview/:Id", {
15     controller: "revieweditController",
16     templateUrl: "/templates/editReview.html"
17   });
18
19   $routeProvider.otherwise({ redirectTo: "/movies" });
20 }]);
21
22 var movieEditController = ["$scope", "dataService", "$window", "$routeParams",
23   function ($scope, dataService, $window, $routeParams) {
24     //Initialize movie and movie id
25     $scope.movie = null;
26     $scope.MovieId = null;
27
28     $scope.cancelMovie = function () {
29       $window.location = "/#movies";
30     }
31     //Fetch the Movie by id
32     dataService.getMovieById($routeParams.Id)
33       .then(function (result) {
34         //Success
35         $scope.movie = result;
36       },
37         function () {
38           //Error
```

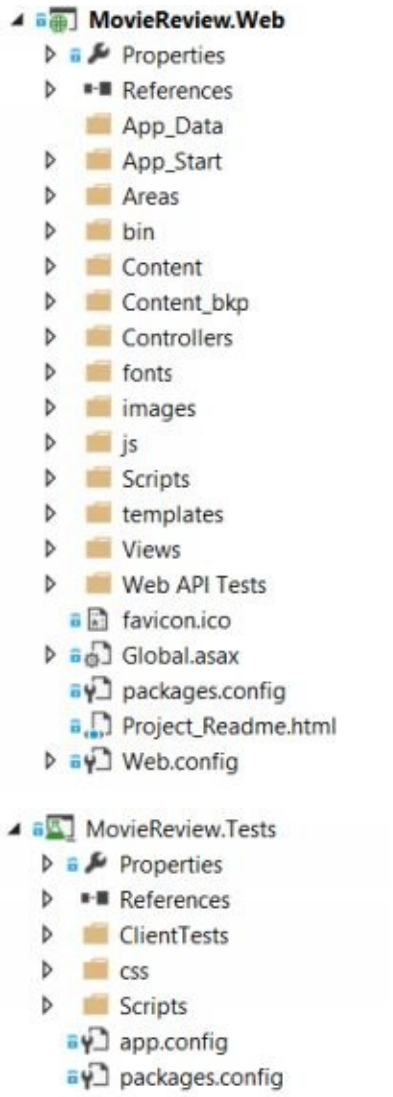
## Glimpse of Movie Review Solution:-

Let me go ahead and show you the solution structure of finished app. Below in the screen shot, I have 5 different projects. Each is having its on dependency and responsibility.




Here, the highlighted one is the web project which is dependent on other infrastructure projects **Data, Contracts** and **Model**. **Data** project is the place where in you maintain initial seed data, Entity Framework DBContext and many more things involve direct interfacing down the layer with database. **Data Contract** is the place where in you manage your repositories and apply **Unit of Work Pattern** on repositories like **movies** and **moviereviews**. **Model** is the place where you will be having your **POCOs** (**Plain Old CLR Objects**). This is the place where in you are maintaining all properties attributed to the tables. Below is the glimpse of all projects in its expanded form.






### Summary:-

In this section, we have seen the bits and bytes of the Single Page Application which we are going to make using tons of client/server side framework. We have also seen the glimpse of app and solution. In the next section, we'll begin the learning by creating the application right from the scratch. I would recommend you to download the app from github URL shown below to help you while building https://github.com/rahulsahay19/MovieReview-Angular-Prod .

## Chapter 2: Creating Solution from the blank slate

### WHAT DO you find in this CHAPTER?

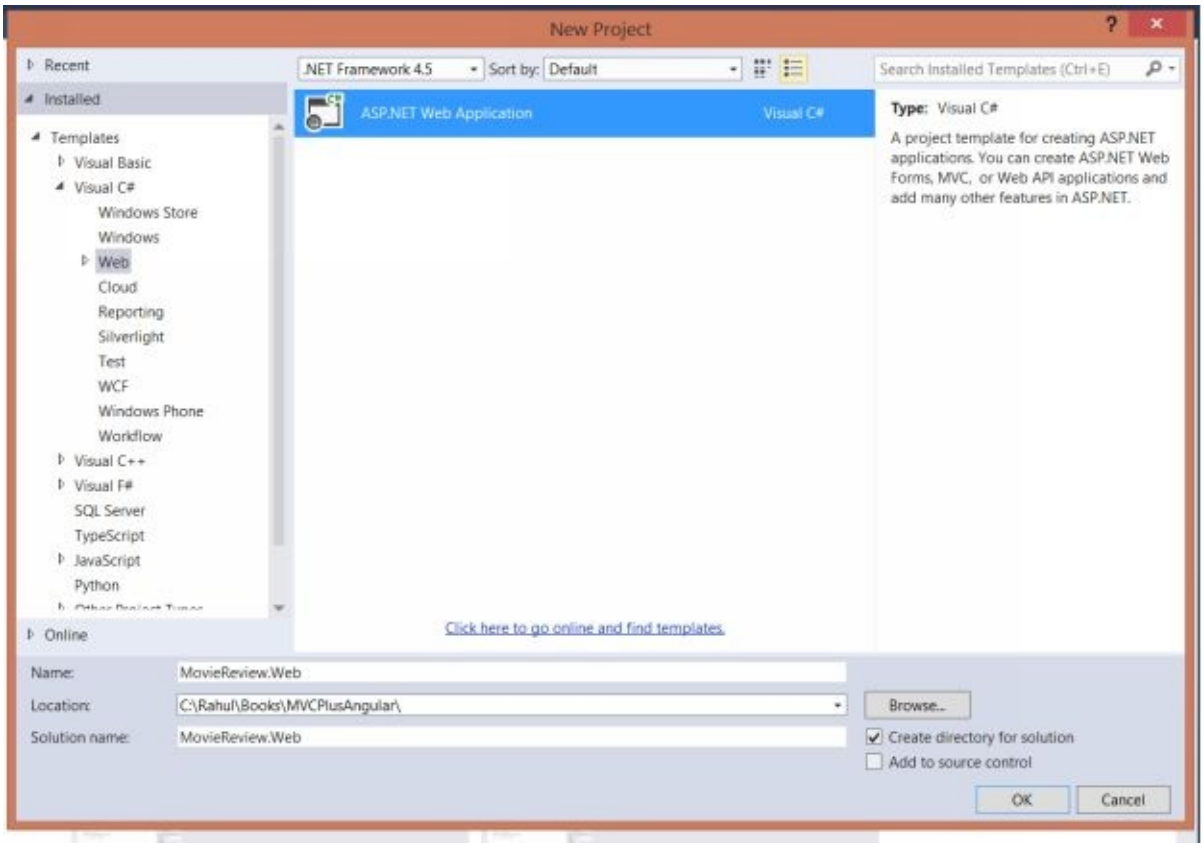
- Introduction
- Solution Creation
- Adding Project References
- Adding Packages
- Important Tools
- Data Technologies
- Creating Models
- Creating Entity Framework
- Database Initializer
- Implementing Repository Pattern
- Creating Unit Of Work Pattern (UOW)
- Summary

## Introduction:-

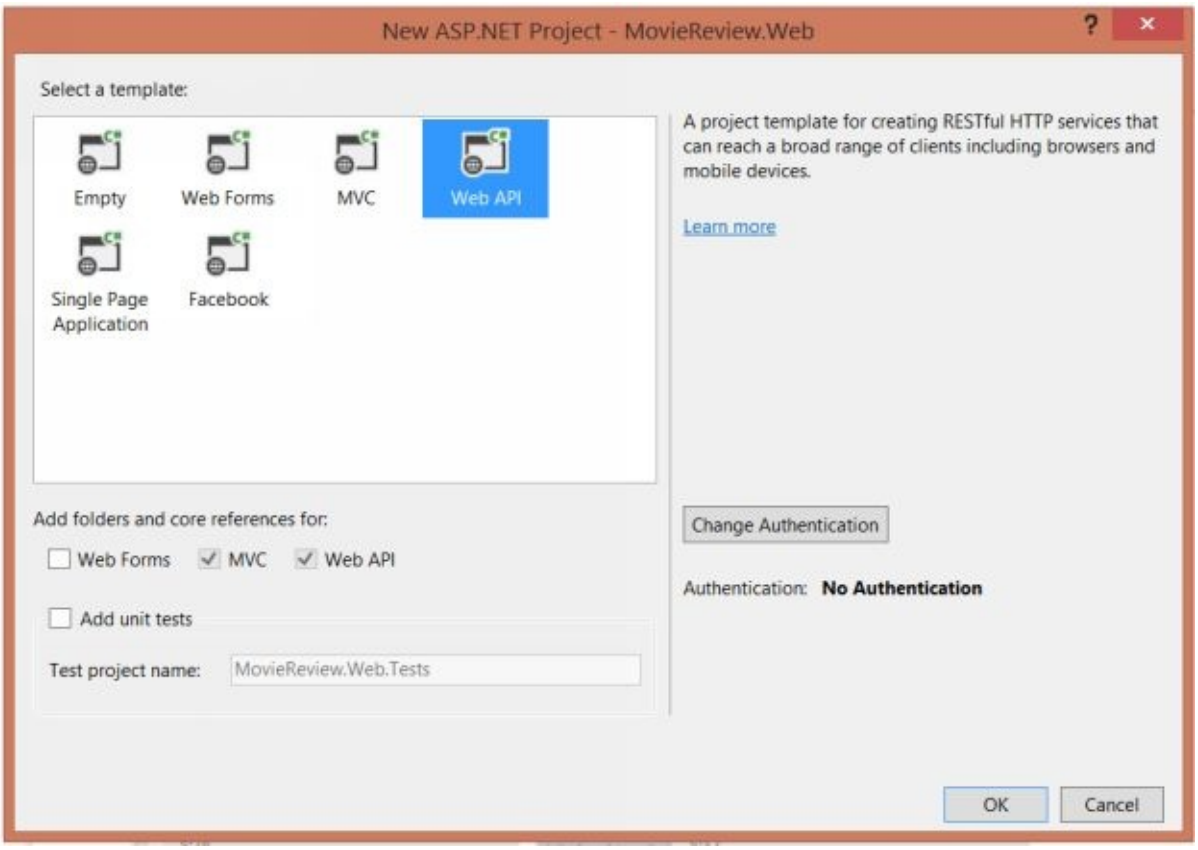
In this module, we'll begin with blank solution as shown below in the screen shot. Now, in this blank solution we'll add couple of different projects to complete the solution structure of our movie app. Then, we'll complete the necessary references of projects to each other. After, resolving all project dependencies we'll begin by installing necessary **client/server** side components using **Nuget**.

## Solution Creation:-

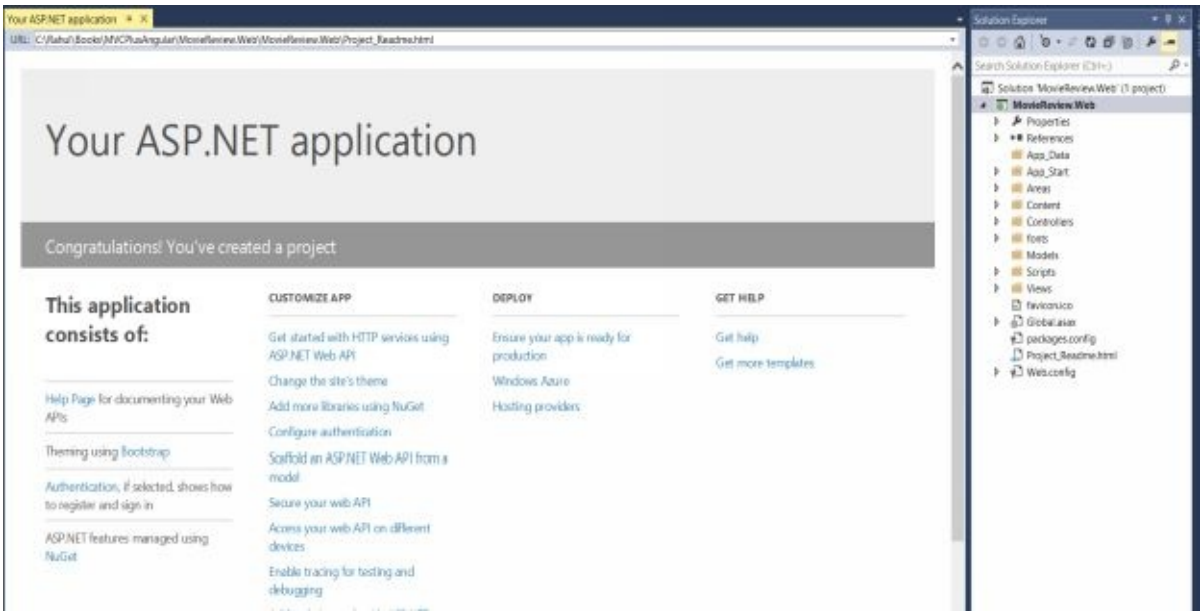
In this section, we'll start creating the complete architecture right from the blank slate. So, without wasting time let's get started. Below in the screen shot I have started creating the application with web project.



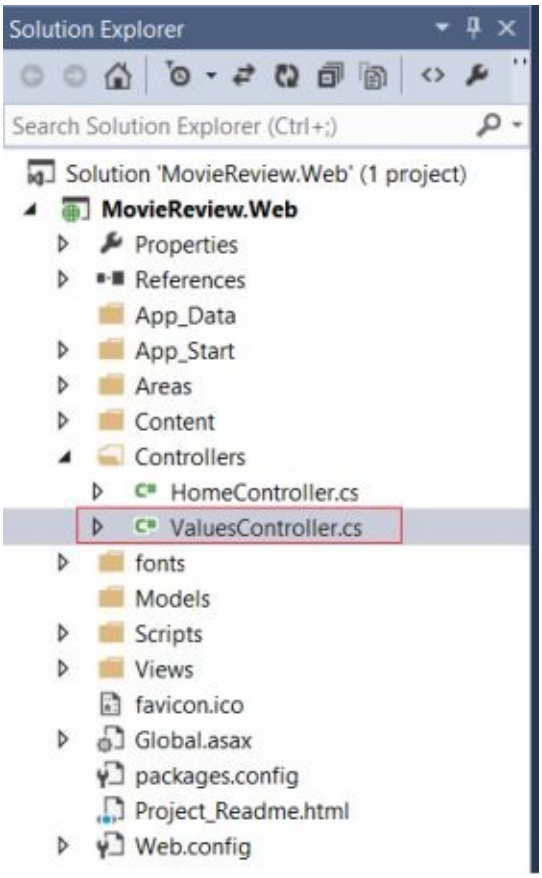

Here, I'll be selecting Web API template just to make sure that I have necessary configurations in place for building my APIs which will talk back and forth with User Interface. I'll pick default implementation as shown below in the screen shot. However, if you notice that I have also left Unit Test project unchecked as I'll introduce new technique to test my APIs. Later I'll add Test Project separately.



Once, the project gets created successfully, it will present one default readme HTML page as shown below in the screen shot with default files and configurations as shown in solution explorer tab.




Let’s examine few key pieces of the project. By default it gave me two controllers Values Controller and Home Controller as shown below but we’ll be deleting these and writing ours right from the scratch.



ValuesController.cs | HomeController.cs

MovieReview.Web.Controllers.ValuesController — Get()

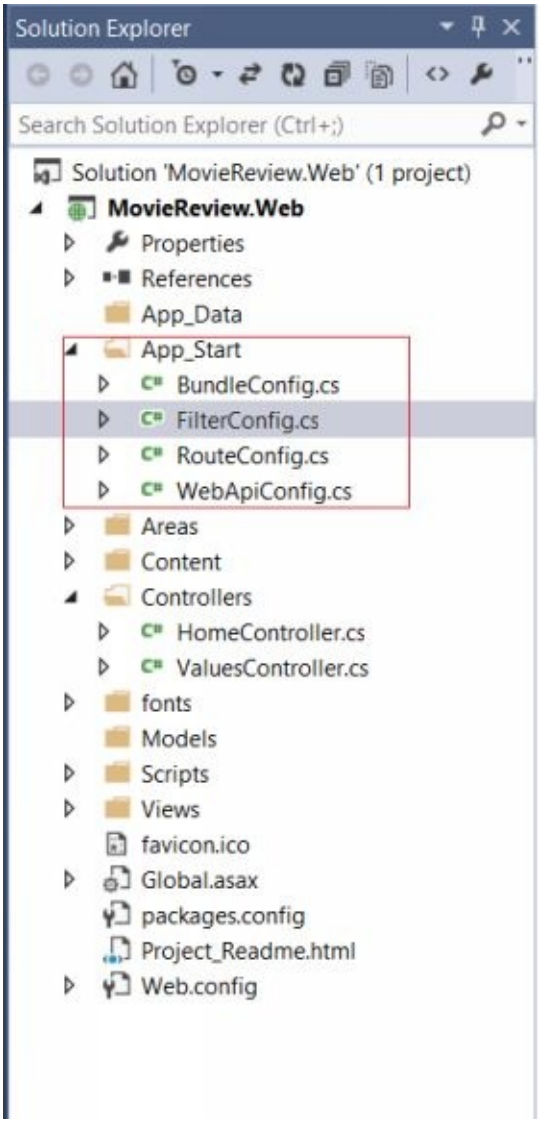
```
using System.Collections.Generic;
using System.Web.Http;

namespace MovieReview.Web.Controllers
{
    0 references
    public class ValuesController : ApiController
    {
        // GET api/values
        0 references
        public IEnumerable<string> Get()
        {
            return new string[] { "value1", "value2" };
        }

        // GET api/values/5
        0 references
        public string Get(int id)
        {
            return "value";
        }

        // POST api/values
        0 references
        public void Post([FromBody]string value)
        {
        }

        // PUT api/values/5
        0 references
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE api/values/5
        0 references
        public void Delete(int id)
        {
```

ValuesController.cs | HomeController.cs

MovieReview.Web.Controllers.HomeController

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MovieReview.Web.Controllers
{
    0 references
    public class HomeController : Controller
    {
        0 references
        public ActionResult Index()
        {
            ViewBag.Title = "Home Page";

            return View();
        }
    }
}
```

Also, there is one more important piece which I want you to focus is the **App_Start** folder. This contains all the configs related files or basically startup files I would say.




Let’s talk about them in briefly now, however we’ll discuss the same in great details in coming chapters with their usage.

- Bundle Config: - This is the place which will take care of bundling and minification of all scripts and css.

- Filter Config: - Filter config is the section which will help with Filter registration if any we are using in the app.

- Route Config: - Route config is the nerve of the app as it will decide all the routing architecture of the application. Throughout in our application we’ll be using it heavily. So stay tuned we’ll discuss the same in detail when we start writing the

routes.

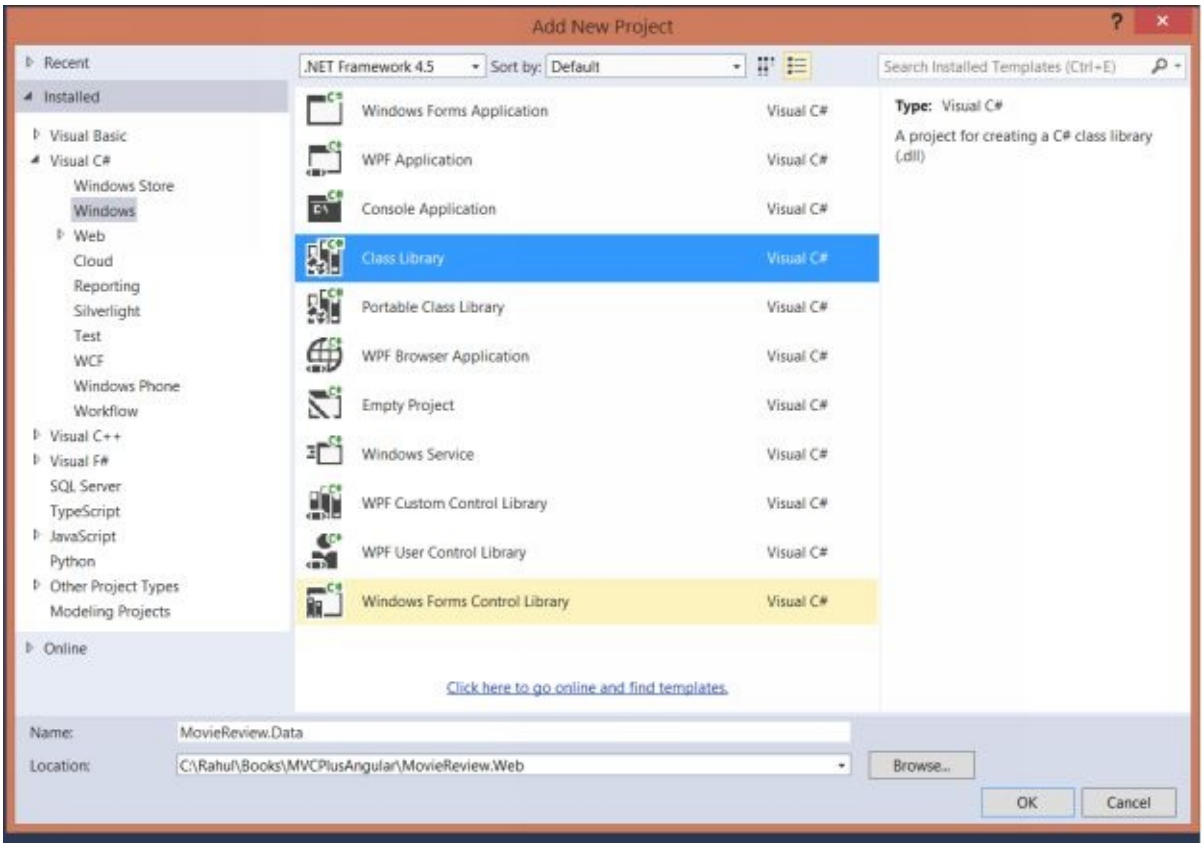
- Web API Config: - All API related stuffs or configurations can be done from this file. We will also cover the same in coming sections.

Now, apart from this there is one more important file which triggers all these settings and that is none other than **Global.asax** file as shown below in the sample snippet.

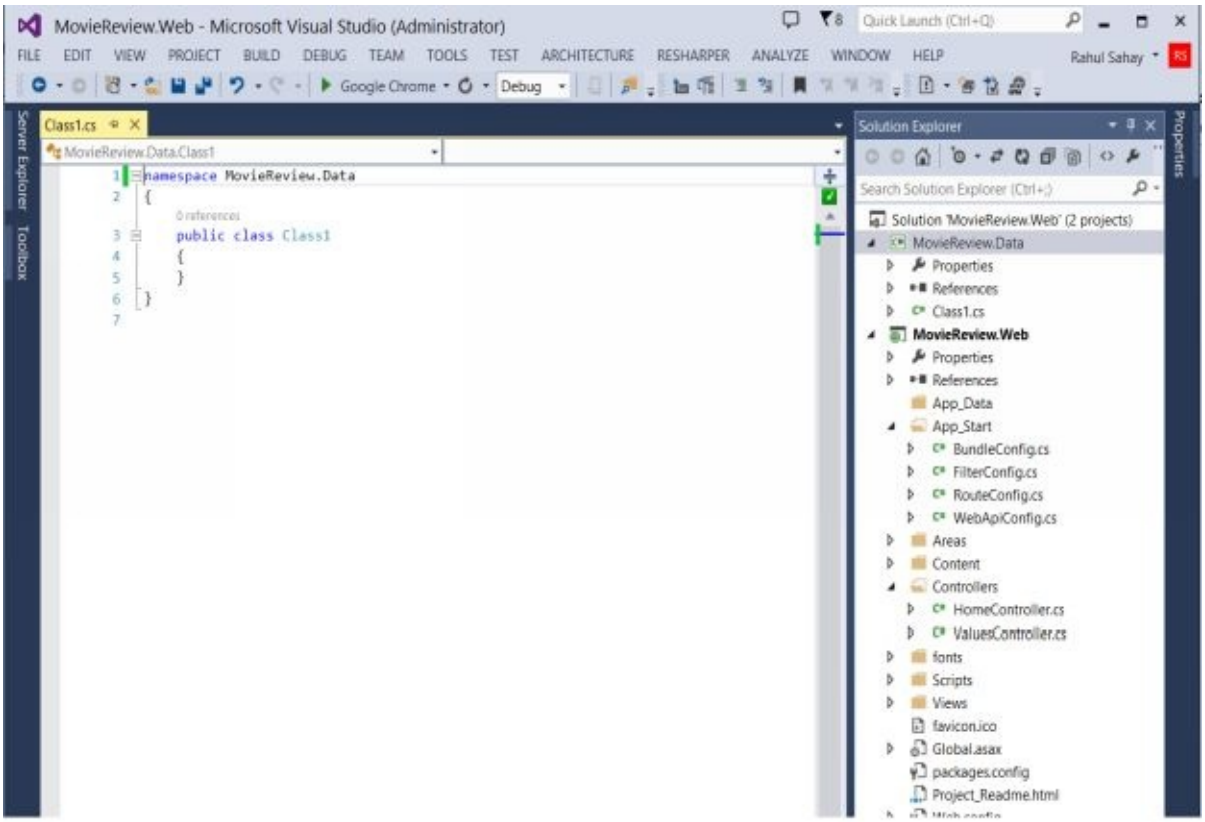
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Http;
using System.Web.Mvc;
using System.Web.Optimization;
using System.Web.Routing;

namespace MovieReview.Web
{
    public class WebApiApplication : System.Web.HttpApplication
    {
    protected void Application_Start()
    {
    AreaRegistration.RegisterAllAreas();
    GlobalConfiguration.Configure(WebApiConfig.Register);
    FilterConfig.RegisterGlobalFilters(GlobalFilters.Filters);
    RouteConfig.RegisterRoutes(RouteTable.Routes);
    BundleConfig.RegisterBundles(BundleTable.Bundles);
    }
    }
}
```

As you can see in the above snippet all these settings are basically invoked by **Global.asax** file. Once the app starts, this is the 1st one which will run and set up the environment. Hence all environment related stuffs kept here itself.

Now, let's go ahead and add other required projects. So, my other project is going to be repository layer, hence I would call the same **MovieReview.Data.** Since, this is a reusable component, hence I am going to add one class library project.

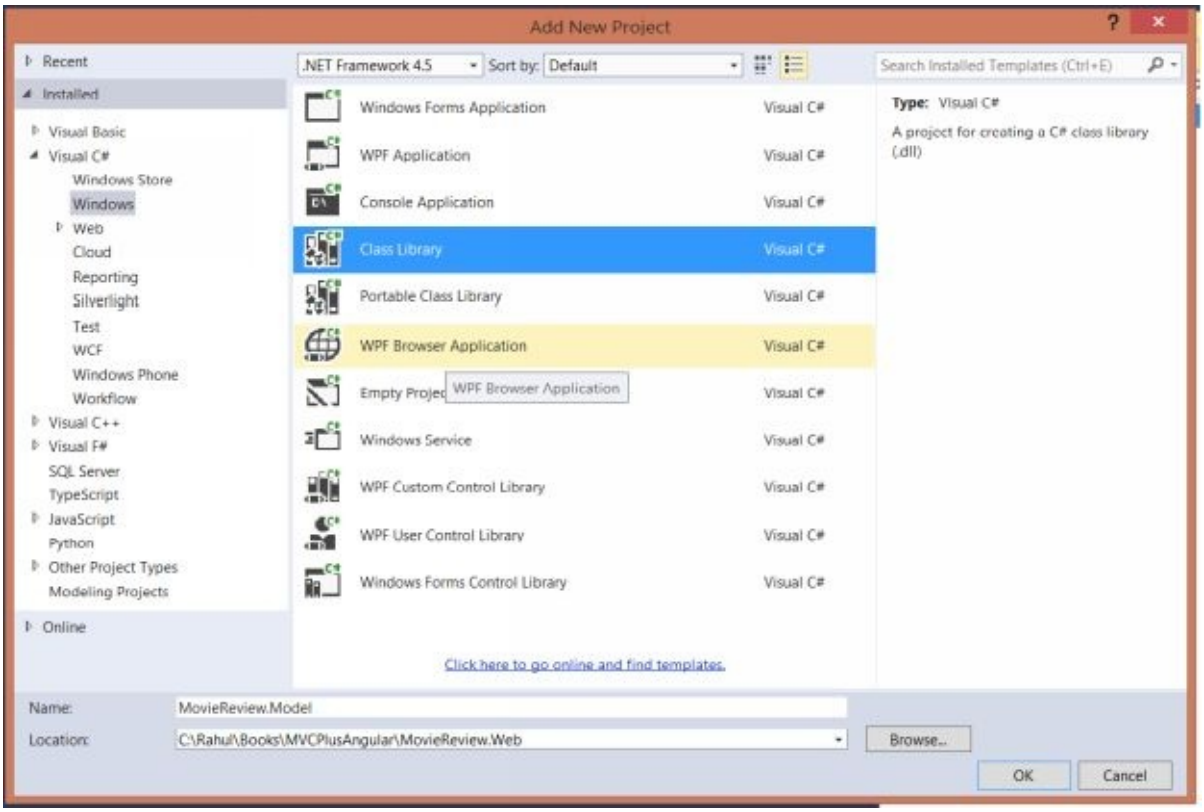



Once this project gets added it will present me one default class as shown below which I will delete as I'll be adding required classes later in the project.




Now, we need another project which will contain our POCO entity, I'll call this project

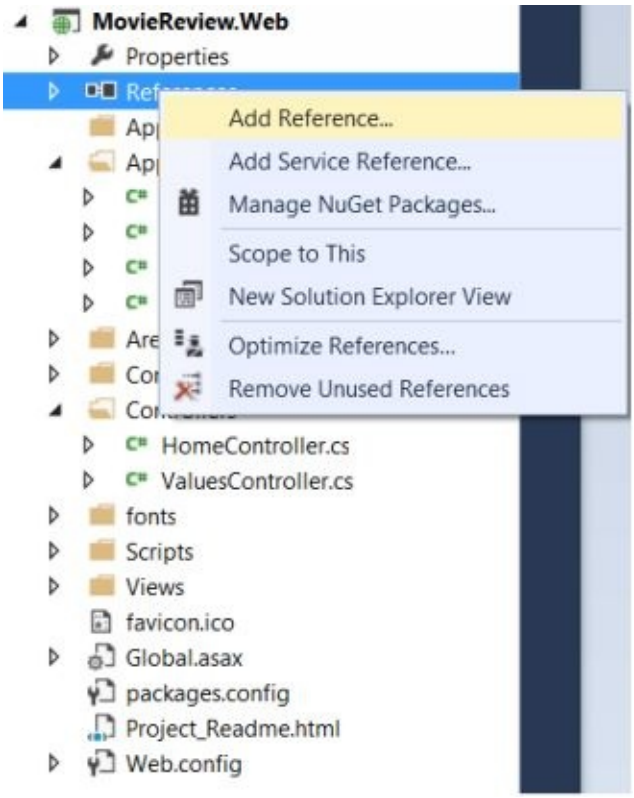
**MovieReview.Model** as shown in the below screen shot.

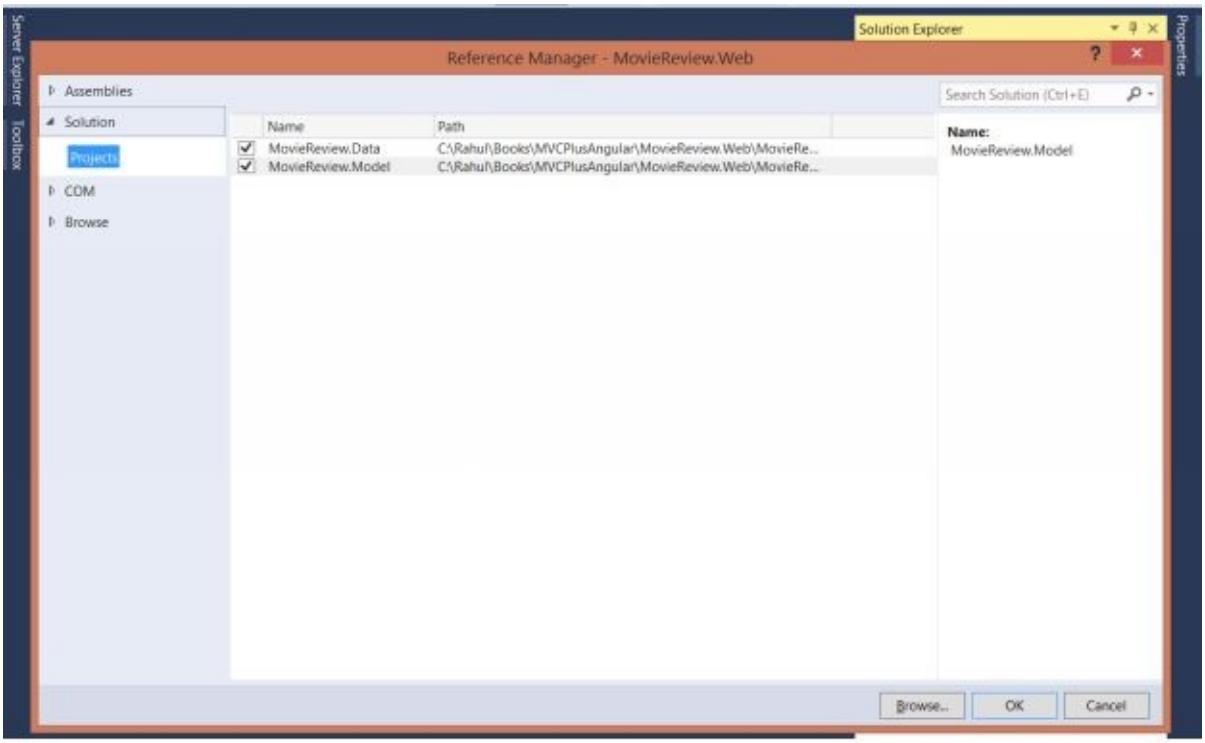



Below I have deleted the default class files from both the projects.

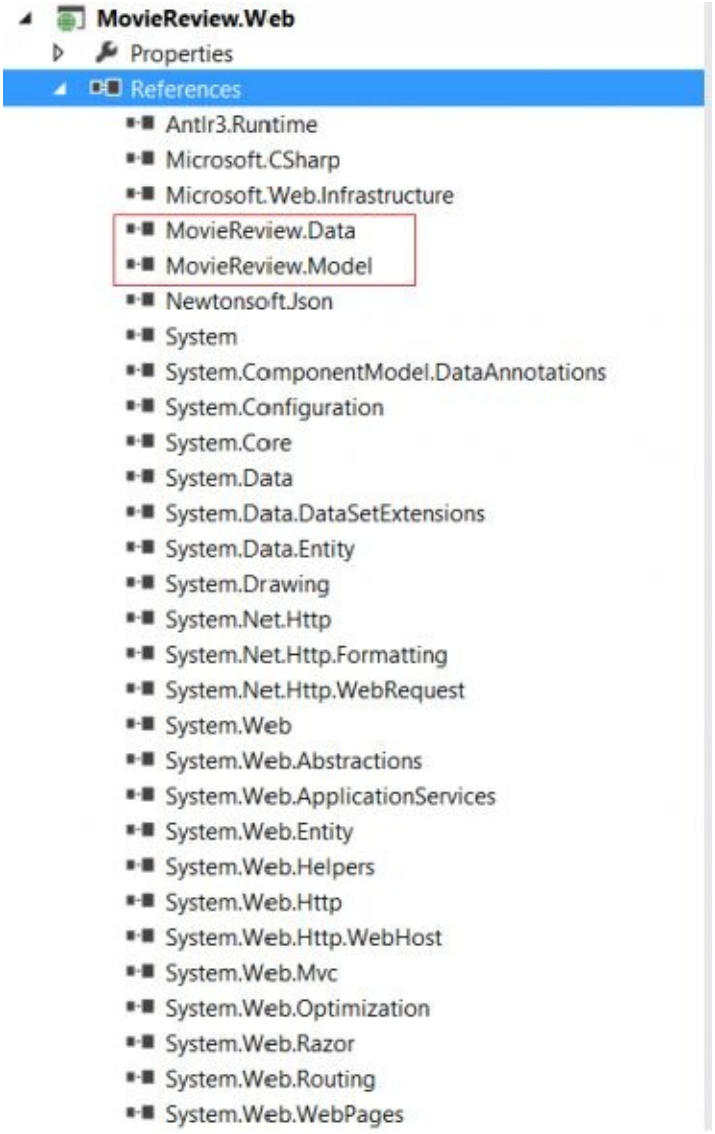

## Adding Project References:-

Once the projects are created, it’s time to make proper reference with each other. Hence, I’ll add my web project with my other two projects as shown below in the screen shot

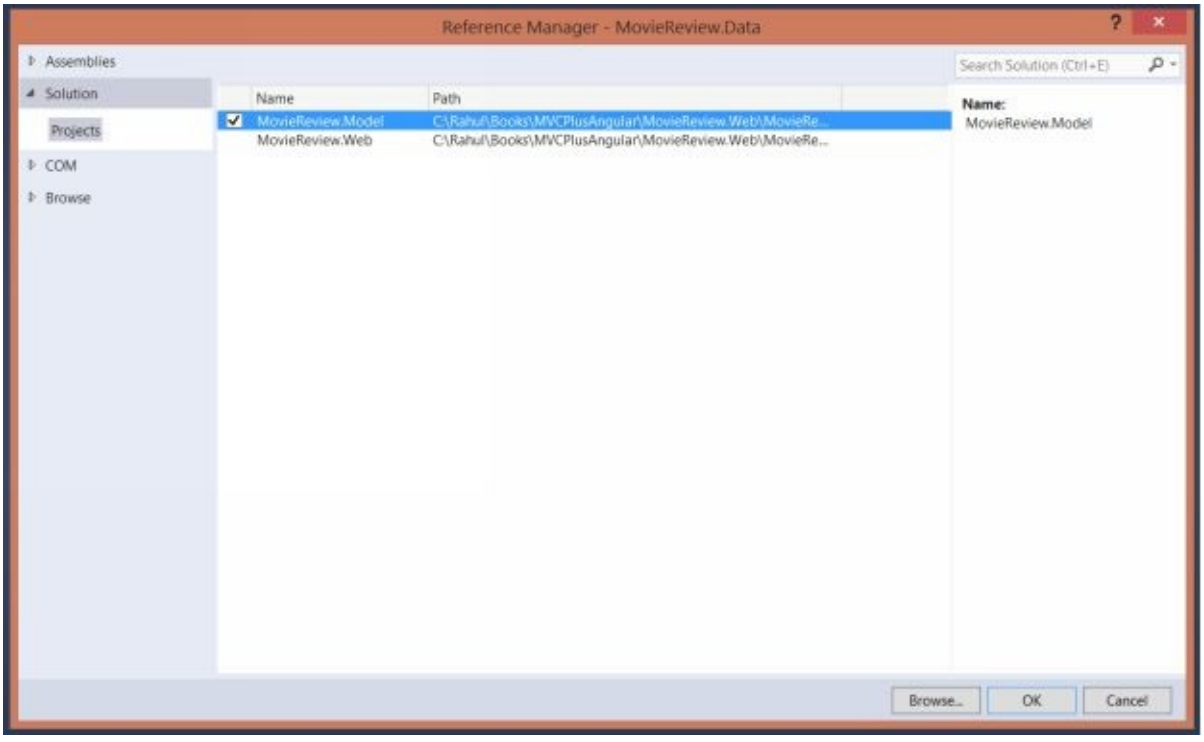


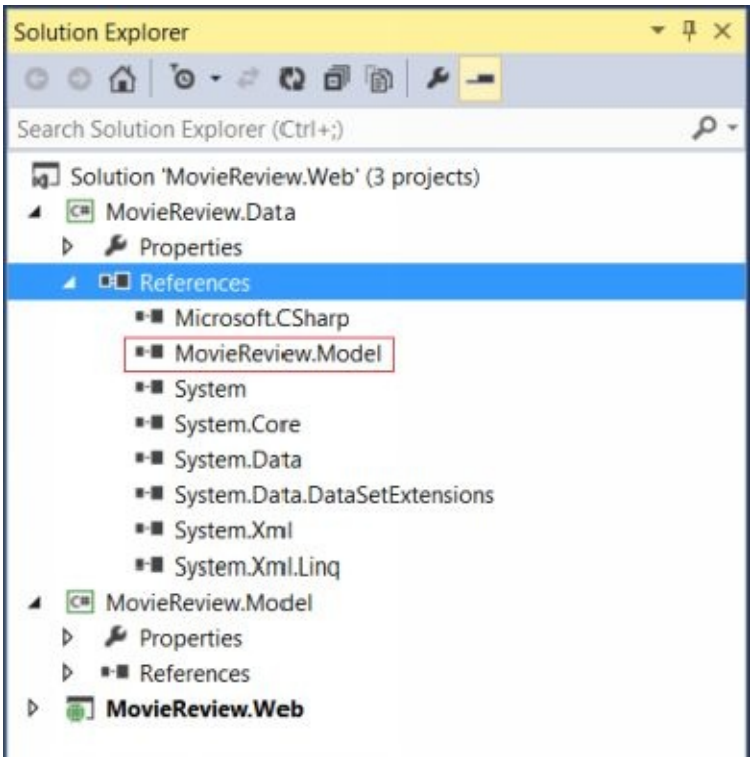




Once the references added successfully, we can verify the same in the references as shown below in the screen shot.



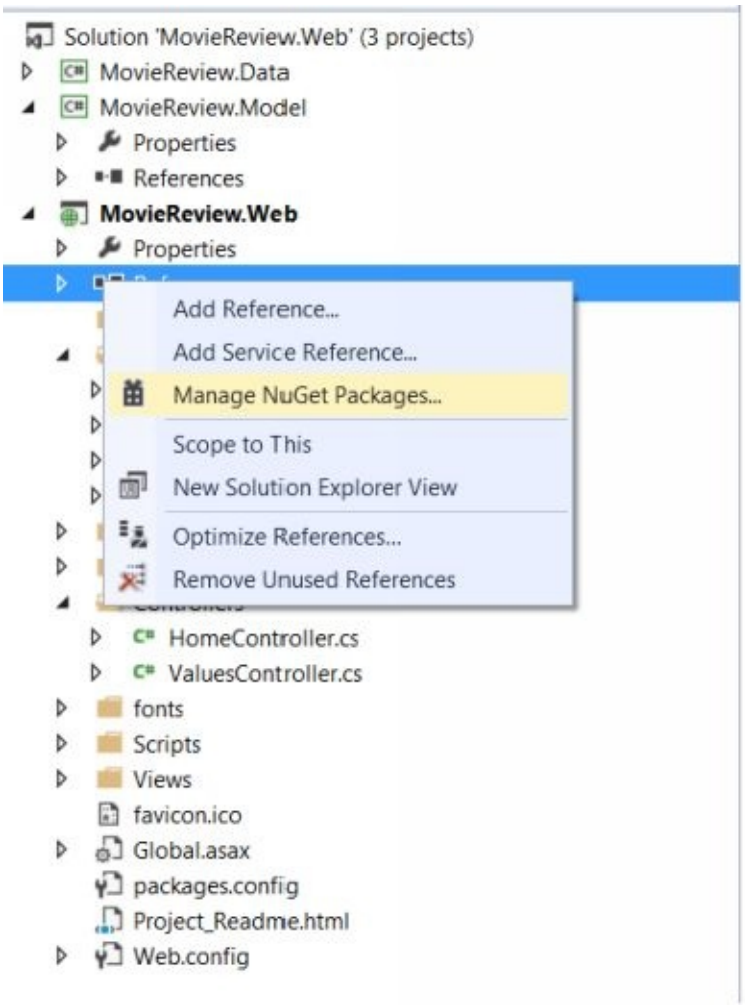

Now, my data project also going to need my POCO models, hence I need to reference that as well



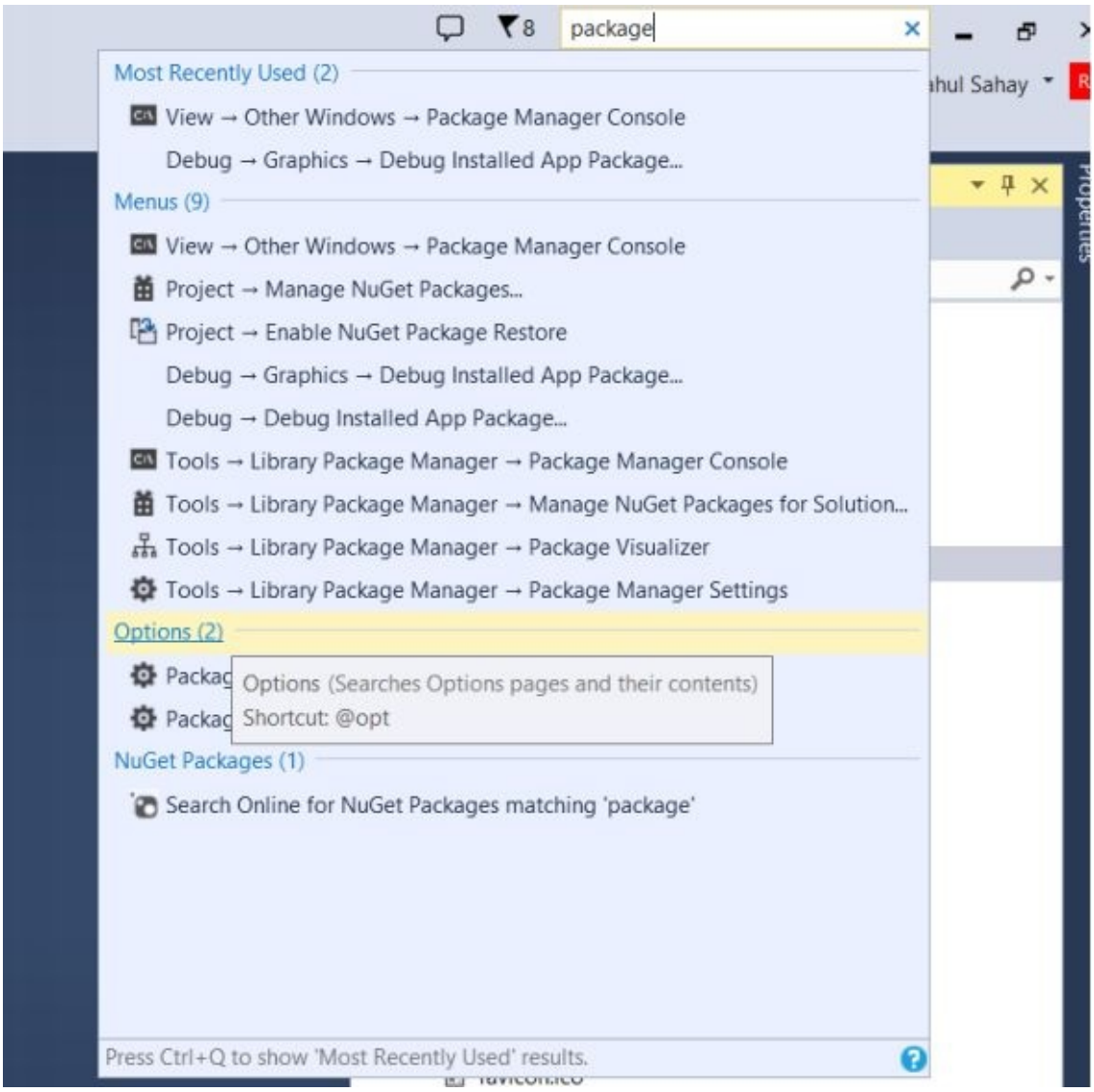


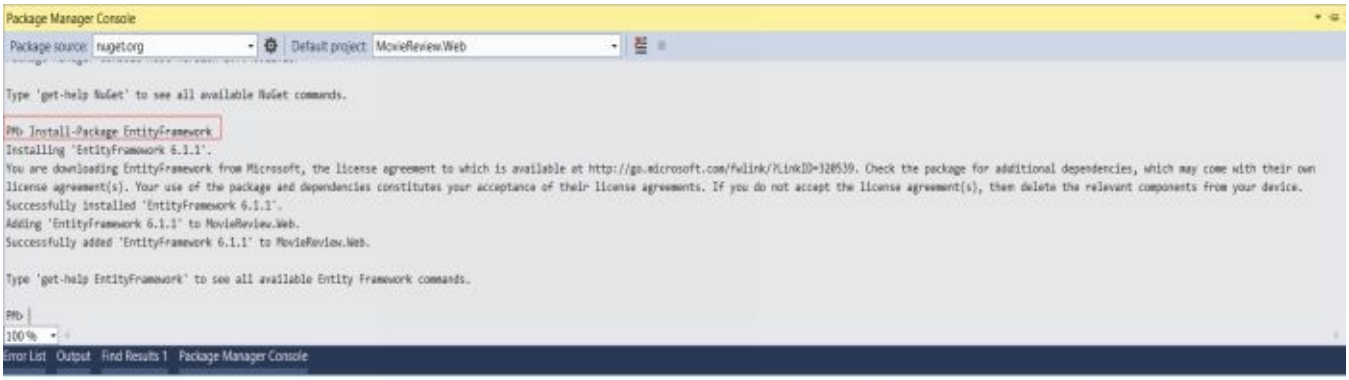

## Adding Packages:-

Once we have done with our project references, it's time to get the required dependencies in there. Hence, for doing that I'll use Package Manage Console to download and install the required packages for me. I can also right-click on the references and say Manage Nuget Package as shown in the below screen shot, but I am much acquainted with the former approach.

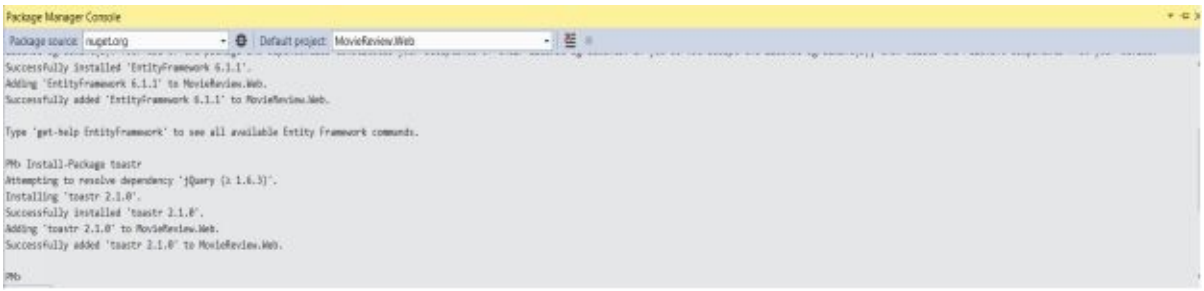



Package Manager Console can be launched by typing in the search bar as shown below. Once opened, I'll go ahead and install required entity framework package in my Web project.



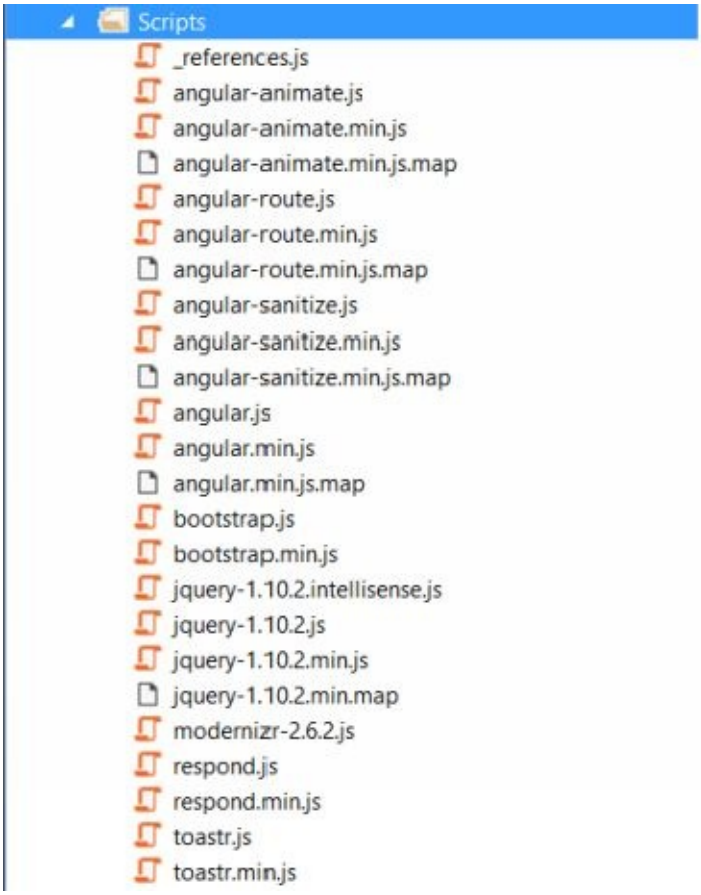




Similarly, I'll go ahead and add my other dependencies like I need Toastr JS for notifications.



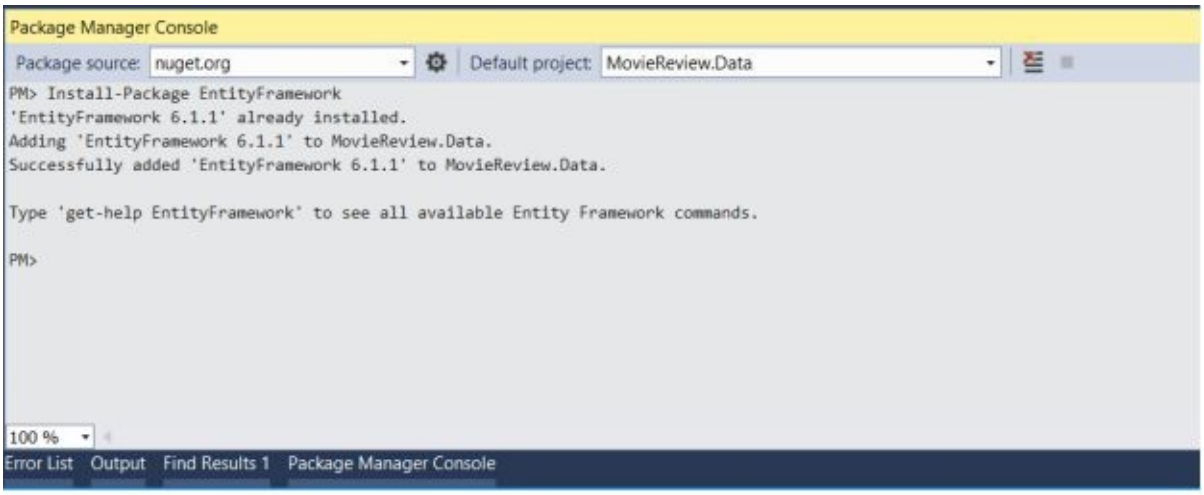

Once, this is done I can go ahead and install my Angular Package as well. Here, is one point of caution, which you must understand with Angular like with every new release angular is moving few components from one file to another file for instance let's suppose that in the base version on which you may have worked, in that version routing features were there in the same angular file but in new versions it has taken out in a different routing file. Like these there are many other pieces which have moved. So, one must keep these dependencies in mind before migrating to higher version of the angular otherwise your app will break.

Another way of doing the same download the file from [http://AngularJS.org](http://AngularJS.org) and refer the same in the application as shown below.

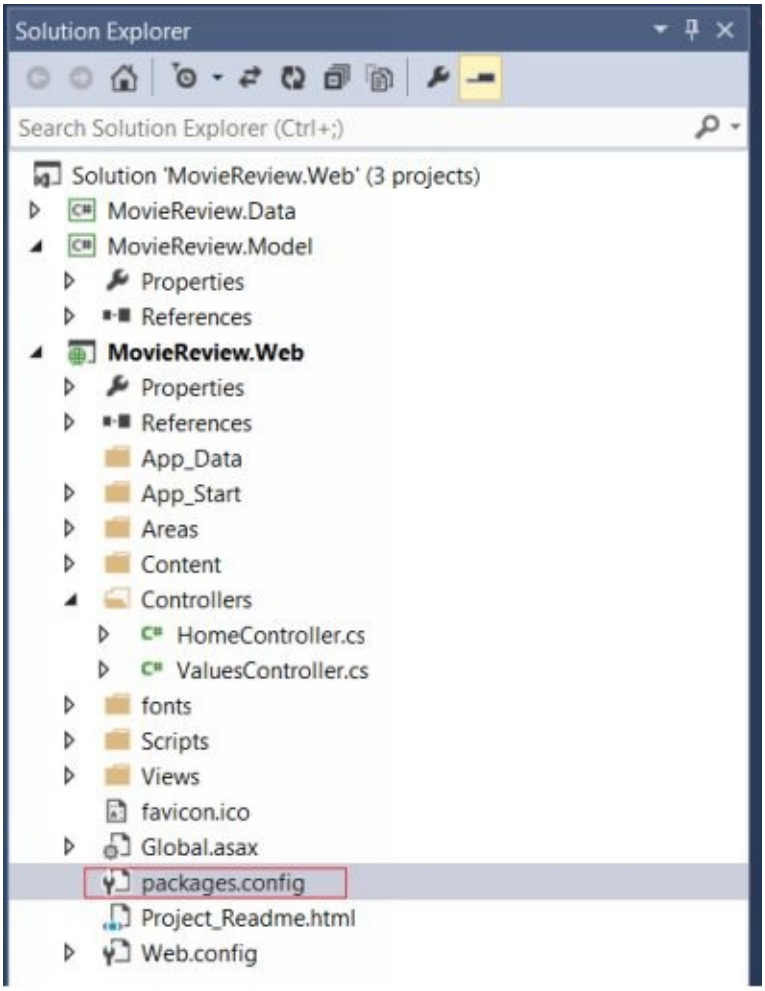



Above is the glimpse of all the required scripts for the time being. As we progresses in the application, we will gradually install other dependencies as well. But, for now it's ok. Now, one more dependency I need to resolve before I begin the show is installing Entity

Framework in the Data project.




As you see in the above screen shot I have changed the project reference to **MovieReview.Data** in the project section and then I installed the package. Now, apart from this there are many other dependencies which visual studio has installed for me with the template I chose. So, if I go to **packages.config** in the web project I could see the complete list of dependencies as shown below in the screen shot.



packages.config

```
<?xml version="1.0" encoding="utf-8"?>
<packages>
  <package id="Antlr" version="3.4.1.9004" targetFramework="net45" />
  <package id="bootstrap" version="3.0.0" targetFramework="net45" />
  <package id="EntityFramework" version="6.1.1" targetFramework="net45" />
  <package id="jQuery" version="1.10.2" targetFramework="net45" />
  <package id="Microsoft.AspNet.Mvc" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.Razor" version="3.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.Web.Optimization" version="1.1.1" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebApi" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebApi.Client" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebApi.Core" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebApi.HelpPage" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebApi.WebHost" version="5.0.0" targetFramework="net45" />
  <package id="Microsoft.AspNet.WebPages" version="3.0.0" targetFramework="net45" />
  <package id="Microsoft.Web.Infrastructure" version="1.0.0.0" targetFramework="net45" />
  <package id="Modernizr" version="2.6.2" targetFramework="net45" />
  <package id="Newtonsoft.Json" version="5.0.6" targetFramework="net45" />
  <package id="Respond" version="1.2.0" targetFramework="net45" />
  <package id="toastr" version="2.1.0" targetFramework="net45" />
  <package id="WebGrease" version="1.5.2" targetFramework="net45" />
</packages>
```

## Important Tools:-

Below I have pasted my tools of choice which I usually use for building any application. However, it's entirely your choice that which one you are going to choose.

- Visual Studio 2010/12/13 - IDE
- Re-Sharper – Gives much better Intellisense out of the box on the top of Visual Studio.
- Notepad++ or Sublime for any outside editing.
- http://www.responsinator.com/
- http://www.electricplum.com/
- http://www.opera.com/developer/mobile-emulator
- Chrome Mobile Emulator.

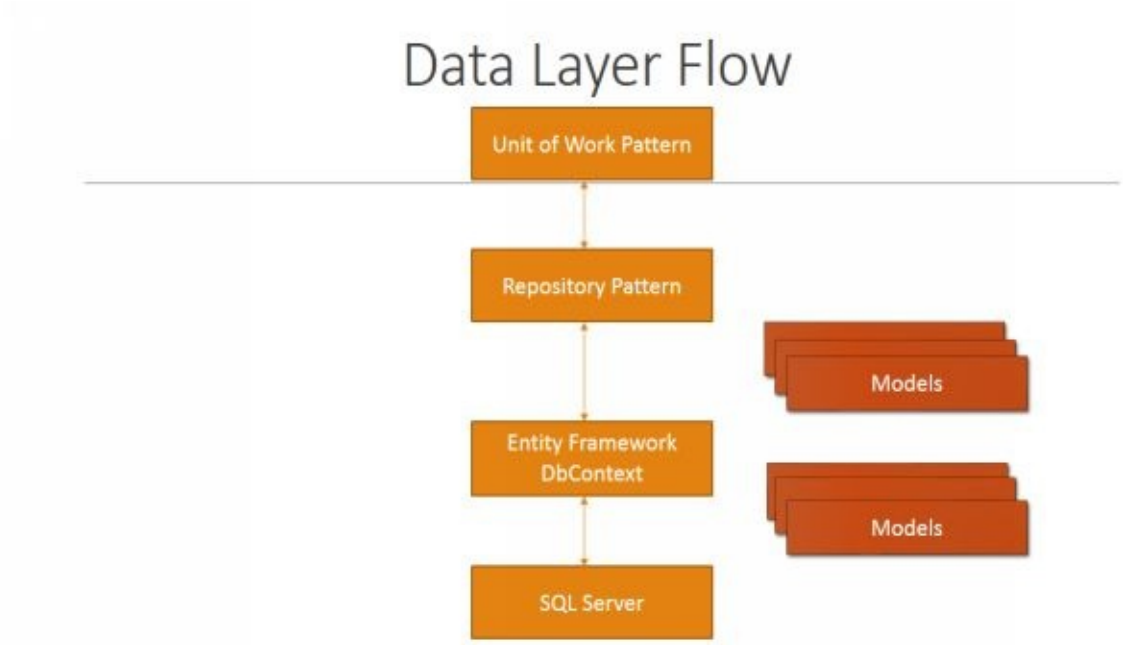
## Data Technologies:-

- Data Storage: - In this context we are going to setup our database. Here, I'll be using **SQL Server** for persisting the data. However, there is no such restriction of using SQL Server only, you may choose any other database of your choice like Oracle or any other.

- Object Relational Mapper: - Now, in order to pull the records from the database and persist the same back to database I used **Entity Framework** as my **ORM**. Here, data is stored of fetched via its own **DBContext**.

- Repository Pattern: - Repositories basically expose the ORM's data for saving and retrieving the data to the context and then it is taken by our database.

- Unit of Work Pattern: - **UOW** is again one more important piece which takes our changes and then issues save/cancel request to the context. Basically, it aggregates the repositories so that we can commit or cancel multiple changes to the repositories.



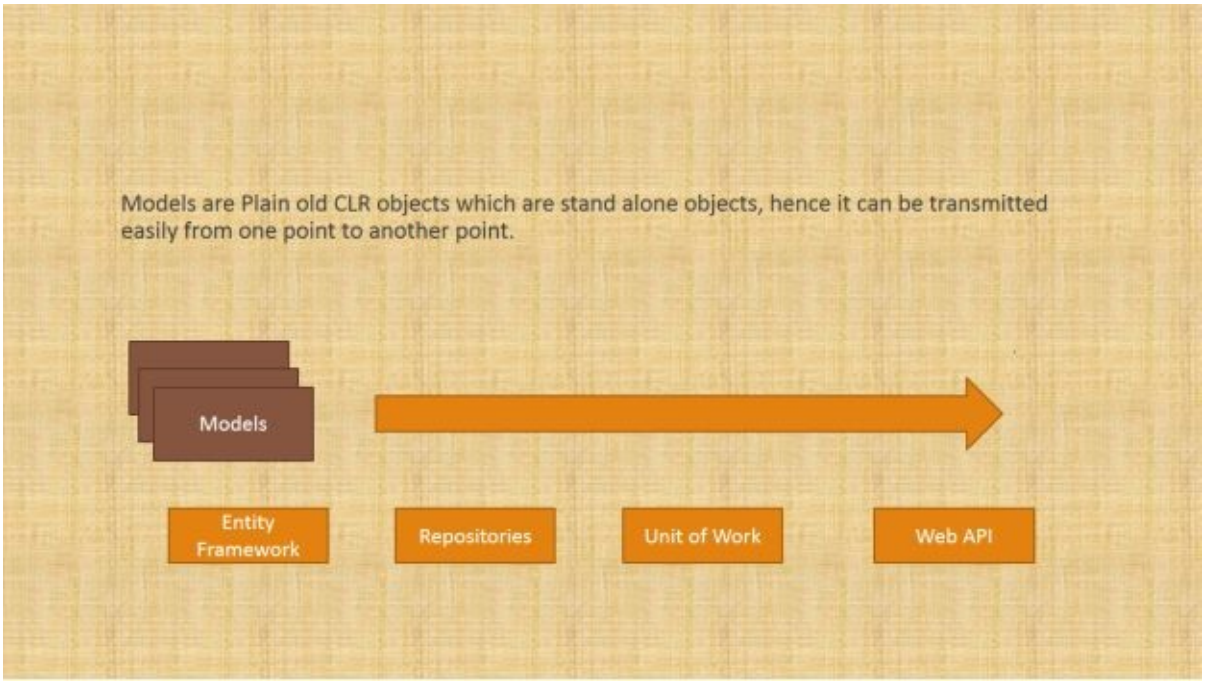

Above I have pasted the data layer flow. This is going to be my basic architecture like how data layer is going to interact with my User Interface. Here, we are going to interact with our SQL Server using Entity Framework DbContext using models. So, EF is going to map let's say **Movie** Plain Old Class Object (POCO) with Movie table back in the database. Now, we can pass these models with Repository Pattern. This extra layer of abstraction **Repository Pattern** allows us to simplify the call to the database. So, all my calls to the database follow a very specific syntax.

Next comes is the **Unit of Work Pattern**. The job of the **UOW** is to take the data from multiple places and put the same in one place. Let's suppose a scenario where in I have to fetch movie which has received highest no of reviews. So, in order to resolve this kind of scenario UOW will come into picture. Here, UOW will get the data from different repositories and pass the same down the layer. Also, another job of the UOW is to commit to the database which we will see in a moment.

## Creating Models:-

Models are basically data containers which carries data. Models are also properties

which define our data. Models are often termed as entities as well. So, these models are like carriers for our data and EF will fill these models.

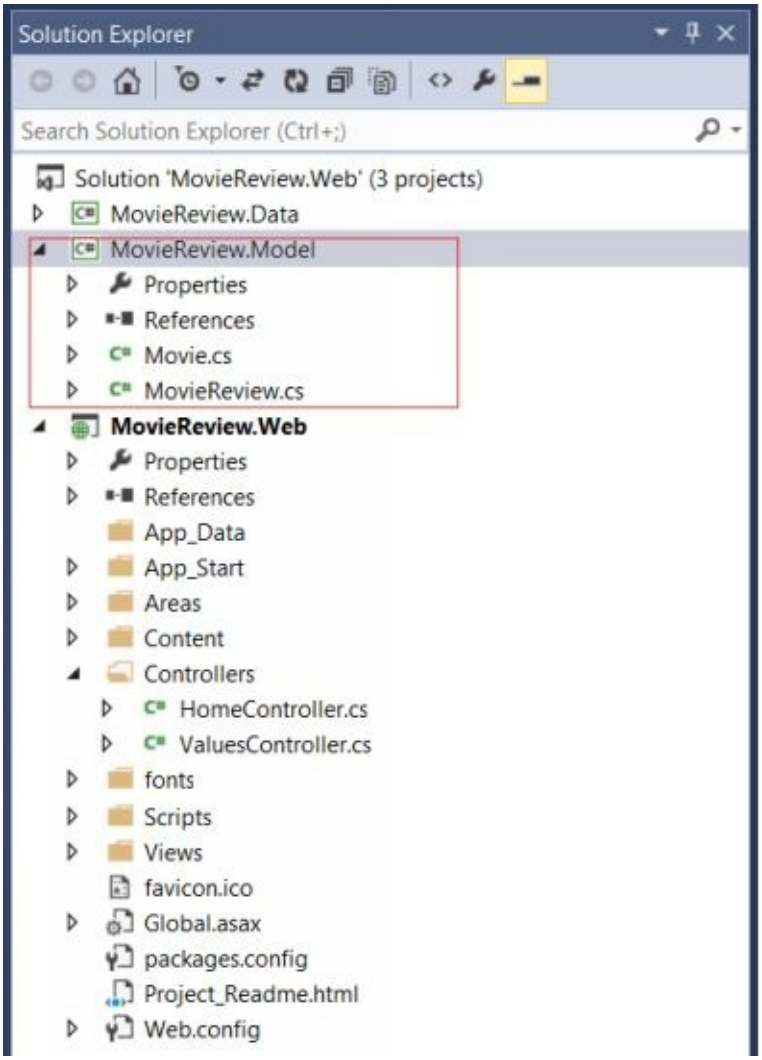



Here, I'll be creating two model classes as **Movie** and **MovieReview** as shown below in the snippet.

```
using System.Collections.Generic;

namespace MovieReview.Model
{
    public class Movie
    {
    public int Id { get; set; }
    public string MovieName { get; set; }
    public string DirectorName { get; set; }
    public string ReleaseYear { get; set; }
    public virtual ICollection<MoviesReview> Reviews { get; set; }
    }
}
```

```
namespace MovieReview.Model
{
    public class MoviesReview
    {
    public int Id { get; set; }
    public string ReviewerName { get; set; }
    public string ReviewerComments { get; set; }
    public int ReviewerRating { get; set; }
    public int MovieId { get; set; }
    }
}
```

Once the model classes added then my model project will look like as shown below in the screen shot.




## Creating Entity Framework:-

Here, we will be creating our DbContext class. Now, the role of DbContext class is to create relations between Models and Database. The benefit of using DbContext is it stores objects and changes in its memory. So, here I'll be using Entity Framework for doing all

CRUD operations. In order to deep dive in Entity Framework with all different varieties of styles you can check my blog on the same http://myview.rahulnivi.net and then check Entity Framework thread. However, let me just give you a brief snapshot on the same.

**Schema First Approach**:- In this approach what we do, we open a graphical designer in Visual Studio where in we are pointing the EF to an existing database so that we can import the database schema. Now, this will import the entire schema and generate the classes which we need to query and update the database back.

**Model First Approach**: - In this approach I use the same graphical designer inside the visual studio, to draw an overall model of my application. And then EF will generate both my class definitions and my database schema.

**Code First Approach**: - Last but not the least is **Code First Approach**. This one is my personal favorite. In this case, I just write C# classes and then EF use the class definitions to create the database for me. And for doing the same, EF follows certain conventions like Naming Conventions. We can also provide explicit mappings wherein I can change the mappings if I don't like the default one. Hence, in this context I'll be using Code First Approach to begin with my database development.

## Creating Entity Framework DbContext Class:-

Now, let's go ahead and create our DbContext class. Below is the snippet of the same

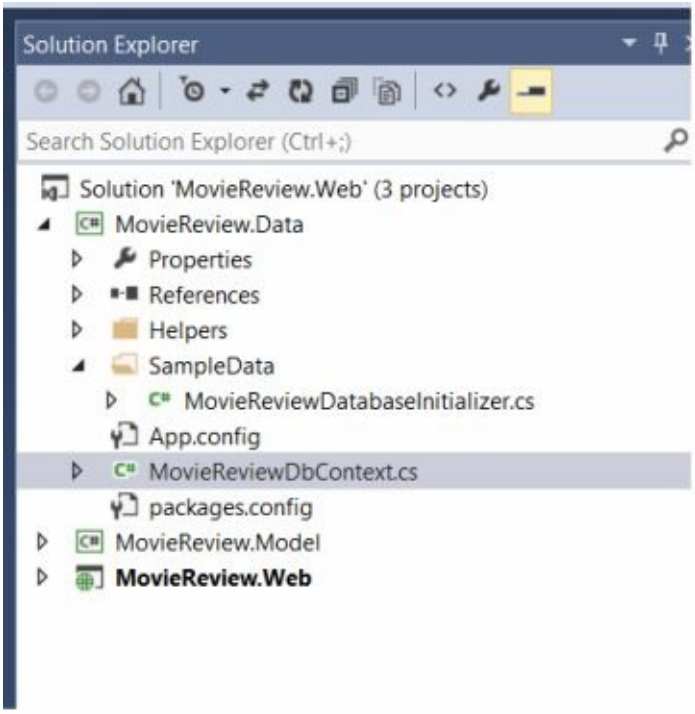
```
using MovieReview.Data.SampleData;
using MovieReview.Model;
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MovieReview.Data
{
    public class MovieReviewDbContext :DbContext
    {

    public MovieReviewDbContext() : base(nameOrConnectionString: "MoviesReviewProd") { }

    public DbSet<Movie> Movies { get; set; }
```

```
    public DbSet<MoviesReview> MovieReviews { get; set; }

    //invoke this to seed default values for the 1st run
    //comment the intializer code in production
    static MovieReviewDbContext()
    {
    Database.SetInitializer(new MovieReviewDatabaseInitializer());
    }

    //setting EF Convetions
    protected override void OnModelCreating(DbModelBuilder modelBuilder)
    {
    //use singular table names
    modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
    base.OnModelCreating(modelBuilder);
    }
    }
}
```

Here, I have done couple of things, the very first thing is I have assigned my connection string name in the constructor piece then I have setup the DBSet for the POCO classes which I have created earlier. In the next phase I have one static method which is nothing but data initializer for the 1st run. This piece is crucial for the time when database is not created initially. Hence, I strongly recommend don't push this piece of code as is in production otherwise it will overwrite your existing data.

However, this piece is good for testing the app at initial phase. I'll walk you through **DBInitializer** class in some time. But, for now let's focus on the next piece. Next, thing is setting some Entity Framework conventions. Here, it will restrict DB from pluralizing the table names.

Now, let's have a glimpse of my Data Solution.




## Database Initializer:-

In the last segment we had a glimpse of Database Initializer. Let's look inside and understand what this piece is doing. Below, I have pasted the snippet of the same.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Migrations;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using MovieReview.Model;

namespace MovieReview.Data.SampleData
{
 public class MovieReviewDatabaseInitializer : DropCreateDatabaseIfModelChanges<MovieReviewDbContext>
 {
    protected override void Seed(MovieReviewDbContext context)
    {
    context.Movies.AddOrUpdate(r => r.MovieName,
    new Movie { MovieName = "Avatar", DirectorName = "James Cameron", ReleaseYear = "2009" },
    new Movie { MovieName = "Titanic", DirectorName = "James Cameron", ReleaseYear = "1997" },
    new Movie { MovieName = "Die Another Day", DirectorName = "Lee Tamahori", ReleaseYear = "2002" },
    new Movie
    {
    MovieName = "Godzilla",
```

```
    DirectorName = “Gareth Edwards”,
    ReleaseYear = “2014”,
    Reviews = new List<MoviesReview>{
    new MoviesReview{ReviewerRating=5,ReviewerComments=“Excellent”,ReviewerName=“Rahul Sahay”}
    }
    });

    }
    }
}
```

So, what this piece will do, it will certainly drop the database and create again if model changes. After that I have code for seed data. It’s pretty straight forward, 1st I added three movies and 4th one with review as well.

## Implementing Repository Pattern:-

Before implementing repository pattern, let’s understand why use Repository Pattern? That’s a great question. To answer this I’ll list out following points as listed below:-

- Code Maintainability: - It’s very easy to use data access code for debugging or for making any changes.

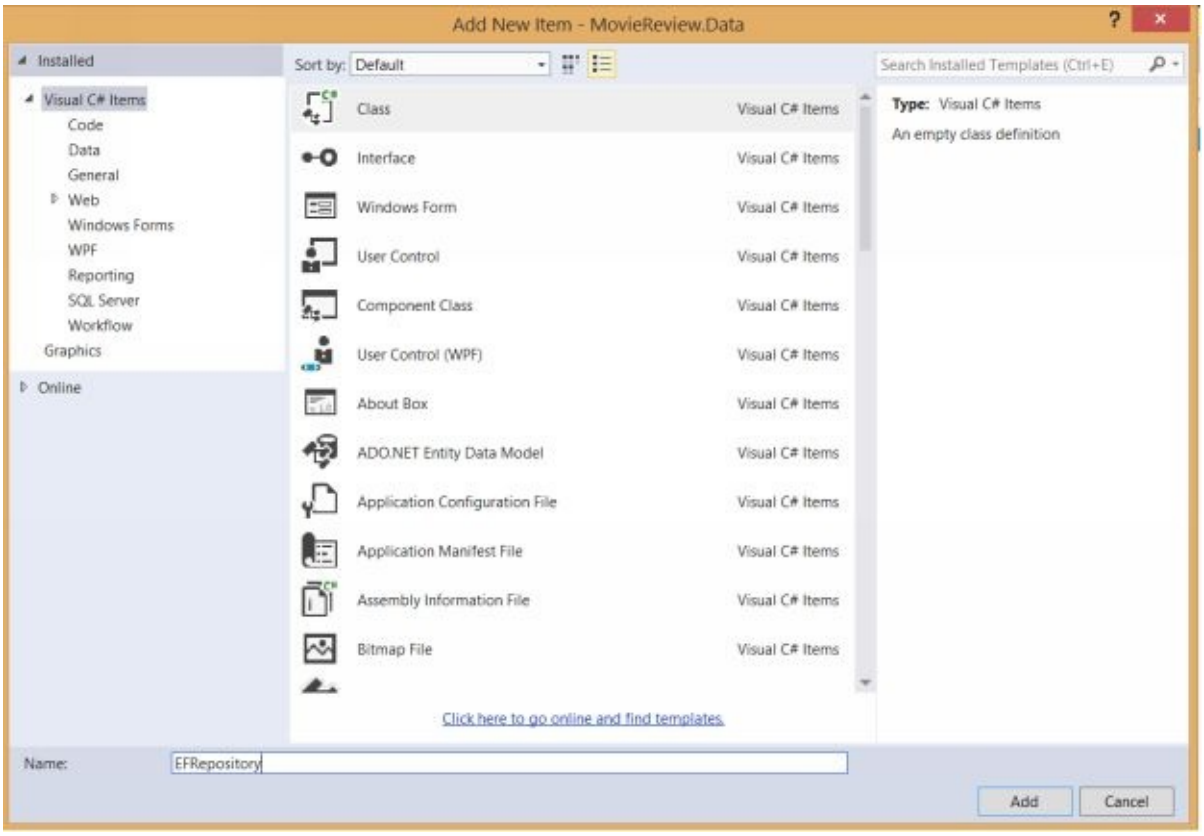
- Code Reuse: - Let’s suppose I am directly talking to my dbcontext via controllers and I have 20 odd controllers. So, every time I make any data access code change, I have to change 20 different places. Hence, rather wasting time on this I’ll have one repository which will take care of data access job. It also raises couple of more points to list here…

- Focused on CRUD operation:-

- Consistent APIs which can be unit tested as well.

- Focused on Single Repository Principle (SRP):- means every class or API will have individual role.

With the above explanations let’s go ahead and create my Entity Framework repository which will serve as a base for all db interactions.




Once the class created successfully, I'll go ahead and paste some code in there to give it final shape. Below, is the **EFRepository** class in finished form.

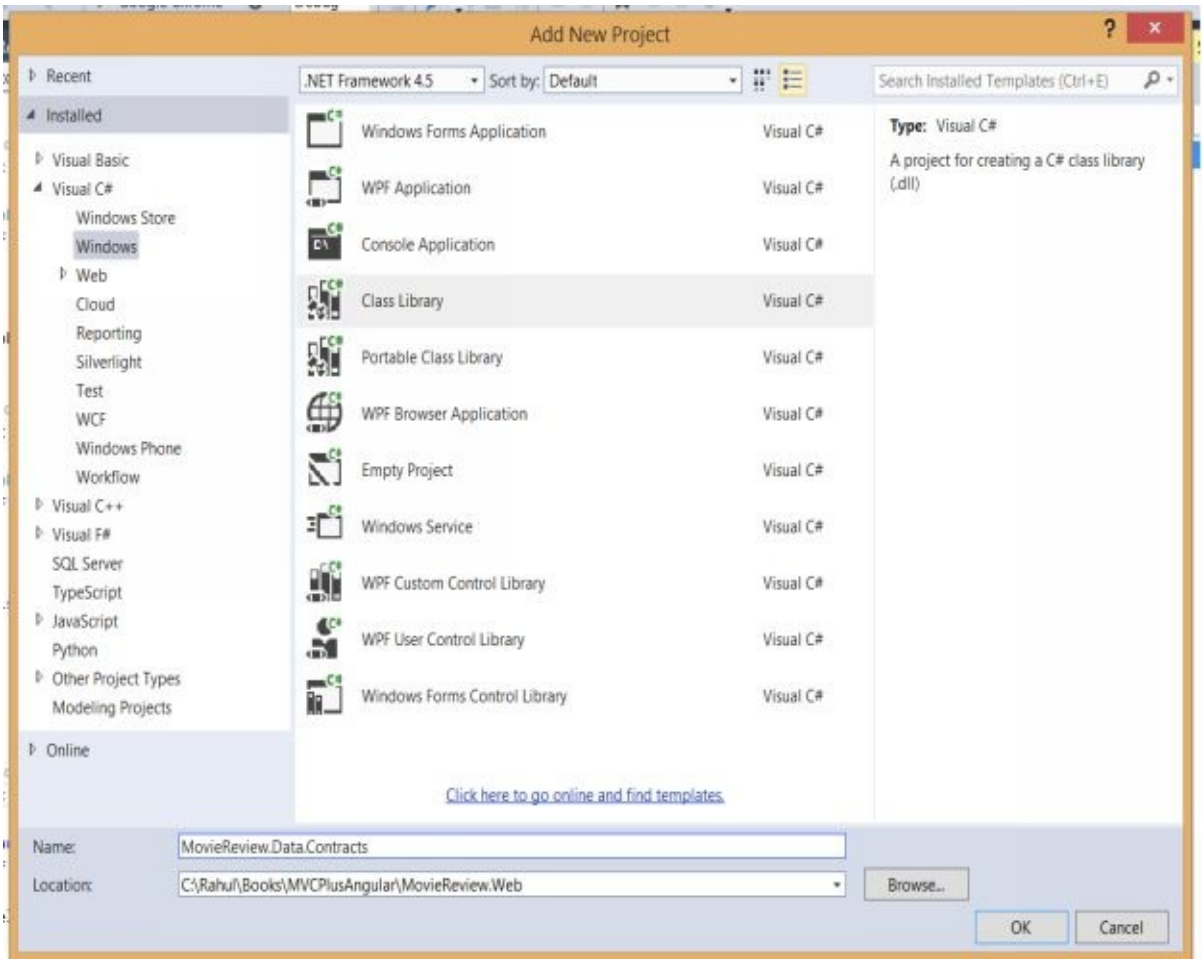
```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MovieReview.Data
{
    public class EFRepository<T> : IRepository<T> where T : class
    {
    public EFRepository(DbContext dbContext)
    {
    if (dbContext == null)
    throw new ArgumentNullException("dbContext");
    DbContext = dbContext;
    DbSet = DbContext.Set<T>();
    }
```

```
protected DbContext DbContext { get; set; }
protected DbSet<T> DbSet { get; set; }
public virtual IQueryable<T> GetAll()
{
return DbSet;
}

public virtual T GetById(int id)
{
return DbSet.Find(id);
}

public virtual void Add(T entity)
{
DbEntityEntry dbEntityEntry = DbContext.Entry(entity);
if (dbEntityEntry.State != EntityState.Detached)
{
dbEntityEntry.State = EntityState.Added;
}
else
{
DbSet.Add(entity);
}
}

public virtual void Update(T entity)
{
DbEntityEntry dbEntityEntry = DbContext.Entry(entity);
if (dbEntityEntry.State != EntityState.Detached)
{
DbSet.Attach(entity);
}
dbEntityEntry.State = EntityState.Modified;
}

public void Delete(T entity)
{
DbEntityEntry dbEntityEntry = DbContext.Entry(entity);
if (dbEntityEntry.State != EntityState.Deleted)
{
dbEntityEntry.State = EntityState.Deleted;
```

```
    }
    else
    {
    DbSet.Attach(entity);
    DbSet.Remove(entity);
    }
    }

    public void Delete(int id)
    {
    var entity = GetById(id);
    if (entity == null) return;

    Delete(entity);
    }
    }
}
```

As you can see in the above snippet, that my **EFRepository** is dependent on **IRepository** which I'll be creating in a moment. But, let's understand what this guy is doing here. Here, it is just taking instructions from **UOW** (Unit of work pattern) say I would like to create new movie, then in that case it will consider **T** as **Movie** and then it will 1st check whether dbcontext is created or not if not then it will go ahead and create the db context and then implement the required CRUD operation based on the API call. Hence, a lot of code reuse, makes sense right .

Now, let's go ahead and create **IRepository** interface. But, before creating this class I would like to put the same in a different project. You can go ahead and place these interfaces in the same models folder but I really like to segregate the same by keeping them apart. Idea is not to mix your interfaces with concrete classes.

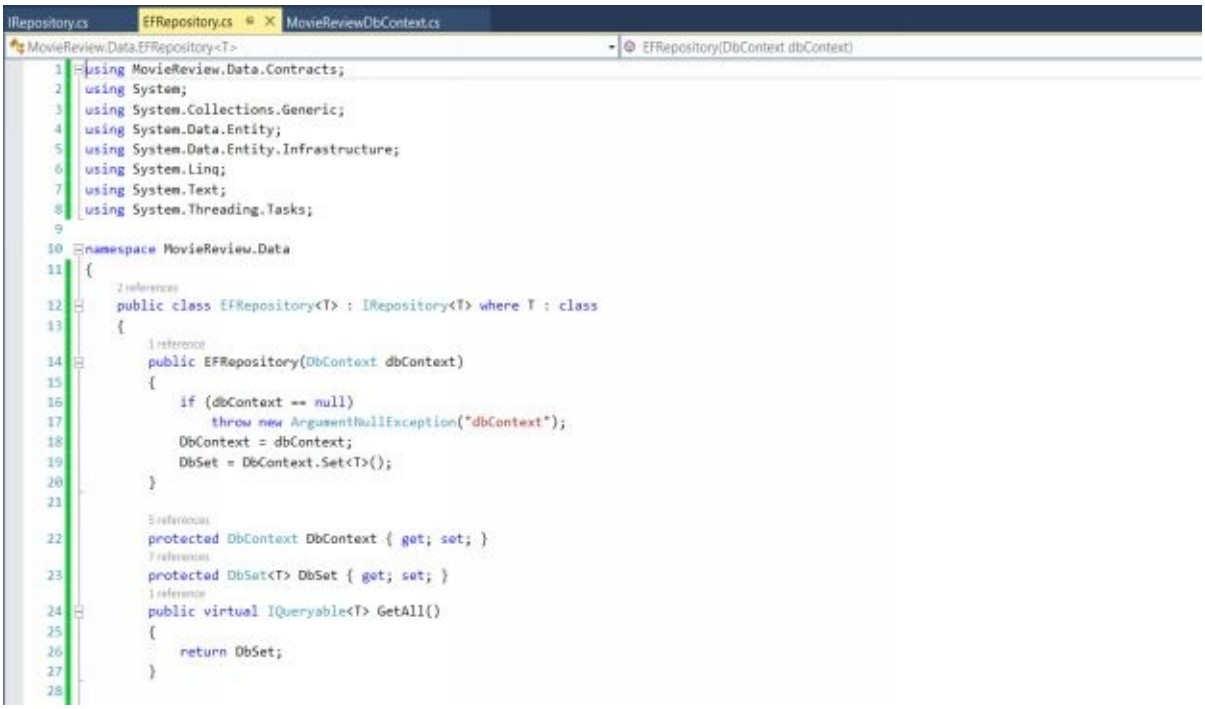



Once, the project got created successfully. I will go ahead and create my required interface as shown below

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MovieReview.Data.Contracts
{
    public interface IRepository<T> where T : class
    {
    //To query using LINQ
    IQueryable<T> GetAll();

    //Returning Movie or Review by id
    T GetById(int id);

    //Adding Movie or Review
    void Add(T entity);
```

```
    //Updating Movie or Review
    void Update(T entity);

    //Deleting Moovie or Review
    void Delete(T entity);

    //Deleting Movie or Review by id
    void Delete(int id);
    }
}
```

So, let's suppose I would like to query Movie table, then in that case **IRepository** of T will replace T with **Movie** and perform any of the invoked operations. This way you can build a very clean and maintainable piece of Data access layer. All right, now the next thing is referencing this project in my data and web project. Once, the project reference is done, then I can go ahead and resolve my required dependency, then it will look like as shown below:-




## Creating Unit Of Work Pattern (UOW):-

In the previous sections, we looked how to create the repository pattern to talk to the Entity Framework DBContext. In this section we'll focus on **Unit of Work Pattern** and see how it will be used to interact with repositories and DBContext to expose our models to our Web API controllers so that the same can be returned to our client in the form of **JSON**. UOW, aggregates all data for our repository pattern. Now, again question comes

why this additional piece of layer required? And to answer this I'll again list out few basic points here…

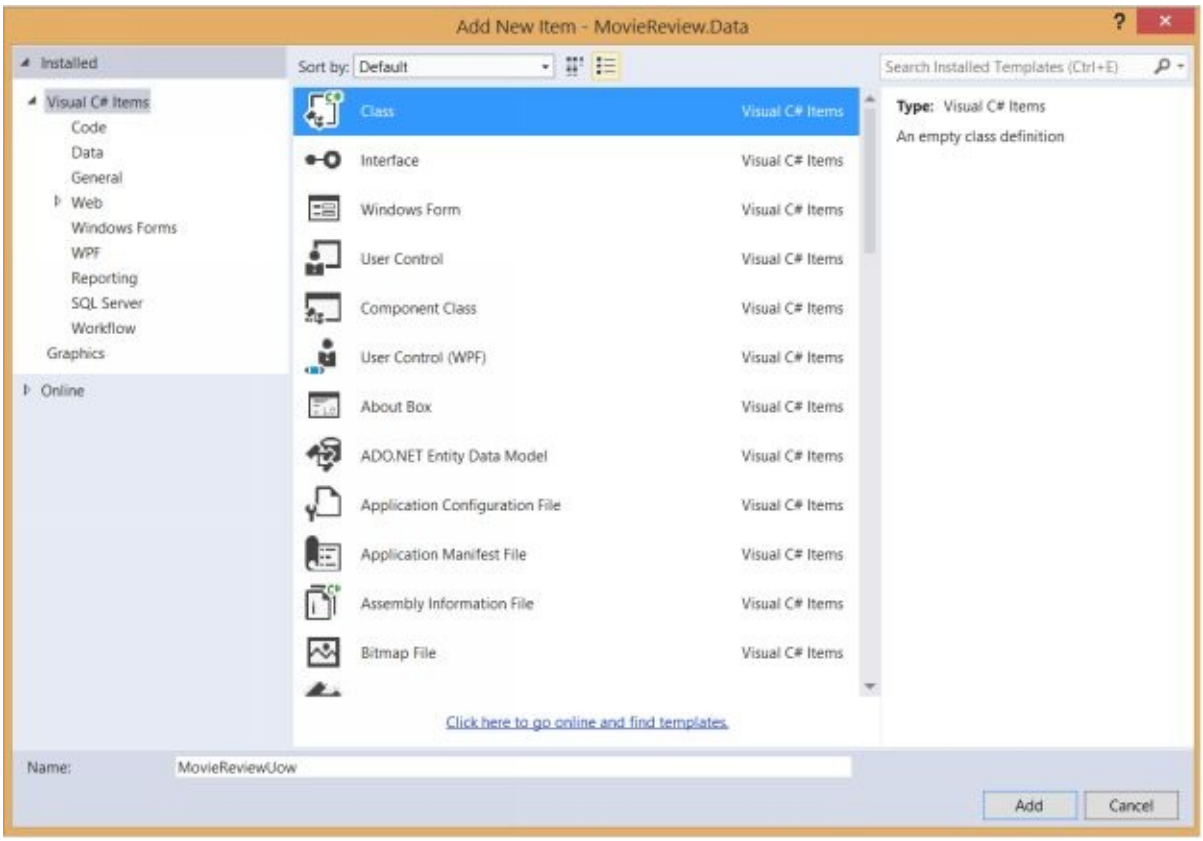
- Fits all tiny pieces together: - Decouples Web API controllers from the Repositories and DBContext.
- Aggregates all calls to Repositories.
- Simple Interface basically implementing IRepository<T> and commit action.

Now, with above explanations, let me go ahead and create my UOW pattern interface 1st in my Data Contract project. Below is the snippet of the same.

```
using MovieReview.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MovieReview.Data.Contracts
{
    /// <summary>
    /// Interface for UOW Movie Review
    /// </summary>
    public interface IMovieReviewUow
    {
    void Commit();
    IRepository<Movie> Movies { get; }
    IRepository<MoviesReview> MovieReviews { get; }



    }
}
```

As you can see in the above snippet, the code is pretty simple and straight forward. It basically aggregates my all repositories, in this case I only have two.

Now, to implement the same I need to create a class in my data project as shown below.




Below is the snippet for the same.

```
using MovieReview.Data.Contracts;
using MovieReview.Model;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace MovieReview.Data
{
    /// <summary>
    /// The Movie review “Unit of Work”
    ///     1) decouples the repos from the controllers
    ///     2) decouples the DbContext and EF from the controllers
    ///     3) manages the UOW
    /// </summary>
    /// <remarks>
    /// This class implements the “Unit of Work” pattern in which
    /// the “UoW” serves as a facade for querying and saving to the database.
    /// Querying is delegated to “repositories”.
```

```
/// </remarks>
///
public class MovieReviewUow : IMovieReviewUow, IDisposable
{


public MovieReviewUow(IRepositoryProvider repositoryProvider)
{
CreateDbContext();
repositoryProvider.DbContext = DbContext;
RepositoryProvider = repositoryProvider;
}

public IRepository<Movie> Movies { get { return GetStandardRepo<Movie>(); } }
public IRepository<MoviesReview> MovieReviews { get { return GetStandardRepo<MoviesReview>(); } }


public void Commit()
{
DbContext.SaveChanges();
}
protected void CreateDbContext()
{
DbContext = new MovieReviewDbContext();

//Do Not enable proxy entities
DbContext.Configuration.ProxyCreationEnabled = false;

//Load navigation property explicitly
DbContext.Configuration.LazyLoadingEnabled = false;

DbContext.Configuration.ValidateOnSaveEnabled = false;
}


protected IRepositoryProvider RepositoryProvider { get; set; }


private IRepository<T> GetStandardRepo<T>() where T : class
{
return RepositoryProvider.GetRepositoryForEntityType<T>();
}
```

```
    private T GetRepo<T>() where T : class
    {
    return RepositoryProvider.GetRepository<T>();
    }
    private MovieReviewDbContext DbContext { get; set; }

    public void Dispose()
    {
    Dispose(true);
    GC.SuppressFinalize(this);
    }
    protected virtual void Dispose(bool disposing)
    {
    if (disposing)
    {
    if (DbContext != null)
    {
    DbContext.Dispose();
    }
    }
    }
    }
}
```

Now, let me explain this piece of code here. **MovieReviewUOW** 1st creates the DbContext. There, I have mentioned that how I want my DbContext to look and behave. Since now we got our DbContext in place, we can go ahead and implement our repositories easily. So, now let's look at example how our repositories created, here **IRepository<Movie>** calls method **GetStandardRepo<Movie>** which in return going to call the Repository Provider. Now, Repository Provider calls **GetRepositoryForEntityType<T>**. Here, T will get replaced by Movie.

**GetRepositoryForEntityType<T>** is a method which is in the file RepositoryProvider under the helper folder in Data project. Now, this method helps in creating the repository for given entity type, so it can be Movie or MovieReview anything. So, basically this method 1st checks whether the instance is created or not, if not then it will create one and return the same. This piece of code is basically following Factory Pattern which does nothing but creates a repository whenever invoked. The best thing is we can easily go ahead and add repository here if any in future if we want. So, this way is really scalable and maintainable. You can refer the Helper folder code and all other pieces in the downloaded project.

Next, thing is commit method here. Commit method is saving changes back to the database. So, until this **SaveChanges()** is called it is not going to commit the changes back to database, it will remain in memory itself.

## Summary:-

In this module we saw how to get started with Repository Pattern. How to setup the same right from the scratch, then how to use Unit of Work Pattern to aggregate different repositories. We have also seen generic methodologies using Factory Pattern and how to use the same. In the next module we'll go one step ahead and Implement Web API and Dependency Injection.

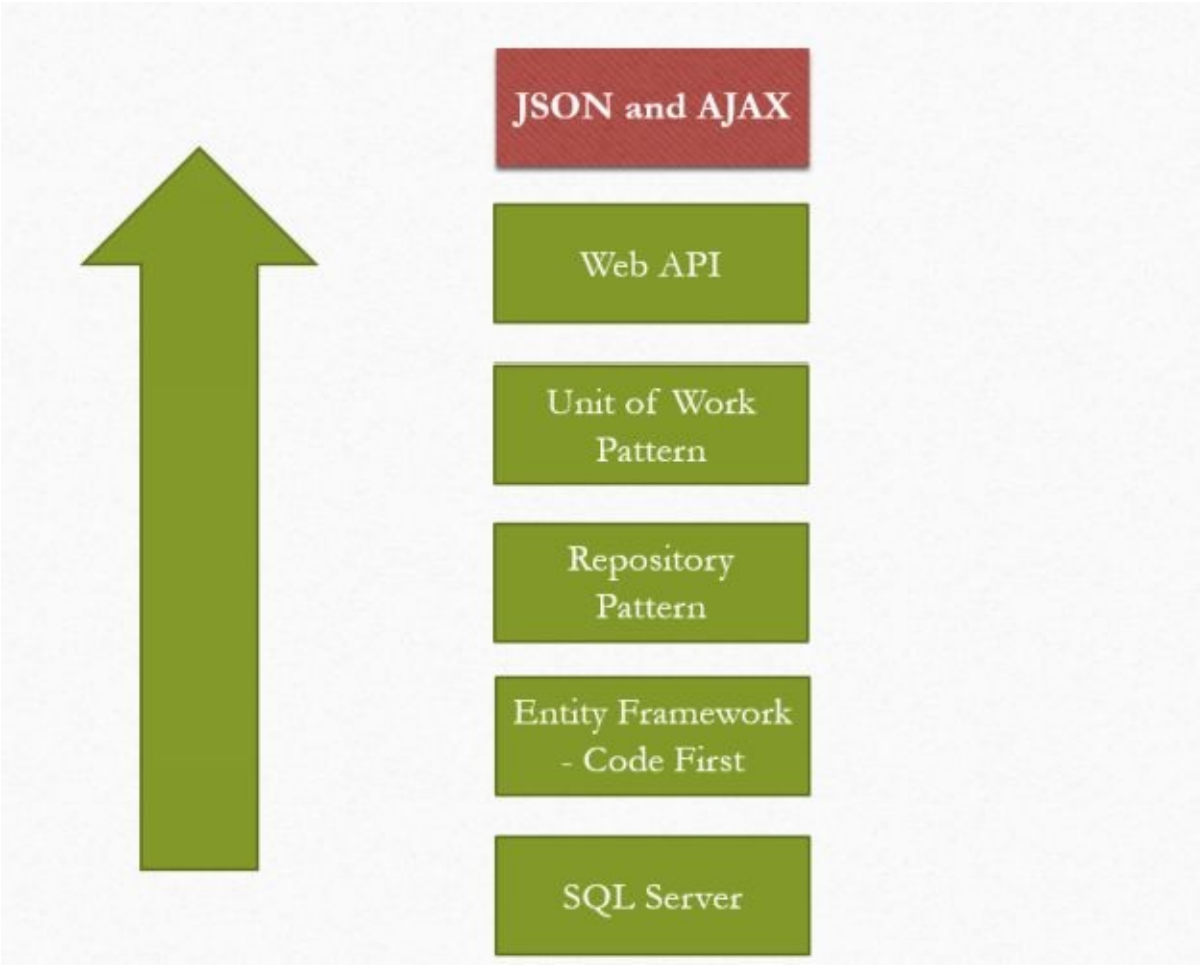
# Chapter 3: Implementing Web API

## WHAT DO you find in this CHAPTER?

- Introduction
- Creating 1st Web API Controller
- Implementing HTTP Put Request
- Implementing HTTP Post Request
- Implementing HTTP Delete Request
- Improvising Web APIs
- Adding More Controllers
- Testing Web APIs with QUnit
- Summary

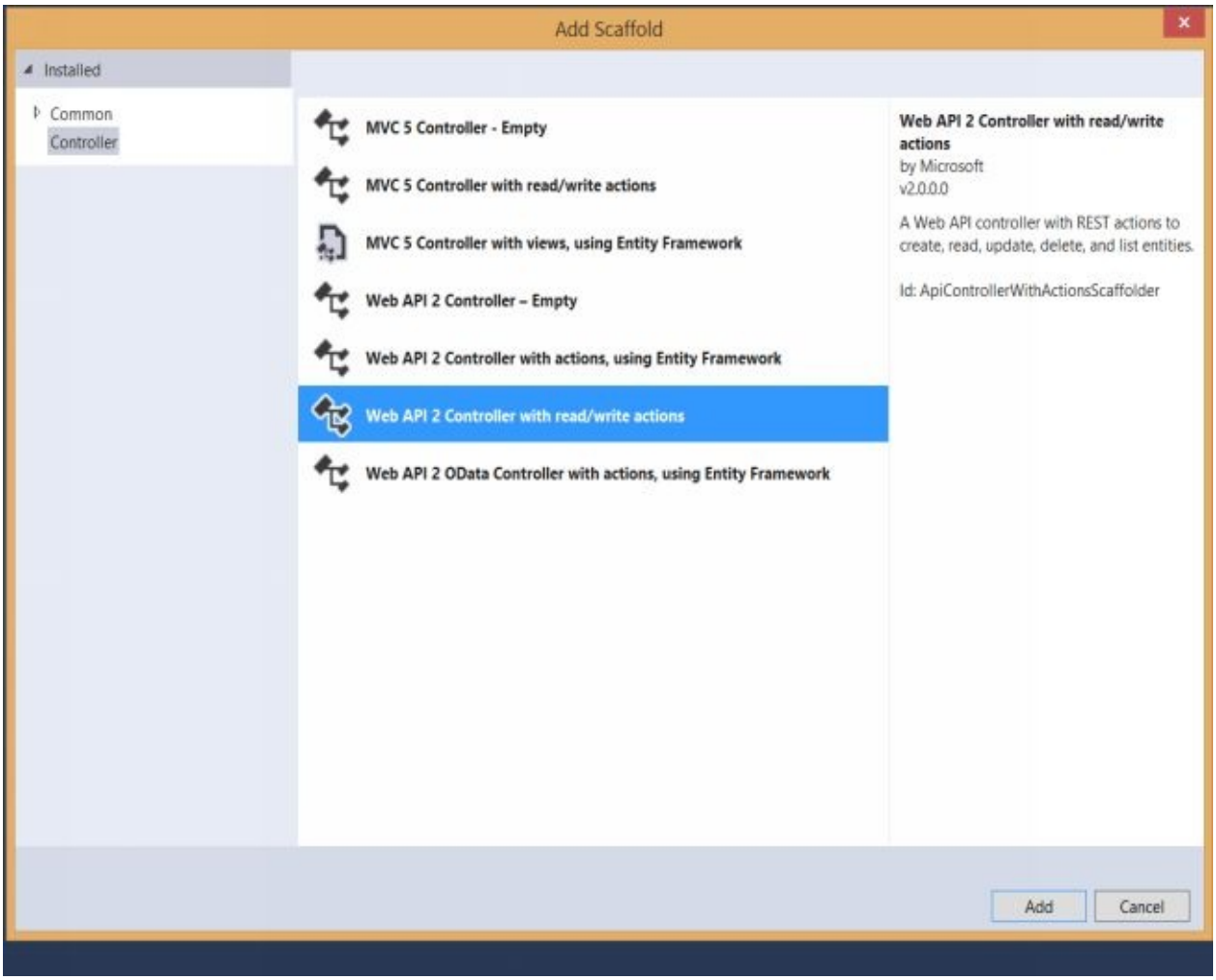
### Introduction:-

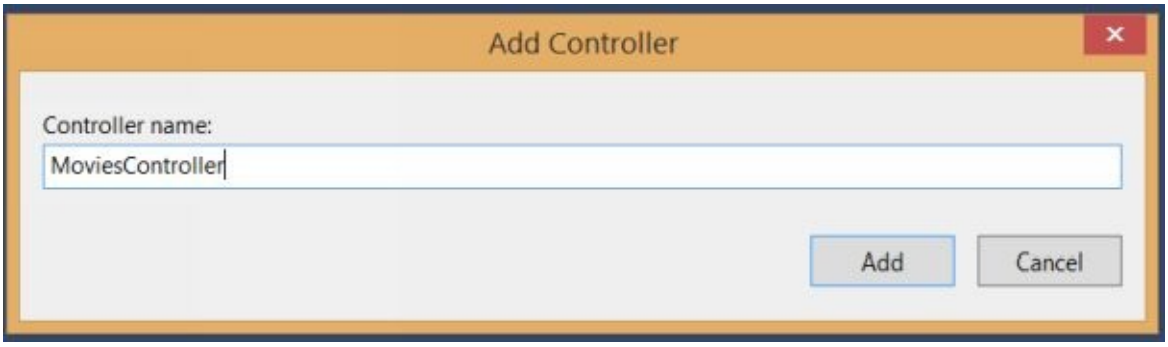
While building any application; passing the data to and fro from the client and saving the data back to the server is really important. So, here Web API is going to be the glue between our client and data layer. In this section, our main focus would be on the top two components as shown below in the diagram. Here, first we will be designing our Web APIs to talk to our repositories via UOW Pattern. Once, that piece is designed, then we'll see how we invoke our APIs. In this case it will be via AJAX. So, without wasting time, let's get started with our API Design.




## Creating 1st Web API Controller:-

I'll right click on the controller's folder and say add new controller with the below shown option.




Once, the controller is successfully created, it will present me the default template with all required HTTP operations as shown below in the snippet

```
using System.Collections.Generic;
using System.Web.Http;

namespace MovieReview.Web.Controllers
{
    public class MoviesController : ApiController
    {
    // GET api/movies
    public IEnumerable<string> Get()
```

```
    {
    return new string[] { “value1”, “value2” };
    }

    // GET api/movies/5
    public string Get(int id)
    {
    return “value”;
    }

    // POST api/movies
    public void Post([FromBody]string value)
    {
    }

    // PUT api/movies/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/movies/5
    public void Delete(int id)
    {
    }
    }
}
```

Now, I really like segregating my code here, hence I will create one abstract class which will inherit from and **ApiController** class. Basically, this will interact down the layers with my all repositories via **UOW(Unit of Work Pattern)** which we created previous chapter.

```
using System.Web.Http;
using MovieReview.Data.Contracts;

namespace MovieReview.Web.Controllers
{
    public class ApiBaseController : ApiController
    {
    protected IMovieReviewUow Uow { get; set; }
    }
}
```

Here, I have listed my interface which will go ahead and interact with my repositories. In this case it will be Movies and MovieReviews. Now, let's go ahead and modify our Movies Controller to interact with this base class.

```
using System.Collections.Generic;
using System.Web.Http;
using MovieReview.Data.Contracts;

namespace MovieReview.Web.Controllers
{
    public class MoviesController : ApiBaseController
    {
    public MoviesController(IMovieReviewUow uow)
    {
    Uow = uow;
    }
    // GET api/movies
    public IEnumerable<Movie> Get()
    {
    return Uow.Movies.GetAll().OrderBy(s => s.MovieName);
    }
    // GET api/movies/5
    public string Get(int id)
    {
    return "value";
    }

    // POST api/movies
    public void Post([FromBody]string value)
    {
    }

    // PUT api/movies/5
    public void Put(int id, [FromBody]string value)
    {
    }

    // DELETE api/movies/5
    public void Delete(int id)
    {
```

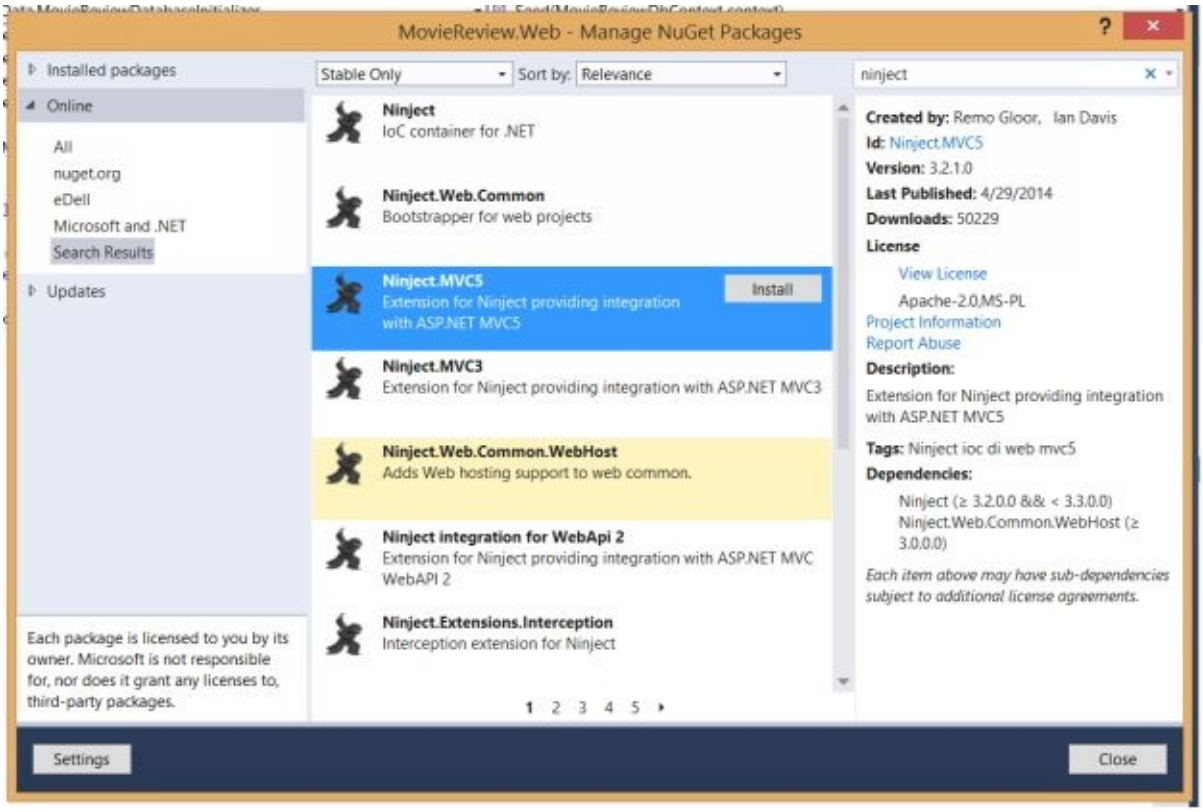
```
    }
    }
}
```

In the above piece of code I made three simple changes. 1st I made my class to inherit from base class which I just created, then in the constructor I assigned the required property with my interface one. I have also modified the 1st Get query to return all movies based on the Unit of Work Pattern.

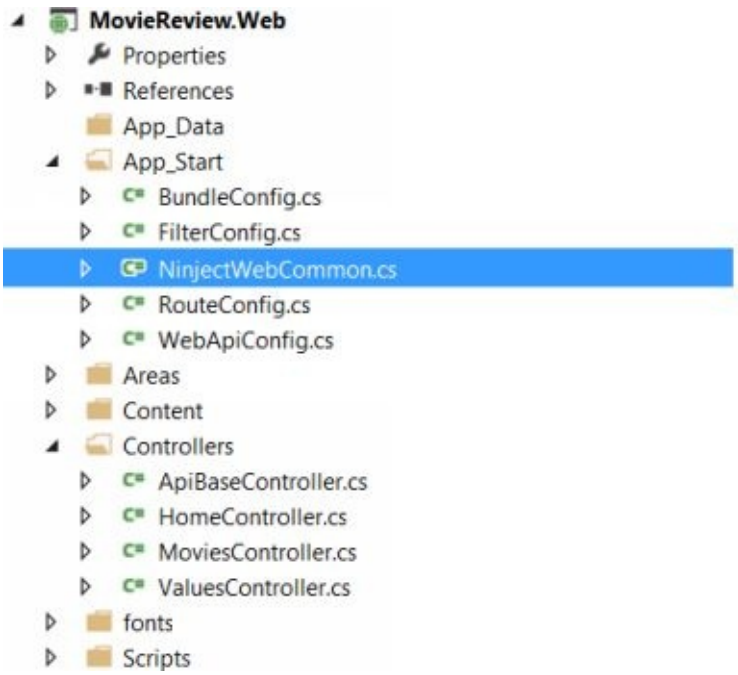
Now, in order to resolve this dependency, we need to implement Dependency Injection. For that I need to rely on one third party library **Ninject**. But, before that let's try and give it a shot whether it works or not. If not then what error it gives.

So, when I navigate to url http://localhost:65116/api/movies, it will present me the below error. Here, port no may vary, when you will be testing the same.


So, this is the typical **DI (Dependency Injection)** error. This happened because it tries to match the request with the concrete type but couldn't resolve the same. So, in order to fix the same, we need to install the below shown DI from Nuget.




Once, Ninject installed successfully, then we can see one new file "**NinjectWebCommon.cs**" which is basically a config file for registering services gets added in App_Start folder.




Below is the snippet from the same file.

```
using MovieReview.Data;
```

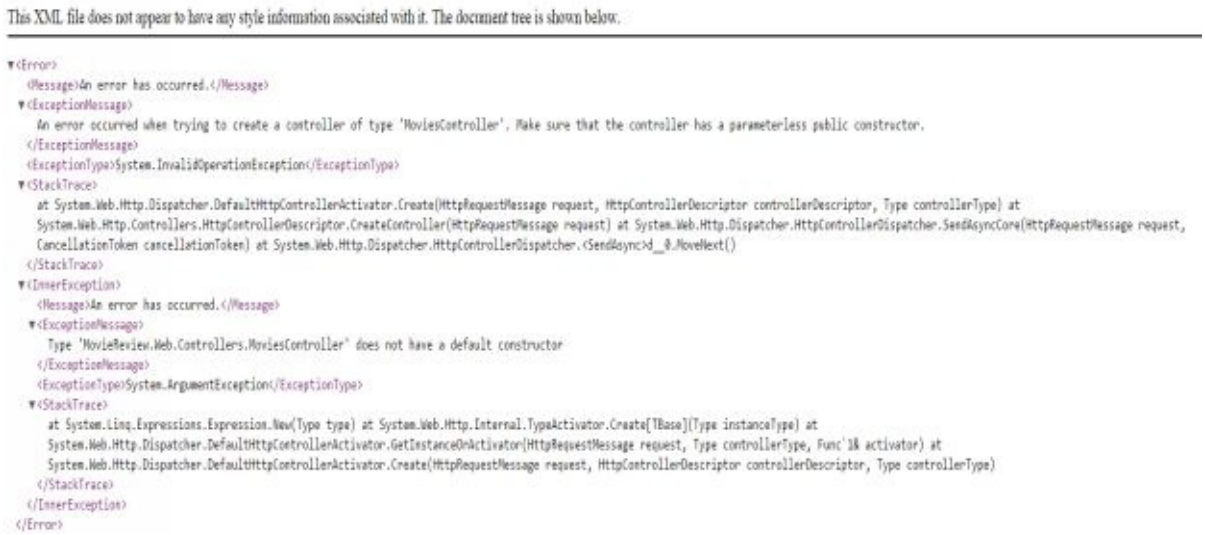
```
using MovieReview.Data.Contracts;

[assembly: WebActivatorEx.PreApplicationStartMethod(typeof(MovieReview.Web.App_Start.NinjectWebCommon), “Start”)]
[assembly: WebActivatorEx.ApplicationShutdownMethodAttribute(typeof(MovieReview.Web.App_Start.NinjectWebCommon), “Stop”)]

namespace MovieReview.Web.App_Start
{
    using System;
    using System.Web;

    using Microsoft.Web.Infrastructure.DynamicModuleHelper;

    using Ninject;
    using Ninject.Web.Common;

    public static class NinjectWebCommon
    {
    private static readonly Bootstrapper bootstrapper = new Bootstrapper();

    /// <summary>
    /// Starts the application
    /// </summary>
    public static void Start()
    {
    DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule));
    DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule));
    bootstrapper.Initialize(CreateKernel);
    }

    /// <summary>
    /// Stops the application.
    /// </summary>
    public static void Stop()
    {
    bootstrapper.ShutDown();
    }

    /// <summary>
    /// Creates the kernel that will manage your application.
    /// </summary>
```

```
    /// <returns>The created kernel.</returns>
    private static IKernel CreateKernel()
    {
    var kernel = new StandardKernel();
    try
    {
    kernel.Bind<Func<IKernel>>().ToMethod(ctx => () => new Bootstrapper().Kernel);
    kernel.Bind<IHttpModule>().To<HttpApplicationInitializationHttpModule>();

    RegisterServices(kernel);
    return kernel;
    }
    catch
    {
    kernel.Dispose();
    throw;
    }
    }

    /// <summary>
    /// Load your modules or register your services here!
    /// </summary>
    /// <param name="kernel">The kernel.</param>
    private static void RegisterServices(IKernel kernel)
    {
    kernel.Bind<RepositoryFactories>().To<RepositoryFactories>().InSingletonScope();


    kernel.Bind<IRepositoryProvider>().To<RepositoryProvider>();
    kernel.Bind<IMovieReviewUow>().To<MovieReviewUow>();
    }
    }
}
```

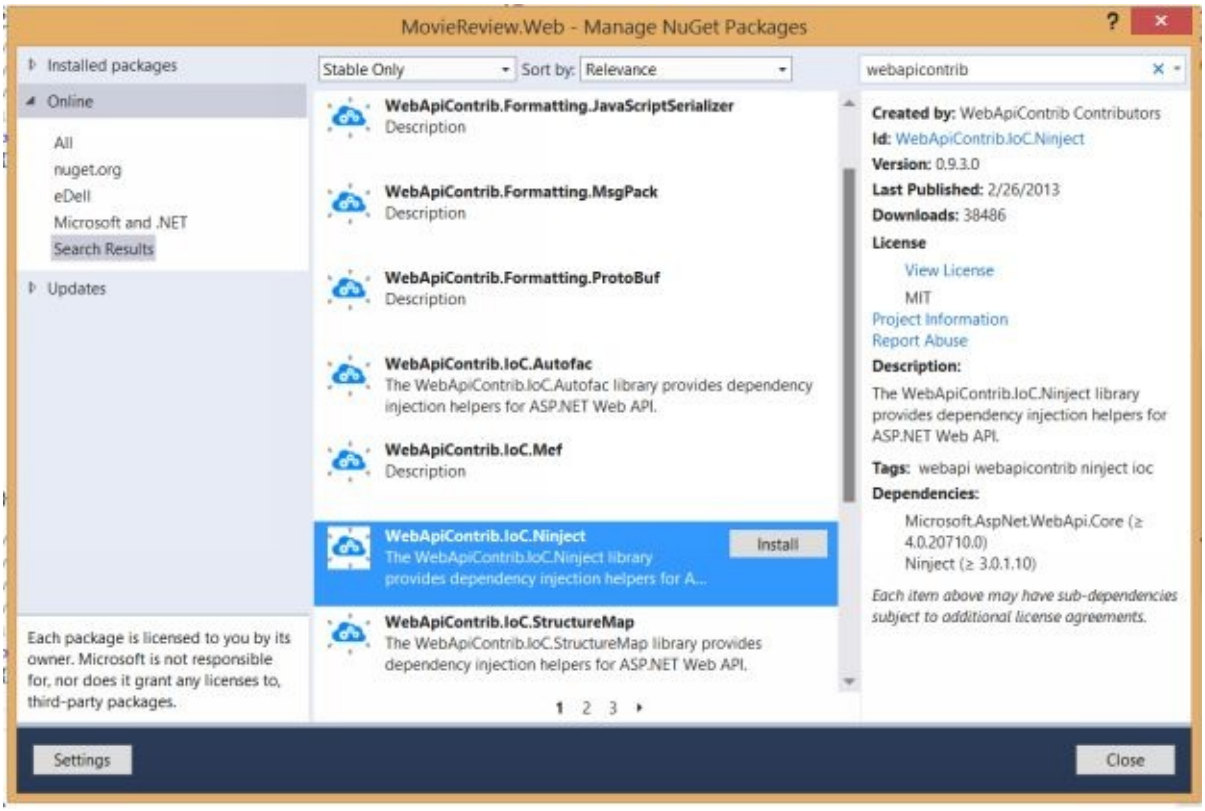
So, as you can see in the above snippet, I have mentioned all my dependencies under **RegisterServices** method. Now, one point to note here, if you want to understand Web API and DI in detail, you can refer my videos on my blog at

http://myview.rahulnivi.net/.Net,ASP.Net,C,EntityFramework,MVC/asp-net-web-api/.

With the above change in place if I go ahead and build the app, then I should expect some significant result. But it again broke as shown below in the screen shot.




But, why this happened. So, one thing to understand here, Ninject IOC doesn't support for API controller by default. If it would have been MVC controller rather than Web API controller, then it would have been resolved. However, in order to fix the same, we need to install one more package **WebApiContrib.IoC.Ninject**




Once, the package is successfully installed, then we need to tell the IOC code to resolve for Web API as well as shown below in the snippet.

```
using System.Web.Http;
using MovieReview.Data;
```

```
using MovieReview.Data.Contracts;
using WebApiContrib.IoC.Ninject;

[assembly: WebActivatorEx.PreApplicationStartMethod(typeof(MovieReview.Web.App_Start.NinjectWebCommon), “Start”)]
[assembly: WebActivatorEx.ApplicationShutdownMethodAttribute(typeof(MovieReview.Web.App_Start.NinjectWebCommon), “Stop”)]

namespace MovieReview.Web.App_Start
{
    using System;
    using System.Web;

    using Microsoft.Web.Infrastructure.DynamicModuleHelper;

    using Ninject;
    using Ninject.Web.Common;

    public static class NinjectWebCommon
    {
    private static readonly Bootstrapper bootstrapper = new Bootstrapper();

    /// <summary>
    /// Starts the application
    /// </summary>
    public static void Start()
    {
    DynamicModuleUtility.RegisterModule(typeof(OnePerRequestHttpModule));
    DynamicModuleUtility.RegisterModule(typeof(NinjectHttpModule));
    bootstrapper.Initialize(CreateKernel);
    }

    /// <summary>
    /// Stops the application.
    /// </summary>
    public static void Stop()
    {
    bootstrapper.ShutDown();
    }

    /// <summary>
    /// Creates the kernel that will manage your application.
```

```
    /// </summary>
    /// <returns>The created kernel.</returns>
    private static IKernel CreateKernel()
    {
    var kernel = new StandardKernel();
    try
    {
    kernel.Bind<Func<IKernel>>().ToMethod(ctx => () => new Bootstrapper().Kernel);
    kernel.Bind<IHttpModule>().To<HttpApplicationInitializationHttpModule>();

    //Web API Settings
    GlobalConfiguration.Configuration.DependencyResolver = new NinjectResolver(kernel);

    RegisterServices(kernel);
    return kernel;
    }
    catch
    {
    kernel.Dispose();
    throw;
    }
    }

    /// <summary>
    /// Load your modules or register your services here!
    /// </summary>
    /// <param name="kernel">The kernel.</param>
    private static void RegisterServices(IKernel kernel)
    {
    kernel.Bind<RepositoryFactories>().To<RepositoryFactories>().InSingletonScope();


    kernel.Bind<IRepositoryProvider>().To<RepositoryProvider>();
    kernel.Bind<IMovieReviewUow>().To<MovieReviewUow>();
    }
    }
}
```

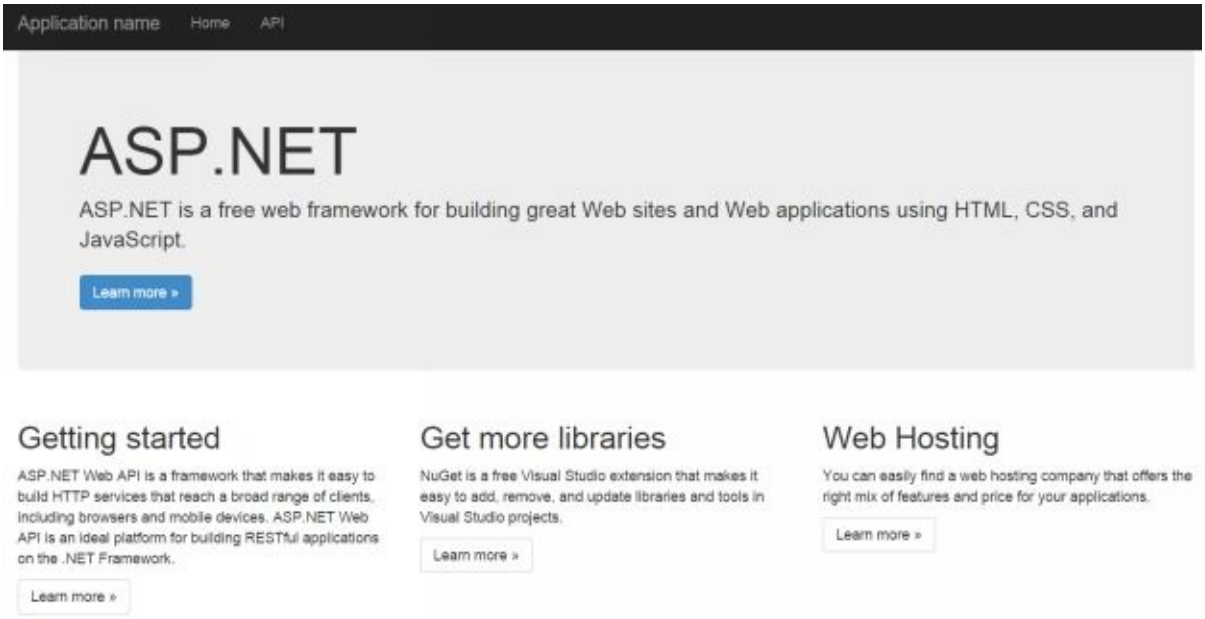
In the above snippet, in the CreateKernel section, I have specified configuration to resolve Web API as well.

Now, after installing the above API it may happen, then that your application crash due to assembly mismatch as shown in the below screen shot.


So, to fix the same, there is a work around which needs to be applied in the web.config as shown below in the snippet.

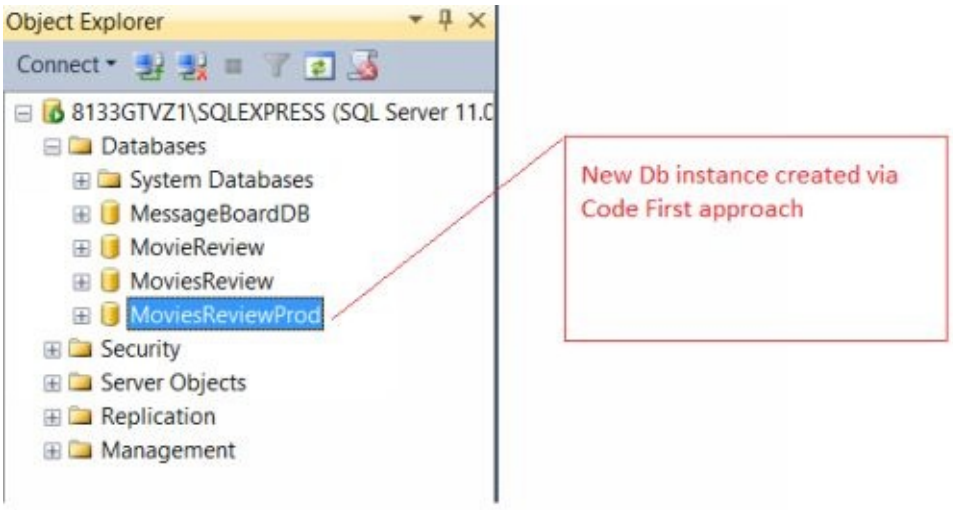
```
<dependentAssembly>
    <assemblyIdentity name="System.Web.Http" publicKeyToken="31bf3856ad364e35" culture="neutral" />
    <bindingRedirect oldVersion="0.0.0.0-5.0.0.0" newVersion="5.0.0.0" />
    </dependentAssembly>
```

This needs to be pasted in the runtime section of the dependent assemblies of web.config file. Once, this is done. Then, application will load properly as shown below in the screen shot.



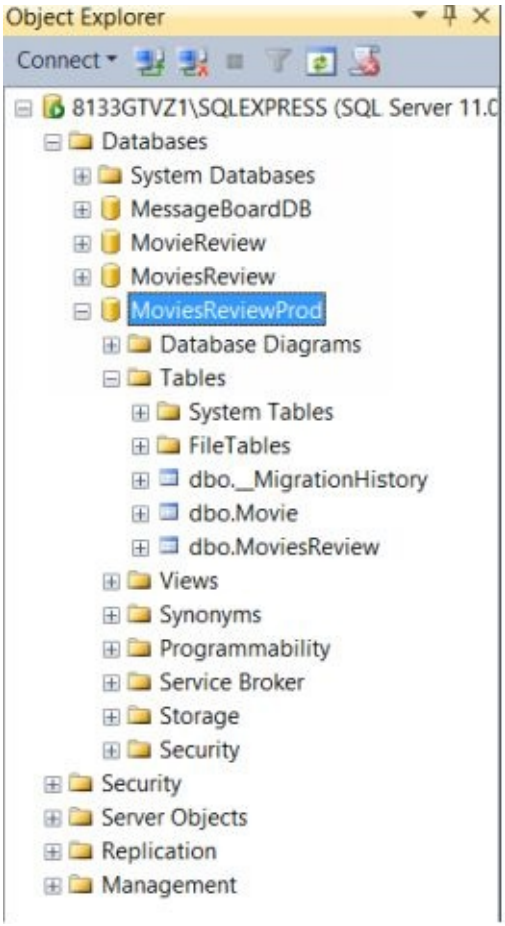

Now, let's try to navigate the API URL [http://localhost:65116/api/movies](http://localhost:65116/api/movies)


Here, it not only resolved the concrete types rather my seed data also got injected via DbInitializer code. So, if I go ahead and refer my database in Sql Server, it will present me the following things.

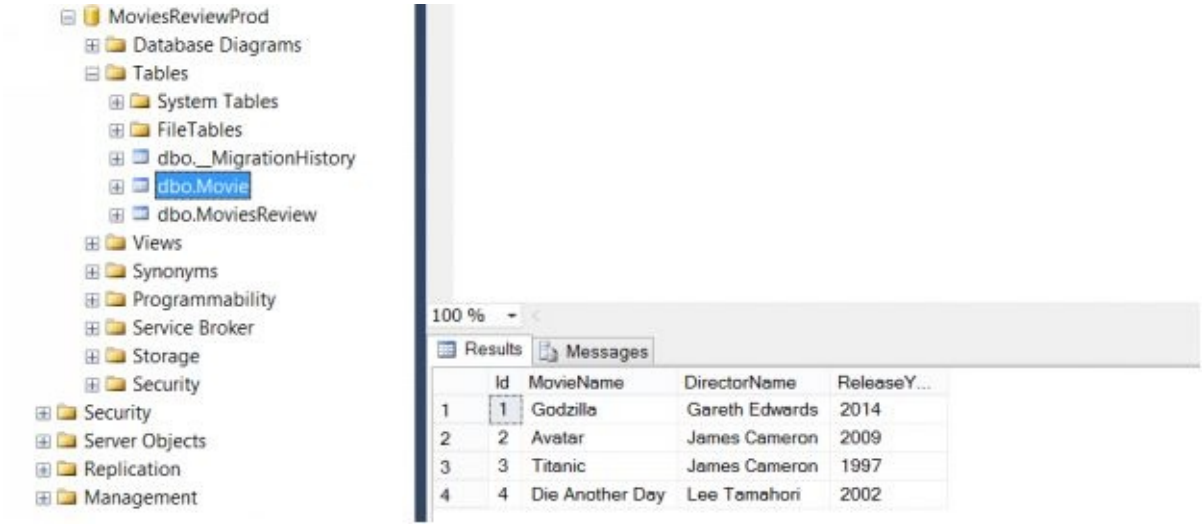


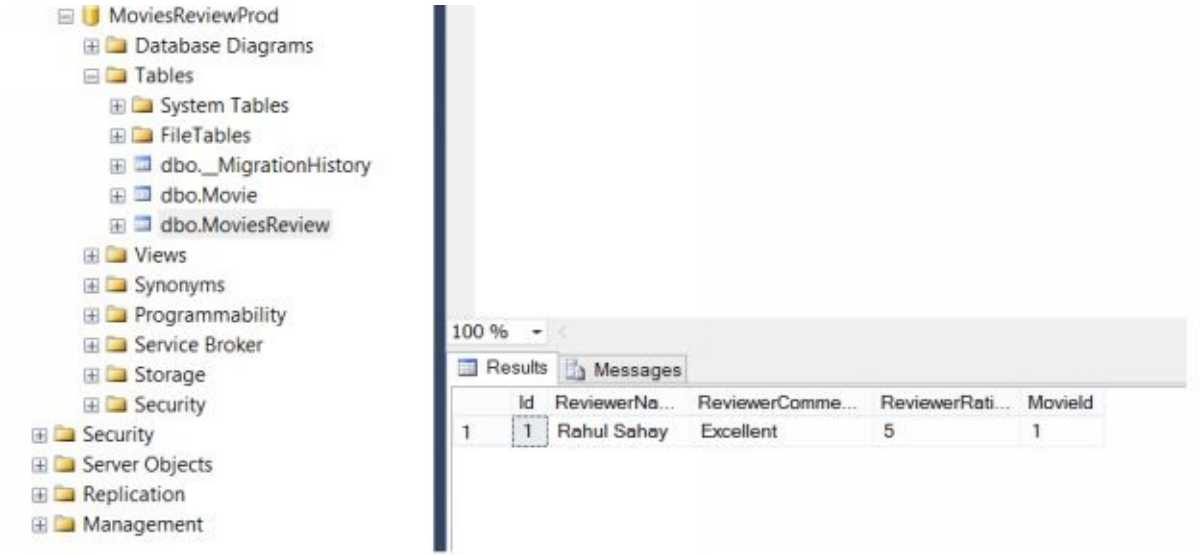

Now, if I expand the database, then we'll see two tables **Movie** and **MoviesReview** got created. Apart from this there is one more table got created that is

_MigrationHistory. This _MigrationHistory is a table which keeps track of all the database migrations.




Now, if we expand both the tables, we can see exactly same values what we seeded as shown below in the screenshot.



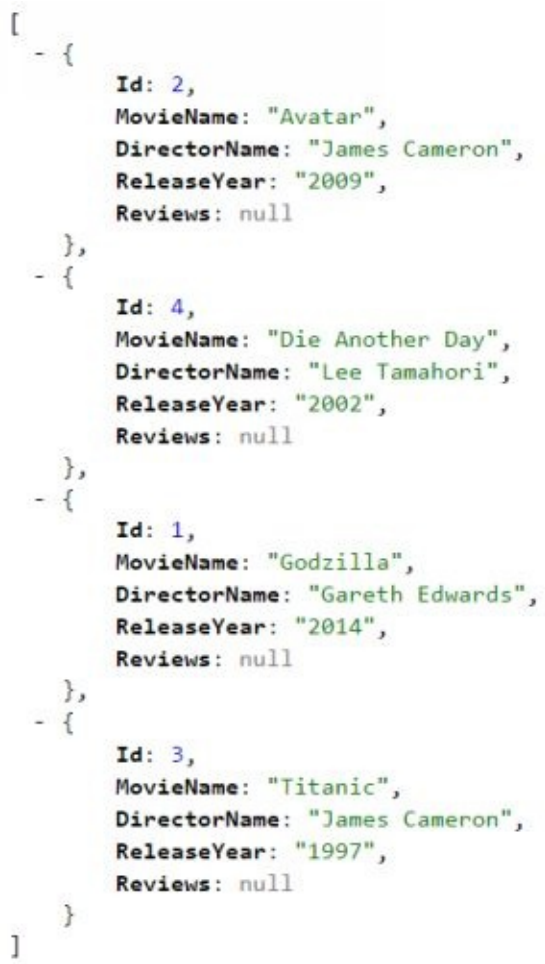



However, the data which is getting listed in the browser is getting listed in the XML format. That's ok but, I really don't like this xml view. I need to resolve the same in JSON view. Now, in order to do the same I need to make the JSON formatting as default formatter in the Web API config as shown below in the snippet.

```
using System.Linq;
using System.Web.Http;

namespace MovieReview.Web
{
    public static class WebApiConfig
    {
    public static void Register(HttpConfiguration config)
    {

    //Below formatter is used for returning the Json result.
    var appXmlType = config.Formatters.XmlFormatter.SupportedMediaTypes.FirstOrDefault(t => t.MediaType == "application/xml");
    config.Formatters.XmlFormatter.SupportedMediaTypes.Remove(appXmlType);

    // Web API configuration and services

    // Web API routes
    config.MapHttpAttributeRoutes();

    config.Routes.MapHttpRoute(
    name: "DefaultApi",
    routeTemplate: "api/{controller}/{id}",
    defaults: new { id = RouteParameter.Optional }
```

```
    );
    }
    }
}
```

With the above change in place when I build and refresh the app, it will produce me the same result but in JSON format as shown below in the screenshot. I have also installed one Chrome Extension "**JSONView**" to make the view more presentable.



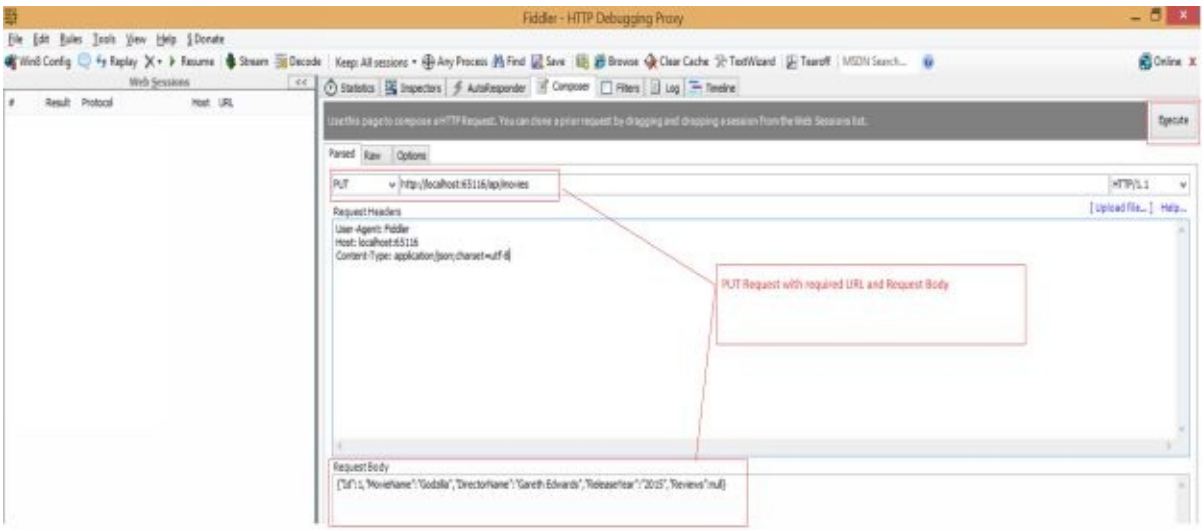

## Implementing HTTP Put Request:-

In the previous section we have seen how to create HTTP GET Request. Now, in this section we'll see how to create the PUT request and test the same with fiddler. Below is a sample snippet for updating the existing movie

```
        // Update an existing movie
    // PUT /api/movies/
    public HttpResponseMessage Put([FromBody]Movie movie)
    {
    Uow.Movies.Update(movie);
```

```
Uow.Commit();
return new HttpResponseMessage(HttpStatusCode.NoContent);
}
```

Let me explain the code in brief. Here it takes the movie object which is nothing but the movie data as sent from the browser, then it takes the request and update the required movie. But, how to test this piece of code now, as our UI is not ready yet. In order to test the same, what I am going to do is I'll simply go ahead and use Fiddler to compose my PUT and POST request and execute the above piece of code. In the below screen shot, you can see that I have composed a PUT request in the fiddler.



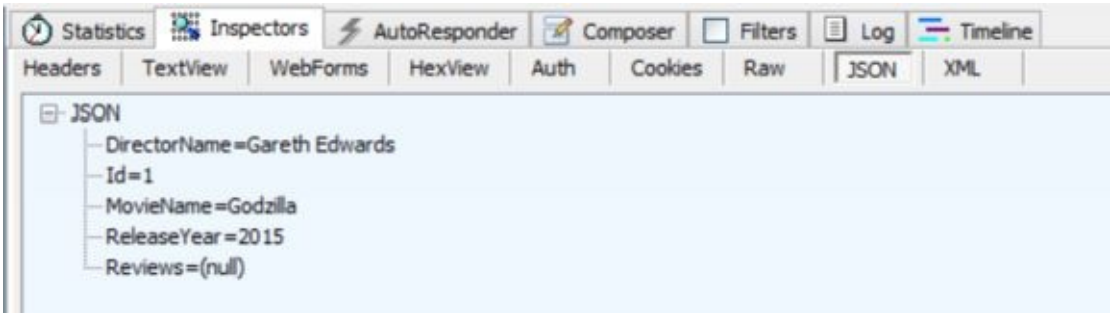

So, as you can see in the above screen shot, I have composed a PUT Request with URL http://localhost:65116/api/movies and body

{"Id":1,"MovieName":"Godzilla","DirectorName":"Gareth Edwards","ReleaseYear":"**2015**","Reviews":null}. In the body section I have modified Release Year as 2015. I have also kept breakpoint at my code, just to make sure it's hitting my piece of code.

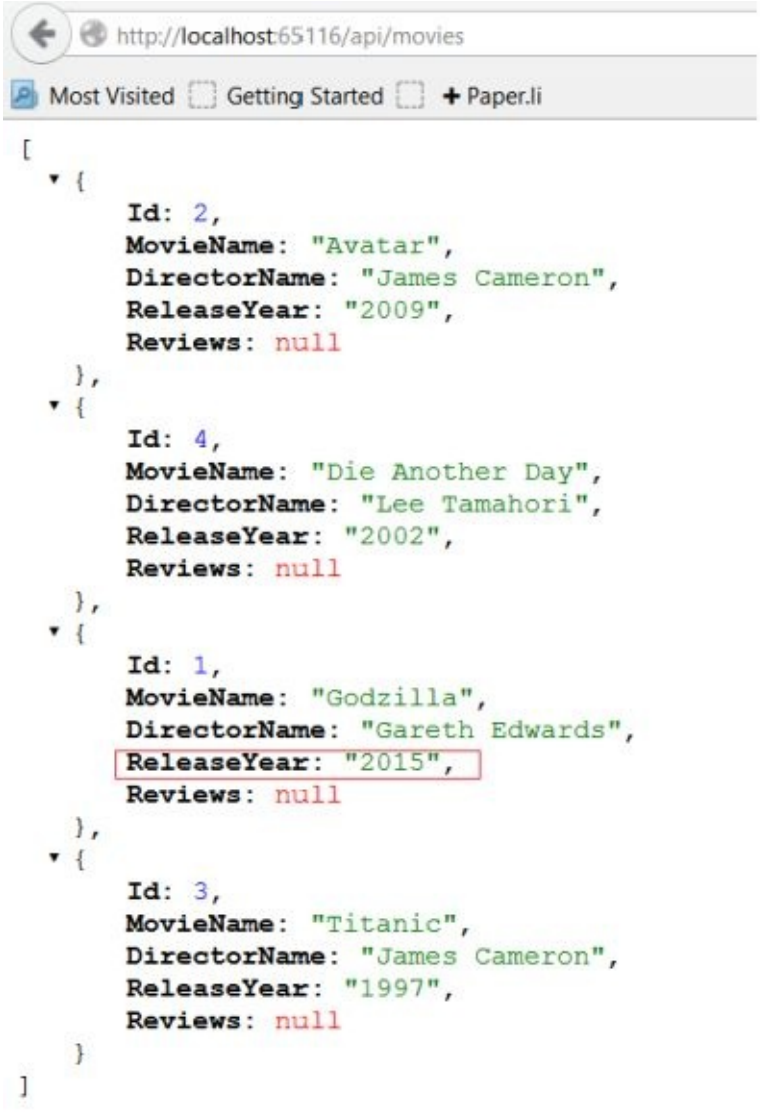

Now, upon the successful execution of the same it will return me **204 HTTP Status Code** as shown below in the screen shot.

| # | Result | Protocol | Host | URL |
|---|---|---|---|---|
| 1 | 204 | HTTP | localhost:65116 | /api/movies |

Now, when you inspect this message, you will find that the content has been successfully updated.




You, can also verify the same in the Get call as shown below in the browser.



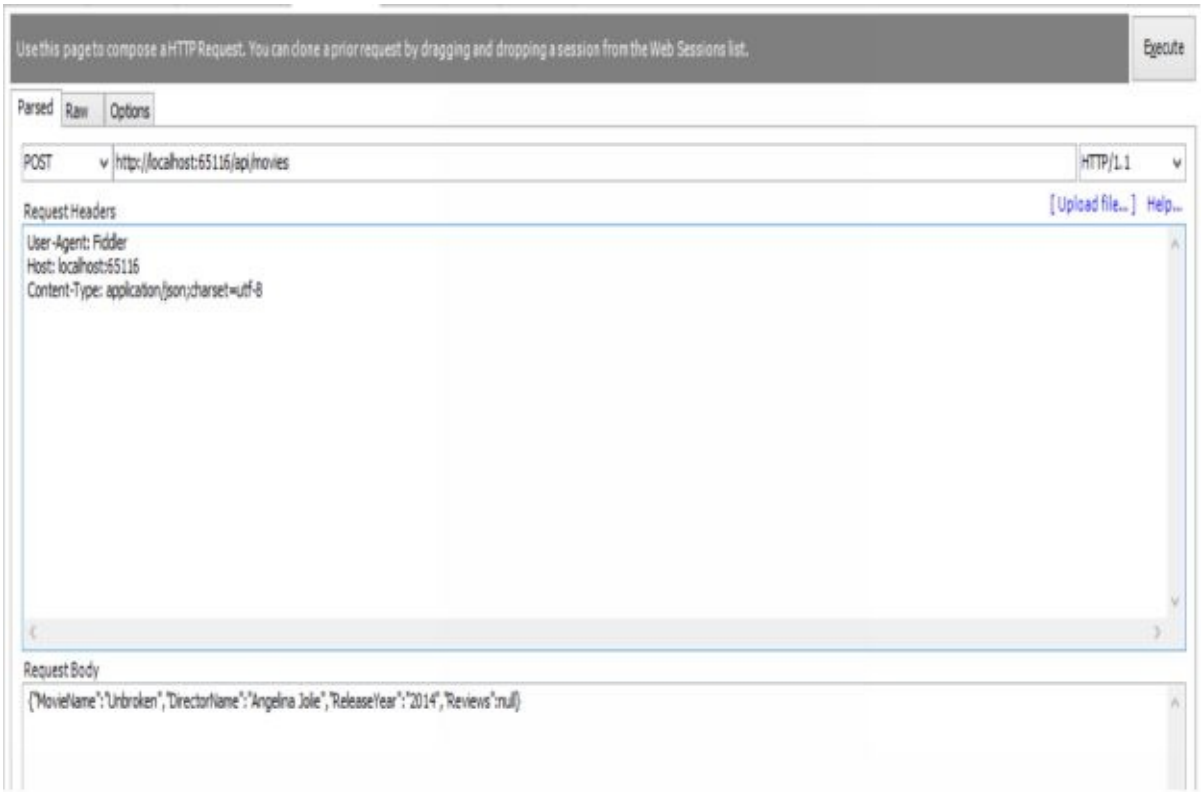

However, let me undo this change, this was just for proving my point.

## Implementing HTTP Post Request:-

Below is the sample code for implementing HTTP Post Request.

```
        // Create a new movie
    // POST /api/movies
    public HttpResponseMessage Post(Movie movie)
    {
    Uow.Movies.Add(movie);
    Uow.Commit();

    var response = Request.CreateResponse(HttpStatusCode.Created, movie);
    return response;
    }
```

In the above snippet, I am simply taking the request from user and then adding the same in the Movies table. Upon successful execution it will return Created status code.



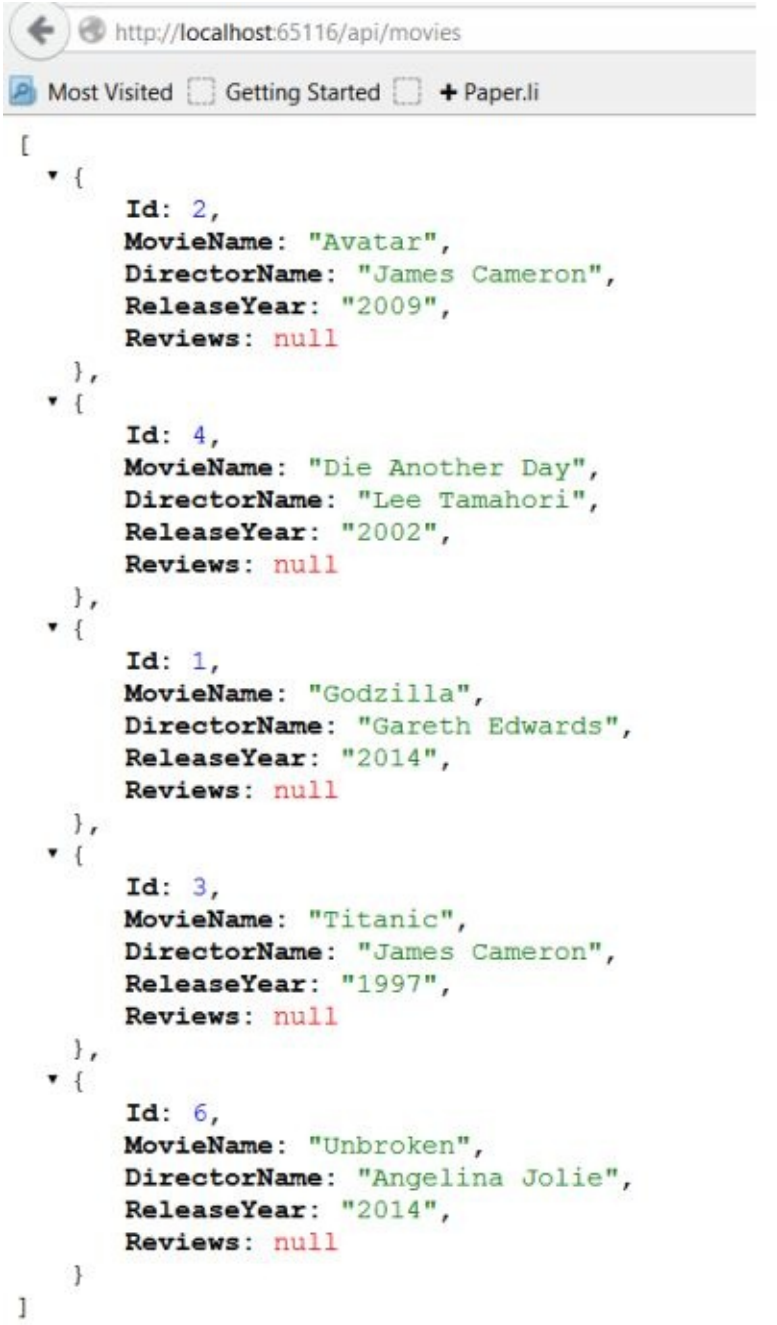
```
37
38      // Create a new movie
39      // POST /api/movies
        0 references
40      public HttpResponseMessage Post(Movie movie)
41      {
42          Uow.Movies.Add(movie);
43          Uow.Commit();
44
45          var response = Request.CreateResponse(Ht
46          return response;
47      }
48
```

movie {MovieReview.Model.Movie}

| | |
|---|---|
| DirectorName | "Angelina Jolie" |
| Id | 0 |
| MovieName | "Unbroken" |
| ReleaseYear | "2014" |
| Reviews | null |

| # | Result | Protocol | Host | URL |
|---|---|---|---|---|
| 1 | 201 | HTTP | localhost:65116 | /api/movies |

Also, you can see in the above snippet that I have passed nothing as ID in the payload and the reason for the same is because it is auto generated property in the database. So, based on the response getting returned back, it will simply create the new record in the database. Now, when I refresh my get call in the browser, you will see the newly added movie as shown in the below screen shot.

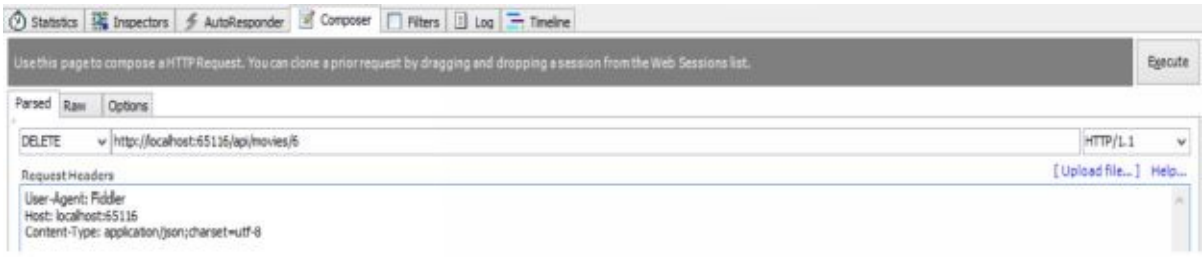



## Implementing HTTP Delete Request:-

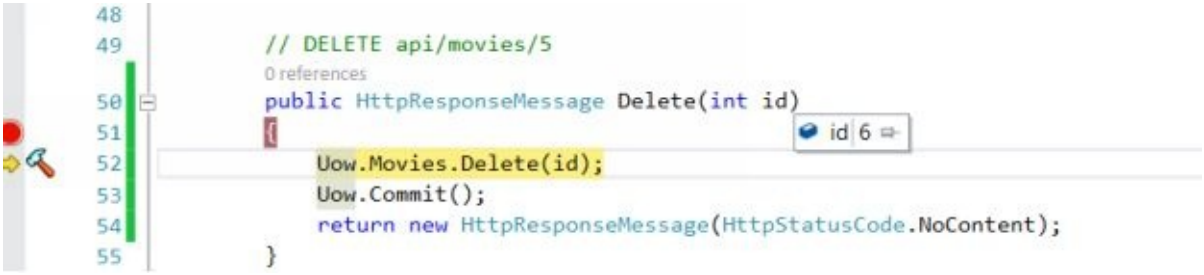
In this section, we'll follow the same pattern but will see how HTTP Delete works.

```
        // DELETE api/movies/5
    public HttpResponseMessage Delete(int id)
    {
    Uow.Movies.Delete(id);
    Uow.Commit();
    return new HttpResponseMessage(HttpStatusCode.NoContent);
    }
```

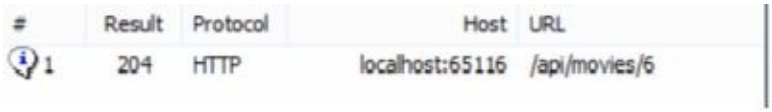
Now, same way in the fiddler, but this time I am going to compose the Delete Request as shown below in the screen shot.




Here, only change is that expects Movie Id in the signature of the method call which needs to be deleted.




Once, it gets deleted successfully, it will return the 204 HTTP Status Code as shown in the below screen shot.

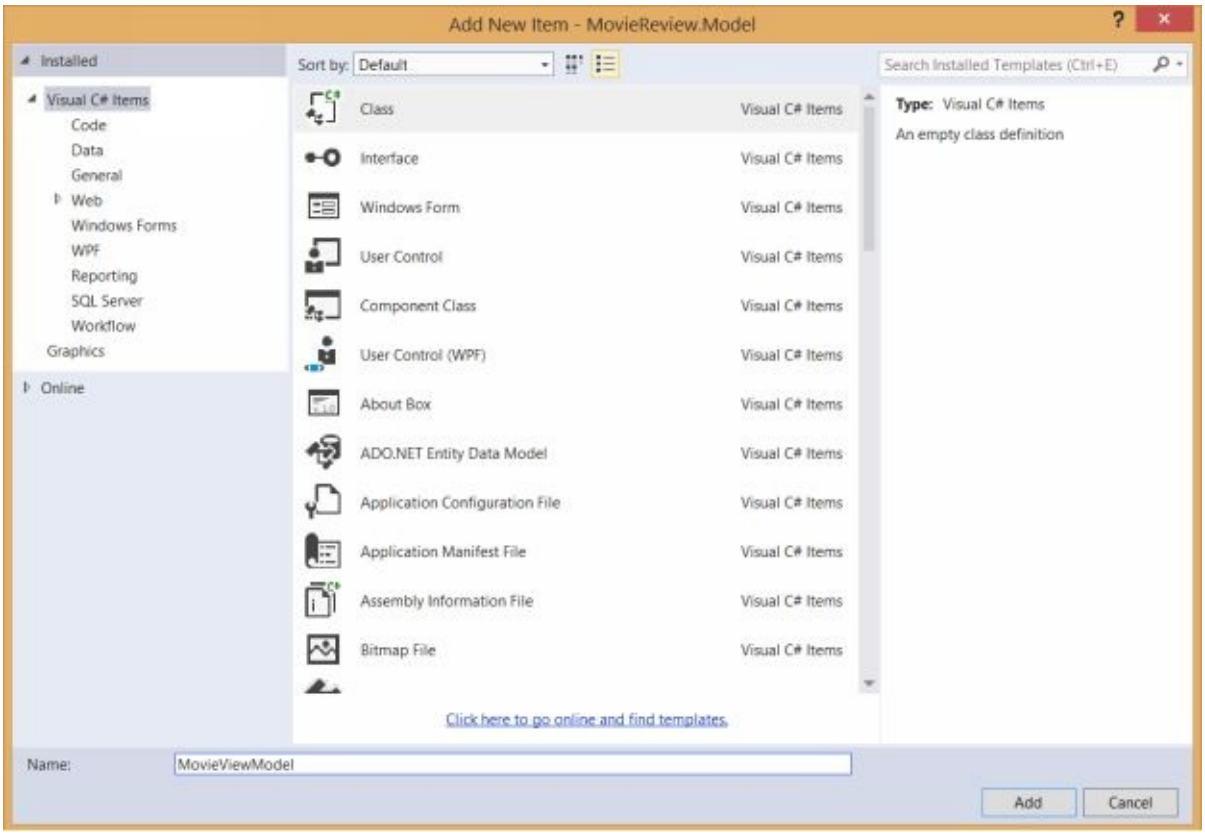


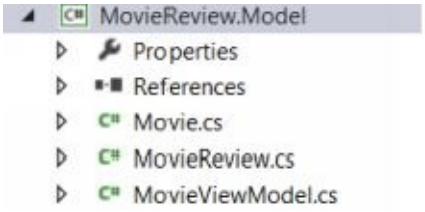

You, can also verify in the GET call as shown below in the screen shot.


However, this was the basic understanding how APIs are going to work throughout in this project. But, let me modify the same to give it final shape.

## Improvising Web APIs:-

Till, now I am not showing reviews here in the Get call, however we do have one review attached with one movie if you remember the same from seed data. Now, in order to show the same I am going to construct one **ViewModel** and I am going to query ViewModel in my Get Call. I'll create this ViewModel in my Models Project as shown below in the screen shot.



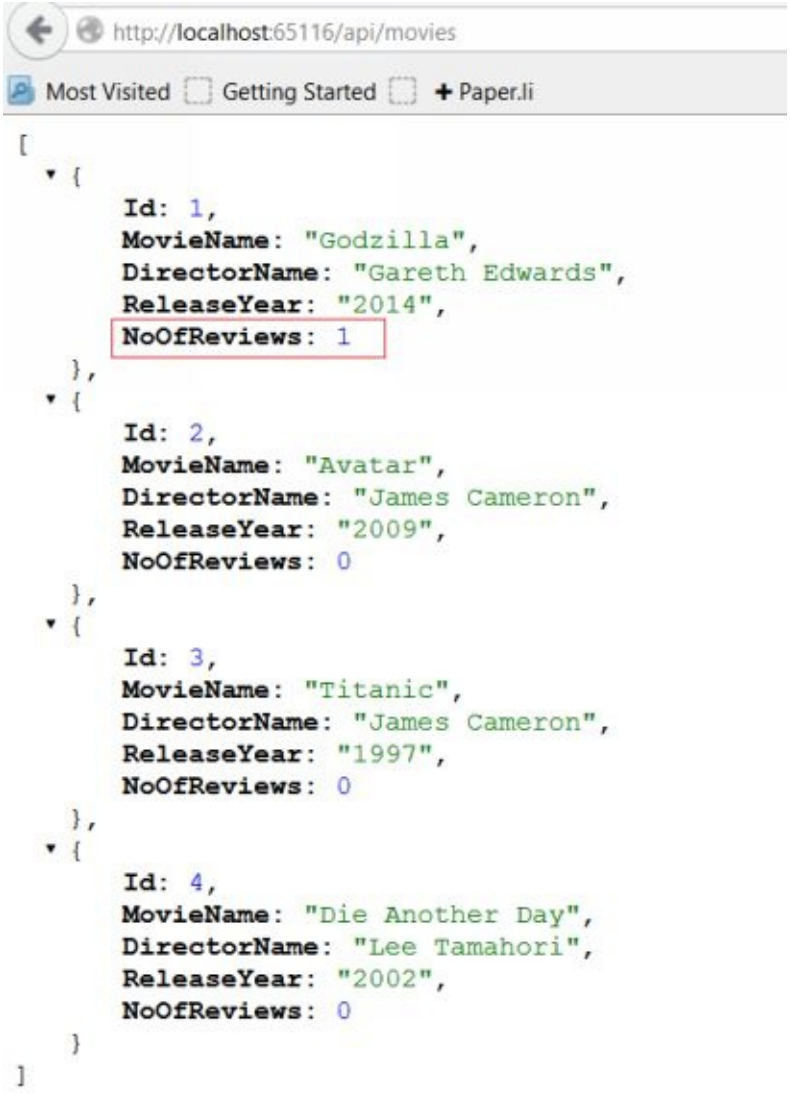

Once, ViewModel successfully created, then I'll add below shown properties in the same.

```
namespace MovieReview.Model
{
    public class MovieViewModel
    {
    public int Id { get; set; }
    public string MovieName { get; set; }
    public string DirectorName { get; set; }
    public string ReleaseYear { get; set; }
    public int NoOfReviews { get; set; }
    }
}
```

Now, I need to improvise my Get call a bit. So, below is the improvised Get Call which will list my Reviews as well.

```
        // GET api/movies
public IQueryable Get()
{
var model = Uow.Movies.GetAll().OrderByDescending(m => m.Reviews.Count())
.Select(m => new MovieViewModel
{
Id = m.Id,
MovieName = m.MovieName,
DirectorName = m.DirectorName,
ReleaseYear = m.ReleaseYear,
NoOfReviews = m.Reviews.Count()
});
return model;

}
```

Now, with the above change in place, if I go build the project and refresh my API call in the browser, then it will present me the following result.




Below, I have pasted the complete movies controller code in its finished form.

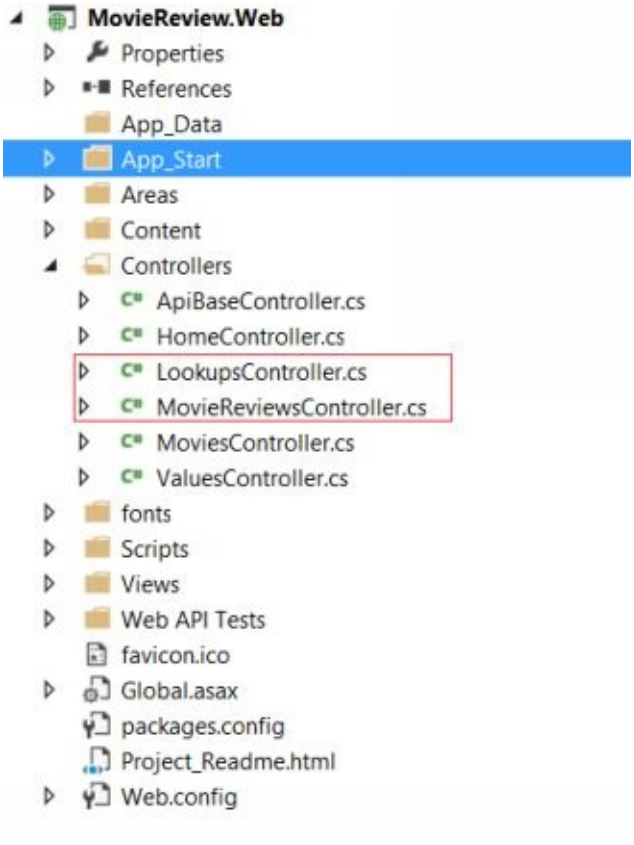
```
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using MovieReview.Data.Contracts;
using MovieReview.Model;

namespace MovieReview.Web.Controllers
{
    public class MoviesController : ApiBaseController
    {
    public MoviesController(IMovieReviewUow uow)
    {
    Uow = uow;
    }
    // GET api/movies
```

```
public IQueryable Get()
{
var model = Uow.Movies.GetAll().OrderByDescending(m => m.Reviews.Count())
.Select(m => new MovieViewModel
{
Id = m.Id,
MovieName = m.MovieName,
DirectorName = m.DirectorName,
ReleaseYear = m.ReleaseYear,
NoOfReviews = m.Reviews.Count()
});
return model;

}

// GET api/movies/5
public Movie Get(int id)
{
var movie = Uow.Movies.GetById(id);
if (movie != null) return movie;
throw new HttpResponseException(new HttpResponseMessage(HttpStatusCode.NotFound));
}

// Update an existing movie
// PUT /api/movies/
public HttpResponseMessage Put([FromBody]Movie movie)
{
Uow.Movies.Update(movie);
Uow.Commit();
return new HttpResponseMessage(HttpStatusCode.NoContent);
}

// Create a new movie
// POST /api/movies
public HttpResponseMessage Post(Movie movie)
{
Uow.Movies.Add(movie);
Uow.Commit();

var response = Request.CreateResponse(HttpStatusCode.Created, movie);
return response;
}
```

```
    // DELETE api/movies/5
    public HttpResponseMessage Delete(int id)
    {
    Uow.Movies.Delete(id);
    Uow.Commit();
    return new HttpResponseMessage(HttpStatusCode.NoContent);
    }
    }
}
```

## Adding More Controllers:-

In this section I'll add couple of more controllers to back up my app. I have already added new controllers in my solution as shown below in the screen shot.




Now, let me explain the same.

```
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
```

```
using MovieReview.Data.Contracts;
using MovieReview.Model;

namespace MovieReview.Web.Controllers
{
    public class MovieReviewsController : ApiBaseController
    {
    public MovieReviewsController(IMovieReviewUow uow)
    {
    Uow = uow;
    }

    public IEnumerable<MoviesReview> Get()
    {
    return Uow.MovieReviews.GetAll().OrderBy(m => m.MovieId);
    }


    public IEnumerable<MoviesReview> Get(int Id)
    {
    return Uow.MovieReviews.GetAll().Where(m => m.MovieId == Id);
    }

    // /api/MovieReviews/getbyreviewername?value=rahul
    [System.Web.Http.ActionName("getbyreviewername")]
    public MoviesReview GetByReviewerName(string value)
    {
    var review = Uow.MovieReviews.GetAll().FirstOrDefault(m => m.ReviewerName.StartsWith(value));

    if (review != null) return review;
    throw new HttpResponseException(new HttpResponseMessage(HttpStatusCode.NotFound));
    }

    // Update an existing review
    // PUT /api/MovieReviews/
    public HttpResponseMessage Put([FromBody]MoviesReview review)
    {
    //review.Id = Id;
    Uow.MovieReviews.Update(review);
    Uow.Commit();
    return new HttpResponseMessage(HttpStatusCode.NoContent);
    }
```

```
    // Create a new review
    // POST /api/MovieReviews
    public HttpResponseMessage Post(MoviesReview review, int Id)
    {
    review.MovieId = Id;
    Uow.MovieReviews.Add(review);
    Uow.Commit();

    var response = Request.CreateResponse(HttpStatusCode.Created, review);



    return response;
    }

    //Delete a review
    //Delete /api/MovieReviews/5
    public HttpResponseMessage Delete(int id)
    {
    Uow.MovieReviews.Delete(id);
    Uow.Commit();
    return new HttpResponseMessage(HttpStatusCode.NoContent);
    }
    }
}
```

Movie Review controller is almost equivalent to Movie controller. Here, nothing new I have written. Only difference is rather using Movie Table, I'm using here Movie Review table that's it. Here also same way I can go ahead and make my API calls as shown below.


One point to note here, I have changed the port no here as the initial one got corrupted due

to some reason. Now, the next controller is **LookUps** controller. This controller is basically a custom controller which you can understand by looking at the code as shown below.

```
using System.Collections.Generic;
using System.Linq;
using System.Web.Http;
using MovieReview.Data.Contracts;
using MovieReview.Model;

namespace MovieReview.Web.Controllers
{
    public class LookupsController : ApiBaseController
    {
    public LookupsController(IMovieReviewUow uow)
    {
    Uow = uow;
    }

    // GET: api/lookups/movies
    [ActionName("movies")]
    public IEnumerable<Movie> GetMovies()
    {
    return Uow.Movies.GetAll().OrderBy(m => m.Id);
    }

    // GET: api/lookups/movieReviews
    [ActionName("movieReviews")]
    public IEnumerable<MoviesReview> GetMoviesReviews()
    {
    return Uow.MovieReviews.GetAll().OrderBy(m => m.MovieId);
    }

    // /api/Lookups/getbyreviewerid?id=1
    [System.Web.Http.ActionName("getbyreviewerid")]
    public MoviesReview GetByReviewerId(int id)
    {
    return Uow.MovieReviews.GetById(id);
    }

    #region OData Future: IQueryable<T>
    //[Queryable]
```

```
    // public IQueryable<Movie> Get()
    // public IQueryable<MovieReview> Get()

    #endregion

    }
}
```

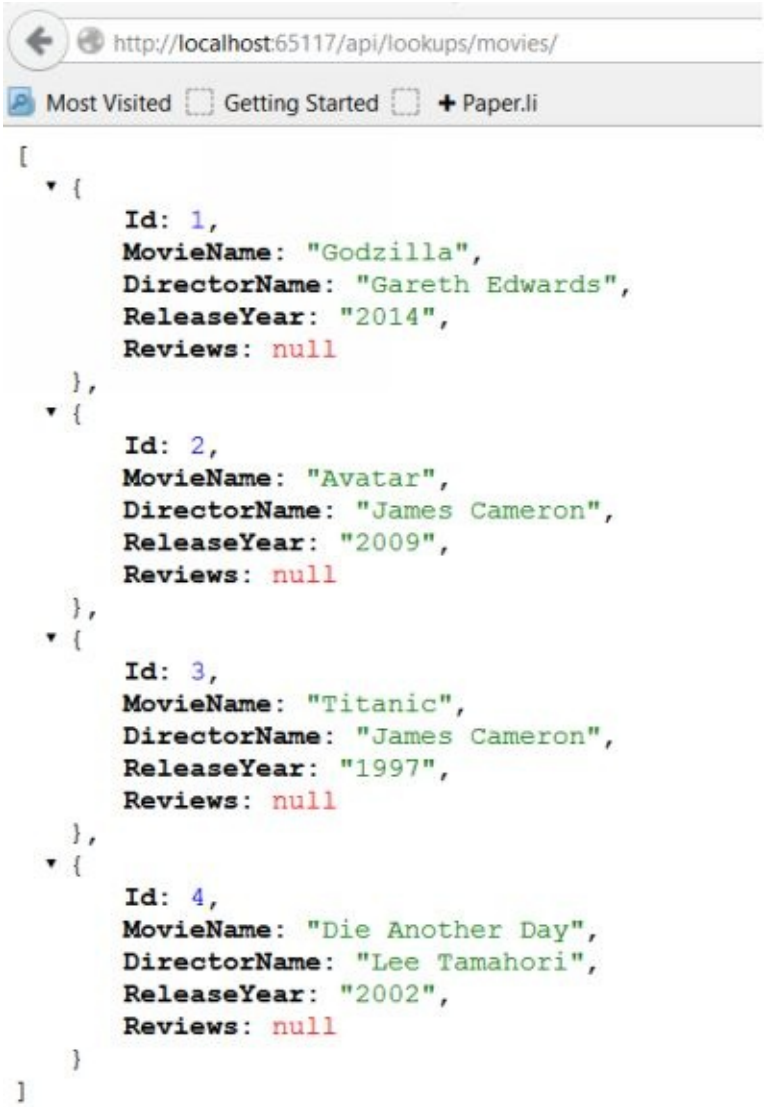
This controller I have written to make more custom calls for future requirement purpose. Like you see above these all results we are fetching from previous two controllers, but let's suppose you want to go ahead and make more unique call down the layer, then these types of controller will be beneficial for you. With this piece of change in place when I build the app and navigate to URL http://localhost:65117/api/lookups/movies/ . Then, it will produce me the below result.


And reason for the same is my existing API route rule couldn't resolve this. So, to fix the same I need to rewrite the rule as shown below

```
using System.Linq;
using System.Web.Http;

namespace MovieReview.Web
{
    public static class WebApiConfig
    {
    public static string ControllerOnly = "ApiControllerOnly";
    public static string ControllerAndId = "ApiControllerAndIntegerId";
    public static string ControllerAction = "ApiControllerAction";
    public static void Register(HttpConfiguration config)
    {
    // Web API configuration and services

    // Web API routes
    config.MapHttpAttributeRoutes();
    //Below formatter is used for returning the Json result.
```

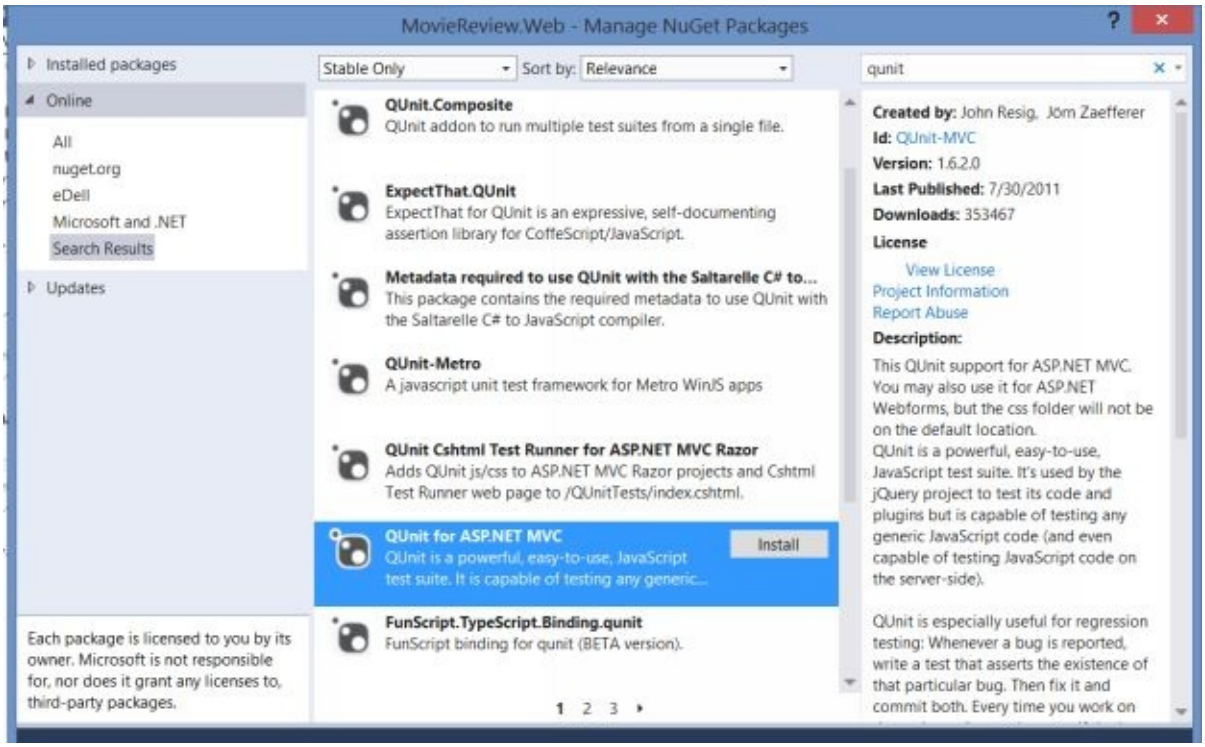
```
    var appXmlType = config.Formatters.XmlFormatter.SupportedMediaTypes.FirstOrDefault(t => t.MediaType == “application/xml”);
    config.Formatters.XmlFormatter.SupportedMediaTypes.Remove(appXmlType);

    config.Routes.MapHttpRoute(
    name: ControllerOnly,
    routeTemplate: “api/{controller}”
    );

    config.Routes.MapHttpRoute(
    name: ControllerAndId,
    routeTemplate: “api/{controller}/{id}”,
    defaults: null, //defaults: new { id = RouteParameter.Optional } //,
    constraints: new { id = @”^\d+$” } // id must be all digits
    );

    config.Routes.MapHttpRoute(
    name: ControllerAction,
    routeTemplate: “api/{controller}/{action}”

    );
    }
    }
}
```

So, here I have written different rules for different kinds of scenarios explicitly. Now with this change when I build and refresh the page, then I’ll get the below result.

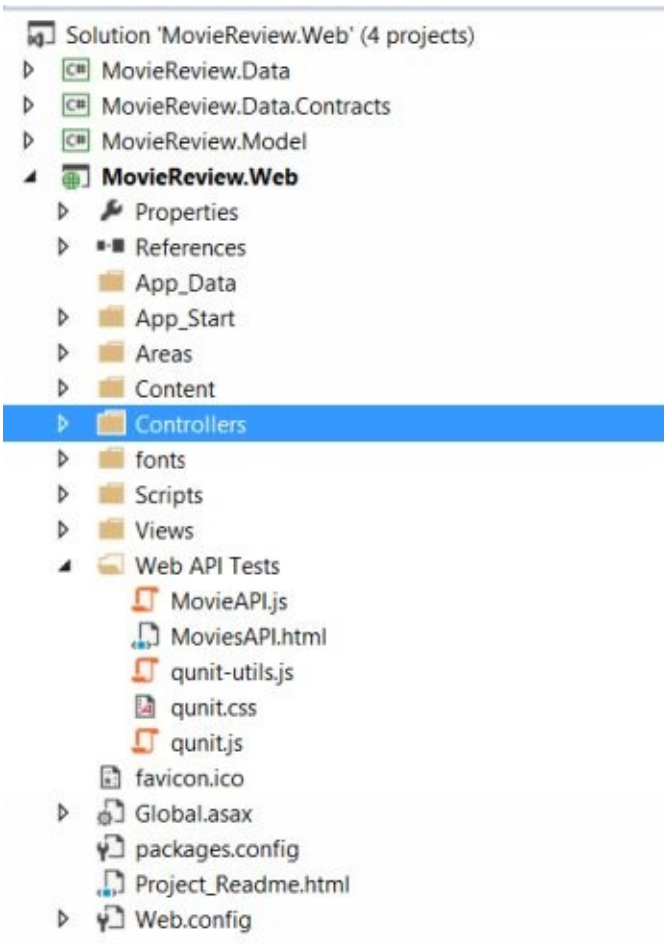



## Testing Web APIs with QUnit:-

In this section, we'll test our APIs with **Qunit.JS** Framework. Now, in order to install the same, I will use Nuget Package Manager as shown below in the screen shot.




Once, installed it will move the files in scripts folder. But, I like to keep the same in my custom folder say "**Web API Tests**" as shown below in the screen shot.



Let me show the files which I have customized to run the test. 1st one is the html file which is nothing but a medium of loading and running the tests. This file also includes the required script and other css references.

```
<!DOCTYPE html>

<html lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head>
    <meta charset="utf-8" />
    <title>Movies API HTML</title>



    <script src="../Scripts/jquery-1.10.2.js"></script>

    <link href="qunit.css" rel="stylesheet" />

    <script type="text/javascript" src="qunit.js"></script>
    <script type="text/javascript" src="qunit-utils.js"></script>
    <script type="text/javascript" src="movieapi.js"></script>
</head>
<body>
    <h1 id="qunit-header">Movie Review Web API Test  </h1>
    <h2 id="qunit-banner"></h2>
    <div id="qunit-testrunner-toolbar"></div>
    <h2 id="qunit-userAgent"></h2>
    <ol id="qunit-tests"></ol>
    <div id="qunit-fixture"></div>
</body>

</html>
```

Now, the most important thing in this context is my JS file.

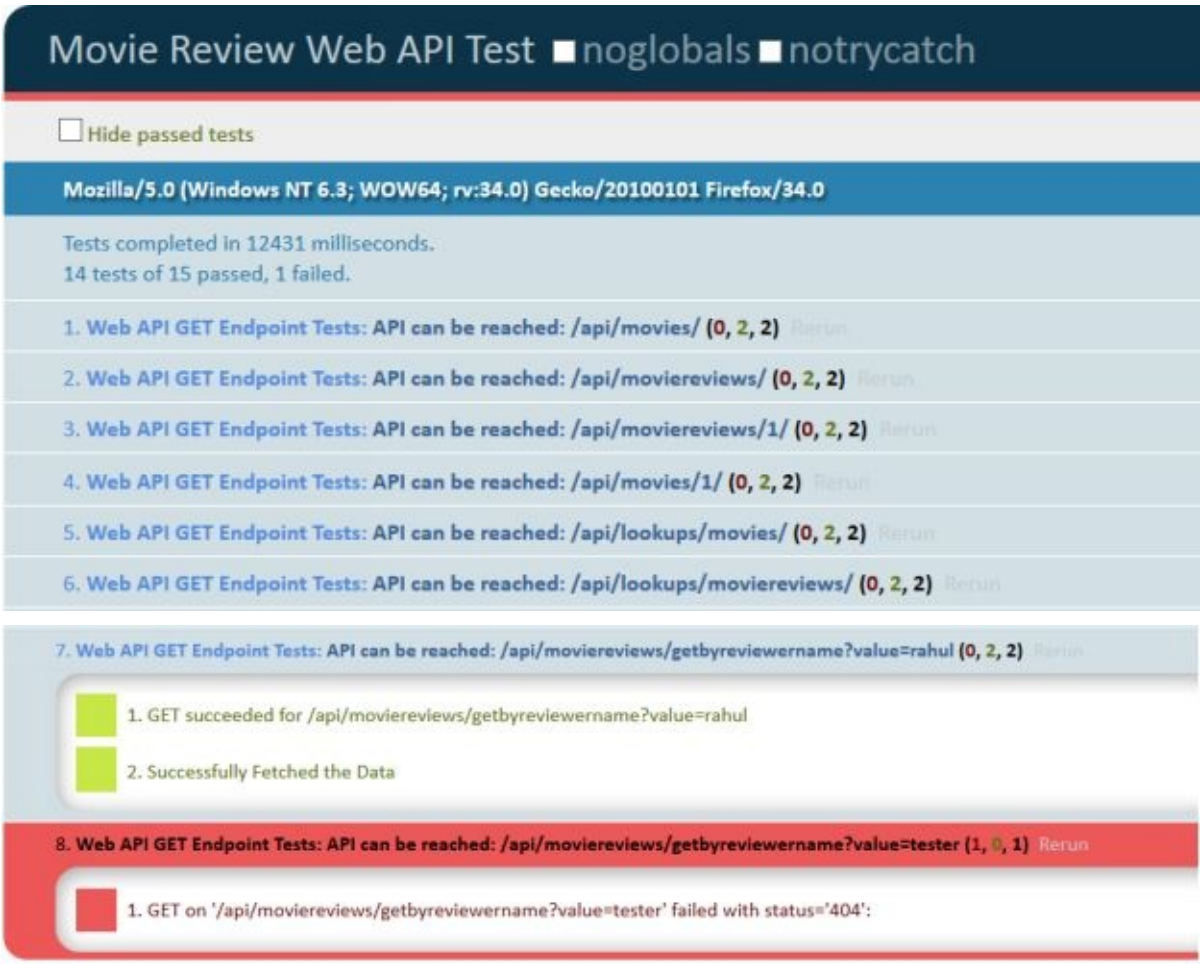
```
(function () {
    QUnit.config.testTimeout = 10000;

    var stringformat = QUnit.stringformat;

    module('Web API GET Endpoint Tests');
```

```
var apiUrls = [

//List all APIs Here

'/api/movies/',
'/api/moviereviews/',
'/api/moviereviews/1/',
'/api/movies/1/',
'/api/lookups/movies/',
'/api/lookups/moviereviews/',
'/api/moviereviews/getbyreviewername?value=rahul',
'/api/moviereviews/getbyreviewername?value=tester'

];


var apiUrlslen = apiUrls.length;

// Test only that the Web API responded to the request with 'success'
var endpointTest = function (url) {
$.ajax({
url: url,
dataType: 'json',
success: function (result) {
ok(true, 'GET succeeded for ' + url);
ok(!!result, 'Successfully Fetched the Data');
start();
},
error: function (result) {
ok(false,
stringformat('GET on '{0}' failed with status='{1}': {2}',
url, result.status, result.responseText));
start();
}
});
};

// Returns an endpointTest function for a given URL
var endpointTests = function (url) {
return function () { endpointTest(url); };
};
```

```
    // Test each endpoint in apiUrls
    for (var i = 0; i < apiUrlslen; i++) {
    var apiUrl = apiUrls[i];
    asyncTest(
    'API can be reached: ' + apiUrl,
    endpointTests(apiUrl));
    };
})();
```

So, if you see the code closely, you will come to know that this is self-executing jQuery function which starts from the **for loop** where in it loops through all the end points and then invoke the test. Very simple and scalable test pattern. Doesn't matter how many endpoints you have, you can go ahead put the same at one place.

Now, let's go ahead and try to run the test. For that I'll simply right click the html file and open the same in browser as shown below.




As you can see in the above screen shot, here all the tests ran through except the last one which actually I just failed purposely to show both pass and failed tests. Like this we can

use this framework for writing all kind of **POST/PUT/DELETE** scenarios. However, if you want to learn more on QUnit JS, you can refer http://qunitjs.com/.

### Summary:-

In this section, we have started with the design of our API controllers. While getting started with basics we wrote some simple queries just to demonstrate the functioning of API. Once, get the feel of the same then extended the same example towards more mature and robust API design. In this section we have also covered all CRUD operation via API call. Then, in the end I have also introduced one testing framework QUnit JS for testing our APIs. In the next section, we'll start focusing on our UI by getting started with Angular JS.

## Chapter 4: Getting Started with Angular JS

### WHAT DO you find in this CHAPTER?

- Introduction
- Getting started with PLNKR
- Getting started with UI Design

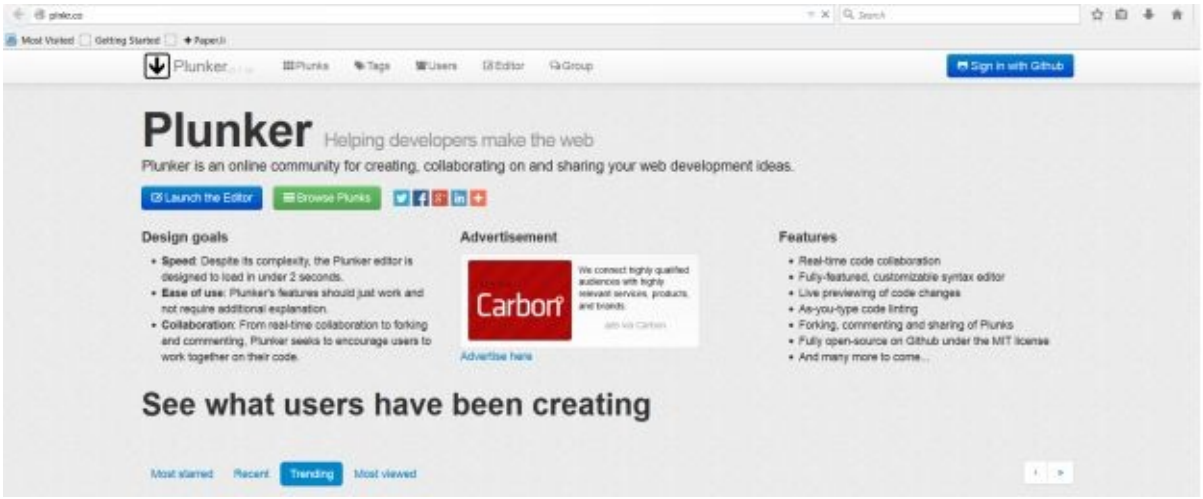
- Creating 1st Angular Controller
- Data-Binding using Angular
- Retrieving Data from API
- Summary

## Introduction:-

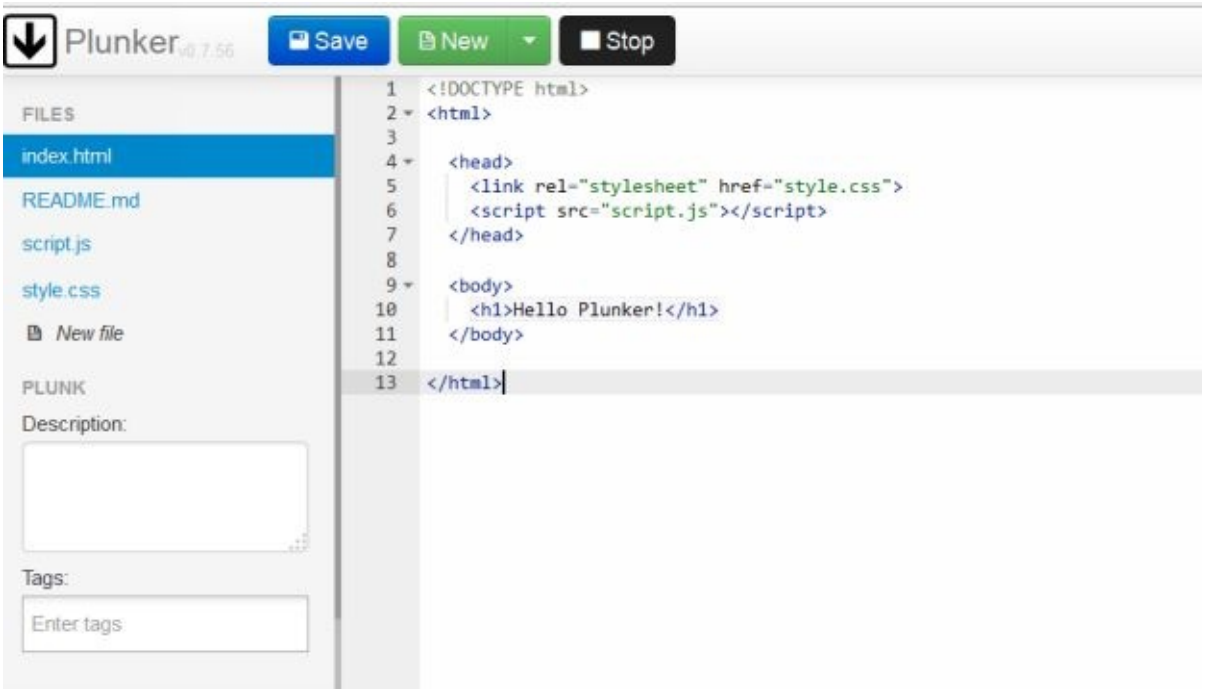
In this section, we'll start with the basics. We'll write some basic functionalities using angular. However, for doing basic hands on with Angular, I suggest you use online tool http://plnkr.co/ to write your script against any library and test the same live. However, I'll be using this tool to explain the basics of angular. Once done, then we'll jump in our application with the explanations.

## Getting started with PLNKR:-

In this section we'll see one online tool just to write script and test the same live. **PLNKR** which I prefer the most to quickly test any scripts on the fly. This can be reached at http://plnkr.co/ . Now, once you open the same, you will see this window as shown in the screen shot.




Now, in there when you click on Launch Editor, it will simply open one editor embedded in the browser itself as shown below in the screen shot.

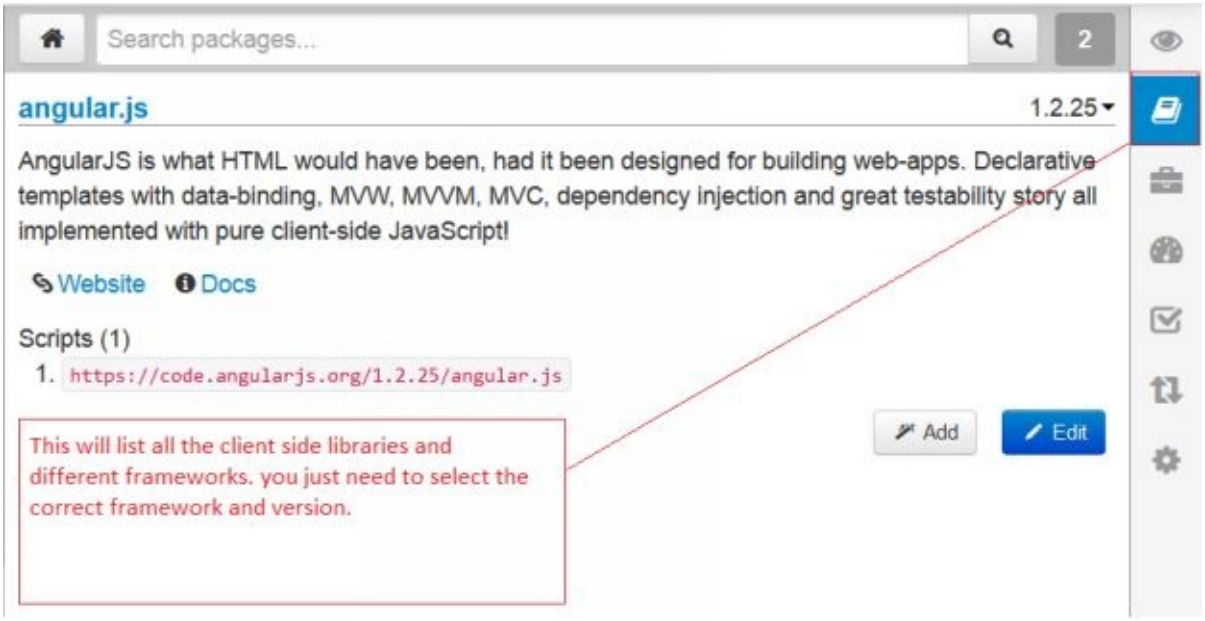



So, this is the editor pane, where in you can see the HTML file, JS file and style file. It also gives option for md as you can go ahead and login with github account and manage plunks with that account as well.

Now, there is also Live View associated with this window which is shown below in the screen shot. So, whatever, you will change in the left pane will appear in the Live Preview pane at the right side window.


Here, if you need to add additional libraries with the editor like I need to add Angular 1.2.25version as a reference to my HTML file, then I need to click on the below shown

button and then search with library name and attach the correct version.




Now, once this is done then, it editor will look like as shown below


With angular script in place, I can go ahead and start writing my basic angular expressions. One point to note here, I'm expecting that reader of this book is well acquainted with angular syntax and its usage. Here, in this section I am going to cover basic angular usages just to showcase how to use the same in plnkr. However, for getting started with angular, you can refer my blog http://myview.rahulnivi.net/ and check for angular category.

With these changes in place let's get started.

```html
<!DOCTYPE html>
<html ng-app="">

  <head>
    <script data-require="jquery@*" data-semver="2.1.3" src="https://code.jquery.com/jquery-2.1.3.min.js"></script>
    <script data-require="angular.js@1.2.25" data-semver="1.2.25" src="https://code.angularjs.org//angular.js"></script>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>

  <body ng-controller="myController">
    <h1>{{message}}</h1>

    <div>
      First Name:- {{detail.fname}}
      <br/>
      Last Name: {{detail.lname}}
      <br/>
      Blog:- <a href={{detail.blog}}>My Blog</a>
      <br/>
       <img ng-src={{detail.pic}} title={{detail.fname}}{{detail.lname}}>
    </div>
  </body>

</html>

```

Let me explain the code briefly here, 1st of all I have couple of references of scripts. I have JQuery and Angular reference here. Then, I have one custom script reference which I have written, which I will show next. If you notice closely I have **ng-app=””** declaration there in the HTML tag which basically says this page is going to be angular driven. If we have any modules, this will get injected here. Since, currently I don’t have any references, hence this piece is marked empty.

Next is **ng-controller=“myController”**. This is a way to include to controller on the page. Here, ng-controller is the angular directive and myController is my controller which I have written in my custom script file. Then next all expressions under these curly braces **{{}}** are data binding expressions. Basically controller takes **$scope** parameter to attach the model. Now, this model is going to be used while data binding. Below is the screen shot of my custom JS file.

```javascript
// Code goes here

var myController = function($scope){

  $scope.message="Hello rahul";

  //Simple employee object have different properties attached to it
  var employee ={
    fname:"Rahul",
    lname:"Sahay",
    blog:"http://myview.rahulnivi.net",
    pic:"https://pbs.twimg.com/profile_images/2306733843/r131mqr0uyiok9ujflbo.jpeg"
  };

  $scope.detail = employee;
};
```

As you can see in the above screen shot, code is quite straight forward and self-explanatory. Here, I have simply created one controller which is having $scope as parameter. $scope is a glue between our controller and view. We will attach all data binding attributes to it and then this will get used in the view. Here, I have attached two

different properties to scope. 1st is straight forward message and 2nd one is an object with different sub properties.

Now, with the above changes in place when I see the live preview of the page, then it will look like


Now, let’s suppose we need to attach modules to it. In angular we use modules for separation of concerns. Basically in large applications this is one of the best ways to keep your code well organized and separated with other dependencies. Same script I have modified a bit and pasted below

```
// Code goes here

//Explaining modules

(function(){

  //Creates module

  var app = angular.module("myModule",[])
var httpController = function($scope,$http){

  var onFetchComplete = function(success){
    //angular return data in data attribute
    $scope.details = success.data;
  };

  var onError = function(error){
    $scope.erros = "Unable to fetch the data";
  };

  $http.get("https://api.github.com/users/rahulsahay19").
  then(onFetchComplete,onError);
};

//Wiring up controller with the module created
app.controller("httpController",httpController);

//Self Executing JS function
}());
```

So, if you see the above code this is also very straight forward. 1st thing which it talks about is creating modules. Modules creation is very easy in angular. It can be easily created using **angular.module** with the name of the module and associated dependencies if any. [ ] is basically for dependency injection. Let's suppose you are writing a module which is dependent on other module, then this will go here otherwise it will remain blank.

Here, I have named my controller **httpController** as this is using **$http.get** service to make a GET call to GIT API, that's why I passed 2 parameters while controller creation. When I made the GET call, I made use of promise. Promise here means when GET is finished with the result either success or failure then only it will execute the next statement. So, here in the promise call I have called two functions based on the incoming result. If results come successfully, then **success.data** will give me the required results otherwise error will get printed.

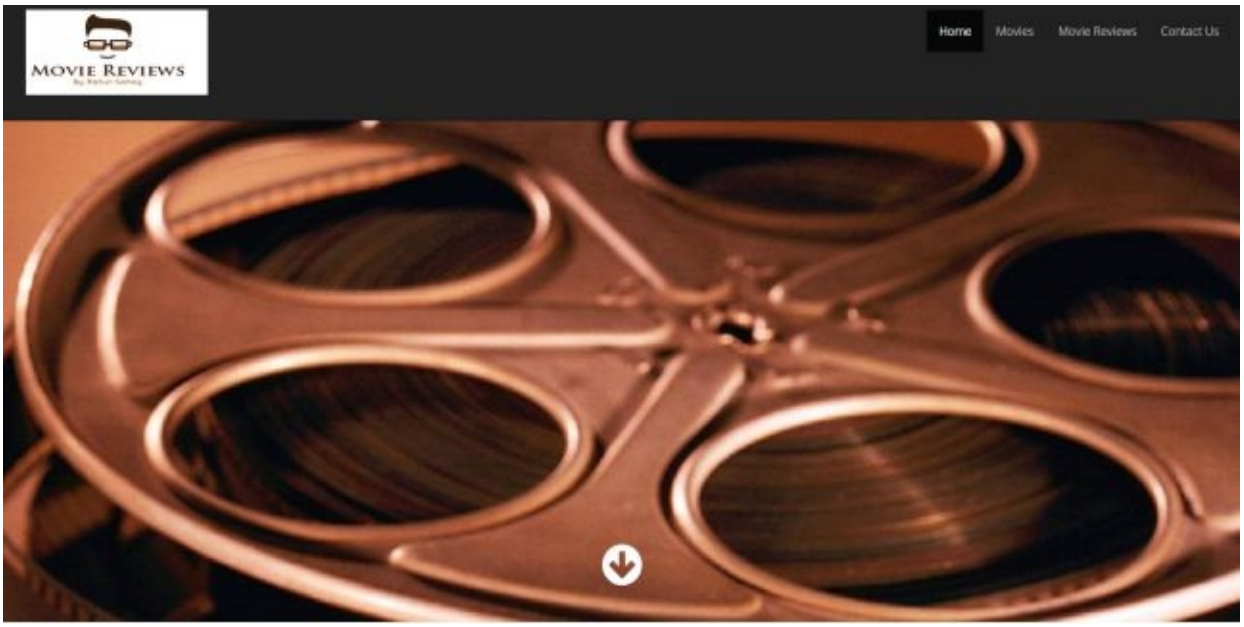
For this to execute I have made minor changed in my HTML file as shown below.

```
<!DOCTYPE html>
<html ng-app="myModule">

  <head>
    <script data-require="angular.js@1.2.25" data-semver="1.2.25" src="https://code.angularjs.org/1.2.25/angular.js"></script>
    <link rel="stylesheet" href="style.css" />
    <script src="script.js"></script>
  </head>

  <body ng-controller="httpController">
    <h1>Explaining Http Service using git api</h1>
    <div>
      {{error}}
    </div>
    <div>Name:{{details.name}}</div>
    <div>Blog:{{details.blog}}</div>
    <div>
      <img ng-src="{{details.avatar_url}}" title={{details.name}}>
    </div>
  </body>

</html>
```

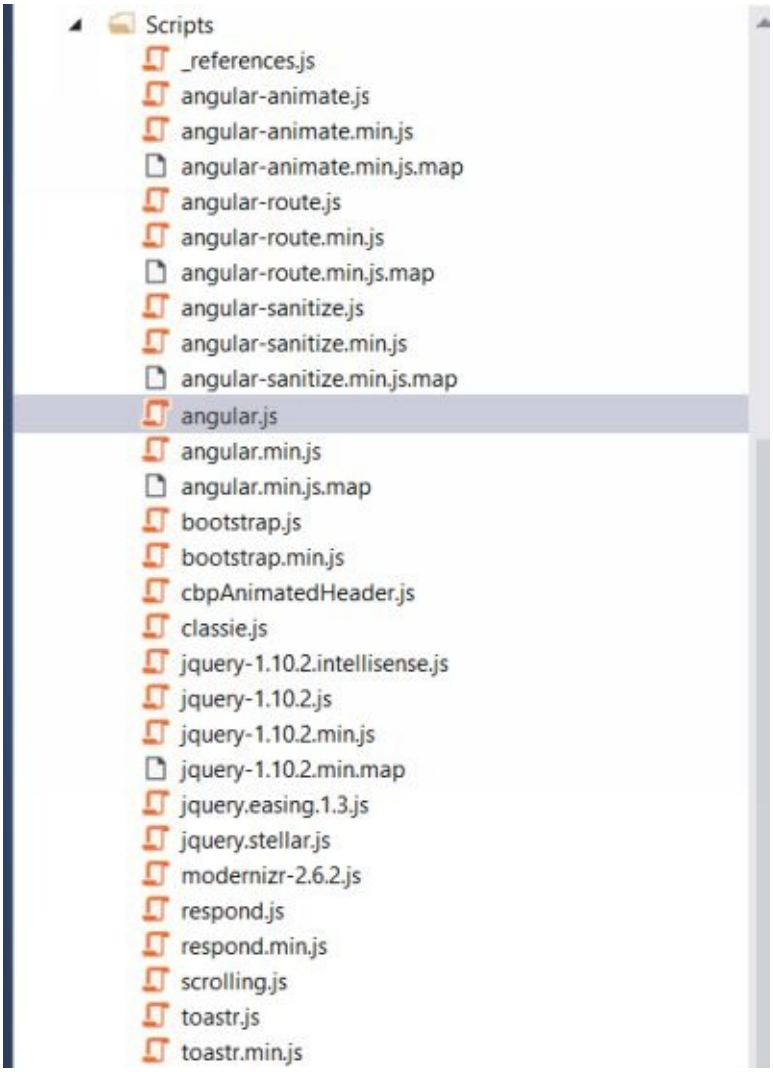
Here, basically I have included the module name in my **ng-app=”myModule”**. Now, with the above changes in place when I go ahead and just see the live view of the same result will be similar basically.


## Getting started with UI Design:-

In this section we’ll get back to our application and get started with our User Interface design. Currently, I have designed my home page using bootstrap templates and font awesome icons. Below is the screen shot of the home page.




Other pages I'll be designing live using Angular. But, the overall layout of the app will be on the above lines. So, in order to get started with Angular JS, we 1st need to install Angular JS in our project. In order to do the same, I'll go ahead add using Nuget Pacakage Manager. However, I have already installed my angular here as you can see below in the screen shot.



Here, when I downloaded I got the **1.2.26 version** as shown below in the screen shot. However, one point to note here, different angular packages behave differently, so let's suppose at the time when you are reading this book that time it might happen with latest angular library code will not work because of code refactoring and migrating dependencies to different angular file. So, either you can stick to the version which I am using or include those libraries which your code is complaining for.

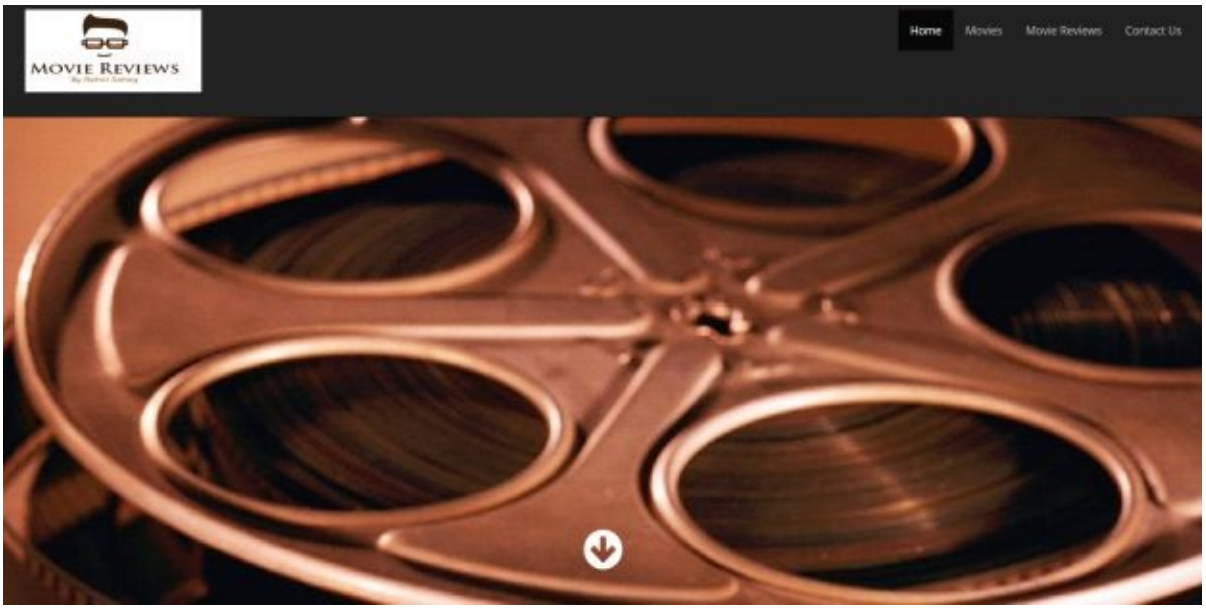
```
/*
 AngularJS v1.2.26
 (c) 2010-2014 Google, Inc. http://angularjs.org
 License: MIT
*/
```

I'll add script reference in my layout page, so that it will be available everywhere.

```
<script src="~/Scripts/jquery-1.10.2.min.js"></script>
<script src="~/Scripts/bootstrap.min.js"></script>
<script src="~/Scripts/classie.js"></script>
<script src="~/Scripts/cbpAnimatedHeader.js"></script>
<script src="~/Scripts/jquery.easing.1.3.js"></script>
<script src="~/Scripts/jquery.stellar.js"></script>
<script src="~/Scripts/scrolling.js"></script>
<script src="~/Scripts/angular.js"></script>
```

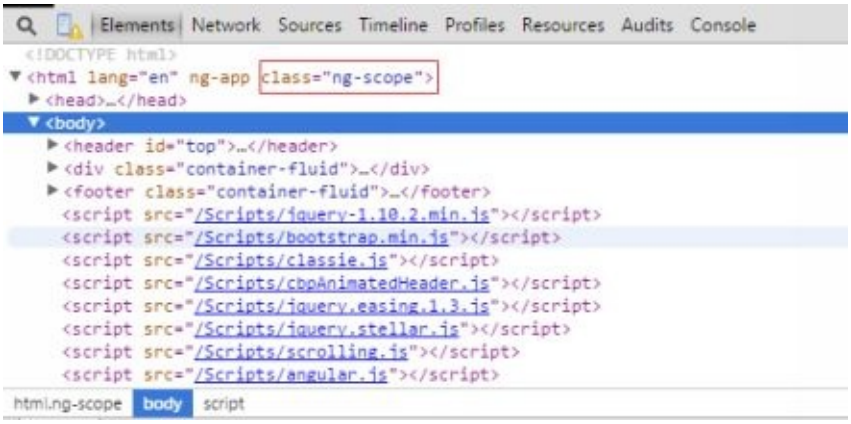
Later will use bundling and minification concept to make the page lightweight and clean as well. However, for development purpose that's ok. Now, we have put the Angular in our page, we can go ahead and initialize our page to use Angular JS and this is one of the core concepts of Angular JS. We use directives here to achieve the same as shown below in the screen shot. So, here **ng-app** tells Angular to look at this document and look for other angular directives. In this case it is going to tell Angular take a look at the page consider the same client side application.


Now, with this change in place when I go ahead and refresh my page, I don't see any visual difference.




But, when we see the debugger tool, we'll see that angular has initialized this by adding an attribute

**ng-scope.**



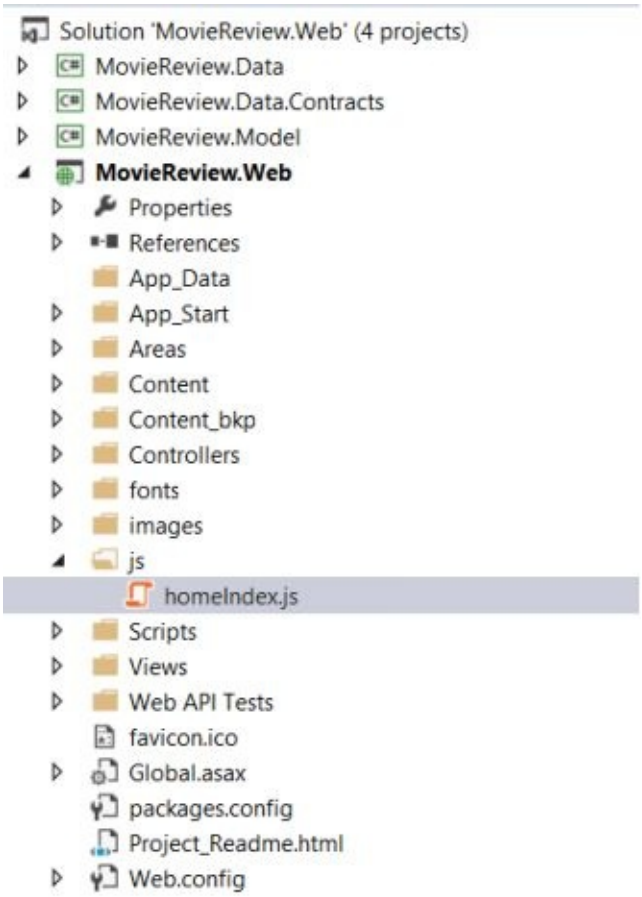

ng-scope is one of the key pieces of angular which we will discuss in detail as we progress further.

## Creating 1st Angular Controller:-

Here, I'll create my 1st controller in the index page. So, the directive which I am going to use here is **ng-controller**. You can give any name to this controller, but I always prefer **Action Method + Controller** combination. Hence **homeIndexController.**

```
Index.cshtml
<div ng-controller="homeIndexController">

</div>
```

Now, this controller name is nothing but the JavaScript function which I am going to write. I usually prefer my custom scripts in a different folder than the normal scripts folder. Hence, here I have created one folder with the name **js** and in there I have created one JavaScript file with the name **homeindex.js** and there I have created my JavaScript function **homeIndexController** as shown below in the screen shot. So, controller is nothing but the JavaScript function name.

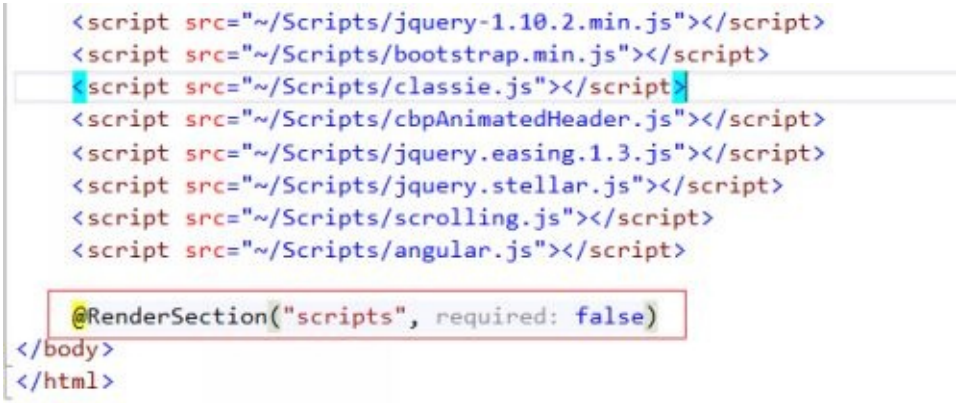



```
homeIndex.js  Index.cshtml
//homeindex.js

var homeIndexController = function() {
    alert("welcome to angular js");
}
```

Now, the next thing is to include this script on the index page, and the way I do is by using

MVC convention as shown below in the screen shot.


Also for the above section to work I need to include scripts on the layout page as well, so that this script will get added at the bottom of the page.

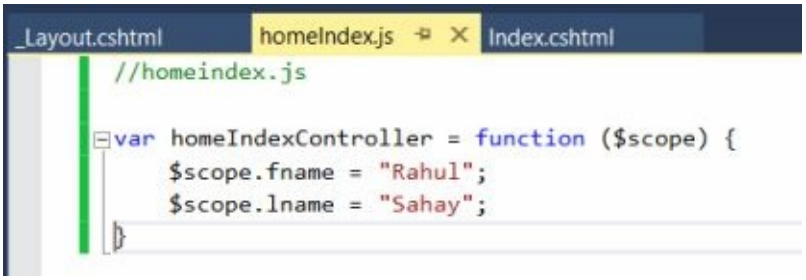



With the above change, when I go ahead and refresh the page, I should see alert on the page as shown below.


## Data-Binding using Angular:-

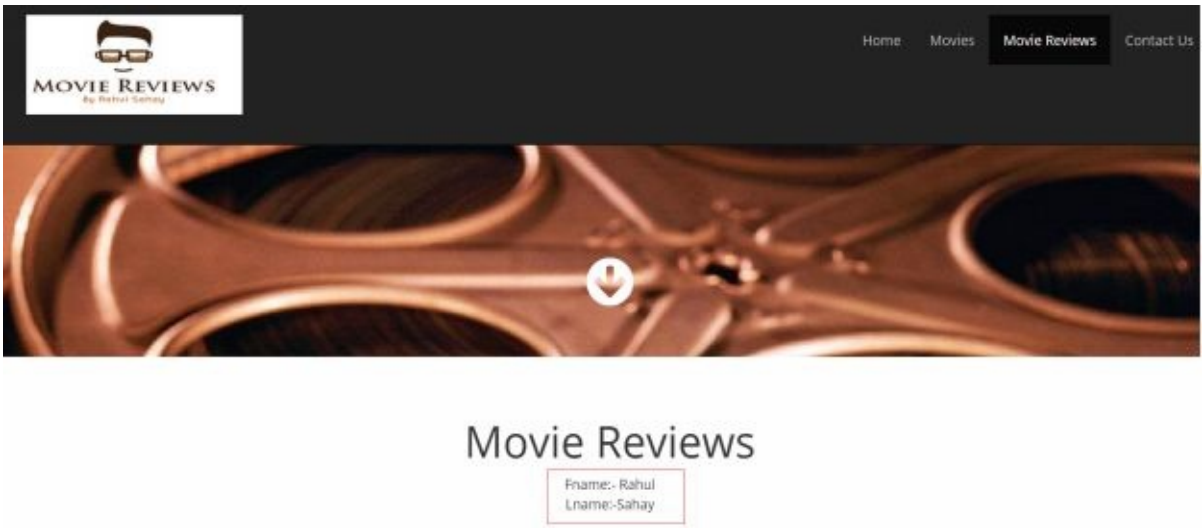
In the above section, we have seen how to create angular controller. Now, rather doing alert from there, let's go ahead and do some meaningful job from the controller. Let's do data-binding using controller. Like every Angular controller do, I'll also use **$scope** angular parameter. **$scope** is the object supplied by the Angular to the controller. So, here

scope is going to take the values from controller and supply the same to the corresponding view. Hence, in a nutshell scope is the glue between controller and view. We can also summarize "***Scope is the container for all the data which is going to be used in DOM within the HTML page.***" Another important piece of scope is we can go ahead and add any property which we want to use on the page like shown below.




Now, to use the same in view I'll be using **{{}} (double curly braces)** to databind.


With the above change in place, when I refresh the screen I'll see the below result.




However, this was the simplest one. Let's try a collection to bind here.

MovieReviewDatabaseInitializer.cs | _Layout.cshtml | homeIndex.js | Index.cshtml

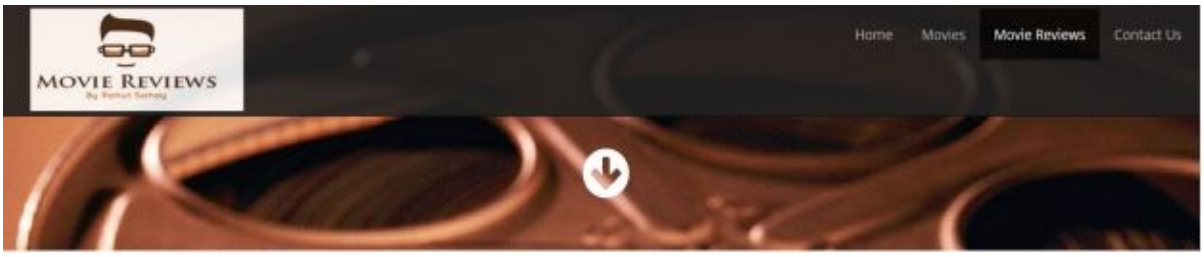
```
//homeindex.js

var homeIndexController = function ($scope) {
    $scope.data = [
        {
            MovieName: "Avatar",
            DirectorName: "James Cameron",
            ReleaseYear: "2009",
            Reviews:"3"
        },
        {
            MovieName: "Titanic",
            DirectorName: "James Cameron",
            ReleaseYear: "1997",
            Reviews: "30"
        },
        {
            MovieName: "Die Another Day",
            DirectorName: "Lee Tamahori",
            ReleaseYear: "2002",
            Reviews: "10"
        },
        {
            MovieName: "Godzilla",
            DirectorName: "Gareth Edwards",
            ReleaseYear: "2014",
            Reviews: "40"
        }
    ];
}
```

Now, in order to use the same I will modify my div section a bit so that it can presented in a well-defined format like shown below.

MovieReviewDatabaseInitializer.cs _Layout.cshtml homeIndex.js Index.cshtml

```
@section scripts
    {
    <script src="~/js/homeIndex.js"></script>
}
<div ng-controller="homeIndexController">

    <div class="row">

        <table class="table table-hover table-striped table-condensed table-bordered">
            <thead>
                <tr>
                    <th>Movie Name</th>
                    <th>Director Name</th>
                    <th>Release Year</th>
                    <th>Reviews</th>
                </tr>
            </thead>
            <tbody>
                <tr ng-repeat="i in data" class="repeat-animation">
                    <td>
                        {{i.MovieName}}
                    </td>
                    <td>
                        {{i.DirectorName}}
                    </td>
                    <td>
                        {{i.ReleaseYear}}
                    </td>
                    <td>
                        {{i.Reviews}}
                    </td>
                </tr>

            </tbody>
        </table>

    </div>
</div>
```

As you will notice in order to loop through the collection, I have used **ng-repeat** directive.




## Movie Reviews

| Movie Name | Director Name | Release Year | Reviews |
|---|---|---|---|
| Avatar | James Cameron | 2009 | 3 |
| Titanic | James Cameron | 1997 | 30 |
| Die Another Day | Lee Tamahori | 2002 | 10 |
| Godzilla | Gareth Edwards | 2014 | 40 |

## Retrieving Data from API:-

In this section we'll use **$http** object to make an API call to get the data from the server. As you can see the code here, it is an Async Call, however there is no guarantee that it will give you data back. Consider a situation where in server is down or some other reason API didn't respond then we can show some custom message like "***Couldn't Fetch the Data***". So, for this kind of scenario we will use promise. Here I have used **then**. **then** executes when **Get** is complete. **then** takes two different functions one when **Get** is successful and one when **Get** Fails.

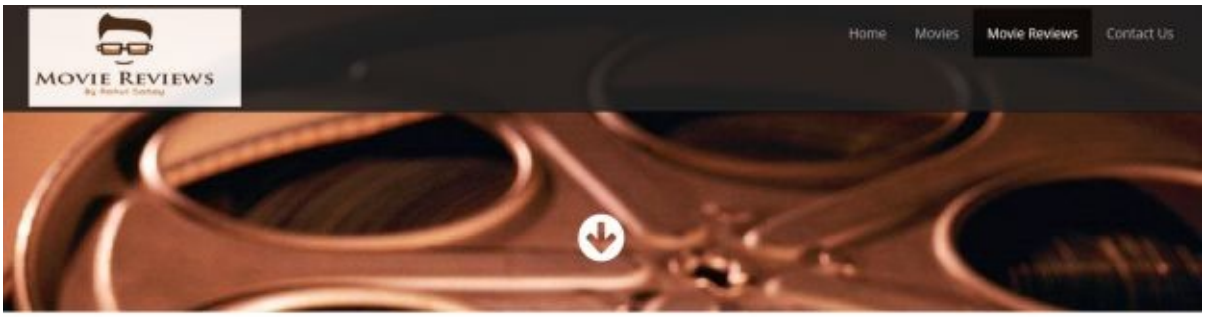
```
//homeindex.js

var homeIndexController = function ($scope, $http) {
    $scope.count = 0;

    //Empty Collection
    $scope.data = [];

    //API Call
    $http.get('http://localhost:65117/api/movies')
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, $scope.data);
    },
    function() {
    //Error
    //To Do:- Will change logging technique later using toastr lib
    console.log("Couldn't Fetch the Data");
    });
}
```

However, to get the reviews, I have changed field name in view as shown below in the snippet.

```
@section scripts
    {
    <script src="~/js/homeIndex.js"></script>
}
<div ng-controller="homeIndexController">

    <div class="row">
```

```
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
    <tr>
    <th>Movie Name</th>
    <th>Director Name</th>
    <th>Release Year</th>
    <th>Reviews</th>
    </tr>
    </thead>
    <tbody>
    <tr ng-repeat="i in data" class="repeat-animation">
    <td>
    {{i.MovieName}}
    </td>
    <td>
    {{i.DirectorName}}
    </td>
    <td>
    {{i.ReleaseYear}}
    </td>
    <td>
    {{i.NoOfReviews}}
    </td>
    </tr>

    </tbody>
    </table>

    </div>
</div>
```

Now, with the above changes in place. When I go ahead and refresh the page, then we will see the data from the server directly.




Movie Reviews

| Movie Name | Director Name | Release Year | Reviews |
|---|---|---|---|
| Godzilla | Gareth Edwards | 2014 | 1 |
| Avatar | James Cameron | 2009 | 0 |
| Titanic | James Cameron | 1997 | 0 |
| Die Another Day | Lee Tamahori | 2002 | 0 |

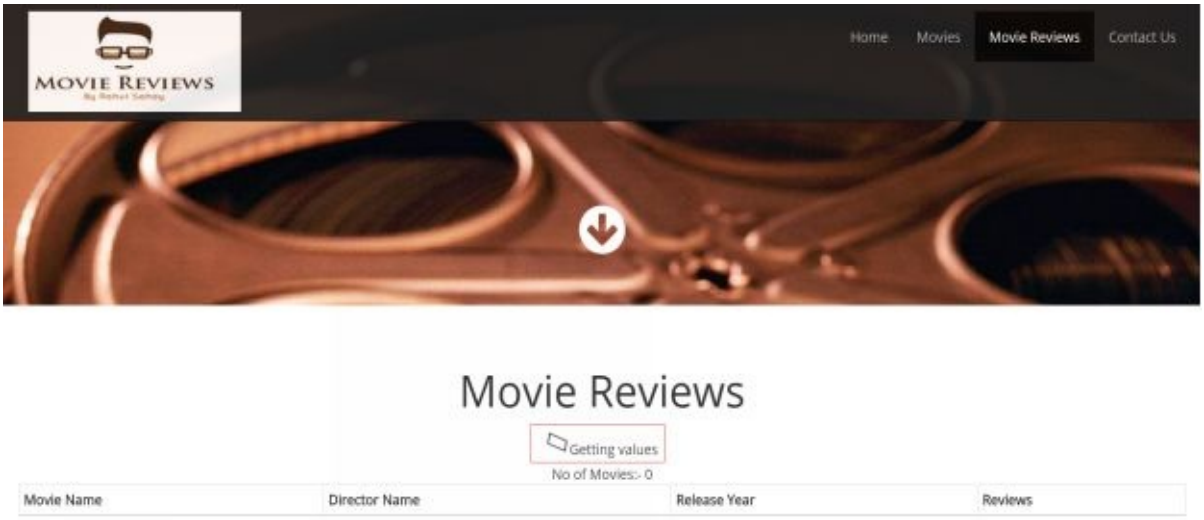
Now let's make one more thing here. Let's introduce spinner so that it will look as if it is getting data from server. So, for that I have made changes both in script and view as shown below

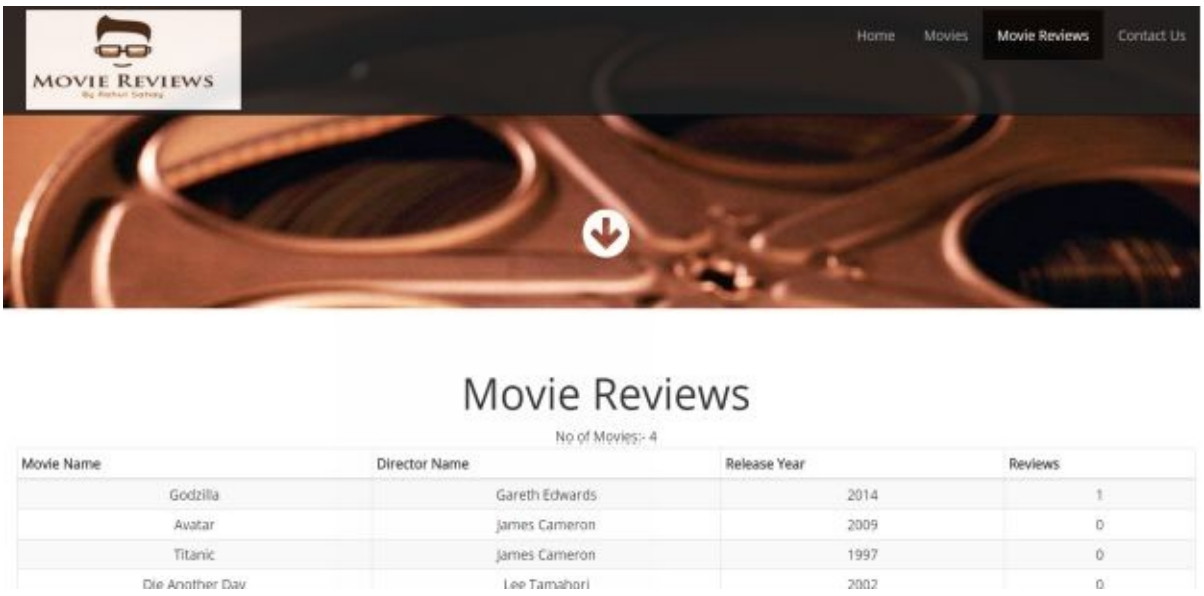
```
//homeindex.js

var homeIndexController = function($scope, $http) {

    //Empty Collection
    $scope.data = [];

    //Making Spinner On
    $('#loader').show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    $http.get('http://localhost:65117/api/movies')
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, $scope.data);
    },
    function() {
    //Error
    //To Do:- Will change logging technique later using toastr lib
    console.log("Couldn't Fetch the Data");
    })
    .then(function() {
```

```
    $('#loader').hide();
    });
    }, 1000);
}

@section scripts
    {
    <script src="~/js/homeIndex.js"></script>
}
<div ng-controller="homeIndexController">

    <div class="row">

    <div id="loader" style="display: none">
    <i class="fa-spinner fa-spin fa-2x"> </i> Getting values
    </div>

    <div>
    No of Movies:- {{data.length}}
    </div>
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
    <tr>
    <th>Movie Name</th>
    <th>Director Name</th>
    <th>Release Year</th>
    <th>Reviews</th>
    </tr>
    </thead>
    <tbody>
    <tr ng-repeat="i in data" class="repeat-animation">
    <td>
    {{i.MovieName}}
    </td>
    <td>
    {{i.DirectorName}}
    </td>
    <td>
    {{i.ReleaseYear}}
    </td>
    <td>
    {{i.NoOfReviews}}
```

```
    </td>
    </tr>

    </tbody>
    </table>

    </div>
</div>
```

Above I have introduced bootstrap spinner. This one I am toggling between the $http call. So as call begins I start to show the div, once call ends I simply hide the div.



### Summary:-

In this section, we have seen how to get started with angular. We have seen few basics of angular executed live on plnkr. We have also written our 1st controller and did some basic operation in that. After that we started doing some mature calls with controller like making http call to our APIs and showing the data back on the page. In the next section, we'll continue angular with more app driven functionalities.

## Chapter 5: Deeper into Angular JS

### WHAT DO you find in this CHAPTER?

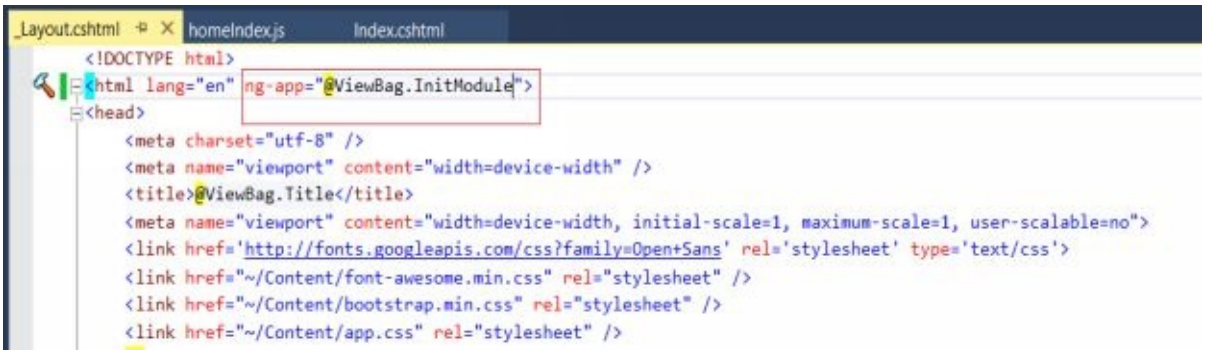
- Introduction
- Angular JS Routing
- Adding More Routes
- Client side validation
- Saving Data using Angular
- Creating Angular JS Service
- Creating Movie Edit Feature
- Creating Reviews Workflow
- Summary

### Introduction

In the last section, we have done basics of Angular JS. Now, in this section we will delve further to enhance our app by using features like **Modularity, Routing, Services** and many other things. So, without wasting time, let's get started.

### Angular JS Routing:-

Before jumping directly to routing, I would like to tell my app to load the module when my app gets loaded and in that module itself. For that, I will go ahead and write my routing logic. So, for doing this I'll 1st go to my layout page and set the **Viewbag** property with the module name, so that on every page this will get available as shown below.




And then from the index page I am going to call my module as shown below.

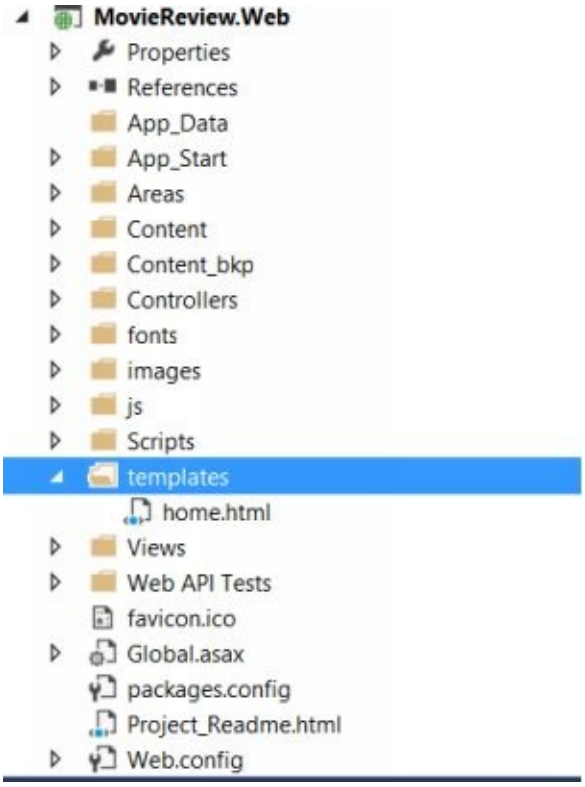
```
@{
    ViewBag.Title = “Home Page”;
    ViewBag.InitModule = “homeIndex”;
}
@section scripts
    {
    <script src=”~/js/homeIndex.js”></script>

}

<div ng-view=””></div>
```

So, up here, **InitModule** is setting my module **homeIndex** which I have created in my script file as shown below. Also, in case of angular **ng-view** is going to inject my required files between the divs. So, this is very much like **@RenderBody()** in MVC.

```
//homeindex.js

//Creating Module with the name homeIndex which takes empty config data for now
var module = angular.module(“homeIndex”, []);

//Route Configuration
module.config([“$routeProvider”,function($routeProvider) {
    $routeProvider.when(“/”, {
    controller: “homeIndexController”,
    templateUrl: “/templates/home.html”
    });

    //Default back to home page, if couldn’t find the path specified
$routeProvider.otherwise({ redirectTo: “/” });
```

```
}]);

var homeIndexController = [
    "$scope", "$http", function($scope, $http) {
    //Empty Collection
    $scope.data = [];

    //Making Spinner On
    $('#loader').show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    $http.get('http://localhost:65117/api/movies')
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, $scope.data);
    },
    function() {
    //Error
    //To Do:- Will change logging technique later using toastr lib
    console.log("Couldn't Fetch the Data");
    })
    .then(function() {
    $('#loader').hide();
    });
    }, 1000);
    }
];
```

Here in the above snippet; 1st I created the module then I simply added the route configuration to it. Now, route provider takes two parameters **when** and **otherwise.** When it matches the route, then it will look for the controller name and template path. Now, here I have created one new templates folder as shown below which contains my html file.

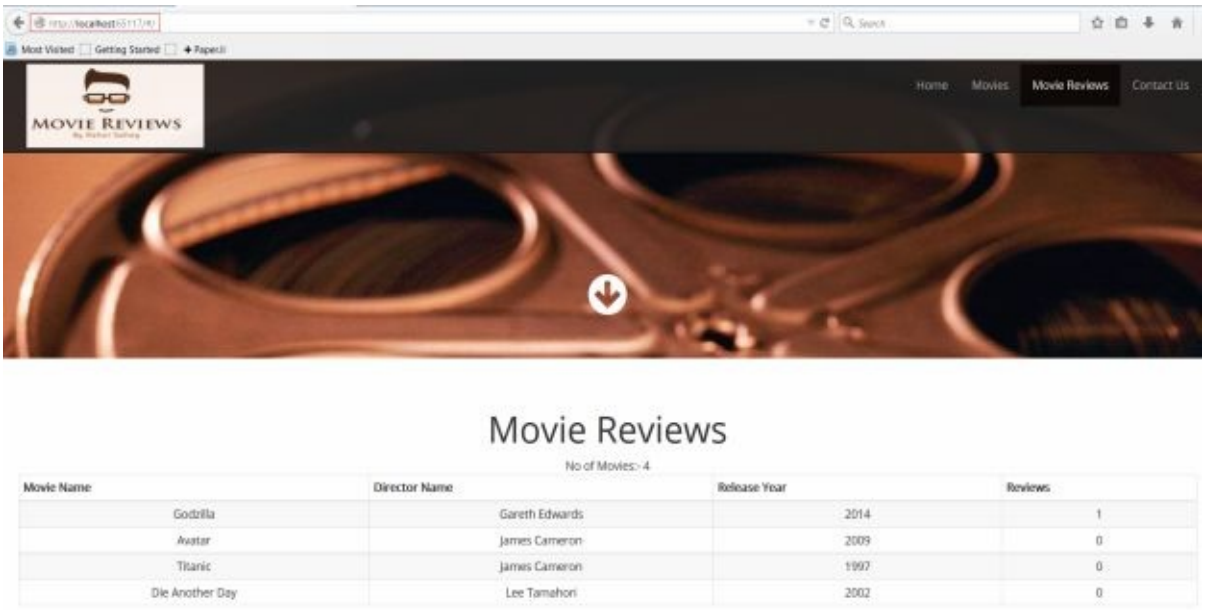



Now, if go ahead and see my html page, it contains only html fragments which is required to get injected at the specific place.

```
<div>

    <div class="row">

    <div id="loader" style="display: none">
    <i class="fa-spinner fa-spin fa-2x"> </i> Getting values
    </div>

    <div>
    No of Movies:- {{data.length}}
    </div>
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
    <tr>
    <th>Movie Name</th>
    <th>Director Name</th>
    <th>Release Year</th>
    <th>Reviews</th>
    </tr>
    </thead>
    <tbody>
```

```
    <tr ng-repeat="i in data" class="repeat-animation">
    <td>
    {{i.MovieName}}
    </td>
    <td>
    {{i.DirectorName}}
    </td>
    <td>
    {{i.ReleaseYear}}
    </td>
    <td>
    {{i.NoOfReviews}}
    </td>
    </tr>

    </tbody>
    </table>

    </div>
</div>
```

If you see the above snippet closely I have removed the controller call from html as it is getting done my module. Modularity helps to keep your code neat and clean. It's like separation of concerns where in you are placing different parts of the application in different modules. Also, in my script, I have everywhere used square brackets and put the dependencies in there as well. Basically, this is just to fix the minification means when you minify your file for production deployment, then in that case your code should not break. Now, with the above changes in place when I go ahead and refresh the app, I will not see any visual difference. However, it is now coming from modules and via routing.




Only difference is the **# declaration** in the URL bar. **Pound** (#) declaration in angular means inter page routing at the client side, hence it is not going to make any server side call. So, let's suppose I would to go a page which doesn't exist as of now, like **http://localhost:65117/#/newMovie** then it will simply redirect to original page.

## Adding More Routes:-

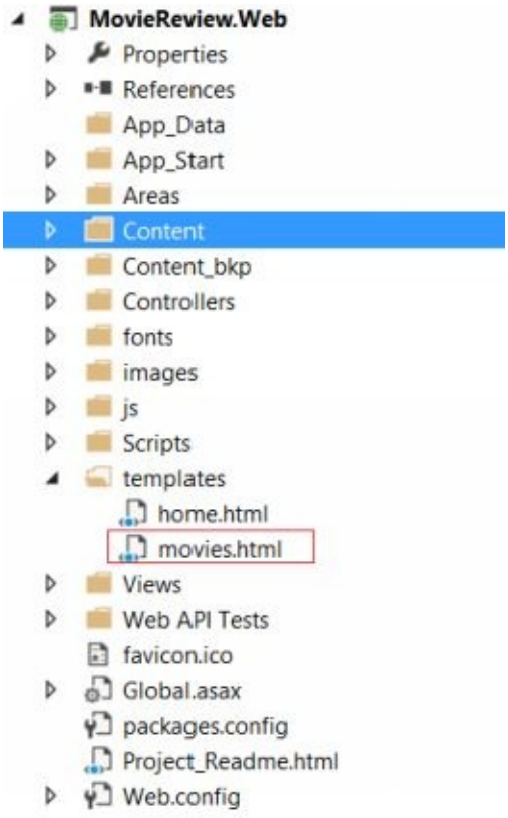
Now, in this section we are going to extend our routing mechanism to support more views. We will add movies template to list all the movies and will also contain a link to add a new movie.

```
//homeindex.js

//Creating Module with the name homeIndex which takes empty config data for now
var module = angular.module("homeIndex", []);

//Route Configuration
module.config(["$routeProvider",function($routeProvider) {
    $routeProvider.when("/", {
    controller: "homeIndexController",
    templateUrl: "/templates/home.html"
    });

    $routeProvider.when("/movies", {
```

```
    controller: “homeIndexController”,
    templateUrl: “/templates/movies.html”
    });


    //Default back to home page, if couldn’t find the path specified
    $routeProvider.otherwise({ redirectTo: “/” });
}]);

var homeIndexController = [
    “$scope”, “$http”, function($scope, $http) {
    //Empty Collection
    $scope.data = [];

    //Making Spinner On
    $(‘#loader’).show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    $http.get(‘http://localhost:65117/api/movies’)
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, $scope.data);
    },
    function() {
    //Error
    //To Do:- Will change logging technique later using toastr lib
    console.log(“Couldn’t Fetch the Data”);
    })
    .then(function() {
    $(‘#loader’).hide();
    });
    }, 1000);
    }
];
```

Here, in the above snippet, I have added one more route to support **movies** route. Now, let’s go ahead and refresh the page and click on the movies link.


After clicking the link when I opened the debugger tool in chrome, it gave me 404 error as it couldn't find movies template because I haven't created so far. So, let's go ahead and fix the same. As shown in the screen shot, I have added new template for **movies**.

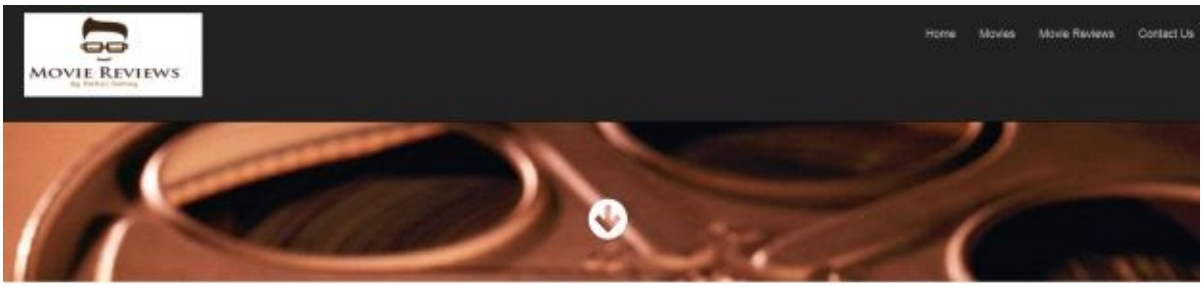



And the template code for the same will look like

```
<div id="loader" style="display: none">
    <i class="fa fa-cog fa-fw fa-spin fa-2x"> </i> Getting values
</div>
<br/>
<form class="form-horizontal" name="movieForm" role="form">
    <fieldset>
    <div class="panel panel-default">
    <!— Default panel contents —>
    <div class="panel-heading"><b>Movies</b></div>
    <div class="panel-body">
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
```

```
<tr>
<th>Movie Name</th>
<th>Director Name</th>
<th>Release Year</th>
<th>Edit</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr data-ng-repeat="i in data">
<td>
{{i.MovieName}}
</td>
<td>
{{i.DirectorName}}
</td>
<td>
{{i.ReleaseYear}}
</td>
<td>
<a data-ng-href="#/editMovie/{{i.Id}}" class="btn">Edit</a>
</td>
<td>
<a data-ng-href="#/reviews/{{i.Id}}">Reviews</a>
</td>
</tr>

</tbody>
</table>
</div>
</div>
</fieldset>
</form>
```

With the above change in place when I go ahead refresh the page it will look like




Movies

| Movie Name | Director Name | Release Year | Edit | Reviews |
|---|---|---|---|---|
| Godzilla | Gareth Edwards | 2014 | Edit | Reviews |
| Avatar | James Cameron | 2009 | Edit | Reviews |
| Titanic | James Cameron | 1997 | Edit | Reviews |
| Die Another Day | Lee Tamahori | 2002 | Edit | Reviews |

Here, in the last two columns I have given the flexibility to edit movie and then check and edit review as well. But, this piece I will code later. 1st let's add a feature to add a new movie. Below is the template after modification.

```
<div class="container">
    <button type="button" id="addMovie" value="addMovie">
    <i class="fa fa-plus btn btn-link"><a href="#/newMovie"> Add Movie</a> </i>
    </button>

</div>

<br/>

<div id="loader" style="display: none">
    <i class="fa fa-cog fa-fw fa-spin fa-2x"> </i> Getting values
</div>
<br/>
<form class="form-horizontal" name="movieForm" role="form">
    <fieldset>
    <div class="panel panel-default">
    <!— Default panel contents —>
    <div class="panel-heading"><b>Movies</b></div>
    <div class="panel-body">
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
    <tr>
    <th>Movie Name</th>
```

```
<th>Director Name</th>
<th>Release Year</th>
<th>Edit</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr data-ng-repeat="i in data">
<td>
{{i.MovieName}}
</td>
<td>
{{i.DirectorName}}
</td>
<td>
{{i.ReleaseYear}}
</td>
<td>
<a data-ng-href="#/editMovie/{{i.Id}}" class="btn">Edit</a>
</td>
<td>
<a data-ng-href="#/reviews/{{i.Id}}">Reviews</a>
</td>
</tr>

</tbody>
</table>
</div>
</div>
</fieldset>
</form>
```

Up in the above snippet. I have added one container which contains the link for adding a new movie. Now, let's go ahead and add template in the templates folder with the HTML markup.

```
<form class="form-horizontal" name="newMovieForm" novalidate role="form">
<fieldset>
<legend>Enter New Movie</legend>
<div class="form-group">
```

```
    <label for="MovieName" class="col-sm-3 control-label">Movie Name</label>
    <div class="col-sm-9">
    <input type="text" id="MovieName" name="MovieName" class="form-control" />
    </div>
    </div>

    <div class="form-group">
    <label for="DirectorName" class="col-sm-3 control-label">Director Name</label>
    <div class="col-sm-9">
    <input type="text" id="DirectorName" name="DirectorName" class="form-control"/>
    </div>
    </div>

    <div class="form-group">
    <label for="ReleaseYear" class="col-sm-3 control-label">Release Year</label>
    <div class="col-sm-9">
    <input type="text" id="ReleaseYear" name="ReleaseYear" class="form-control"/>
    </div>
    </div>

    <div class="container">
    <button type="button" id="addNewMovie" value="addNewMovie">
    <i class="fa fa-plus btn btn-primary"> Add Movie</i>
    </button>

    <button type="button" id="cancelMovie" value="cancelMovie">
    <i class="fa fa-undo btn btn-link"><a href="#/movie"> Cancel</a></i>
    </button>
    </div>
    </fieldset>
</form>
```

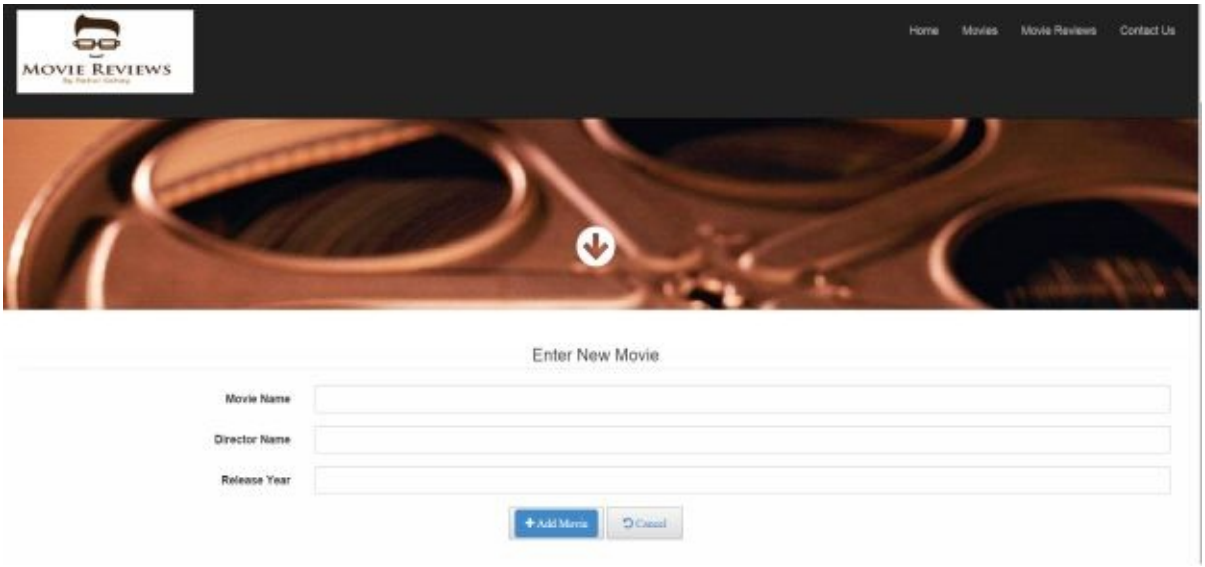
Here, above markup is for adding a new movie and below I have simply added one route so that this page can be navigated. However, I have still not written **newMovieController**, but still it should show me the page.

```
//homeindex.js

//Creating Module with the name homeIndex which takes empty config data for now
var module = angular.module("homeIndex", []);
```

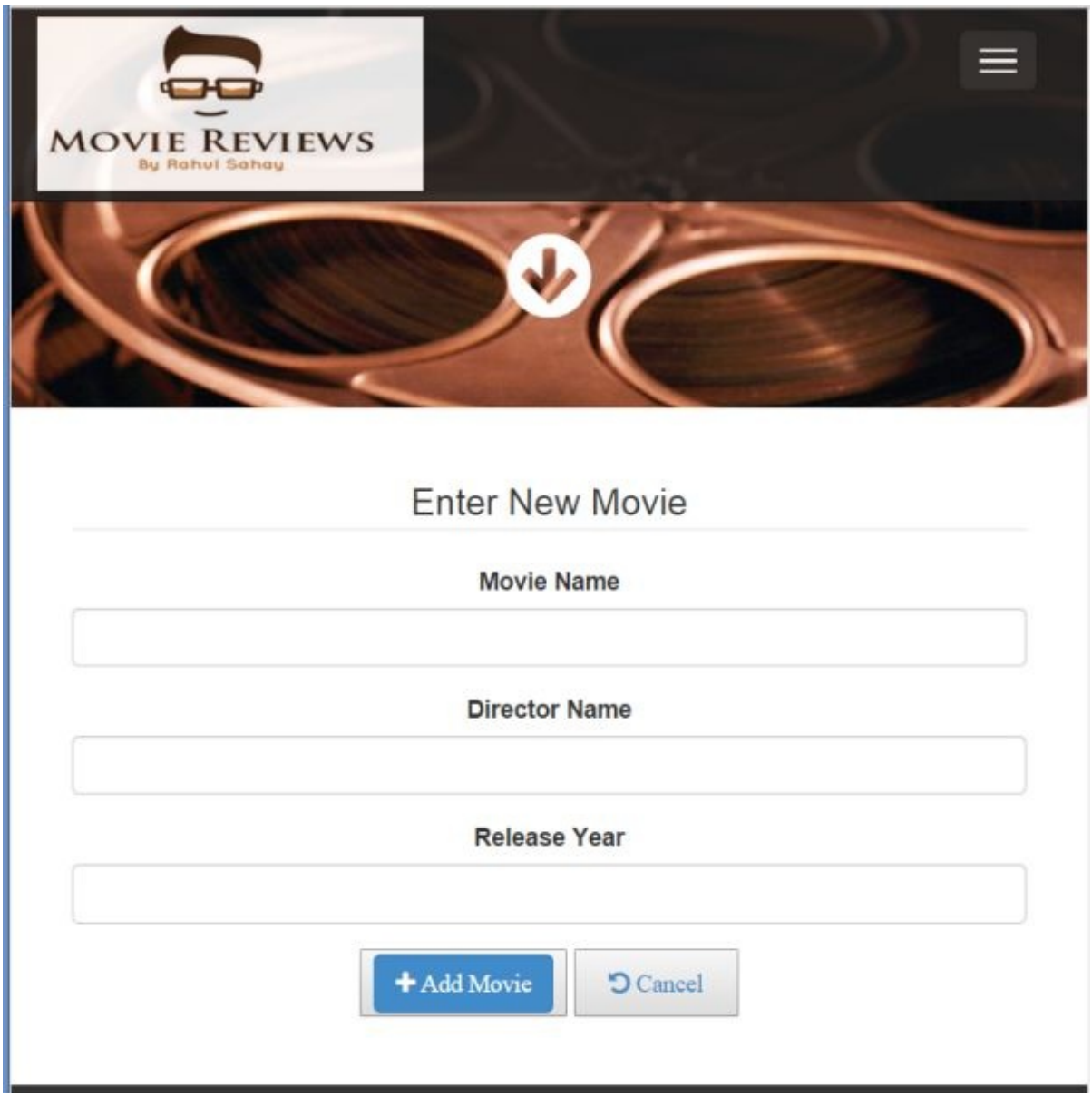
```
//Route Configuration
module.config(["$routeProvider",function($routeProvider) {
    $routeProvider.when("/", {
    controller: "homeIndexController",
    templateUrl: "/templates/home.html"
    });

    $routeProvider.when("/movies", {
    controller: "homeIndexController",
    templateUrl: "/templates/movies.html"
    });

    $routeProvider.when("/newMovie", {
    controller: "newMovieController",
    templateUrl: "/templates/newMovie.html"
    });

    //Default back to home page, if couldn't find the path specified
    $routeProvider.otherwise({ redirectTo: "/" });
}]);

var homeIndexController = [
    "$scope", "$http", function($scope, $http) {
    //Empty Collection
    $scope.data = [];

    //Making Spinner On
    $('#loader').show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    $http.get('http://localhost:65117/api/movies')
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, $scope.data);
    },
    function() {
    //Error
    //To Do:- Will change logging technique later using toastr lib
    console.log("Couldn't Fetch the Data");
    })
```

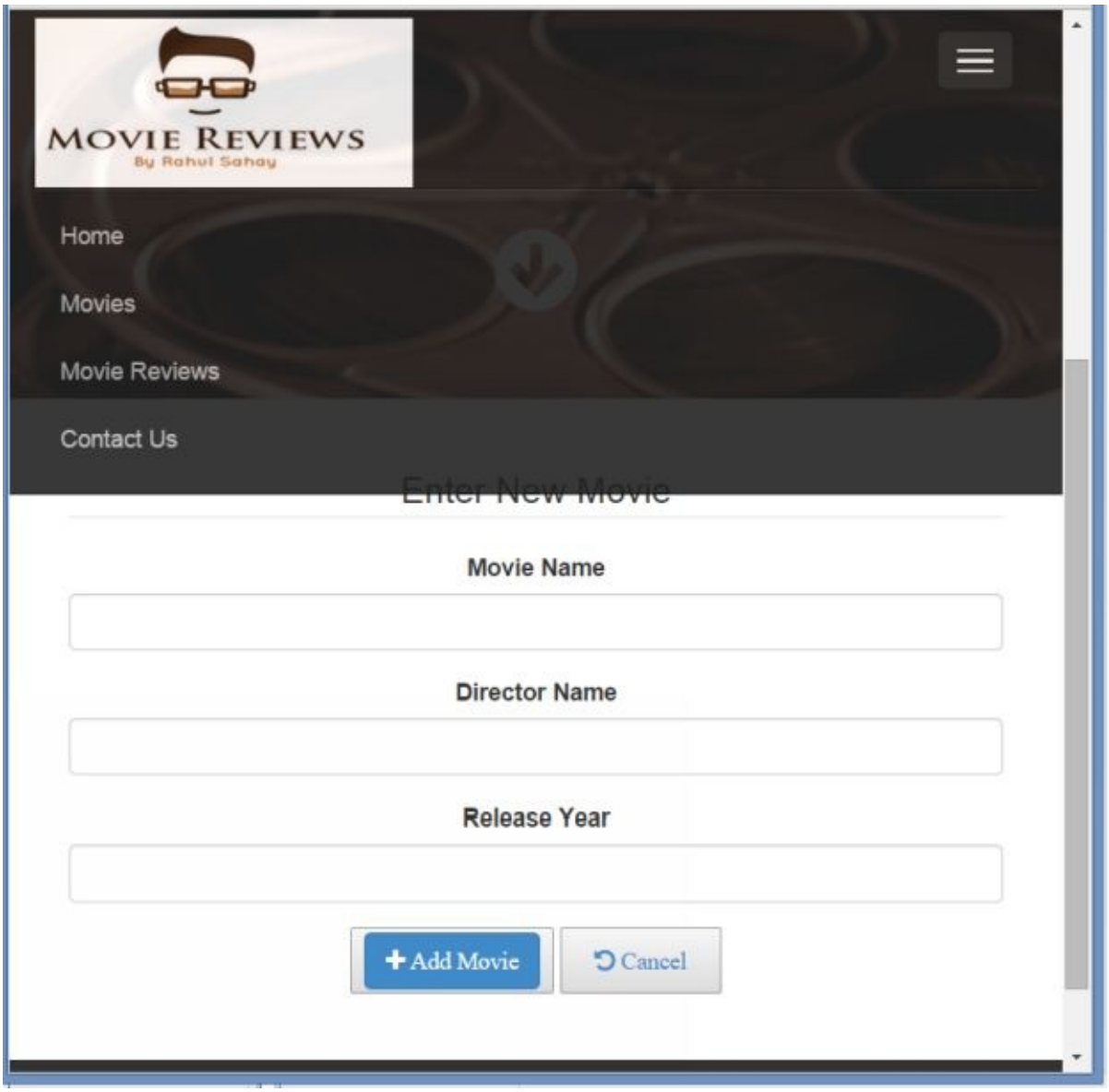
```
    .then(function() {
    $('#loader').hide();
    });
    }, 1000);
    }
];
```

Now, with the above change in place when I go ahead click on Add Movie link from the movies page, then it will take me to new movie page as shown below in the screen shot.



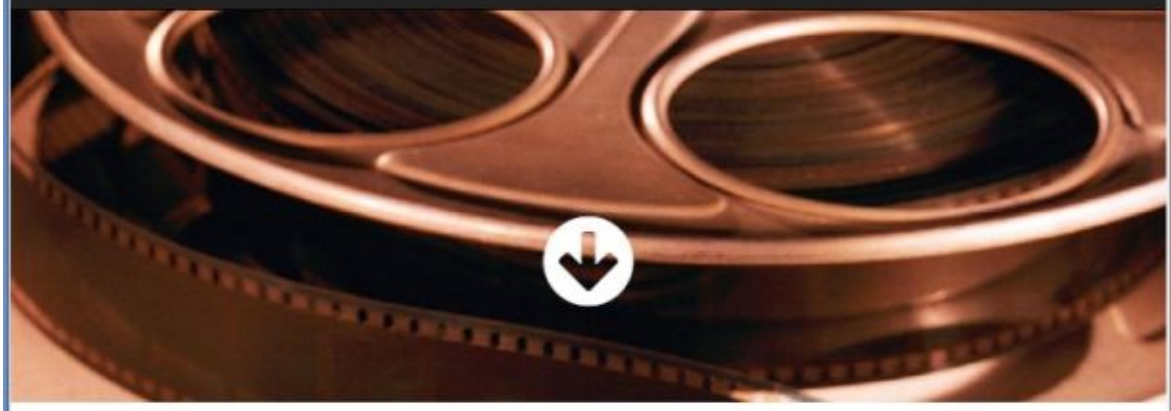

Currently, Add Movie will do nothing as I have not written this piece of code yet. However, while clicking cancel button, it will take us to the root of the application means on the home page. One more thing to note here, all the coding standard which I am following is bootstrap standard. You can get more details on responsive templates and how to use the same on http://getbootstrap.com link. However, our app is completely responsive, hence it will work on any device on any resolution. Every component whether menus, forms images… everything get optimized based on the change as shown below in the screen shots.


MOVIE REVIEWS
By Rahul Sahay
Enter New Movie
Movie Name
Director Name
Release Year
Add Movie
Cancel


MOVIE REVIEWS
By Rahul Sahay
Home
Movies
Movie Reviews
Contact Us
Enter New Movie
Movie Name
Director Name
Release Year
Add Movie
Cancel

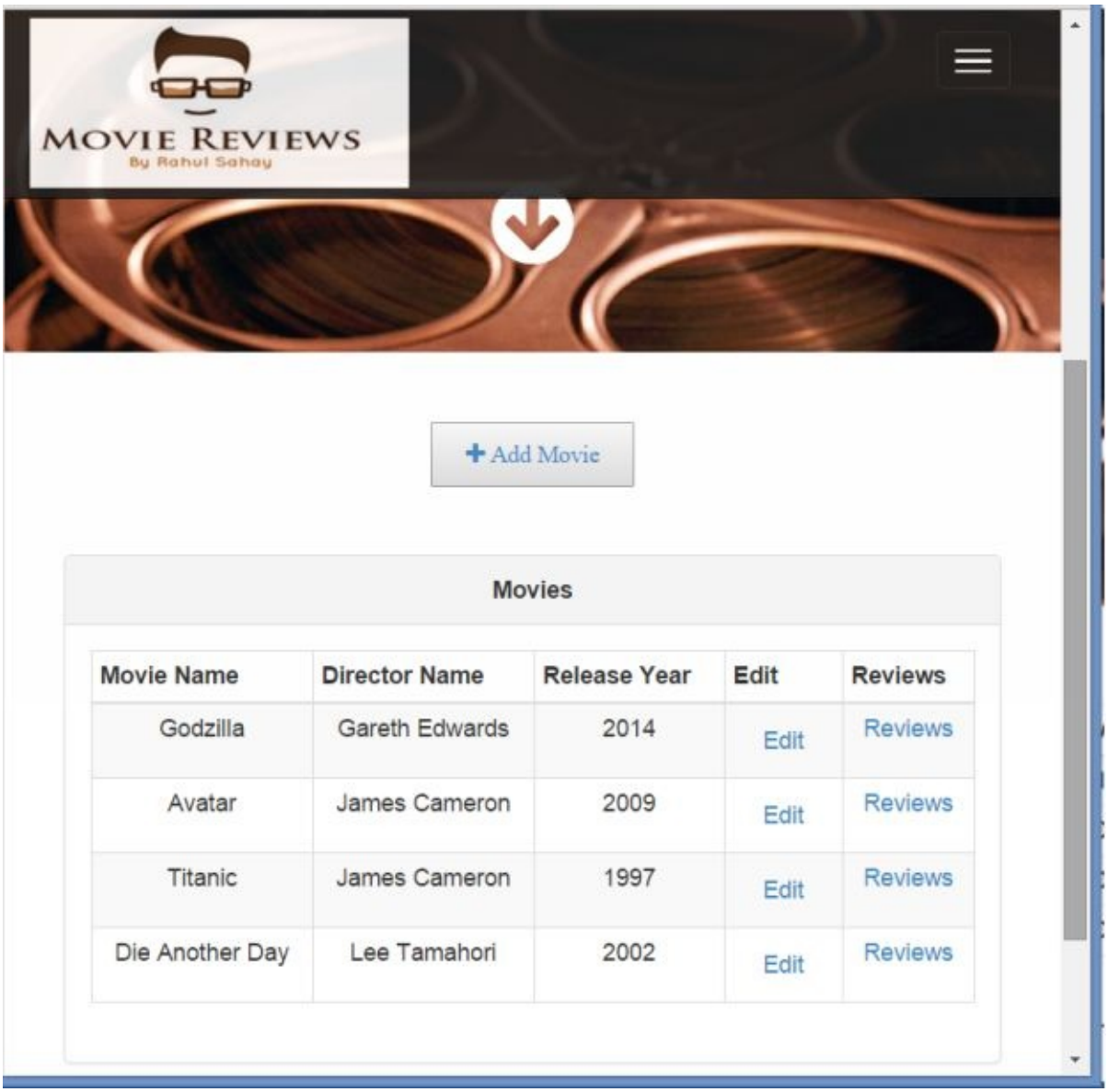
MOVIE REVIEWS

By Rahul Sahay



No of Movies:- 4

| Movie Name | Director Name | Release Year | Reviews |
|---|---|---|---|
| Godzilla | Gareth Edwards | 2014 | 1 |
| Avatar | James Cameron | 2009 | 0 |
| Titanic | James Cameron | 1997 | 0 |
| Die Another Day | Lee Tamahori | 2002 | 0 |




So, it doesn't matter on which device you are seeing this site, it will open in its best shape.

## Client side validation:-

Now, let's go ahead and do some client validation in form itself, so that data which we will be sending back to the server will be legitimate. It doesn't mean that server is not going to validate the data, it will validate but idea of doing angular is do the heavy lifting at client side itself.

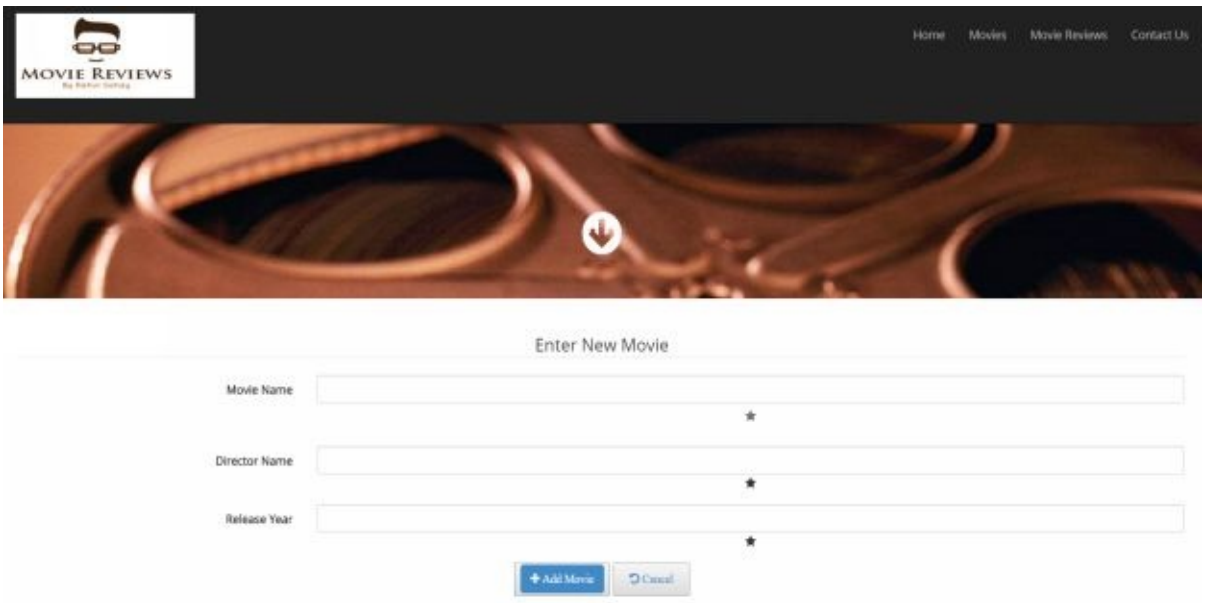
As you can see below in the snippet, 1st thing I have added in the form is **novalidate** which tells the browser don't do any built in validation, we will do the same via code.

```
<form class="form-horizontal" name="newMovieForm" novalidate role="form" ng-submit="save()">
    <fieldset>
```

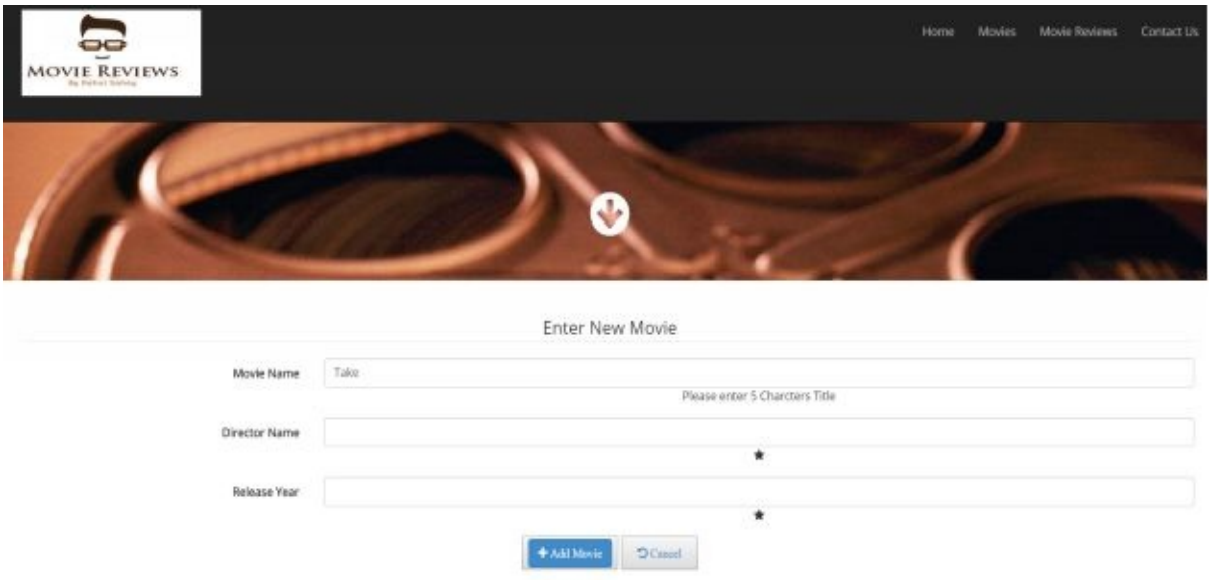
```
<legend>Enter New Movie</legend>
<div class="form-group">
<label for="MovieName" class="col-sm-3 control-label">Movie Name</label>
<div class="col-sm-9">
<input type="text" id="MovieName" name="MovieName" class="form-control" ng-model="newMovie.MovieName" required ng-minlength="5" />
<span class="help-block" ng-show="newMovieForm.MovieName.$error.required"><i class="fa fa-star"></i></span>
<span class="input-validation-error" ng-show="newMovieForm.MovieName.$error.minlength">Please enter 5 Charcters Title</span>
</div>
</div>

<div class="form-group">
<label for="DirectorName" class="col-sm-3 control-label">Director Name</label>
<div class="col-sm-9">
<input type="text" id="DirectorName" name="DirectorName" class="form-control" ng-model="newMovie.DirectorName" required ng-minlength="5" />
<span class="input-validation-error" ng-show="newMovieForm.DirectorName.$error.required"><i class="fa fa-star"></i></span>
<span class="input-validation-error" ng-show="newMovieForm. DirectorName.$error.minlength">Please enter 5 Charcters Name</span>
</div>
</div>

<div class="form-group">
<label for="ReleaseYear" class="col-sm-3 control-label">Release Year</label>
<div class="col-sm-9">
<input type="text" id="ReleaseYear" name="ReleaseYear" class="form-control" ng-model="newMovie.ReleaseYear" required />
<span class="input-validation-error" ng-show="newMovieForm.ReleaseYear.$error.required"><i class="fa fa-star"></i></span>
</div>
</div>

<div class="container">
<button type="submit" id="addNewMovie" value="addNewMovie">
<i class="fa fa-plus btn btn-primary"> Add Movie</i>
</button>

<button type="button" id="cancelMovie" value="cancelMovie">
<i class="fa fa-undo btn btn-link"><a href="#/movie"> Cancel</a></i>
</button>
</div>
```

```
    </fieldset>
</form>
```

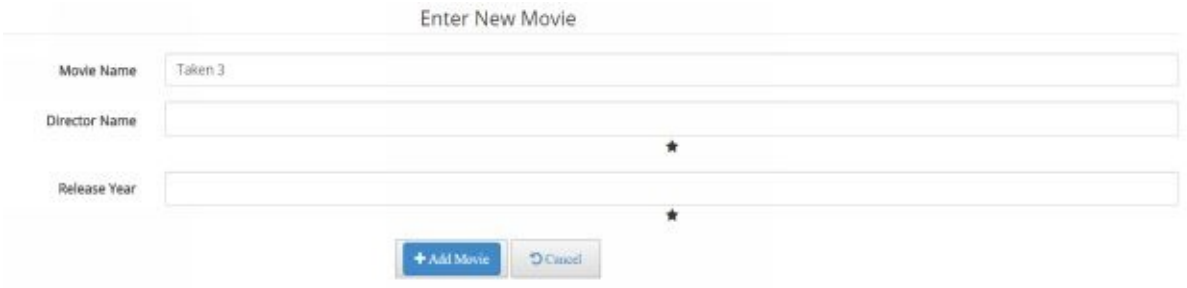
Now, here for data binding I am using **ng-model**. Now, there is a specific reason for that because **ng-model** supports two way binding. However, **{{}}** curly braces binding is one way basically for showing values only. Two way data binding can take your value and send it back to the server. Also, I have used **span** tags to emit warning message and required fields. Here, required is an attribute to make the field mandatory. So, with the above changes in place when I go ahead and load the page, it will look like as shown below.




If I type some movie name then it will prompt the message as shown below and once it gets satisfied, then it will get disappear.



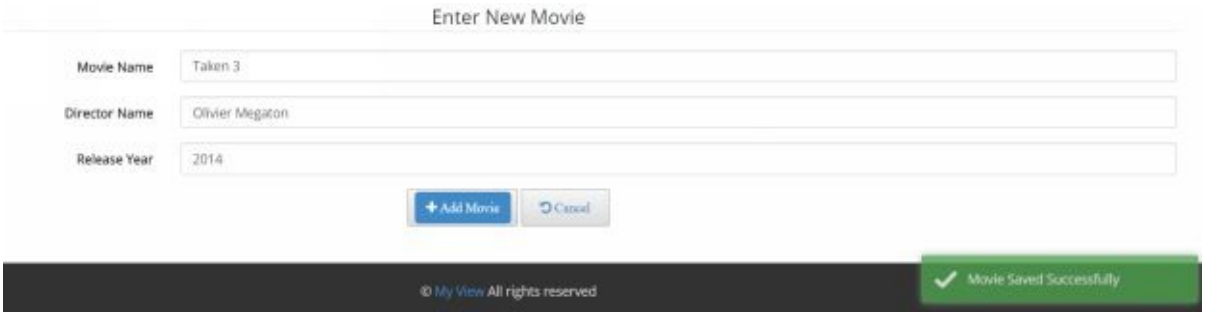
Now, once I complete the required thing, then that message will disappear as shown below.




Same behavior implies similarly for other fields also. However, if you notice closely, **Add Movie** button is enabled even when form is invalid. To fix this, I can use angular button disabled feature as shown below.

```
<button type="submit" id="addNewMovie" value="addNewMovie" ng-disabled="newMovieForm.$invalid">
  <i class="fa fa-plus btn btn-primary"> Add Movie</i>
  </button>
```

With these changes in place, when I go ahead and hit the Add Movie button, it won't submit until form is valid. Now, let's submit the form with some valid data as shown below.




Upon successful submission, one toast message appeared with the message. For this toast message, I have used toastr library. You can download the same from nuget package manager. However, toastr library is used for showing notifications. Below is the controller code for the same.

```
var newMovieController = [
    "$scope", "$http", "$window", function($scope, $http, $window) {
    $scope.newMovie= {};
    $scope.save= function() {
    toastr.success("Movie Saved Successfully");
    }
    }
```

```
];
```

However, currently in this controller I have not written data saving logic. Let's complete the same and test again.

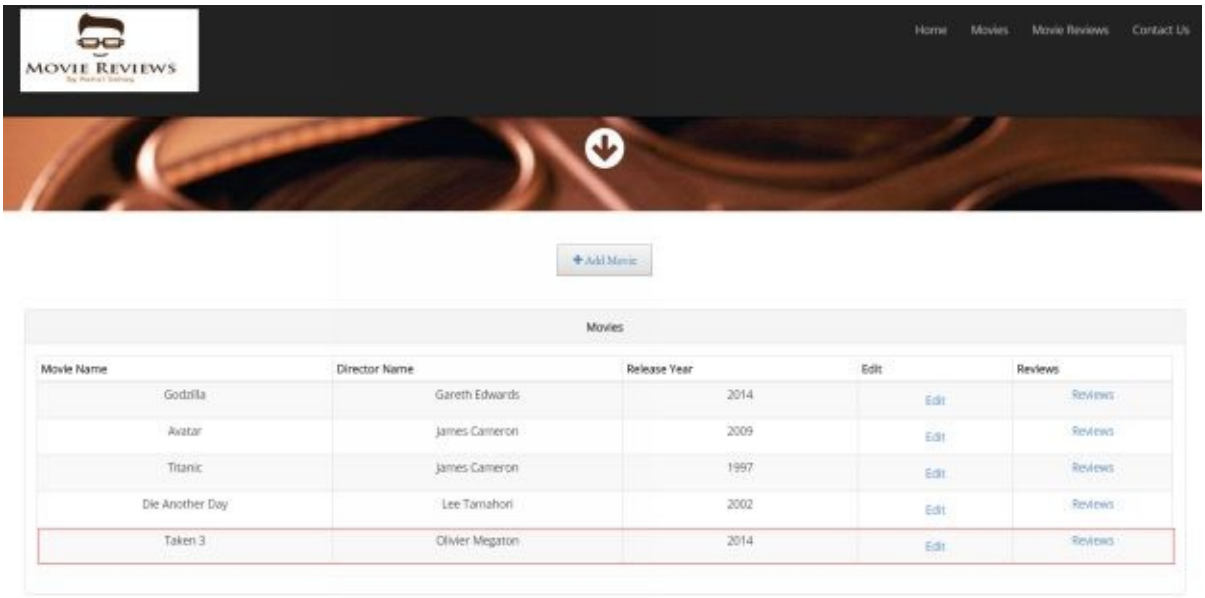
## Saving Data using Angular:-

In this section, we are going to complete the newMovieController by completing the save logic. Below, I have completed my logic for saving the new movie.

```
var newMovieController = [
    "$scope", "$http", "$window", function($scope, $http, $window) {
    $scope.newMovie = {};
    $scope.save = function () {

    $http.post('http://localhost:65117/api/movies', $scope.newMovie)
    .then(function(result) {
    //Success
    var newMovie = result.data;
    toastr.success("Movie Saved Successfully");

    //Once Saved successfully return to the movies page
    $window.location = "#/movies";
    },
    function() {
    //Error
    toastr.error("Unable to Save the Movie");
    });
    }
    }
];
```

Let me explain the logic in brief. Here, I have introduced one more parameter in the function that is **$window**. **$window,** I have used here for redirecting the page back to the movies page, once data saved successfully. Then my **$scope.newMovie** is the collection of **ng-model** which I have taken from the form. Then, **$http.Post** takes the API URL with the collection. Once data saved successfully, it will return the data with the ID generated from the server else simply return error message.

With the above change in place, let's go ahead and try saving a new movie.


## Creating Angular JS Service:-

Let's go ahead and create angular service which will be having Single Repository for doing all CRUD operations like **Creating/Updating/Deleting** record. Below is the snippet of modified code with angular service in place. In the below snippet you can see that I have created the service with **angular.factory()** method by providing the required service name and required dependencies need to be executed in the service.

```
//homeindex.js

//Creating Module with the name homeIndex which takes empty config data for now
var module = angular.module("homeIndex", ['ui.bootstrap']);

//Route Configuration
module.config(["$routeProvider",function($routeProvider) {
    $routeProvider.when("/", {
    controller: "homeIndexController",
```

```
    templateUrl: “/templates/home.html”
    });

    $routeProvider.when(“/movies”, {
    controller: “homeIndexController”,
    templateUrl: “/templates/movies.html”
    });

    $routeProvider.when(“/newMovie”, {
    controller: “newMovieController”,
    templateUrl: “/templates/newMovie.html”
    });

    //Default back to home page, if couldn’t find the path specified
    $routeProvider.otherwise({ redirectTo: “/” });
}]);

module.factory(“dataService”, [“$http”,“$q”,function($http,$q) {

    var _movies = [];

    //function to retrieve movies
    var _getMovies = function () {

    //For resolving the promise
    var deferred = $q.defer();
    $http.get(‘http://localhost:65117/api/movies’)
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, _movies);
    deferred.resolve();
    },
    function() {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    };
    //make available below properties for other parts of angular to use
    return {
    movies: _movies,
```

```
    getMovies:_getMovies
    };
}]);
var homeIndexController = [
    "$scope", "$http", "dataService", function ($scope, $http, dataService) {
    //Empty Collection
    $scope.data = dataService;

    //Making Spinner On
    $('#loader').show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    dataService.getMovies()
    .then(function() {
    //success
    toastr.success("Movies Fetched Successfully");
    },function() {
    //error
    toastr.error("Error in fetching movies");
    })
    .then(function() {
    $('#loader').hide();
    });
    }, 1000);
    }
];

var newMovieController = [
    "$scope", "$http", "$window", function ($scope, $http, $window) {

    $scope.newMovie = {};
    $scope.save = function () {

    $http.post('http://localhost:65117/api/movies', $scope.newMovie)
    .then(function(result) {
    //Success
    var newMovie = result.data;
    toastr.success("Movie Saved Successfully");

    //Once Saved successfully return to the movies page
    $window.location = "#/movies";
```

```
    //To Do:- Return the newly added movie to the existing collection
    },
    function() {
    //Error
    toastr.error("Unable to Save the Movie");
    });
    }
    }
];
```

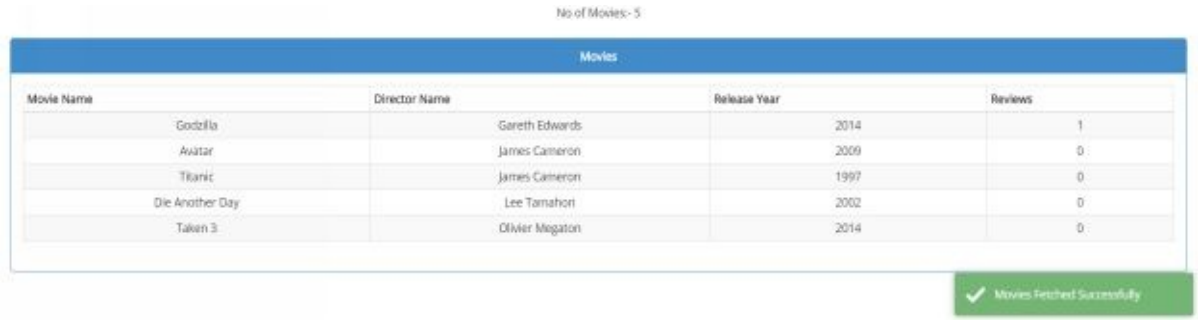
Now, $q is a special parameter which allows us to create a deferrable object. A deferrable object can expose a promise so that the consumer of **_getMovies** function can use the promise like syntax like what needs to be done when success and failure. So, here when $q successfully fetching the data it is just passing down the layer with promise **resolve** and when not able to fetch the same then simply passing the message with **reject**.

Hence, **dataService's** job is only to deal with API calls and fetch data, other pieces like updating UI, passing the notification is delegated to controller code itself. You may also notice in the controller code, I have injected **dataService** just to make sure that we can go ahead use the **dataService APIs**. Also, I have copied data as **dataService** to the scope object, hence I need to modify my HTML code as well.

```
<div>

    <div class="row">

    <div id="loader" style="display: none">
    <i class="fa fa-cog fa-fw fa-spin fa-2x"> </i> Getting Movies
    </div>

    <div>
    No of Movies:- {{data.movies.length}}
    </div>
    <br/>
    <form class="form-horizontal" name="homeMovieForm" role="form">
    <fieldset>
    <div class="panel panel-primary">
    <!— Default panel contents —>
    <div class="panel-heading"><b>Movies</b></div>
    <div class="panel-body">
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
```

```
<tr>
<th>Movie Name</th>
<th>Director Name</th>
<th>Release Year</th>
<th>Reviews</th>
</tr>
</thead>
<tbody>
<tr ng-repeat="i in data.movies">
<td>
{{i.MovieName}}
</td>
<td>
{{i.DirectorName}}
</td>
<td>
{{i.ReleaseYear}}
</td>
<td>
{{i.NoOfReviews}}
</td>
</tr>

</tbody>
</table>
</div>
</div>

</fieldset>
</form>
</div>
</div>
```

Here, in the above snippet, I went down one layer and bind to **data.movies.** Remember, we have returned in **movies** property the required data in **dataService**. With the above change in place, when I go ahead and refresh my page, I should see the same result, but this time fetching data via service.

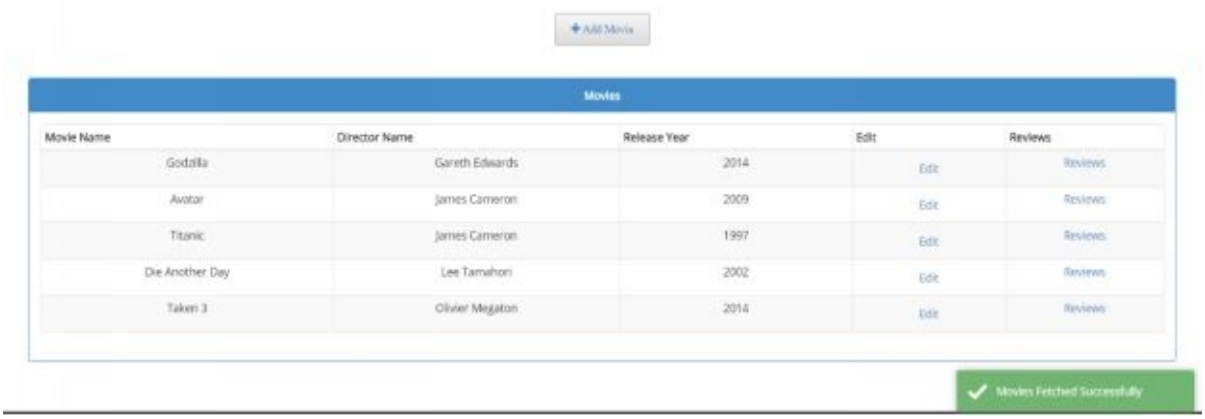



Now, similar kind of HTML change is required in the movies template as well. Below is modified snippet after the change.

```
<div class=“container”>
    <button type=“button” id=“addMovie” value=“addMovie”>
    <i class=“fa fa-plus btn btn-link”><a href=”#/newMovie”> Add Movie</a> </i>
    </button>

</div>

<br/>

<div id=“loader” style=”display: none”>
    <i class=“fa fa-cog fa-fw fa-spin fa-2x”> </i> Getting Movies
</div>
<br/>
<form class=“form-horizontal” name=“movieForm” role=“form”>
    <fieldset>
    <div class=“panel panel-primary”>
    <!— Default panel contents —>
    <div class=“panel-heading”><b>Movies</b></div>
    <div class=“panel-body”>
    <table class=“table table-hover table-striped table-condensed table-bordered”>
    <thead>
    <tr>
    <th>Movie Name</th>
    <th>Director Name</th>
    <th>Release Year</th>
    <th>Edit</th>
    <th>Reviews</th>
    </tr>
    </thead>
    <tbody>
    <tr data-ng-repeat=“i in data.movies”>
```

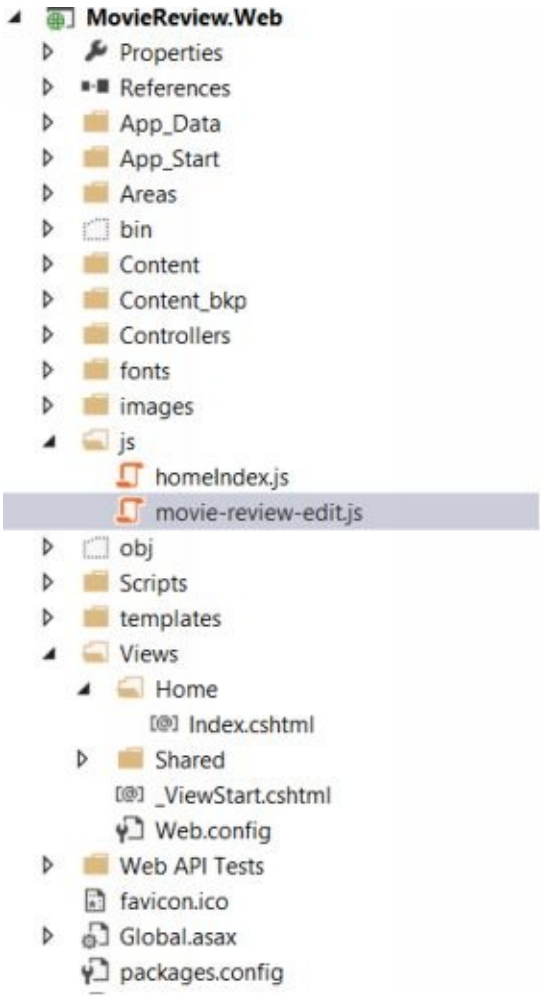
```
    <td>
    {{i.MovieName}}
    </td>
    <td>
    {{i.DirectorName}}
    </td>
    <td>
    {{i.ReleaseYear}}
    </td>
    <td>
    <a data-ng-href="#/editMovie/{{i.Id}}" class="btn">Edit</a>
    </td>
    <td>
    <a data-ng-href="#/reviews/{{i.Id}}">Reviews</a>
    </td>
    </tr>

    </tbody>
    </table>
    </div>
    </div>
    </fieldset>
</form>
```

With the above change in place, when I refresh movies link, I should see the same result as shown below.




## Creating Movie Edit Feature:-

In this section we will extend our existing logic and use the same for editing Movies. Idea is when user clicks on **Edit** corresponding link of **Movie**, user should be able to **delete/modify** the movie. Also, for doing any update part whether its movie or reviews I would like to create a separate JS file.

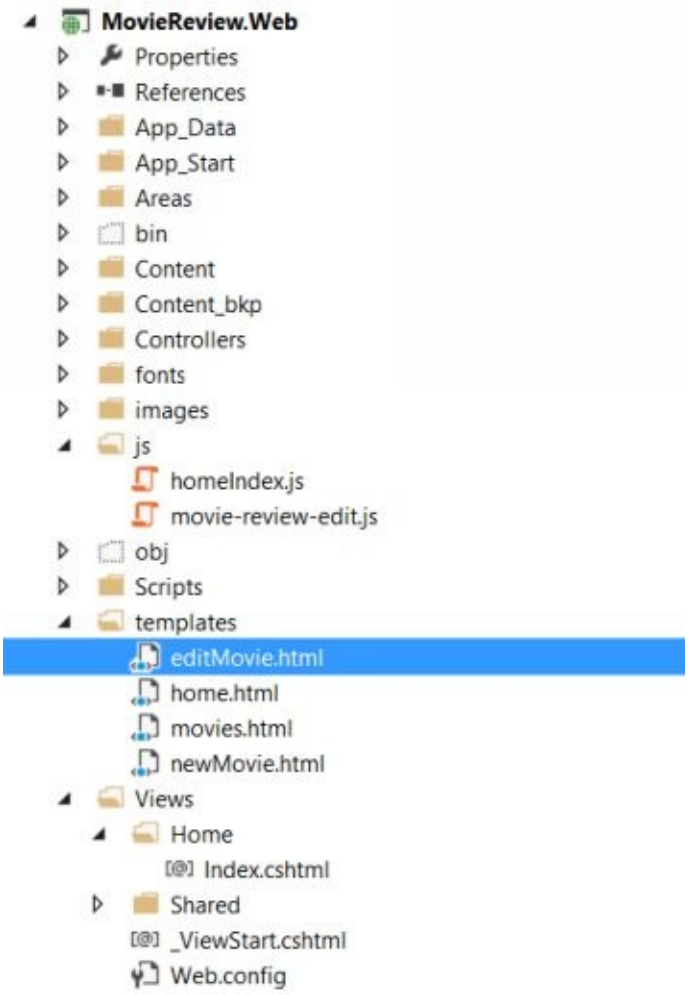



Now, let's go ahead and add my logic to it. Below is the straight forward snippet where I begin by creating module and then adding route to it.

```
//movie-review-edit.js

//Defined Module
var module = angular.module("movieReviewEdit", []);

//Defined Routes
module.config(["$routeProvider", function ($routeProvider) {

    $routeProvider.when("/editMovie/:Id", {
    controller: "movieEditController",
    templateUrl: "/templates/editMovie.html"
    });
```

```
    $routeProvider.otherwise({ redirectTo: "/movies" });
}]);
```

However, one point to notice here I have used **:Id** in the route path. This is done to identify the associated Id for the movie, so that I can make a get request to fetch the details based on this Id and then I can go ahead and edit the record. I also need to create **editMovie** template to edit the record. So, let's go ahead and create the same.




But, in order to use the new JS file, we need to tell angular that this file is having dependency for doing any update. Hence, my new module is dependent on the existing module. So, easy way to wire up dependency via dependency injection as shown below.

```
//Created Module with the name homeIndex and wired up required dependencies
var module = angular.module("homeIndex", ['ui.bootstrap', 'movieReviewEdit']);
```

Here, **movieReviewEdit** is my new module and other one is the bootstrap library which I'll be using later for some beautification in UI. Also, we need to include this JS file in index.cshtml page as shown below.

```
@{
    ViewBag.Title = “Home Page”;
    ViewBag.InitModule = “homeIndex”;
}
@section scripts
    {
    <script src=”~/js/homeIndex.js”></script>
    <script src=”~/js/movie-review-edit.js”></script>
}

<div ng-view=””></div>
```

Now, 1st thing is required to fetch the movie based on the Id, so that it will be available for editing. Let’s do that now. So, the very 1st thing which I have done is I have added the **getMovieById** in the service part as shown below

```
//homeindex.js

//Created Module with the name homeIndex and wired up required dependencies
var module = angular.module(“homeIndex”, [‘ui.bootstrap’, ‘movieReviewEdit’]);

//Route Configuration
module.config([“$routeProvider”,function($routeProvider) {
    $routeProvider.when(“/”, {
    controller: “homeIndexController”,
    templateUrl: “/templates/home.html”
    });

    $routeProvider.when(“/movies”, {
    controller: “homeIndexController”,
    templateUrl: “/templates/movies.html”
    });

    $routeProvider.when(“/newMovie”, {
    controller: “newMovieController”,
    templateUrl: “/templates/newMovie.html”
    });

    //Default back to home page, if couldn’t find the path specified
    $routeProvider.otherwise({ redirectTo: “/” });
```

```
}]);

module.factory("dataService", ["$http","$q",function($http,$q) {

    var _movies = [];

    //function to retrieve movies
    var _getMovies = function () {

    //For resolving the promise
    var deferred = $q.defer();
    $http.get('http://localhost:65117/api/movies')
    .then(function(result) {
    //Success
    //angular.copy copies the collection from source to destination
    angular.copy(result.data, _movies);
    deferred.resolve();
    },
    function() {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    };

    //Fetch Movie By Id
    var _getMovieById = function(Id) {
    var deferred = $q.defer();
    $http.get('http://localhost:65117/api/movies/' + Id)
    .then(function(result) {
    //Success
    //result.data will return the data back to the caller
    deferred.resolve(result.data);
    }, function() {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    }
    //make available below properties for other parts of angular to use
    return {
    movies: _movies,
```

```
    getMovies: _getMovies,
    getMovieById: _getMovieById
    };
}]);
var homeIndexController = [
    "$scope", "$http", "dataService", function ($scope, $http, dataService) {
    //Empty Collection
    $scope.data = dataService;

    //Making Spinner On
    $('#loader').show();
    //API Call
    //Timeout function to show spinner
    setTimeout(function() {
    dataService.getMovies()
    .then(function() {
    //success
    toastr.success("Movies Fetched Successfully");
    },function() {
    //error
    toastr.error("Error in fetching movies");
    })
    .then(function() {
    $('#loader').hide();
    });
    }, 1000);
    }
];

var newMovieController = [
    "$scope", "$http", "$window", function ($scope, $http, $window) {

    $scope.newMovie = {};
    $scope.save = function () {

    $http.post('http://localhost:65117/api/movies', $scope.newMovie)
    .then(function(result) {
    //Success
    var newMovie = result.data;
    toastr.success("Movie Saved Successfully");

    //Once Saved successfully return to the movies page
```

```
    $window.location = "#/movies";
    //To Do:- Return the newly added movie to the existing collection
    },
    function() {
    //Error
    toastr.error("Unable to Save the Movie");
    });
    }
    }
];
```

And then I have used the same in my **movieEditController** as shown below.

```
//movie-review-edit.js

//Defined Module
var module = angular.module("movieReviewEdit", []);

//Defined Routes
module.config(["$routeProvider", function ($routeProvider) {

    $routeProvider.when("/editMovie/:Id", {
    controller: "movieEditController",
    templateUrl: "/templates/editMovie.html"
    });

    $routeProvider.otherwise({ redirectTo: "/movies" });
}]);

var movieEditController = ["$scope", "dataService", "$window", "$routeParams",
    function($scope, dataService, $window, $routeParams) {
    //Initialize movie and movie id
    $scope.movie = null;
    $scope.MovieId = null;

    //Fetch the Movie by id
    dataService.getMovieById($routeParams.Id)
    .then(function(result) {
    //Success
    $scope.movie = result;
    },
```

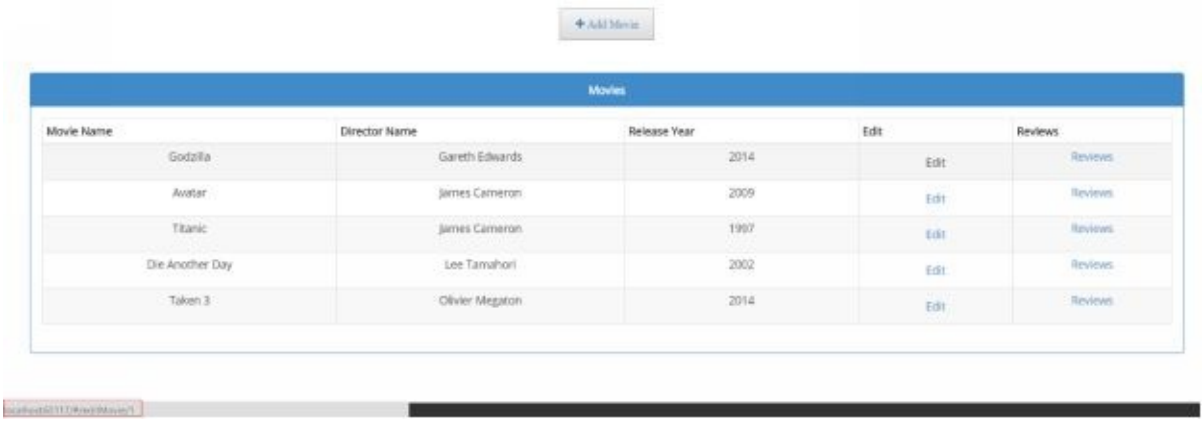
```
    function() {
    //Error
    toastr.error("Error Fetching Movie with Id:", +$routeParams.Id);
    });
}];
```

So, currently I have added only Fetch code. Now, the code is pretty much straight forward. Here it makes the call from **movieEditController** to fetch the movie based on the id via **dataService** and **dataService** gives the data back to the controller which in return gets bind to the **$scope.movie**. Now, this particular movie is available for editing. So, for this change template needs to be updated accordingly as shown below.

```
<form class="form-horizontal" name="editMovieForm" novalidate role="form" data-ng-submit="editMovie()">
    <div class="panel panel-primary">
    <div class="panel-heading"><b>Edit Movie</b></div>
    <div class="panel-body">
    <fieldset>
    <div class="form-group" style="display: none">
    <label for="MovieID" class="col-sm-3 control-label">Movie Name</label>
    <div class="col-sm-9">
    <input type="text" id="MovieID" name="MovieID" class="form-control" value="{{movie.Id}}" ng-model="movie.Id" />

    </div>
    </div>

    <div class="form-group">
    <label for="MovieName" class="col-sm-3 control-label">Movie Name</label>
    <div class="col-sm-9">
    <input type="text" id="MovieName" name="MovieName" class="form-control" ng-model="movie.MovieName" value="{{movie.MovieName}}" required ng-minlength="5" />
    <span class="help-block" ng-show="editMovieForm.MovieName.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="editMovieForm.MovieName.$error.minlength">Please enter 5 Charcters Title</span>
    </div>
    </div>

    <div class="form-group">
    <label for="DirectorName" class="col-sm-3 control-label">Director Name</label>
    <div class="col-sm-9">
    <input type="text" id="DirectorName" name="DirectorName" class="form-control" ng-model="movie.DirectorName" value="{{movie.DirectorName}}" required ng-minlength="5" />
    <span class="input-validation-error" ng-show="editMovieForm.DirectorName.$error.required"><i class="fa fa-star"></i></span>
```

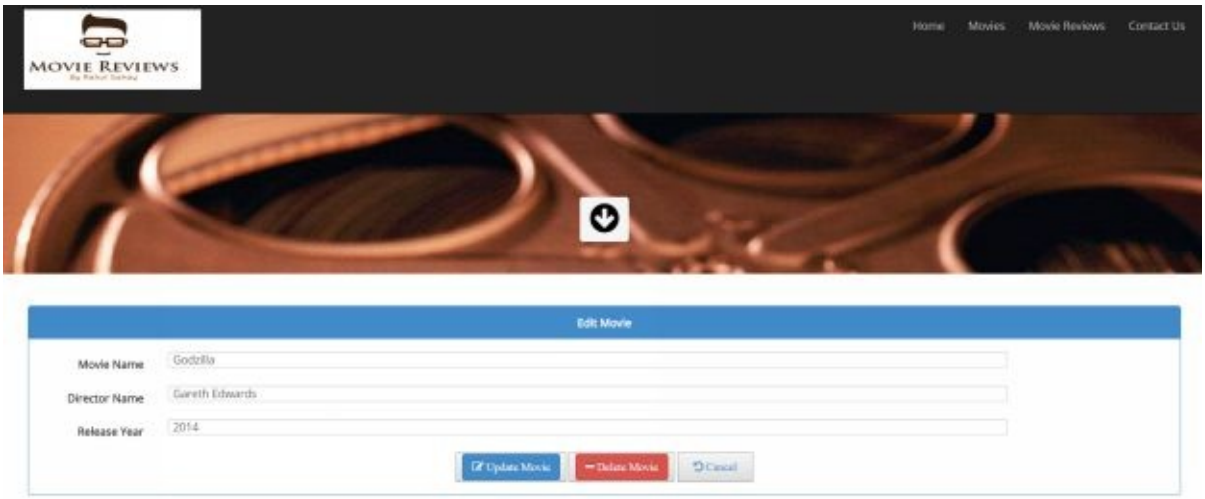
```
    <span class="input-validation-error" ng-show="editMovieForm.DirectorName.$error.minlength">Please enter 5 Charcters Name</span>
    </div>
    </div>

    <div class="form-group">
    <label for="ReleaseYear" class="col-sm-3 control-label">Release Year</label>
    <div class="col-sm-9">
    <input type="text" id="ReleaseYear" name="ReleaseYear" class="form-control" ng-model="movie.ReleaseYear" value="{{movie.ReleaseYear}}" required ng-pattern="/(19\d{2})|(200\d)|(201[0-5])/" />
    <span class="input-validation-error" ng-show="editMovieForm.ReleaseYear.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="editMovieForm.ReleaseYear.$error.pattern">Please enter year between 1900-2015</span>
    </div>
    </div>

    <div class="container">
    <button type="submit" id="updateMovie" value="updateMovie" ng-disabled="editMovieForm.$invalid">
    <i class="fa fa-edit btn btn-primary"> Update Movie</i>
    </button>
    <button type="button" id="deleteMovie" value="deleteMovie" data-ng-click="deleteMovie()">
    <i class="fa fa-minus btn btn-danger"> Delete Movie</i>
    </button>
    <button type="button" id="cancelMovie" value="cancelMovie">
    <i class="fa fa-undo btn btn-link"><a href="#/movies"> Cancel</a></i>
    </button>
    </div>
    </fieldset>
    </div>
    </div>

</form>
```

Here I have used both type of data-bindings. The one with **{{}} curly braces** are used for populating the value on load in textbox and **ng-model** is there to take the value back to the server. However, I have also presented markup for editing and deleting movie, but Javascript I have not written yet. But, that's ok, we will see the same in a moment. For now, let's save the changes and check the page.




Once I hover on the **Edit** link of 1st movie, it will flag me following path at the bottom of the screen as shown below.


Now, upon clicking on this link, it will take me to the designated page with values as shown below.




Now, we are in the Edit Movie window. Here, we should be able to update the existing movie or delete the movie. Let's go ahead and implement the same. Below in the snippet, I have updated my both the JS files with the required changes.

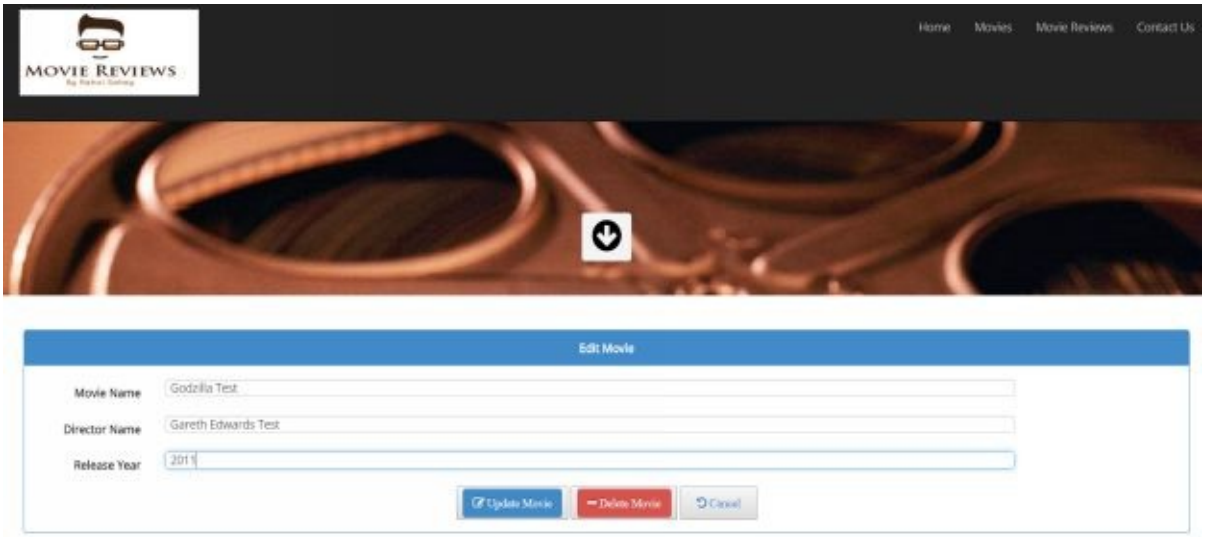
```
//Editing Movie
    var _movieEdit = function (movie) {
    var deferred = $q.defer();
    $http.put('http://localhost:65117/api/Movies/', movie)
    .then(function() {
    //Success
    deferred.resolve();
    }, function() {
```

```
//Error
deferred.reject();
});
return deferred.promise;
}
```

Here, in the above snippet I have added dataservice code for editing the movie. Then, in the below snippet I have added the controller code from where it is being called.

```
//Editing the Movie
$scope.editMovie=function() {
dataService.movieEdit($scope.movie)
.then(function() {
//Success
toastr.success("Movie Updated Successfully!");
$window.location = "#/movies";
}, function() {
//Error
toastr.error("Error in Updating Movie");
});
}
```

With these changes in place, I should be able to edit the movie as shown below in the screen shots.




So, update is working fine. Now, let's go ahead and undo these changes and then implement the delete movie feature. Below is the snippet for Deleting Movie from both JS files

```
//Deleting the movie
    $scope.deleteMovie = function () {
    dataService.removeMovie($scope.movie.Id)
    .then(function() {
    //Success
    toastr.success("Movie Deleted Successfully");
    $window.location = "/#movies";
    }, function() {
    //Error
    toastr.error("Error Deleting Movie with Id:", +$scope.movie.Id);
    });
    }

//Deleting Movie
    var _removeMovie =function(Id) {
    var deferred = $q.defer();
    $http.delete('http://localhost:65117/api/Movies/' + Id)
    .then(function() {
    //Success
    deferred.resolve();
    },
    function() {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    }
```

## Creating Reviews Workflow:-

In the previous section we have completed Movie CRUD operation. Now, let's go ahead and complete the Reviews workflow. Below, I have mentioned the controller code for the same

```
var reviewsController = [
    "$scope", "$routeParams", "$window", "dataService", function ($scope, $routeParams, $window, dataService) {
    $scope.reviews = null;
    $scope.MovieId = null;

    //Setting Timeout for spinner
    $('#loader').show();
    //Timeout function to show spinner
    setTimeout(function () {
    dataService.getReviewById($routeParams.Id)
    .then(function (review) {
    //Success
    //For pagination
    $scope.currentPage = 1;
    $scope.numPerPage = 4;
    $scope.maxSize = 5;

    $scope.numPages = function () {
    return Math.ceil(review.length / $scope.numPerPage);
    };

    $scope.$watch('currentPage + numPerPage', function () {
    var begin = (($scope.currentPage - 1) * $scope.numPerPage)
    , end = begin + $scope.numPerPage;
    $scope.filteredReviews = review.slice(begin, end);

    });
    $scope.reviews = review;
    $scope.MovieId = $routeParams.Id;
    toastr.success("Reviews retrieved Successfully");
    }, function () {
    //Error
    toastr.error("Error in Fetching Reviews");
    })
    .then(function () {
    $('#loader').hide();
```

```
    });
    }, 1000);
    }
];
```

Then, for API call, below code will get the required review for the movie.

```
var _getReviewById = function (Id) {
    var deferred = $q.defer();
    _getReviews(Id)
    .then(function () {
    //Success
    if (_reviews) {
    deferred.resolve(_reviews);
    } else {
    deferred.reject();
    }
    }, function () {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    }

var _getReviews = function (Id) {
    var deferred = $q.defer();
    $http.get('/api/MovieReviews/' + Id)
    .then(function (result) {
    //Success
    angular.copy(result.data, _reviews);
    deferred.resolve();
    }, function () {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    }
```

I have also modified the route to support the template as well. Below is the route for the same.

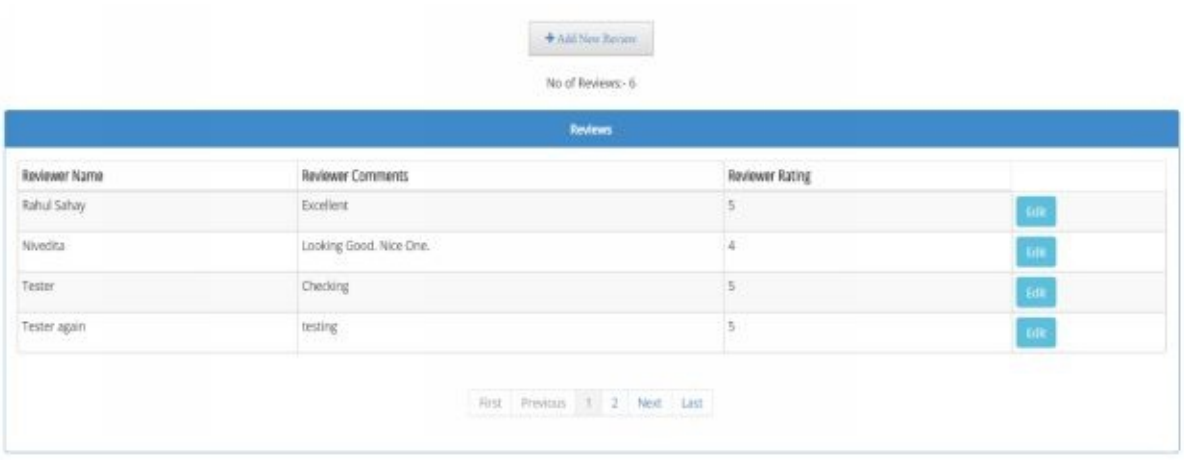
```
$routeProvider.when("/reviews/:Id", {
    controller: "reviewsController",
    templateUrl: "/templates/reviews.html"
    });
```

## And below is the template for the same.

```
<div class="container">
    <button type="button" id="addReview" value="addReview">
    <i class="fa fa-plus btn btn-link"><a data-ng-href="#/newReview/{{MovieId}}"> Add New Review</a> </i>
    </button>

</div>

<br />
<div>

    <div class="row">

    <div id="loader" style="display: none">
    <i class="fa fa-cog fa-fw fa-spin fa-2x"> </i> Getting Reviews
    </div>

    <div>
    No of Reviews:- {{reviews.length}}
    </div>
    <br />
    <form class="form-horizontal" name="reviewsForm" role="form">
    <fieldset>
    <div class="panel panel-primary">
    <!— Default panel contents —>
    <div class="panel-heading"><b>Reviews</b></div>
    <div class="panel-body">
    <table class="table table-hover table-striped table-condensed table-bordered">
    <thead>
    <tr>
    <th>Reviewer Name</th>
    <th>Reviewer Comments</th>
    <th>Reviewer Rating</th>
    </tr>
    </thead>
```

```
    <tbody>
    <tr ng-repeat="i in filteredReviews">
    <td>
    {{i.ReviewerName}}
    </td>
    <td>
    {{i.ReviewerComments}}
    </td>
    <td>
    {{i.ReviewerRating}}
    </td>
    <td>
    <a data-ng-href="#/editReview/{{i.Id}}" class="btn btn-info">Edit</a>
    </td>
    </tr>

    </tbody>
    </table>
    <div data-pagination="" data-num-pages="numPages()"
    data-current-page="currentPage" data-max-size="maxSize"
    data-boundary-links="true"></div>
    </div>
    </div>


    </fieldset>
    </form>
    </div>
</div>
```

Now, when I go ahead and see any of movie review, it will look like as shown below.



I have also provided one link to add new Review for the movie here. Let's dig into the code now. This code is also very straight forward what we have seen earlier.

```
$routeProvider.when("/newReview/:Id", {
    controller: "newreviewController",
    templateUrl: "/templates/newReview.html"
    });
```

Then, I have my controller code for the same as shown below.

```
var newreviewController = [
    "$scope", "$routeParams", "$window", "dataService", function ($scope, $routeParams, $window, dataService) {
    $scope.ReviewerRating = 3;
    $scope.max = 5;
    $scope.isReadonly = false;
    $scope.MovieId = null;
    $scope.newReview = {};

    $scope.cancelReview = function () {
    $window.history.back();
    }
    $scope.saveReview = function () {
    dataService.addReview($routeParams.Id, $scope.newReview)
    .then(function () {
    //success
    toastr.success("Thanks for your feedback!");
    $window.location = "#/movies";
    },
    function () {
    //error
    toastr.error("Couldn't Save the New Review");
    });
    };
    }
]
```

Followed by my dataService code to support the same.

```
  //Adding Review
    var _addReview = function (MovieId, newReview) {
```

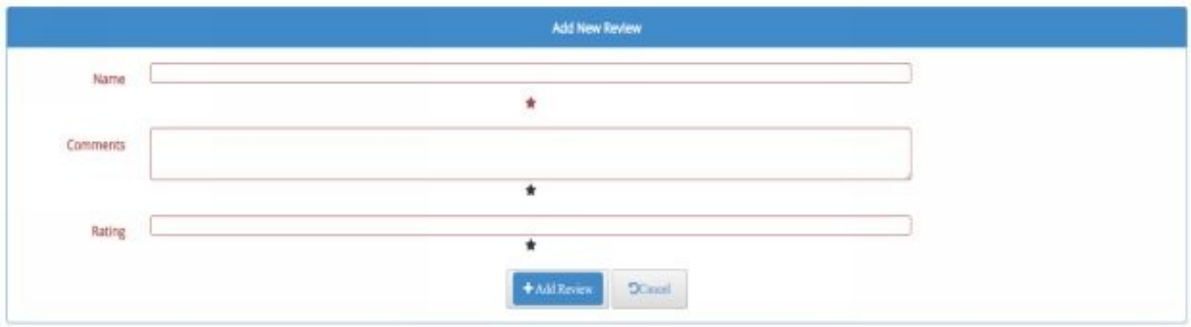
```
var deferred = $q.defer();
$http.post('/api/MovieReviews/' + MovieId, newReview)
.then(function () {
//success
deferred.resolve();
},
function () {
//error
deferred.reject();
});
return deferred.promise;
}
```

Now, in the end below is my template code for the same.

```
<div id="loader" style="display: none">
    <i class="fa fa-cog fa-fw fa-spin fa-2x"> </i> Saving Review
</div>
<form class="form-horizontal" name="newReviewForm" novalidate role="form" ng-submit="saveReview()">
    <div class="panel panel-primary">
    <div class="panel-heading"><b>Add New Review</b></div>
    <div class="panel-body">
    <fieldset>
    <div class="form-group" ng-class="{'has-error' : newReviewForm.ReviewerName.$invalid}">
    <label for="ReviewerName" class="col-sm-3 control-label">Name</label>
    <div class="col-sm-8">
    <input type="text" id="ReviewerName" name="ReviewerName" class="form-control" ng-model="newReview.ReviewerName" required ng-minlength="5" />
    <span class="help-block" ng-show="newReviewForm.ReviewerName.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="newReviewForm.ReviewerName.$error.minlength">Please enter 5 Characters Reviewer Name</span>
    </div>
    </div>

    <div class="form-group" ng-class="{'has-error' : newReviewForm.ReviewerComments.$invalid}">
    <label for="ReviewerComments" class="col-sm-3 control-label">Comments</label>
    <div class="col-sm-8">
    <textarea id="ReviewerComments" name="ReviewerComments" class="form-control" ng-model="newReview.ReviewerComments" required ng-minlength="15"></textarea>
    <span class="input-validation-error" ng-show="newReviewForm.ReviewerComments.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="newReviewForm.ReviewerComments.$error.minlength">Please
```

```
enter 15 Characters Comments atleast!</span>
    </div>
    </div>

    <div class="form-group" ng-class="{'has-error' : newReviewForm.ReviewerRating.$invalid}">
    <label for="ReviewerRating" class="col-sm-3 control-label">Rating</label>
    <div class="col-sm-8">
    <input type="text" id="ReviewerRating" name="ReviewerRating" class="form-control" ng-model="newReview.ReviewerRating" required ng-pattern="/^[0-5]$/" />
    <span class="input-validation-error" ng-show="newReviewForm.ReviewerRating.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="newReviewForm.ReviewerRating.$error.pattern">Please enter rating between 1-5</span>
    </div>
    </div>

    <div class="container">
    <button type="submit" id="addNewReview" value="addNewReview" ng-disabled="newReviewForm.$invalid">
    <i class="fa fa-plus btn btn-primary"> Add Review</i>
    </button>

    <button type="button" id="cancelMovie" value="cancelReview" ng-click="cancelReview()">
    <i class="fa fa-undo btn btn-link">Cancel</i>
    </button>
    </div>
    </fieldset>
    </div>
    </div>
</form>
```

With the above changes in place, when I click on **Add New Review** link, then it will take me to the brand new page to add the review.



From here I can go ahead and add any no of reviews. Now, let’s go ahead and give user flexibility to edit review and delete the same. Below is the snippet for the same.

```
//Route Code
  $routeProvider.when(“/editReview/:Id”, {
      controller: “revieweditController”,
      templateUrl: “/templates/editReview.html”
      });

//Controller Code
var revieweditController = [
      “$scope”, “dataService”, “$window”, “$routeParams”,
      function ($scope, dataService, $window, $routeParams) {
      $scope.review = null;
      $scope.newReview = {};

      //Fetching the Review by id and setting  $scope.review
      dataService.getReviewByReviewerId($routeParams.Id)
      .then(function (result) {
      $scope.review = result;
      },
      function () {
      toastr.error(“Unable to Fetch the review”);
      });

      //Editing the Review
      $scope.editReview = function () {

      dataService.updateReview($scope.review)
      .then(function () {
      //success
      toastr.success(“Review edited Successfully”);
      $window.location = “#/movies”;

      },
      function () {
      //error
      toastr.error(“Error in editing the Review”);
      });
      };

      //Deleting the review
```

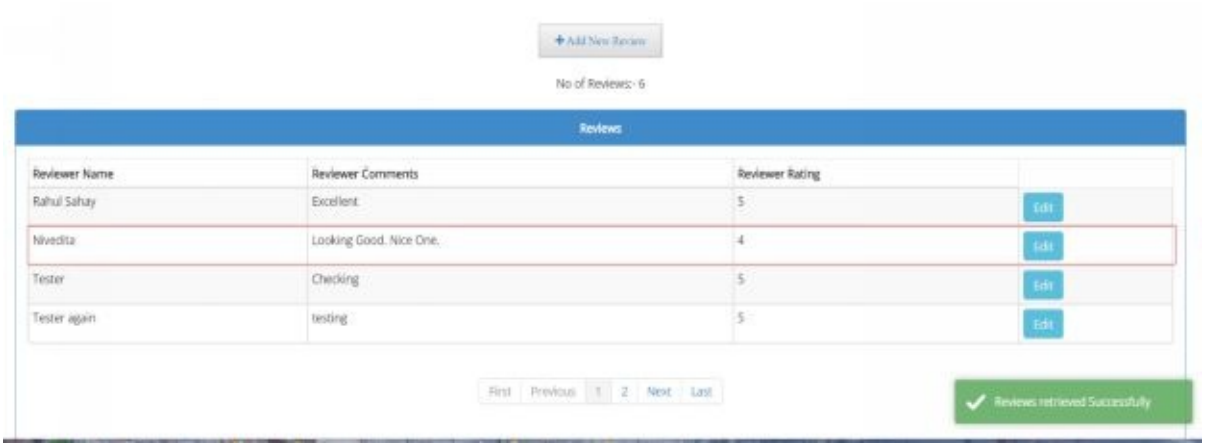
```
    $scope.deleteReview = function () {
    dataService.removeReview($scope.review.Id)
    .then(function () {
    //success
    toastr.success("Review Deleted Successfully");
    $window.location = "#/movies";
    },
    function () {
    //error
    toastr.error("Error Deleting Review with Id:", +$scope.review.Id);
    });
    };



    }
];

//dataService Code
//Get Review by ReviewerID
    var _getReviewByReviewerId = function (Id) {
    var deferred = $q.defer();
    $http.get('/api/Lookups/getbyreviewerid?id=' + Id)
    .then(function (result) {
    //Success
    deferred.resolve(result.data);
    }, function () {
    //Error
    deferred.reject();
    });
    return deferred.promise;
    };

  //Updating Review
    var _updateReview = function (newReview) {
    var deferred = $q.defer();
    $http.put('/api/MovieReviews/', newReview)
    .then(function () {
    //Success
    deferred.resolve();
    }, function () {
    deferred.reject();
    });
```

```
    return deferred.promise;
    };

    //Deleting the Review
    var _removeReview = function (Id) {
    var deferred = $q.defer();

    $http.delete('/api/MovieReviews/' + Id)
    .then(function () {
    //success
    deferred.resolve();
    },
    function () {
    //error
    deferred.reject();
    });
    return deferred.promise;
    }

//Markup Code
<form class="form-horizontal" name="editReviewForm" novalidate role="form" data-ng-submit="editReview()">
    <div class="panel panel-primary">
    <div class="panel-heading"><b>Edit Review</b></div>
    <div class="panel-body">
    <fieldset>
    <div class="form-group" style="display: none">
    <label for="MovieID" class="col-sm-3 control-label">Review ID</label>
    <div class="col-sm-9">
    <input type="text" id="MovieID" name="MovieID" class="form-control" value="{{review.Id}}" ng-model="review.Id" readonly />

    </div>
    </div>

    <div class="form-group" ng-class="{'has-error' : editReviewForm.ReviewerName.$invalid}">
    <label for="ReviewerName" class="col-sm-3 control-label">Name</label>
    <div class="col-sm-9">
    <input type="text" id="ReviewerName" name="ReviewerName" class="form-control" ng-model="review.ReviewerName" value="{{review.ReviewerName}}" required ng-minlength="5" />
    <span class="help-block" ng-show="editReviewForm.ReviewerName.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="editReviewForm.ReviewerName.$error.minlength">Please enter 5
```

```
Reviewer Name</span>
    </div>
    </div>

    <div class="form-group" ng-class="{'has-error' : editReviewForm.ReviewerComments.$invalid}">
    <label for="ReviewerComments" class="col-sm-3 control-label">Comments</label>
    <div class="col-sm-9">
    <textarea id="ReviewerComments" name="ReviewerComments" class="form-control" ng-model="review.ReviewerComments" value="{{review.ReviewerComments}}" required ng-minlength="15">
</textarea>
    <span class="input-validation-error" ng-show="editReviewForm.ReviewerComments.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="editReviewForm.ReviewerComments.$error.minlength">Please enter 15 character comment atleast!</span>
    </div>
    </div>

    <div class="form-group" ng-class="{'has-error' : editReviewForm.ReviewerRating.$invalid}">
    <label for="ReviewerRating" class="col-sm-3 control-label">Rating</label>
    <div class="col-sm-9">
    <input type="text" id="ReviewerRating" name="ReviewerRating" class="form-control" ng-model="review.ReviewerRating" value="{{review.ReviewerRating}}" required ng-pattern="/^[0-5]$/" />
    <span class="input-validation-error" ng-show="editReviewForm.ReviewerRating.$error.required"><i class="fa fa-star"></i></span>
    <span class="input-validation-error" ng-show="editReviewForm.ReviewerRating.$error.pattern">Please enter rating between 1-5</span>
    </div>
    </div>

    <div class="form-group" style="display: none">
    <label for="ReviewerMovieId">Movie Id</label>

    <input name="ReviewerMovieId" type="text" value="{{review.MovieId}}" data-ng-model="review.MovieId" required readonly />

    </div>

    <div class="container">
    <button type="submit" id="updateReview" value="updateReview" ng-disabled="editReviewForm.$invalid">
    <i class="fa fa-edit btn btn-primary"> Update Review</i>
    </button>
    <button type="button" id="deleteMovie" value="deleteMovie" data-ng-click="deleteReview()">
    <i class="fa fa-minus btn btn-danger"> Delete Review</i>
    </button>
```

```
<button type="button" id="cancelReview" value="cancelReview">
<i class="fa fa-undo btn btn-link"><a href="#/movies"> Cancel</a></i>
</button>
</div>
</fieldset>
</div>
</div>

</form>
```

Now, let me explain the code a bit. 1st version of code is to fetch the review and show the user at the time of editing, then I have code for updating and deleting the review. Now, with the above changes in place, let's go ahead and check the app.

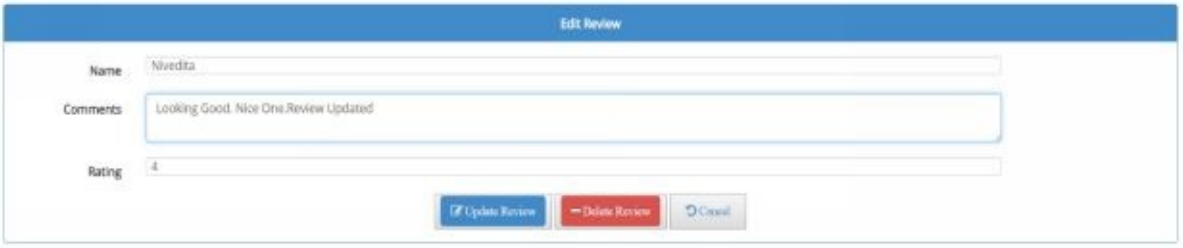



When I click on the highlighted one, then it will take me to the below shown window


Let's suppose you want to edit the entry and try to submit an empty form, then it will show the below error.


Basically, you can't submit the form. Here, **Update Review** will get disabled. All the fields are also having some input validation like **required**, **minlength** or **pattern.** Let's inspect them as well.


So, until you correct above errors; you can't submit the form. Once, you correct the above things, then you can go ahead and update the review like shown below.




Once review updated successfully, then it will navigate back to movies link. Similarly, it works for **Delete Review**.

## Summary:-

In this section, we have completed the App with full CRUD functionality. However, in order to achieve the same we have used standard API calls what we have designed earlier. Also, in order to keep things separate and clean, we have used services to do specific job. In the last section, we'll see how to write Unit Test cases for the client side code and to run the same from Visual Studio Test runner.

# Chapter 6: Unit Testing

## WHAT DO you find in this CHAPTER?

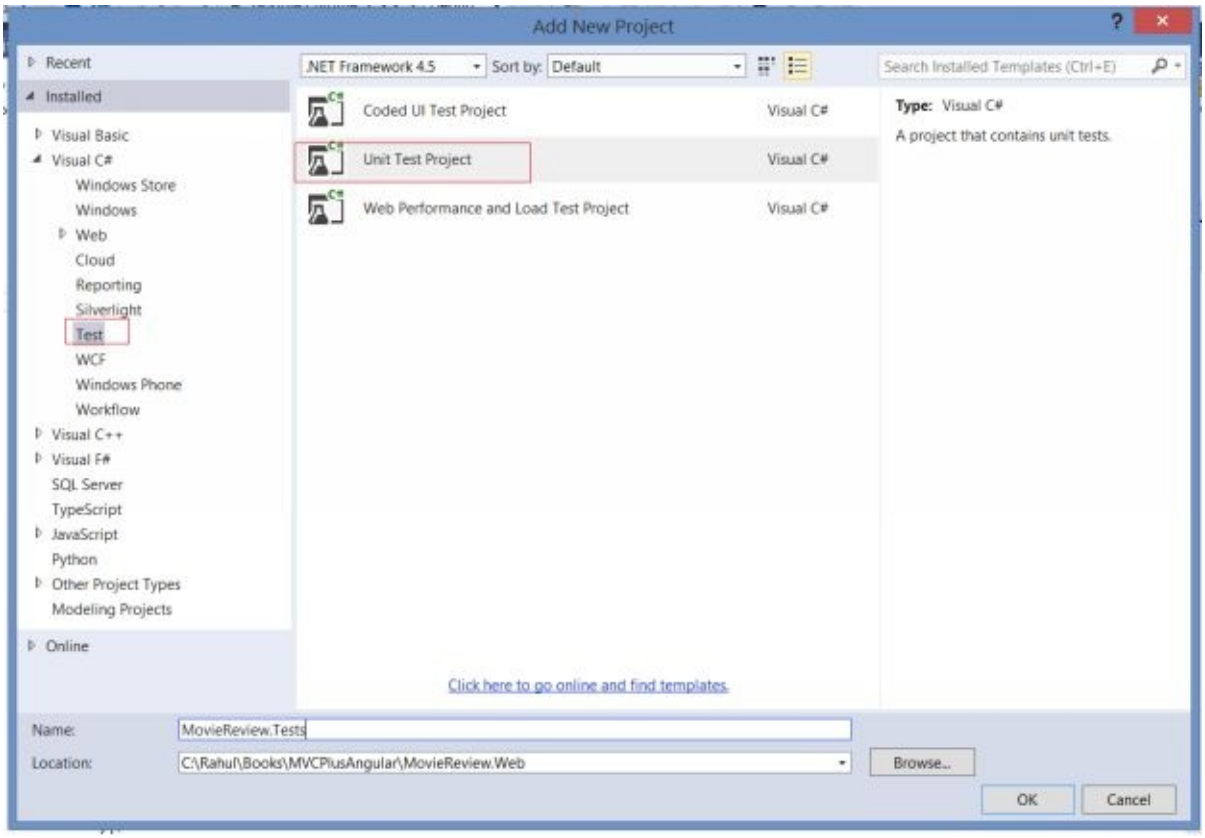
- Introduction
- Creating Test Project
- Installing Chutzpah Test Adapter
- Writing 1st JavaScript Test
- Writing Angular Test
- Using $httpBackend Service
- Writing Controller Tests
- Code Coverage
- Summary

### Introduction-

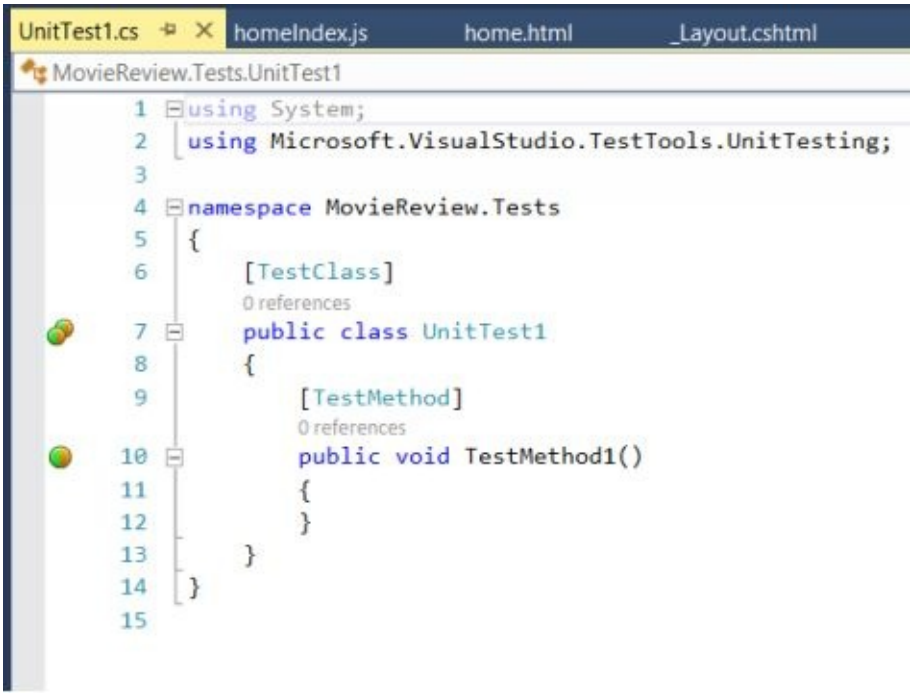
In this section we'll get started with writing test case for our API controllers then we'll write test case for angular scripts. In this segment I will introduce variety of test framework to write client side script test cases. Here, we will see how to integrate the client side testing framework in Visual Studio test runner itself. Other than this we are going to cover various significant and out of the box things in this segment. So, let's get started.

### Creating Test Project-

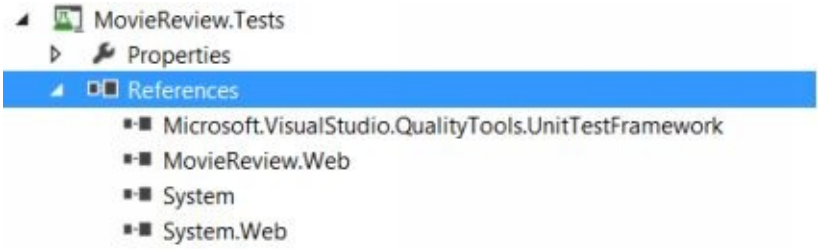
First of all I am going to add a new Unit Test Project from the Test tab as shown below in the screen shot.



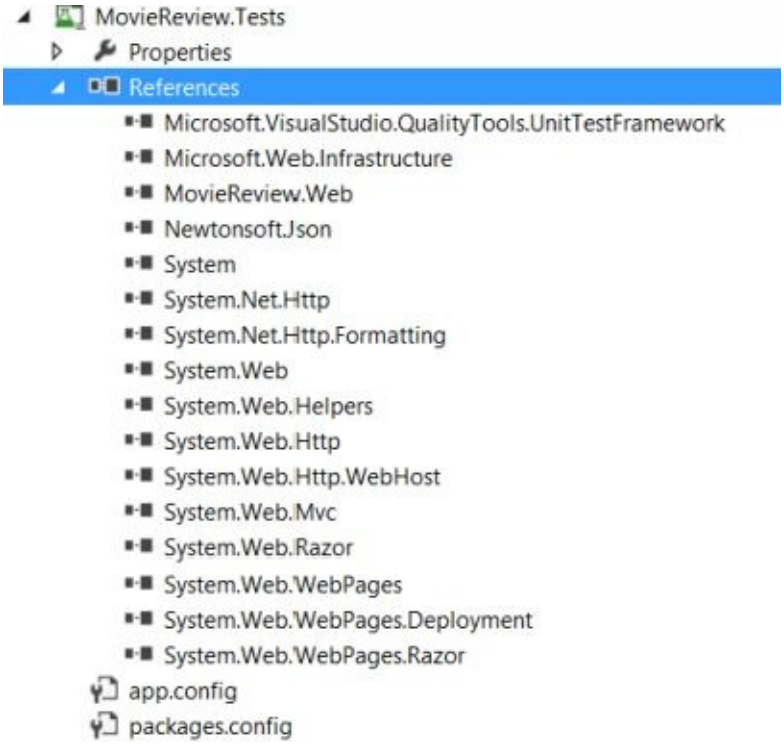

Upon successful creation, it will present me a default template. However, I usually don't use the default one hence I delete this and start from scratch.




Now, before getting started, I am going to add few references. So, the 1st thing which I am going to add here is **System.Web** as we'll be testing our website, then I'll also add reference of our own project as shown below.



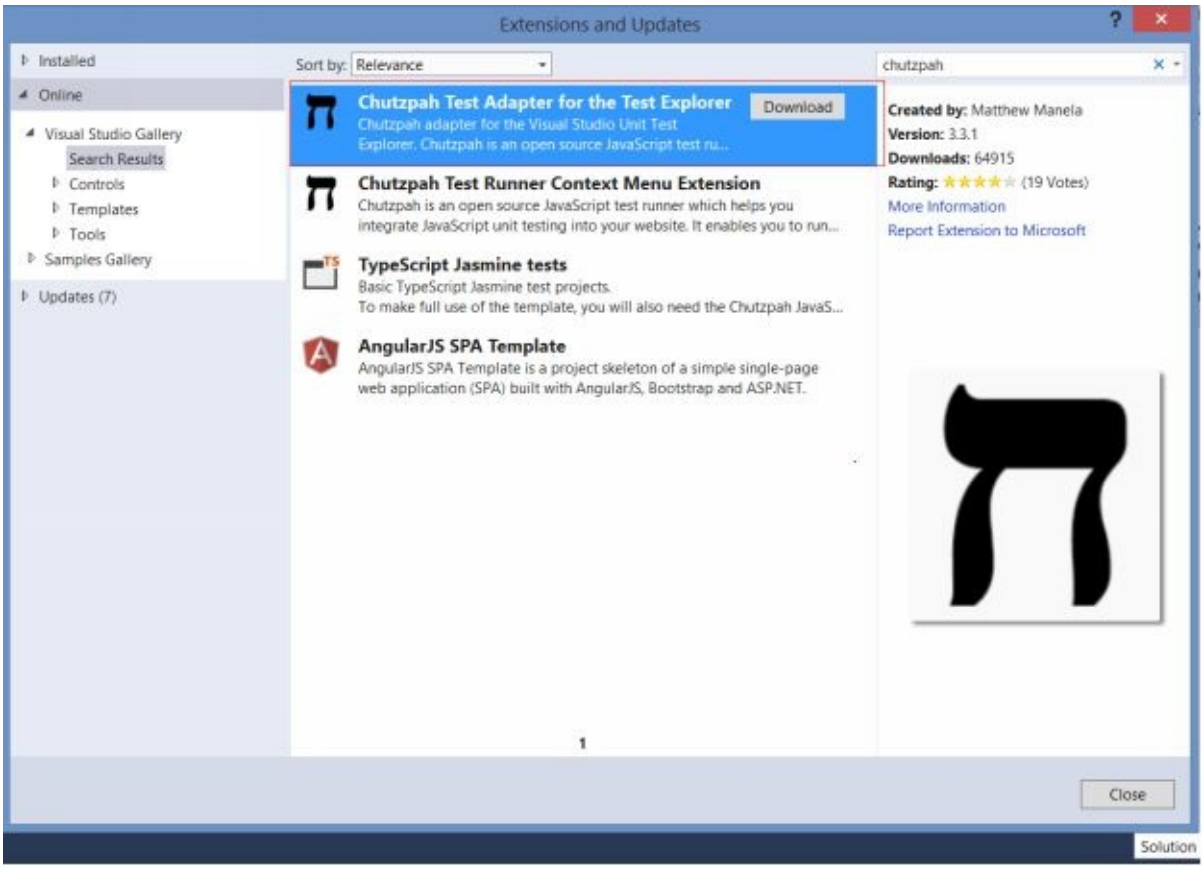

Then, I also need **ASP.NET MVC** and **ASP.NET Web API**. So, to install these best way is install via nuget or Package Manager Console Window. Now, after installing these when I see my references, then it will look like




So, now I have got all my required dependency to get started with test cases. Here, one more point I would like to mention that I have already covered Web API Tests using **QUnit** in previous sections. However, we are going to cover more Java Script Tests but with a different approach using different libraries.

## Installing Chutzpah Test Adapter-

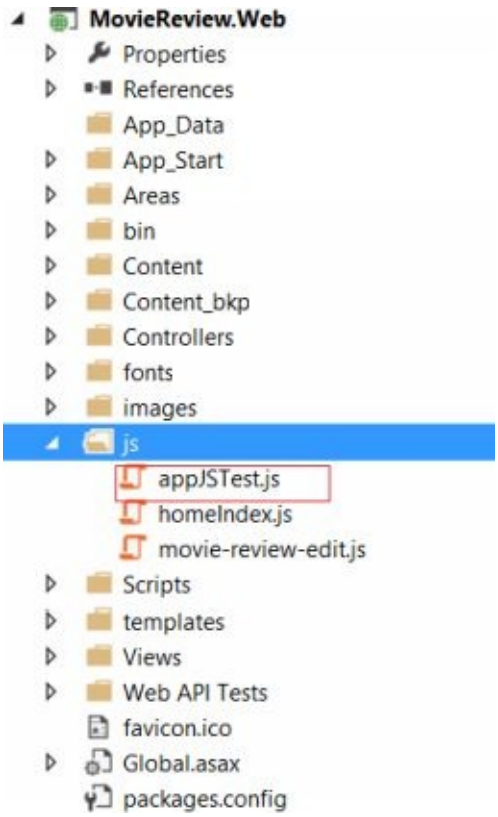
In this section, we'll focus on testing Java Script tests or client side code tests. By default, Microsoft Test runner don't include any client side test. So, to enable Client Side testing in MS Test runner, we need to install one extension **Chutzpah Test Adapter** in Visual Studio as shown below in the screen shot.




This project is open source available on [https://github.com/mmanela/chutzpah](https://github.com/mmanela/chutzpah) . So, you can see how it works and entire architecture of the same. However, once this extension gets installed you need to restart your visual studio like all other extensions. Once it gets installed successfully, it will simply look for JavaScript test in test window. No more configuration is required. So, now, let's go ahead and write some JavaScript Tests.

## Writing 1st JavaScript Test-

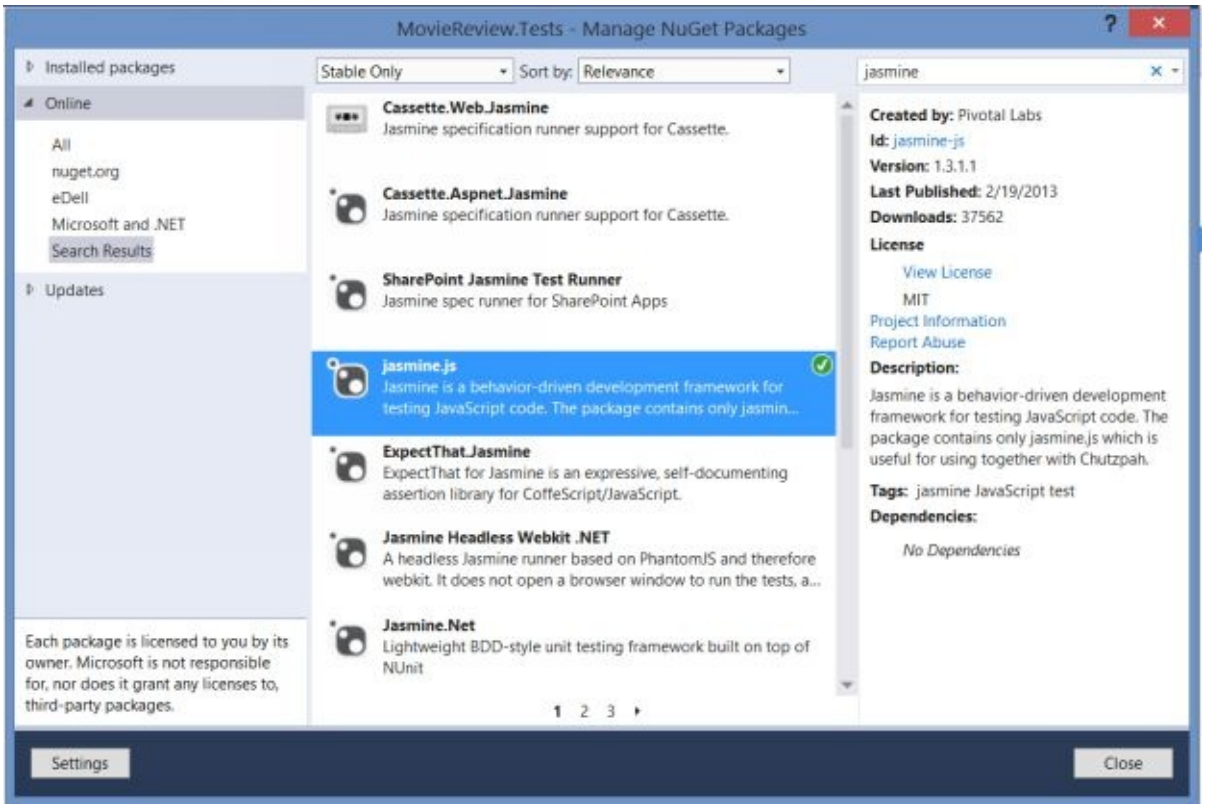
Here, the very 1st thing which I am going to do is add one simple demo file which in my web project under js directory.




As you can see in the below snippet, this is just a self-executing JavaScript function where in it is setting Locale property to true and if the Locale is true then it will log the message in the console.

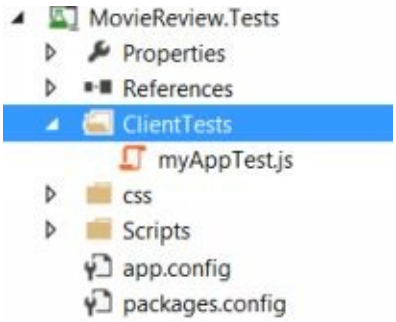
```
//appJSTest
//Created self executing function with one global variable
(function (myapp) {

    //setting isLocale to true
    myapp.isLocale = true;

    //logging message in console
    //if locale is true
    myapp.log = function (msg) {
    if (myapp.isLocale) {
    console.log(msg);
    }
    };
})(window.myapp = window.myapp || {});
```

Now, before writing the test, I am going to need one testing framework say **Jasmine**. You can use **Qunit** also to write and test against the same. But, I prefer Jasmine for testing my Angular code as Angular team uses the same thing. So, let’s go ahead and install Jasmine from Nuget.




As shown in the above screen shot, I have successfully installed the **Jasmine.JS** in my test project. Upon successful installation it added two folders one for **Scripts** and one for **CSS** like qunit. But, I am not going to use the same as I'll be running my test in Test Runner itself.

Now, I have created one folder **ClientTests** in my Test project for keeping all my client side tests at one location. Here I have also added one test file for testing the code which I just added in web project. Now, this sounds a bit weird, rather than creating any mocking for the same in test project, I am directly going to reference the same file. But, that's ok, we can do inter project reference in client side testing as well. Since, this is just a test file we can go ahead and execute the full path there is no harm in that.



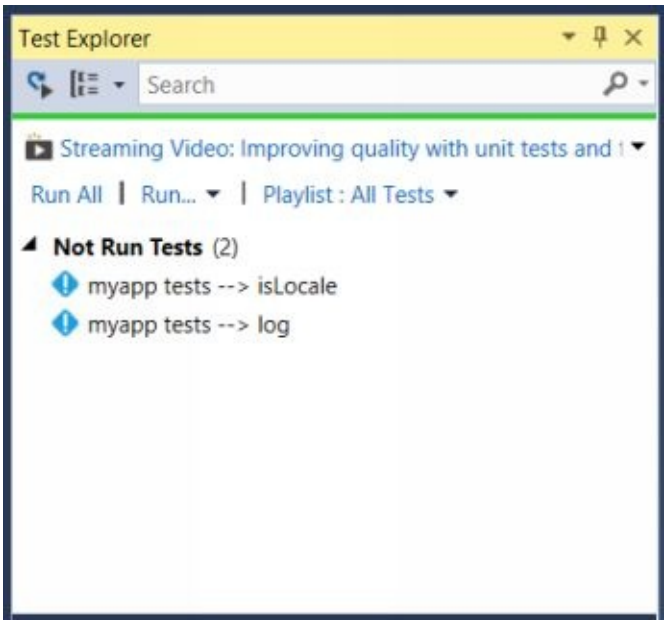

```
/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/js/appjstest.js" />
```

```
//myAppTest

describe("myapp tests —>",function() {

    //it is sub grouping or group of tests
    it("isDebug", function() {
    expect(myapp.isLocale).toEqual(true);
    });

    it("log", function() {
    expect(myapp.log).toBeDefined();
    });
})
```

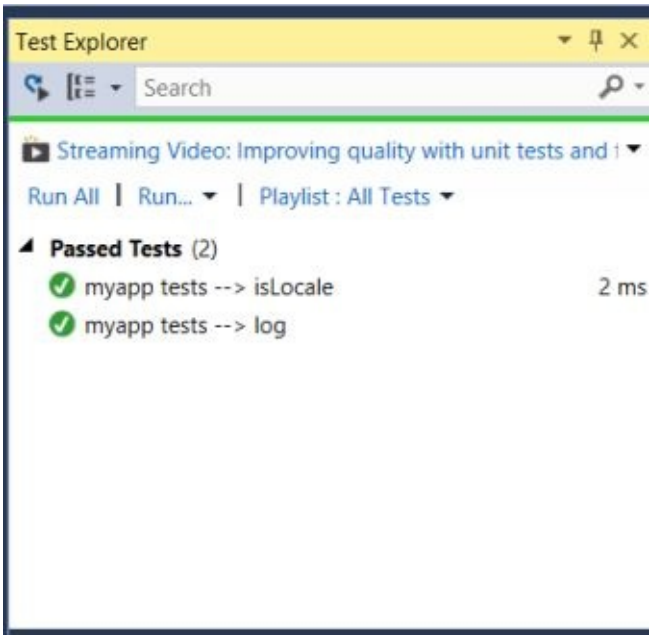
Now, in the above snippet 1st thing which I have done is added the reference of jasmine to my file so that chutzpah recognizes this as a test file and also it provides me intellisense while I am writing my test case. 2nd line is the reference of file which in under test. So, here you can see that there is relative file path for inter project reference.

Next thing is my test description. It starts with the keyword describe which takes the group name followed by callback function. And inside that callback function we can have multiple tests starting with **it**. **it** also takes the Test name as 1st parameter then under the callback function expect to check the Assertion. I recommend to have a look at http://jasmine.github.io/1.3/introduction.html for broader picture on writing test cases.

With the above change in place when I build and refresh the Test Explorer, then it will show my tests.



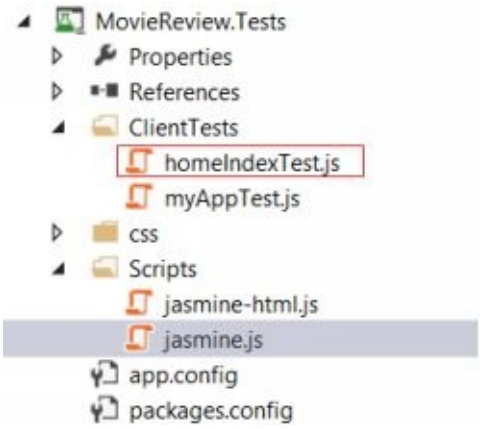

Now, when I click on Run All, then it will show me the below results.




So, my both tests get passed. However, when I right-click on any individual test and run the same it will run all the tests because in case JavaScript Tests it runs all the test which is there in the file. However, this was the simple demo how to write and test client side tests. Now, let's go ahead and write actual angular tests.

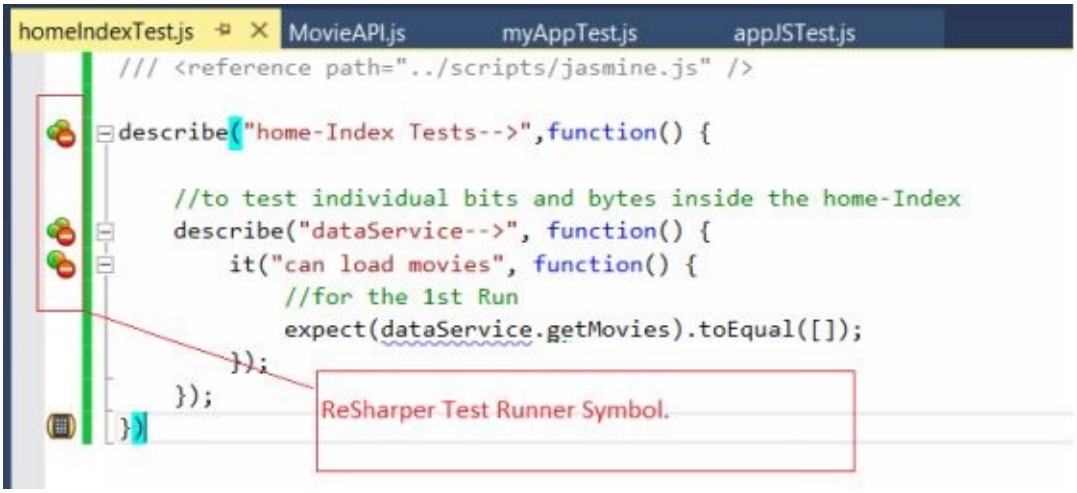
## Writing Angular Test-

In this section, I am going to start with writing my Angular Tests. Again, the 1st thing is going to be the same creating a new JavaScript file in Test project as shown below.




```
/// <reference path="../scripts/jasmine.js" />

describe("home-Index Tests—>",function() {

    //to test individual bits and bytes inside the home-Index
    describe("dataService—>", function() {
    it("can load movies", function() {
    //for the 1st Run
    expect(dataService.getMovies).toEqual([]);
    });
    });
})
```

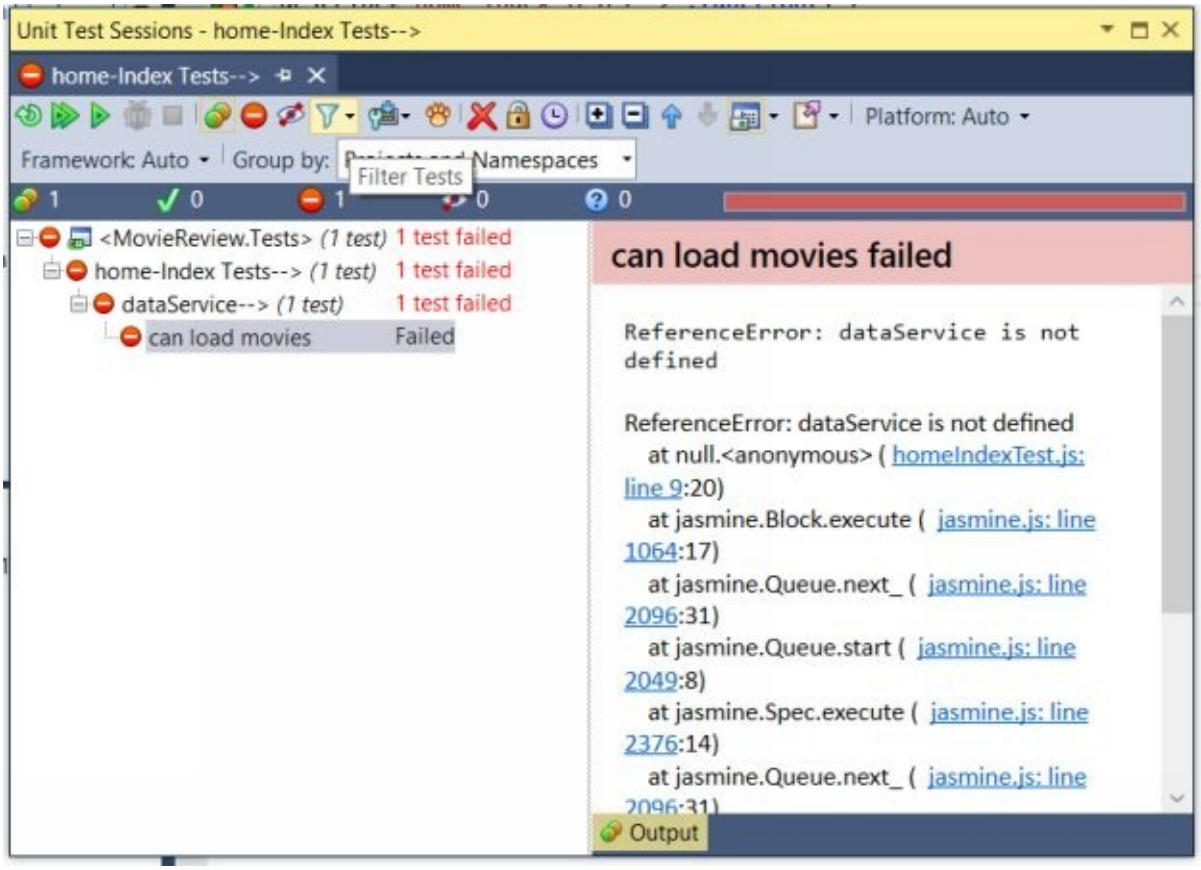
Above snippet also starts on the same line like by adding Jasmine reference and then writing the test case. So, the peculiar thing is I am using couple of describes here; this is just to dive inside the bits and bytes of that file and test the same. However, for the 1st run, I would expect empty array. But, here I would like to show one more thing.




Since, I am also using Re-sharper with Visual Studio, hence it offers me its Test Runner to run the Test in Re-sharper Test Server. Now, when I click on any of these symbols, it will

offer me to run the test as shown below.

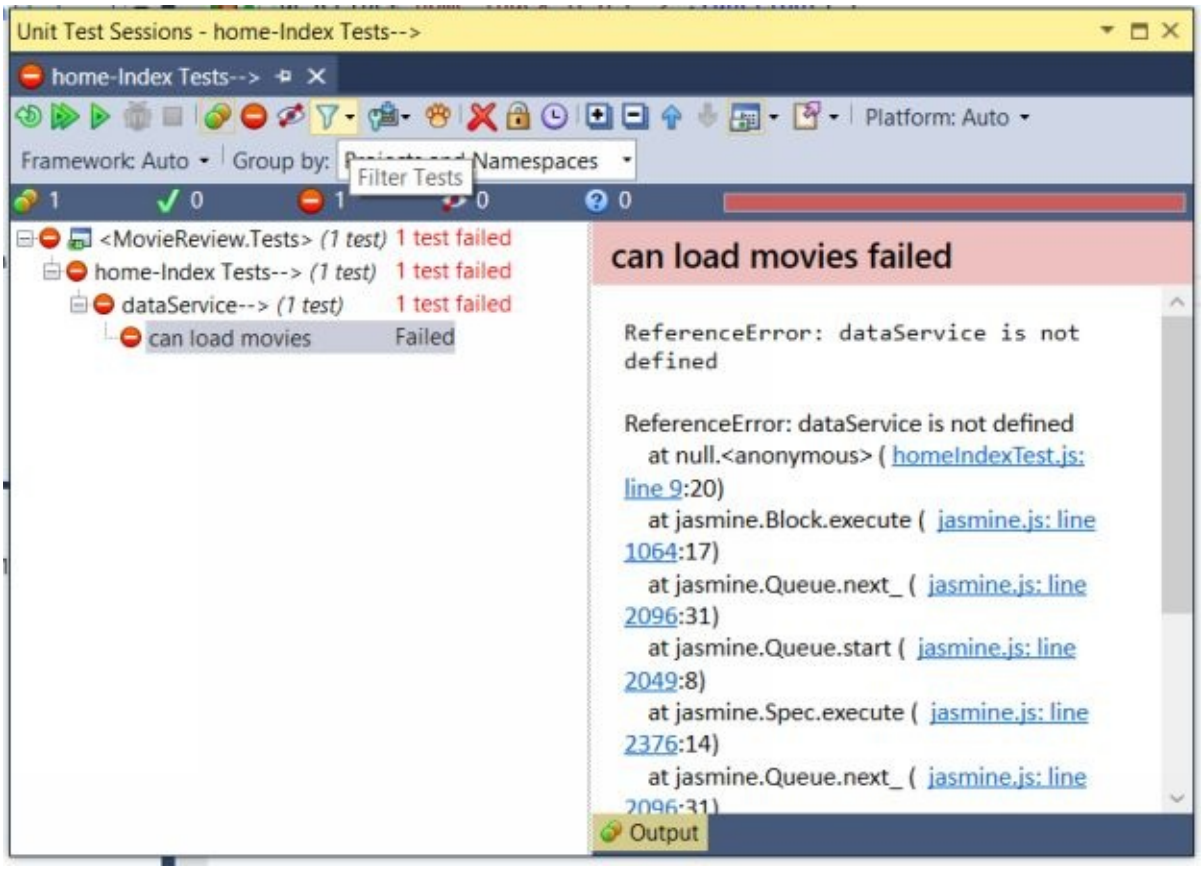

When I click on Run, this will launch two things, one is the browser window which will be blank because Re-sharper expects the test file to be in the same project where actual file is. Another window will be in Visual Studio, which will list the error message as shown below.




Here, the error is obvious. It appeared because **dataService** reference wasn't there. So, now let's include the dependency.

```
homeIndexTest.js | MovieAPI.js | myAppTest.js | appJSTest.js

/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/scripts/angular.min.js" />
/// <reference path="../../moviereview.web/js/homeindex.js" />

describe("home-Index Tests-->",function() {

    //to test individual bits and bytes inside the home-Index
    describe("dataService-->", function() {
        it("can load movies", function() {
            //for the 1st Run
            expect(dataService.getMovies).toEqual([]);
        });
    });
})
```

Let’s test this again.

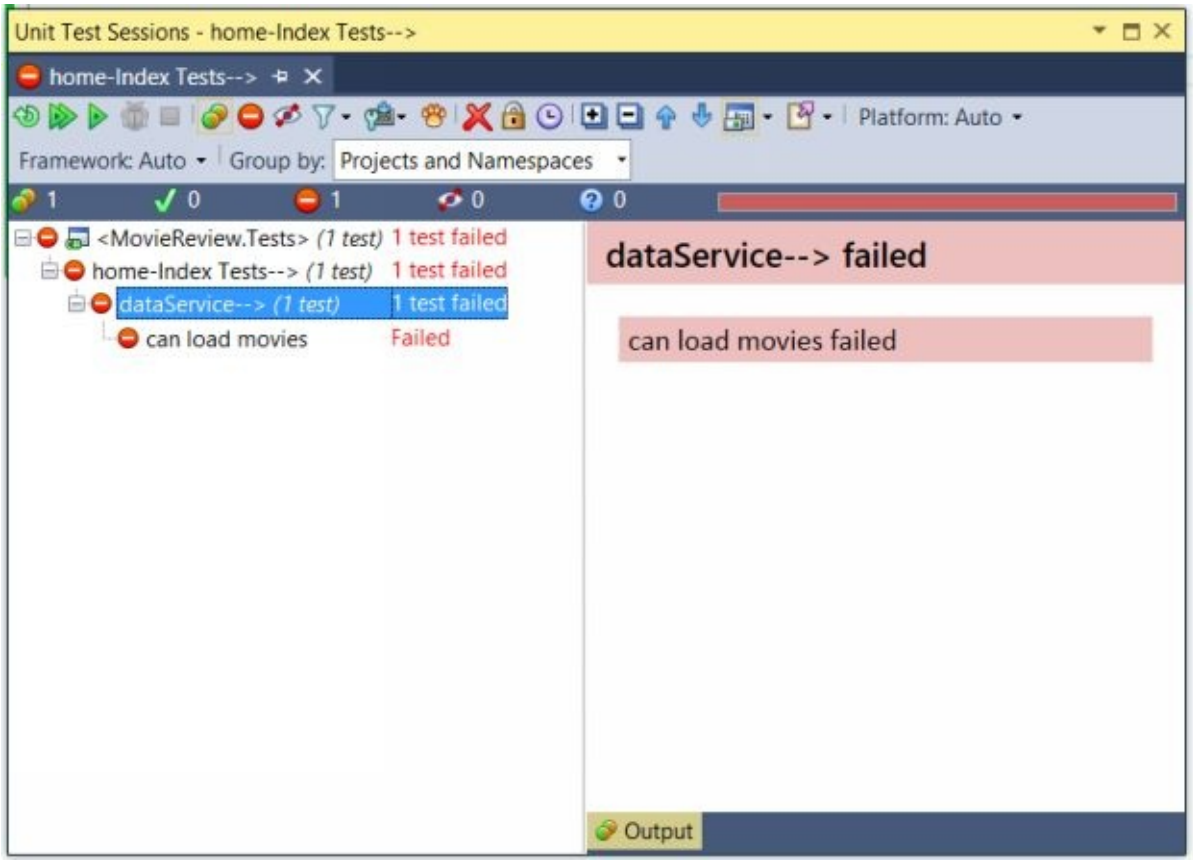



Here, when I ran the same it again produced me the same error. And the reason for the same is very simple. Angular JS injects the dependency at times when it is needed. So, now we need a way to inject these dependencies in there. Angular JS provides a mocking file as well for resolving the all the required dependencies. Now, I am going to drop mock file just before my homeIndex file as shown below.

```
/// <reference path=”../scripts/jasmine.js” />
/// <reference path=”../../moviereview.web/scripts/angular.min.js” />
/// <reference path=”../../moviereview.web/scripts/angular-mocks.js” />
```

```
/// <reference path="../../moviereview.web/js/homeindex.js" />

describe("home-Index Tests—>",function() {

    //to test individual bits and bytes inside the home-Index
    describe("dataService—>", function() {
    it("can load movies", inject(function(dataService) {
    //for the 1st Run
    expect(dataService.getMovies).toEqual([]);
    }));
    });
})
```

Here, I have used **inject** property to inject **dataService** for me. Now, with this change in place when I go ahead and run the test.

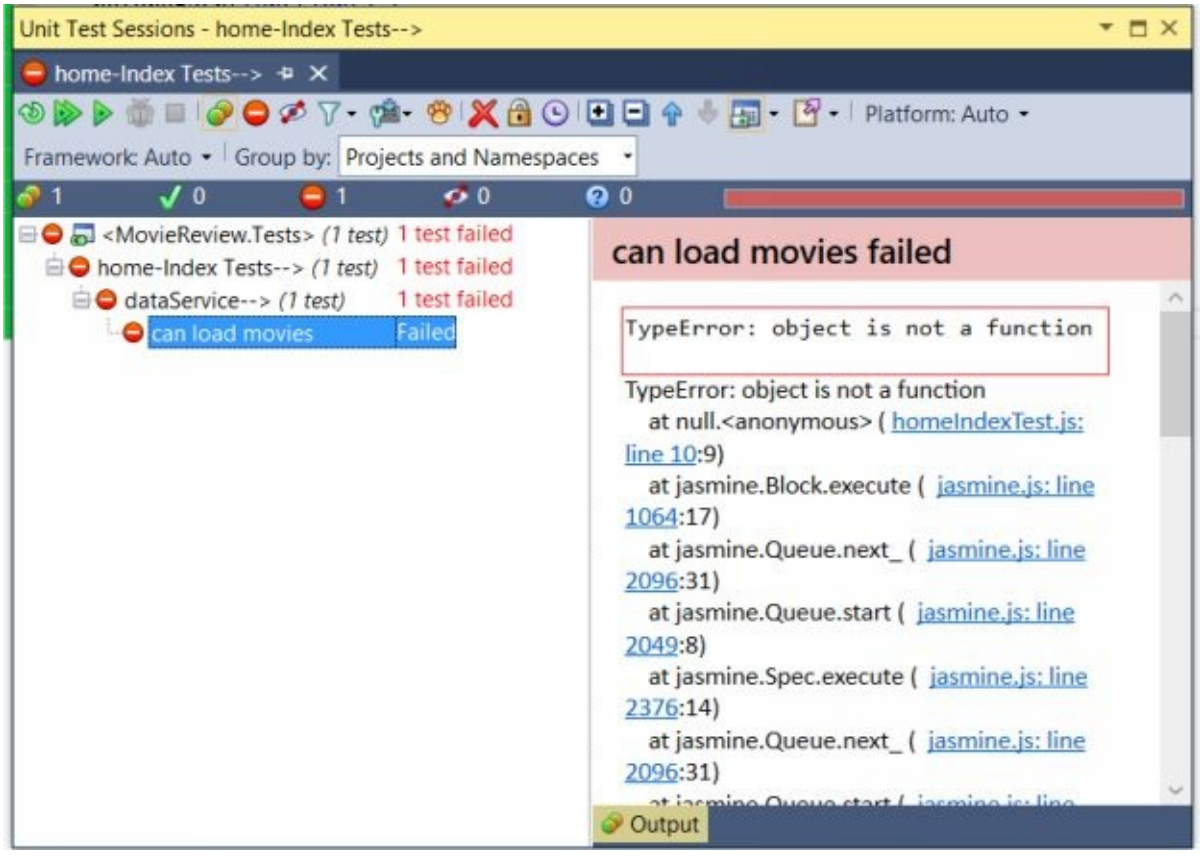



It again failed. Its ok, we are close to fix the issue. This time it said dataService failed because we didn't instantiate the required module.

```
/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/scripts/angular.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-mocks.js" />
/// <reference path="../../moviereview.web/js/homeindex.js" />
```

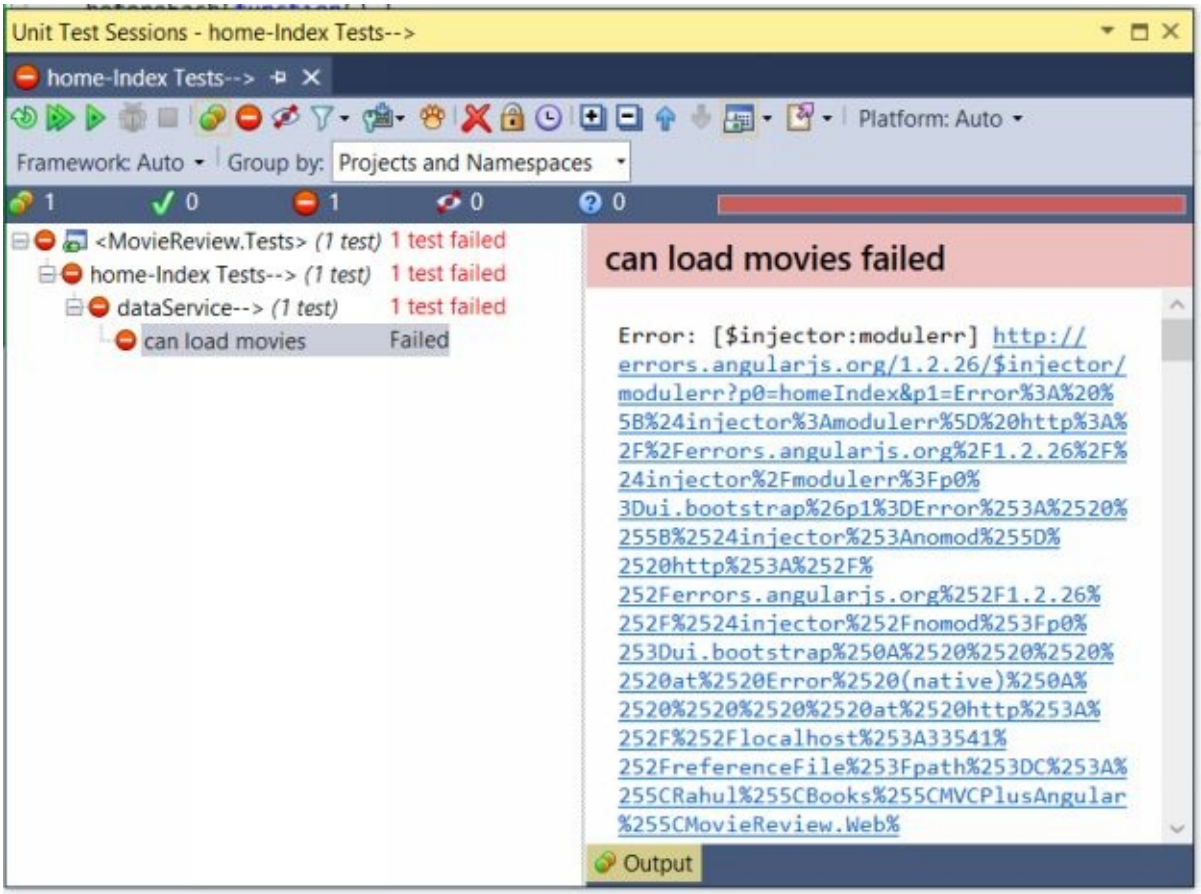
```
describe("home-Index Tests—>",function() {

    //instantiate the module 1st
    beforeEach(function() {
    module("homeIndex");
    });
    //to test individual bits and bytes inside the home-Index
    describe("dataService—>", function() {
    it("can load movies", inject(function(dataService) {
    //for the 1st Run
    expect(dataService.getMovies).toEqual([]);
    }));
    });
})
```

Here in the above snippet, I have used the mechanism **beforeEach** which will run before any test runs. With this change in place when I go ahead and run the test, then it failed again and this time it produced me the below result.




And reason for the same is we have used module as global variable in our **homeindex** and in **movie-review-edit** file. So, to fix this either we simply rename there it to some other name or wrap the same in self executing function. Here, I have renamed the same with

**homeIndexModule** and **movieReviewModule** then replaced at the required places. Now, when I ran the test, it's again failed, but with a different reason.

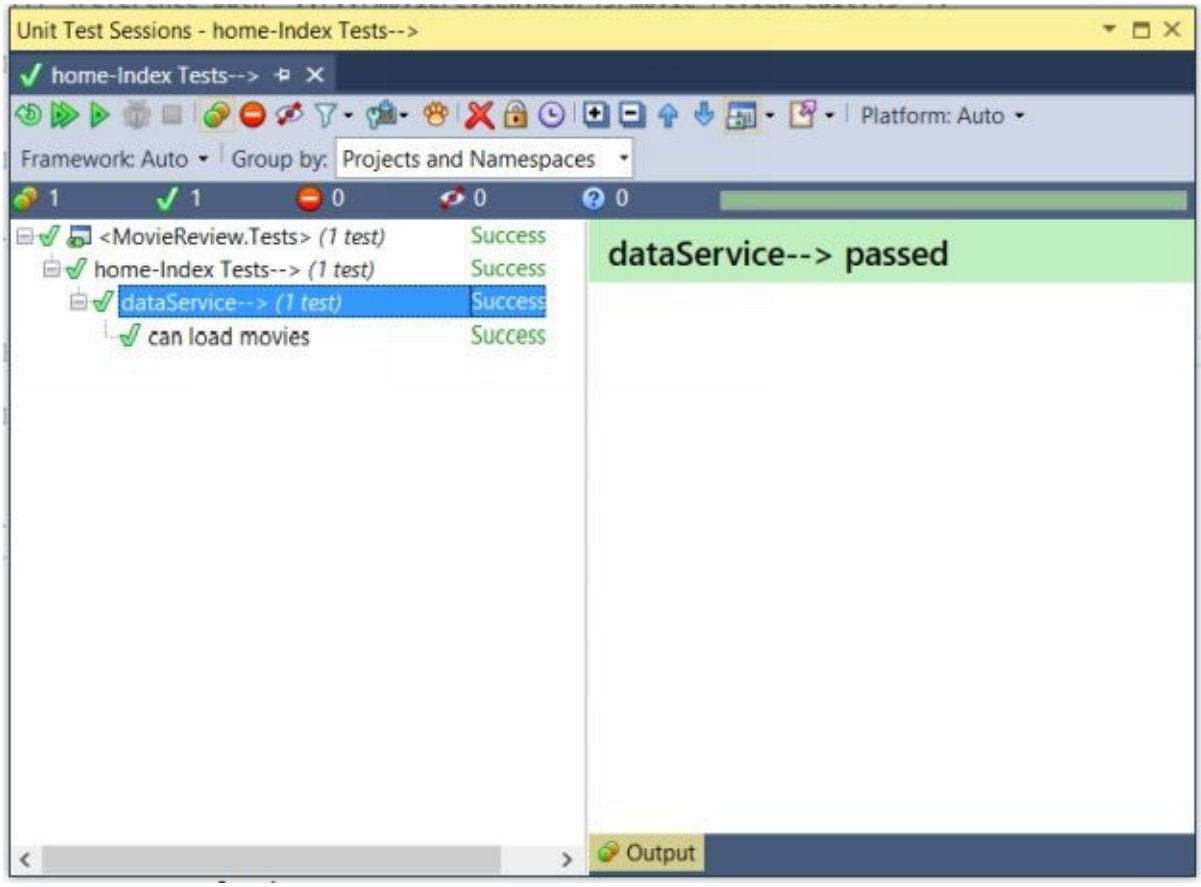



Now, it says the required dependency for **ngRoute** is missing. So, in the latest version of Angular, this is moved to different file that is route file. Now, I need to include that file as well as shown below.

```
/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/scripts/angular.min.js" />
/// <reference path="../../moviereview.web/scripts/ui-bootstrap-tpls.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-route.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-mocks.js" />
/// <reference path="../../moviereview.web/js/homeindex.js" />
/// <reference path="../../moviereview.web/js/movie-review-edit.js" />

describe("home-Index Tests—>", function () {


    beforeEach(function () {
    module("homeIndex");
    });
```

```
//to test individual bits and bytes inside the home-Index
describe("dataService—>", function () {

it("can load movies", inject(function (dataService) {
//for the 1st Run
expect(dataService.movies.length).toEqual(0);
}));
});
```

Now, when I ran the same, it gets passed as shown below in the screen shot.




Now, let's try to work with actual data via **$http** call.

## Using $httpBackend Service-

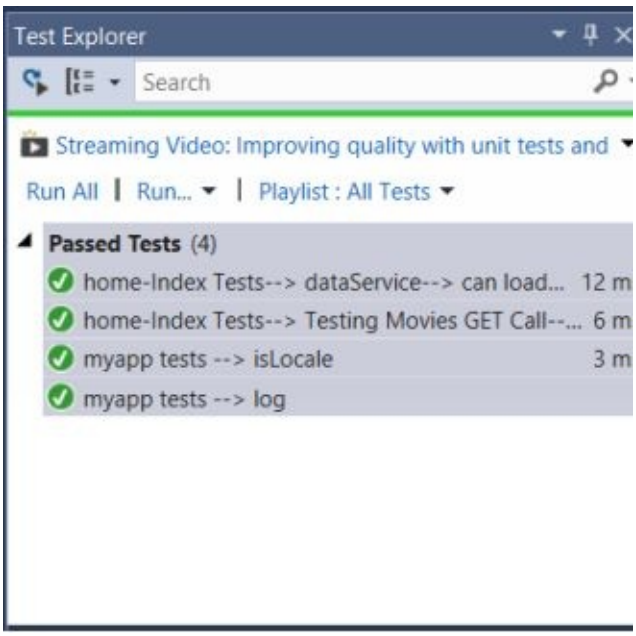
In this section, we will use mocked up version of **$http** service. One point to note here, our goal to test here JavaScript code not the complete system. Hence, we will be using mocking capability to create some mock data for me and give us back. **$httpBackend** is going to mimic the backend. So, as soon as my test see **$http**, it won't go to backend rather it will go to **$httpBackend** and return the mocked up version of data as shown below.

```
/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/scripts/angular.min.js" />
/// <reference path="../../moviereview.web/scripts/ui-bootstrap-tpls.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-route.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-mocks.js" />
/// <reference path="../../moviereview.web/js/homeindex.js" />
/// <reference path="../../moviereview.web/js/movie-review-edit.js" />

describe("home-Index Tests—>", function () {


    beforeEach(function () {
    module("homeIndex");
    });


    //to test individual bits and bytes inside the home-Index
    describe("dataService—>", function () {

    it("can load movies", inject(function (dataService) {
    //for the 1st Run
    expect(dataService.movies.length).toEqual(0);
    }));
    });

    //$httpbackend service
    var $httpBackend;
    var url = '/api/movies';

    var fakedMoviesResponse=[{
    Id: 1,
    MovieName: "Godzilla",
    DirectorName: "Gareth Edwards",
    ReleaseYear: "2014",
    NoOfReviews: 6
    },
    {
    Id: 3,
    MovieName: "Titanic",
    DirectorName: "James Cameron",
    ReleaseYear: "1997",
```

```
NoOfReviews: 3
},
{
Id: 4,
MovieName: “Die Another Day”,
DirectorName: “Lee Tamahori”,
ReleaseYear: “2002”,
NoOfReviews: 0
},
{
Id: 7,
MovieName: “Taken 3”,
DirectorName: “Olivier Megaton”,
ReleaseYear: “2014”,
NoOfReviews: 0
},
{
Id: 9,
MovieName: “Top Gun”,
DirectorName: “Tony Scott”,
ReleaseYear: “1986”,
NoOfReviews: 0
}
];
beforeEach(inject(function ($injector) {

$httpBackend = $injector.get(“$httpBackend”);

$httpBackend.whenGET(url)
.respond(fakedMoviesResponse);

}));

afterEach(function () {
$httpBackend.verifyNoOutstandingExpectation();
$httpBackend.verifyNoOutstandingRequest();
});
//test the backend call

describe(“Testing Movies GET Call—>”, function () {

it(“Loaded Movies”, inject(function (dataService) {
```

```
    $httpBackend.expectGET(url);
    dataService.getMovies();
    $httpBackend.flush();
    expect(dataService.movies.length).toEqual(5);
    }));
    });
})
```

Now, let me explain the code a bit. Here, I have used **$httpBackend** to mimic the **$http** call. So, when the request comes for the actual API call, it will return me the faked JSON request. **$httpBackend.flush()** will wait till it gets the response, once successfully fetched, then I can go ahead assert the same. So, now the test result looks something like below.



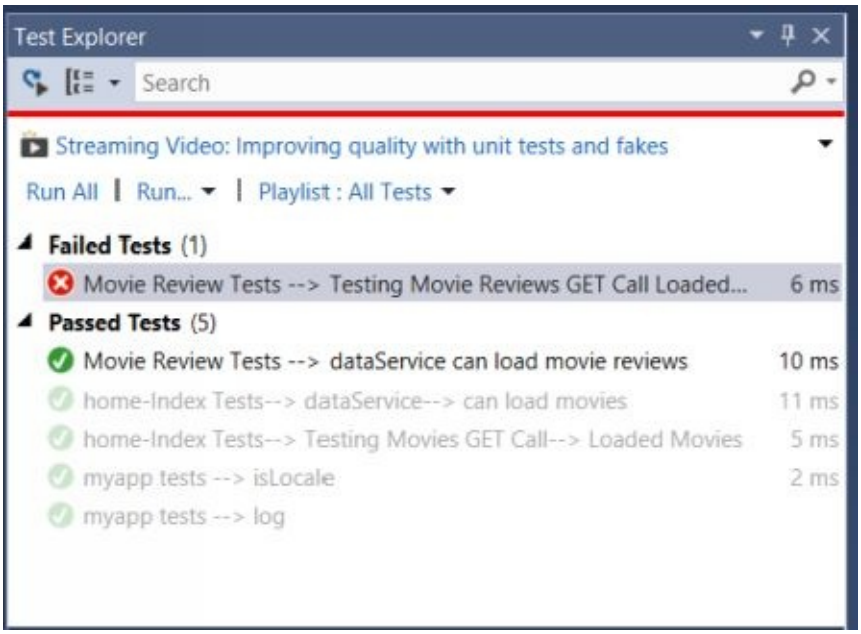

Now, similarly I can go ahead and write tests for other APIs. Below in the snippet I have added another file for movie review test and there I have written below test case.

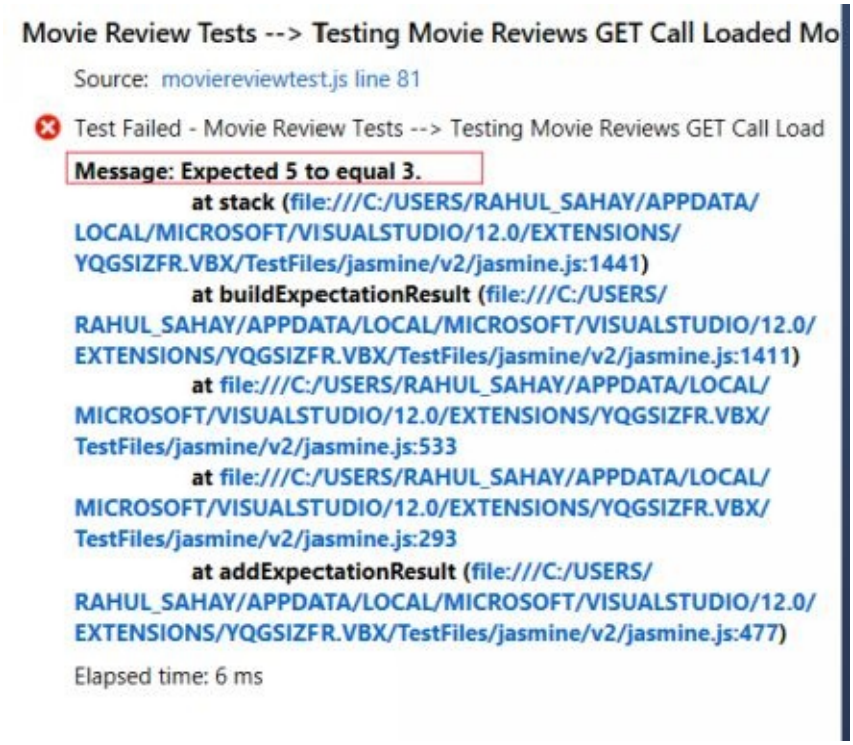
```
/// <reference path="../scripts/jasmine.js" />
/// <reference path="../../moviereview.web/scripts/angular.min.js" />
/// <reference path="../../moviereview.web/scripts/ui-bootstrap-tpls.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-route.min.js" />
/// <reference path="../../moviereview.web/scripts/angular-mocks.js" />
/// <reference path="../../moviereview.web/js/homeindex.js" />
/// <reference path="../../moviereview.web/js/movie-review-edit.js" />

describe("Movie Review Tests —>",function() {

    beforeEach(function () {
```

```
module("homeIndex");
});

//to test individual bits and bytes inside the movie-review-edit
describe("dataService", function () {

it("can load movie reviews", inject(function (dataService) {
//for the 1st Run
expect(dataService.reviews.length).toEqual(0);
}));
});

//$httpbackend service
var $httpBackend;
var url = '/api/MovieReviews/1';

var fakedMovieReviewsResponse = [{
Id: 1,
ReviewerName: "Rahul Sahay",
ReviewerComments: "Awesome Movie. Looks very Nice!",
ReviewerRating: 5,
MovieId: 1
},
{
Id: 2,
ReviewerName: "Nivedita",
ReviewerComments: "Looking Good. Nice One.Review Updated again",
ReviewerRating: 4,
MovieId: 1
},
{
Id: 4,
ReviewerName: "Tester",
ReviewerComments: "Checking",
ReviewerRating: 5,
MovieId: 1
},
{
Id: 5,
ReviewerName: "Tester again",
ReviewerComments: "testing",
ReviewerRating: 5,
```

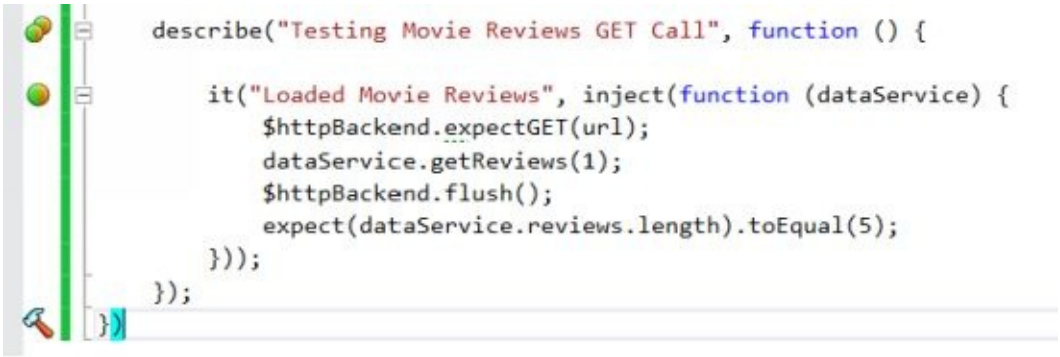
```
    MovieId: 1
    },
    {
    Id: 6,
    ReviewerName: “Tester Again”,
    ReviewerComments: “Testing”,
    ReviewerRating: 4,
    MovieId: 1
    }
    ];
    beforeEach(inject(function ($injector) {

    $httpBackend = $injector.get(“$httpBackend”);

    $httpBackend.whenGET(url)
    .respond(fakedMovieReviewsResponse);

    }));

    afterEach(function () {
    $httpBackend.verifyNoOutstandingExpectation();
    $httpBackend.verifyNoOutstandingRequest();
    });

    describe(“Testing Movie Reviews GET Call”, function () {

    it(“Loaded Movie Reviews”, inject(function (dataService) {
    $httpBackend.expectGET(url);
    dataService.getReviews(1);
    $httpBackend.flush();
    expect(dataService.reviews.length).toEqual(3);
    }));
    });
})
```

Now, here I have mocked 5 objects for the same movie but, I am testing for 3. So, when I ran the same then it failed with the following reason.



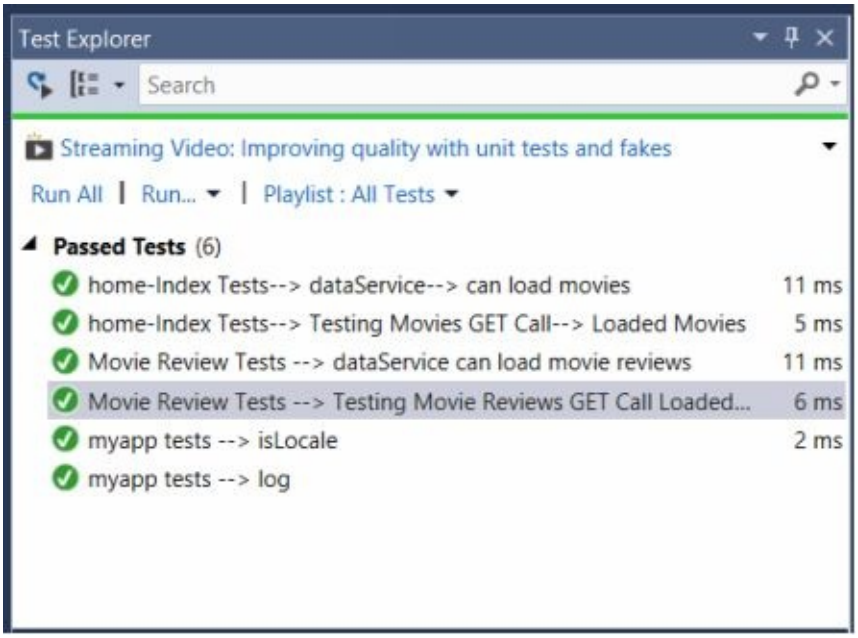




So, now to pass the test when I simply assert for 5, then it will pass as shown below in the screen shot.



```
describe("Testing Movie Reviews GET Call", function () {

    it("Loaded Movie Reviews", inject(function (dataService) {
        $httpBackend.expectGET(url);
        dataService.getReviews(1);
        $httpBackend.flush();
        expect(dataService.reviews.length).toEqual(5);
    }));
});
}
```

With the above change in place. Now, my all tests are passing. Now, let’s go ahead and quickly write one test case for controller.

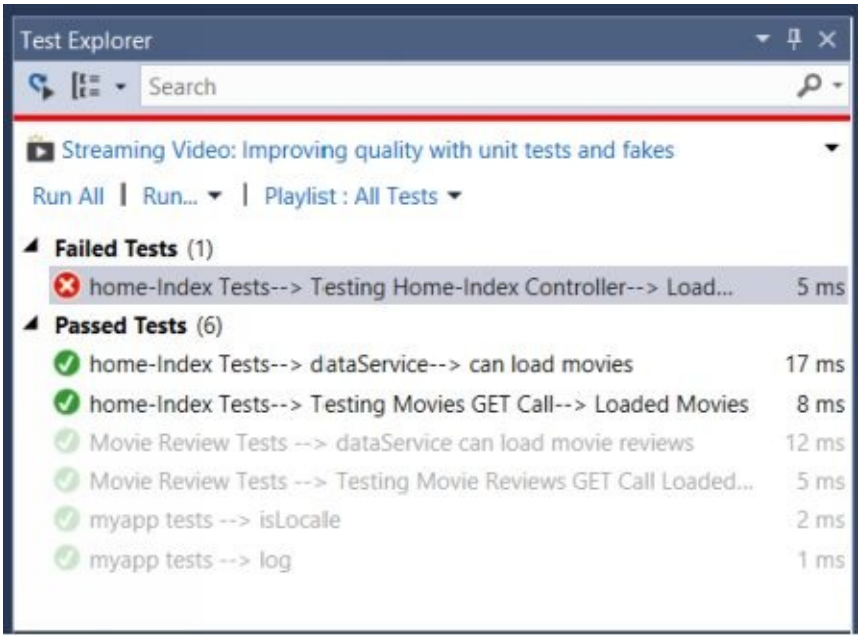


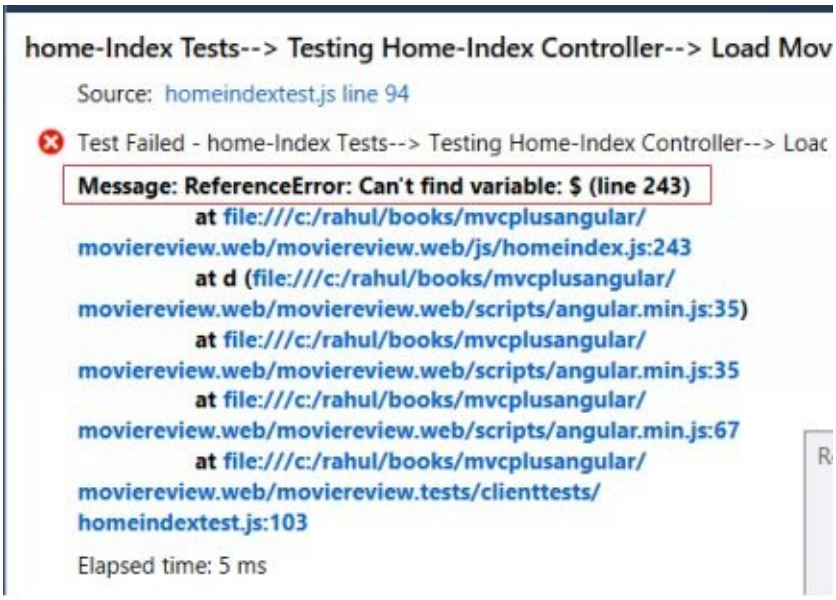

## Writing Controller Tests-

In this section, I am going test my controller. This is also very straight forward. Below is the snippet which I have written for my controller test.

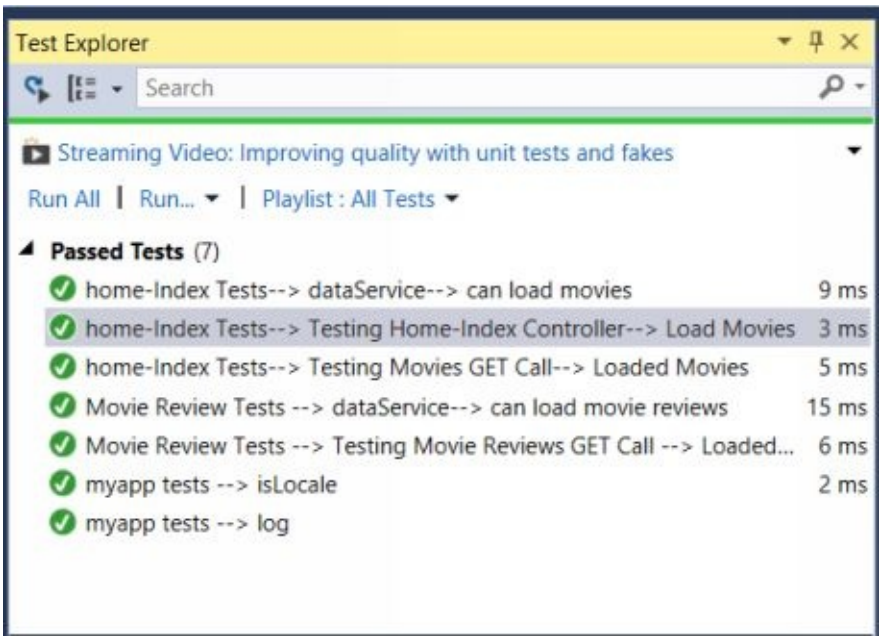
```
describe(“Testing Home-Index Controller—>”, function() {

    it(“Load Movies”, inject(function($controller, $http, dataService) {

    var scopeObj = {};

    $httpBackend.expectGET(url);

    var ctrl = $controller(“homeIndexController”, {
    $scope: scopeObj,
    $http: $http,
    dataService: dataService
    });

    dataService.getMovies();
    $httpBackend.flush();
    expect(ctrl).not.toBeNull();
    expect(scopeObj.data).toBeDefined();
    }));
    });
```

Let me go ahead and explain the code briefly. Since, I have already mocked **dataService** and **$httpBackend**, hence I have used the same. Then, I have simply created the controller which I want to test. In order to inject the controller I have used **$controller**. This will go ahead and give me the instance of the required controller. Now, this also requires other dependencies which are used in the controller like **$scope, $http** etc. When I ran the same, it got failed for a weird reason as shown below.



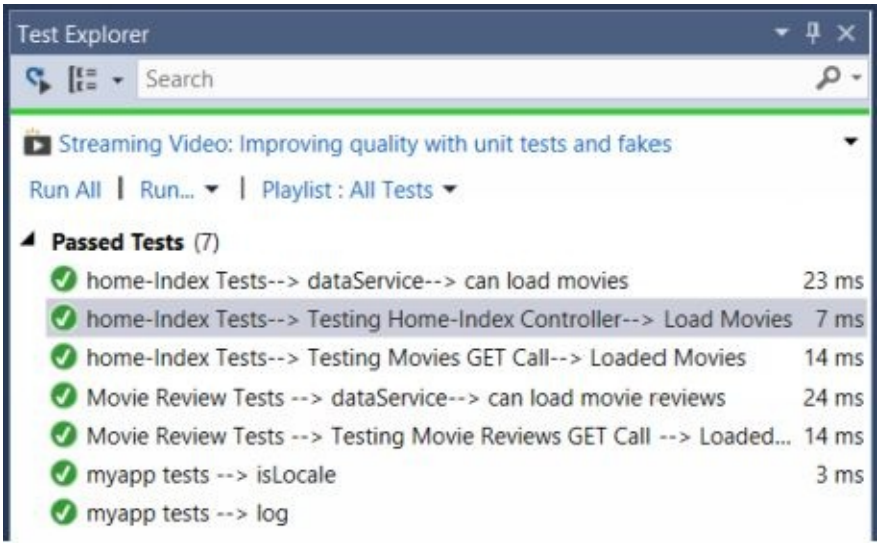




And when I checked my controller code, I saw there is a JQuery Selector call which **Angular.Mock** couldn't resolve. Hence, to fix this I added **JQuery script reference** at the top of my home-index file. Now, with the above change in place when I go ahead and run the tests, then it will produce me the below result.




## Code Coverage-

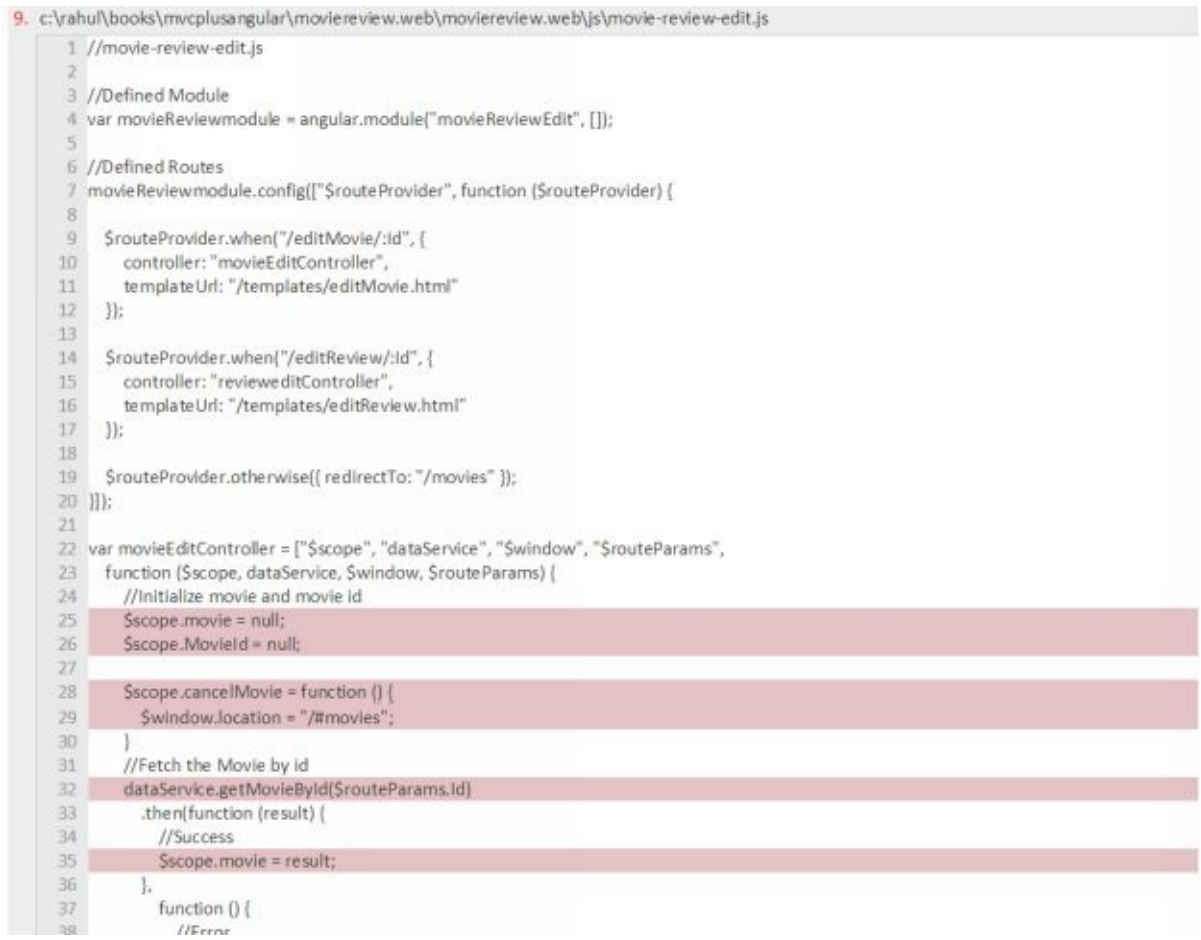
Code coverage is one of the crucial piece of any development. Here in this case I have written all client side code. So, in order to test the code coverage, rule will remain the same. I just need to run all the tests and analyze the code coverage for the same.




Now, when I check code coverage results for my tests, Chutzpah will open a new window in browser with the code coverage results for the client side as shown below in the screen shot.

| Chutzpah code coverage via blanket.js results | Covered/Total Smts. | Coverage (%) |
|---|---|---|
| 1. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\appjstest.js | 3/5 | 60 % |
| 2. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\myapptest.js | 5/5 | 100 % |
| 3. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular.min.js | 117/208 | 56.25 % |
| 4. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\ui-bootstrap-tpls.min.js | 2/2 | 100 % |
| 5. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular-route.min.js | 6/8 | 75 % |
| 6. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\scripts\angular-mocks.js | 219/558 | 39.25 % |
| 7. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\Scripts\jquery-1.10.2.min.js | 3/3 | 100 % |
| 8. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\homeindex.js | 43/148 | 29.05 % |
| 9. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\movie-review-edit.js | 7/51 | 13.73 % |
| 10. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\moviereviewtest.js | 21/21 | 100 % |
| 11. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.tests\clienttests\homeindextest.js | 30/30 | 100 % |
| 12. Total | 456/1039 | 43.89 % |

The ones which are highlighted in red are the ones which are not covered 100%, so when I click on any of this link, it will open the code in browser and show what is covered and what is not



9. c:\rahul\books\mvcplusangular\moviereview.web\moviereview.web\js\movie-review-edit.js

```
1  //movie-review-edit.js
2
3  //Defined Module
4  var movieReviewmodule = angular.module("movieReviewEdit", []);
5
6  //Defined Routes
7  movieReviewmodule.config(["$routeProvider", function ($routeProvider) {
8
9    $routeProvider.when("/editMovie/:Id", {
10     controller: "movieEditController",
11     templateUrl: "/templates/editMovie.html"
12   });
13
14   $routeProvider.when("/editReview/:Id", {
15     controller: "revieweditController",
16     templateUrl: "/templates/editReview.html"
17   });
18
19   $routeProvider.otherwise({ redirectTo: "/movies" });
20 }]);
21
22 var movieEditController = ["$scope", "dataService", "$window", "$routeParams",
23   function ($scope, dataService, $window, $routeParams) {
24     //Initialize movie and movie id
25     $scope.movie = null;
26     $scope.MovieId = null;
27
28     $scope.cancelMovie = function () {
29       $window.location = "/#movies";
30     }
31     //Fetch the Movie by id
32     dataService.getMovieById($routeParams.Id)
33       .then(function (result) {
34         //Success
35         $scope.movie = result;
36       },
37         function () {
38           //Error
```

Hence, this is very simple yet robust to test the quality of your code with your test cases. I hope you enjoyed the journey. Thanks for Joining me.

## Summary-

In this section, we have seen how to get started with client side Unit Test Cases. Like how to setup Chutzpah to run the client side tests. Then we have installed Jasmine framework

to write our sample and application tests. Here, for testing angular dependencies, we have used Angular mock library as well to do mocking for us.

# About the Author

**Rahul Sahay** is a software developer living in Bangalore, India. Rahul has been working in various aspects of the software development life cycle since 7 years, focusing on Microsoft technology-specific development. He has been part of the development in different applications, ranging from client applications to web services to websites.

Rahul is a Senior Consultant at **Capgemini**. But he works for Capgemini's client **Dell R&D**, on their premier e-commerce portal("*http://www.dell.com/account*"). His roles and responsibilities at this project are very tech-oriented like analyzing exiting use cases and taking the new requirements to add features on the existing segment. Prior to Capgemini, he has been associated with Mindtree and TCS. He is also active blogger, his writings can be viewed at *http://myview.rahulnivi.net/*. You can also refer his professional profile @ http://in.linkedin.com/in/rahulsahay19**.** Or follow him at twitter "*@rahulsahay19*".

He has also authored one book on MVC **Hands-On with ASP.NET MVC**; written completely right from the scratch with live demo by hosting the same on azure. You can refer this book at this URL http://ow.ly/JetAi.