This "MvcMovie Case Study" is to help work thru the online lab and help pre-define initial steps & goals.

Background

You are developing a content management system for cataloging movies for learning MVC patterns of development using Visual Studio. This application can be published to Azure and is useable by customers all over the world.

MvcMovie - Get Started: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/

In the lab use the SQL Server Object Explorer but do not make changes to the Web.config file <connectionString>: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/creating-a-connection-string

Business Requirements

1. The user-facing portion of the application is an ASP.NET MVC application. It provides an interface for users to sign in and insert, edit and delete records.
2. Searching by title or genre.
3. Customers require support for Microsoft Internet Explorer 7 and later.
4. UI/UX must be responsive and interactive to screen size and orientation.
5. Column names in views must have spaces between words (Release Date – not ReleaseDate) using Data Annotations.

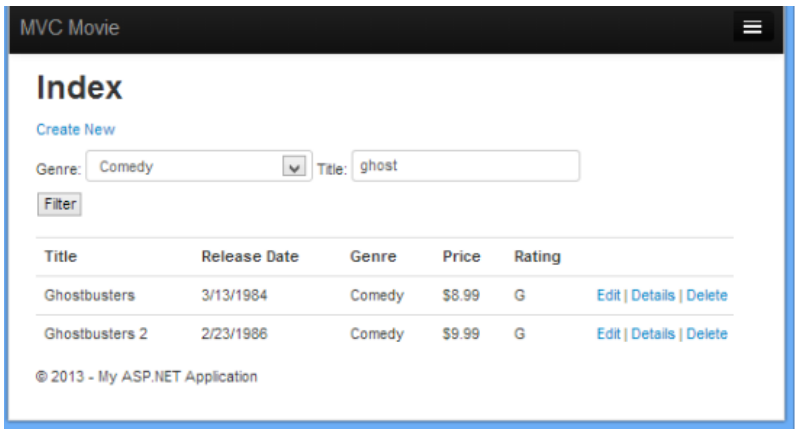
Technical Requirements

User Experience:

The front-end web MVC application enables a user to view a Home page with navigation to About, Contact, Register, Log in or the Movies pages.

The main view of the application is the web page that displays the list of movies.

Users must be able to Search by genre or title.



Styling is done with bootstrap.css.

Compatibility:

Some customers use browsers that do not support HTML5 or CSS3.

Development: These notes are to be used with the online tutorial:

1. Developers must use Microsoft Visual Studio C# MVC ASP 5 templates with Individual Accounts selected to use SQL server for user account management.
    a. Create the Project Namespace MvcMovie.
    b. https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/getting-started
    c. All views must use _layout.cshtml.
    d. Model binding must be used to map parameters from the querystring.
2. URL patterns must use {controller}/{action}/{id}:
    a. http://localhost:1234/movies/details/1
    b. http://localhost:1234/movies/details?id=1
3. Create the "HelloWorldController" test controller and views to examine URL routes and querystring use for "friendliness" and readability for the user and perform passing data tests.
    a. https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-controller
4. Passing Data from the Controller to the View "tests" in the HelloWorldController.cs.

<table>
<tr><td>Use ViewBag to pass data from controller to "generic" view as a string. (no layout)

URL patterns:

http://localhost:1234/HelloWorld

http://localhost:xxxx/HelloWorld/Welcome</td><td>

```
namespace MvcMovie.Controllers
{
    public class HelloWorldController : Controller
    {
        //
        // GET: /HelloWorld/

        public string Index()
        {
            return "This is my <b>default</b> action...";
        }

        //
        // GET: /HelloWorld/Welcome/

        public string Welcome()
        {
            return "This is the Welcome action method...";
        }
    }
}
```

</td></tr>
<tr><td>Pass parameter information from the URL to the controller as querystring data:

/HelloWorld/Welcome?name=Scott&numtimes=4

The "?" (question mark) in the above URL is a separator, and the query strings follow. The "&" character separates query strings.</td><td>Change your Welcome method to include two parameters:

```
public string Welcome(string name, int numTimes = 1)
{
     return HttpUtility.HtmlEncode("Hello " + name +
", NumTimes is: " + numTimes);
}
```

</td></tr>
<tr><td>Pass parameter information from the URL to the controller as route data that matches the RegisterRoutes url: "{controller}/{action}/{id}".

/HelloWorld/Welcome/3?name=Rick

The third URL segment matches the route parameter ID.</td><td>Replace the Welcome method – note the difference:

```
public string Welcome(string name, int ID = 1)
{
    return HttpUtility.HtmlEncode("Hello " + name +
", ID: " + ID);
}
```

</td></tr>
</table>

Add a route to pass both the name and numtimes in parameters as route data in the URL.
 	
/HelloWorld/Welcome/Scott/3
 	
Note {name} is added in MapRoute – url:
	In the App_Start\RouteConfig.cs file, add the "Hello" route under first routes ending "); ":
	
	`routes.MapRoute(`
	`name: "Hello",`
	`url: "{controller}/{action}/{name}/{id}"`
	`);`

Create Helloworld view and use _layout.cshtml "master page".
 	
https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-view
	Change the HelloWorldControllers - Index method to return a View instead of a string:
	
	`Public ActionResult Index()`
	`{`
	`return View();`
	`}`

Right click the Views\HelloWorld folder and click Add, then click MVC 5 View Page with (Layout Razor) to create:
 	
MvcMovie\Views\HelloWorld\Index.cshtml
 	
After making the code additions:
 	
Right click the Index.cshtml file and select View in Browser.
 	
Note: View in Page Inspector has been deprecated in Visual Studio 2015 and replaced by Browser Link
	Add the following highlighted code to Views\HelloWorld\Index.cshtml:
	
	`@{`
	`Layout = "~/Views/Shared/_Layout.cshtml";`
	`}`
	
	`{`
	`ViewBag.Title = "Index";`
	
	`h2>Index</h2>`
	
	`<p>Hello from our View Template!</p>`

Change the "Application name" link at the top of the page and in the browser tab:
 	
The ActionLink visible text from "Application name" to "MVC Movie":
 	
And the controller the link selects from Home to Movies:
 	
Run the application and view the changes – refresh or empty History if no changes show…
	Open the /Views/Shared/_Layout.cshtml change the <title>:
	
	`<title>@ViewBag.Title - Movie App</title>`
	
	Change ActionLink:
	
	`@Html.ActionLink("MVC Movie", "Index", "Movies", null, new { @class = "navbar-brand" })`

Comment out the Layout code in Views\HelloWorld\Index.cshtml as this is handled in the Views_ViewStart.cshtml file.	`*@{`
	`Layout = "~/Views/Shared/_Layout.cshtml";`
	`}*@`

Change the title of the MvcMovie\Views\HelloWorld\Index.cshtml.
 	
Change the highlighted code:
 	
Run and view changes.
	`@{`
	`ViewBag.Title = "Movie List";`
	`}`
	
	`<h2>My Movie List</h2>`
	
	`<p>Hello from our View Template!</p>`

Open to the HelloWorldController.cs file

Change the Welcome method to add a Message and NumTimes value to the ViewBag object.

Build Solution (or Ctrl+Shift+B) to make sure the project is compiled – or the Add View step may error.

```
public ActionResult Welcome(string name, int numTimes
= 1)
        {
            ViewBag.Message = "Hello " + name;
            ViewBag.NumTimes = numTimes;

            return View();
        }
```

Add the Welcome view:

Right click the Views\HelloWorld folder and click Add, then click MVC 5 View Page with Layout (Razor).

Views\HelloWorld\Welcome.cshtml file is created.

Test URL:

/HelloWorld/Welcome?name=Scott&numtimes=4

Replace the markup in the Welcome.cshtml file:

```
@{
    ViewBag.Title = "Welcome";
}
<h2>Welcome</h2>
<ul>
    @for (int i = 0; i < ViewBag.NumTimes; i++)
    {
        <li>@ViewBag.Message</li>
    }
</ul>
```

5. Use Entity Framework Code First to develop the movies database from MovieDBContext with the model to allow future field changes with migrations.

Build the application then:

Add the Movie model

a. MovieDBContext definition

https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-a-model

b. Create MoviesController

c. Scaffold strongly typed views

https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/accessing-your-models-data-from-a-controller

```
namespace MvcMovie.Models
{
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }

    public class MovieDBContext : DbContext
    {
        public DbSet<Movie> Movies { get; set; }
    }
}
```

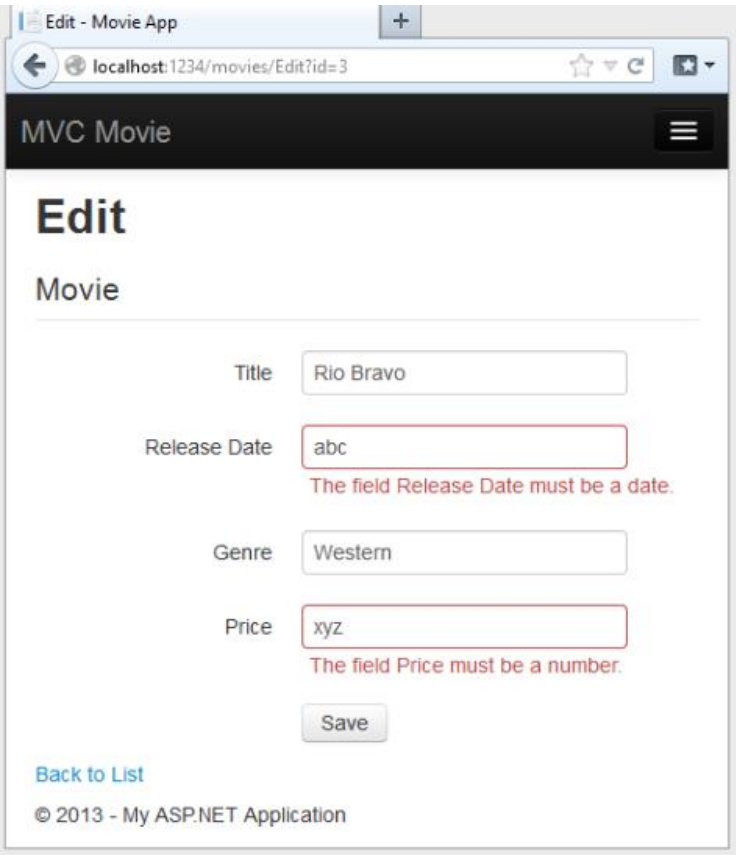
Make the release date look better with Data Annotations:

In Models\Movie.cs file and add the highlighted lines:

```
using System.ComponentModel.DataAnnotations;
    public class Movie
    {
        public int ID { get; set; }
        public string Title { get; set; }

      [Display(Name = "Release Date")]
      [DataType(DataType.Date)]
        [DisplayFormat(DataFormatString = "{0:yyyy-MM-
dd}", ApplyFormatInEditMode = true)]
```

Make the date culture specific	`[DisplayFormat(DataFormatString = "{0:d}", ApplyFormatInEditMode = true)]` `public DateTime ReleaseDate { get; set; }`

6. Edit actions must use HttpPost and ValidateAntiForgeryTokens: This is noted in the review of the Edit actions at the bottom of the page: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/examining-the-edit-methods-and-edit-view
7. Use the Bind attribute to prevent over-posting data to the model.
8. Enable Search by genre or name.
    a. https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/adding-search
9. Forms must validate data on submission. Use ModelState.IsValid to ensure validation occurs on the client or on the server is the client has JavaScript disabled.
10. Display form validation errors with Html.ValidationMessageFor
11. 

Reference

1. MvcMovie Lab: https://docs.microsoft.com/en-us/aspnet/mvc/overview/getting-started/introduction/

Developers Info

2. Page Inspector replaced by Browser Link (2015)
    a. Browser Link is a feature in Visual Studio that creates a communication channel between the development environment and one or more web browsers: https://docs.microsoft.com/en-us/aspnet/core/client-side/using-browserlink
3. Over Posting Attack in MVC: http://www.abhijainsblog.com/2015/04/over-posting-attack-in-mvc.html
4. Overloading and Overriding: https://en.wikibooks.org/wiki/Java_Programming/Overloading_Methods_and_Constructors
5. The difference between "instantiated" and "initialized": https://stackoverflow.com/questions/2330767/what-is-the-difference-between-instantiated-and-initialized
    a. All variables are always given an initial value at the point the variable is declared and are initialized.
    b. An object is an instance of some Class. The act of creating an instance of a Class is called instantiation
6. XSRF/CSRF Prevention (Cross-site request forgery): https://docs.microsoft.com/en-us/aspnet/mvc/overview/security/xsrfcsrf-prevention-in-aspnet-mvc-and-web-pages
7. Understanding LINQ in MVC: https://stackoverflow.com/questions/29824798/need-help-understanding-linq-in-mvc

```
public ActionResult Index(string id)
{
    string searchString = id;
    var movies = from m in db.Movies
                 select m;
    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(s => s.Title.Contains(searchString));
    }
    return View(movies);
}
```

The variables `m` and `s` are variables for each instance of `Movie` within the `db.Movies`

Conceptually these are similar to using and sql alias `m` in the following sql:

```
select m.* from Movies m
```

When you later apply the `where` clause you're conceptually ending up with:

```
select * from (
    select m.* from Movies m
) s
where s.Title like '%' + searchString + '%'
```

the value of `movies` is an `IQueryable` or similar.

    a.
8. Space