




# **MVC and databases tutorial**

## **Part 2 – Validation, Filtering, Styling and Images**

# Before you begin

- Please complete Part 1 of this tutorial, so that you have the application that is the starting point for this tutorial
  - If anything has gone wrong with that code, you can find it on the VLE in the Week 4 - ASP.NET folder. It is in the note MVC Movies – Main Functionality

# The application being created

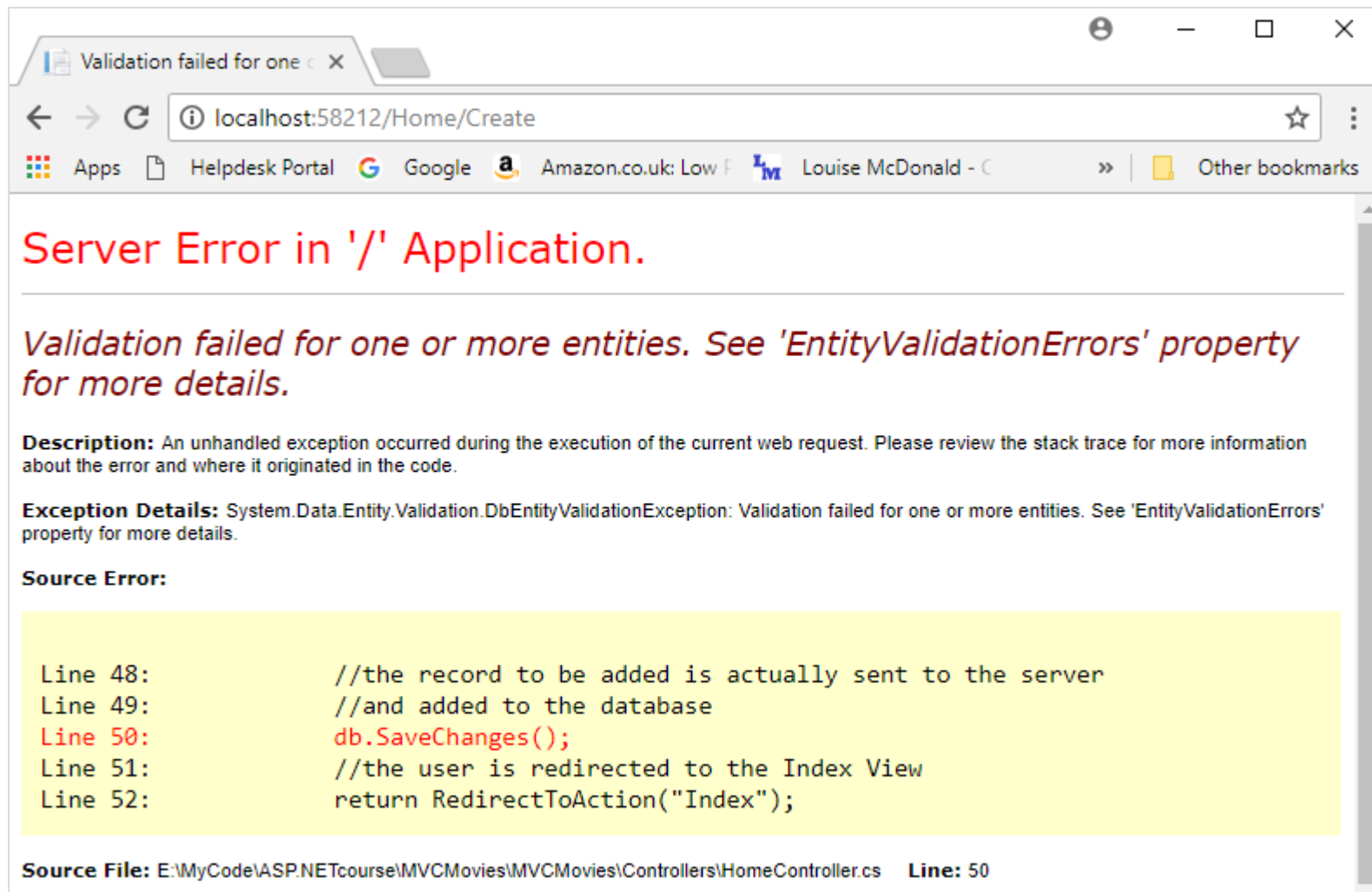
Movies DB   Home   About   Contact					
Movies					
<a href="#">Create new movie</a>					
Genre: <input type="text" value="All"/> Title: <input type="text"/> <input type="button" value="Filter"/>					
Title	Release Date	Genre	Price		
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Force Awakens	16-Dec-2015	Science Fiction	£9.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>

- This is an MVC5 application with a single-table SQL database
- It has Create, Read, Update and Delete (CRUD) functionality
- It has filtering and searching
- A database-first approach is used to create the database, which is then attached to the project by creating an ADO.NET Entity Framework model

# Validation

# User input validation

- If you were to try and add a blank record in the Create view, you would get an error like this
- It's important to stop users doing things like this, so that the database doesn't contain invalid data and the application doesn't crash
- User input validation prevents the user from adding records with incomplete or invalid data to the database
  - This needs to be implemented in the Create and Edit action methods and views
  - We also need to protect other views from invalid record ids (because these appear in the URL, they could be changed or deleted by a user)



# Data annotations

- MVC's approach to user input validation starts with the rules for each property being entered in the model class
  - This means the rules are only entered once and are then consistent throughout the application
- Data annotations are used to implement the rules
- The simplest data annotation is to make a property required by adding [Required] above the property
  - This should be done for all properties that are NOT NULL in the database
    - These will be all the properties that don't have Nullable in the model class, with the possible exception of strings, which can always take null values in C#
    - The Id doesn't need data annotations, as it is never entered by users
    - Properties with nulls allowed in the database can be required in the software if you choose
  - You will also need using System.ComponentModel.DataAnnotations;

```
using System;
using System.ComponentModel.DataAnnotations;

public partial class Movie
{
    public int Id { get; set; }

    [Required]
    public string Title { get; set; }

    public Nullable<System.DateTime> ReleaseDate { get; set; }

    [Required]
    public string Genre { get; set; }

    public Nullable<decimal> Price { get; set; }
    public string ImageUrl { get; set; }
}
```

# Data annotations

```
public partial class Movie
{
```

```
    public int Id { get; set; }
```

```
    [Required]
```

```
    [StringLength(60, MinimumLength = 2)]
```

```
    public string Title { get; set; }
```

```
    [Display(Name = "Release Date")]
```

```
    [DataType(DataType.Date)]
```

```
    [DisplayFormat(DataFormatString = "{0:yyyy-MM-dd}", ApplyFormatInEditMode = true)]
```

```
    public Nullable<System.DateTime> ReleaseDate { get; set; }
```

```
    [Required]
```

```
    [StringLength(30)]
```

```
    [RegularExpression(@"^[A-Z]+[a-zA-Z'\s\-\']*")]
```

```
    public string Genre { get; set; }
```

```
    [Range(0, 1000)]
```

```
    [DataType(DataType.Currency)]
```

```
    public Nullable<decimal> Price { get; set; }
```

```
    public string imageUrl { get; set; }
```

```
}
```

Maximum and minimum length of data input

Setting the label for the data to something other than the database column heading, e.g. to have spaces between words or make the label more user-friendly

This changes the display of the date from date and time to just date

This changes the display format of the date. It works properly in Firefox and IE, but not in Chrome or Edge

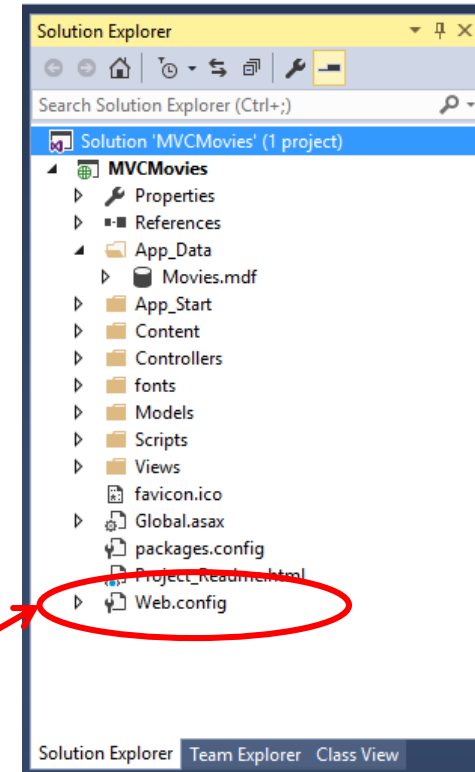
This is a regular expression, which can be used to check patterns of text, e.g. email addresses. The pattern here is: start with a capital letter, then have 0 or more letters, spaces, hyphens or apostrophes

This sets the range of any numerical value

This displays a currency symbol – see the next slide to choose which currency symbol is displayed

# Setting the £ sign

- You will have noticed that Price now has a currency symbol, because it has a data annotation to say that it is currency
  - The currency symbol that you see depends on the culture settings of your machine
    - If the culture on your machine is set to British English, you will see a £ sign. If it's set to another country and language, you will see the currency symbol for that country
  - The culture on the servers of your hosting service may be different, which means the currency symbol could change when you deploy your application
  - That means it is important to set the culture in your application, as this will override the machine settings on your local machine or hosting service
  - You can do this in the Web.config file (the one at the bottom of the file tree in the Solution Explorer):
    - Add this line inside the <system.web> tags:  
<globalization uiCulture="en-GB" culture="en-GB" />
    - Other culture codes can be found at:  
[https://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo\(vs.71\).aspx](https://msdn.microsoft.com/en-us/library/system.globalization.cultureinfo(vs.71).aspx)



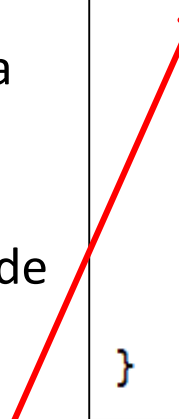
```
<system.web>
  <compilation debug="true" targetFramework="4.5.2" />
  <httpRuntime targetFramework="4.5.2" />
  <globalization uiCulture="en-GB" culture="en-GB" />
</system.web>
```



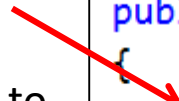
# Making data annotations work

- You have added rules for user input validation as data annotations, but the application doesn't yet enforce them, so it will still crash if invalid data is entered
- To make this happen, you need to add some extra code to the POST Create and Edit action methods in the Home Controller
- Please add an `if (ModelState.IsValid)` statement and put the code for the database changes and the return to the Index view inside it
  - If the data isn't valid, then the user should be able return to the Create or Edit view and have a chance to fix the faulty data
- Now if you run the code and try to add an empty record or edit an existing record so that it's invalid, you will stay in the Create or Edit view
  - However, a user might not know what had gone wrong, so it would be helpful to give them some error messages

```
[HttpPost]
public ActionResult Create(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Movies.Add(movie);
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```



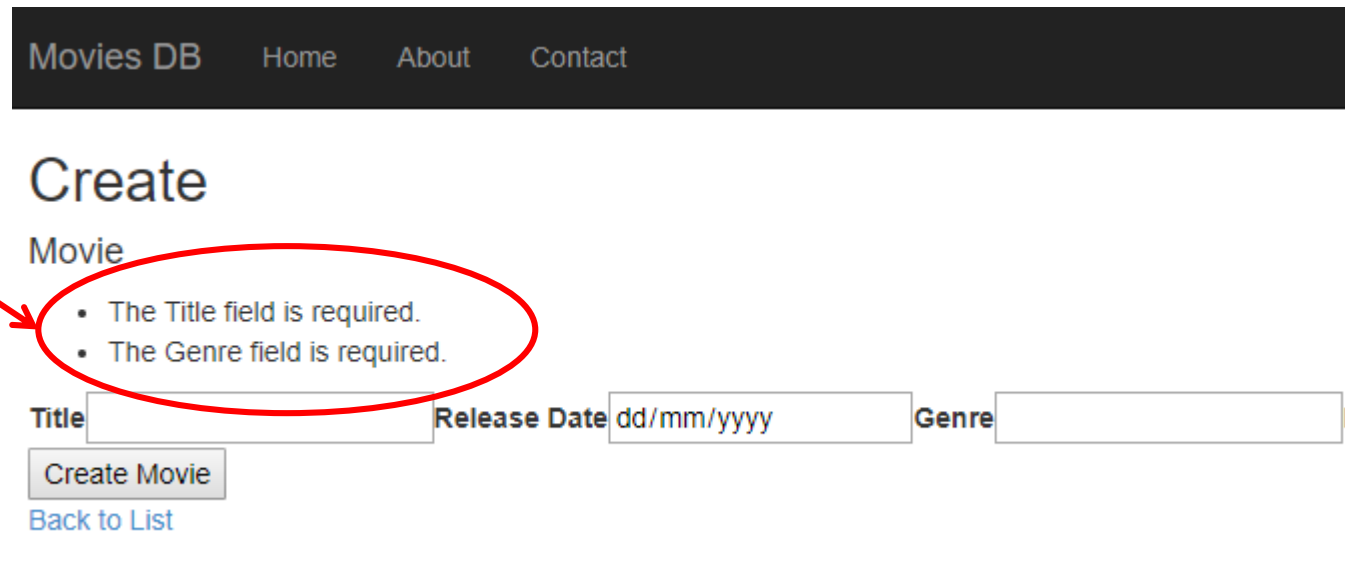
```
[HttpPost]
public ActionResult Edit(Movie movie)
{
    if (ModelState.IsValid)
    {
        db.Entry(movie).State = EntityState.Modified;
        db.SaveChanges();
        return RedirectToAction("Index");
    }
    return View(movie);
}
```



# Adding error messages

- The easiest way to add error messages is to add `@Html.ValidationSummary()` to the Create and Edit views, just under the Movie heading
- Now, if you try to add a blank record you will get error messages

```
@using (Html.BeginForm())  
{  
    <h4>Movie</h4>  
    @Html.ValidationSummary()  
}
```



The screenshot shows a web application with a dark navigation bar at the top containing links for 'Movies DB', 'Home', 'About', and 'Contact'. Below the navigation bar is a section titled 'Create Movie'. Under this heading, there is a red oval containing two bullet points: '• The Title field is required.' and '• The Genre field is required.'. Below the oval are three input fields: 'Title' (empty), 'Release Date' (with a placeholder 'dd/mm/yyyy'), and 'Genre' (empty). At the bottom of the form are two buttons: 'Create Movie' and 'Back to List'.

# Customising error messages

- At the moment, you are using the default error messages from the data annotations
  - Some of these are not very user-friendly
  - You can add a custom error message to any data annotation

## Create Movie

- The field Genre must match the regular expression `^[A-Z]+[a-zA-Z'-'\s\-\]*$`.

Title  Release Date  Genre

[Back to List](#)

```
[RegularExpression(@"^[A-Z]+[a-zA-Z'-'\s\-\]*$", ErrorMessage = "Genre must begin with a capital letter")]
```

## Create

### Movie

- Genre must begin with a capital letter

Title  Release Date  Genre

[Back to List](#)

# Dealing with missing or invalid ids

- MVC routing means that record ids are passed between controllers and views using the URL
- That means that the id in the URL could be changed
  - If an invalid id were entered, the code would crash
  - A similar crash occurs if the id is blank

localhost:58212/Home/Details/99

Server Error in '/' Application.

*Object reference not set to an instance of an object.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.NullReferenceException: Object reference not set to an instance of an object.

**Source Error:**

```
Line 25: @*Link*  
Line 26: Edit  
Line 27: @Html.ActionLink("Edit", "Edit", new { id = Model.Id })  
Line 28: @*Link*  
Line 29: @Html.ActionLink("Delete", "Delete", new { id = Model.Id })
```

Source File: E:\MyCode\ASP.NET\mvc-movies\Controllers\HomeController.cs

localhost:58212/Home/Details

Server Error in '/' Application.

*The parameters dictionary contains a null entry for parameter 'id' of non-nullable type 'System.Int32' for method 'System.Web.Mvc.ActionResult Details(Int32)' in 'MVCMovies.Controllers.HomeController'. An optional parameter must be a reference type, a nullable type, or be declared as an optional parameter.*

**Parameter name:** parameters

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.ArgumentException: The parameters dictionary contains a null entry for parameter 'id' of non-nullable type 'System.Int32' for method 'System.Web.Mvc.ActionResult Details(Int32)' in 'MVCMovies.Controllers.HomeController'. An optional parameter must be a reference type, a nullable type, or be declared as an optional parameter.

**Parameter name:** parameters

**Source Error:**

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

# Dealing with missing or invalid ids

- It is possible to prevent these kinds of crashes
- In the GET methods which get a movie from the database – Details, Edit and Delete:
  - Make the input parameter nullable by putting a question mark after the int
  - Add an if statement to check if the id is null
    - If it is, call a standard HTML error page
    - You will need to add using System.Net at the top of the Home Controller to deal with the compilation error on HttpStatusCode
  - After the database search for the record, add an if statement to check that the record exists
    - If it doesn't, call a standard HTML error page

```
public ActionResult Details(int? id)
{
    //if a null id is passed in, display an HTML error page
    //add using System.Net to get rid of the error on HttpStatusCode
    if (id == null)
    {
        return new HttpStatusCodeResult(HttpStatusCode.BadRequest);
    }
    //get the record from the database using its id
    Movie movie = db.Movies.Find(id);
    //if an invalid id was passed in, and the record doesn't
    //exist in the database, display an HTML error page
    if (movie == null)
    {
        return HttpNotFound();
    }
    //pass the data to the Details view to be displayed
    return View(movie);
}
```

# Dealing with missing or invalid ids

- Now if an id is missing or invalid, the problem is handled gracefully with standard error pages which give the user intelligible information about what has gone wrong

**HTTP Error 404.0 - Not Found**  
The resource you are looking for has been removed, had its name changed, or is temporarily unavailable.

**Most likely causes:**

- The directory or file specified does not exist on the Web server.
- The URL contains a typographical error.
- A custom filter or module, such as URLScan, restricts access to the URL.

**Things you can try:**

- Create the content on the Web server.
- Review the browser URL.
- Check the failed request tracing log and see which module generated the error.

**Detailed Error Information:**

Module	ManagedPipelineHandler	Requested URL	localhost:58212/Home/Details/99
Notification	ExecuteRequestHandler	Physical Path	\\localhost\inetpub\wwwroot\Home\Details\99.aspx
Handler	System.Web.Mvc.MvcHandler	Logon Method	Anonymous
Error Code	0x00000000	Logon User	Anonymous
		Request Tracing Directory	VC:\Users\Louise\Documents\IISExpress\TraceLogFiles\MVCMOVIES

**HTTP Error 400.0 - Bad Request**  
**Bad Request**

**Most likely causes:**

- The request is malformed or the request body is too large.

**Things you can try:**

- Check the failed request tracing logs for additional information about this error. For more information, click [here](#).

**Detailed Error Information:**

Module	ManagedPipelineHandler	Requested URL	http://localhost:58212/Home/Details
Notification	ExecuteRequestHandler	Physical Path	E:\MyCode\ASP.NETcourse\MVCMovies\MVCMovies\Home\Details
Handler	System.Web.Mvc.MvcHandler	Logon Method	Anonymous
Error Code	0x00000000	Logon User	Anonymous
		Request Tracing Directory	C:\Users\Louise\Documents\IISExpress\TraceLogFiles\MVCMOVIES

# Searching

# Searching by title

- It's helpful if users can search for a movie by all or part of its title
- You can implement this by adding a text box inside a form to the Index view and adding some extra logic to the Index action method in the Home Controller
- Start by adding a form, a textbox and a button to the Index view under the <h2>Movies</h2> heading
- When you run the code, it should look like this, but the search won't work yet

```
@using (Html.BeginForm())  
{  
    <p>  
        Title: @Html.TextBox("SearchString")  
        <input type="submit" value="Filter" />  
    </p>  
}
```

## Movies

[Create new movie](#)

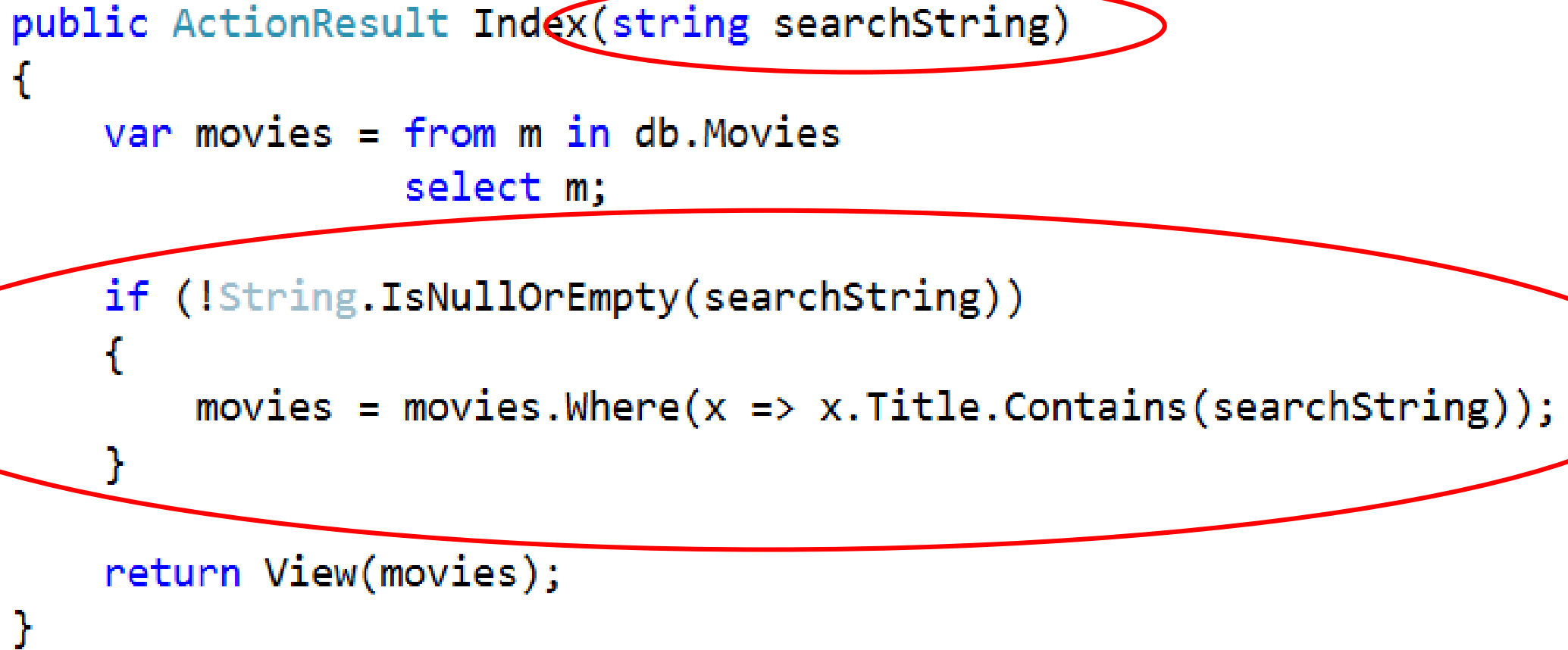
Title:

Title	Release Date	Genre	Price
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99
The Force Awakens	18-Dec-2015	Science Fiction	£9.99
Hidden Figures	25-Dec-2016	Biographical	£12.99

This is `Html.TextBox` (not `Html.TextBoxFor`) because we are not working directly with the model in this part of the code



# Searching by title



```
public ActionResult Index(string searchString)
{
    var movies = from m in db.Movies
                  select m;

    if (!String.IsNullOrEmpty(searchString))
    {
        movies = movies.Where(x => x.Title.Contains(searchString));
    }

    return View(movies);
}
```

- The form in the view passes a string parameter called searchString to the Index action method in the Home Controller
- You then use this parameter (if it isn't empty) to reduce the list of movies to include only those that contain the search string

# Searching by title

- Run the code and type a letter in the text box (ideally one that is only in some of your movie titles)
- Click the filter button and your list should be filtered to show only those movies which contain the search string

## Movies

[Create new movie](#)

Title: g

Filter

Title	Release Date	Genre	Price
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99
The Force Awakens	18-Dec-2015	Science Fiction	£9.99
Hidden Figures	25-Dec-2016	Biographical	£12.99

## Movies

[Create new movie](#)

Title: g

Filter

Title	Release Date	Genre	Price
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99
Hidden Figures	25-Dec-2016	Biographical	£12.99

# Filtering

# Filtering by genre

- It's helpful if users can filter the data in your application, especially in a real application with dozens of records
- Genre is a good category to filter on, as there are likely to be at least a few records in each genre in a larger database
- Start by adding a dropdown list to the form in the Index view, just above the text box for the title search
- Don't run the code yet – if you do, you will get an error, as the SelectList hasn't been created yet

```
@using (Html.BeginForm())
{
    <p>
        Genre: @Html.DropDownList("movieGenre", "All")
        Title: @Html.TextBox("SearchString")
        <input type="submit" value="Filter" />
    </p>
}
```

movieGenre is the name of a SelectList that will contain the options for the dropdown list. It will be created in the Index action method and put in the ViewBag so the view can find it. The name of the input parameter of the Index action method should also be the same as this

"All" is the text that will appear at the top of the dropdown list when nothing has been selected

# Filtering by genre

- In the Index action method, add a string input parameter to get the genre selected from the form in the Index view
- Then create the SelectList by:
  - Creating an empty list of type string
  - Getting the genres from the db in order
  - Adding unique genres to the list
  - Creating a SelectList using the list of genres and putting it into the ViewBag so the view can find it
- Further down, filter the movies list by the selected genre that was passed in from the Index view

```
public ActionResult Index(string searchString, string movieGenre)
{
    //creating the SelectList for the Genre dropdown list
    List<string> genreList = new List<string>();
    var genreQuery = from g in db.Movies
                     orderby g.Genre
                     select g.Genre;
    genreList.AddRange(genreQuery.Distinct());
    ViewBag.movieGenre = new SelectList(genreList);
    //getting all the movies from the db
    var movies = from m in db.Movies
                 select m;
    //filtering by genre
    if (!String.IsNullOrEmpty(movieGenre))
    {
        movies = movies.Where(x => x.Genre == movieGenre);
    }
    //searching by title
    if (!String.IsNullOrEmpty(searchString))
```

When you declare the List, there will be a compilation error. Add using System.Collections.Generic; at the top of the HomeController to get rid of it

# Filtering by genre

- Now run the code and you should see a dropdown list for Genre
- Make a selection from the list and click the Filter button
- Only movies of the correct genre should be shown
- Note that each genre is only shown once in the dropdown list, even if there are several movies of the same genre in the database
  - This is because `.Distinct()` was used to remove duplicates from the genre list
- Also, the genre list will be updated every time a movie with a different genre is added, because the genres were fetched from the database (not hard-coded)

## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl	
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99		<a href="#">Edit</a>
The Fault in Our Stars	12-Dec-2015	Science Fiction	£9.99		<a href="#">Edit</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99		<a href="#">Edit</a>
The Imitation Game	28-Nov-2014	Biographical	£7.99		<a href="#">Edit</a>

## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl	
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99		<a href="#">Edit Record</a>

# A bit about security

- MVC has some features that help make your application more secure
  - Security is a big topic and it is unlikely that these features will protect against all attacks, but they will help a bit
- Anti-forgery tokens can protect against CSRF (Cross Site Request Forgery) attacks
  - An anti-forgery token is placed in the forms of the Create, Edit and Delete views (i.e. any views that lead to POST action methods that change the contents of the database)
  - An annotation [ValidateAntiForgeryToken] is put before each of the corresponding action methods
  - You can test this by commenting out one of the anti-forgery tokens in a view and trying to complete that action – you should get an error like this

```
@using (Html.BeginForm())  
{  
    //helps protect against CSRF  
    //(Cross Site Request Forgery) attack  
    @Html.AntiForgeryToken()  
    <h4>Movie</h4>  
    @Html.ValidationSummary()  
}
```

```
[HttpPost]  
[ValidateAntiForgeryToken]  
public ActionResult Create(Movie movie)  
{  
    if (ModelState.IsValid)
```

Server Error in '/' Application.

*The required anti-forgery form field "\_\_RequestVerificationToken" is not present.*

**Description:** An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

**Exception Details:** System.Web.Mvc.HttpAntiForgeryException: The required anti-forgery form field "\_\_RequestVerificationToken" is not present.

**Source Error:**

An unhandled exception was generated during the execution of the current web request. Information regarding the origin and location of the exception can be identified using the exception stack trace below.

**Stack Trace:**

```
[HttpAntiForgeryException (0x80004005): The required anti-forgery form field "  
System.Web.Helpers.AntiXsrf.TokenValidator.ValidateTokens(HttpContextBase h  
System.Web.Helpers.AntiXsrf.AntiForgeryWorker.Validate(HttpContextBase http
```

# A bit about security

- Another security feature is binding, which can prevent cross-site scripting attacks
- If you add binding to the input parameter list in the POST Create and Edit methods, this means that only data for those fields will be allowed into the action method

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "Id,Title,ReleaseDate,Genre,Price,ImageUrl")]Movie movie)
{
    if (ModelState.IsValid)
```



# Styling

# Styling the application

- The application works now, but looks messy. It needs styling with CSS and Bootstrap

[Movies DB](#) [Home](#) [About](#) [Contact](#)

## Details

Movie

**Title**  
Grosse Pointe Blank

**Release Date**  
11-Apr-1997

**Genre**  
Comedy

**Price**  
£4.99

[Edit Record](#) | [Back to List](#)

---

© 2018 - Movies DB Ltd.

[Movies DB](#) [Home](#) [About](#) [Contact](#)

## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	<a href="#">Edit Record Details</a>
The Force Awakens	18-Dec-2015	Science Fiction	£9.99	<a href="#">Edit Record Details</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99	<a href="#">Edit Record Details</a>
The Imitation Game	28-Nov-2014	Biographical	£7.99	<a href="#">Edit Record Details</a>

---

© 2018 - Movies DB Ltd.

[Movies DB](#) [Home](#) [About](#) [Contact](#)

## Create

Movie

Title  Release Date  Genre  Price  ImageUrl

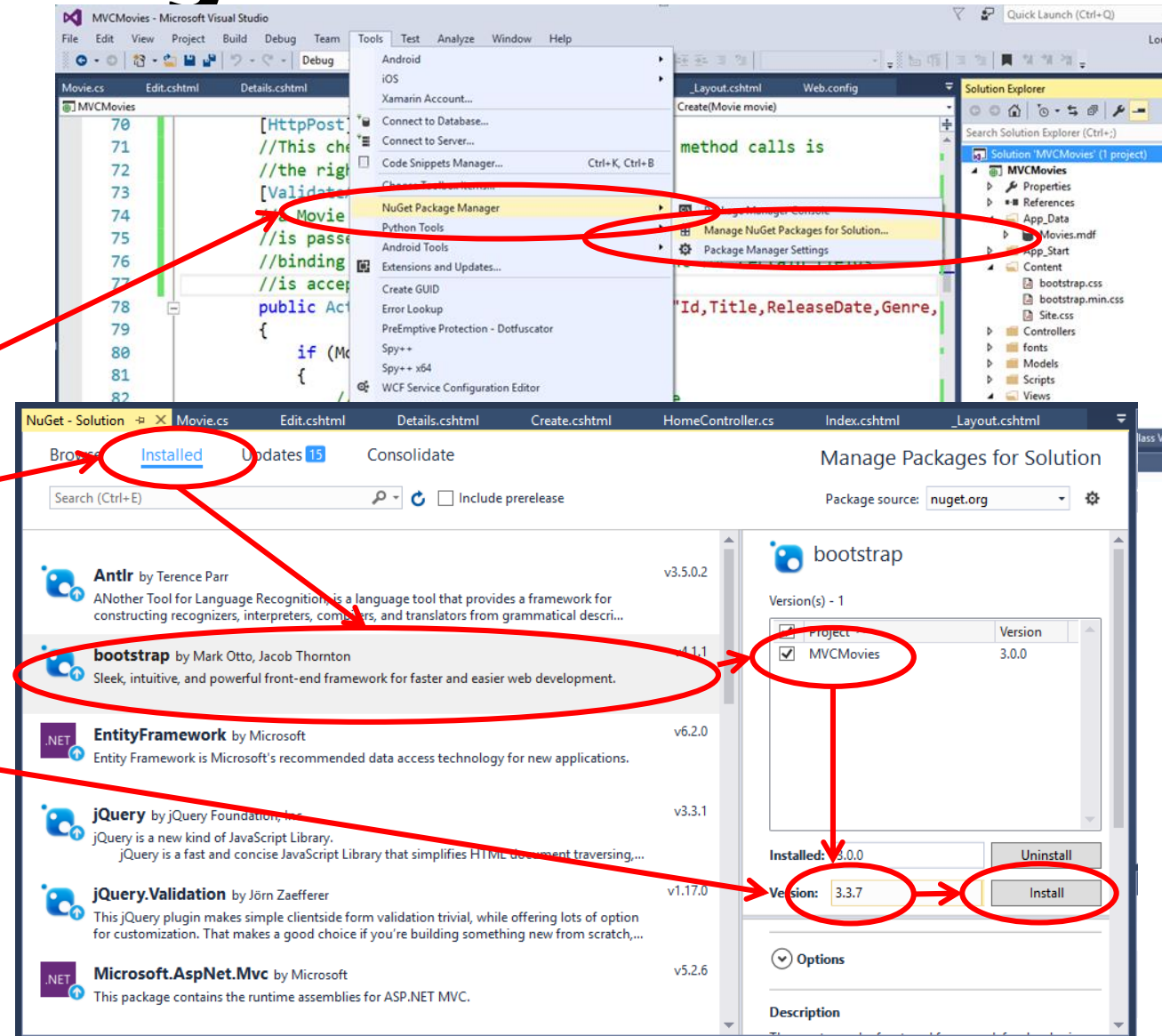
[Back to List](#)

---

© 2018 - Movies DB Ltd.

# Adding a Nuget package

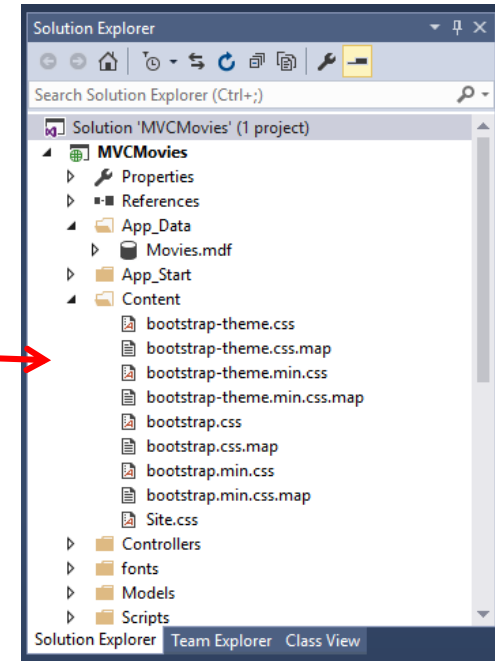
- The version of Bootstrap that comes with the application template that we used is rather outdated, so it's useful to update it
- From the Tools menu, select NuGet Package Manager then Manage Nuget Packages for Solution
- Under the installed tab, select Bootstrap
  - Check the box next to the project name
  - In the Version box, click and select **version 3.3.7** from the dropdown list
    - Do NOT use later versions, unless you have checked with a trainer, as at least one of them causes serious errors in Visual Studio
  - Then click the Install button



# Adding a Nuget package

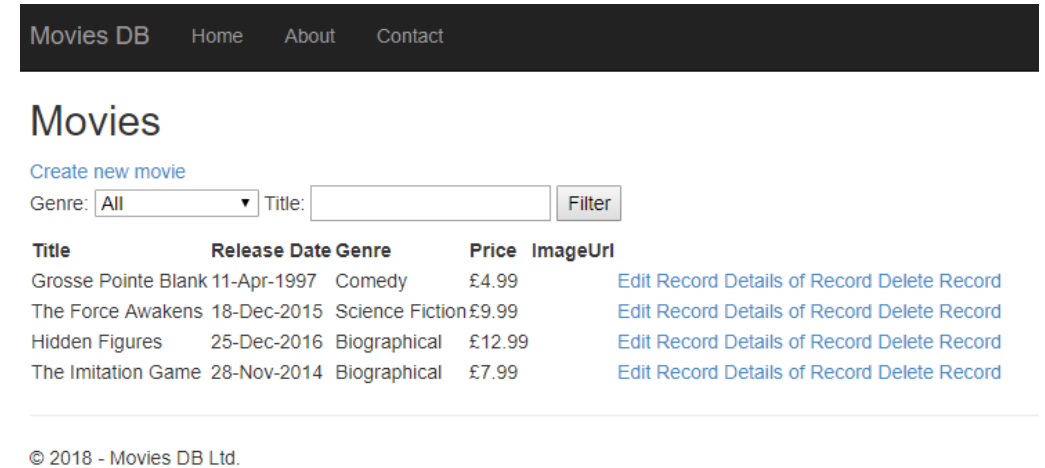
- In the Content folder in the Solution Explorer, you will now see more Bootstrap files. Font and JavaScript files have also been installed
- There is also a Site.css file in the Content folder. You can use this for your CSS or add a new file to the folder
- CSS, JavaScript and any other files that are used throughout the application, are linked in the \_Layout.cshtml in the <head> section
  - Styles.Render() is used to link to all the files in a bundle
  - If you want to control the order that files are linked in, comment out Styles.Render() and use <link> tags to link your files in the correct order
  - You can find more details of which files are included in each bundle in the BundleConfig.cs file in the App\_Start folder

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <!--This appears in the browser tab *** changed in tutorial -->
  <title>Movies DB - @ViewBag.Title</title>
  <!--This links the CSS and JavaScript files, but you may want to add
        your own links so you have control of the order in which files are
        linked -->
  @Styles.Render("~/Content/css")
  @Scripts.Render("~/bundles/modernizr")
</head>
<body>
  <!--menu bar (Bootstrap)-->
  <div class="navbar navbar-inverse navbar-fixed-top">
```



# Styling the Index view

- The Index view works well, but is messy
- In the <table> tag, add class="table"
- This is the result



Movies DB Home About Contact

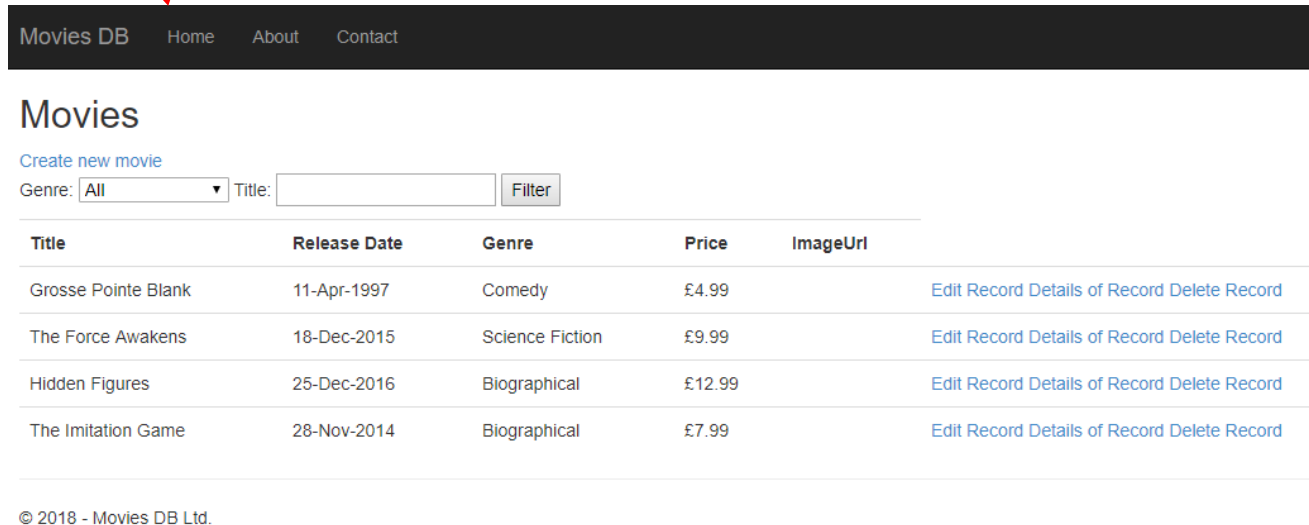
## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Force Awakens	18-Dec-2015	Science Fiction	£9.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Imitation Game	28-Nov-2014	Biographical	£7.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>

© 2018 - Movies DB Ltd.



Movies DB Home About Contact

## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Force Awakens	18-Dec-2015	Science Fiction	£9.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Imitation Game	28-Nov-2014	Biographical	£7.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>

© 2018 - Movies DB Ltd.

# Styling the Create view

- The Create view works well, but looks terrible
- The labels and input boxes should be arranged vertically
- Start by adding a div with class="form-horizontal" above the Movie heading
  - The closing div tag should go just before the } at the end of the form
- Add a horizontal rule below the Movie heading
- Add the parameters: true, "", new { @class = "text-danger" } to Html.ValidationSummary
  - True means it will only show model-level errors (not errors for individual fields, as these error messages will be added below each field)
  - "" is where a message could be added
  - new { @class = "text-danger" } is where a CSS class is added to make the text red in case of an error

Movies DB Home About Contact

## Create

Movie

Title  Release Date  Genre  Price  ImageUrl

[Create Movie](#)

[Back to List](#)

© 2018 - Movies DB Ltd.

```
@using (Html.BeginForm())  
{  
    //helps protect against CSRF (Cross Site Request Forgery) attack  
    @Html.AntiForgeryToken()  
  
    <div class="form-horizontal">  
        <h4>Movie</h4>  
        <hr/>  
        @Html.ValidationSummary(true, "", new { @class = "text-danger" })  
    </div>  
}
```

# Styling the Create view

- Each label and text box pair should be styled as shown below

```
<div class="form-group">
  @Html.LabelFor(m => m.Title, htmlAttributes: new { @class = "control-label col-md-2" })
  <div class="col-md-10">
    @Html.EditorFor(m => m.Title, new { htmlAttributes = new { @class = "form-control" } })
    @Html.ValidationMessageFor(model => model.Title, "", new { @class = "text-danger" })
  </div>
</div>
```

Brings the label and input box over to the left

Aligns the label

Creates a column 2 units wide

Creates a column 10 units wide

Adds an error message for the field

Makes the text red

Styles the input box with grey edges, rounded corners, etc.

Error message text could be added here. If it's left blank, the error messages from the data annotations in the model class are used

# Styling the Create view

- The button should be styled as shown below

```
<div class="form-group">  
  <div class="col-md-offset-2 col-md-10">  
    <input type="submit" value="Create Movie" class="btn btn-default" />  
  </div>  
</div>
```

Aligns the button with the other form elements

Moves the next column over by 2 units

Creates a column 10 units wide

Styles the button with rounded corners and light background, etc.

- The Create view now looks much better
- The same styling should now be applied to the Edit view

Create  
Movie

---

Title	<input type="text"/>
Release Date	<input type="text" value="dd/mm/yyyy"/>
Genre	<input type="text"/>
Price	<input type="text"/>
ImageUrl	<input type="text"/>

[Back to List](#)



# Styling the Details view

- The styling of the Details view could also be improved
- Add a horizontal rule `<hr/>` under the `<h4>Movie</h4>` heading
- Add a `class="dl-horizontal"` to the `<dl>` tag
- The labels and information in the Details view are now arranged horizontally, which is a bit easier to read

```
<h2>Details</h2>
```

```
<h4>Movie</h4>
```

```
<hr/>
```

```
<dl class="dl-horizontal">
```

```
<dt>@Html.DisplayNameFor(m => m.Title)</dt>
```

```
<dd>@Html.DisplayFor(m => m.Title)</dd>
```

Movies DB Home About Contact

## Details

Movie

**Title**

Grosse Pointe Blank

**Release Date**

11-Apr-1997

**Genre**

Comedy

**Price**

£4.99

[Edit Record](#) | [Back to List](#)

© 2018 - Movies DB Ltd.

Movies DB Home About Contact

## Details

Movie

<b>Title</b>	Grosse Pointe Blank
<b>Release Date</b>	11-Apr-1997
<b>Genre</b>	Comedy
<b>Price</b>	£4.99

[Edit Record](#) | [Back to List](#)

© 2018 - Movies DB Ltd.

# Styling the Delete view

- The styling of the is very similar to the Details view, though there is also a bit of extra styling on the button
- Add a horizontal rule `<hr/>` under the `<h4>Movie</h4>` heading
- Add a class="dl-horizontal" to the `<dl>` tag
- The styling for the button is on the next slide

```
<h3>Are you sure you want to delete this?</h3>
```

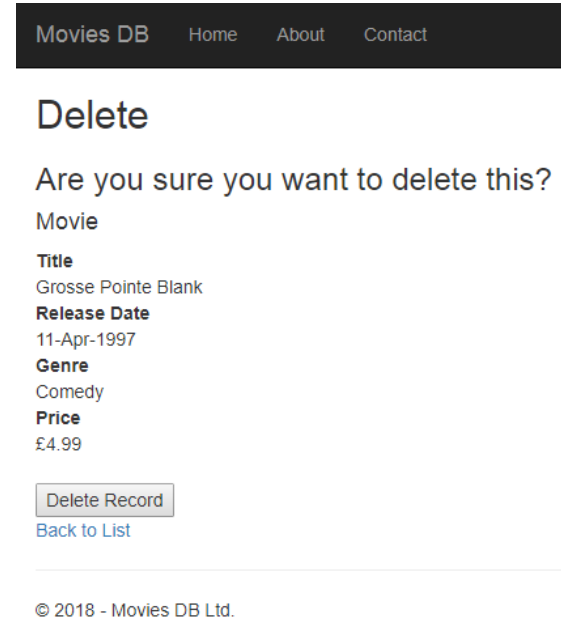
```
<h4>Movie</h4>
```

```
<hr/>
```

```
<dl class="dl-horizontal">
```

```
<dt>@Html.DisplayNameFor(m => m.Title)</dt>
```

```
<dd>@Html.DisplayFor(m => m.Title)</dd>
```



Unstyled version

# Styling the Delete view

- The button is put in a form group (which increases the space around it) by wrapping it in a `<div class="form-group">`
- It is moved over – away from the Back to List link, so it's not clicked accidentally – by putting another div inside the first one: `<div class="col-sm-offset-1 col-sm-11">`
- Then the button is styled by adding `class="btn btn-default"` to the `<input>` tag

```
@using (Html.BeginForm())
{
    @Html.AntiForgeryToken()
    <div class="form-group">
        <div class="col-sm-offset-1 col-sm-11">
            <input type="submit" class="btn btn-default" value="Delete Record" />
        </div>
    </div>
</div>
}
```

Movies DB

Home

About

Contact

## Delete

Are you sure you want to delete this?

Movie

Title	Grosse Pointe Blank
Release Date	11-Apr-1997
Genre	Comedy
Price	£4.99

Delete Record

[Back to List](#)

© 2018 - Movies DB Ltd.

Styled version

# Testing

# MVC Movies is finished and styled

- The functionality of MVC Movies is finished
  - Creating, reading, updating and deleting records from a database
  - Searching records by title and filtering by genre
- It has also been styled
- There are some additional options
  - Tidying up memory by releasing unused resources
  - Adding images using URLs

The screenshot displays the MVC Movies application interface. At the top is a navigation bar with links for 'Movies DB', 'Home', 'About', and 'Contact'. The main content area shows a 'Movies' section with a 'Create new movie' link, a search form with 'Genre' (set to 'All') and 'Title' fields, and a 'Filter' button. Below this is a table of movies:


Title	Release Date	Genre	Price	ImageUrl
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
		Science Fiction	£9.99	<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
				<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
				<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>

Overlaid on the main view are four other views: 'Create Movie', 'Edit Movie', 'Details Movie', and 'Delete'. The 'Create' view has fields for Title, Release Date, Genre, Price, and ImageUrl, with a 'Create Movie' button. The 'Edit' view has similar fields and a 'Save' button. The 'Details' view shows the movie's information and has 'Edit Record' and 'Back to List' links. The 'Delete' view asks 'Are you sure you want to delete this?' and has a 'Delete Record' button and 'Back to List' link. All views include a footer: '© 2018 - Movies DB Ltd.'

# Tidying up memory by releasing unused resources

- It is good practice to dispose of any resources in an application as soon as you are finished with them
  - This releases memory and may improve performance
  - You can do this by adding a Dispose method to the HomeController
- This method is called automatically by the application whenever it's needed
  - The using in using(@Html.BeginFor()) tells MVC to release the resources for the form as soon as they are no longer needed

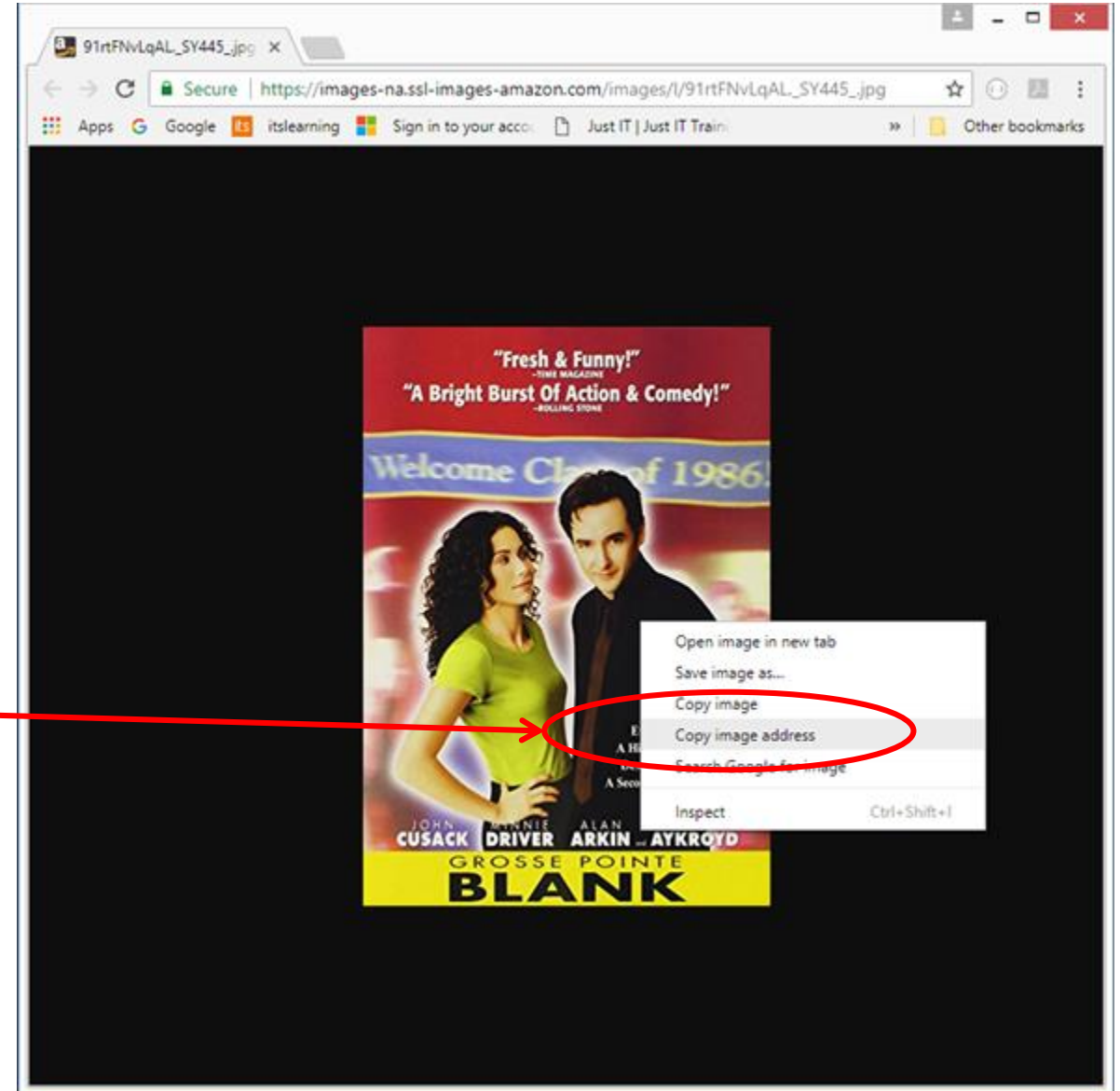
```
protected override void Dispose(bool disposing)
{
    if (disposing)
    {
        db.Dispose();
    }
    base.Dispose(disposing);
}
```



# Images

# Getting image URLs from the web

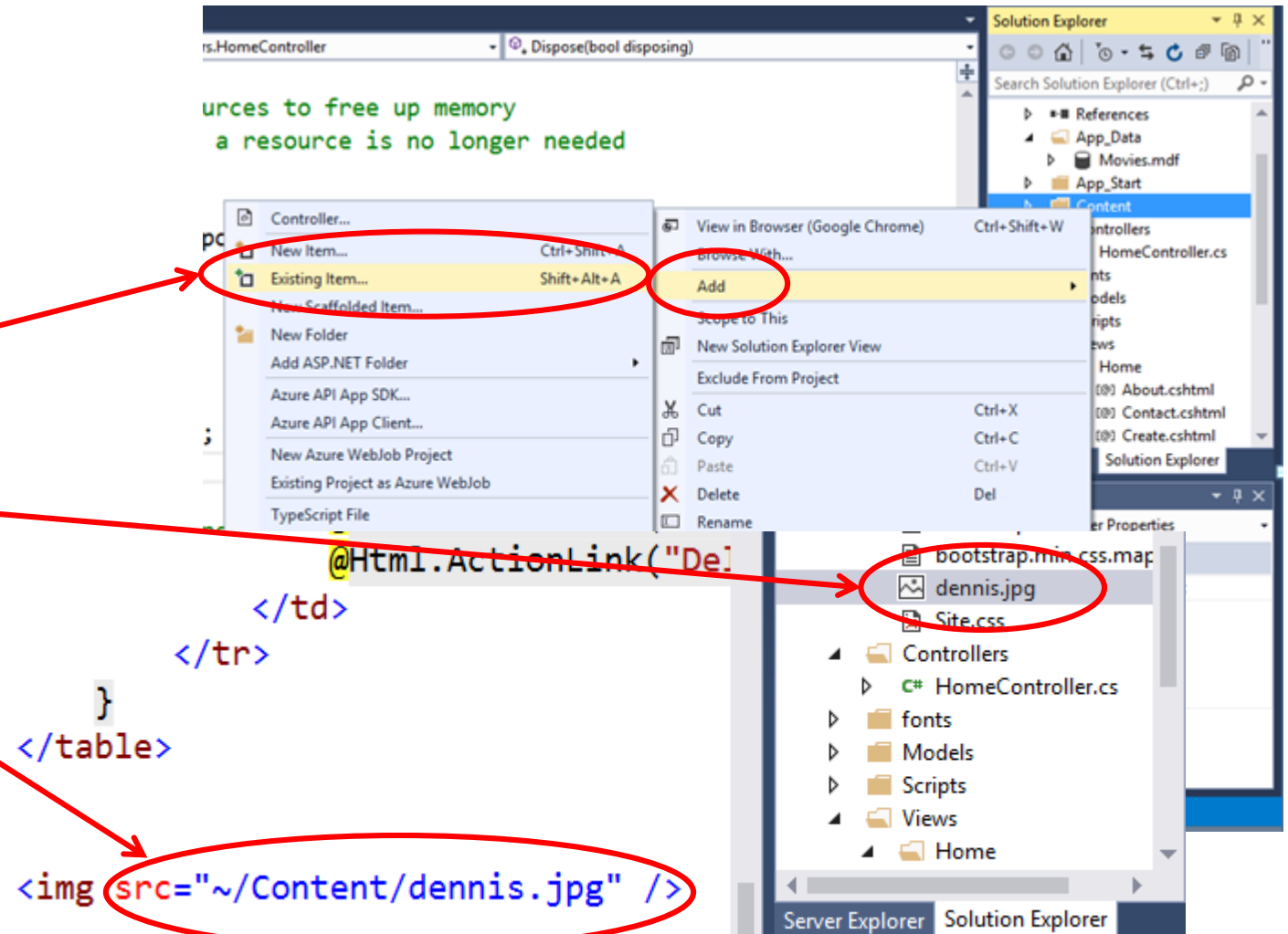
- You can add images to the application easily by using URLs
  - These can be URLs from the web
  - You can also use images that you have added to the project in Visual Studio
- To get an image URL from the web:
  - Do a Google Images search to locate it
  - Open the page with the image
  - Right-click on the image and select open in new tab
  - Then copy the URL using the menu or from the address bar
  - In the application, edit the record, adding the URL
  - For now, the text of the URL will display, but this is easy to render as an image





# Getting URLs for image files

- If you want to link to an image through Visual Studio
  - Add the image file to the Content folder by right-clicking on the folder and selecting Add -> Existing Item
  - Once the image file has been added, drag it into any view
  - An HTML <img> tag will be created, which contains the URL
  - Copy the URL from the tag
  - Edit the relevant record, adding the URL
  - The <img> tag can be deleted
  - For now, the text of the URL will display, but this is easy to render as an image



# Rendering URLs as images

- At the moment, you can see the text of the URLs
- To render them as images, go into the Index view and, in the foreach loop, replace `@Html.DisplayFor(m => item.ImageUrl)` with ``

## Movies

[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl	
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	https://images-na.ssl-images-amazon.com/images/I/91rtFNvLqAL_SY445_.jpg	<a href="#">Edit</a> <a href="#">Record Details</a> <a href="#">Record</a> <a href="#">Delete</a> <a href="#">Record</a>
The Force Awakens	16-Dec-2015	Science Fiction	£9.99	https://ia.media-imdb.com/images/M/MV5BOTAzODEzNDZlMjU5MTU1MTgzNzE@._V1_UX182_CR0,0,182,268_AL_.jpg	<a href="#">Edit</a> <a href="#">Record Details</a> <a href="#">Record</a> <a href="#">Delete</a> <a href="#">Record</a>

```
@foreach (var item in Model)
{
    <tr>
        @*data for the table*@
        <td>@Html.DisplayFor(m => item.Title)</td>
        <td>@Html.DisplayFor(m => item.ReleaseDate)</td>
        <td>@Html.DisplayFor(m => item.Genre)</td>
        <td>@Html.DisplayFor(m => item.Price)</td>
        <td></td>
    </tr>
}
```

# Rendering URLs as images


- The URLs should now display as images
- Non-existent URLs will display as broken links
- If a URL is null, the application will now crash, as the `Url.Content` helper won't accept null values
- To prevent this, add the code below to the POST Create and Edit methods, to save a default value for the `ImageUrl` (of course, please make your own choice of URL)

```
[HttpPost]
[ValidateAntiForgeryToken]
public ActionResult Create([Bind(Include = "Id,Title,ReleaseDate,Genre,Price,ImageUrl")]Movie movie)
{
    if (movie.ImageUrl == "")
    {
        movie.ImageUrl = "http://globalapk.com/uploads/posts/2014-07/1404728819_unnamed.png";
    }
    if (ModelState.IsValid)
    {
        db.Movies.Add(movie),
    }
}
```

## Movies

[Create new movie](#)


Genre:  Title:

Title	Release Date	Genre	Price	ImageUrl
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	
The Force Awakens	16-Dec-2015	Science Fiction	£9.99	

# Rendering URLs as images

- Now, the application should add a default image if a record is created (or edited) with a null value for ImageUrl
- One last thing – the ImageURL heading on the Index page is redundant, so could be removed from the Index view by deleting `<th>@Html.DisplayNameFor(model => model.ImageUrl)</th>`
- This is now a working application with useful functionality

Movie with no Image      01-Dec-2017      None      £1.00






Movies DB   Home   About   Contact

## Movies






[Create new movie](#)

Genre:  Title:

Title	Release Date	Genre	Price	
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99	 <a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Force Awakens	16-Dec-2015	Science Fiction	£9.99	 <a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99	 <a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>

# Next steps

- This application is good preparation for your second project, which should be a single-table database with CRUD functionality
- There are many possible improvements, e.g.
  - Improve the appearance – definitely get rid of the black menu bar at the top and do some other styling
  - Like/Dislike buttons are possible – an AJAX call is ideal to reduce flicker
  - Embedded video on the Details page would be nice – you can use an iframe for this
  - Feel free to use your imagination – choose a topic you're interested in and have fun!

Movies DB   Home   About   Contact					
Movies					
<a href="#">Create new movie</a>					
Genre: <span>All</span> Title: <input type="text"/> <input type="button" value="Filter"/>					
Title	Release Date	Genre	Price		
Grosse Pointe Blank	11-Apr-1997	Comedy	£4.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Force Awakens	16-Dec-2015	Science Fiction	£9.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Hidden Figures	25-Dec-2016	Biographical	£12.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
The Imitation Game	28-Nov-2014	Biographical	£7.99		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>
Movie with no Image	01-Dec-2017	None	£1.00		<a href="#">Edit Record</a> <a href="#">Details of Record</a> <a href="#">Delete Record</a>