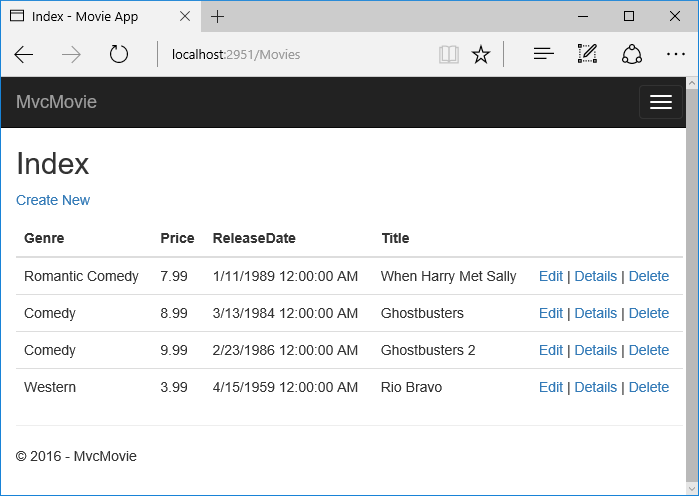
MVC Step 6 (https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views[)](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model)

Controller methods and views

We have a good start to the movie app, but the presentation is not ideal. We don't want to see the time (12:00:00 AM in the image below) and ReleaseDate should be two words.+

+

Open the Models/Movie.cs file and add the highlighted lines shown below:+

Copy

C#

using System;

using System.ComponentModel.DataAnnotations;

namespace MvcMovie.Models

{

public class Movie

{

public int ID { get; set; }

public string Title { get; set; }

[Display(Name = "Release Date")]

[DataType(DataType.Date)]

public DateTime ReleaseDate { get; set; }

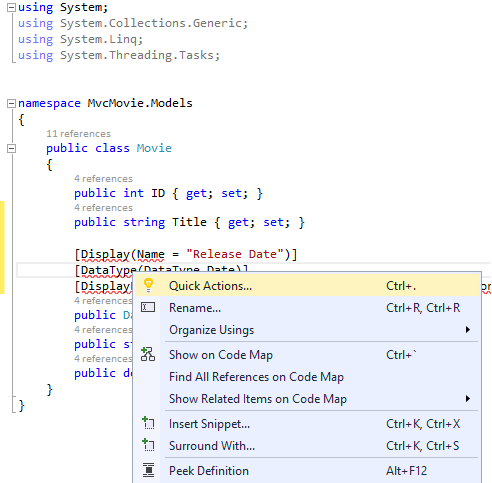
public string Genre { get; set; }

public decimal Price { get; set; }

}

}

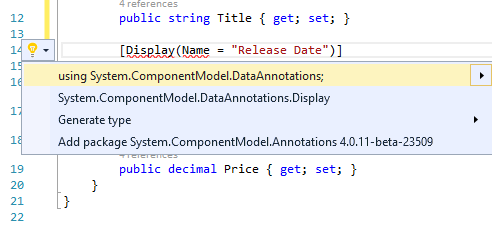
Right click on a red squiggly line > Quick Actions.



Note

In Visual Studio 2017 RC 2, the menu command is > Quick Actions and Refactorings.

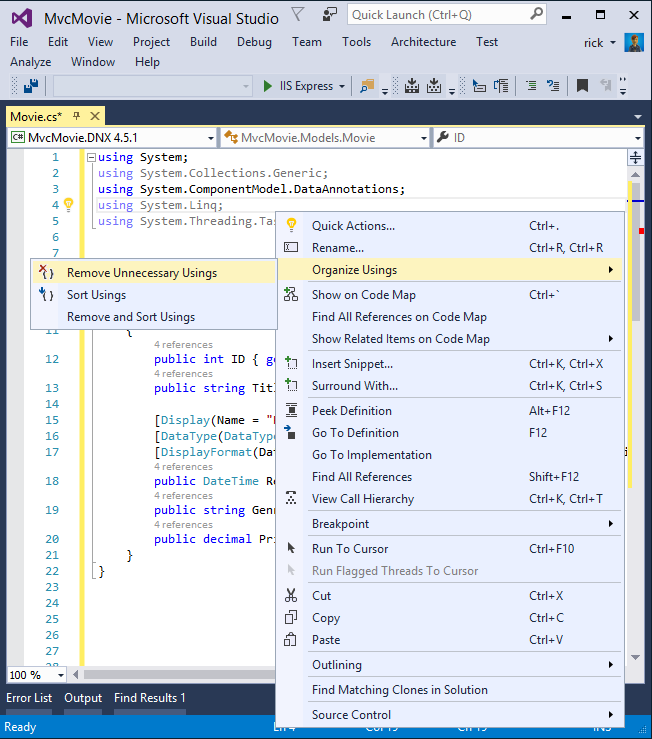
Tap using System.ComponentModel.DataAnnotations;



Visual studio adds using System.ComponentModel.DataAnnotations;.

+

Let's remove the using statements that are not needed. They show up by default in a light grey font. Right click anywhere in the Movie.cs file > Organize Usings > Remove Unnecessary Usings.+

+

Visual Studio 2017 RC 2 note: This step isn't necessary. Unneeded using statements were automatically removed.+

The updated code:+

Copy

C#

using System;

using System.ComponentModel.DataAnnotations;

namespace MvcMovie.Models

{

public class Movie

{

public int ID { get; set; }

public string Title { get; set; }

[Display(Name = "Release Date")]

[DataType(DataType.Date)]

public DateTime ReleaseDate { get; set; }

public string Genre { get; set; }

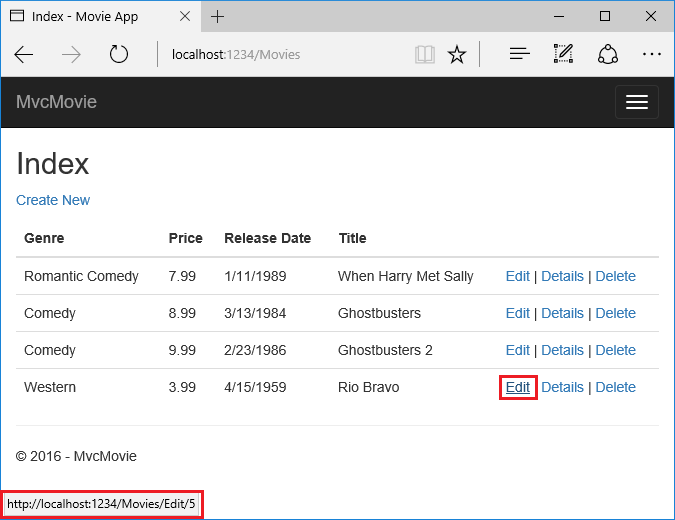
public decimal Price { get; set; }

}

}

We'll cover [DataAnnotations](http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.aspx) in the next tutorial. The [Display](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayattribute.aspx) attribute specifies what to display for the name of a field (in this case "Release Date" instead of "ReleaseDate"). The [DataType](https://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.datatypeattribute.aspx) attribute specifies the type of the data, in this case it's a date, so the time information stored in the field is not displayed.+

Browse to the Movies controller and hold the mouse pointer over an Edit link to see the target URL.+

+

The Edit, Details, and Delete links are generated by the MVC Core Anchor Tag Helper in the Views/Movies/Index.cshtml file.+

Copy

HTML

<a asp-action="Edit" asp-route-id="@item.ID">Edit</a> |

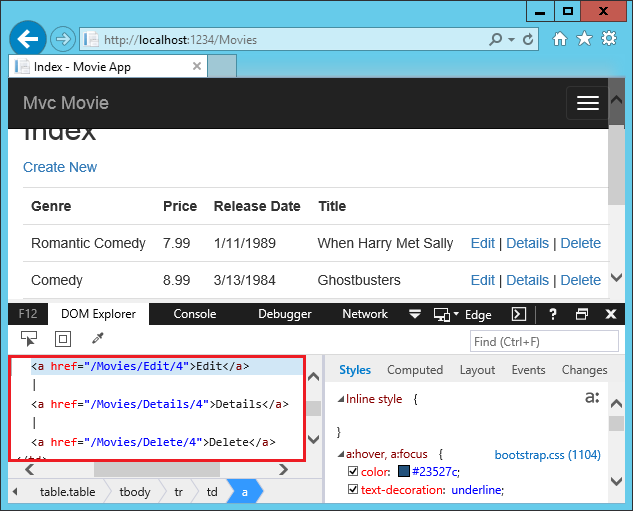
<a asp-action="Details" asp-route-id="@item.ID">Details</a> |

<a asp-action="Delete" asp-route-id="@item.ID">Delete</a>

</td>

</tr>

[Tag Helpers](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/intro) enable server-side code to participate in creating and rendering HTML elements in Razor files. In the code above, the AnchorTagHelper dynamically generates the HTML href attribute value from the controller action method and route id. You use View Source from your favorite browser or use the F12 tools to examine the generated markup. The F12 tools are shown below.+

+

Recall the format for routing set in the Startup.cs file.+

Copy

C#

app.UseMvc(routes =>

{

routes.MapRoute(

name: "default",

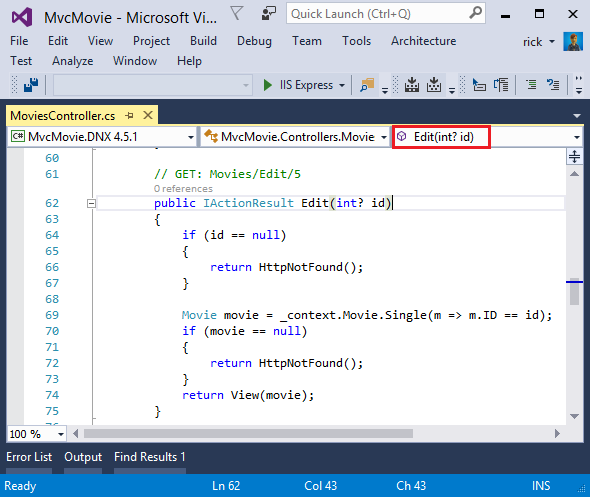
template: "{controller=Home}/{action=Index}/{id?}");

});

ASP.NET Core translates http://localhost:1234/Movies/Edit/4 into a request to the Edit action method of the Movies controller with the parameter Id of 4. (Controller methods are also known as action methods.)+

[Tag Helpers](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/tag-helpers/index) are one of the most popular new features in ASP.NET Core. See [Additional resources](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/controller-methods-views#additional-resources) for more information.+

Open the Movies controller and examine the two Edit action methods:+

1

Copy

C#

// GET: Movies/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var movie = await \_context.Movie.SingleOrDefaultAsync(m => m.ID == id);

if (movie == null)

{

return NotFound();

}

return View(movie);

}

This method will be updated after we add a rating (add it to the bind).+

Copy

C#

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("ID,Genre,Price,ReleaseDate,Title")] Movie movie)

{

if (id != movie.ID)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(movie);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MovieExists(movie.ID))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

return View(movie);

}

The [Bind] attribute is one way to protect against [over-posting](http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application#overpost). You should only include properties in the [Bind] attribute that you want to change. See [Protect your controller from over-posting](http://www.asp.net/mvc/overview/getting-started/getting-started-with-ef-using-mvc/implementing-basic-crud-functionality-with-the-entity-framework-in-asp-net-mvc-application#overpost) for more information. [ViewModels](http://rachelappel.com/use-viewmodels-to-manage-data-amp-organize-code-in-asp-net-mvc-applications/) provide an alternative approach to prevent over-posting.+

Notice the second Edit action method is preceded by the [HttpPost] attribute.1

Copy

C#

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("ID,Genre,Price,ReleaseDate,Title")] Movie movie)

{

if (id != movie.ID)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(movie);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MovieExists(movie.ID))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

return View(movie);

}

The HttpPostAttribute attribute specifies that this Edit method can be invoked only for POST requests. You could apply the [HttpGet] attribute to the first edit method, but that's not necessary because [HttpGet] is the default.+

The ValidateAntiForgeryTokenAttribute attribute is used to prevent forgery of a request and is paired up with an anti-forgery token generated in the edit view file (Views/Movies/Edit.cshtml). The edit view file generates the anti-forgery token with the [Form Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms).+

Copy

HTML

<form asp-action="Edit">

The [Form Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms) generates a hidden anti-forgery token that must match the [ValidateAntiForgeryToken] generated anti-forgery token in the Edit method of the Movies controller. For more information, see [🔧 Anti-Request Forgery](https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery).+

The HttpGet Edit method takes the movie ID parameter, looks up the movie using the Entity Framework SingleOrDefaultAsync method, and returns the selected movie to the Edit view. If a movie cannot be found, NotFound (HTTP 404) is returned.+

Copy

C#

// GET: Movies/Edit/5

public async Task<IActionResult> Edit(int? id)

{

if (id == null)

{

return NotFound();

}

var movie = await \_context.Movie.SingleOrDefaultAsync(m => m.ID == id);

if (movie == null)

{

return NotFound();

}

return View(movie);

}

When the scaffolding system created the Edit view, it examined the Movie class and created code to render <label> and <input> elements for each property of the class. The following example shows the Edit view that was generated by the Visual Studio scaffolding system:2

Copy

HTML

@model MvcMovie.Models.Movie

@{

ViewData["Title"] = "Edit";

}

<h2>Edit</h2>

<form asp-action="Edit">

<div class="form-horizontal">

<h4>Movie</h4>

<hr />

<div asp-validation-summary="ModelOnly" class="text-danger"></div>

<input type="hidden" asp-for="ID" />

<div class="form-group">

<label asp-for="Genre" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="Genre" class="form-control" />

<span asp-validation-for="Genre" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="Price" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="Price" class="form-control" />

<span asp-validation-for="Price" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="ReleaseDate" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="ReleaseDate" class="form-control" />

<span asp-validation-for="ReleaseDate" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<label asp-for="Title" class="col-md-2 control-label"></label>

<div class="col-md-10">

<input asp-for="Title" class="form-control" />

<span asp-validation-for="Title" class="text-danger"></span>

</div>

</div>

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

</form>

<div>

<a asp-action="Index">Back to List</a>

</div>

@section Scripts {

@{await Html.RenderPartialAsync("\_ValidationScriptsPartial");}

}

Notice how the view template has a @model MvcMovie.Models.Movie statement at the top of the file — this specifies that the view expects the model for the view template to be of type Movie.+

The scaffolded code uses several Tag Helper methods to streamline the HTML markup. The - [Label Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms) displays the name of the field ("Title", "ReleaseDate", "Genre", or "Price"). The [Input Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms) renders an HTML <input> element. The [Validation Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms) displays any validation messages associated with that property.+

Run the application and navigate to the /Movies URL. Click an Edit link. In the browser, view the source for the page. The generated HTML for the <form> element is shown below.+

Copy

HTML

<form action="/Movies/Edit/7" method="post">

<div class="form-horizontal">

<h4>Movie</h4>

<hr />

<div class="text-danger" />

<input type="hidden" data-val="true" data-val-required="The ID field is required." id="ID" name="ID" value="7" />

<div class="form-group">

<label class="control-label col-md-2" for="Genre" />

<div class="col-md-10">

<input class="form-control" type="text" id="Genre" name="Genre" value="Western" />

<span class="text-danger field-validation-valid" data-valmsg-for="Genre" data-valmsg-replace="true"></span>

</div>

</div>

<div class="form-group">

<label class="control-label col-md-2" for="Price" />

<div class="col-md-10">

<input class="form-control" type="text" data-val="true" data-val-number="The field Price must be a number." data-val-required="The Price field is required." id="Price" name="Price" value="3.99" />

<span class="text-danger field-validation-valid" data-valmsg-for="Price" data-valmsg-replace="true"></span>

</div>

</div>

<!-- Markup removed for brevity -->

<div class="form-group">

<div class="col-md-offset-2 col-md-10">

<input type="submit" value="Save" class="btn btn-default" />

</div>

</div>

</div>

<input name="\_\_RequestVerificationToken" type="hidden" value="CfDJ8Inyxgp63fRFqUePGvuI5jGZsloJu1L7X9le1gy7NCIlSduCRx9jDQClrV9pOTTmqUyXnJBXhmrjcUVDJyDUMm7-MF\_9rK8aAZdRdlOri7FmKVkRe\_2v5LIHGKFcTjPrWPYnc9AdSbomkiOSaTEg7RU" />

</form>

The <input> elements are in an HTML <form> element whose action attribute is set to post to the /Movies/Edit/id URL. The form data will be posted to the server when the Save button is clicked. The last line before the closing </form> element shows the hidden [XSRF](https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery) token generated by the [Form Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms).+

Processing the POST Request

The following listing shows the [HttpPost] version of the Edit action method.+

Copy

C#

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Edit(int id, [Bind("ID,Genre,Price,ReleaseDate,Title")] Movie movie)

{

if (id != movie.ID)

{

return NotFound();

}

if (ModelState.IsValid)

{

try

{

\_context.Update(movie);

await \_context.SaveChangesAsync();

}

catch (DbUpdateConcurrencyException)

{

if (!MovieExists(movie.ID))

{

return NotFound();

}

else

{

throw;

}

}

return RedirectToAction("Index");

}

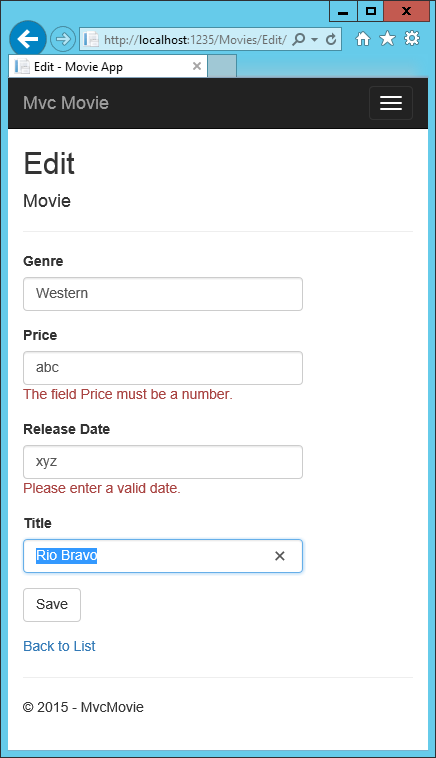
return View(movie);

}

The [ValidateAntiForgeryToken] attribute validates the hidden [XSRF](https://docs.microsoft.com/en-us/aspnet/core/security/anti-request-forgery) token generated by the anti-forgery token generator in the [Form Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms)+

The [model binding](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/model-binding) system takes the posted form values and creates a Movie object that's passed as the movie parameter. The ModelState.IsValid method verifies that the data submitted in the form can be used to modify (edit or update) a Movie object. If the data is valid it's saved. The updated (edited) movie data is saved to the database by calling the SaveChangesAsync method of database context. After saving the data, the code redirects the user to the Index action method of the MoviesController class, which displays the movie collection, including the changes just made.+

Before the form is posted to the server, client side validation checks any validation rules on the fields. If there are any validation errors, an error message is displayed and the form is not posted. If JavaScript is disabled, you won't have client side validation but the server will detect the posted values that are not valid, and the form values will be redisplayed with error messages. Later in the tutorial we examine [Model Validation](https://docs.microsoft.com/en-us/aspnet/core/mvc/models/validation) validation in more detail. The [Validation Tag Helper](https://docs.microsoft.com/en-us/aspnet/core/mvc/views/working-with-forms) in the Views/Movies/Edit.cshtml view template takes care of displaying appropriate error messages.+

+

All the HttpGet methods in the movie controller follow a similar pattern. They get a movie object (or list of objects, in the case of Index), and pass the object (model) to the view. The Create method passes an empty movie object to the Create view. All the methods that create, edit, delete, or otherwise modify data do so in the [HttpPost] overload of the method. Modifying data in an HTTP GET method is a security risk. Modifying data in a HTTP GET method also violates HTTP best practices and the architectural [REST](http://rest.elkstein.org/) pattern, which specifies that GET requests should not change the state of your application. In other words, performing a GET operation should be a safe operation that has no side effects and doesn't modify your persisted data.