

[ASP.NET Core 1.1](#) ▾

Version

[2.2 Preview 2](#)


[2.1](#)

[2.0](#)

[1.1](#)

[1.0](#)

# Controller methods and views in ASP.NET Core

📅 04/07/2017 ⌚ 10 minutes to read Contributors 

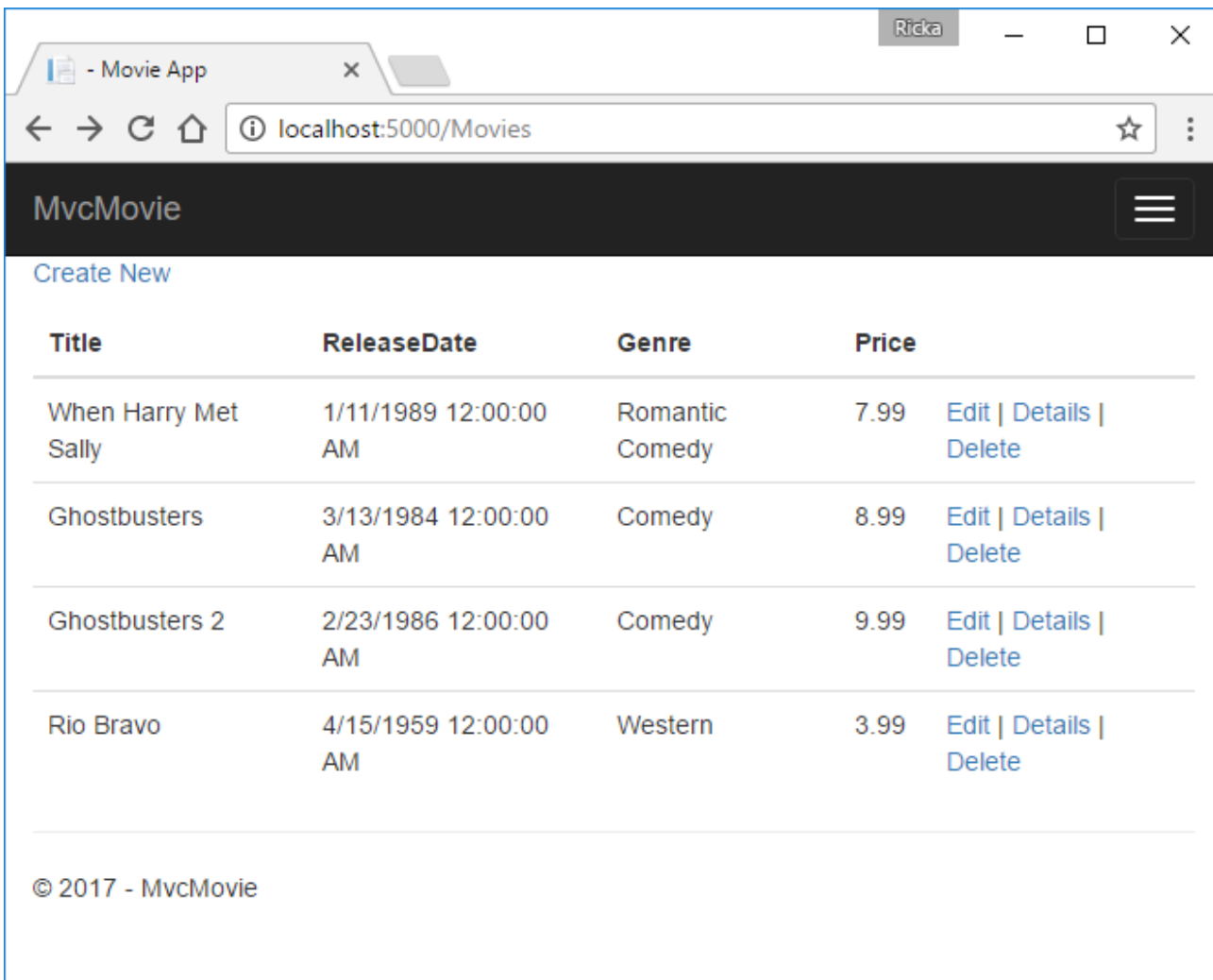
## In this article

[Processing the POST Request](#)

[Additional resources](#)

By [Rick Anderson](#)

We have a good start to the movie app, but the presentation isn't 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:

```
C# Copy

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; }
    }
}
```

Build and run the app.

We cover [DataAnnotations](#) in the next tutorial. The [Display](#) attribute specifies what to display for the name of a field (in this case "Release Date" instead of "ReleaseDate"). The [DataType](#) attribute specifies the type of the data (Date), so the time information stored in the field isn't displayed.

The `[Column(TypeName = "decimal(18, 2)"]` data annotation is required so Entity Framework Core can correctly map `Price` to currency in the database. For more information, see [Data Types](#).

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

Index - Movie App

localhost:1234/Movies

MvcMovie

## Index

[Create New](#)

Genre	Price	Release Date	Title	
Romantic Comedy	7.99	1/11/1989	When Harry Met Sally	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Comedy	8.99	3/13/1984	Ghostbusters	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Comedy	9.99	2/23/1986	Ghostbusters 2	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>
Western	3.99	4/15/1959	Rio Bravo	<a href="#">Edit</a>   <a href="#">Details</a>   <a href="#">Delete</a>


© 2016 - MvcMovie

<http://localhost:1234/Movies/Edit/5>

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

HTML	Copy
<pre>&lt;a asp-action="Edit" asp-route-id="@item.ID"&gt;Edit&lt;/a&gt;   &lt;a asp-action="Details" asp-route-id="@item.ID"&gt;Details&lt;/a&gt;   &lt;a asp-action="Delete" asp-route-id="@item.ID"&gt;Delete&lt;/a&gt; &lt;/td&gt; &lt;/tr&gt;</pre>	

[Tag Helpers](#) 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 developer tools to examine the generated markup. A portion of the generated HTML is shown below:

HTML	
<pre>&lt;td&gt;   &lt;a href="/Movies/Edit/4"&gt; Edit &lt;/a&gt;     &lt;a href="/Movies/Details/4"&gt; Details &lt;/a&gt;     &lt;a href="/Movies/Delete/4"&gt; Delete &lt;/a&gt; &lt;/td&gt;</pre>	


Recall the format for [routing](#) set in the *Startup.cs* file:

C#	
<pre>app.UseMvc(routes =&gt; {     routes.MapRoute(         name: "default",         template: "{controller=Home}/{action=Index}/{id?}"); });</pre>	

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](#) are one of the most popular new features in ASP.NET Core. See [Additional resources](#) for more information.

Open the `Movies` controller and examine the two `Edit` action methods. The following code shows the `HTTP GET Edit` method, which fetches the movie and populates the edit form generated by the *Edit.cshtml* Razor file.

C#	
<pre>// GET: Movies/Edit/5 public async Task&lt;IActionResult&gt; Edit(int? id) {     if (id == null)     {         return NotFound();     } }</pre>	

```

var movie = await _context.Movie.SingleOrDefaultAsync(m => m.ID == id);
if (movie == null)
{
    return NotFound();
}
return View(movie);
}

```

The following code shows the `HTTP POST Edit` method, which processes the posted movie values:

C#

 Copy

```

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind
// to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] 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](#). You should only include properties in the `[Bind]` attribute that you want to change. See [Protect your controller from over-posting](#) for more information. [ViewModels](#) provide an alternative approach to prevent over-posting.

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

C#

 Copy

```
// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind
// to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] 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 `HttpPost` 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 `ValidateAntiForgeryToken` 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](#).


HTML	
<pre>&lt;form asp-action="Edit"&gt;</pre>	

The [Form Tag Helper](#) 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](#).

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.

C#	
<pre>// GET: Movies/Edit/5 public async Task&lt;IActionResult&gt; Edit(int? id) {     if (id == null)     {         return NotFound();     }      var movie = await _context.Movie.SingleOrDefaultAsync(m =&gt; m.ID == id);     if (movie == null)     {         return NotFound();     }     return View(movie); }</pre>	

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:

HTML	
<pre>@model MvcMovie.Models.Movie  @{     ViewData["Title"] = "Edit"; }</pre>	

```
<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="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">
      <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="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">
      <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.

`@model MvcMovie.Models.Movie` 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](#) displays the name of the field ("Title", "ReleaseDate", "Genre", or "Price"). The [Input Tag Helper](#) renders an HTML `<input>` element. The [Validation Tag Helper](#) 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.

HTML	 Copy
<pre>&lt;form action="/Movies/Edit/7" method="post"&gt;   &lt;div class="form-horizontal"&gt;     &lt;h4&gt;Movie&lt;/h4&gt;     &lt;hr /&gt;     &lt;div class="text-danger" /&gt;     &lt;input type="hidden" data-val="true" data-val-required="The ID field is required." id="ID" name="ID" value="7" /&gt;     &lt;div class="form-group"&gt;       &lt;label class="control-label col-md-2" for="Genre" /&gt;       &lt;div class="col-md-10"&gt;         &lt;input class="form-control" type="text" id="Genre" name="Genre" value="Western" /&gt;         &lt;span class="text-danger field-validation-valid" data-valmsg-for="Genre" data- valmsg-replace="true"&gt;&lt;/span&gt;       &lt;/div&gt;     &lt;/div&gt;     &lt;div class="form-group"&gt;       &lt;label class="control-label col-md-2" for="Price" /&gt;       &lt;div class="col-md-10"&gt;         &lt;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" /&gt;         &lt;span class="text-danger field-validation-valid" data-valmsg-for="Price" data- valmsg-replace="true"&gt;&lt;/span&gt;       &lt;/div&gt;     &lt;/div&gt;     &lt;!-- Markup removed for brevity --&gt;     &lt;div class="form-group"&gt;       &lt;div class="col-md-offset-2 col-md-10"&gt;         &lt;input type="submit" value="Save" class="btn btn-default" /&gt;       &lt;/div&gt;     &lt;/div&gt;   &lt;/div&gt;   &lt;input name="__RequestVerificationToken" type="hidden" value="CfDJ8Inyxgp63fRFqUePGvui5jGZsloJu1L7X9le1gy7NCI1SduCRx9jDQC1rV9pOTTmqUyXnJBXhmrjcUVDJyD</pre>	

```
UMm7-MF_9rK8aAZdRd1Ori7FmKVkRe_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](#) token generated by the [Form Tag Helper](#).

## Processing the POST Request

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

```
C#  Copy

// POST: Movies/Edit/5
// To protect from overposting attacks, please enable the specific properties you want to bind
// to, for
// more details see http://go.microsoft.com/fwlink/?LinkId=317598.
[HttpPost]
[ValidateAntiForgeryToken]
public async Task<IActionResult> Edit(int id, [Bind("ID,Title,ReleaseDate,Genre,Price")] 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](#) token generated by the anti-forgery token generator in the [Form Tag Helper](#)

The [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 isn't 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](#) in more detail. The [Validation Tag Helper](#) in the `Views/Movies/Edit.cshtml` view template takes care of displaying appropriate error messages.

http://localhost:1235/Movies/Edit/

Edit - Movie App

Mvc Movie

## Edit

### Movie

---

**Genre**

**Price**

The field Price must be a number.

**Release Date**

Please enter a valid date.

**Title**

Save

[Back to List](#)

---

© 2015 - MvcMovie

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 an `HTTP GET` method also violates HTTP best practices and the architectural [REST](#) pattern, which specifies that GET requests shouldn't 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.

# Additional resources

- [Globalization and localization](#)
- [Introduction to Tag Helpers](#)
- [Author Tag Helpers](#)
- [Anti-Request Forgery](#)
- Protect your controller from [over-posting](#)
- [ViewModels](#)
- [Form Tag Helper](#)
- [Input Tag Helper](#)
- [Label Tag Helper](#)
- [Select Tag Helper](#)
- [Validation Tag Helper](#)

[Previous - Working with SQLite](#)

[Next - Add search](#)