MVC Step 4 [(https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model)](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model)

Adding a model

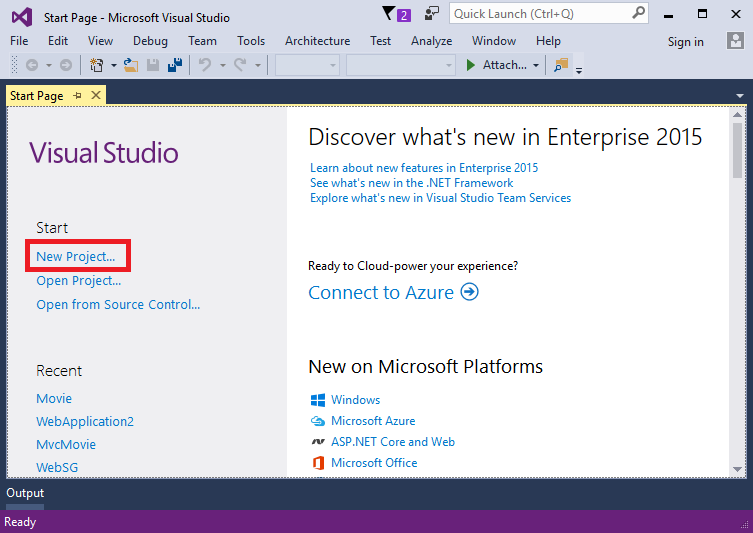
In this section you'll add some classes for managing movies in a database. These classes will be the "**M**odel" part of the **M**VC app.1

You’ll use a .NET Framework data-access technology known as the [Entity Framework Core](http://ef.readthedocs.org/) to define and work with these data model classes. Entity Framework Core (often referred to as **EF** Core) features a development paradigm called Code First. You write the code first, and the database tables are created from this code. Code First allows you to create data model objects by writing simple classes. (These are also known as POCO classes, from "plain-old CLR objects.") The database is created from your classes. If you are required to create the database first, you can still follow this tutorial to learn about MVC and EF app development.+

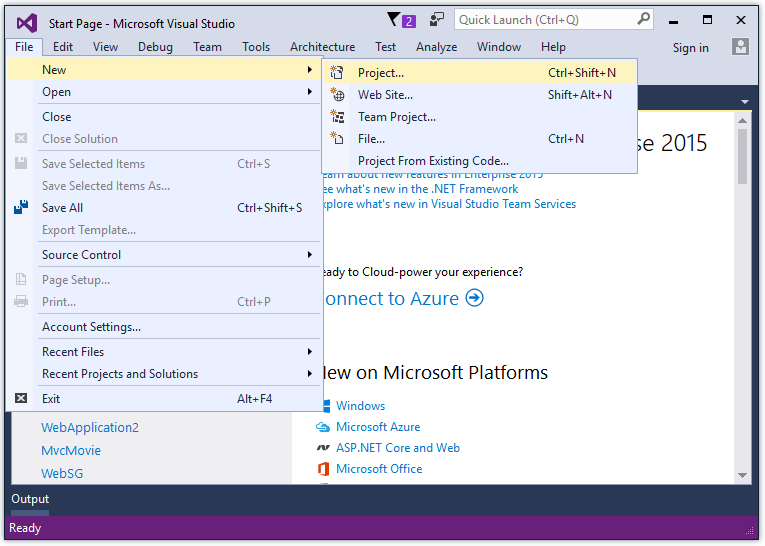
## Create a new project with individual user accounts

In the current version of the ASP.NET Core MVC tools for Visual Studio, scaffolding a model is only supported when you create a new project with individual user accounts. We hope to have this fixed in the next tooling update. Until that's fixed, you'll need to create a new project with the same name. Because the project has the same name, you'll need to create it in another directory.1

From the Visual Studio **Start** page, tap **New Project**.+

+

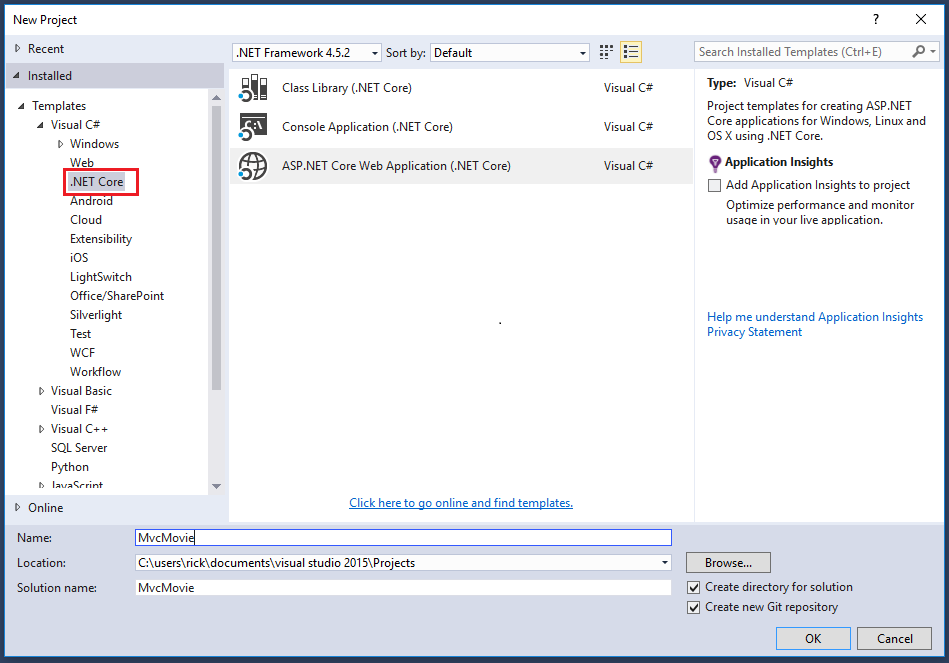
Alternatively, you can use the menus to create a new project. Tap **File > New > Project**.+

+

Complete the **New Project** dialog:+

* In the left pane, tap **Web**
* In the center pane, tap **ASP.NET Core Web Application (.NET Core)**
* Change the location to a different directory from the previous project you created or you'll get an error
* Name the project "MvcMovie" (It's important to name the project "MvcMovie" so when you copy code, the namespace will match.)
* Tap **OK**

+

+

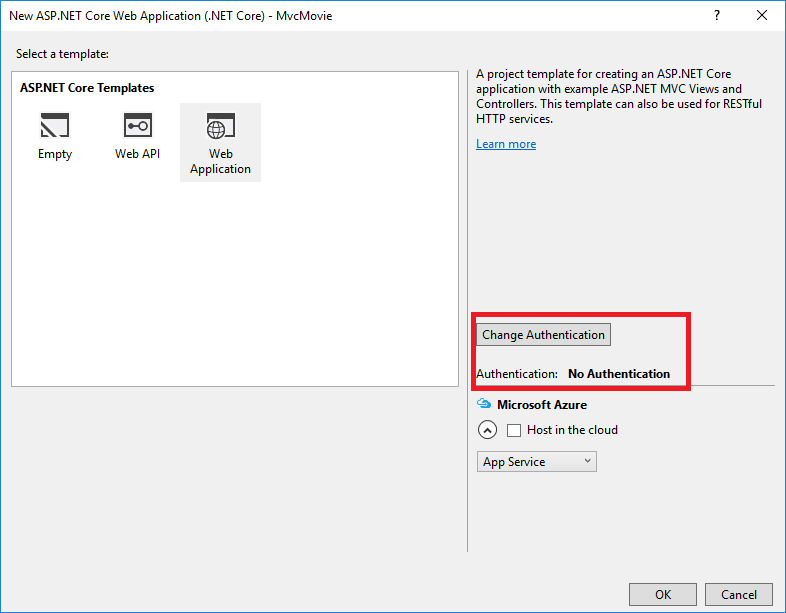
##### Warning

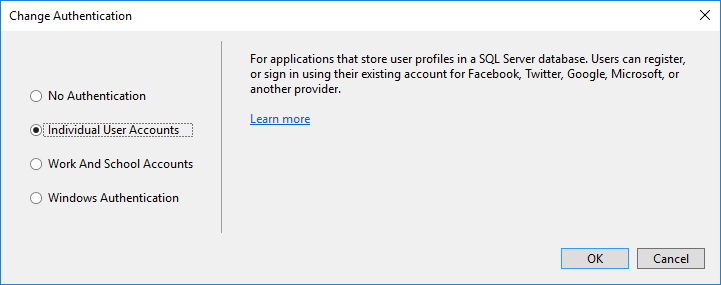
You must have the **Authentication** set to **Individual User Accounts** in this release for the scaffolding engine to work.+

In the **New ASP.NET Core Web Application - MvcMovie** dialog:+

* tap **Web Application**
* tap the **Change Authentication** button and change the authentication to **Individual User Accounts** and tap **OK**

1

+

+

Follow the instructions in [Change the title and menu link in the layout file](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-view#change-title-link-reference-label) so you can tap the **MvcMovie** link to invoke the Movie controller. We'll scaffold the movies controller in this tutorial.1

### Adding data model classes

In Solution Explorer, right click the Models folder > **Add** > **Class**. Name the class **Movie** and add the following properties:+

Copy

C#

using System;

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; }

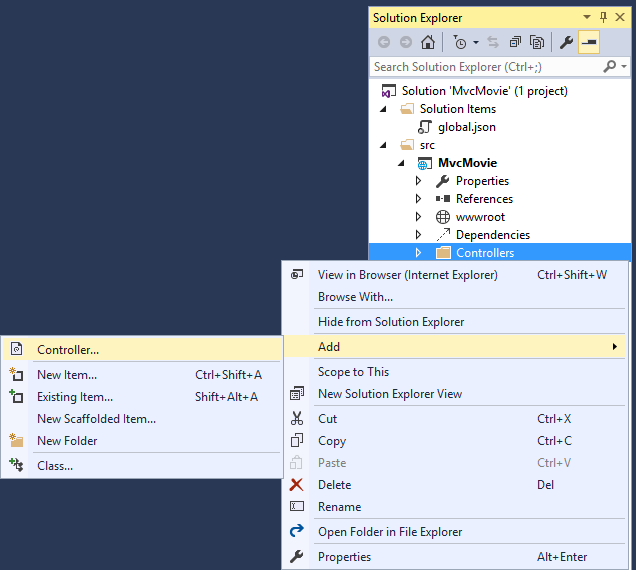
}

}

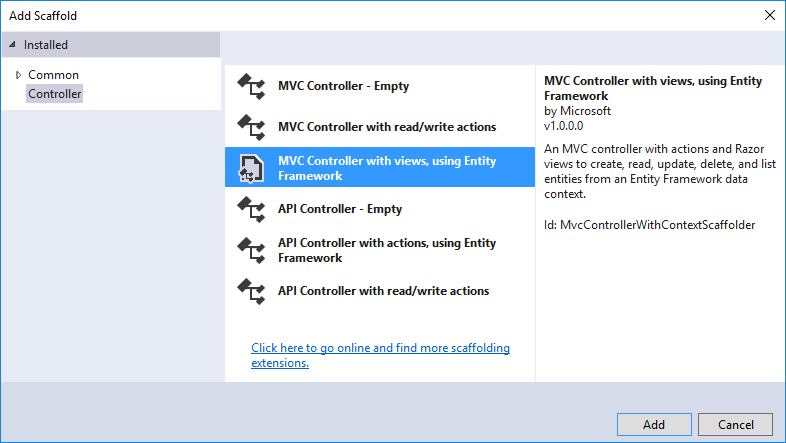
In addition to the properties you'd expect to model a movie, the ID field is required by the DB for the primary key. Build the project. If you don't build the app, you'll get an error in the next section. We've finally added a **M**odel to our **M**VC app.3

### Scaffolding a controller

In **Solution Explorer**, right-click the Controllers folder **> Add > Controller**.+

+

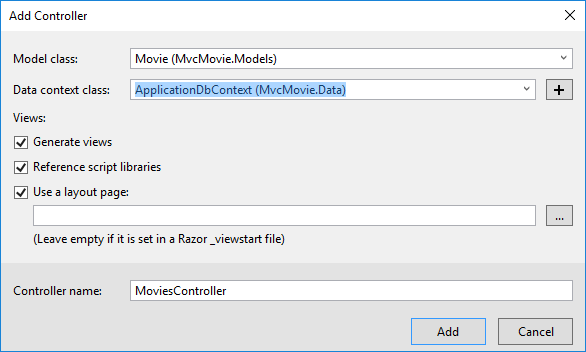
In the **Add Scaffold** dialog, tap **MVC Controller with views, using Entity Framework > Add**.+

+

Complete the **Add Controller** dialog:+

* **Model class:** Movie (MvcMovie.Models)
* **Data context class:** ApplicationDbContext (MvcMovie.Data)
* **Views:** Keep the default of each option checked
* **Controller name:** Keep the default MoviesController
* Tap **Add**

+

+

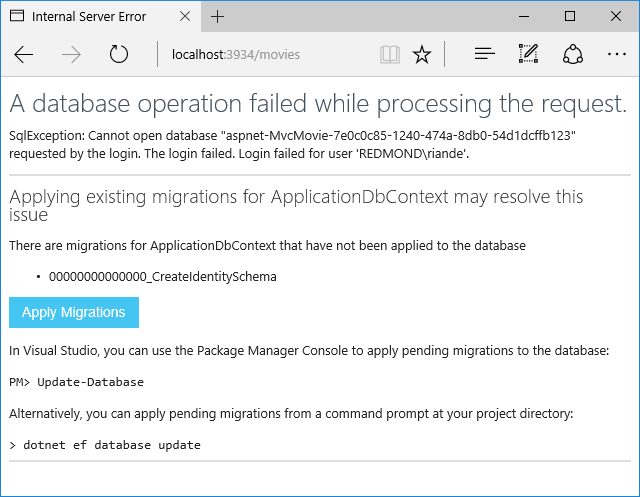
The Visual Studio scaffolding engine creates the following:+

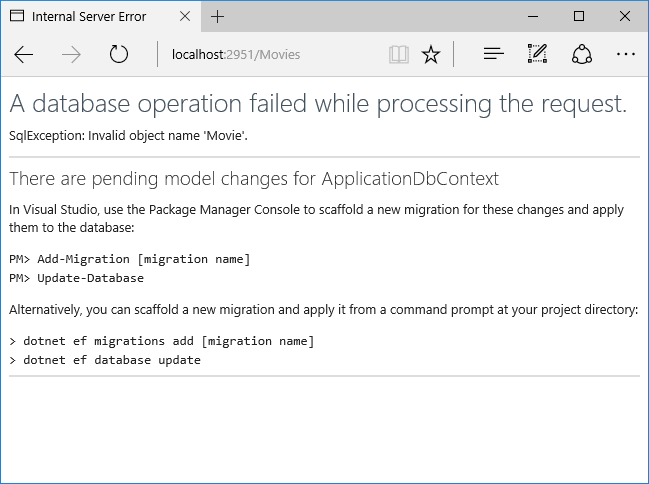
* A movies controller (Controllers/MoviesController.cs)
* Create, Delete, Details, Edit and Index Razor view files (Views/Movies)

+

Visual Studio automatically created the [CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) (create, read, update, and delete) action methods and views for you (the automatic creation of CRUD action methods and views is known as scaffolding). You'll soon have a fully functional web application that lets you create, list, edit, and delete movie entries.+

If you run the app and click on the **Mvc Movie** link, you'll get the following errors:5

1

+

We'll follow those instructions to get the database ready for our Movie app.+

### Update the database

* Open a command prompt and navigate to the project directory (your\_path/MvcMovie/src/MvcMovie). You can find the path by selecting the web.config file in Solution Explorer and viewing the Full Path property in the Properties window.
* Run the following commands in the command prompt:

Copy

console

dotnet ef migrations add Initial

dotnet ef database update

9

## dotnet ef commands

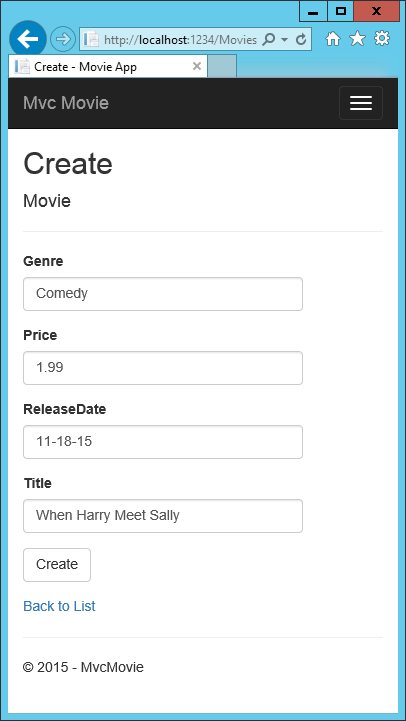
* dotnet (.NET Core) is a cross-platform implementation of .NET. You can read about it [here](http://go.microsoft.com/fwlink/?LinkID=517853)
* dotnet ef migrations add Initial Runs the Entity Framework .NET Core CLI migrations command and creates the initial migration. The parameter "Initial" is arbitrary, but customary for the first (initial) database migration. This operation creates the Data/Migrations/\_Initial.cs file containing the migration commands to add (or drop) the Movie table to the database
* dotnet ef database update Updates the database with the migration we just created

2

### Test the app

Notes: +

* If your browser is unable to connect to the movie app you might need to wait for IIS Express to load the app. It can sometimes take up to 30 seconds to build the app and have it ready to respond to requests.
  + Run the app and tap the **Mvc Movie** link
  + Tap the **Create New** link and create a movie



* You may not be able to enter decimal points or commas in the Price field. To support [jQuery validation](http://jqueryvalidation.org/) for non-English locales that use a comma (",") for a decimal point, and non US-English date formats, you must take steps to globalize your app. See [Additional resources](https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/adding-model#additional-resources) for more information. For now, just enter whole numbers like 10.

+

+

* In some locales you'll need to specify the date format. See the highlighted code 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; }

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

public DateTime ReleaseDate { get; set; }

public string Genre { get; set; }

public decimal Price { get; set; }

}

}

* You may wish to change the ReleaseDate label text from the default property name text. To accomplish this, add the highlighted code 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")]

public DateTime ReleaseDate { get; set; }

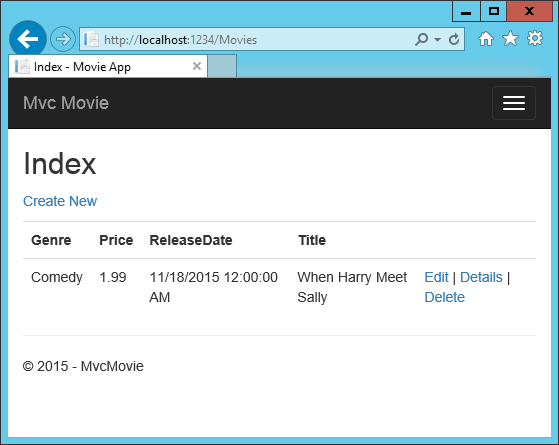
public string Genre { get; set; }

public decimal Price { get; set; }

}

}

Tapping **Create** causes the form to be posted to the server, where the movie information is saved in a database. You are then redirected to the /Movies URL, where you can see the newly created movie in the listing.2

+

Create a couple more movie entries. Try the **Edit**, **Details**, and **Delete** links, which are all functional.1

### Examining the Generated Code

Open the Controllers/MoviesController.cs file and examine the generated Index method. A portion of the movie controller with the Index method is shown below:1

Copy

C#

public class MoviesController : Controller

{

private readonly ApplicationDbContext \_context;

public MoviesController(ApplicationDbContext context)

{

\_context = context;

}

// GET: Movies

public async Task<IActionResult> Index()

{

return View(await \_context.Movie.ToListAsync());

}

}

The constructor uses [Dependency Injection](https://docs.microsoft.com/en-us/aspnet/core/fundamentals/dependency-injection) to inject the database context into the controller. The database context is used in each of the [CRUD](https://en.wikipedia.org/wiki/Create,_read,_update_and_delete) methods in the controller.+

A request to the Movies controller returns all the entries in the Movies table and then passes the data to the Index view.+

+

## Strongly typed models and the @model keyword

Earlier in this tutorial, you saw how a controller can pass data or objects to a view using the ViewData dictionary. The ViewData dictionary is a dynamic object that provides a convenient late-bound way to pass information to a view.+

MVC also provides the ability to pass strongly typed objects to a view. This strongly typed approach enables better compile-time checking of your code and richer [IntelliSense](https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx) in Visual Studio (VS). The scaffolding mechanism in VS used this approach (that is, passing a strongly typed model) with the MoviesController class and views when it created the methods and views.+

Examine the generated Details method in the Controllers/MoviesController.cs file:+

Copy

C#

// GET: Movies/Details/5

public async Task<IActionResult> Details(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);

}

The id parameter is generally passed as route data, for example http://localhost:1234/movies/details/1 sets:+

* The controller to the movies controller (the first URL segment)
* The action to details (the second URL segment)
* The id to 1 (the last URL segment)

+

You could also pass in the id with a query string as follows:+

http://localhost:1234/movies/details?id=1+

If a Movie is found, an instance of the Movie model is passed to the Details view:+

Copy

C#

return View(movie);

Examine the contents of the Views/Movies/Details.cshtml file:+

Copy

html

@model MvcMovie.Models.Movie

@{

ViewData["Title"] = "Details";

}

<h2>Details</h2>

<div>

<h4>Movie</h4>

<hr />

<dl class="dl-horizontal">

<dt>

@Html.DisplayNameFor(model => model.Genre)

</dt>

<dd>

@Html.DisplayFor(model => model.Genre)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Price)

</dt>

<dd>

@Html.DisplayFor(model => model.Price)

</dd>

<dt>

@Html.DisplayNameFor(model => model.ReleaseDate)

</dt>

<dd>

@Html.DisplayFor(model => model.ReleaseDate)

</dd>

<dt>

@Html.DisplayNameFor(model => model.Title)

</dt>

<dd>

@Html.DisplayFor(model => model.Title)

</dd>

</dl>

</div>

<div>

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

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

</div>

By including a @model statement at the top of the view file, you can specify the type of object that the view expects. When you created the movie controller, Visual Studio automatically included the following @model statement at the top of the Details.cshtml file:+

Copy

HTML

@model MvcMovie.Models.Movie

This @model directive allows you to access the movie that the controller passed to the view by using a Model object that's strongly typed. For example, in the Details.cshtml view, the code passes each movie field to the DisplayNameFor and DisplayFor HTML Helpers with the strongly typed Model object. The Create and Edit methods and views also pass a Movie model object.+

Examine the Index.cshtml view and the Index method in the Movies controller. Notice how the code creates a List object when it calls the View method. The code passes this Movies list from the Index action method to the view:+

Copy

C#

// GET: Movies

public async Task<IActionResult> Index()

{

return View(await \_context.Movie.ToListAsync());

}

When you created the movies controller, Visual Studio automatically included the following @model statement at the top of the Index.cshtml file:+

Copy

html

@model IEnumerable<MvcMovie.Models.Movie>

The @model directive allows you to access the list of movies that the controller passed to the view by using a Model object that's strongly typed. For example, in the Index.cshtml view, the code loops through the movies with a foreach statement over the strongly typed Model object:+

Copy

html

@model IEnumerable<MvcMovie.Models.Movie>

@{

ViewData["Title"] = "Index";

}

<h2>Index</h2>

<p>

<a asp-action="Create">Create New</a>

</p>

<table class="table">

<thead>

<tr>

<th>

@Html.DisplayNameFor(model => model.Genre)

</th>

<th>

@Html.DisplayNameFor(model => model.Price)

</th>

<th>

@Html.DisplayNameFor(model => model.ReleaseDate)

</th>

<th>

@Html.DisplayNameFor(model => model.Title)

</th>

<th></th>

</tr>

</thead>

<tbody>

@foreach (var item in Model) {

<tr>

<td>

@Html.DisplayFor(modelItem => item.Genre)

</td>

<td>

@Html.DisplayFor(modelItem => item.Price)

</td>

<td>

@Html.DisplayFor(modelItem => item.ReleaseDate)

</td>

<td>

@Html.DisplayFor(modelItem => item.Title)

</td>

<td>

<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>

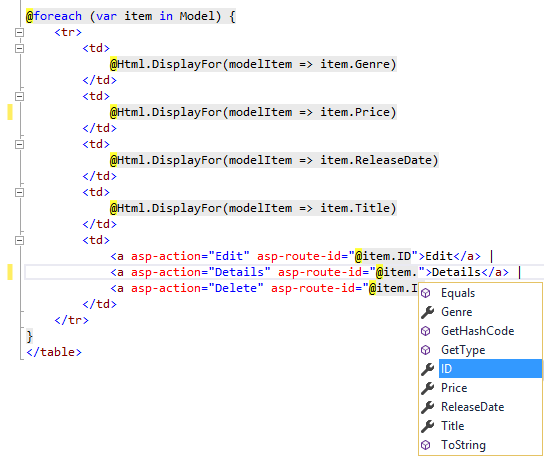
</tr>

}

</tbody>

</table>

Because the Model object is strongly typed (as an IEnumerable<Movie> object), each item in the loop is typed as Movie. Among other benefits, this means that you get compile-time checking of the code and full [IntelliSense](https://msdn.microsoft.com/en-us/library/hcw1s69b.aspx) support in the code editor:+

+

You now have a database and pages to display, edit, update and delete data. In the next tutorial, we'll work with the database.