Parte 4. Adición de un modelo a una aplicación de ASP.NET Core MVC

16/11/2020Tiempo de lectura: 25 minutos

En este artículo

Por Rick Anderson y Jon P Smith.

En esta sección, se agregan clases para administrar películas en una base de datos. Estas clases son el elemento "**M** odel" de la aplicación **M** VC.

Estas clases de modelo se usan con Entity Framework Core (EF Core) para trabajar con una base de datos. EF Core es un marco de trabajo de asignación relacional de objetos (ORM) que simplifica el código de acceso de datos que se debe escribir.

Las clases de modelo creadas se conocen como clases *__POCO__ _ de _*P**lain **O** ld **C** LR **O**bjects. Las clases POCO no tienen ninguna dependencia de EF Core. Solo definen las propiedades de los datos que se almacenan en la base de datos.

En este tutorial, primero se crean las clases de modelo, y EF Core crea la base de datos.

Agregar una clase de modelo de datos

Visual Studio | Visual Studio Code | Visual Studio para Mac

Haga clic con el botón derecho en la carpeta *Models* > **Agregar** > **Clase**. Asigne el nombre *Movie.cs* al archivo.

Actualice el archivo *Models/Movie.cs* con el código siguiente:

C# Copiar

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

namespace MvcMovie.Models
```

```
{
    public class Movie
    {
        public int Id { get; set; }
        public string Title { get; set; }

        [DataType(DataType.Date)]
        public DateTime ReleaseDate { get; set; }
        public string Genre { get; set; }
        public decimal Price { get; set; }
    }
}
```

La clase `Movie` contiene un campo `Id`, que la base de datos requiere para la clave principal.

El atributo DataType en `ReleaseDate` especifica el tipo de los datos (`Date`). Con este atributo:

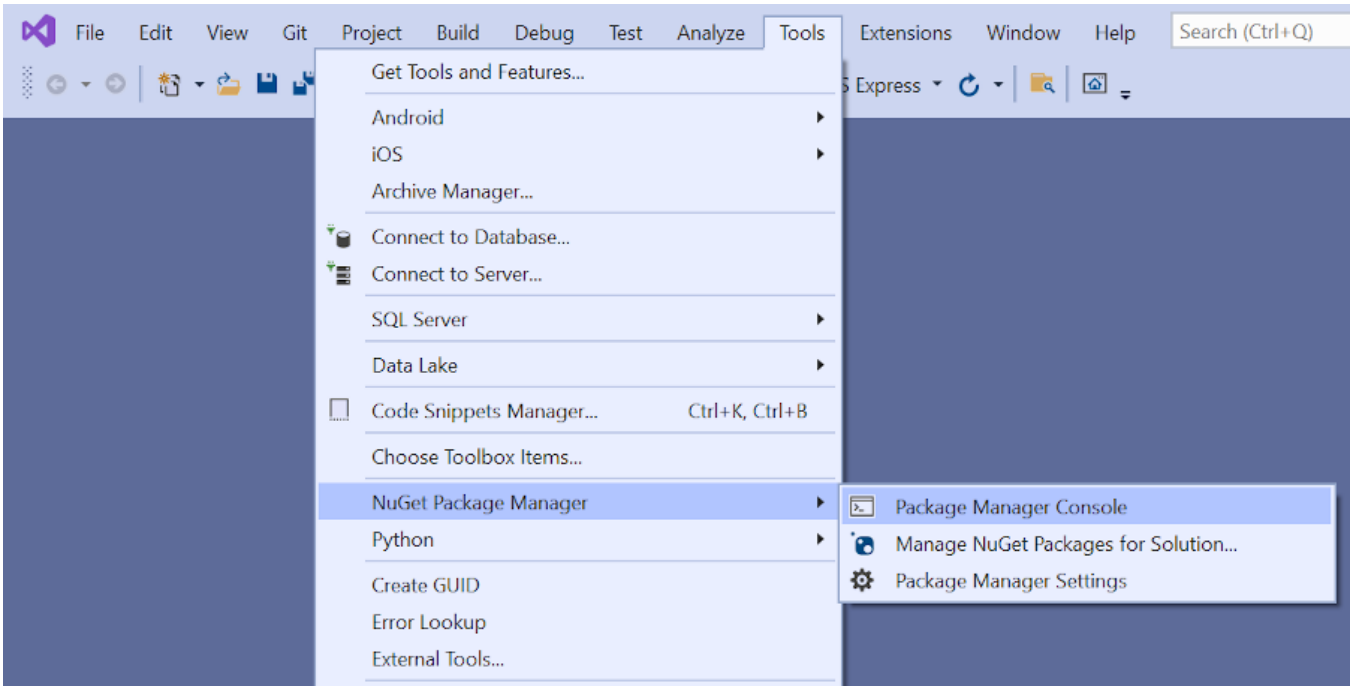
- El usuario no tiene que especificar información horaria en el campo de fecha.
- Solo se muestra la fecha, no información horaria.

Los elementos DataAnnotations se tratan en un tutorial posterior.

Adición de paquetes NuGet

Visual Studio | Visual Studio Code | Visual Studio para Mac

En el menú **Herramientas**, seleccione **Administrador de paquetes NuGet** > **Consola del Administrador de paquetes** (PMC).



En la Consola del administrador de paquetes, ejecute el comando siguiente:

PowerShell

Copiar

```
Install-Package Microsoft.EntityFrameworkCore.Design
```

Los comandos anteriores agregan:

- El proveedor de SQL Server de EF Core. El paquete de proveedor instala el paquete de EF Core como una dependencia.
- Las utilidades utilizadas por los paquetes se instalaron de forma automática en el paso de scaffolding, más adelante en el tutorial.

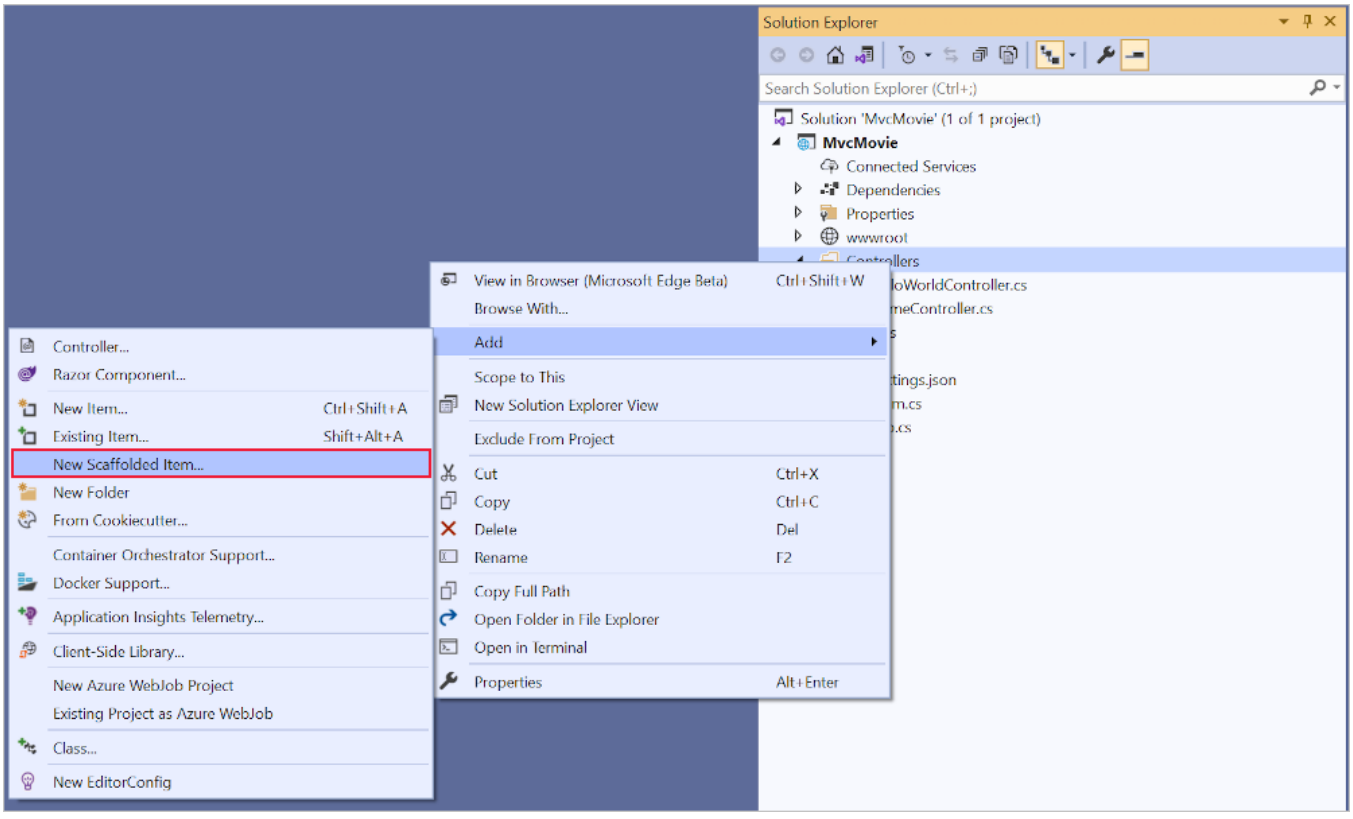
Compile el proyecto para comprobar si hay errores del compilador.

Scaffolding de las páginas de películas

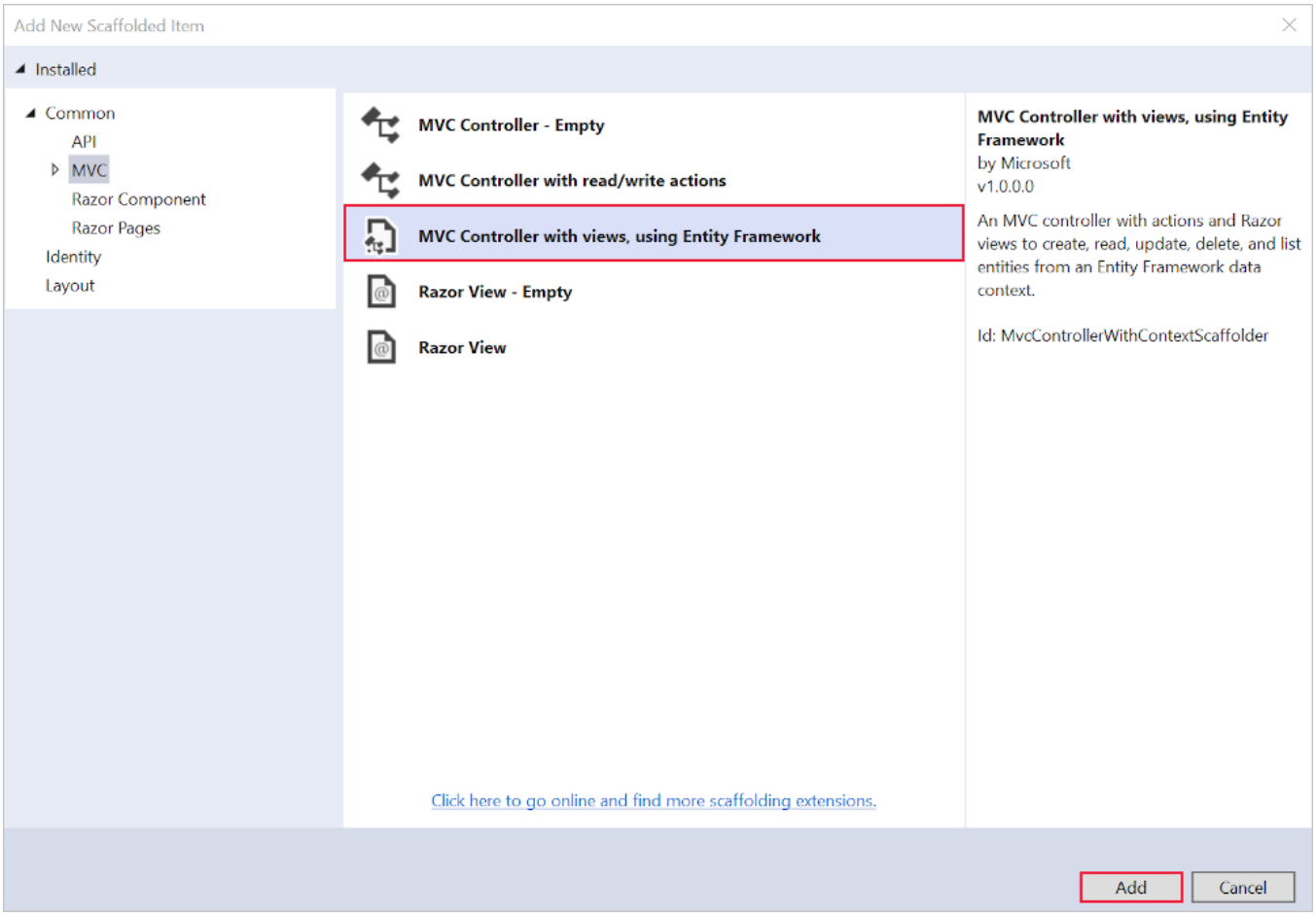
Use la herramienta de scaffolding para generar las páginas `Create`, `Read`, `Update` y `Delete` (CRUD) para el modelo de película.

Visual Studio | Visual Studio Code | Visual Studio para Mac

En el **Explorador de soluciones**, haga clic con el botón derecho en la carpeta *Controladores* y seleccione **Agregar** > **Nuevo elemento con scaffolding**.

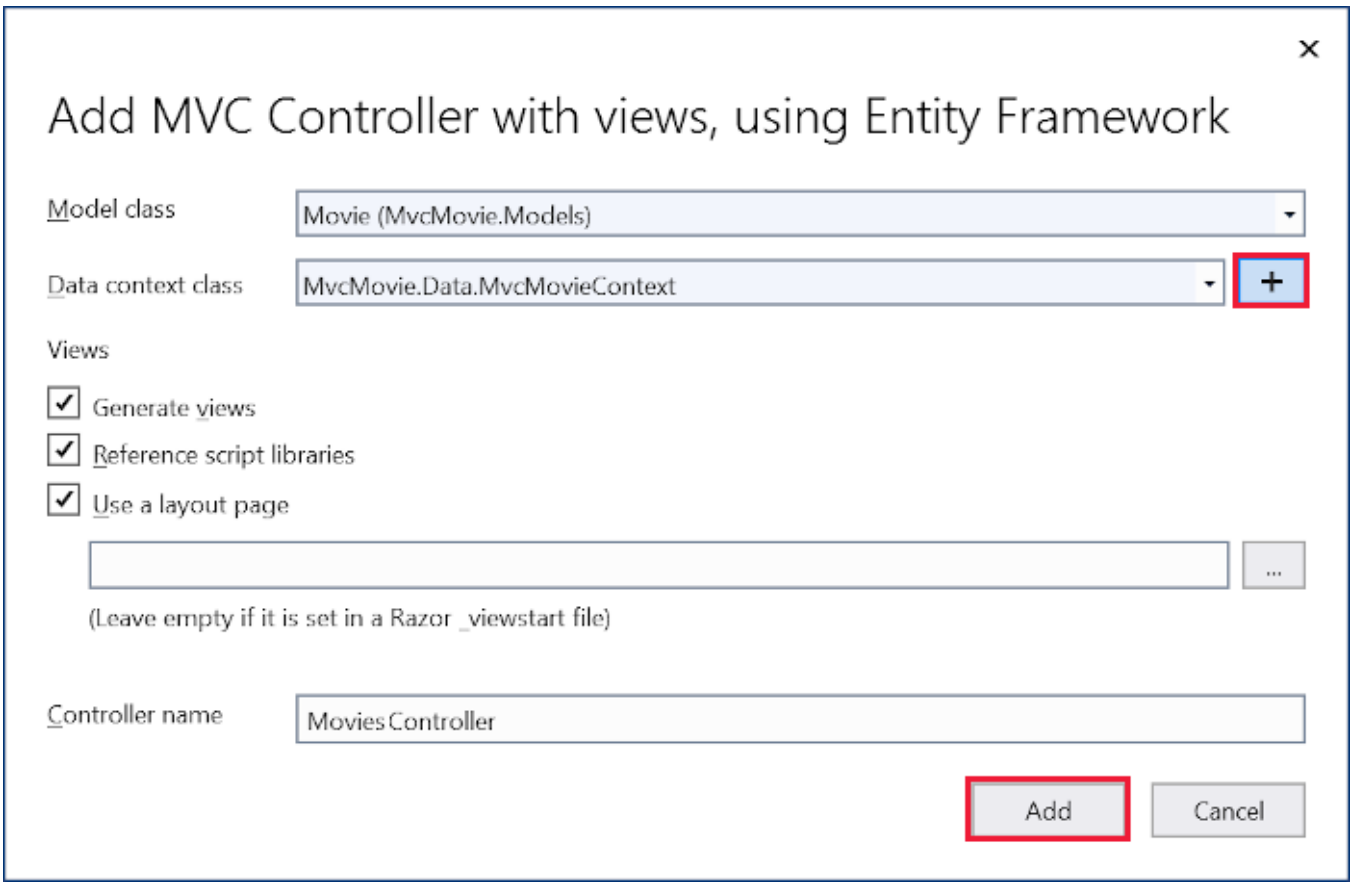


En el cuadro de diálogo **Agregar scaffold**, seleccione **Controlador de MVC con vistas que usan Entity Framework** > **Agregar**.



Complete el cuadro de diálogo **Add MVC Controller with views, using Entity Framework** (Agregar un controlador de MVC con vistas que usan Entity Framework):

- En la lista desplegable **Clase de modelo**, seleccione **Movie (MvcMovie.Models)** .
- En la fila **Clase de contexto de datos**, seleccione el signo + (más).
  - En el cuadro de diálogo **Agregar contexto de datos**, se genera el nombre de clase *MvcMovie.Data.MvcMovieContext*.
  - Seleccione **Agregar**.
- **Vistas** y **Nombre del controlador**: mantenga el valor predeterminado.
- Seleccione **Agregar**.



Scaffolding actualiza lo siguiente:

- Inserta las referencias de paquete necesarias en el archivo del proyecto *MvcMovie.csproj*.
- Registra el contexto de la base de datos en `Startup.ConfigureServices` del archivo *Startup.cs*.
- Agrega una cadena de conexión de la base de datos al archivo *appsettings.json* .

Scaffolding crea lo siguiente:

- Un controlador de películas: *Controllers/MoviesController.cs*
- Archivos de la vista Razor para las páginas **Crear**, **Eliminar**, **Detalles**, **Editar** e **Índice**: `Views/Movies/*.cshtml`
- Una clase de contexto de base de datos: *Data/MvcMovieContext.cs*

La creación automática de estos archivos y actualizaciones de archivos se conoce como *scaffolding*.

Todavía no se pueden usar las páginas con scaffolding porque la base de datos no existe. La ejecución de la aplicación y la selección del vínculo **Movie App** (Aplicación de película) genera un mensaje de error *No se puede abrir la base de datos* o *no se encuentra dicha tabla: Movie*.

Migración inicial

Use la característica Migraciones de EF Core para crear la base de datos. Las migraciones son un conjunto de herramientas que crean y actualizan una base de datos para que coincida con el modelo de datos.

Visual Studio | Visual Studio Code/Visual Studio para Mac

En el menú **Herramientas**, seleccione **Administrador de paquetes NuGet** > **Consola del Administrador de paquetes**.

En la Consola del Administrador de paquetes (PMC), escriba los comandos siguientes:

PowerShell Copiar

```
Add-Migration InitialCreate
Update-Database
```

- `Add-Migration InitialCreate`: Genera un archivo de migración *Migrations/{marca de tiempo}_InitialCreate.cs*. El argumento `InitialCreate` es el nombre de la migración. Se puede usar cualquier nombre, pero, por convención, se selecciona uno que describa la migración. Como se trata de la primera migración, la clase generada contiene código para crear el esquema de la base de datos. El esquema de la base de datos se basa en el modelo especificado en la clase `MvcMovieContext`.
- `Update-Database`: actualiza la base de datos a la migración más reciente, que ha creado el comando anterior. El comando ejecuta el método `Up` en el archivo *Migrations/{marca de tiempo}_InitialCreate.cs*, que crea la base de datos.

El comando `Update-Database` genera la siguiente advertencia:

> No type was specified for the decimal column "Price" on entity type "Movie" (No se ha especificado ningún tipo en la columna decimal "Price" en el tipo de entidad "Movie"). This will cause values to be silently truncated if they do not fit in the default precision and scale. Explicitly specify the SQL server column type that can accommodate all the values using "HasColumnType()" (Especifique de forma explícita el tipo de columna de SQL Server que pueda acomodar todos los valores mediante "HasColumnType()").

Ignore la advertencia anterior, ya que se ha corregido en un tutorial posterior.

Para obtener más información sobre las herramientas de PMC para EF Core, vea Referencia de herramientas de EF Core: PMC en Visual Studio.

Prueba de la aplicación

Ejecute la aplicación y seleccione el vínculo **Movie App** (Aplicación de película).

Si recibe una excepción similar a la siguiente, es posible que haya perdido el paso de migraciones:

Visual Studio | Visual Studio Code/Visual Studio para Mac

Consola Copiar

```
SqlException: Cannot open database "MvcMovieContext-1" requested by the login. The
login failed.
```

Nota

Es posible que no pueda escribir comas decimales en el campo `Price`. La aplicación debe globalizarse para que la **validación de jQuery** sea compatible con configuraciones regionales distintas del inglés que usan una coma (",") en lugar de un punto decimal y formatos de fecha distintos del de Estados Unidos. Para obtener instrucciones sobre la globalización, consulte **esta cuestión en GitHub**.

Examen del registro y la clase del contexto de la base de datos generados

Con EF Core, el acceso a datos se realiza mediante un modelo. Un modelo se compone de clases de entidad y un objeto de contexto que representa una sesión con la base de datos. Este objeto de contexto permite consultar y guardar datos. El contexto de base de datos se deriva de Microsoft.EntityFrameworkCore.DbContext y especifica las entidades que se van a incluir en el modelo de datos.

Scaffolding crea la clase de contexto de la base de datos *Data/MvcMovieContext.cs*:

C# Copiar

```
using Microsoft.EntityFrameworkCore;
using MvcMovie.Models;

namespace MvcMovie.Data
{
    public class MvcMovieContext : DbContext
    {
        public MvcMovieContext (DbContextOptions<MvcMovieContext> options)
            : base(options)
        {
        }

        public DbSet<Movie> Movie { get; set; }
    }
}
```

El código anterior crea una propiedad DbSet<Movie> que representa las películas de la base de datos.

ASP.NET Core integra la inserción de dependencias (DI). Los servicios, como el contexto de la base de datos, deben registrarse con DI en `Startup`. Estos servicios se proporcionan a los componentes que los necesitan a través de parámetros de constructor.

En el archivo *Controllers/MoviesController.cs*, el constructor usa Inserción de dependencias para insertar el contexto de la base de datos `MvcMovieContext` en el controlador. El contexto de base de datos se usa en cada uno de los métodos CRUDdel controlador.

El scaffolding generó el siguiente código resaltado en `Startup.ConfigureServices`:

Visual Studio | Visual Studio Code | Visual Studio para Mac

C# Copiar

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddControllersWithViews();

    services.AddDbContext<MvcMovieContext>(options =>
    options.UseSqlServer(Configuration.GetConnectionString("MvcMovieContext")));
}
```

El sistema de configuración de ASP.NET Core lee la cadena de conexión de la base de datos "MvcMovieContext".

Examen de la cadena de conexión de base de datos generada

Scaffolding agregó una cadena de conexión al archivo *appsettings.json* :

Visual Studio | Visual Studio Code/Visual Studio para Mac

JSON Copiar

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MvcMovieContext": "Server=(localdb)\\mssqllocaldb;Database=MvcMovieContext-
1;Trusted_Connection=True;MultipleActiveResultSets=true"
  }
}
```

Para el desarrollo local, el sistema de configuración de ASP.NET Core lee la clave `ConnectionString` del archivo *appsettings.json* .

La clase `InitialCreate`.

Examine el archivo de migración *Migrations/{marca de tiempo}_InitialCreate.cs*:

C# Copiar

```
public partial class InitialCreate : Migration
{
    protected override void Up(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.CreateTable(
            name: "Movie",
            columns: table => new
            {
                Id = table.Column<int>(type: "int", nullable: false)
                    .Annotation("SqlServer:Identity", "1, 1"),
                Title = table.Column<string>(type: "nvarchar(max)", nullable: true),
                ReleaseDate = table.Column<DateTime>(type: "datetime2", nullable:
false),
                Genre = table.Column<string>(type: "nvarchar(max)", nullable: true),
                Price = table.Column<decimal>(type: "decimal(18,2)", nullable: false)
            },
            constraints: table =>
            {
                table.PrimaryKey("PK_Movie", x => x.Id);
            });
    }

    protected override void Down(MigrationBuilder migrationBuilder)
    {
        migrationBuilder.DropTable(
            name: "Movie");
    }
}
```

En el código anterior:

- `InitialCreate.Up` crea la tabla Movie y configura `Id` como la clave principal.
- `InitialCreate.Down` revierte los cambios de esquema realizados por la migración `Up`.

Inserción de dependencias en el controlador

Abra el archivo *Controllers/MoviesController.cs* y examine el constructor:

C# Copiar

```
public class MoviesController : Controller
{
    private readonly MvcMovieContext _context;

    public MoviesController(MvcMovieContext context)
    {
        _context = context;
    }
```

El constructor usa la inserción de dependencias para insertar el contexto de base de datos (`MvcMovieContext`) en el controlador. El contexto de base de datos se usa en cada uno de los métodos CRUD del controlador.

Pruebe la página **Create**. Escriba y envíe los datos.

Pruebe las páginas **Edit**, **Details** y **Delete**.

Modelos fuertemente tipados y la directiva `@model`

Anteriormente en este tutorial, vimos cómo un controlador puede pasar datos u objetos a una vista mediante el diccionario `ViewData`. El diccionario `ViewData` es un objeto dinámico que proporciona una cómoda manera enlazada en tiempo de ejecución de pasar información a una vista.

MVC ofrece la capacidad de pasar objetos de modelo fuertemente tipados a una vista. Este enfoque fuertemente tipado permite comprobar el código en tiempo de compilación. El mecanismo de scaffolding pasó un modelo fuertemente tipado en las vistas y la clase `MoviesController`.

Examine el método `Details` generado en el archivo *Controllers/MoviesController.cs*:

C# Copiar

```
// GET: Movies/Details/5
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return NotFound();
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return NotFound();
    }

    return View(movie);
}
```

El parámetro `id` suele pasarse como datos de ruta. Por ejemplo, `https://localhost:5001/movies/details/1` establece:

- El controlador en el controlador `movies`, el primer segmento de dirección URL.
- La acción en `details`, el segundo segmento de dirección URL.
- `id` en 1,el último segmento de dirección URL.

`id` se puede pasar con una cadena de consulta, como en el ejemplo siguiente:

`https://localhost:5001/movies/details?id=1`

El parámetro `id` se define como un tipo que acepta valores NULL (`int?`) en caso de que no se proporcione un valor `id`.

Se pasa una expresión lambda al método FirstOrDefaultAsync para seleccionar entidades de película que coincidan con los datos de enrutamiento o el valor de consulta de cadena.

C# Copiar

```
var movie = await _context.Movie
    .FirstOrDefaultAsync(m => m.Id == id);
```

Si se encuentra una película, se pasa una instancia del modelo `Movie` a la vista `Details`:

C# Copiar

```
return View(movie);
```

Examine el contenido del archivo *Views/Movies/Details.cshtml*:

CSHTML Copiar

```
@model MvcMovie.Models.Movie

@{
    ViewData["Title"] = "Details";
}

<h1>Details</h1>

<div>
    <h4>Movie</h4>
    <hr />
    <dl class="row">
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Title)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Title)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.ReleaseDate)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.ReleaseDate)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Genre)
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Genre)
        </dd>
        <dt class="col-sm-2">
            @Html.DisplayNameFor(model => model.Price)
```

```
        </dt>
        <dd class="col-sm-10">
            @Html.DisplayFor(model => model.Price)
        </dd>
    </dl>
</div>
<div>
    <a asp-action="Edit" asp-route-id="@Model.Id">Edit</a> |
    <a asp-action="Index">Back to List</a>
</div>
```

La instrucción `@model` de la parte superior del archivo de vista especifica el tipo de objeto que espera la vista. Cuando se ha creado el controlador de película, se ha incluido la siguiente instrucción `@model`:

CSHTML Copiar

```
@model MvcMovie.Models.Movie
```

Esta directiva `@model` permite el acceso a la película que el controlador ha pasado a la vista. El objeto `Model` está fuertemente tipado. Por ejemplo, en la vista *Details.cshtml*, el código pasa cada campo de película a los asistentes de HTML `DisplayNameFor` y `DisplayFor` con el objeto `Model` fuertemente tipado. Los métodos `Create` y `Edit` y las vistas también pasan un objeto de modelo `Movie`.

Examine la vista *Index.cshtml* y el método `Index` en el controlador Movies. Observe cómo el código crea un objeto `List` cuando llama al método `View`. El código pasa esta lista `Movies` desde el método de acción `Index` a la vista:

C# Copiar

```
// GET: Movies
public async Task<IActionResult> Index()
{
    return View(await _context.Movie.ToListAsync());
}
```

Cuando se ha creado el controlador de películas, el scaffolding ha incluido la siguiente instrucción `@model` en la parte superior del archivo *Index.cshtml*:

CSHTML Copiar

```
@model IEnumerable<MvcMovie.Models.Movie>
```

La directiva `@model` permite acceder a la lista de películas que el controlador pasó a la vista usando un objeto `Model` fuertemente tipado. Por ejemplo, en la vista *Index.cshtml*, el código recorre en bucle las películas con una instrucción `foreach` sobre el objeto `Model` fuertemente tipado:

CSHTML Copiar

```
@model IEnumerable<MvcMovie.Models.Movie>


@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Title)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.ReleaseDate)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Genre)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Price)
            </th>
            <th></th>
        </tr>
    </thead>
    <tbody>
@foreach (var item in Model) {

        <tr>
            <td>
                @Html.DisplayFor(modelItem => item.Title)

            </td>
            <td>
                @Html.DisplayFor(modelItem => item.ReleaseDate)


            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Genre)

            </td>
            <td>
                @Html.DisplayFor(modelItem => item.Price)

            </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>
```

Como el objeto `Model` es fuertemente tipado como un objeto `IEnumerable<Movie>`, cada elemento del bucle está tipado como `Movie`. Entre otras ventajas, el compilador valida los tipos usados en el código.

Registros SQL de Entity Framework Core

La configuración de registros suele proporcionarla la sección `Logging` de los archivos *appsettings.* `{Environment}` *.json*. Para registrar instrucciones SQL, agregue `"Microsoft.EntityFrameworkCore.Database.Command": "Information"` al archivo *appsettings.Development.json*:

JSON Copiar

```
{
  "ConnectionStrings": {
    "DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=MyDB-
2;Trusted_Connection=True;MultipleActiveResultSets=true"
  },
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
     ,"Microsoft.EntityFrameworkCore.Database.Command": "Information"

    }
  },
  "AllowedHosts": "*"
}
```

Con el archivo JSON anterior, las instrucciones SQL se muestran en la línea de comandos y en ventana de salida de Visual Studio.

Para más información, consulte Registros en .NET Core y ASP.NET Core y esta incidencia de GitHub.

Recursos adicionales

- Asistentes de etiquetas
- Globalización y localización