Tutorial: Create an ASP.NET Core Razor Pages web app project

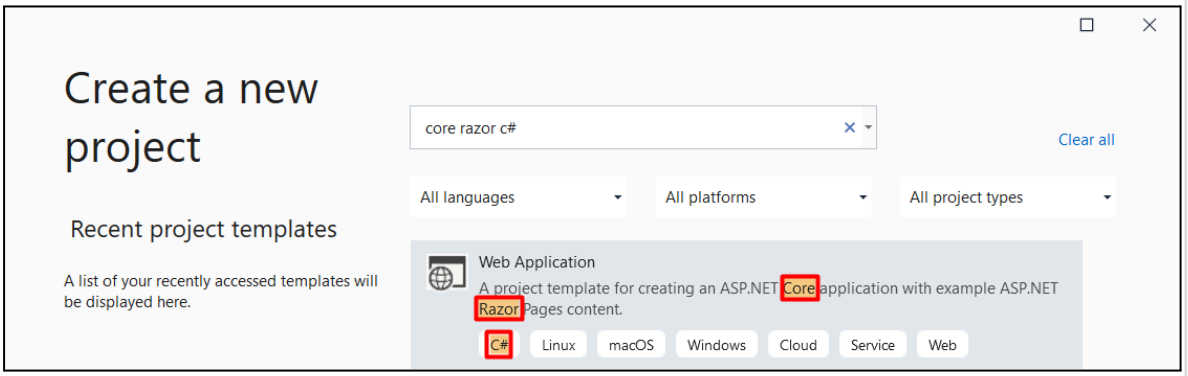
⚠️ This tutorial is based on Visual Studio 2019 16.11 and ASP.Net Core 3.1

This project will create a database file in the user directory

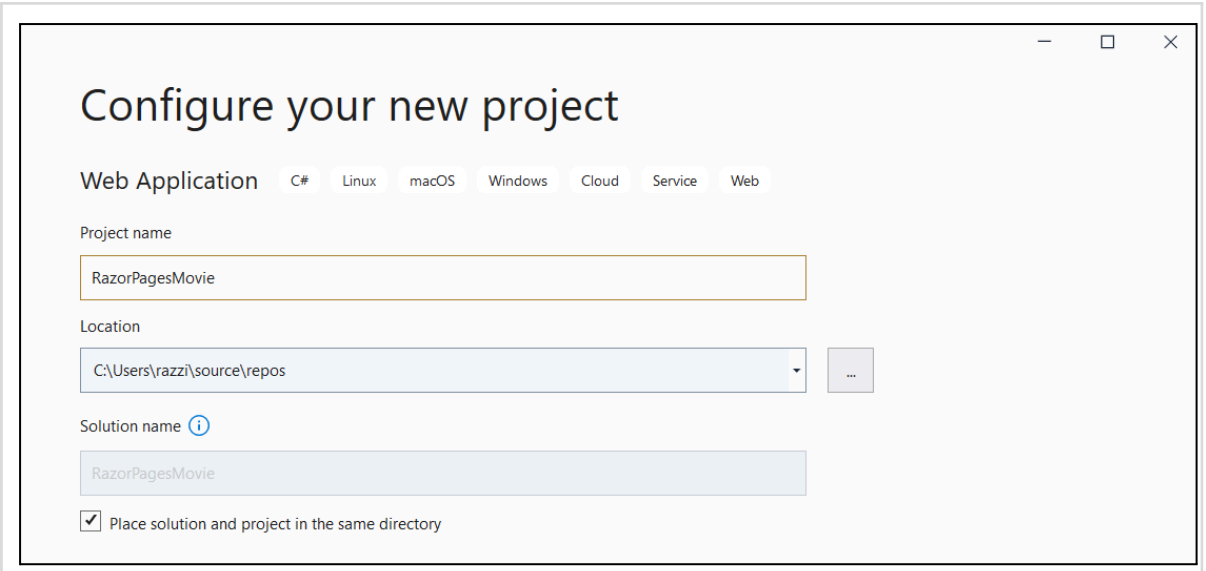
e.g.
C:\Users\razzi\RazorPagesMovieContext.mdf
And
C:\Users\razzi\RazorPagesMovieContext_log.ldf

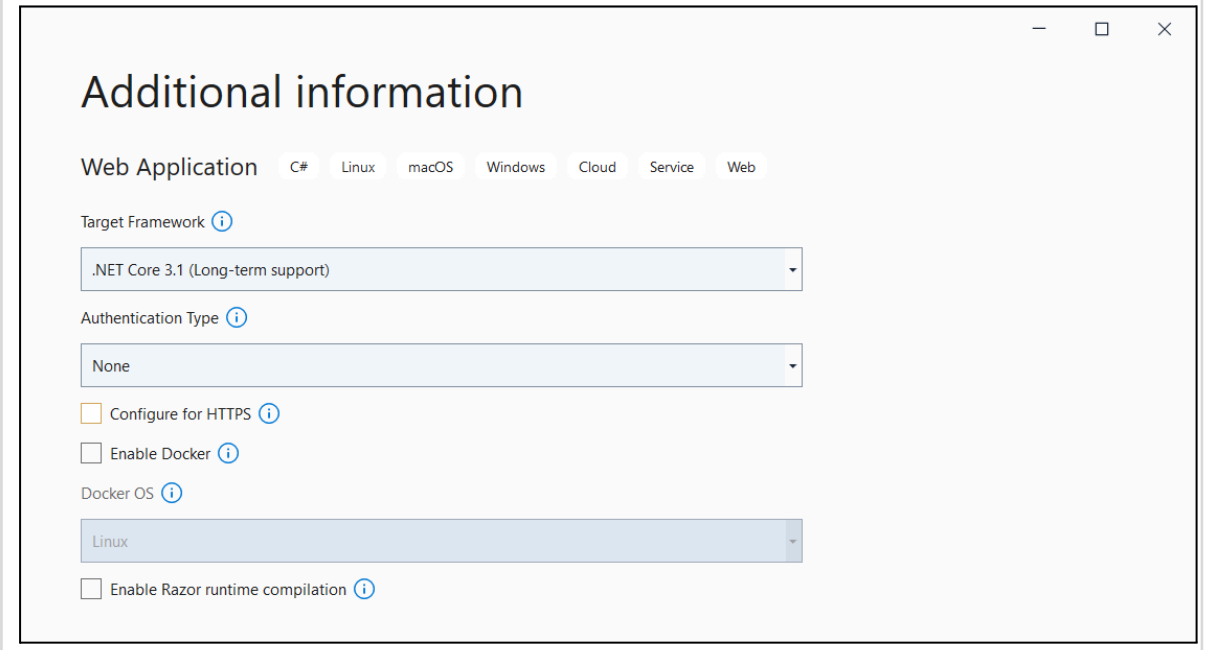
Create a new ASP.NET Core Web Razor project

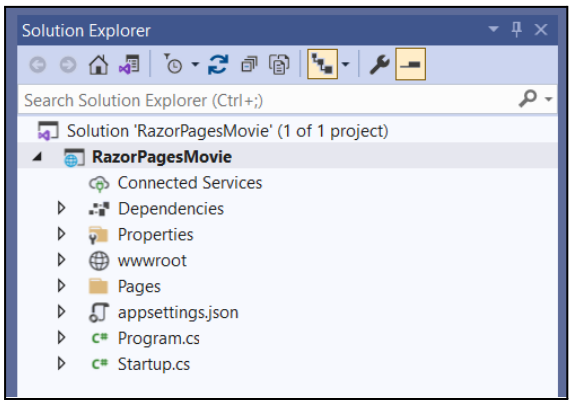
Search for the template using the keywords core razor c#



Project name = RazorPagesMovie


Configure your new project
Web Application
C#
Linux
macOS
Windows
Cloud
Service
Web
Project name
RazorPagesMovie
Location
C:\Users\razzi\source\repos
...
Solution name
RazorPagesMovie
Place solution and project in the same directory

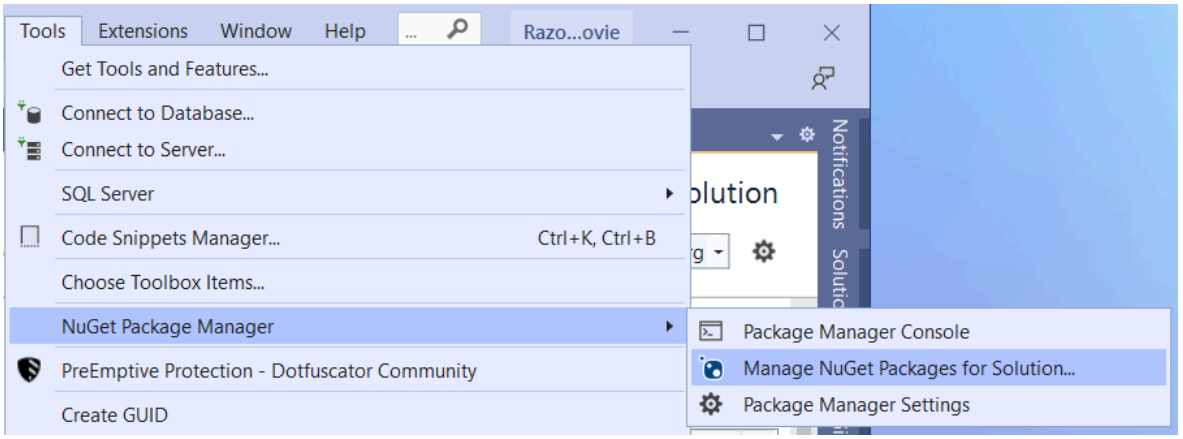

Additional information
Web Application
C#
Linux
macOS
Windows
Cloud
Service
Web
Target Framework
.NET Core 3.1 (Long-term support)
Authentication Type
None
Configure for HTTPS
Enable Docker
Docker OS
Linux
Enable Razor runtime compilation


Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'RazorPagesMovie' (1 of 1 project)
RazorPagesMovie
Connected Services
Dependencies
Properties
wwwroot
Pages
appsettings.json
Program.cs
Startup.cs

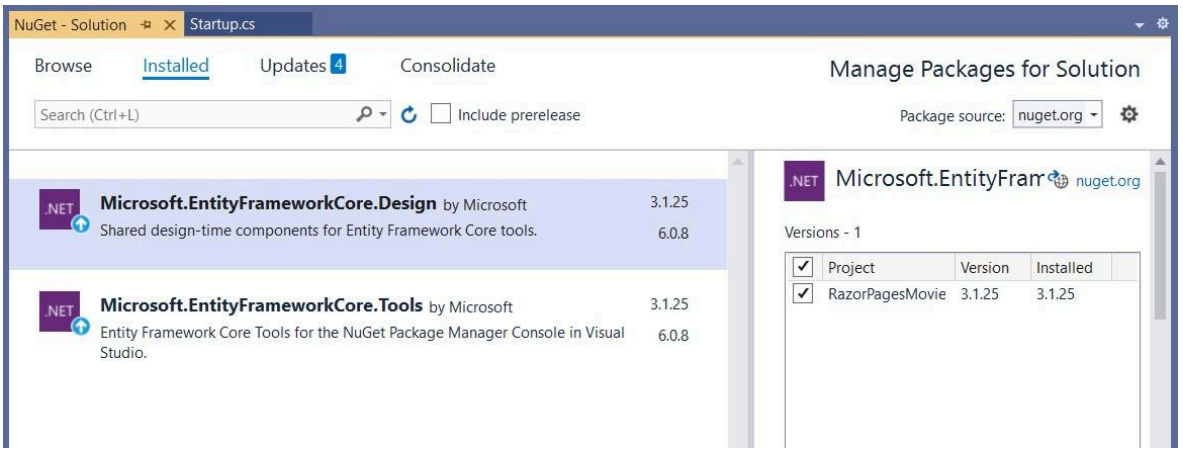
Check Prerequisite Packages

This tutorial requires the following packages:
Microsoft.EntityFrameworkCore.Design
Microsoft.EntityFrameworkCore.Tools

Open the NuGet Package Manager i.e. select menu Tools/NuGet Package Manager/Manage Nuget Packages for Solution …

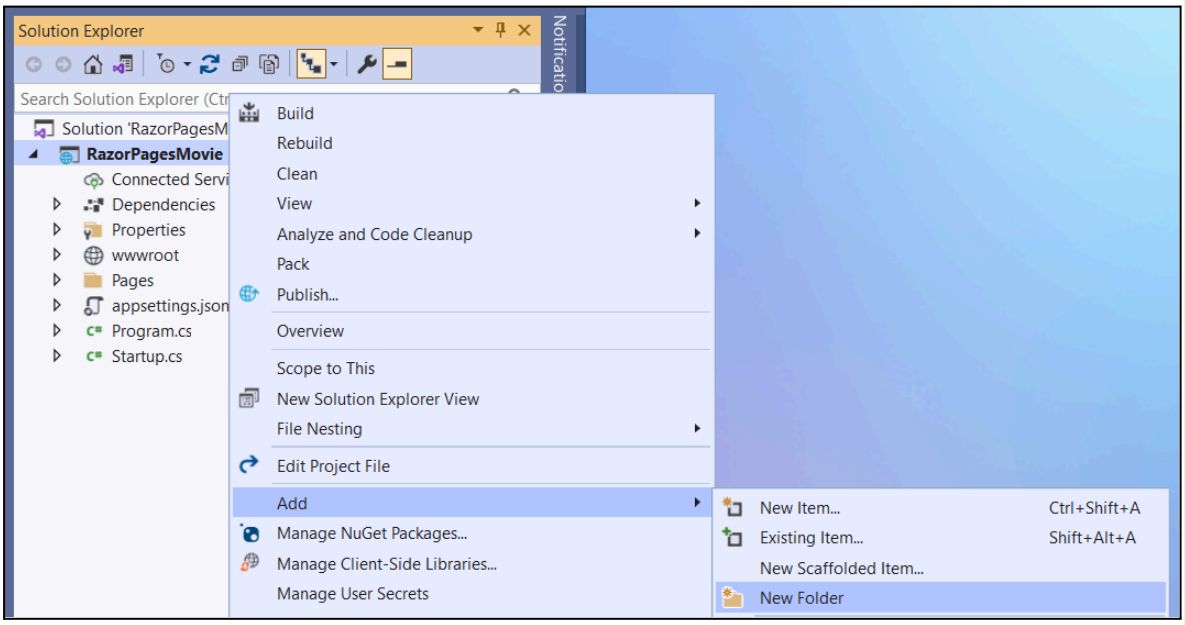


Click the Installed tab.
Check that the package name and its version number is compatible, e.g. 3.1.25.
If the package is not displayed, click Browse tab, search for the package and install.

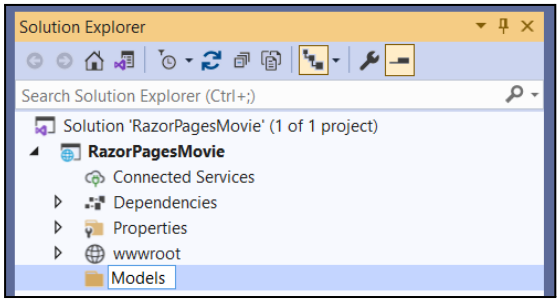


Add a data model

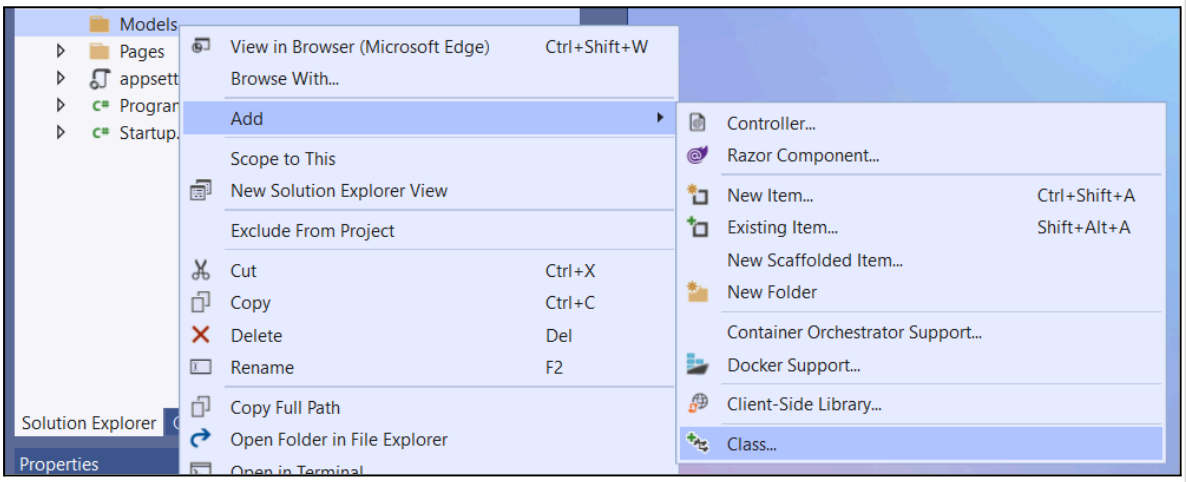
Right-click the RazorPagesMovie project > Add > New Folder.

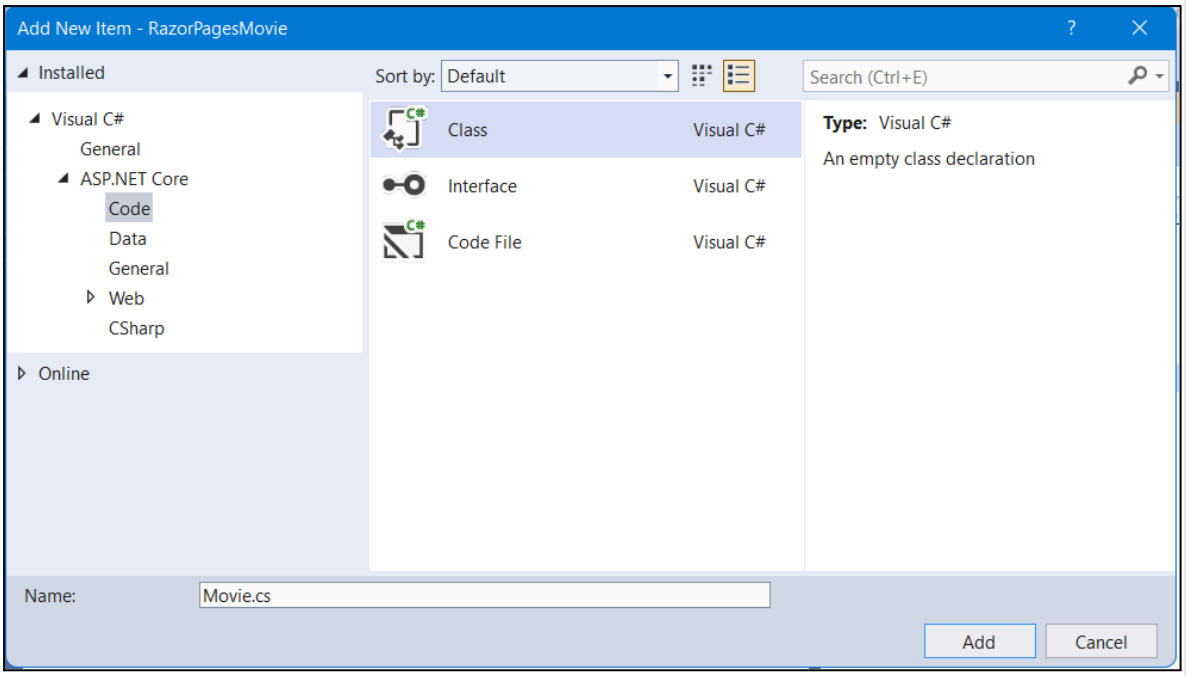


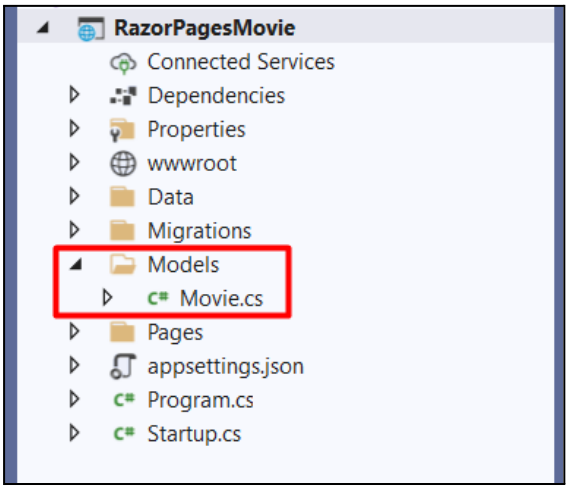
Name the folder Models.



Right-click the Models folder. Select Add > Class. Name the class Movie.



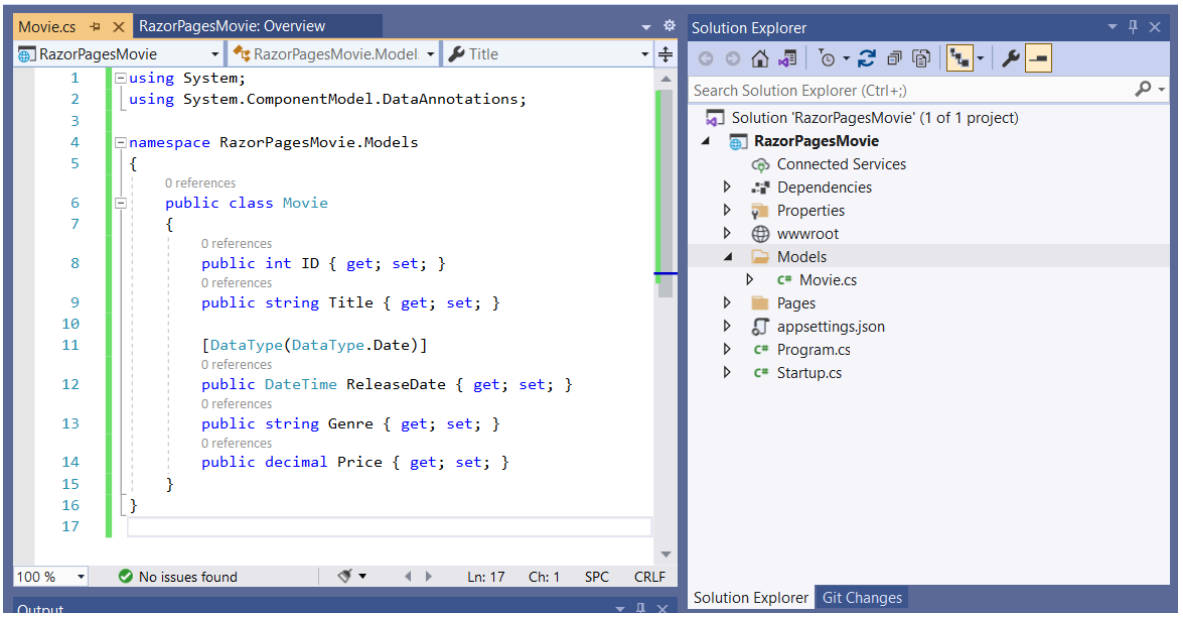




Add the following properties to the Movie class:

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

namespace RazorPagesMovie.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; }
    }
}
```



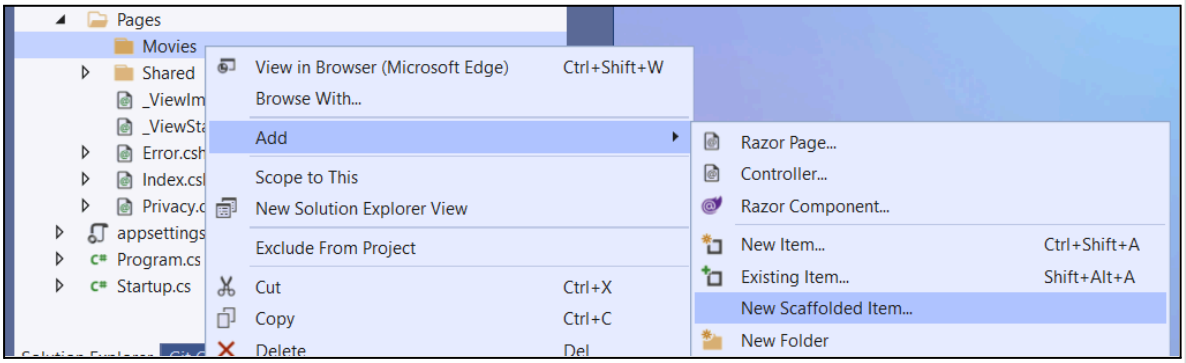
The `Movie` class contains:

- The `ID` field is required by the database for the primary key.
- `[DataType(DataType.Date)]`: The DataType attribute specifies the type of the data (`Date`). With this attribute:
    - The user is not required to enter time information in the date field.
    - Only the date is displayed, not time information.

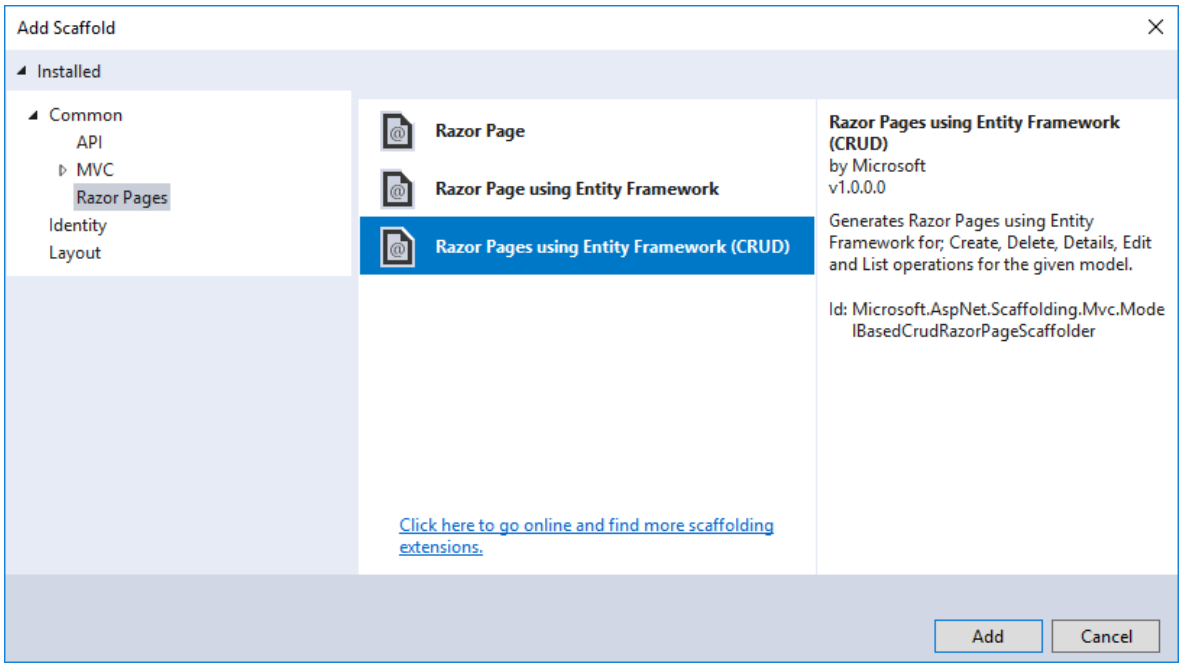
Scaffold the movie model

Create a Pages/Movies folder:

Right-click on the Pages folder > Add > New Folder.
Name the folder Movies.
Right-click on the Pages/Movies folder > Add > New Scaffolded Item.



In the Add Scaffold dialog, select Razor Pages using Entity Framework (CRUD) > Add.

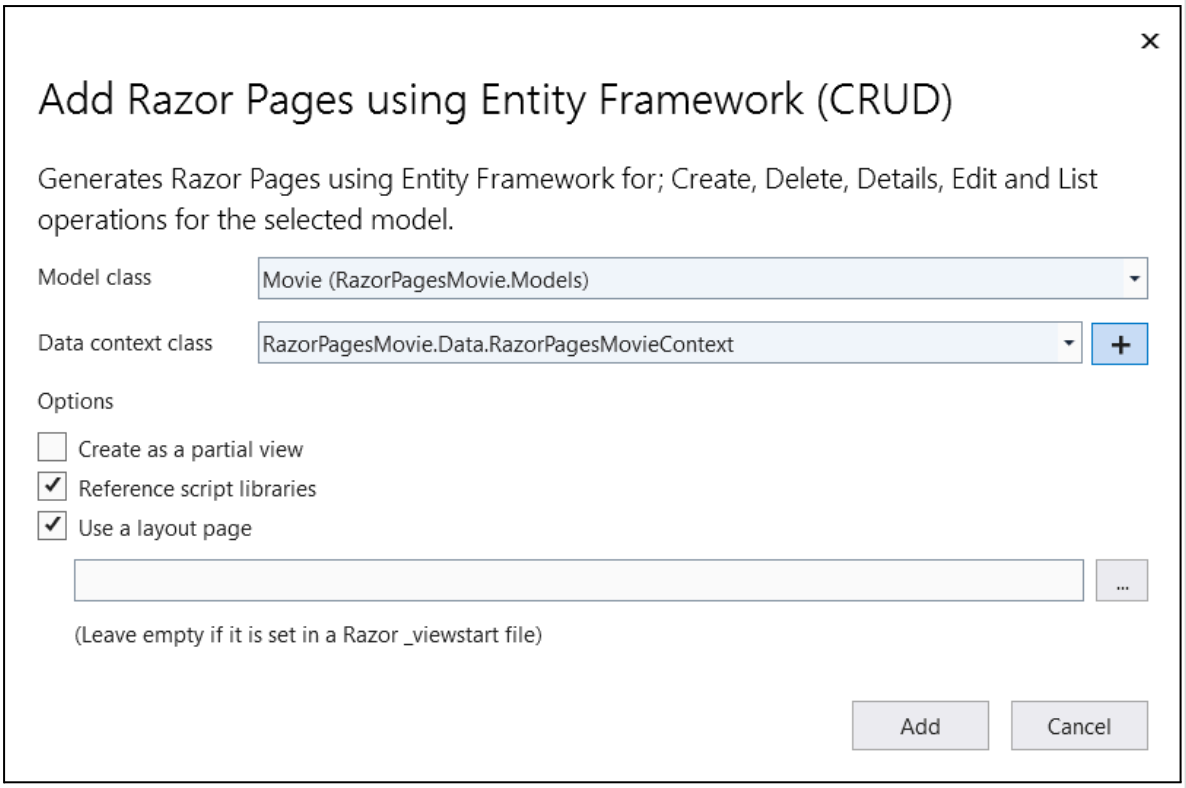


Complete the Add Razor Pages using Entity Framework (CRUD) dialog:

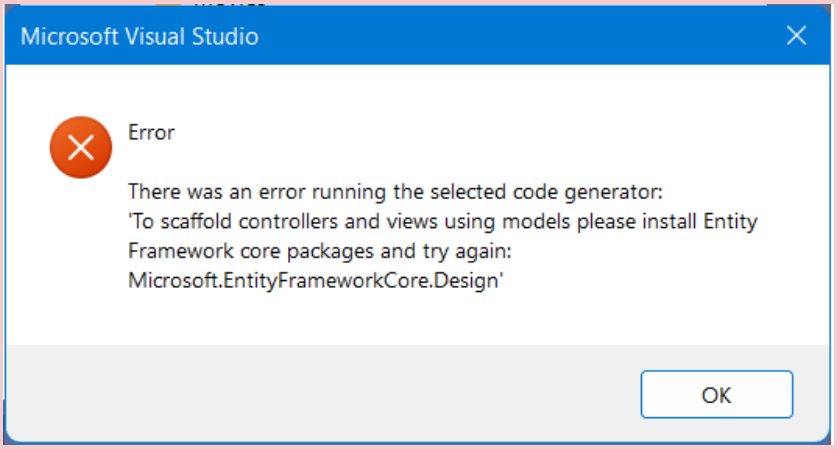
In the Model class drop down, select Movie (RazorPagesMovie.Models).

In the Data context class row, select the + (plus) sign and change the generated name from RazorPagesMovie.Models.RazorPagesMovieContext to RazorPagesMovie.Data.RazorPagesMovieContext.

Select Add.



If you receive an error message stating that a package is missing i.e. Microsoft.EntityFrameworkCore.Design, use Nuget Package Manager to install the package.

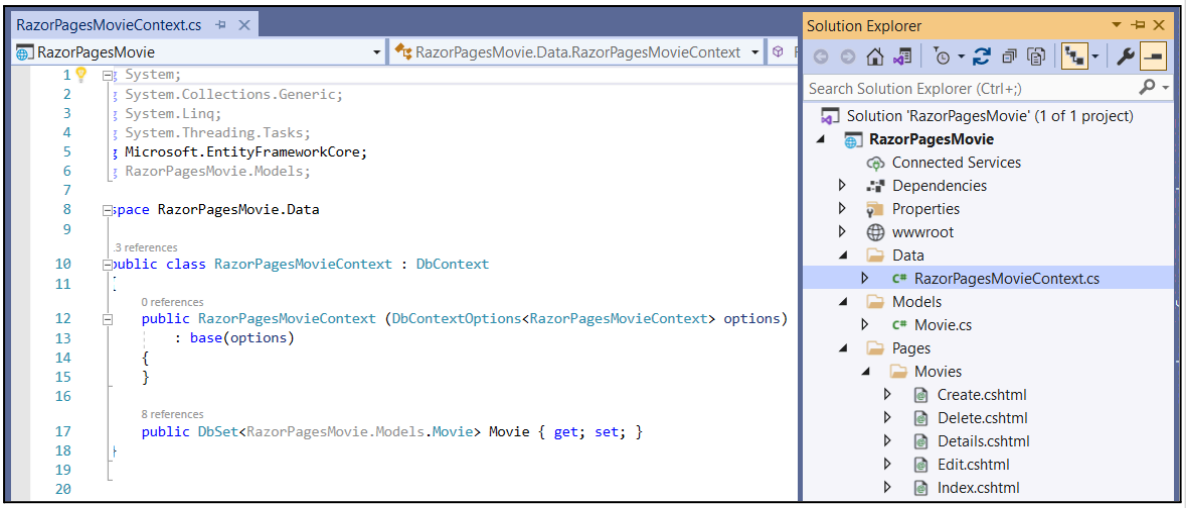


Repeat the Scaffolding step again.

The scaffold process creates the following files:

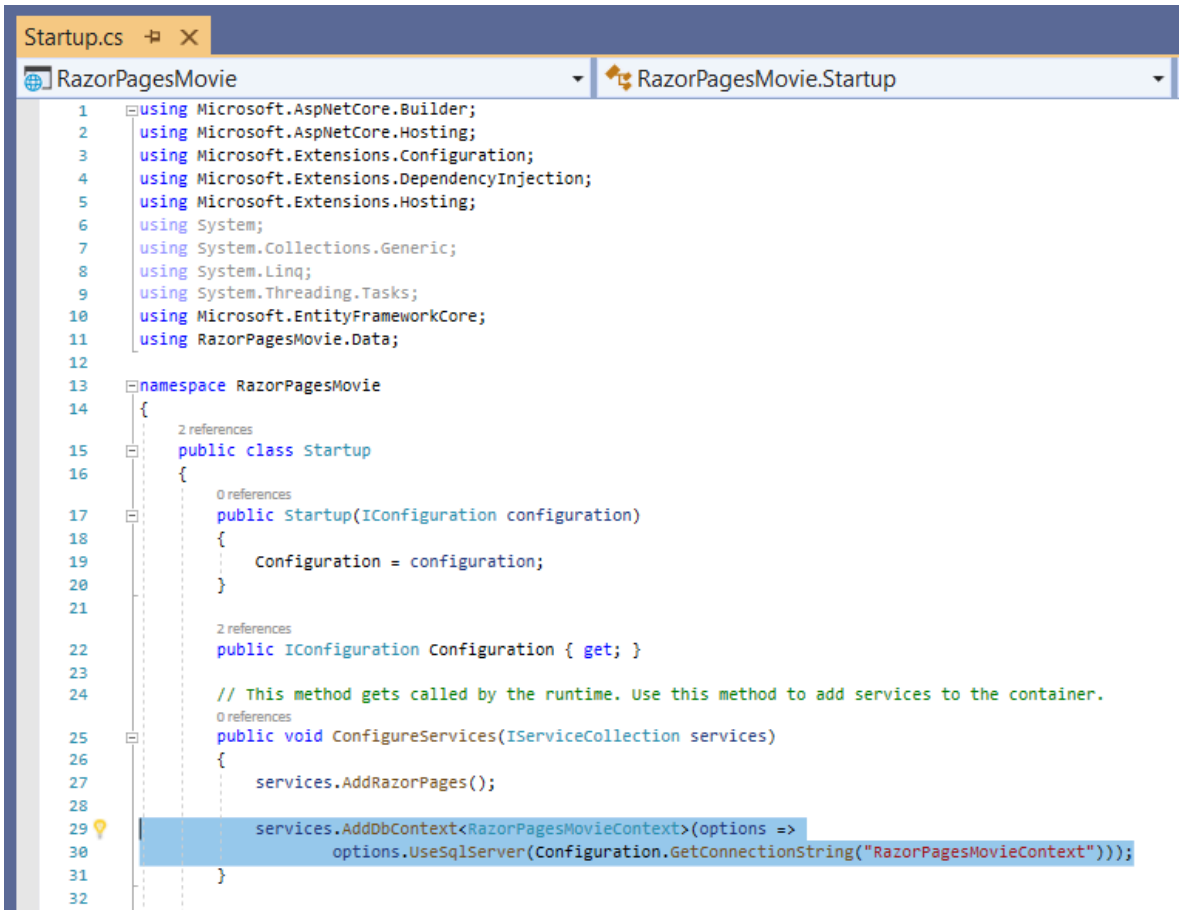
Pages/Movies: Create, Delete, Details, Edit, and Index.

Data/RazorPagesMovieContext.cs



The scaffold process updates the following file:

Startup.cs



File name: Startup.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.HttpsPolicy;
using Microsoft.Extensions.Configuration;
```

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;
using RazorPagesMovie.Data;

namespace RazorPagesMovie
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add
services to the container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddRazorPages();

            services.AddDbContext<RazorPagesMovieContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("RazorPagesMovieContext"))
);
        }

        // This method gets called by the runtime. Use this method to configure
the HTTP request pipeline.
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            else
            {
                app.UseExceptionHandler("/Error");
                // The default HSTS value is 30 days. You may want to change this
for production scenarios, see https://aka.ms/aspnetcore-hsts.
                app.UseHsts();
            }

            app.UseHttpsRedirection();
            app.UseStaticFiles();

            app.UseRouting();

            app.UseAuthorization();

            app.UseEndpoints(endpoints =>
            {
```

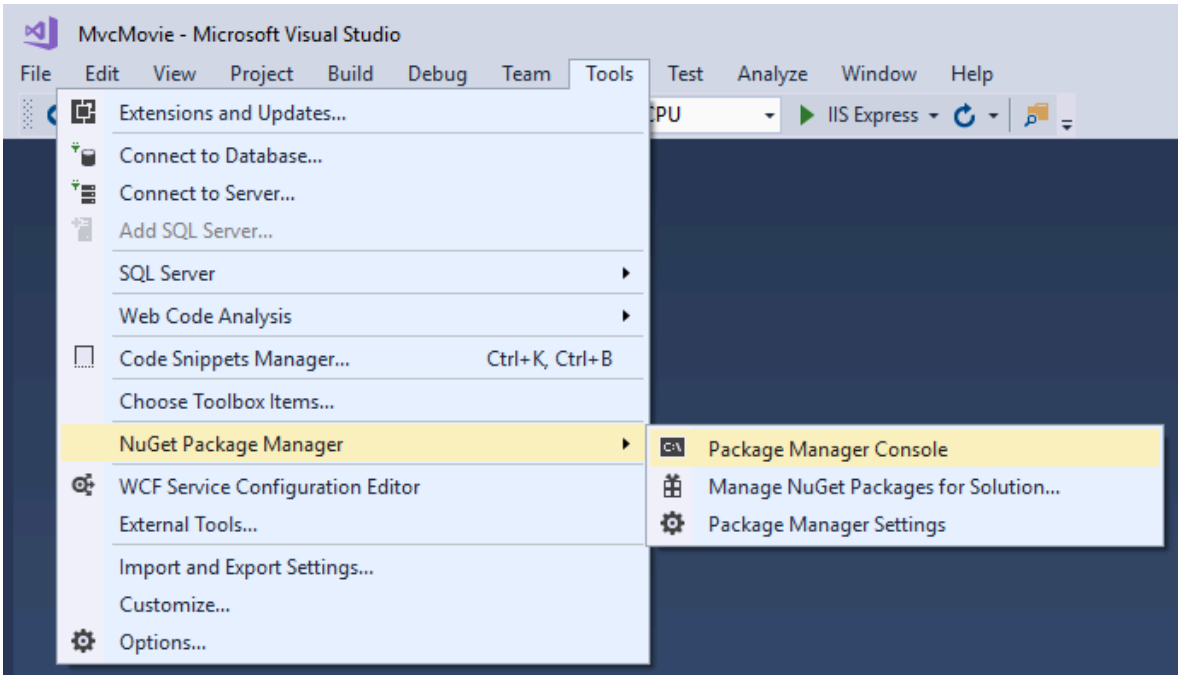
```
                endpoints.MapRazorPages();
            });
        }
    }
}
```

Initial migration

In this section, the Package Manager Console (PMC) is used to:

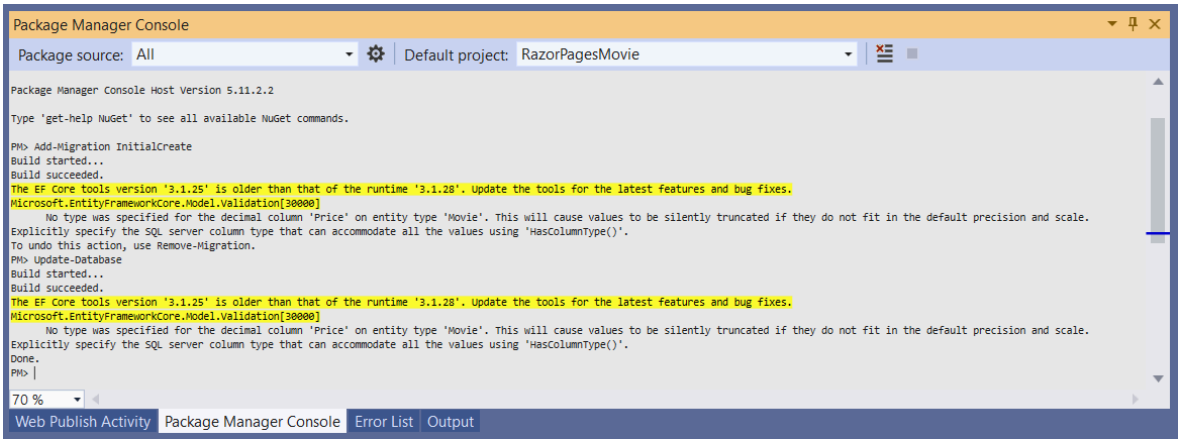
- Add an initial migration.
- Update the database with the initial migration.

From the Tools menu, select NuGet Package Manager > Package Manager Console.



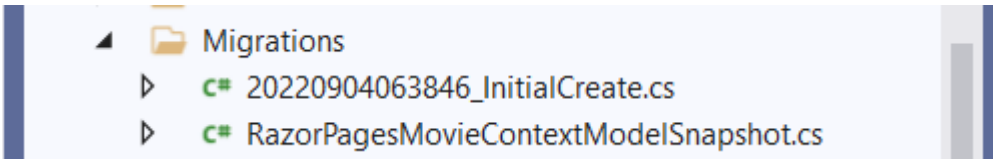
In the PMC, enter the following commands:

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



The migrations command generates code to create the initial database schema. The schema is based on the model specified in `DbContext`. The `InitialCreate` argument is used to name the migrations. Any name can be used, but by convention a name is selected that describes the migration.

The `update` command runs the `Up` method in migrations that have not been applied. In this case, `update` runs the `Up` method in `Migrations/<time-stamp>_InitialCreate.cs` file, which creates the database.



Examine the context registered with dependency injection

ASP.NET Core is built with dependency injection. Services, such as the EF Core database context context, are registered with dependency injection during application startup. Components that require these services, such as Razor Pages, are provided via constructor parameters. The constructor code that gets a database context context instance is shown later in the tutorial.

The scaffolding tool automatically created a database context context and registered it with the dependency injection container.

Examine the `Startup.ConfigureServices` method. The highlighted line was added by the scaffolder:

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

    services.AddDbContext<RazorPagesMovieContext>(options =>

options.UseSqlServer(Configuration.GetConnectionString("RazorPagesMovieContext")));
}
```

The `RazorPagesMovieContext` coordinates EF Core functionality, such as Create, Read, Update and Delete, for the `Movie` model. The data context (`RazorPagesMovieContext`) is derived from Microsoft.EntityFrameworkCore.DbContext. The data context specifies which entities are included in the data model.

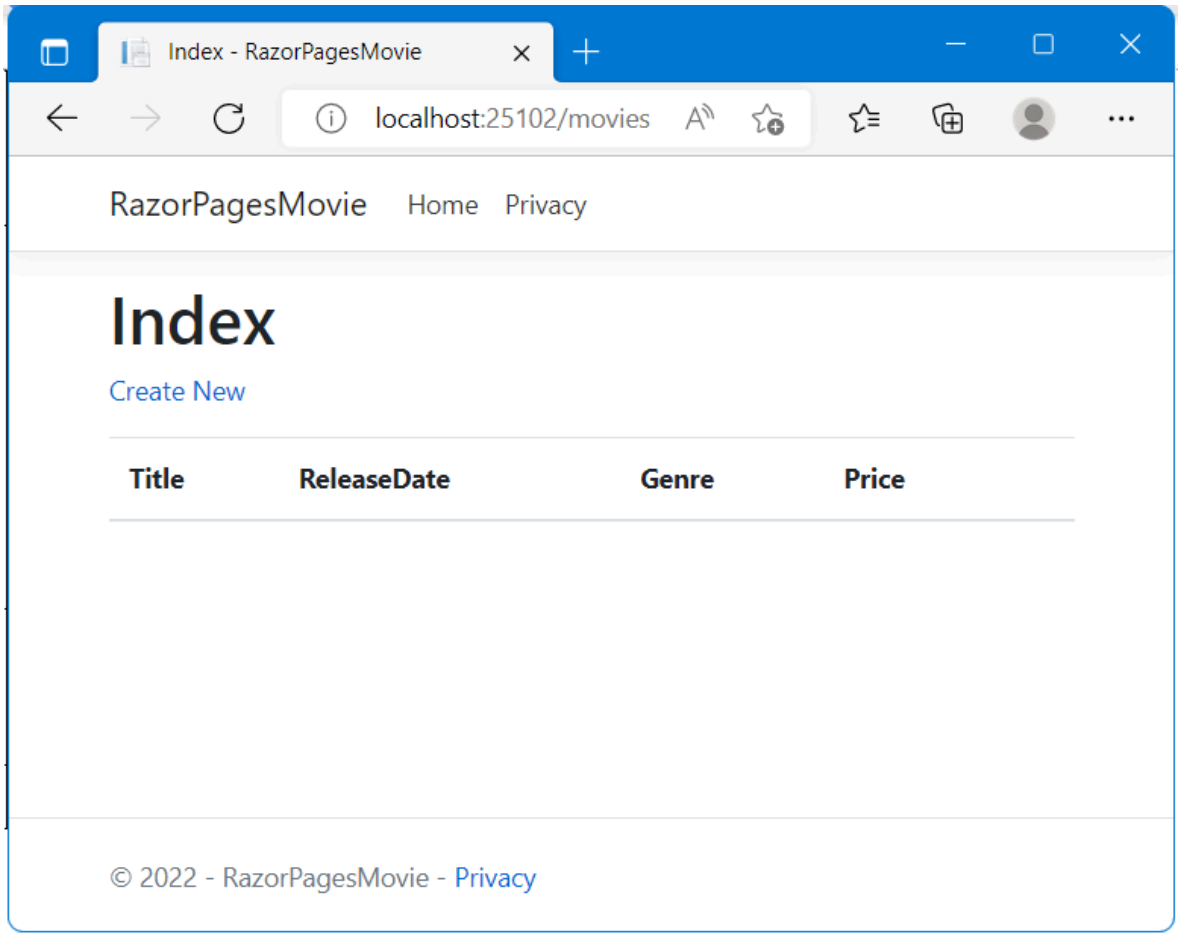
```
using Microsoft.EntityFrameworkCore;

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

        public DbSet<RazorPagesMovie.Models.Movie> Movie { get; set; }
    }
}
```

The preceding code creates a DbSet<Movie> property for the entity set. In Entity Framework terminology, an entity set typically corresponds to a database table. An entity corresponds to a row in the table.

The name of the connection string is passed in to the context by calling a method on a DbContextOptions object. For local development, the Configuration system reads the connection string from the `appsettings.json` file.

Test the app

Run the app and append `/Movies` to the URL in the browser (`http://localhost:port/movies`).

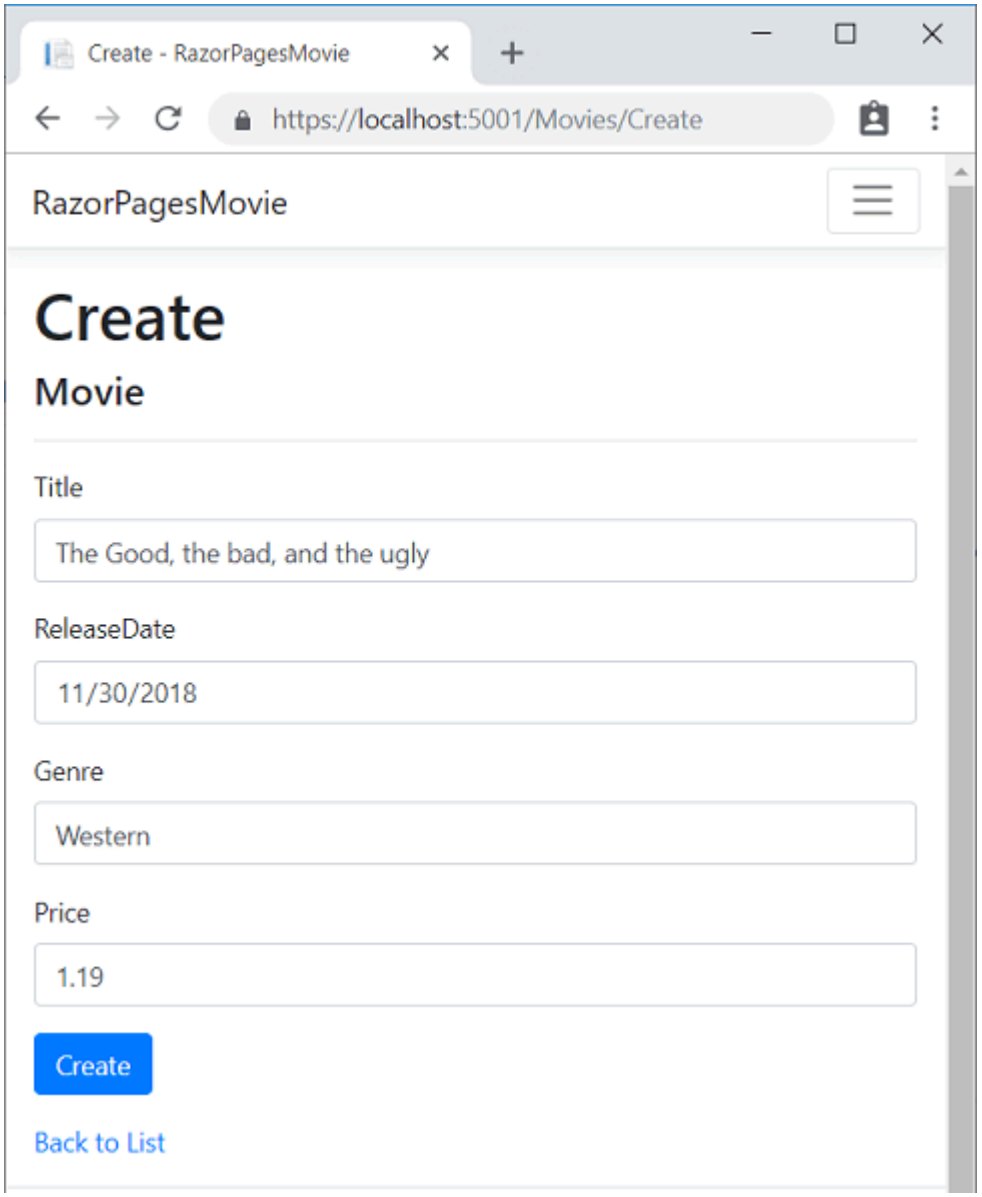


If you get the following error, you might have missed the migration step:

SqlException: Cannot open database "RazorPagesMovieContext-GUID" requested by the login. The login failed.
Login failed for user 'User-name'.

Test the Create link.

https://localhost:5001/Movies/Create



Seed the database

Create a new class named `SeedData` in the *Models* folder with the following code:

```
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.DependencyInjection;
```

```
using RazorPagesMovie.Data;
using System;
using System.Linq;

namespace RazorPagesMovie.Models
{
    public static class SeedData
    {
        public static void Initialize(IServiceProvider serviceProvider)
        {
            using (var context = new RazorPagesMovieContext(
                serviceProvider.GetRequiredService<
                    DbContextOptions<RazorPagesMovieContext>>()))
            {
                // Look for any movies.
                if (context.Movie.Any())
                {
                    return;   // DB has been seeded
                }

                context.Movie.AddRange(
                    new Movie
                    {
                        Title = "When Harry Met Sally",
                        ReleaseDate = DateTime.Parse("1989-2-12"),
                        Genre = "Romantic Comedy",
                        Price = 7.99M
                    },

                    new Movie
                    {
                        Title = "Ghostbusters ",
                        ReleaseDate = DateTime.Parse("1984-3-13"),
                        Genre = "Comedy",
                        Price = 8.99M
                    },

                    new Movie
                    {
                        Title = "Ghostbusters 2",
                        ReleaseDate = DateTime.Parse("1986-2-23"),
                        Genre = "Comedy",
                        Price = 9.99M
                    },

                    new Movie
                    {
                        Title = "Rio Bravo",
                        ReleaseDate = DateTime.Parse("1959-4-15"),
                        Genre = "Western",
                        Price = 3.99M
                    }
                );
                context.SaveChanges();
```

```
        }
    }
  }
}
```

Add the seed (benih) initializer (pemula)

File name: Program.cs

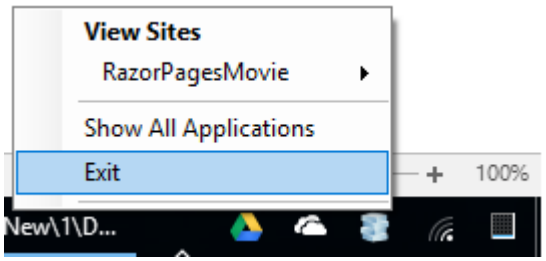
```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using RazorPagesMovie.Models;
using System;

namespace RazorPagesMovie
{
    public class Program
    {
        public static void Main(string[] args)
        {
            var host = CreateHostBuilder(args).Build();

            using (var scope = host.Services.CreateScope())
            {
                var services = scope.ServiceProvider;

                try
                {
                    SeedData.Initialize(services);
                }
                catch (Exception ex)
                {
                    var logger = services.GetRequiredService<ILogger<Program>>();
                    logger.LogError(ex, "An error occurred seeding the DB.");
                }
            }

            host.Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
```

```
                webBuilder.UseStartup<Startup>();
            });
    }
}
```

Test the app

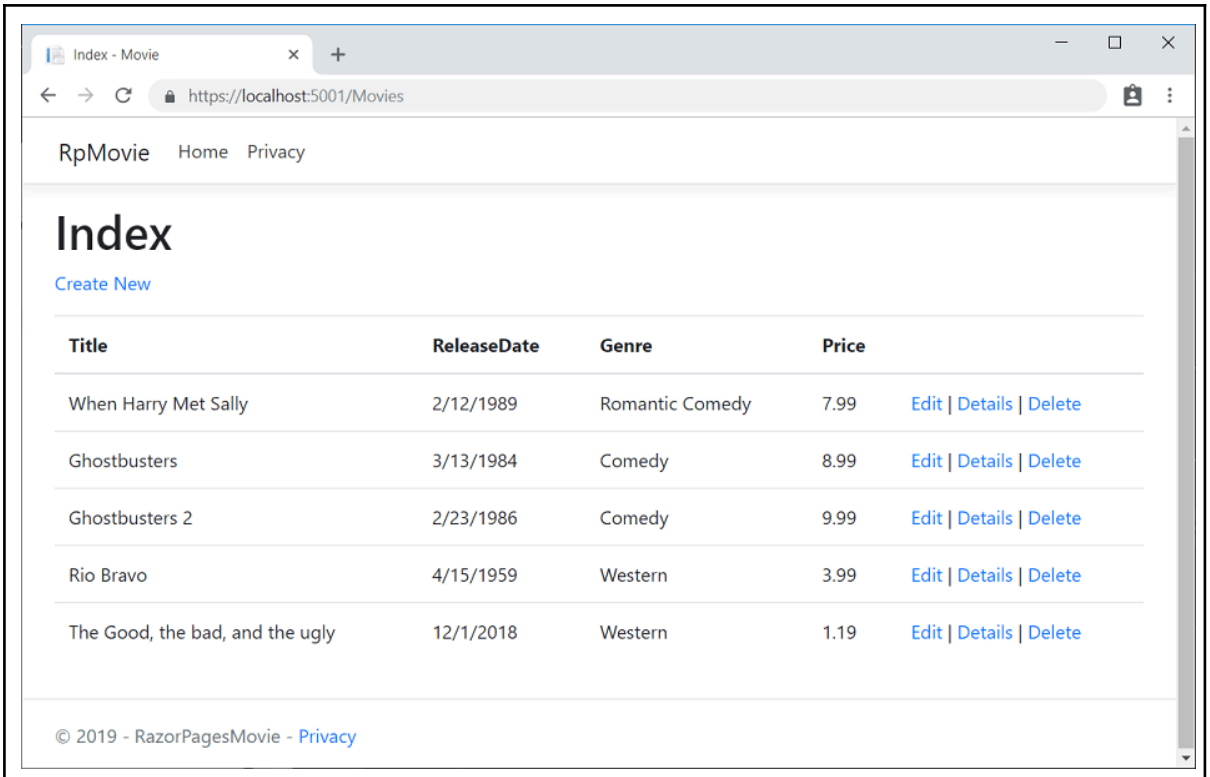
Delete all the records in the database. Use the delete links in the browser

- Force the app to initialize by calling the methods in the `Startup` class, so the seed method runs. To force initialization, IIS Express must be stopped and restarted. Stop and restart IIS with any of the following approaches:
    - Right-click the IIS Express system tray icon in the notification area and tap Exit or Stop Site:



        - If the app is running in non-debug mode, press F5 to run in debug mode.
        - If the app in debug mode, stop the debugger and press F5.

The app shows the seeded data:



Download Complete Solution:

Without Seed:
RazorPagesMovie-0.zip

With Seed:
RazorPagesMovie-1.zip

Database Connection Information

The database connection information is kept in the file appsettings.json

The default connection is localdb.

To change to an sql server, refer the following example:

```
"Server=smacomdb.mssql.somee.com;Database=smacomdb;Trusted_Connection=False;User Id=smarcompu_SQLLogin_1;Password="
```

File name: appsettings.json

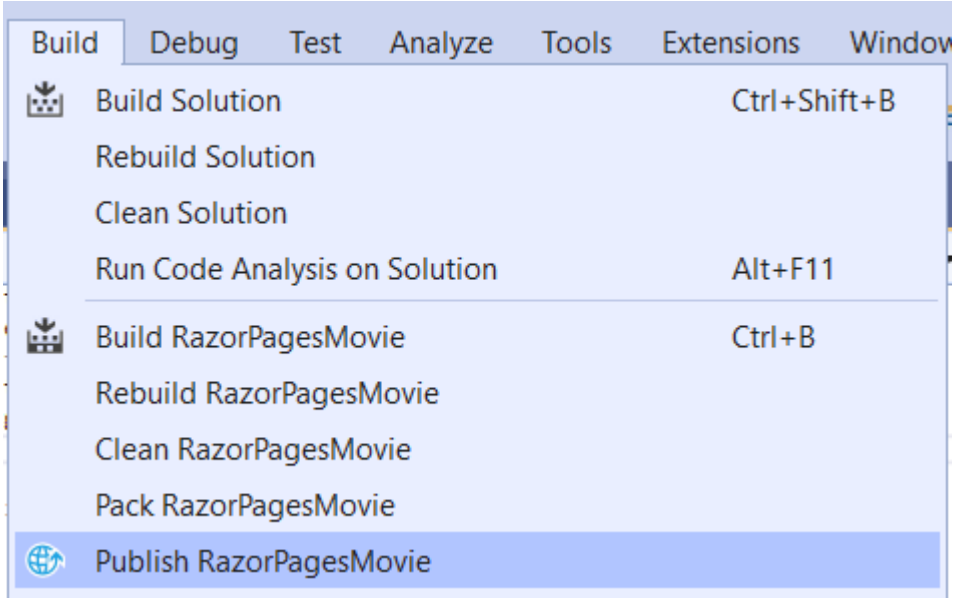
```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "RazorPagesMovieContext":
"Server=(localdb)\\mssqllocaldb;Database=RazorPagesMovieContext-6d
ed67ec-fcd5-48c6-b4bb-fd8b25ce415f;Trusted_Connection=True;Multipl
eActiveResultSets=true"
  }
}
```

zpx6ci6uu6

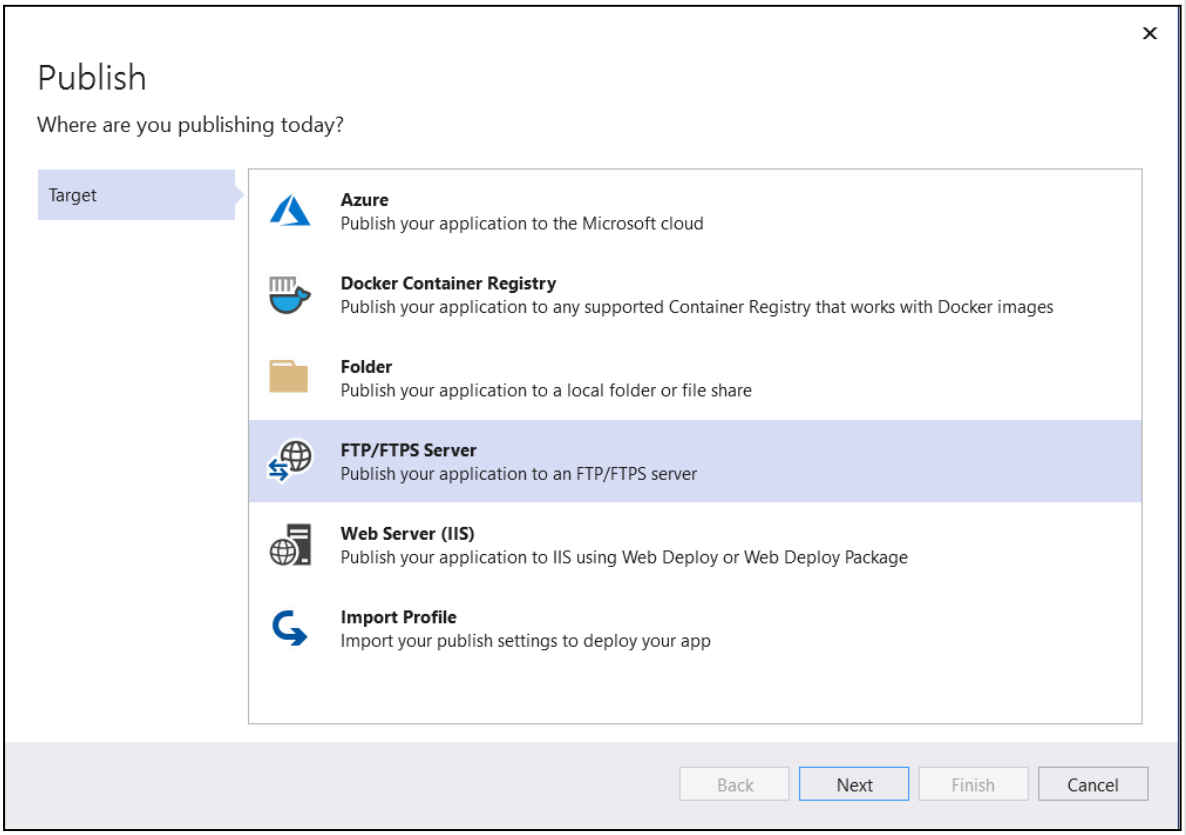
Publishing To Server Via FTP

⚠ Before publishing, change the localdb connection to Sql Server connection.

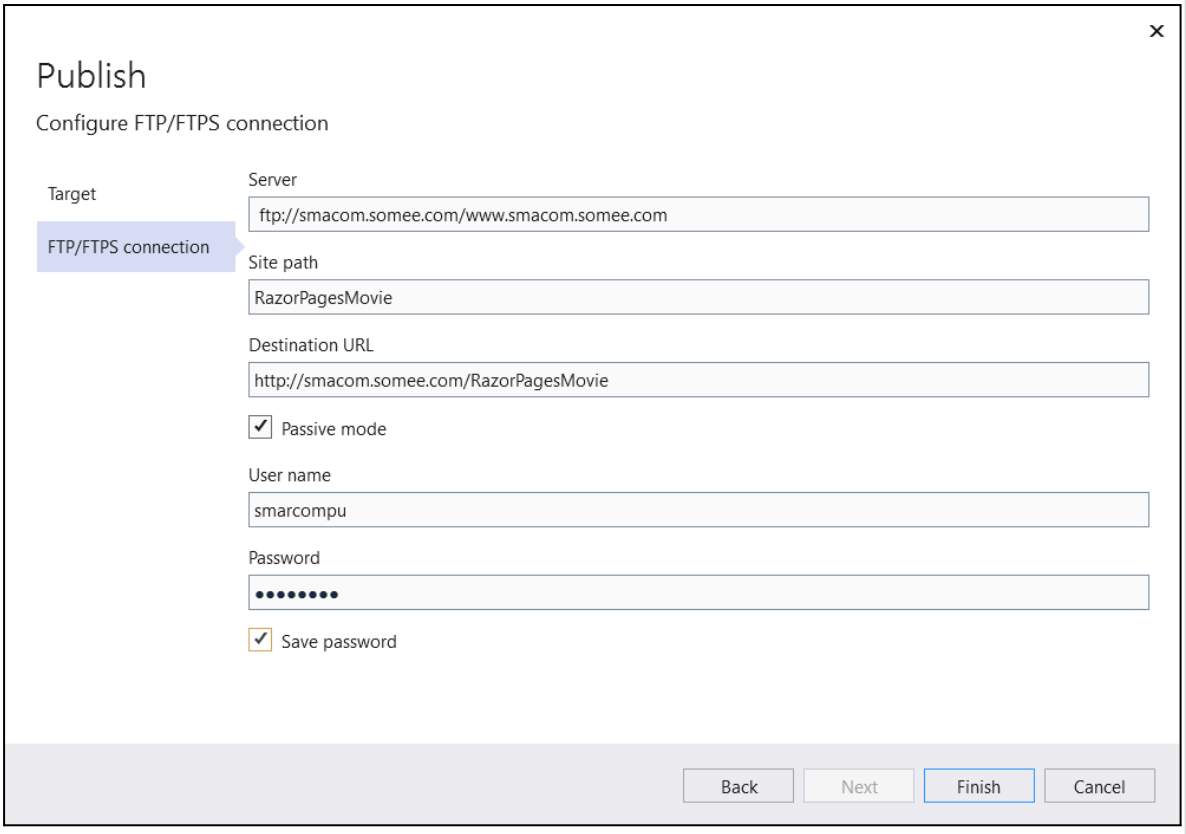
Select menu Build/Publish RazorPagesMovie



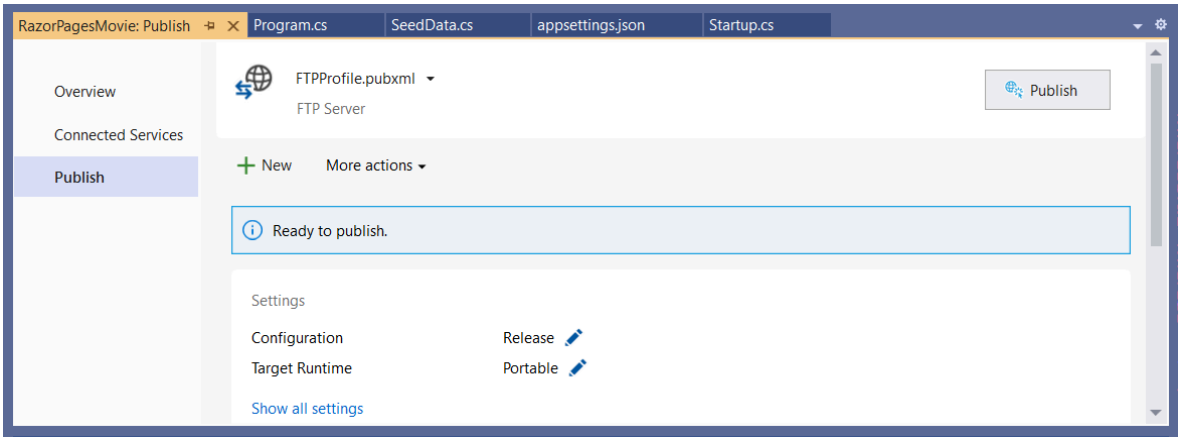
Select FTP



Enter server connection details.



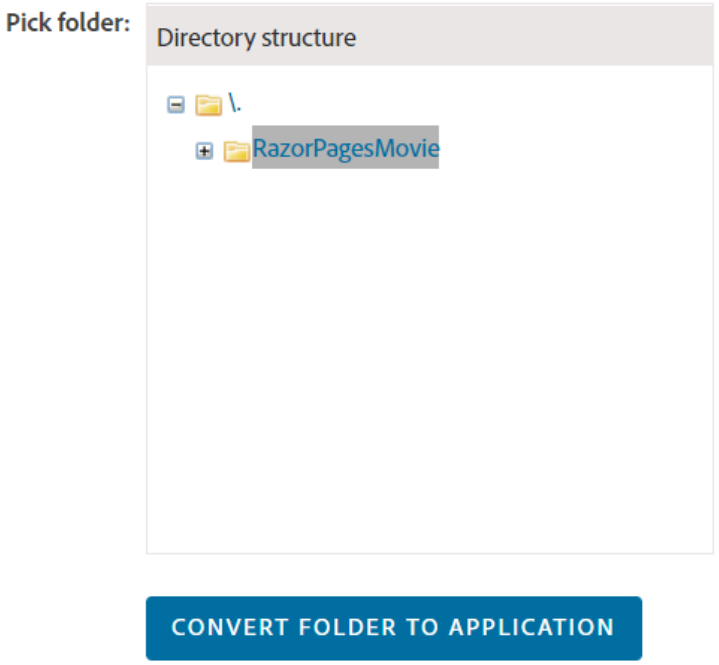
Click Publish when you are ready



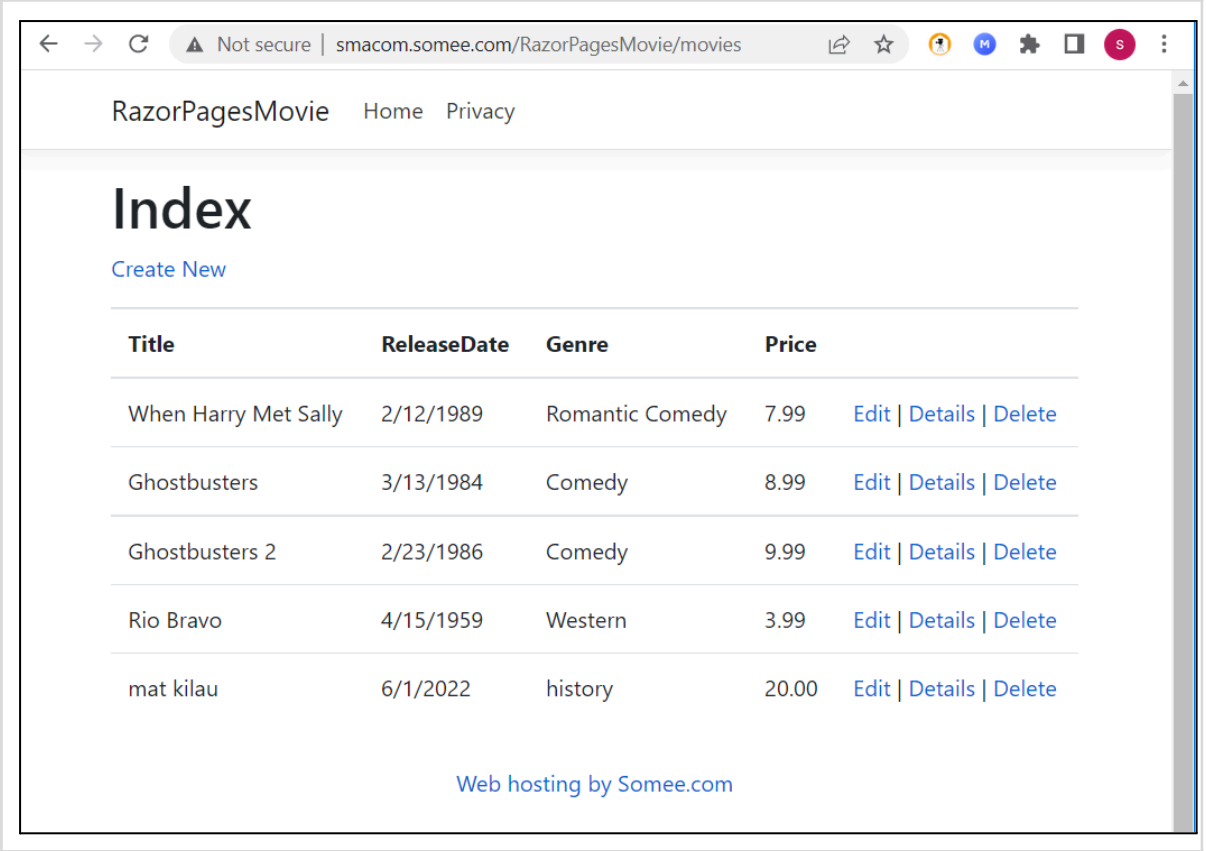
Build started...
1>------ Build started: Project: RazorPagesMovie, Configuration: Release Any CPU ------
1>RazorPagesMovie -> C:\Users\razzi\source\repos\RazorPagesMovie\bin\Release\netcoreapp3.1\RazorPagesMovie.dll
1>RazorPagesMovie ->
C:\Users\razzi\source\repos\RazorPagesMovie\bin\Release\netcoreapp3.1\RazorPagesMovie.Views.dll
2>------ Publish started: Project: RazorPagesMovie, Configuration: Release Any CPU ------
Connecting to ftp://smacom.somee.com/www.smacom.somee.com/RazorPagesMovie...
RazorPagesMovie -> C:\Users\razzi\source\repos\RazorPagesMovie\bin\Release\netcoreapp3.1\RazorPagesMovie.dll
RazorPagesMovie ->
C:\Users\razzi\source\repos\RazorPagesMovie\bin\Release\netcoreapp3.1\RazorPagesMovie.Views.dll
RazorPagesMovie -> C:\Users\razzi\source\repos\RazorPagesMovie\obj\Release\netcoreapp3.1\PubTmp\Out\
Publishing folder /...
Publishing folder cs...
Publishing folder de...
Publishing folder es...
Publishing folder fr...
Publishing folder it...
Publishing folder ja...
Publishing folder ko...
Publishing folder pl...
Publishing folder pt-BR...
Publishing folder ru...
Publishing folder runtimes...
Publishing folder runtimes/unix...
Publishing folder runtimes/unix/lib...
Publishing folder runtimes/unix/lib/netcoreapp2.0...
Publishing folder runtimes/unix/lib/netcoreapp2.1...
Publishing folder runtimes/win...
Publishing folder runtimes/win/lib...
Publishing folder runtimes/win/lib/netcoreapp2.0...
Publishing folder runtimes/win/lib/netcoreapp2.1...
Publishing folder runtimes/win/lib/netstandard2.0...
Publishing folder runtimes/win-arm64...
Publishing folder runtimes/win-arm64/native...
Publishing folder runtimes/win-x64...
Publishing folder runtimes/win-x64/native...
Publishing folder runtimes/win-x86...
Publishing folder runtimes/win-x86/native...
Publishing folder tr...
Publishing folder wwwroot...
Publishing folder wwwroot/css...
Publishing folder wwwroot/js...
Publishing folder wwwroot/lib...
Publishing folder wwwroot/lib/bootstrap...
Publishing folder wwwroot/lib/bootstrap/dist...
Publishing folder wwwroot/lib/bootstrap/dist/css...
Publishing folder wwwroot/lib/bootstrap/dist/js...
Publishing folder wwwroot/lib/jquery...
Publishing folder wwwroot/lib/jquery/dist...
Publishing folder wwwroot/lib/jquery-validation...

Publishing folder wwwroot/lib/jquery-validation/dist...
Publishing folder wwwroot/lib/jquery-validation-unobtrusive...
Publishing folder zh-Hans...
Publishing folder zh-Hant...
Web App was published successfully ftp://smacom.somee.com/www.smacom.somee.com/RazorPagesMovie
Web App was published successfully http://smacom.somee.com/RazorPagesMovie
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========

Set the folder for the published project as an IIS application



Test browsing the published project.



SQL Server Connection In Console App (C#)	SQL Server Connection In Console App (VB)
CLI: dotnet new console -lang c# -o csdbcon1	CLI: dotnet new console -lang vb -o vbdbcon1
CLI: dotnet add package Microsoft.Data.SqlClient --version 5.0.0	CLI: dotnet add package Microsoft.Data.SqlClient --version 5.0.0

using System;
using Microsoft.Data.SqlClient;
 	
namespace csdbcon1 {
class Program {
static void Main(string[] args) {
try {
SqlConnectionStringBuilder builder = new SqlConnectionStringBuilder();
	Imports System
	Imports Microsoft.Data.SqlClient
	
	Module Program
	Sub Main(args As String())
	Try
	
	Dim builder As New Microsoft.Data.SqlClient.SqlConnectionStringBuilder()

```
        builder.DataSource =
"smacomdb.mssql.somee.com";
        builder.UserID =
"smarcompu_SQLLogin_1";
        builder.Password = "";
        builder.InitialCatalog = "smacomdb";
        builder.TrustServerCertificate = true;

        using(SqlConnection connection = new
SqlConnection(builder.ConnectionString)) {
          Console.WriteLine("\nQuery data
example:");

Console.WriteLine("==================
======================\n");

          connection.Open();

          String sql = "SELECT name,
collation_name FROM sys.databases";

          using(SqlCommand command = new
SqlCommand(sql, connection)) {
            using(SqlDataReader reader =
command.ExecuteReader()) {
              if (reader.HasRows) {

                while (reader.Read()) {
                  /*Console.WriteLine("{0}",
reader.GetString(0));*/

Console.WriteLine($"{reader.GetString(0)}")
;
                }
              }
            }
          }
        }
      } catch (SqlException e) {
        Console.WriteLine(e.ToString());
      }
      Console.WriteLine("\nDone. Press
enter.");
      Console.ReadLine();
    }
  }
}
```

```
    builder.DataSource =
"smacomdb.mssql.somee.com"
    builder.UserID =
"smarcompu_SQLLogin_1"
    builder.Password = ""
    builder.InitialCatalog = "smacomdb"
    builder.TrustServerCertificate = True

    Using connection As New
SqlConnection(builder.ConnectionString)

    Console.WriteLine(vbCrLf & "Query data
example:")

Console.WriteLine("==================
======================" & vbCrLf)

    connection.Open()

    Dim sql As String = "SELECT name,
collation_name FROM sys.databases"

    Using command As New
SqlCommand(sql,connection)

    Using reader As SqlDataReader =
command.ExecuteReader()
    If reader.HasRows Then
        Do Whilereader.Read
            'Console.WriteLine("{0}",
reader.GetString(0))

Console.WriteLine($"{reader.GetString(0)}")
        Loop
    End If
End Using
End Using
connection.Close()

End Using
Catch ex As Exception

End Try

Console.WriteLine(vbCrLf & "Done. Press
enter.")
Console.ReadLine()

End Sub
End Module
```

CLI: cd csdbcon1 CLI: dotnet build CLI: dotnet run	CLI: cd vbdbcon1 CLI: dotnet build CLI: dotnet run

zpx6ci6uu6

FURTHER READINGS

Razor Tutorial
https://docs.microsoft.com/en-us/aspnet/core/tutorials/razor-pages/?view=aspnetcore-3.1

MVC Tutorial
https://docs.microsoft.com/en-us/aspnet/core/tutorials/first-mvc-app/start-mvc?view=aspnetcore-3.1&tabs=visual-studio