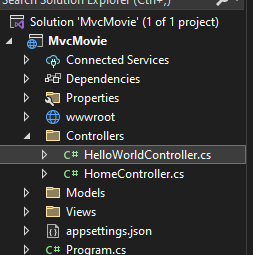
WORKING WITH ASP.NET CORE APPLICATIONS

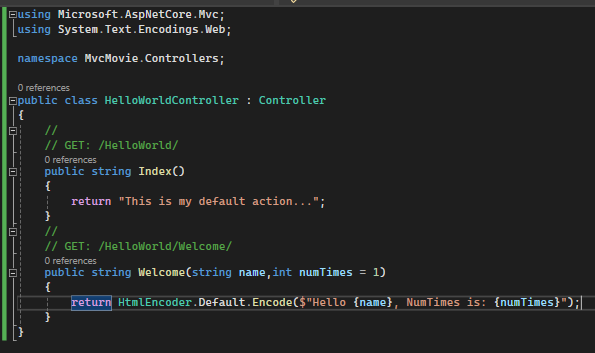
1. Creating a controller

Here, I created a controller named HelloWorldController.cs on the controller folder of the root.



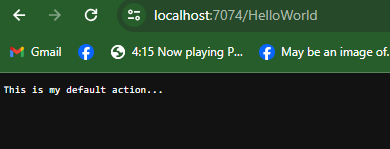
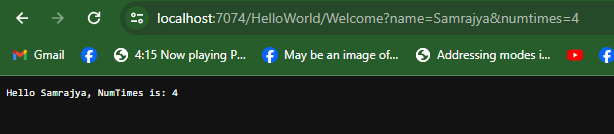
1. Created Actions

I learnt that Actions are all the public methods inside a controller class



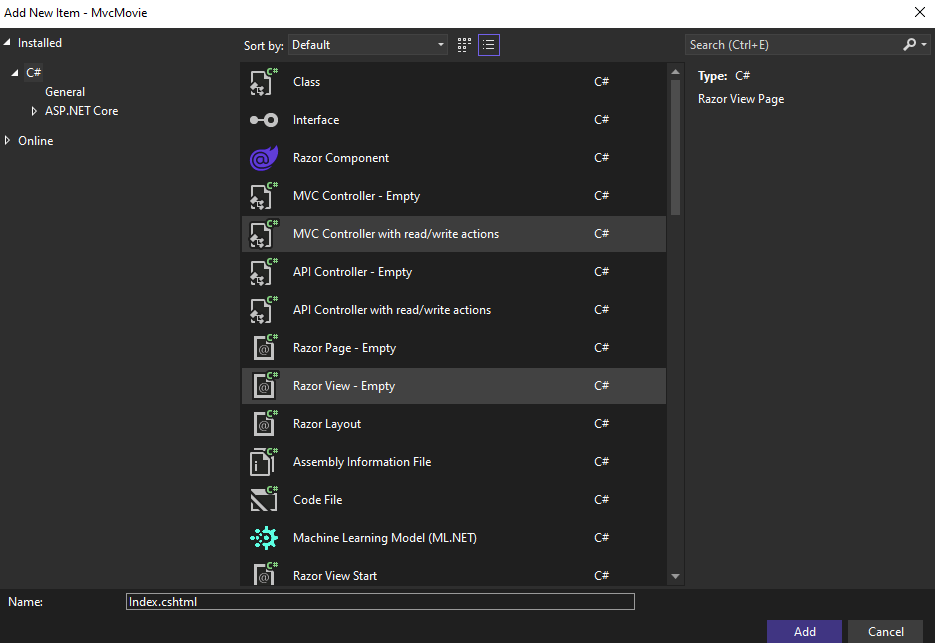
Here are two actions I created one is the default one and another is Welcome action.

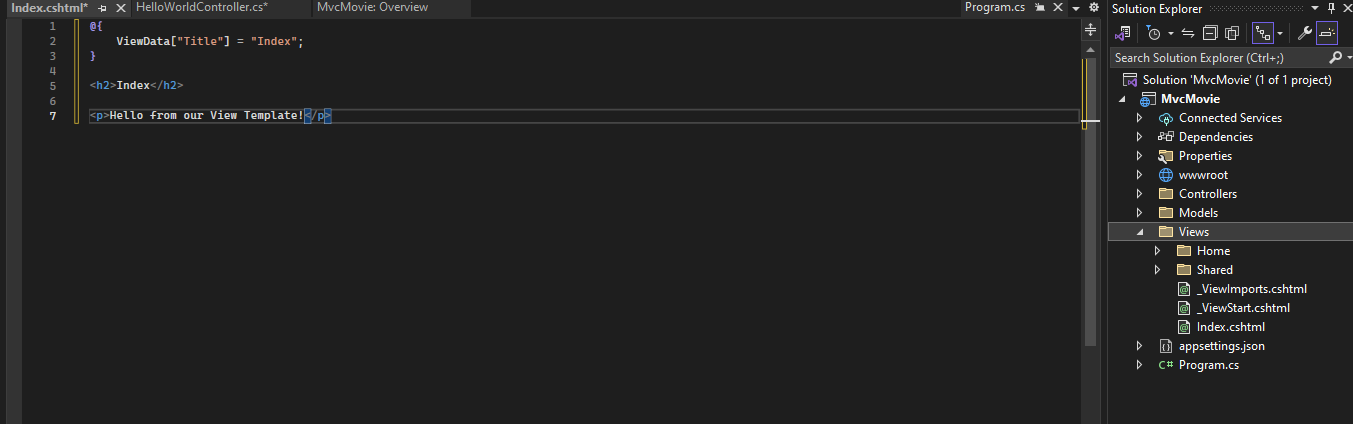
1. Default Action 2) Welcome Action



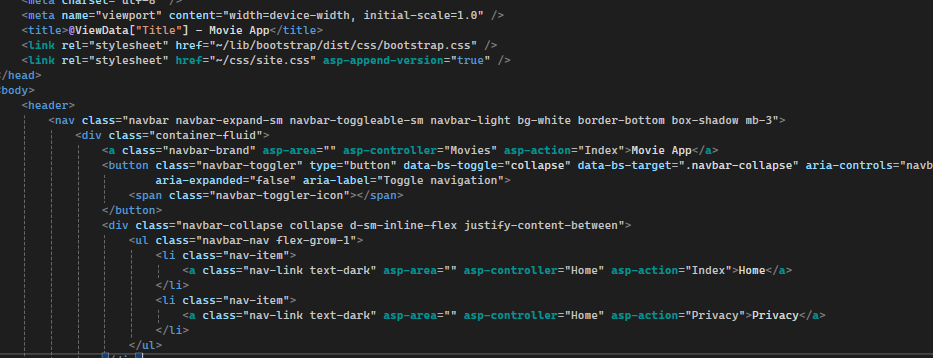
As you can see my welcome method accepts two parameters that is name as string and numtimes as the number of times it was called.

1. Creating a View

I created a view called index.cshtml to load the html markup

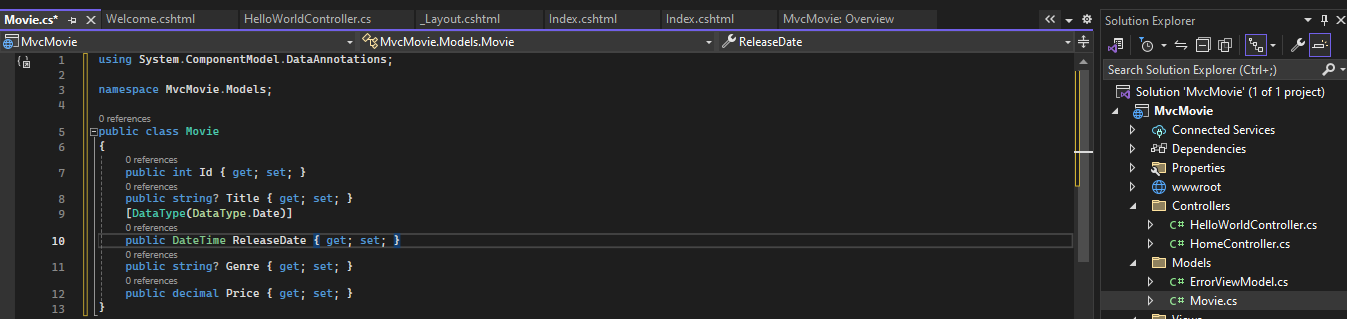


1. Tag Helpers in our layout.cshtml file



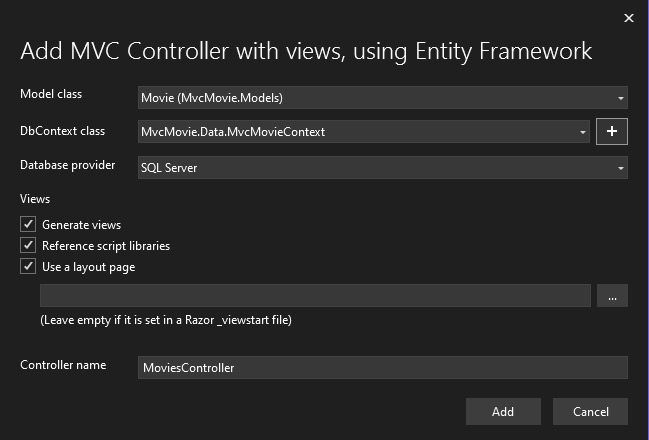
Tag helpers used are asp-area, asp-controller, and asp-action-asp-append-version

1. Adding a model with entity framework to work with database



The Movie class contains an Id field, which is required by the database for the primary key. The DataType attribute on ReleaseDate specifies the type of the data (Date). The DataType attribute on ReleaseDate specifies the type of the data (Date)

1. New Scaffold Item Added

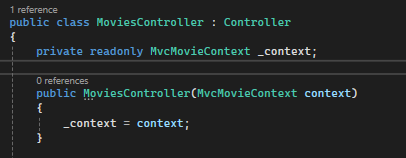


EF core migrations were used to create the database. In the package manager console (pmc) following commands were entered

Add-Migration InitialCreate: Generates a Migrations/ {timestamp} \_InitialCreate.cs migration file. The InitialCreate argument is the migration name. Because this is the first migration, the generated class contains code to create the database schema. The database schema is based on the model specified in the MvcMovieContext class.

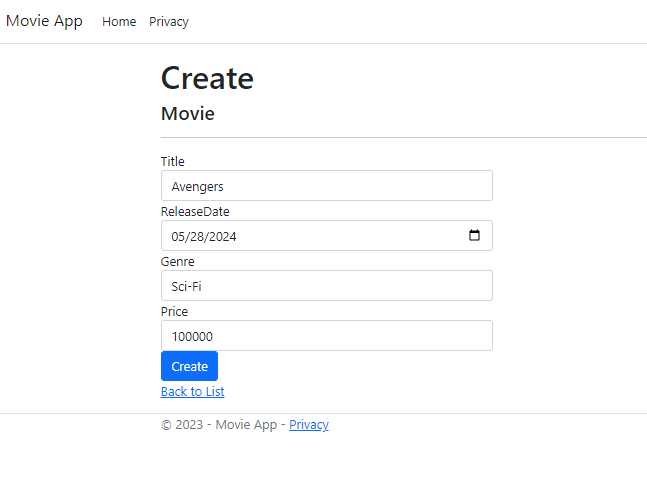
Update-Database: Updates the database to the latest migration, which the previous command created.

1. Dependency injection in the controller

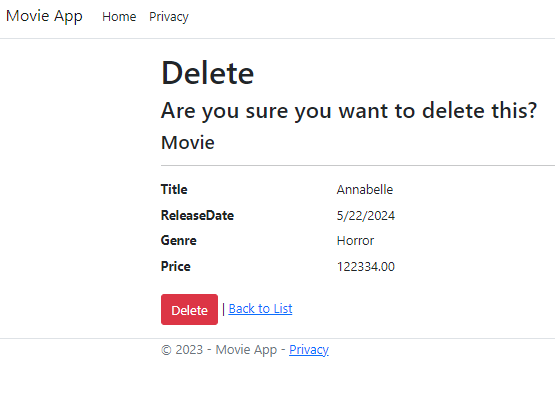


The constructor uses Dependency Injection to inject the database context (MvcMovieContext) into the controller. The database context is used in each of the CRUD methods in the controller.

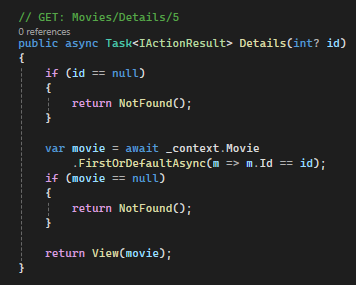
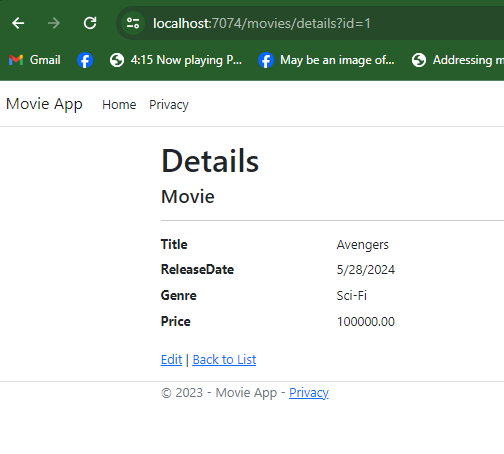
1. Creating a new data

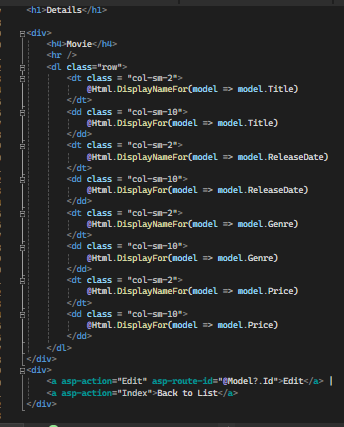


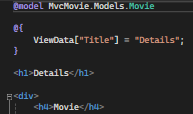
1. Deleting data



1. Getting movies detail with the help of id

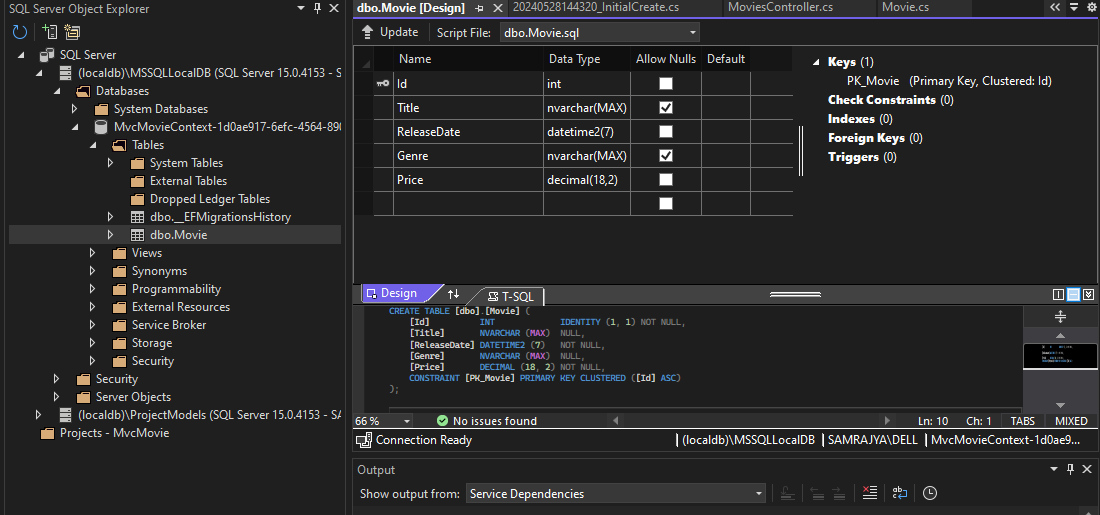






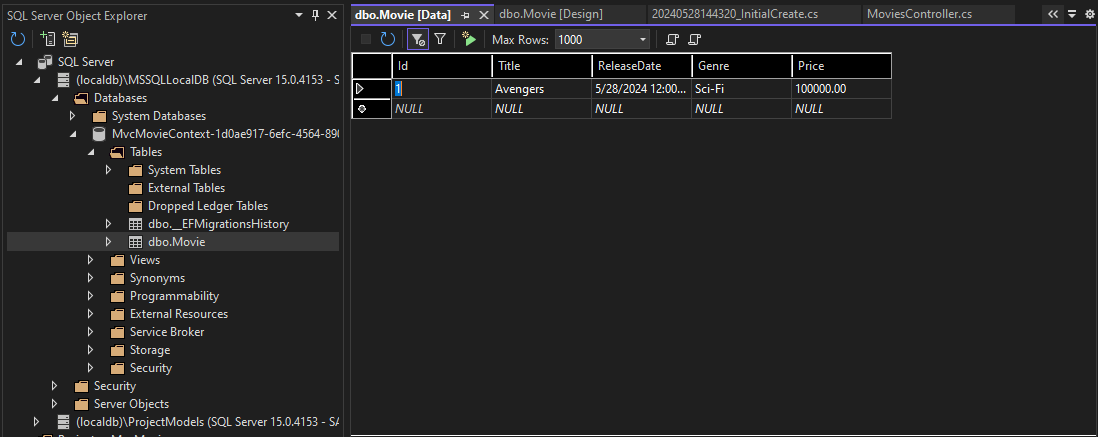
Details.cshtml file that displays the data of the movie. The @model statement at the top of the view file specifies the type of object that the view expects. This @model directive allows access to the movie that the controller passed to the view.

1. Design View of dbo.movie table

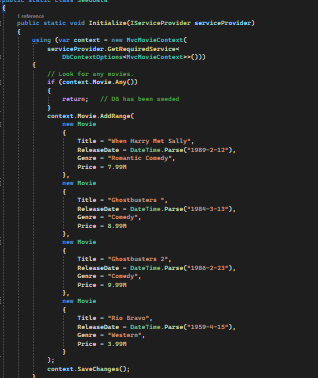


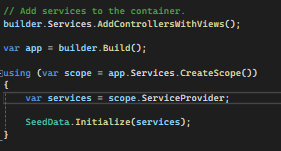
ID is the default primary key made by the Entity Framework

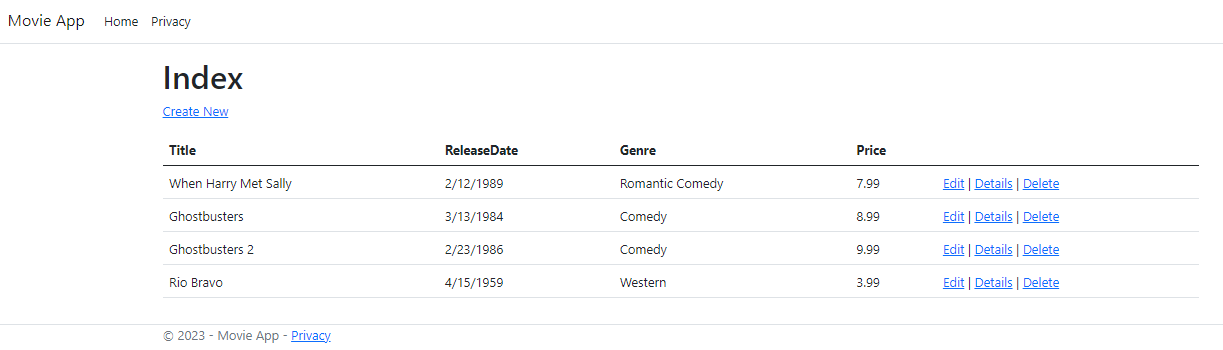
1. Data view of the table



1. Seeding the database and adding the seed initializer in program.cs



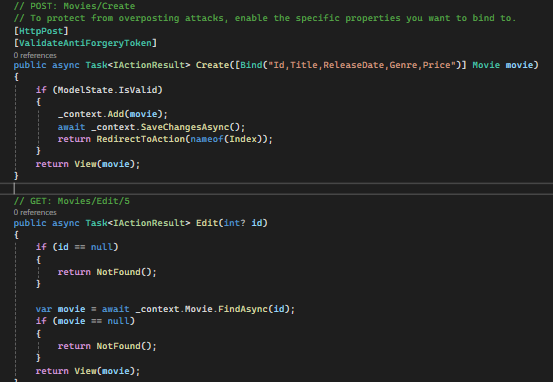




1. Adding data annotation “Display”

The Display attribute specifies what to display for the name of a field (in this case "Release Date" instead of "ReleaseDate").

1. Using Bind to be safe against over-posting attack

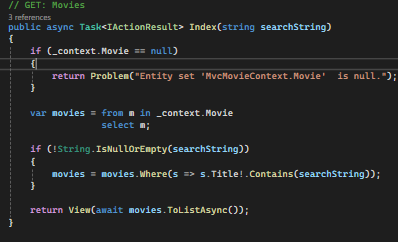
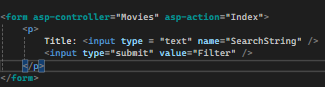


The [Bind] attribute is one way to protect against over-posting. We should only include properties in the [Bind] attribute that we want to change. Mass assignment, also known as over-posting, is an attack used on websites that involve some sort of model-binding to a request. It is used to set values on the server that a developer did not expect to be set.

The [ValidateAntiForgeryToken] attribute validates the hidden XSRF token generated by the anti-forgery token generator in the Form Tag Helper. 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.

1. Adding search button to our website

We used a LINQ query to select the movie, the query is only defined at this point it hasn’t been run against the database. Notice that there is a lambda expression used to create a function



1. Adding genre also as a search

