# Practice 4-1 Build the Movie List app – Part A

The purpose of chapter 4 has been to get you started with learning how to develop multi-page; data driven MVC web apps. This exercise guides you through the development of the Movie List app that is presented in this chapter. This will give you a chance to generate a database from entity classes. In other words, you will use Entity Framework to generate your SQL Database. Upon completion of this exercise, you should be able to develop apps of your own that use multiple pages to work with a database.

## Set up the file structure

1. Create a new ASP.NET Core Web Application in your INFS 4950 folder with a project name of Ch04MovieList and a solution name of Ch04MovieList. Base this app on the Web Application (Model-View-Controller) template.

2. Remove all unnecessary files such as Models/ ErrorViewModel.cs, Views/Home/Privacy.cshtml, and so on.

3. Using figure 4-3 as a guide, install the EF Core SqlServer and EF Core Tools NuGet packages.

4. Using figure 3-2 as a guide, use LibMan to install version 4.3.1 of the Bootstrap CSS library. In the Solution Explorer, expand the wwwroot/lib folder and make a note of the path to the bootstrap.min.css file.

## Modify existing files

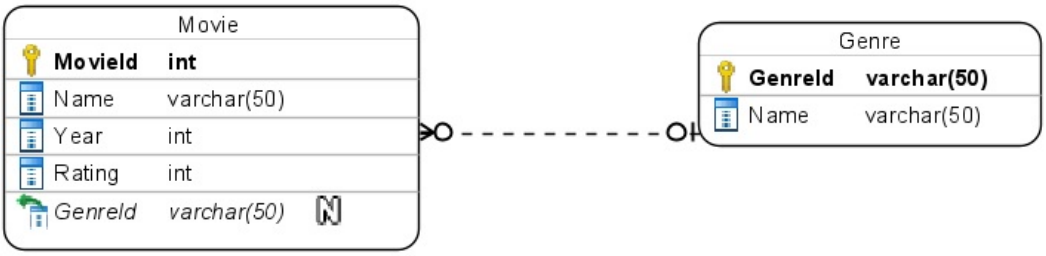
5. Modify the Views/Shared/_Layout.cshtml file so it includes the code below:

```
<!DOCTYPE html>
<html>
<head>
        <meta name="viewport" content="width=device-width" />
        <title>@ViewBag.Title</title>
        <link rel="stylesheet" type="text/css"
                href="~/lib/bootstrap/css/bootstrap.min.css">
</head>
<body>
        <div class="container">
                <header class="jumbotron">
                        <h1>My Movies</h1>
                </header>
        @RenderBody()
        </div>
</body>
</html>
```

Make sure the href attribute specifies the correct path to the bootstrap.min.css file. Another option you have would be to use a CDN (such as http://cdnjs.com) instead of installing the client-side libraries in your app.

6. Remove all action methods from the HomeController except for the Index() method.

## Code the classes for the model and the DB context




# Movies Database

**Movies Table**

| MovieId | Name | Year | Rating | GenreId |
|---|---|---|---|---|
| 1 | Casablanca | 1942 | 5 | D |
| 2 | Wonder Woman | 2017 | 3 | A |
| 3 | Moonstruck | 1988 | 4 | R |

**Genre Table**

| GenreId | Name |
|---|---|
| A | Action |
| C | Comedy |
| D | Drama |
| H | Horror |
| M | Musical |
| R | RomCom |
| S | SciFi |

7. Add a Genre class to the Models folder. Then, modify this class so it contains the code shown in figure 4-14. Add a Movie class to the Models folder and edit it so it contains the code shown in figure 4-4. Don't forget to add the using directive for data annotations **using System.ComponentModel.DataAnnotations;**.

```
namespace Ch04MovieList.Models
{
    public class Genre
    {
        public string GenreId{get; set;}
        public string Name { get; set; }

    }
}
```

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

namespace Ch04MovieList.Models
{
    public class Movie
    {
        //EF Core will configure the database to generate the value for the Id

        public int MovieId { get; set; }

        [Required(ErrorMessage = "Please enter a name.")]
        public string Name { get; set; }

        [Required(ErrorMessage="Please enter a year.")]
        [Range(1889, 2999, ErrorMessage ="Year must be after 1889")]
        public int? Year { get; set; }

        [Required(ErrorMessage = "PLease enter a rating.")]
        [Range(1, 5, ErrorMessage = "Rating must be between 1 and 5.")]
        public int? Rating { get; set; }

        //Specify GenreId as a foreign key property
        [Required(ErrorMessage ="Please enter a genre")]
        public string GenreId { get; set; }

        // Add a Genre property to the Movie class with the entity class Genre as its data type
        // to relate Movies and Genres
        public Genre Genre { get; set;}

    }
}
```

8. Add a MovieContext class to the Models folder and edit it so it contains the code shown in figures 4-4 and 4-15. Don't forget to add the using directive for the EF Core namespace.

9. Modify the MovieContext class to contain the code for the OnModelCreating() method shown in figure 4-5 and 4-15. Notice the line base.OnModelCreating(modelBuilder); is included in the code below and in figure 4-15 of your textbook. However, it is not shown in figure 4-5 of your textbook. This line of code seems to be included in some programming manuals and not in others and is optional, if you are interested additional research on this line of code please see the following posts: https://stackoverflow.com/questions/39576176/is-base-onmodelcreatingmodelbuilder-necessary and https://entityframework.net/knowledge-base/39576176/is-base-onmodelcreating-modelbuilder--necessary-

```
using Microsoft.EntityFrameworkCore;

namespace Ch04MovieList.Models
{
    public class MovieContext : DbContext
    {
        public MovieContext(DbContextOptions<MovieContext> options)
                    : base(options)
        { }
        public DbSet<Movie> Movies { get; set; }

        public DbSet<Genre> Genres { get; set; }
```

```
        protected override void OnModelCreating(ModelBuilder modelBuilder)
        {

            base.OnModelCreating(modelBuilder);

            modelBuilder.Entity<Genre>().HasData(
                new Genre { GenreId = "A", Name = "Action" },
                new Genre { GenreId = "C", Name = "Comedy" },
                new Genre { GenreId = "D", Name = "Drama" },
                new Genre { GenreId = "H", Name= "Horror" },
                new Genre { GenreId = "M", Name = "Musical" },
                new Genre { GenreId = "R", Name = "RomCom" },
                new Genre { GenreId = "S", Name = "SciFi" }
            );


            modelBuilder.Entity<Movie>().HasData(
              new Movie { MovieId = 1, Name = "Casablanca", Year = 1942, Rating = 5, GenreId = "D"},
              new Movie { MovieId = 2, Name = "Wonder Woman", Year = 2017, Rating = 3, GenreId = "A"},
              new Movie { MovieId = 3, Name = "Moonstruck", Year = 1988, Rating = 4, GenreId = "R" }
            );
        }
    }
}
```

10. Add the connection string to appsettings.json as shown in figure 4-6. However, to avoid database conflicts (if you ran the Movies app in Chapter 3, then a database named Movies already exists), either specify a name of MoviesExercise for the database by editing the Database parameter like this: Database=MoviesExercise, or you may delete the existing Movies database using the SQL Server Object Explorer. Either option is good, but be careful if you delete a database that you delete the correct one. **Make sure to enter the entire connection string on one line and to add a comma to the end of the previous line.**

```
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "MovieContext": "Server=(localdb)\\mssqllocaldb;Database=Movies;
                     Trusted_Connection=True;MultipleActiveResultSets=true;"
```

11. Modify the Startup.cs file so it includes the code shown in figure 4-6. This enables dependency injection for DbContext objects. At the top of the file, make sure to include all of the necessary using directives, including the using directive for the Models namespace and EntityFrameworkCore.

```
…
using Microsoft.EntityFrameworkCore;
using Ch04MovieList.Models;

namespace Ch04MovieList
{
    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)
        {
              …

            //Add code that enables dependency injection for DbContext objects
            services.AddDbContext<MovieContext>(options =>
                options.UseSqlServer(
                Configuration.GetConnectionString("MovieContext")));

        }
        …
    }
}
```

12. The entity framework coding in the previous steps used coding by convention. Another option is to use Entity Framework – Fluent API. See the following for references on using Fluent API:

- https://www.tutorialspoint.com/entity_framework/entity_framework_fluent_api.htm
- https://www.entityframeworktutorial.net/efcore/fluent-api-in-entity-framework-core.aspx
- https://www.entityframeworktutorial.net/efcore/configure-one-to-many-relationship-using-fluent-api-in-ef-core.aspx
- https://www.learnentityframeworkcore.com/configuration/fluent-api/haskey-method

Below are additional references on Entity Framework Core:

- https://docs.microsoft.com/en-us/ef/core/modeling/relationships?tabs=fluent-api%2Cfluent-api-simple-key%2Csimple-key
- https://www.learnentityframeworkcore.com/configuration/one-to-many-relationship-configuration
- https://www.learnentityframeworkcore.com/conventions/one-to-many-relationship

## Create the Movies database

13. Using figure 4-7 as a guide, open the Package Manager Console. At the command prompt, enter the "Add-Migration Initial" command. This should add a Migrations folder and migration files to the Solution Explorer. If you get an error, troubleshoot the problem.

14. At the Package Manager Console command prompt, enter the "Update-Database" command. This should create the database. If you get an error, troubleshoot the problem.

15. View your database. To do that, display the SQL Server Object Explorer as described in figure 4-7. Then, expand the nodes until you can see the MoviesExercise database that you just created.

16. View the seed data. To do that, expand the MoviesExercise node and the Tables node. Then, right-click on the dbo.Movies table and select View Data. This should show the data that's stored in the Movies table.

17. Instead of running the Migrations at the Package Management Console, run database migrations in the Startup (Note: use the SQL Server Object Explorer to first delete the Movies database).

```
public void Configure(IApplicationBuilder app, MovieContext movieContext)
     {
        app.UseDeveloperExceptionPage();
        app.UseHttpsRedirection();
        app.UseStaticFiles();

        //Run Migrations for the database
        movieContext.Database.Migrate();
```

## Modify the Home controller and its view

18. Modify the Home controller so it contains the code shown in figure 4-10. At the top of the controller, make sure to include using directives for all necessary namespaces including the Models namespace.

```
using Microsoft.AspNetCore.Mvc;
using System.Linq;
using Microsoft.EntityFrameworkCore;
using Ch04MovieList.Models;


namespace Ch04MovieList.Controllers
{
    public class HomeController : Controller
    {
        private MovieContext context { get; set; }
        public HomeController(MovieContext ctx)
        {
            context = ctx;
        }

        public IActionResult Index()
        {

            var query = context.Movies
                .Include(m => m.Genre)
                .OrderBy(m => m.Name);

            var movies = query.ToList();

            return View(movies);
        }

    }
}
```

19. Modify the Home/Index view so it contains the code shown in figure 4-10.

```
@model List<Movie>
@{
    ViewBag.Title = "My Movies";
}
<a href="/">Add a New Movie</a>
<table class="table table-bordered table-striped table-active">
    <thead>
        <tr>
            <th>Name</th>
            <th>Year</th>
            <th>Rating</th>
            <th>Genre</th>
```

```
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var movie in Model)
        {
        <tr>
            <td>@movie.Name</td>
            <td>@movie.Year</td>
            <td>@movie.Rating</td>
            <td>@movie.Genre.Name</td>
            <td>
                <a href="/">Edit</a>
                <a href="/">Delete</a>
            </td>
        </tr>
        }
    </tbody>
</table>
```

20. Try some additional Linq queries in the Index action of the Home controller, such as including all moves whose year is greater than 1950 and rating greater than 3. Another example could be all movies produced between the year 1950 and 2000. Spend some time trying out different queries before returning the Index page to include a listing of all movies sorted in ascending order by name. See the following for additional Linq references:

- https://www.tutorialsteacher.com/linq/linq-standard-query-operators
- https://www.tutorialspoint.com/linq/index.htm

21. From the Start drop-down list, select the name of the app to use Kestrel, not IISExpress. Then, run the app. It should display the list of movies in the default browser. However, clicking the Add, Edit, or Delete links should not work yet.

## Turn in the Movie List app

1. Once you are satisfied with your application and have worked with Linq enough to become more comfortable with using the methods to select data in the database, go ahead and specify a Linq query to include movies with a Year of greater than 1970 and a Rating of greater than 3. Test the application one more time, then close Visual Studio, compress the Ch04MovieList app at the root level, and rename the compressed folder using the following naming convention:

   FirstInitial_LastName_Ch04MovieLIst_PartA

   For example:

   M_Korzaan_Ch04MovieLIst_PartA

2. Upload the renamed, compressed folder to the appropriate Dropbox in D2L.