Exercise

ASP.NET MVC - Views

April 2019

The MusicStore web application (Part 2)

Complete the steps below.

Step 1 – Create a store controller

- Add an empty 'Store' controller.
- Add a Class Library project "MusicStoreCore.Data" in the solution. This project will be our data layer. Add a folder "DomainClasses" to the data layer and add the following domain classes in this folder:

C#

```
namespace MusicStoreCore.Data.DomainClasses
{
    public class Album
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Artist { get; set; }
        public int GenreId { get; set; }
    }
}
```

C#

```
namespace MusicStore.Data.DomainClasses
{
    public class Genre
    {
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

- Add a "StoreControllerTests" class in the "Controllers" folder of the test project.
- Add an 'Index' action method. The 'Index' action should return a view that will display a list of the music genres in our music store.
    - Add a unit test "Index_ShowsListOfMusicGenres" and make it green
        - Mock an IGenreRepository. IGenreRepository will be respossible for retrieving the data.

        - The model of the view should contain a list of all the genres returned by the IGenreRepository.
    - Add a dummy repository for genres

C#

```
namespace MusicStore.Data
{
    public class GenreDummyRepository : IGenreRepository
    {
        private static List<Genre> _genres = new List<Genre> {
            new Genre
            {
                Id = 1,
                Name = "Metal"
            },
            new Genre
            {
                Id = 2,
                Name = "Pop"
            },
            new Genre
            {
                Id = 3,
                Name = "Jazz"
            }
        };

        public IEnumerable<Genre> All()
        {
            return _genres;
        }
    }
}
```

    - Use dependency injection to inject the dummy repository into the StoreController

- Add a 'Browse' action method that takes an "id" parameter (int). The "Browse" action returns a view that displays a list of music albums in a particular genre
    - Add a test "Browse_ShowsAlbumsOfGenre" and make it green
        - Mock an IAlbumRepository. IAlbumRepository will be responsible for retrieving the data
        - The Id of the genre is passed into the "Browse" action
        - The model of the view should contain a list of albums to display

        - The (dynamic) ViewBag property of the controller should have a property “Genre” set to the name of the genre.
    - Add a test “Browse_InvalidGenreId_ReturnsNotFound” and make it green
        - When an invalid Id is passed into the action the resulting ActionResult should be a “NotFoundResult”
    - Add a test “Browse_ValidGenreIdButNoAlbumsForGenre_ShowsEmptyList” and make it green
        - When no albums are found in the repository for a (valid) genre, the model of the view should contain an empty list (it may not be null)
- Add a “Details” action method that takes an “id” parameter (int). The “Details” page displays information about a specific music album. We will not implement it now but in a later stage you should be able to change the properties of the album on this page (post).
    - Add a test “Details_ShowsDetailsOfAlbum” and make it green
        - The IAlbumRepository should have a “GetById” method to retrieve a specific album
        - The model of the view should be the album retrieved from the repository
    - Ad a test “Details_InvalidId_ReturnsNotFound” and make it green
    - Add a dummy repository for albums

C#

```
namespace MusicStore.Data
{
    public class AlbumDummyRepository : IAlbumRepository
    {
        public IEnumerable<Album> GetByGenre(int genreId)
        {
            var albums = new List<Album>();
            var nbrOfAlbums = 3;
            for (int i = 1; i <= nbrOfAlbums; i++)
            {
                albums.Add(new Album {
                        Id = i + (nbrOfAlbums * (genreId - 1)),
                        Artist = "Artist " + i,
                        Title = "Title " + i,
                        GenreId = genreId
                });
            }
            return albums;
        }
```

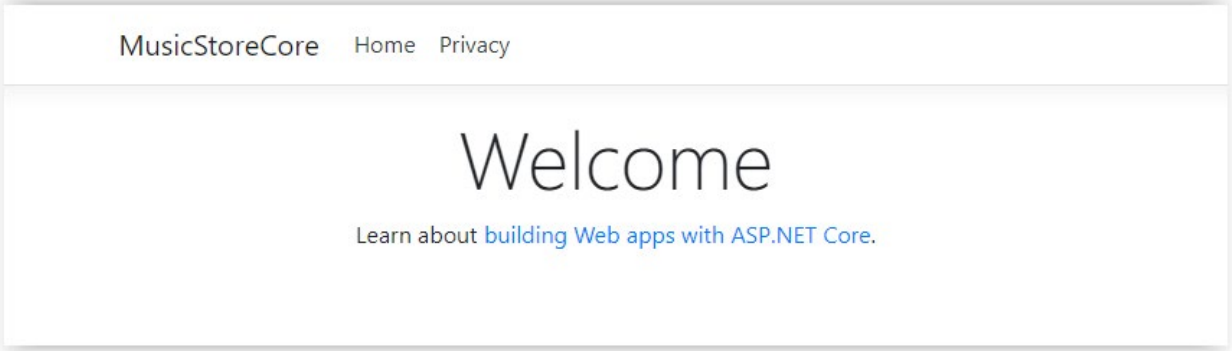
```
        public Album GetById(int id)
        {
            var genreRepo = new GenreDummyRepository();
            var genre = genreRepo.All().First();

            return new Album
            {
                Id = id,
                Artist = "Artist " + id,
                Title = "Title " + id,
                GenreId = genre.Id
            };
        }
    }
}
```

- Use dependency injection to inject the dummy repository into the StoreController

Step 2 – The views

- Rename the test “Index_ReturnsContentContainingControllerNameAndActionName” to “Index_ReturnsDefaultView” and change the test so that it checks if the “Index” action of the “HomeController” returns a ViewResult with no ViewName specified. Make the test green.
- Starting the application should give you the welcome page:

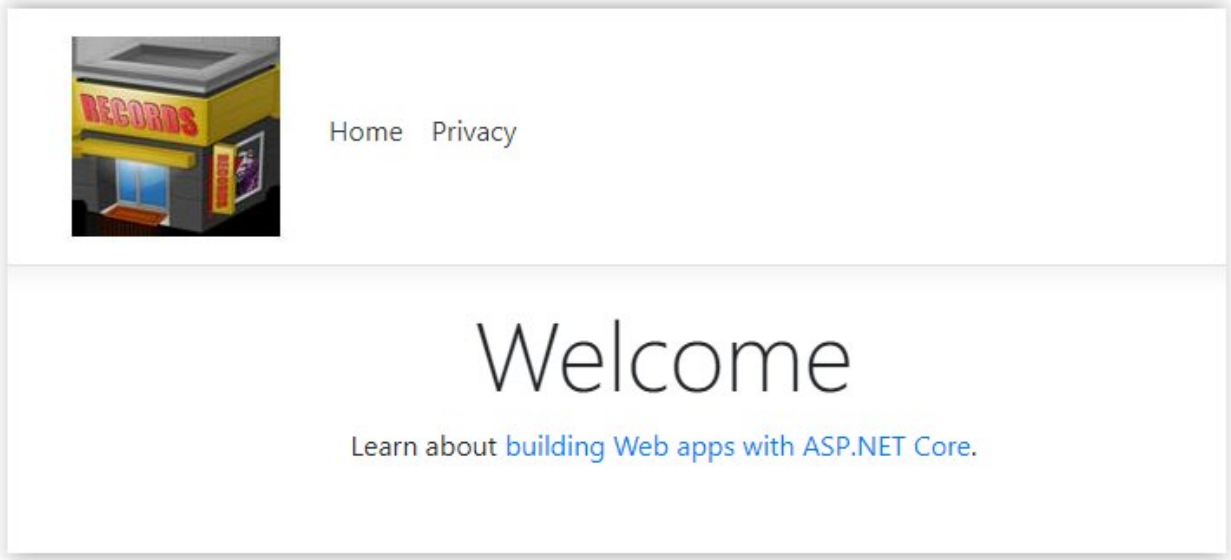


- Alter the general layout of the application
    - Download the logo from BlackBoard and add it to a new “images” folder in the “wwwroot” folder.
    - Copy the “_Layout.cshtml” file to “_LayoutWithLogo.cshtml” in the “Views/Shared” folder
    - Add the logo to “_LayoutWithLogo”.

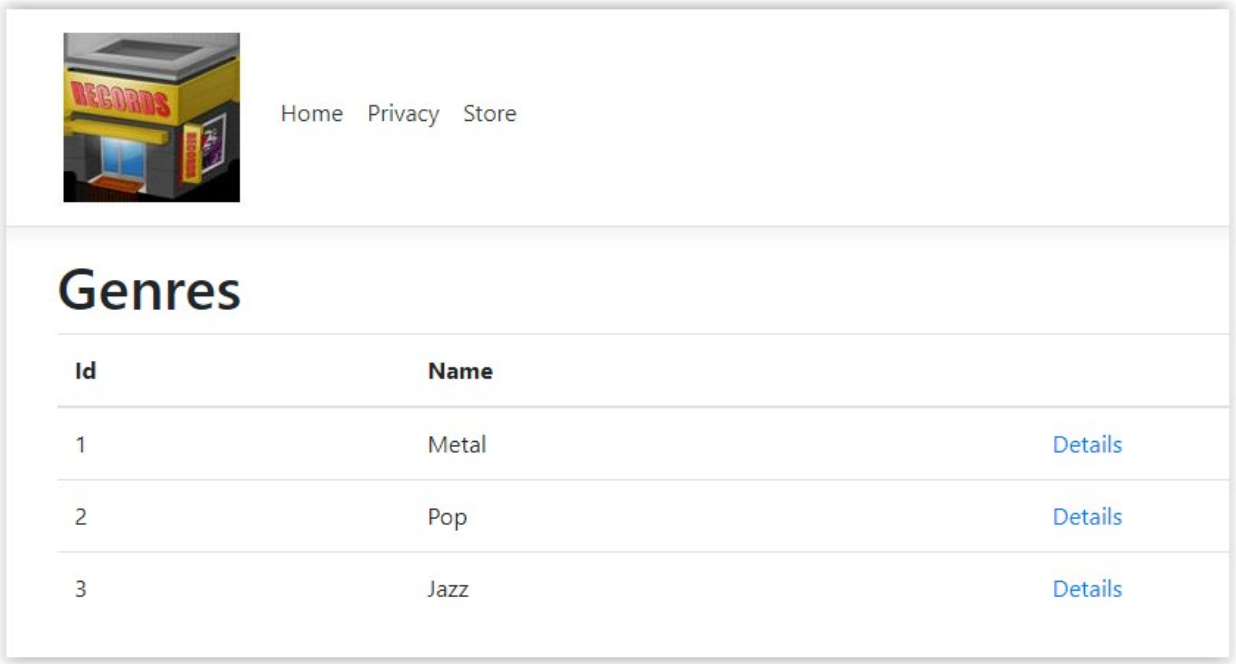
Razor (Incomplete)

```
...
    <nav class="navbar navbar-expand-sm navbar-toggleable-sm navbar-light bg-
white border-bottom box-shadow mb-3">
            <div class="container">
                <a class="navbar-brand" asp-area="" asp-controller="Home" asp-
action="Index">
                    <img src="~/images/logo.jpg" />
                </a>
                <button
...
```

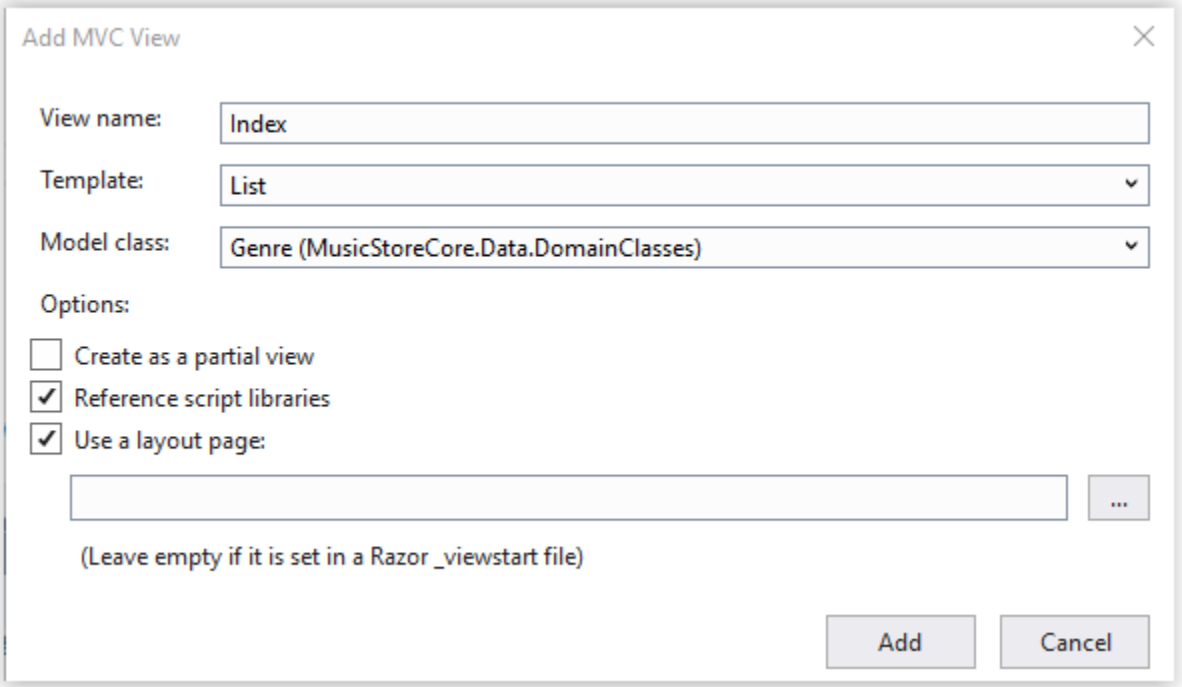
- Alter the "_ViewStart.cshtml" file so that the new layout is used for every view



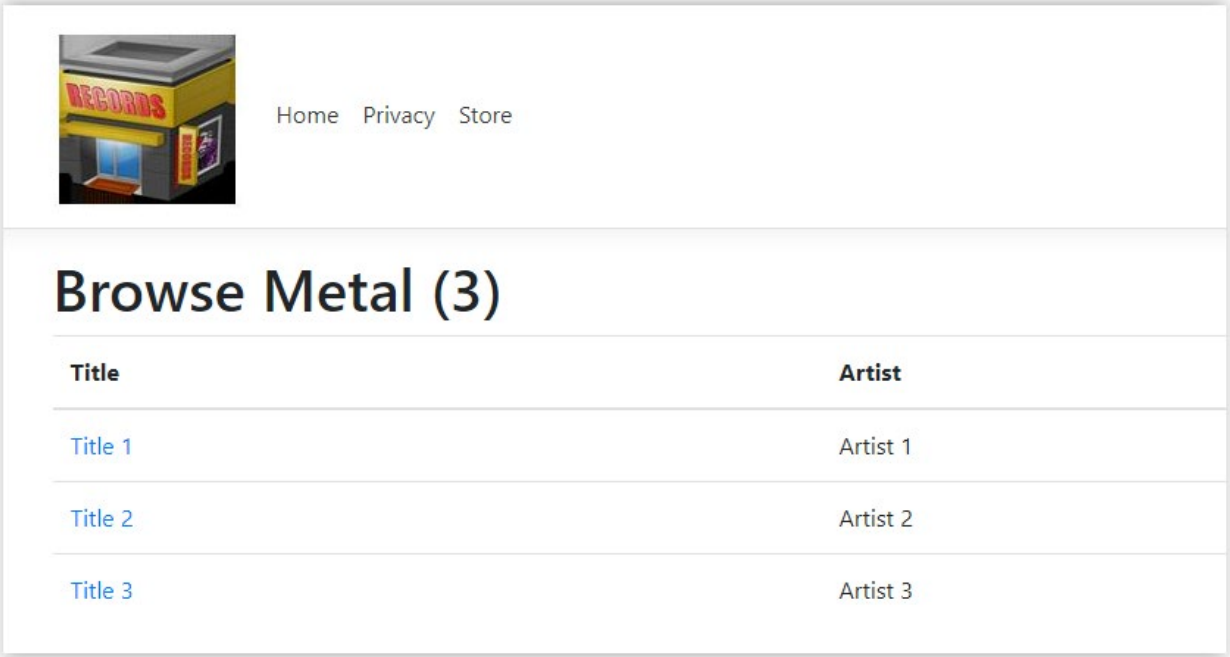
- Add a link to the overview of all the genres. The link text should be “Store”.



  - When creating the view you can use the “List” scaffold template with model class “Genre”

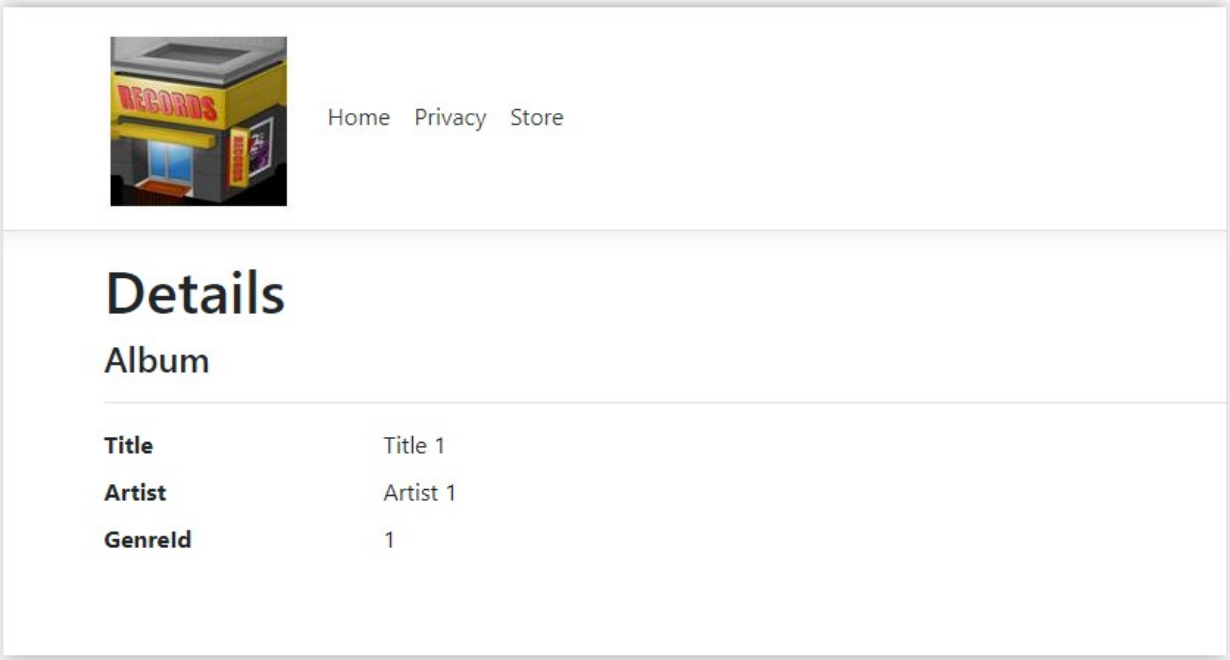


- When you click on a genre you should navigate to the page that contains an overview of all the albums of that genre.



  - When creating the view you can use the “List” scaffold template
  - The first column should display the title and be a link to the detail page at the same time.
  - Use the “ViewBag.Genre” (set by the controller action) to display the name of the genre in the title. Next to the name of the genre, the number of albums is displayed.

- When you click an album, you get the following page



  - When creating the view you can use the “Details” scaffold template

Step 3 – Html Helpers

Create your own Html helper to generate a h2-tag (subtitle).

C#

```
using Microsoft.AspNetCore.Html;
using Microsoft.AspNetCore.Mvc.Rendering;

namespace MusicStoreCore.Extensions
{
    public static class HtmlHelperExtensions
    {
        public static IHtmlContent SubTitle(this IHtmlHelper helper, string
text)
        {
            var html = $"<h2>{text}</h2>";
            return new HtmlString(html);
        }
    }
}
```

Use the helper do display the subtitles in the index, browse and detail view (instead of the literal h2-tag).

Step 4 – Viewmodel

On the details page of an album the name genre of the album should be displayed (instead of the id of the genre). So we should create a viewmodel that contains exactly the data we need to render.

- In the “Models” folder, add a “AlbumViewModelClass”

```
C#
namespace MusicStore.Models
{
    public class AlbumViewModel
    {
        public string Title { get; set; }
        public string Artist { get; set; }
        public string Genre { get; set; }
    }
}
```

- Modify the test “Details_ShowsDetailsOfAlbum”
    - The controller will need some class that can take an album and a genre and create an albumViewModel -> IAlbumViewModelFactory (“Create” method)
    - The IGenreRepository will need to have a “GetById” method that should be called once with the GenreId of the album
    - The “Create” method of the IAlbumViewModelFactory should have been called once with the correct parameters
    - The model of the view should be of type “AlbumViewModel”
- Add a folder “Models” in the test project.
- Add a test class “AlbumViewModelFactoryTests”
- Add a class “AlbumViewModelFactory” in the “Models” folder of the MVC project. Write the following tests:
    - “Create_ValidAlbumAndValidGenre_CorrectlyMapped”
    - “Create_MissingGenre_TrowsException”
    - “Create_MissingAlbum_TrowsException”
    - “Create_MismatchBetweenAlbumAndGenre_TrowsException”
- Register “AlbumViewModelFactory” in the dependency injection container (as a singleton)
- Modify the “Details” view to display the genre

Step 5 –View Component

- We want to reuse the display html for an album
    - Add a partial view "_Album" in the "Views/Shared" folder (with "AlbumViewModel" as the model class)
    - Copy and paste the <dl> tag from the "Details" view in the partial view.
    - Use the partial view in the "Details" view
- We want to display the album of the day in the footer. To do this we need to have an "AlbumOfTheDay" View Component. For this exercise we'll add some dummy logic. So no unit testing is needed. Copy and paste the <dl> tag from the "Details" view to the View of the ViewComponent.

C#

```
using Microsoft.AspNetCore.Mvc;
using MusicStoreCore.Models;

namespace MusicStoreCore.ViewComponents.AlbumOfTheDay
{
	public class AlbumOfTheDay:ViewComponent
	{
		private readonly AlbumViewModel album;
		public AlbumOfTheDay()
		{
			album = new AlbumViewModel()
			{
				Artist = "A popular artist",
				Genre = "Some genre",
				Title = "Some title"
			};
		}
		public IViewComponentResult Invoke()
		{
			return View("_Album", album);
		}

	}
}
```

- Display the album of the day in the footer.

Razor (Incomplete)

```
...
<footer class="border-top footer text-muted">
   <div class="container">
```

```
        <div class="card">
            <div class="card-body">
                <h5 class="card-title">Album of the day</h5>
                @await Component.InvokeAsync("AlbumOfTheDay")
            </div>
        </div>
        &copy; 2019 - MusicStoreCore - <a asp-area="" asp-controller="Home"
asp-action="Privacy">Privacy</a>
    </div>
</footer>
...
```