Walkthrough 4 - MVC Application

Setup

This walkthrough will fully implement an MVC application to track movies.

1. Start Visual Studio
2. Click Create a new project.
3. Set language to **C#** and project type to **Web**.
4. Select the ASP.NET Core Web App (Model-View-Controller) template, click Next.
5. Set Project name to **MovieTracker**.
6. Set Location to a **folder of your choosing**.
7. Ensure Place solution and project in the same directory is not selected, click Next.
8. Set version to **.NET 5.0**, unselect Configure for HTTPS, click Create.
9. Run the site and navigate around it. Close the browser.
10. This site has been styled with **Bootstrap 4.3.1** (https://getbootstrap.com/docs/4.3/getting-started/introduction/) .

Startup.cs

1. Put a breakpoint on the constructor (F9) and press F5 to run in debug mode.
2. Hover over configuration and expand it and Providers to see appsettings.json, environment variables, command line parameters and more.
3. Press Shift+F5 to halt the debugger. Remove the breakpoint.

Movie.cs

1. In the Models folder, create a new class named **Movie.cs**. The question mark after DateTime and int indicates that DateSeen and Rating are nullable types, or optional.
2.
```
public class Movie
{
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime? DateSeen { get; set; }
    public string Genre { get; set; }
    public int? Rating { get; set; }
}
```
3. Add a Key annotation to Id. Use quick actions to add the **using System.ComponentModel.DataAnnotations;** directive.
4.
```
public class Movie
{
    [Key]
    public int Id { get; set; }
    public string Title { get; set; }
    public DateTime? DateSeen { get; set; }
    public string Genre { get; set; }
    public int? Rating { get; set; }
}
```
5. Add Required and Range annotations.
6.
```
public class Movie
{
    [Key]
    public int Id { get; set; }

    [Required]
    public string Title { get; set; }

    public DateTime? DateSeen { get; set; }

    public string Genre { get; set; }

    [Range(1, 10)]
    public int? Rating { get; set; }
}
```
7. Save the file.

MoviesController.cs

1. In Solution Explorer, right-click the Controllers folder and select Add / Controller...
2. Select the MVC Controller with read/write actions template, click Add and name it **MoviesController**.
3. Add a list of movies to act as data, add the **using MovieTracker.Models;** directive.
4.
```
public class MoviesController : Controller
{
    private static List<Movie> movies = new List<Movie>
    {
        new Movie
        {
            Id = 1,
            Title = "Birds of Prey",
            DateSeen = DateTime.Now.AddDays(-50),
            Genre = "Action",
            Rating = 6
        },
        new Movie
        {
            Id = 2,
            Title = "Palm Springs",
            DateSeen = DateTime.Now.AddDays(-25),
            Rating = 7
        },
        new Movie
        {
            Id = 3,
            Title = "Hamilton",
            Genre = "Drama"
        }
    };

    // GET: Movies
    public ActionResult Index()
    ...
```

5. Right-click in the Index method and select Add View... .
6. Select Razor View, click Add.
7. Set Template to **List**.
8. Set Model class to **Movie (MovieTracker.Models)**, click Add.
9. At the bottom of Index.cshtml, notice the three links that have commented placeholders for the primary keys.
10. Return to MoviesController.cs and update the Index method to return the list of movies.
11.
```
public ActionResult Index()
{
    return View(movies);
}
```

12. Press Ctrl+F5 to launch the site.
13. Browse to **http://localhost:12345/movies** to see the list of movies.

_Layout.cshtml

1. Press Ctrl+T to open the all-purpose Go to tool. Begin typing _Layout, select it when its recognized.
2. Update the title and application link. Also add a link to the MoviesController / Index action. Finally, update the footer.
3.
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>@ViewData["Title"] - ~~MovieTracker~~Movie Tracker</title>

    <environment include="Development">
    ...
```

```
...
<body>
    <header>
        <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">~~MovieTracker~~Movie Tracker</a>
                <button class="navbar-toggler" type="button" data-toggle="collapse" data-target=".navbar-collapse" aria-controls="navbarSu
pportedContent"
                        aria-expanded="false" aria-label="Toggle navigation">
                    <span class="navbar-toggler-icon"></span>
                </button>
                <div class="navbar-collapse collapse d-sm-inline-flex flex-sm-row-reverse">
                    <ul class="navbar-nav flex-grow-1">
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Index">Home</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Movies" asp-action="Index">Movies</a>
                        </li>
                        <li class="nav-item">
                            <a class="nav-link text-dark" asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
                        </li>
                    </ul>
                </div>
                ...
```

4.
```
...
<footer class="border-top footer text-muted">
    <div class="container">
        &copy; ~~2021~~@DateTime.Now.Year - ~~MovieTracker~~Movie Tracker - <a asp-area="" asp-controller="Home" asp-action="Privacy">Privacy</a>
    </div>
</footer>
...
```

5. Save the file, refresh the browser.
6. Hover over the Edit, Details and Delete links and note that they are lacking the routing information to access a specific movie.

Index.cshtml

1. Update the links so that they will use the movie's Id

2.
```
...
<td>
    @Html.ActionLink("Edit", "Edit", new { ~~/* id=item.PrimaryKey */~~id = item.Id }) |
    @Html.ActionLink("Details", "Details", new { ~~/* id=item.PrimaryKey */~~id = item.Id }) |
    @Html.ActionLink("Delete", "Delete", new { ~~/* id=item.PrimaryKey */~~id = item.Id })
</td>
...
```

3. Save the file, refresh the browser. Hover the links again; they are now correct. If you click any of them you will receive an error message, because none of those views exist yet.

MoviesController.cs

1. Update the Details method to find the selected movie and return it.
2.
```
public ActionResult Details(int id)
{
    var movie = movies.Find(m => m.Id == id);
    return View(movie);
}
```

3. Save the file.
4. Right-click in the Details method and select Add View... .
5. Select Razor View, click Add.
6. Set Template to **Details**.
7. Set Model class to **Movie (MovieTracker.Models)**, click Add.

Details.cshtml

1. Update the Edit link.
2.
```
...
<div>
    @Html.ActionLink("Edit", "Edit", new { ~~/* id = Model.PrimaryKey */~~id = Model.Id }) |
    <a asp-action="Index">Back to List</a>
</div>
```

3. Save the file, refresh the browser. Click one of the Details links.
4. Notice that the time displays for movies and that the heading is DateSeen.

Movie.cs

1. Add additional annotations to tidy the display.
2.
```
    public class Movie
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public string Title { get; set; }

        [DataType(DataType.Date)]
        [Display(Name = "Date Seen")]
        public DateTime? DateSeen { get; set; }

        public string Genre { get; set; }

        [Range(1, 10)]
        public int? Rating { get; set; }
    }
```

3. Alternatively, multiple attributes can be specified in one annotation.
4.
```
    public class Movie
    {
        [Key]
        public int Id { get; set; }

        [Required]
        public string Title { get; set; }

        [DataType(DataType.Date)], Display(Name = "Date Seen")]
        public DateTime? DateSeen { get; set; }

        public string Genre { get; set; }

        [Range(1, 10)]
        public int? Rating { get; set; }
    }
```

5. Save the file, refresh the browser. Note the improved output in the browser.
6. Click the Back to List link and notice that the model updates appear here too.

MoviesController.cs

1. Update the method signature of the POST Create method to accept a Movie object instead of the more general IFormCollection object, verify that the model state is valid and if so add the movie to the list; if not to return the view with the movie.
2.
```
public ActionResult Create(~~IFormCollection collection~~Movie movie)
{
    try
    {
        if (ModelState.IsValid)
        {
            movies.Add(movie);
            return RedirectToAction(nameof(Index));
        }
        else
        {
            return View(movie);
        }
    }
    catch
    {
        return View();
    }
}
```

3. Save the file.
4. Right-click in the Create method and select Add View... .
5. Select Razor View, click Add.
6. Set Template to **Create**.
7. Set Model class to **Movie (MovieTracker.Models)**, click Add.
8. Refresh the browser.
9. Click the Create New link. Add a new movie.

Create.cshtml

1. Add an autofocus attribute to the input tag for Id.
2.
```
...
<div class="form-group">
    <label asp-for="Id" class="control-label"></label>
    <input asp-for="Id" class="form-control" autofocus />
    <span asp-validation-for="Id" class="text-danger"></span>
</div>
...
```
3. Save the file.
4. Click the Create New link again. Notice focus in the Id text box.
5. Click the Create button right away to see the validation errors.
6. Correct the errors. Set the rating higher than 10, click the Create button.
7. Click the Back to List link.

MoviesController.cs

1. Add a method to get a movie, knowing the id.
2.
```
private Movie GetMovie(int id)
{
    return movies.Find(m => m.Id == id);
}
```
3. Update the GET Edit method to find the selected movie and return it.
4.
```
public ActionResult Edit(int id)
{
    var movie = GetMovie(id);
    return View(movie);
}
```
5. Update the Details method to also use the new method.
6.
```
public ActionResult Details(int id)
{
    ~~var movie = movies.Find(m => m.Id == id);~~
    var movie = GetMovie(id);
    return View(movie);
}
```
7. Update the method signature of the POST Edit method to accept a Movie object instead of an IFormCollection object, verify that the model state is valid and if so find the index of the selected movie and update the list with it; if not to return the view with the movie.
8.
```
public ActionResult Edit(int id, ~~IFormCollection collection~~Movie movie)
{
    try
    {
        if (ModelState.IsValid)
        {
            var index = movies.FindIndex(m => m.Id == id);
            movies[index] = movie;
            return RedirectToAction(nameof(Index));
        }
        else
        {
            return View(movie);
        }
    }
    catch
    {
        return View();
    }
}
```

9. Save the file.
10. Right-click in the Edit method and select Add View... .
11. Select Razor View, click Add.
12. Set Template to **Edit**.
13. Set Model class to **Movie (MovieTracker.Models)**, click Add.

Edit.cshtml

1. Add an autofocus attribute to the input tag for Id.
2.
```
...
<div class="form-group">
    <label asp-for="Id" class="control-label"></label>
    <input asp-for="Id" class="form-control" autofocus />
    <span asp-validation-for="Id" class="text-danger"></span>
</div>
...
```
3. Save the file, refresh the browser.
4. Edit one of the movies.

MoviesController.cs

1. Update the GET Delete method to find the selected movie and return it.
2.
```
public ActionResult Delete(int id)
{
    var movie = GetMovie(id);
    return View(movie);
}
```
3. Add a method to get the movie's index in the array, knowing the id.
4.
```
private int GetMovieIndex(int id)
{
    return movies.FindIndex(m => m.Id == id);
}
```
5. Update the POST Edit method to also use the new method.
6.
```
public ActionResult Edit(int id, Movie movie)
{
    try
    {
        if (ModelState.IsValid)
        {
            ~~var index = movies.FindIndex(m => m.Id == id);~~
            var index = GetMovieIndex(id);
            movies[index] = movie;
            return RedirectToAction(nameof(Index));
        }
        else
        {
            return View(movie);
        }
    }
    catch
    {
        return View();
    }
}
```
7. Update the POST Delete method to find the index of the selected movie and remove it from the list.
8.
```
public ActionResult Delete(int id, IFormCollection collection)
{
    try
    {
        var index = GetMovieIndex(id);
        movies.RemoveAt(index);
        return RedirectToAction(nameof(Index));
    }
    catch
    {
        return View();
    }
}
```
9. Save the file.
10. Right-click in the Delete method and select Add View... .
11. Select Razor View, click Add.
12. Set Template to **Delete**.
13. Set Model class to **Movie (MovieTracker.Models)**, click Add.
14. Refresh the browser.
15. Delete one of the movies.