Walkthrough 7 - Testing

Setup

This walkthrough will add testing to the MVC movie tracking application.

1. Open MovieTracker from the end of the previous walkthrough.
2. Right-click the solution in the Solution Explorer, select Add / New Project...
3. Set language to **C#** and project type to **Test**.
4. Select the xUnit Test Project template, click Next.
5. Set Project name to **MovieTrackerTest**, leave Location as is, click Next.
6. Set version to **.NET 5.0**, click Create.

MovieTrackerTest

1. In solution explorer, under the new project, expand Dependencies.
2. Right-click the new project and select Add / Project Reference... .
3. Select MovieTracker, click OK.
4. Note that the MovieTracker project gets added as a dependency.
5. In the PMC, change the Default project drop-down list (near the top of the PMC) to **MovieTrackerTest**.
6. Issue the command **Install-Package Microsoft.EntityFrameworkCore.InMemory**.

UnitTest1.cs

1. Add a method that will initialize and return an in-memory database.
2. Add the **using MovieTracker.Data;**, **using Microsoft.EntityFrameworkCore;** and **using MovieTracker.Models;** directives.
3.
```
private MovieTrackerContext CreateContext(string databaseName)
{
    var options = new DbContextOptionsBuilder<MovieTrackerContext>()
        .UseInMemoryDatabase(databaseName: databaseName)
        .Options;
    var context = new MovieTrackerContext(options);

    context.Movie.AddRange(
        new Movie
        {
            Id = 1,
            Title = "Car Chases and Explosions",
            DateSeen = new DateTime(2021, 7, 1).Date,
            Genre = "Action",
            Rating = 6
        },
        new Movie
        {
            Id = 2,
            Title = "Silly Misunderstandings",
            DateSeen = new DateTime(2021, 8, 15).Date,
            Genre = "Comedy",
            Rating = 7
        },
        new Movie
        {
            Id = 3,
            Title = "Serious Discussions",
            DateSeen = new DateTime(2021, 9, 30).Date,
            Genre = "Drama",
            Rating = 8
        }
    );

    context.SaveChanges();
    return context;
}
```
4. Rename Test1 to **Index_NoInput_ReturnsMovies**.
5.
```
[Fact]
public void ~~Test1~~Index_NoInput_ReturnsMovies()
{
}
```
6. Add 3 comments to outline the structure of the method.
7.
```
[Fact]
public void Index_NoInput_ReturnsMovies()
{
    // Arrange
    // Act
    // Assert
}
```
8. Create a context, instantiate a new instance of MoviesController, add the **using MovieTracker.Controllers;** directive.
9. Note that dependency injection is allowing us to pass the in memory database to the controller.
10.
```
[Fact]
public void Index_NoInput_ReturnsMovies()
```

```
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    // Assert
}
```

11. Add a breakpoint to the line of code where the context is instantiated (F9).
12. From the Test menu, select Test Explorer.
13. In the Text Explorer, expand the test until you reach Index_NoInput_ReturnsMovies. Right-click it and select Debug.
14. The code will stop at the breakpoint, press F11 to step into CreateContext. Press F11 to keep stepping until the context.Movie.AddRange statement is about to execute.
15. Hover over context and expand it, then Movie / Results View / [0]. Note that it is the first Movie created in OnModelCreating method of the context class.
16. Press F11 until the return statement.
17. The app will crash before the return because of a duplicate primary key. Press Shift+F5 to stop the application, if necessary.

MovieTrackerContext.cs

1. Comment out the call to the EnsureCreated method in the constructor.
2.
```
public MovieTrackerContext (DbContextOptions<MovieTrackerContext> options)
    : base(options)
{
    //Database.EnsureCreated();
}
```

3. Save the file.

UnitTest1.cs

1. Debug again, this time the test should pass.
2. Remove the breakpoint.
3. Call the Index method.
4.
```
[Fact]
public void Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = moviesController.Index();

    // Assert
}
```

5. Right-click the Index method and select Go To Definition (F12).
6. Note that the Index method is asynchronous and returns a Task<IActionResult>.
7. Update the call to the Index method to make it an asynchronous call.
8.
```
[Fact]
public void Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
}
```

9. This requires updating this method to be asynchronous, add the **using System.Threading.Tasks;** directive.
10.
```
[Fact]
public async ~~void~~Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
}
```

11. Add a breakpoint to the act line of code (F9).
12. Debug the test.
13. The code will stop at the breakpoint, press F10 to step over the call to the Index method.
14. Hover over actionResult to see that the data type is actually a ViewResult.
15. The Autos and Locals windows should be at the bottom of the screen, if they are not, either can be accessed from the Debug / Windows menu.
16. In Autos or Locals, expand actionResult and note that the Model property is a List of Movie objects.

17. Press F5 to allow the test to finish.
18. Remove the breakpoint.
19. Add an assertion to check the return type, add the **using Microsoft.AspNetCore.Mvc;** directive.
20.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
}
```
21. From the Test Explorer, click Run All to run all tests. It should pass.
22. Convert the more general actionResult to its specific ViewResult type and check that its model is a list of movies. Add the **using System.Collections.Generic;** directive.
23.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
}
```
24. Convert the view result model to a list of movies, check the movie count, and incorrectly the Id of the 1st movie.
25.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(10, movies[0].Id);
}
```
26. Run the test, note the test results appear in the bottom area of the Test Explorer.
27. Fix the assertion.
28.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(1~~0~~, movies[0].Id);
}
```
29. Run the test again, it should pass.
30. Test each attribute of the model, use different models.
31.
```
[Fact]
public async Task Index_NoInput_ReturnsMovies()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
```

```
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(1, movies[0].Id);
    Assert.Equal("Silly Misunderstandings", movies[1].Title);
    Assert.Equal(new DateTime(2021, 9, 30).Date, movies[2].DateSeen);
    Assert.Equal("Action", movies[0].Genre);
    Assert.Equal(7, movies[1].Rating);
}
```

32. Run the test again, it should pass.
33. Copy the Index_NoInput_ReturnsMovies method and rename it **Details_MovieId_ReturnsMovie**.
34.
```
[Fact]
public async Task ~~Index_NoInput_ReturnsMovies~~Details_MovieId_ReturnsMovie()
{
    // Arrange
    var context = CreateContext("Index");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Index();

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(1, movies[0].Id);
    Assert.Equal("Silly Misunderstandings", movies[1].Title);
    Assert.Equal(new DateTime(2021, 9, 30).Date, movies[2].DateSeen);
    Assert.Equal("Action", movies[0].Genre);
    Assert.Equal(7, movies[1].Rating);
}
```

35. Update the database named passed to CreateContext; this is being done so that each test gets its own new copy of the in-memory database. Also, update the act code to call the Details method.
36.
```
[Fact]
public async Task Details_MovieId_ReturnsMovie()
{
    // Arrange
    var context = CreateContext("~~Index~~Details");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.~~Index~~Details(1);

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<List<Movie>>(viewResult.Model);
    var movies = viewResult.Model as List<Movie>;
    // Check the number of movies and a portion of every record and all fields
    Assert.Equal(3, movies.Count);
    Assert.Equal(1, movies[0].Id);
    Assert.Equal("Silly Misunderstandings", movies[1].Title);
    Assert.Equal(new DateTime(2021, 9, 30).Date, movies[2].DateSeen);
    Assert.Equal("Action", movies[0].Genre);
    Assert.Equal(7, movies[1].Rating);
}
```

37. Update the assertion code to handle a single movie and check all of its attributes.
38.
```
[Fact]
public async Task Details_MovieId_ReturnsMovie()
{
    // Arrange
    var context = CreateContext("Details");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Details(1);

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType~~<List~~<Movie>~~>~~(viewResult.Model);
    var movie~~s~~ = viewResult.Model as ~~List<~~Movie~~>~~;
    // ~~Check the number of movies and a portion of every record and all fields~~Test all properties
    ~~Assert.Equal(3, movies.Count);~~
    Assert.Equal(1, movie~~s[0]~~.Id);
    Assert.Equal("~~Silly Misunderstandings~~Car Chases and Explosions", movie~~s[1]~~.Title);
    Assert.Equal(new DateTime(2021, ~~9~~7, ~~30~~1).Date, movie~~s[2]~~.DateSeen);
    Assert.Equal("Action", movie~~s[0]~~.Genre);
    Assert.Equal(~~7~~6, movie~~s[1]~~.Rating);
}
```

39. In the Test Explorer, run each test individually by right-clicking them and selecting Run, they pass.
40. Run UnitTest1 which will run both tests, they pass.
41. Copy the Details_MovieId_ReturnsMovie method and rename it **Create_Movie_RedirectsToIndex**.
42.
```
[Fact]
public async Task ~~Details_MovieId_ReturnsMovie~~Create_Movie_ReturnsToIndex()
{
    // Arrange
    var context = CreateContext("Details");
    var moviesController = new MoviesController(context);

    // Act
```

```
    var actionResult = await moviesController.Details(1);

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<Movie>(viewResult.Model);
    var movie = viewResult.Model as Movie;
    // Test all properties
    Assert.Equal(1, movie.Id);
    Assert.Equal("Car Chases and Explosions", movie.Title);
    Assert.Equal(new DateTime(2021, 7, 1).Date, movie.DateSeen);
    Assert.Equal("Action", movie.Genre);
    Assert.Equal(6, movie.Rating);
}
```

43. Update the database name and the act code to call the Create method and delete most of the assert code.
44.
```
[Fact]
public async Task Create_Movie_ReturnsToIndex()
{
    // Arrange
    var context = CreateContext("DetailsCreate");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.DetailsCreate(1
        new Movie
        {
            Title = "Testing for Fun and Profit",
            DateSeen = DateTime.Now.Date,
            Genre = "Drama",
            Rating = 9
        });

    // Assert
    Assert.IsType<ViewResult>(actionResult);
    var viewResult = actionResult as ViewResult;
    Assert.IsType<Movie>(viewResult.Model);
    var movie = viewResult.Model as Movie;
    // Test all properties
    Assert.Equal(1, movie.Id);
    Assert.Equal("Car Chases and Explosions", movie.Title);
    Assert.Equal(new DateTime(2021, 7, 1).Date, movie.DateSeen);
    Assert.Equal("Action", movie.Genre);
    Assert.Equal(6, movie.Rating);
}
```

45. Add a breakpoint to the act line of code (F9).
46. In the Test Explorer, right-click the new test and debug it.
47. Step into the code with F11. Notice that the new movie gets added to the database.
48. Continue stepping to the assert, it will error.
49. The return type isn't a ViewResult, but a RedirectToActionResult. Press F5 to allow the code to complete. Remove the breakpoint.
50. Update the assert and test the ActionName property of the RedirectToActionResult.
51.
```
[Fact]
public async Task Create_Movie_ReturnsToIndex()
{
    // Arrange
    var context = CreateContext("Create");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Create(
        new Movie
        {
            Title = "Testing for Fun and Profit",
            DateSeen = DateTime.Now.Date,
            Genre = "Drama",
            Rating = 9
        });

    // Assert
    Assert.IsType<ViewResultRedirectToActionResult>(actionResult);
    var redirectToActionResult = actionResult as RedirectToActionResult;
    Assert.Equal("Index", redirectToActionResult.ActionName);
}
```

52. Run the new test, it should pass.
53. Call the Index method to get a count of movies, to verify the create succeeded.
54.
```
[Fact]
public async Task Create_Movie_ReturnsToIndex()
{
    // Arrange
    var context = CreateContext("Create");
    var moviesController = new MoviesController(context);

    // Act
    var actionResult = await moviesController.Create(
        new Movie
        {
            Title = "Testing for Fun and Profit",
            DateSeen = DateTime.Now.Date,
            Genre = "Drama",
            Rating = 9
        });

    // Assert
    Assert.IsType<RedirectToActionResult>(actionResult);
    var redirectToActionResult = actionResult as RedirectToActionResult;
    Assert.Equal("Index", redirectToActionResult.ActionName);
    // Verify count
```

```
    actionResult = await moviesController.Index();
    var viewResult = actionResult as ViewResult;
    var movies = viewResult.Model as List<Movie>;
    Assert.Equal(4, movies.Count);
}
```

55. Run all tests. They should pass.

MoviesController.cs

1. Open MoviesController.cs.
2. Scroll to the Details method.
3. From the Test menu, select Live Unit Testing / Start (this is only available in Visual Studio Enterprise editions).
4. Notice the green checkmarks and blue dashes that appear.
5. In the Details method, change the return to not return a movie to the view.
6.
```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id must be specified."
            });
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }

    return View(~~movie~~);
}
```

7. Note that some of the tests now fail. Change the return back.
8.
```
public async Task<IActionResult> Details(int? id)
{
    if (id == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                Description = "Movie id must be specified."
            });
    }

    var movie = await _context.Movie
        .FirstOrDefaultAsync(m => m.Id == id);
    if (movie == null)
    {
        return View("Error",
            new ErrorViewModel
            {
                RequestId = id.ToString(),
                Description = $"Unable to find movie with id={id}."
            });
    }

    return View(movie);
}
```

9. All the tests pass again.
10. From the Test menu, select Live Unit Testing / Stop.