# In-class Lab 07

ASP.NET Core MVC

## 1 Beginning the lab

1. To begin this lab, download the code from the book web site, or from Github, at `https://github.com/Apress/pro-asp.net-core-mvc-2`. Copy the solution and supporting files from `pro-asp.net-core-mvc-2/06 - Working with Visual Studio` to your lab directory. Open the solution in Visual Studio and build the application. If everything work, start the application with debugging. This will be the starting point for Lab 7.

2. Create a `_ViewImports.cshtml` file by right clicking on the `Views` folder and selecting `Add` ▶ `New Item`. Select `ASP.NET Core` ▶ `Web` ▶ `ASP.NET` ▶ `Razor View Imports`. Name the file `_ViewImports.cshtml`. Note the leading underscore. See figure 1. Click `Add`.

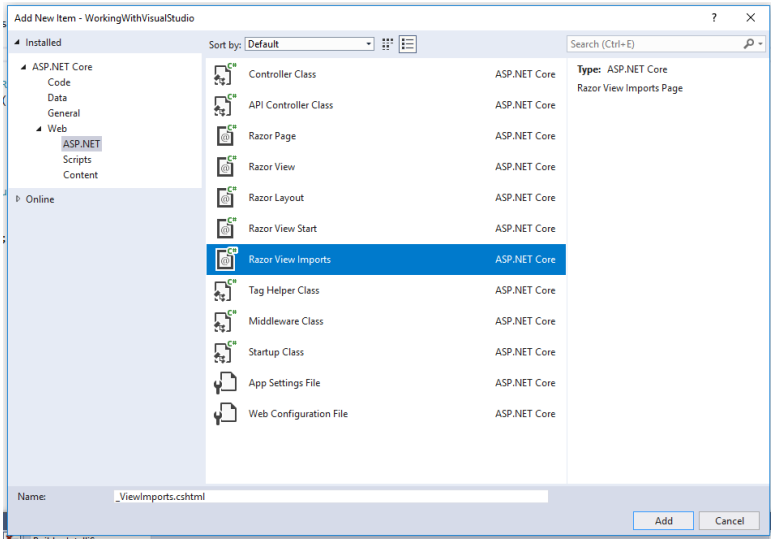


Figure 1: Adding `_ViewImports.cshtml`

3. Edit `_ViewImports.cshtml` by adding this line:
**`@addTagHelper *, Microsoft.AspNetCore.Mvc.TagHelpers`**.

4. Edit `HomeController.cs` as shown in listing 1.

Listing 1: Edits to `HomeController.cs`

```
using Microsoft.AspNetCore.Mvc;
using WorkingWithVisualStudio.Models;
using System.Linq;

namespace WorkingWithVisualStudio.Controllers
{
```

```
    public class HomeController : Controller
    {
        SimpleRepository Repository = SimpleRepository.SharedRepository;
        public IActionResult Index() => View(Repository.Products
                        .Where(p => p?.Price < 50));
        [HttpGet]
        public IActionResult AddProduct() => View(new Product());

        [HttpPost]
        public IActionResult AddProduct(Product p)
        {
            Repository.AddProduct(p);
            return RedirectToAction("Index");
        }
    }
}
```

5. Create a new view in `/Views/Home/`. Name the view `AddProduct.cshtml` and edit it as shown in listing 2.

Listing 2: Adding view `AddProduct.cshtl`

```
@model WorkingWithVisualStudio.Models.Product
@{ Layout = null; }

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Working with Visual Studio</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
</head>
<body class="p-2">
    <h3 class="text-center">Create Product</h3>
    <form asp-action="AddProduct" method="post">
        <div class="form-group">
            <label asp-for="Name">Name:</label>
            <input asp-for="Name" class="form-control" />
        </div>
        <div class="form-group">
            <label asp-for="Price">Price:</label>
            <input asp-for="Price" class="form-control" />
        </div>
        <div class="text-center">
            <button type="submit" class="btn␣btn-primary">Add</button>
            <a asp-action="Index" class="btn␣btn-secondary">Cancel</a>
        </div>
    </form>
</body>
</html>
```

6. Edit view `Index.cshtml` as shown in listing 3.

Listing 3: Edits to `Index.cshtml`

```
@model IEnumerable<WorkingWithVisualStudio.Models.Product>
@{ Layout = null; }

<!DOCTYPE html>
<html>
<head>
    <meta name="viewport" content="width=device-width" />
    <title>Working with Visual Studio</title>
    <link rel="stylesheet" href="/lib/bootstrap/dist/css/bootstrap.min.css" />
</head>
    <body class="p-1">
    <h3 class="text-center">Products</h3>
    <table class="table table-bordered table-striped">
        <thead>
            <tr><td>Name</td><td>Price</td></tr>
        </thead>
        <tbody>
            @foreach (var p in Model) {
                <tr>
                    <td>@p.Name</td>
                    <td>@($"{p.Price:C2}")</td>
                </tr>
            }
        </tbody>
    </table>
<div class="text-center">
    <a class="btn btn-primary" asp-action="AddProduct">Add New Product</a>
</div>
</body>
</html>
```

7. Start withoug debugging. Add a new product, name and price. View the results.

## 2 Unit testing

8. Begin by adding a testing project to your application. Right click the *solution* and select `Add` ▶ `New Project`. Select `Visual C#` ▶ `Web` ▶ `.NET Core` ▶ `xUnit Test Project (,NET Core)`. Name the project `WorkingWthVisualStudio.Tests`. See figure 2. Click `OK`.

9. Delete the file `UnitTest1.cs` by right clicking on it and selecting `Delete` ▶ `OK`.

10. To add the appropriate reference to `WorkingWithVisualStudio.Tests`, right click the project and select `Add` ▶ `Reference ...`. Check `WorkingWithVisualStudio`. See figure 3. Click `OK`.

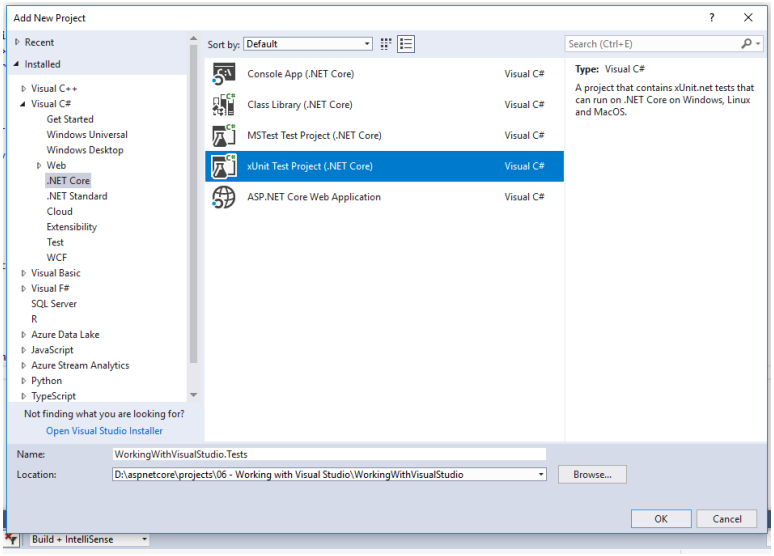


Figure 2: Adding a test project

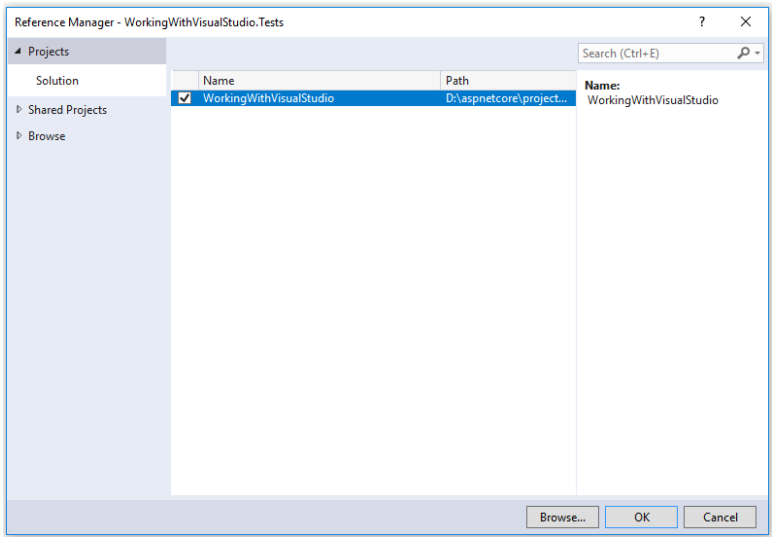


Figure 3: Addig reference to the test project

11. Add a test file to the Tests project by right clicking the Tests project and selecting `Add` ▶ `Class`. Add a Class file and name it `ProductTests.cs` Edit the fill as shown in listing 4. This adds two methods named `CanChangeProductName()` and`CanChangeProductPrice()`.

Listing 4: Class `PrroductTests.cs`

```
using System;
using System.Collections.Generic;
using System.Text;
using WorkingWithVisualStudio.Models;
using Xunit;
```

```
namespace WorkingWithVisualStudio.Tests
{
    public class ProductTests
    {
        [Fact]
        public void CanChangeProductName()
        {
            // Arrange
            var p = new Product { Name = "Test", Price = 100M };
            // Act
            p.Name = "New Name";
            //Assert
            Assert.Equal("New Name", p.Name);
        }
        [Fact]
        public void CanChangeProductPrice()
        {
            // Arrange
            var p = new Product { Name = "Test", Price = 100M };
            // Act
            p.Price = 200M;
            //Assert
            Assert.Equal(100M, p.Price); //Error, should be 100M
        }
    }
}
```

12. Build the solution. You can do this by the keyboard shortcut `ALT-B, B`.
13. Invoke the Test Window (`ALT-S, W, T`), and click `Run All`. See figure 4. Correct the errorr in line 29 in listing 4 by changing `100M` to `200M`, rebuild, and Run Failed Tests.
14. Add a new class file to `WorkingWithVisualStudio.Tests` and name it `Comparer.cs`. This content is shown in listing 5.

Listing 5: Contents of `Comparer.cs`

```
using System;
using System.Collections.Generic;
using System.Text;

namespace WorkingWithVisualStudio.Tests
{
    public class Comparer
    {
        public static Comparer<U> Get<U>(Func<U, U, bool> func)
        {
            return new Comparer<U>(func);
        }
    }
```

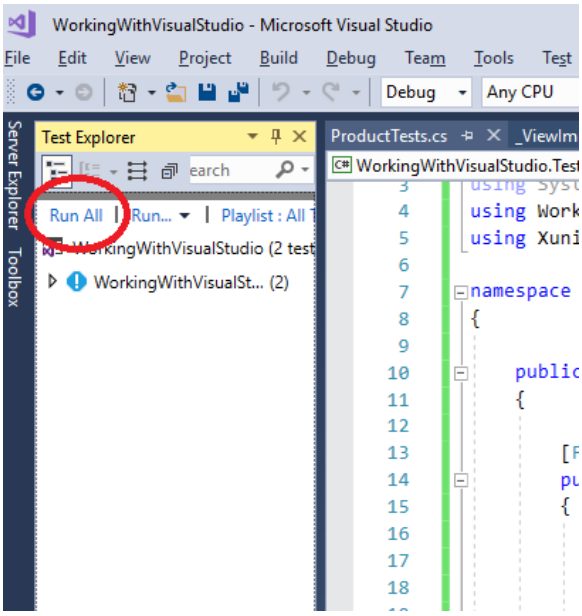


Figure 4: Test Window, Runn All

```
    public class Comparer<T> : Comparer, IEqualityComparer<T>
    {
        private Func<T, T, bool> comparisonFunction;
        public Comparer(Func<T, T, bool> func)
        {
            comparisonFunction = func;
        }
        public bool Equals(T x, T y)
        {
            return comparisonFunction(x, y);
        }
        public int GetHashCode(T obj)
        {
            return obj.GetHashCode();
        }
    }
}
```

15. Add a new class file to `WorkingWithVisualStudio.Tests` and name it `HomeControllerTests.cs`. This content is shown in listing 6.

Listing 6: Contents of `HomeControllerTests.cs`

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.AspNetCore.Mvc;
using WorkingWithVisualStudio.Controllers;
using WorkingWithVisualStudio.Models;
```

```
using Xunit;

namespace WorkingWithVisualStudio.Tests
{
    public class HomeControllerTests
    {

        [Fact]
        public void IndexActionModelIsComplete()
        {

            // Arrange
            var controller = new HomeController();

            // Act
            var model = (controller.Index() as ViewResult)?.ViewData.Model
                as IEnumerable<Product>;

            // Assert
            Assert.Equal(SimpleRepository.SharedRepository.Products, model,
                Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
                    && p1.Price == p2.Price));
        }
    }
}
```

16. Rebuild the solution (`ALT-B, B`) and Run All tests. One test will fail. Do you know why?
17. In the `Models` folder, create a new class named `IRepository.cs` with the contents as shown in listing 7.

Listing 7: Contents of `IRepository.cs`

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace WorkingWithVisualStudio.Models
{

    public interface IRepository
    {

        IEnumerable<Product> Products { get; }
        void AddProduct(Product p);
    }
}
```

18. In the `Simplerepository.cs` file, replace the line that reads **`public class SimpleRepository`** with this line: **`public class SimpleRepository : IRepository`**.

19. In the `HomeController.cs` file, replace the line that reads **`SimpleRepository Repository = SimpleRepository.SharedRepository;`** with this line: **`public IRepository Repository = SimpleRepository.SharedRepository;`**.

20. Edit `HoeControllerTests.cs` s sown in listing 8.

Listing 8: Contents of `HomeControllerTests.cs`

```
public class HomeControllerTests
{
    class ModelCompleteFakeRepository : IRepository
    {
    public IEnumerable<Product> Products  get;  = new Product[]
    {
        new Product Name = "p1", Price = 275M ,
        new Product Name = "p2", Price = 48.95M ,
        new Product Name = "p3", Price = 19.50M,
        new Product Name = "p4", Price = 34.95M
    };
    public void AddProduct(Product p)
    { }
    }

    [Fact]
    public void IndexActionModelIsComplete()
    {

        // Arrange
        var controller = new HomeController();
    controller.Repository = new ModelCompleteFakeRepository();

        // Act
        var model = (controller.Index() as ViewResult)?.ViewData.Model
            as IEnumerable<Product>;

        // Assert
    Assert.Equal(controller.Repository.Products, model,
            Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
                && p1.Price == p2.Price));
    }
}
```

21. To help diagnose the error, add another class and test as shown in listing 9 to `HomeControllerTests.cs`, rebuild the solution, and Run All tests again. What happens?

Listing 9: Addition to `HomeControllerTests.cs`

```
class ModelCompleteFakeRepositoryPricesUnder50 : IRepository
{
    public IEnumerable<Product> Products { get; } = new Product[]
    {
```

```
        new Product {Name = "p1", Price = 5M },
        new Product {Name = "p2", Price = 48.95M },
        new Product {Name = "p3", Price = 19.50M},
        new Product {Name = "p4", Price = 34.95M }
    };
    public void AddProduct(Product p)
    { }
}

[Fact]
public void IndexActionModelIsCompletePricesUnder50()
{

    // Arrange
    var controller = new HomeController();
    controller.Repository = new ModelCompleteFakeRepositoryPricesUnder50();

    // Act
    var model = (controller.Index() as ViewResult)?.ViewData.Model
        as IEnumerable<Product>;

    // Assert
    Assert.Equal(controller.Repository.Products, model,
        Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
            && p1.Price == p2.Price));
}
```

22. Remove the LINQ filter in `HomeController.cs` as shown in listing 10. Rebuild the solution and Rull All tests. This tine all the tests should pass. What was the "error?"

Listing 10: removing the LINQ filter

```
public IActionResult Index() => View(Repository.Products
                  //.Where(p => p?.Price < 50)
                  );
```

# 3 Improving unit tests

23. Make the changes to class file `HomeControllerTests.cs` shown in listing 11. Rebuild the solution and Run All tests. All the tests should pass.

Listing 11: Edits to `HomeControllerTests.cs`

```
namespace WorkingWithVisualStudio.Tests
{
    public class HomeControllerTests
    {
        class ModelCompleteFakeRepository : IRepository
        {
```

```
            public IEnumerable<Product> Products { get; set;  }
            public void AddProduct(Product p)
            { }
        }

        [Theory]
        [InlineData(275, 48.95, 9.50, 34.95)]
        [InlineData(5, 48.95, 9.50, 34.95)]
        public void IndexActionModelIsComplete(decimal price1, decimal price2, decimal
            price3, decimal price4)
        {

            // Arrange
            var controller = new HomeController();
            controller.Repository = new ModelCompleteFakeRepository
            {
                Products = new Product[]
                {
                    new Product { Name = "p1", Price = price1 },
                    new Product { Name = "p2", Price = price2 },
                    new Product { Name = "p3", Price = price3 },
                    new Product { Name = "p4", Price = price4 }
                }
            };


            // Act
            var model = (controller.Index() as ViewResult)?.ViewData.Model
                as IEnumerable<Product>;

            // Assert
            Assert.Equal(controller.Repository.Products, model,
                Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
                    && p1.Price == p2.Price));
        }
    }
}
```

24. Add a new class fule to the `WorkingWithVisualStudio.Tests` project. Name it `ProductTestData.cs`. The content is shown in listing 12.

Listing 12: `ProductTestData.cs`

```
using System.Collections;
using System.Collections.Generic;
using WorkingWithVisualStudio.Models;

namespace WorkingWithVisualStudio.Tests
{
```

```
    public class ProductTestData : IEnumerable<object[]>
    {

        public IEnumerator<object[]> GetEnumerator()
        {
            yield return new object[] { GetPricesUnder50() };
            yield return new object[] { GetPricesOver50 };
        }

        IEnumerator IEnumerable.GetEnumerator()
        {
            return this.GetEnumerator();
        }

        private IEnumerable<Product> GetPricesUnder50()
        {
            decimal[] prices = new decimal[] { 275, 49.95M, 19.50M, 24.95M };
            for (int i = 0; i < prices.Length; i++)
            {
                yield return new Product { Name = $"P{i␣+␣1}", Price = prices[i] };
            }
        }

        private Product[] GetPricesOver50 => new Product[] {
            new Product { Name = "P1", Price = 5 },
            new Product { Name = "P2", Price = 48.95M },
            new Product { Name = "P3", Price = 19.50M },
            new Product { Name = "P4", Price = 24.95M }};
    }
}
```

25. Edit class file `HomeControllerTests.cs` as shown in listing 13. Rebuild the solution and Run All tests. All the tests should pass.

Listing 13: Edits to `HomeControllerTests.cs`

```
public class HomeControllerTests
{
    class ModelCompleteFakeRepository : IRepository
    {
        public IEnumerable<Product> Products { get; set;  }
        public void AddProduct(Product p)
        { }
    }

    [Theory]
    [ClassData(typeof(ProductTestData))]
    public void IndexActionModelIsComplete(Product[] products)
    {
```

```
        // Arrange
        var controller = new HomeController();
        controller.Repository = new ModelCompleteFakeRepository
        {
            Products = products
        };


        // Act
        var model = (controller.Index() as ViewResult)?.ViewData.Model
            as IEnumerable<Product>;

        // Assert
        Assert.Equal(controller.Repository.Products, model,
            Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
                && p1.Price == p2.Price));
    }
}
```

26. Reenable the LINQ filter. Rebuild the projject and Run All Tests. Then, start without debugging and add a product.

27. Add the Moq package to your application. Right click `WrkingWithVisualStudio.Tests` and select `Manage NuGet Packages ...`. On the `Browse` tab, search for "moq". See figure 5. Install Moq and accept all dialog boxes.

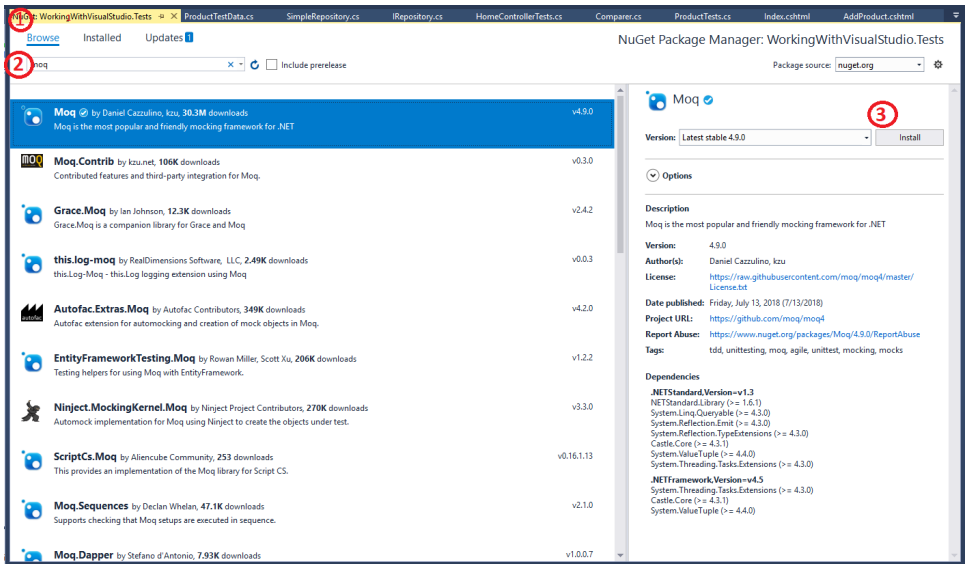


Figure 5: Installing Moq

28. Edit `HomeCotrollerTests.cs` as shown in listiing 14.

Listing 14: Edits to `HomeControllerTests.cs`

```
using Microsoft.AspNetCore.Mvc;
```

```
using System.Collections.Generic;
using WorkingWithVisualStudio.Controllers;
using WorkingWithVisualStudio.Models;
using Xunit;
using System;
using Moq;

namespace WorkingWithVisualStudio.Tests
{
    public class HomeControllerTests
    {

        [Theory]
        [ClassData(typeof(ProductTestData))]
        public void IndexActionModelIsComplete(Product[] products)
        {

            // Arrange
            var mock = new Mock<IRepository>();
            mock.SetupGet(m => m.Products).Returns(products);
            var controller = new HomeController  Repository = mock.Object ;

            // Act
            var model = (controller.Index() as ViewResult)?.ViewData.Model
                as IEnumerable<Product>;

            // Assert
            Assert.Equal(controller.Repository.Products, model,
                Comparer.Get<Product>((p1, p2) => p1.Name == p2.Name
                    && p1.Price == p2.Price));
        }

        [Fact]
        public void RepositoryPropertyCalledOnce()
        {

            // Arrange
            var mock = new Mock<IRepository>();
            mock.SetupGet(m => m.Products)
                .Returns(new[]  new Product  Name = "P1", Price = 100  );
            var controller = new HomeController  Repository = mock.Object ;

            // Act
            var result = controller.Index();

            // Assert
            mock.VerifyGet(m => m.Products, Times.Once);
        }
```

```
    }
}
```

29. Rebuild the solution and Run All tests. Then, disable the LINQ filter, rebuild the solution, and Run Failed Tests. All should pass successfully.

30. Start without debugging and put your application through its paces.