

Lab: Unit Testing in Visual Studio with xUnit

May 2020

Let's open the system under test (sut) Brainstorm

1. Clone repository to local folder from
 - <https://github.com/jagojar/workshop-testing>
2. Under folder **src** find solution file **TestingControllersSample.sln**.
3. Double click to open solution **TestingControllersSample.sln** in Visual Studio.
4. Run project **TestingControllersSample**
 - With F5 or
 - Top menu > Debug > Start Debugging
5. The browser loads the web application
See **image 2**

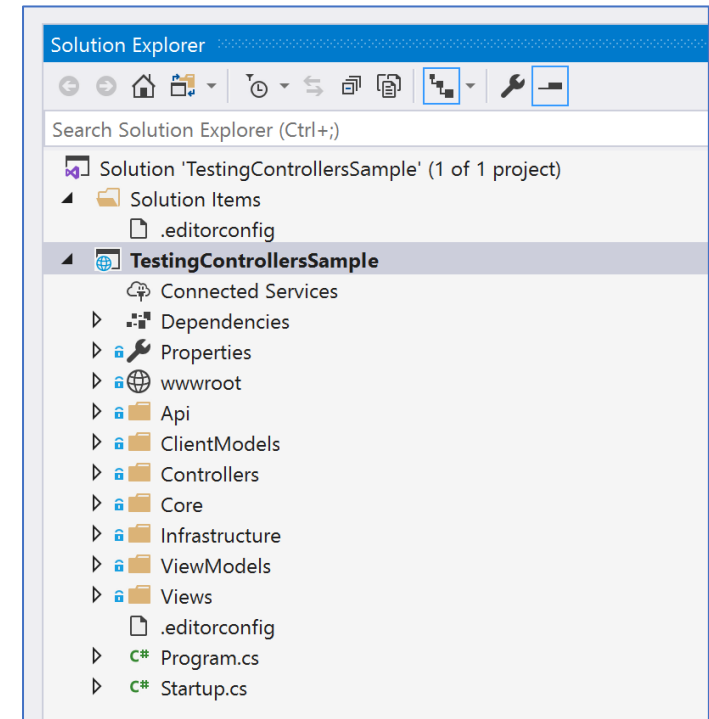


image 1

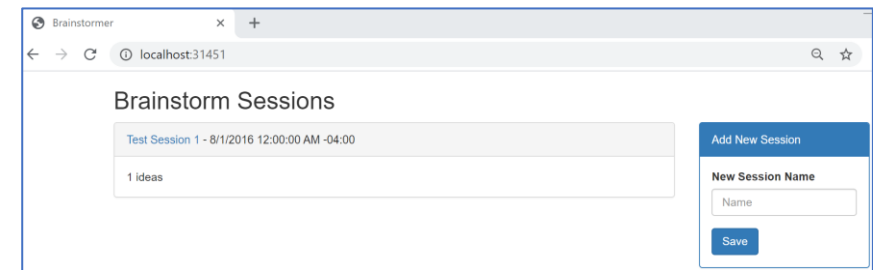
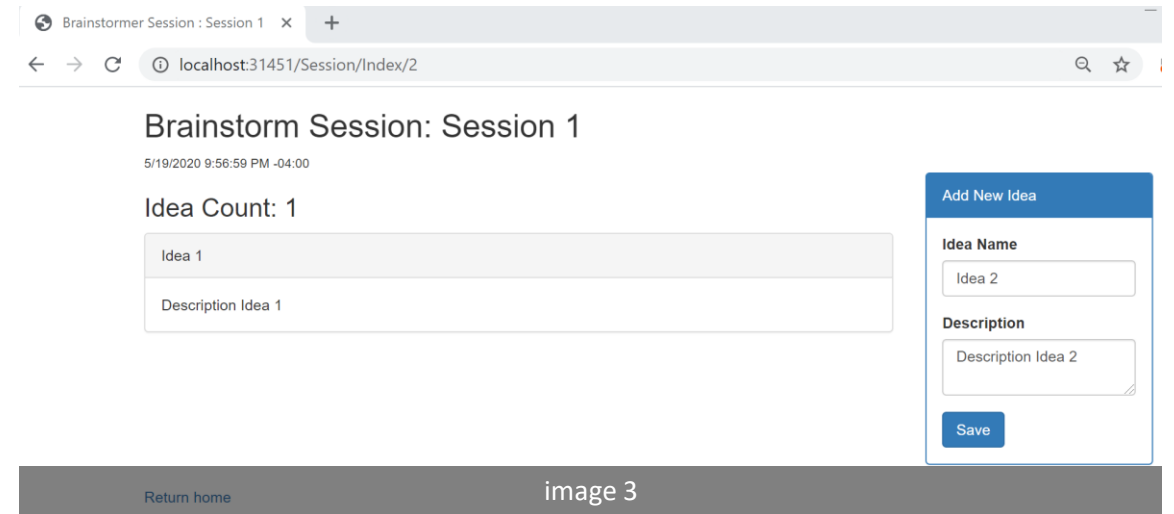


image 2

Let's play with Brainstorm

6. This web site helps to add Brainstorming sessions and add ideas to each session
7. The ideas for each session is displayed in the Session index. Example;
<http://localhost:31451/Session/Index/2>
8. This project needs unit tests. We are adding a new project to accomplish that



Let's add a Unit Test Project

9. In solution explorer, right click on **TestingControllersSample** solution name. Select from the menu Add > New Project (see **image 4**)
10. Select **xUnit Test Project C# (.Net Core)** (see **image 5**). Click Next
11. Name the project **TestingControllersSample.UnitTests**. Click Create

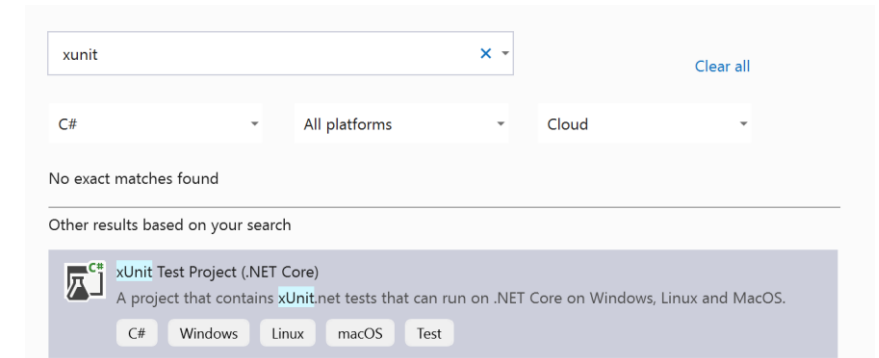


image 4

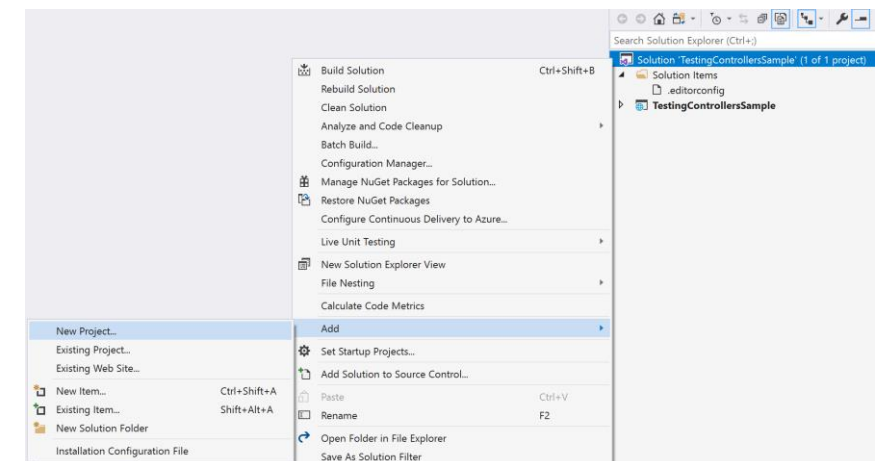
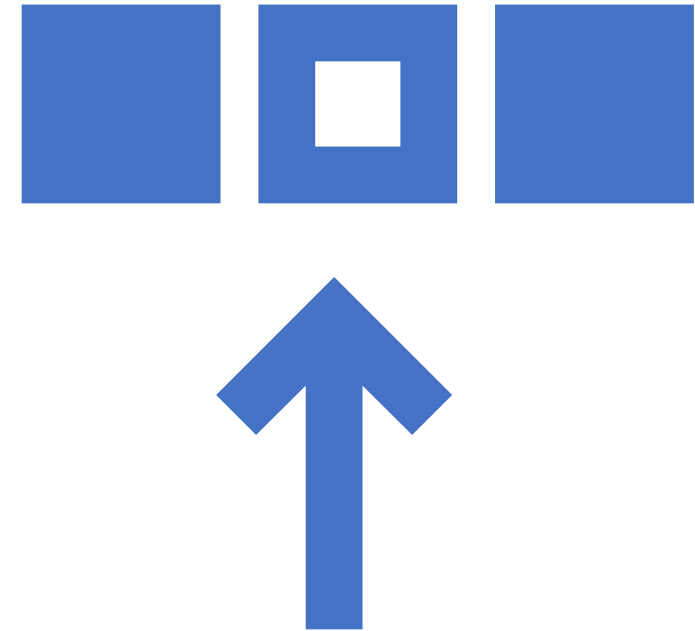


image 5

Let's add some unit tests

12. To enable mocking. We need to add nuget package Moq to the unit test project.
13. In solution explorer, under **TestingControllersSample.UnitTests** right click on **Dependencies**. Select **Manage Nuget Packages...**
14. Click **Browse**. Search for **Moq**. Click on Moq item. Click **Install** button to add the package to the project.
15. It should be now listed in the **Installed** tab for nuget packages manager.
16. Now, we need to add the reference to the Brainstorm project.
17. Under **TestingControllersSample.UnitTests**, right click on **Dependencies**. Select **Add Reference**. Under Projects, check the box for **TestingControllerSample**. Click **Ok**.



Let's add some unit tests

18. In solution explorer, under **TestingControllersSample.UnitTests** delete the default file **UnitTest1.cs**
19. Add a new class file to the project named **HomeControllerTests.cs**
20. In this file, we are going to add the first unit test method. Replace the content of the file with snippet 1. **See note.**
21. You can notice a few things:
 1. Name of the method describes the element to test and the goal of the test.
 2. The AAA pattern (Arrange, Act and Assert)
 3. We want to test only the Index method
22. Now, we need to add some code to the arrange part. See snippet 2

Note: find snippets in folder
workshop-testing/src/Snippets/unit tests

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Moq;
using TestingControllersSample.Controllers;
using TestingControllersSample.Core.Interfaces;
using TestingControllersSample.Core.Model;
using TestingControllersSample.ViewModels;
using Xunit;

namespace TestingControllersSample.UnitTests
{
    public class HomeControllerTests
    {
        [Fact]
        public async Task
Index_ReturnsAViewResult_WithAListOfBrainstormSessions()
        {
            //Arrange

            //Act

            //Assert
        }
    }
}
```

Snippet 1

Let's add some unit tests

- 23. This code mocks the repository object based on the interface `IBrainstormSessionRepository`. To control the behavior for the test operation, `ListAsync` method is mocked with the method `GetTestSessions`. See snippet 5 in next page.
- 24. The controller object is instantiated passing the mocked repository as a parameter. This adds a dependency with behavior expected for this test. Next step is **Act**. Add the code in snippet 3.
- 25. This code calls the method `Index` from the controller.
- 26. The next step is to check for the expected result.

```
//Snippet 2:

// Arrange
var mockRepo = new Mock<IBrainstormSessionRepository>();
mockRepo.Setup(repo => repo.ListAsync())
           .ReturnsAsync(GetTestSessions());

var controller = new HomeController(mockRepo.Object);
//

// Snippet 3:

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

Let's add some unit tests

- 27. Add snippet 4.
- 28. In this code, we are asserting the type of the result and the expected number of items in the model.
- 29. To complete this unit test, we need to add code for the method `GetTestSessions`. Use snippet 5 to add method to class.

```
// Snippet 4

// Assert
var viewResult = Assert.IsType<ViewResult>(result);
var model = Assert.IsAssignableFrom<IEnumerable<StormSessionViewModel>>(
    viewResult.ViewData.Model);

Assert.Equal(2, model.Count());
```

```
// Snippet 5

private List<BrainstormSession> GetTestSessions()
{
    var sessions = new List<BrainstormSession>();
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 2),
        Id = 1,
        Name = "Test One"
    });
    sessions.Add(new BrainstormSession()
    {
        DateCreated = new DateTime(2016, 7, 1),
        Id = 2,
        Name = "Test Two"
    })
    return sessions;
}
```


Let's run unit tests

30. Build the solution from Solution Explorer. Right click on **TestingControllersSample** and select **Rebuild Solution**.
31. From menu **Test > Select Test Explorer** (see **image 6**).
32. Right click on **Index_ReturnsAViewResult_WithAListOfBrainstormSessions** and select **Run**.
33. After the test runs successfully, it should look like **image 7**.

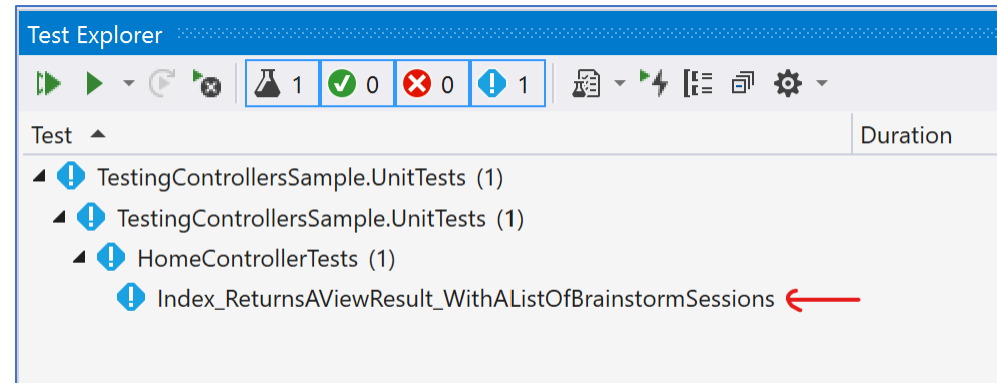


Image 6

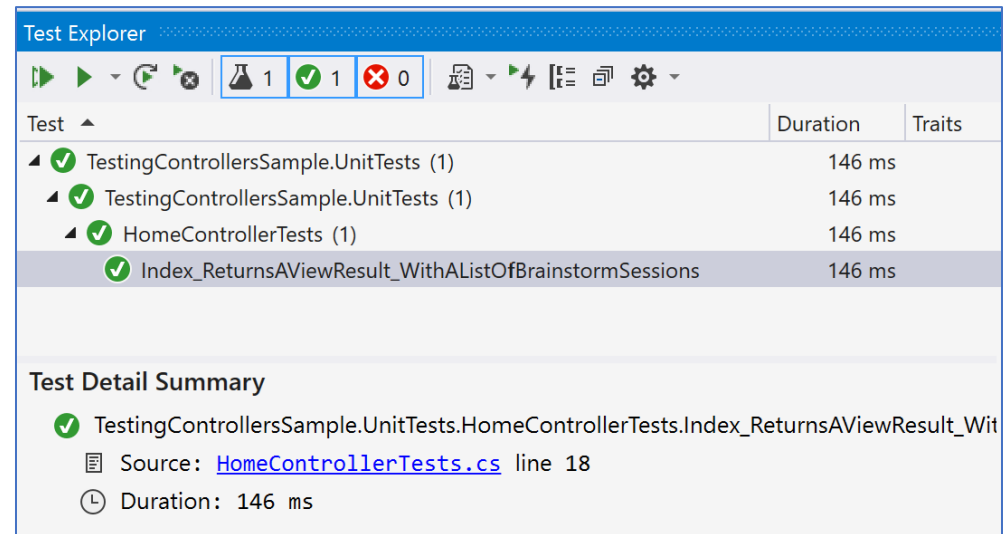


Image 7

Let's add more unit tests

- 34. Open file **HomeControllerTests.cs**.
- 35. Add another unit test method. Use **snippet 6**.
- 36. This method tests `BadRequestResult` is returned when the model is not valid.

```
// Snippet 6
[Fact]
public async Task IndexPost_ReturnsBadRequestResult_WhenModelStateIsInvalid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());

    var controller = new HomeController(mockRepo.Object);

    controller.ModelState.AddModelError("SessionName", "Required");

    var newSession = new HomeController.NewSessionModel();

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

    // Assert
    var badRequestResult =
        Assert.IsType<BadRequestObjectResult>(result);
    Assert.IsType<SerializableError>(badRequestResult.Value);
}
```

Let's add more unit tests

Snippet 7

37. Add another file with name **ApiIdeasControllerTests.cs** to add some tests for the api. Use **snippet 7**.
38. Build solution **TestingControllersSample**.
39. In **Test Explorer**, the new unit tests are now available to run.
40. Click **TestingControllersSample.UnitTests** node and use the tool bar to run all the unit tests available.

```
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Moq;
using TestingControllersSample.Api;
using TestingControllersSample.ClientModels;
using TestingControllersSample.Core.Interfaces;
using TestingControllersSample.Core.Model;
using Xunit;

namespace TestingControllersSample.Tests.UnitTests
{
    public class ApiIdeasControllerTests
    {
        [Fact]
        public async Task Create_ReturnsBadRequest_GivenInvalidModel()
        {
            // Arrange & Act
            var mockRepo = new Mock<IBrainstormSessionRepository>();
            var controller = new IdeasController(mockRepo.Object);
            controller.ModelState.AddModelError("error", "some error");

            // Act
            var result = await controller.Create(model: null);

            // Assert
            Assert.IsType<BadRequestObjectResult>(result);
        }

        [Fact]
        public async Task ForSession_ReturnsHttpNotFound_ForInvalidSession()
        {
            // Arrange
            int testSessionId = 123;
            var mockRepo = new Mock<IBrainstormSessionRepository>();
            mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
                .ReturnsAsync((BrainstormSession)null);

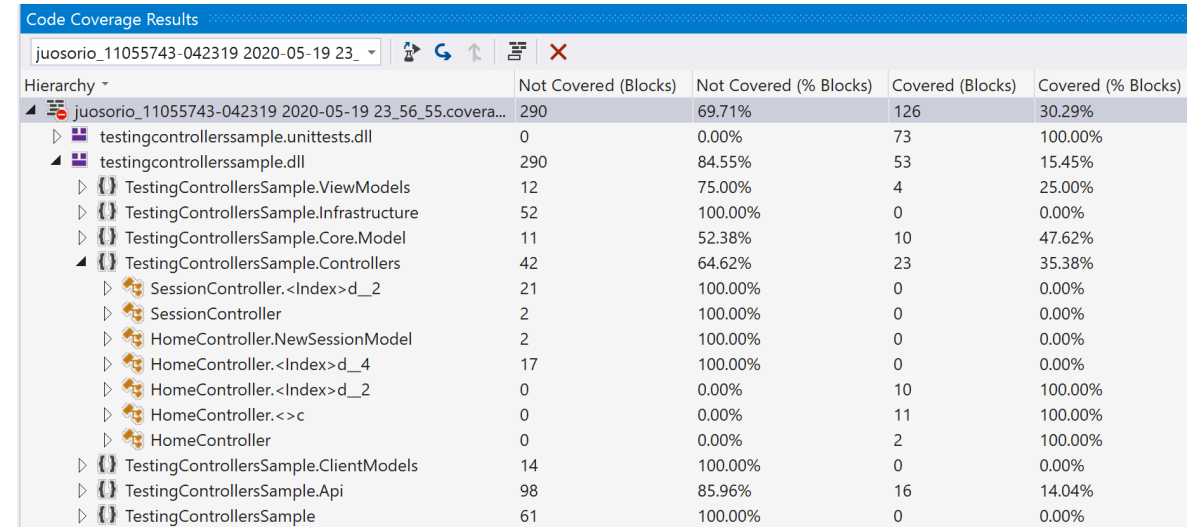
            var controller = new IdeasController(mockRepo.Object);

            // Act
            var result = await controller.ForSession(testSessionId);

            // Assert
            var notFoundObjectResult = Assert.IsType<NotFoundObjectResult>(result);
            Assert.Equal(testSessionId, notFoundObjectResult.Value);
        }
    }
}
```

Let's run Code Coverage

41. From **Test Explorer**, right click on **TestingControllersSample.UnitTests**. Select **Analyze Code Coverage**.
42. This action will start a build in the solution. After some seconds, the report is available in **Code Coverage Results** window. See image
43. *Optional*. Add your own unit test to increase the code coverage percentage.



| | Not Covered (Blocks) | Not Covered (% Blocks) | Covered (Blocks) | Covered (% Blocks) |
|--|----------------------|------------------------|------------------|--------------------|
| juosorio_11055743-042319 2020-05-19 23_56_55.covera... | 290 | 69.71% | 126 | 30.29% |
| testingcontrollerssample.unittests.dll | 0 | 0.00% | 73 | 100.00% |
| testingcontrollerssample.dll | 290 | 84.55% | 53 | 15.45% |
| TestingControllersSample.ViewModels | 12 | 75.00% | 4 | 25.00% |
| TestingControllersSample.Infrastructure | 52 | 100.00% | 0 | 0.00% |
| TestingControllersSample.Core.Model | 11 | 52.38% | 10 | 47.62% |
| TestingControllersSample.Controllers | 42 | 64.62% | 23 | 35.38% |
| SessionController.<Index>d__2 | 21 | 100.00% | 0 | 0.00% |
| SessionController | 2 | 100.00% | 0 | 0.00% |
| HomeController.NewSessionModel | 2 | 100.00% | 0 | 0.00% |
| HomeController.<Index>d__4 | 17 | 100.00% | 0 | 0.00% |
| HomeController.<Index>d__2 | 0 | 0.00% | 10 | 100.00% |
| HomeController.<>c | 0 | 0.00% | 11 | 100.00% |
| HomeController | 0 | 0.00% | 2 | 100.00% |
| TestingControllersSample.ClientModels | 14 | 100.00% | 0 | 0.00% |
| TestingControllersSample.Api | 98 | 85.96% | 16 | 14.04% |
| TestingControllersSample | 61 | 100.00% | 0 | 0.00% |

Image 8