Unit test controller logic in ASP.NET Core

11/07/2019 • 25 minutes to read • +11

In this article

By Steve Smith

Unit tests involve testing a part of an app in isolation from its infrastructure and dependencies. When unit testing controller logic, only the contents of a single action are tested, not the behavior of its dependencies or of the framework itself.

Unit testing controllers

Set up unit tests of controller actions to focus on the controller's behavior. A controller unit test avoids scenarios such as filters, routing, and model binding. Tests that cover the interactions among components that collectively respond to a request are handled by *integration tests*. For more information on integration tests, see Integration tests in ASP.NET Core.

If you're writing custom filters and routes, unit test them in isolation, not as part of tests on a particular controller action.

To demonstrate controller unit tests, review the following controller in the sample app.

View or download sample code (how to download)

The Home controller displays a list of brainstorming sessions and allows the creation of new brainstorming sessions with a POST request:

C# Copy

```csharp
public class HomeController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public HomeController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index()
    {
        var sessionList = await _sessionRepository.ListAsync();

        var model = sessionList.Select(session => new StormSessionViewModel()
        {
            Id = session.Id,
            DateCreated = session.DateCreated,
            Name = session.Name,
            IdeaCount = session.Ideas.Count
        });

        return View(model);
    }

    public class NewSessionModel
    {
        [Required]
        public string SessionName { get; set; }
    }

    [HttpPost]
    public async Task<IActionResult> Index(NewSessionModel model)
    {
        if (!ModelState.IsValid)
        {
            return BadRequest(ModelState);
        }
        else
        {
            await _sessionRepository.AddAsync(new BrainstormSession()
            {
                DateCreated = DateTimeOffset.Now,
                Name = model.SessionName
            });
        }

        return RedirectToAction(actionName: nameof(Index));
    }
}
```

The preceding controller:

- Follows the Explicit Dependencies Principle.
- Expects dependency injection (DI) to provide an instance of `IBrainstormSessionRepository`.
- Can be tested with a mocked `IBrainstormSessionRepository` service using a mock object framework, such as Moq. A *mocked object* is a fabricated object with a predetermined set of property and method behaviors used for testing. For more information, see Introduction to integration tests.

The `HTTP GET Index` method has no looping or branching and only calls one method. The unit test for this action:

- Mocks the `IBrainstormSessionRepository` service using the `GetTestSessions` method. `GetTestSessions` creates two mock brainstorm sessions with dates and session names.
- Executes the `Index` method.
- Makes assertions on the result returned by the method:
  - A ViewResult is returned.
  - The ViewDataDictionary.Model is a `StormSessionViewModel`.

- There are two brainstorming sessions stored in the `ViewDataDictionary.Model`.

C# Copy

```csharp
[Fact]
public async Task Index_ReturnsAViewResult_WithAListOfBrainstormSessions()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.ListAsync())
        .ReturnsAsync(GetTestSessions());
    var controller = new HomeController(mockRepo.Object);

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

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsAssignableFrom<IEnumerable<StormSessionViewModel>>(
        viewResult.ViewData.Model);
    Assert.Equal(2, model.Count());
}
```

C# Copy

```csharp
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;
    }
```

The Home controller's `HTTP POST Index` method tests verifies that:

- When ModelState.IsValid is `false`, the action method returns a *400 Bad Request* ViewResult with the appropriate data.
- When `ModelState.IsValid` is `true`:
  - The `Add` method on the repository is called.
  - A RedirectToActionResult is returned with the correct arguments.

An invalid model state is tested by adding errors using AddModelError as shown in the first test below:

C# Copy

```
[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);
}

[Fact]
public async Task IndexPost_ReturnsARedirectAndAddsSession_WhenModelStateIsValid()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.AddAsync(It.IsAny<BrainstormSession>()))
        .Returns(Task.CompletedTask)
        .Verifiable();
    var controller = new HomeController(mockRepo.Object);
    var newSession = new HomeController.NewSessionModel()
    {
        SessionName = "Test Name"
    };

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

    // Assert
    var redirectToActionResult = Assert.IsType<RedirectToActionResult>(result);
    Assert.Null(redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
    mockRepo.Verify();
}
```

When ModelState isn't valid, the same `ViewResult` is returned as for a GET request. The test doesn't attempt to pass in an invalid model. Passing an invalid model isn't a valid approach, since model binding isn't running (although an integration test does use model binding). In this case, model binding isn't tested. These unit tests are only testing the code in the action method.

The second test verifies that when the `ModelState` is valid:

- A new `BrainstormSession` is added (via the repository).
- The method returns a `RedirectToActionResult` with the expected properties.

Mocked calls that aren't called are normally ignored, but calling `Verifiable` at the end of the setup call allows mock validation in the test. This is performed with the call to `mockRepo.Verify`, which fails the test if the expected method wasn't called.

Note

The Moq library used in this sample makes it possible to mix verifiable, or "strict", mocks with non-verifiable mocks (also called "loose" mocks or stubs). Learn more about customizing Mock behavior with Moq.

SessionController in the sample app displays information related to a particular brainstorming session. The controller includes logic to deal with invalid `id` values (there are two `return` scenarios in the following example to cover these scenarios). The final `return` statement returns a new `StormSessionViewModel` to the view (*Controllers/SessionController.cs*):

C# Copy

```csharp
public class SessionController : Controller
{
    private readonly IBrainstormSessionRepository _sessionRepository;

    public SessionController(IBrainstormSessionRepository sessionRepository)
    {
        _sessionRepository = sessionRepository;
    }

    public async Task<IActionResult> Index(int? id)
    {
        if (!id.HasValue)
        {
            return RedirectToAction(actionName: nameof(Index),
                controllerName: "Home");
        }

        var session = await _sessionRepository.GetByIdAsync(id.Value);
        if (session == null)
        {
            return Content("Session not found.");
        }

        var viewModel = new StormSessionViewModel()
        {
            DateCreated = session.DateCreated,
            Name = session.Name,
            Id = session.Id
        };

        return View(viewModel);
    }
}
```

The unit tests include one test for each `return` scenario in the Session controller `Index` action:

C# Copy

```
[Fact]
public async Task IndexReturnsARedirectToIndexHomeWhenIdIsNull()
{
    // Arrange
    var controller = new SessionController(sessionRepository: null);

    // Act
    var result = await controller.Index(id: null);

    // Assert
    var redirectToActionResult =
        Assert.IsType<RedirectToActionResult>(result);
    Assert.Equal("Home", redirectToActionResult.ControllerName);
    Assert.Equal("Index", redirectToActionResult.ActionName);
}

[Fact]
public async Task IndexReturnsContentWithSessionNotFoundWhenSessionNotFound()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync((BrainstormSession)null);
    var controller = new SessionController(mockRepo.Object);

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

    // Assert
    var contentResult = Assert.IsType<ContentResult>(result);
    Assert.Equal("Session not found.", contentResult.Content);
}

[Fact]
public async Task IndexReturnsViewResultWithStormSessionViewModel()
{
    // Arrange
    int testSessionId = 1;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSessions().FirstOrDefault(
            s => s.Id == testSessionId));
    var controller = new SessionController(mockRepo.Object);

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

    // Assert
    var viewResult = Assert.IsType<ViewResult>(result);
    var model = Assert.IsType<StormSessionViewModel>(
        viewResult.ViewData.Model);
    Assert.Equal("Test One", model.Name);
    Assert.Equal(2, model.DateCreated.Day);
    Assert.Equal(testSessionId, model.Id);
}
```

Moving to the Ideas controller, the app exposes functionality as a web API on the `api/ideas` route:

- A list of ideas (`IdeaDTO`) associated with a brainstorming session is returned by the `ForSession` method.
- The `Create` method adds new ideas to a session.

C# Copy

```
[HttpGet("forsession/{sessionId}")]
public async Task<IActionResult> ForSession(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);
    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return Ok(result);
}
```

```
[HttpPost("create")]
public async Task<IActionResult> Create([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);
    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return Ok(session);
}
```

Avoid returning business domain entities directly via API calls. Domain entities:

- Often include more data than the client requires.
- Unnecessarily couple the app's internal domain model with the publicly exposed API.

Mapping between domain entities and the types returned to the client can be performed:

- Manually with a LINQ `Select`, as the sample app uses. For more information, see LINQ (Language Integrated Query).
- Automatically with a library, such as AutoMapper.

Next, the sample app demonstrates unit tests for the `Create` and `ForSession` API methods of the Ideas controller.

The sample app contains two `ForSession` tests. The first test determines if `ForSession` returns a NotFoundObjectResult (HTTP Not Found) for an invalid session:

C# Copy

```csharp
[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);
}
```

The second `ForSession` test determines if `ForSession` returns a list of session ideas (`<List<IdeaDTO>>`) for a valid session. The checks also examine the first idea to confirm its `Name` property is correct:

C# Copy

```
[Fact]
public async Task ForSession_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

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

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(okResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}
```

To test the behavior of the `Create` method when the `ModelState` is invalid, the sample app adds a model error to the controller as part of the test. Don't try to test model validation or model binding in unit tests—just test the action method's behavior when confronted with an invalid `ModelState`:

C# Copy

```
[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);
}
```

The second test of `Create` depends on the repository returning `null`, so the mock repository is configured to return `null`. There's no need to create a test database (in memory or otherwise) and construct a query that returns this result. The test can be accomplished in a single statement, as the sample code illustrates:

C# Copy

```
[Fact]
public async Task Create_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.Create(new NewIdeaModel());

    // Assert
    Assert.IsType<NotFoundObjectResult>(result);
}
```

The third `Create` test, `Create_ReturnsNewlyCreatedIdeaForSession`, verifies that the repository's `UpdateAsync` method is called. The mock is called with `Verifiable`, and the mocked repository's `Verify` method is called to confirm the verifiable method is executed. It's not the unit test's responsibility to ensure that the `UpdateAsync` method saved the data—that can be performed with an integration test.

C# Copy

```csharp
[Fact]
public async Task Create_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
    int testSessionId = 123;
    string testName = "test name";
    string testDescription = "test description";
    var testSession = GetTestSession();
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(testSession);
    var controller = new IdeasController(mockRepo.Object);

    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = testSessionId
    };
    mockRepo.Setup(repo => repo.UpdateAsync(testSession))
        .Returns(Task.CompletedTask)
        .Verifiable();

    // Act
    var result = await controller.Create(newIdea);

    // Assert
    var okResult = Assert.IsType<OkObjectResult>(result);
    var returnSession = Assert.IsType<BrainstormSession>(okResult.Value);
    mockRepo.Verify();
    Assert.Equal(2, returnSession.Ideas.Count());
    Assert.Equal(testName, returnSession.Ideas.LastOrDefault().Name);
    Assert.Equal(testDescription, returnSession.Ideas.LastOrDefault().Description);
}
```

Test ActionResult<T>

In ASP.NET Core 2.1 or later, ActionResult<T> (ActionResult<TValue>) enables you to return a type deriving from `ActionResult` or return a specific type.

The sample app includes a method that returns a `List<IdeaDTO>` for a given session `id`. If the session `id` doesn't exist, the controller returns NotFound:

C#

```csharp
[HttpGet("forsessionactionresult/{sessionId}")]
[ProducesResponseType(200)]
[ProducesResponseType(404)]
public async Task<ActionResult<List<IdeaDTO>>> ForSessionActionResult(int sessionId)
{
    var session = await _sessionRepository.GetByIdAsync(sessionId);

    if (session == null)
    {
        return NotFound(sessionId);
    }

    var result = session.Ideas.Select(idea => new IdeaDTO()
    {
        Id = idea.Id,
        Name = idea.Name,
        Description = idea.Description,
        DateCreated = idea.DateCreated
    }).ToList();

    return result;
}
```

Two tests of the `ForSessionActionResult` controller are included in the `ApiIdeasControllerTests`.

The first test confirms that the controller returns an `ActionResult` but not a nonexistent list of ideas for a nonexistent session `id`:

- The `ActionResult` type is `ActionResult<List<IdeaDTO>>`.
- The Result is a NotFoundObjectResult.

C# Copy

```csharp
[Fact]
public async Task ForSessionActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    var nonExistentSessionId = 999;

    // Act
    var result = await controller.ForSessionActionResult(nonExistentSessionId);

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}
```

For a valid session `id`, the second test confirms that the method returns:

- An `ActionResult` with a `List<IdeaDTO>` type.
- The ActionResult<T>.Value is a `List<IdeaDTO>` type.
- The first item in the list is a valid idea matching the idea stored in the mock session (obtained by calling `GetTestSession`).

C# Copy

```csharp
[Fact]
public async Task ForSessionActionResult_ReturnsIdeasForSession()
{
    // Arrange
    int testSessionId = 123;
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
        .ReturnsAsync(GetTestSession());
    var controller = new IdeasController(mockRepo.Object);

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

    // Assert
    var actionResult = Assert.IsType<ActionResult<List<IdeaDTO>>>(result);
    var returnValue = Assert.IsType<List<IdeaDTO>>(actionResult.Value);
    var idea = returnValue.FirstOrDefault();
    Assert.Equal("One", idea.Name);
}
```

The sample app also includes a method to create a new `Idea` for a given session. The controller returns:

- BadRequest for an invalid model.
- NotFound if the session doesn't exist.
- CreatedAtAction when the session is updated with the new idea.

C#

Copy

```
[HttpPost("createactionresult")]
[ProducesResponseType(201)]
[ProducesResponseType(400)]
[ProducesResponseType(404)]
public async Task<ActionResult<BrainstormSession>> CreateActionResult([FromBody]NewIdeaModel model)
{
    if (!ModelState.IsValid)
    {
        return BadRequest(ModelState);
    }

    var session = await _sessionRepository.GetByIdAsync(model.SessionId);

    if (session == null)
    {
        return NotFound(model.SessionId);
    }

    var idea = new Idea()
    {
        DateCreated = DateTimeOffset.Now,
        Description = model.Description,
        Name = model.Name
    };
    session.AddIdea(idea);

    await _sessionRepository.UpdateAsync(session);

    return CreatedAtAction(nameof(CreateActionResult), new { id = session.Id }, session);
}
```

Three tests of `CreateActionResult` are included in the `ApiIdeasControllerTests`.

The first text confirms that a BadRequest is returned for an invalid model.

C# Copy

```
[Fact]
public async Task CreateActionResult_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.CreateActionResult(model: null);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<BadRequestObjectResult>(actionResult.Result);
}
```

The second test checks that a NotFound is returned if the session doesn't exist.

C# Copy

```
[Fact]
public async Task CreateActionResult_ReturnsNotFoundObjectResultForNonexistentSession()
{
    // Arrange
    var nonExistentSessionId = 999;
    string testName = "test name";
    string testDescription = "test description";
    var mockRepo = new Mock<IBrainstormSessionRepository>();
    var controller = new IdeasController(mockRepo.Object);
    var newIdea = new NewIdeaModel()
    {
        Description = testDescription,
        Name = testName,
        SessionId = nonExistentSessionId
    };

    // Act
    var result = await controller.CreateActionResult(newIdea);

    // Assert
    var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
    Assert.IsType<NotFoundObjectResult>(actionResult.Result);
}
```

For a valid session `id`, the final test confirms that:

- The method returns an `ActionResult` with a `BrainstormSession` type.
- The ActionResult<T>.Result is a CreatedAtActionResult. `CreatedAtActionResult` is analogous to a *201 Created* response with a `Location` header.
- The ActionResult<T>.Value is a `BrainstormSession` type.
- The mock call to update the session, `UpdateAsync(testSession)`, was invoked. The `Verifiable` method call is checked by executing `mockRepo.Verify()` in the assertions.
- Two `Idea` objects are returned for the session.
- The last item (the `Idea` added by the mock call to `UpdateAsync`) matches the `newIdea` added to the session in the test.

C# Copy

```csharp
[Fact]
public async Task CreateActionResult_ReturnsNewlyCreatedIdeaForSession()
{
    // Arrange
```

```
        int testSessionId = 123;
        string testName = "test name";
        string testDescription = "test description";
        var testSession = GetTestSession();
        var mockRepo = new Mock<IBrainstormSessionRepository>();
        mockRepo.Setup(repo => repo.GetByIdAsync(testSessionId))
            .ReturnsAsync(testSession);
        var controller = new IdeasController(mockRepo.Object);

        var newIdea = new NewIdeaModel()
        {
            Description = testDescription,
            Name = testName,
            SessionId = testSessionId
        };
        mockRepo.Setup(repo => repo.UpdateAsync(testSession))
            .Returns(Task.CompletedTask)
            .Verifiable();

        // Act
        var result = await controller.CreateActionResult(newIdea);

        // Assert
        var actionResult = Assert.IsType<ActionResult<BrainstormSession>>(result);
        var createdAtActionResult = Assert.IsType<CreatedAtActionResult>(actionResult.Result);
        var returnValue = Assert.IsType<BrainstormSession>(createdAtActionResult.Value);
        mockRepo.Verify();
        Assert.Equal(2, returnValue.Ideas.Count());
        Assert.Equal(testName, returnValue.Ideas.LastOrDefault().Name);
        Assert.Equal(testDescription, returnValue.Ideas.LastOrDefault().Description);
    }
```

Additional resources

- Integration tests in ASP.NET Core
- Create and run unit tests with Visual Studio
- MyTested.AspNetCore.Mvc - Fluent Testing Library for ASP.NET Core MVC – Strongly-typed unit testing library, providing a fluent interface for testing MVC and web API apps. (*Not maintained or supported by Microsoft.*)