

Integration Testing ASP.NET Core Applications: Best Practices

INTRODUCING ASP.NET CORE INTEGRATION TESTS



Steve Gordon

MICROSOFT DEVELOPER TECHNOLOGIES MVP

@stevejgordon www.stevejgordon.co.uk



Overview



What are integration tests?

Unit tests vs. integration tests

Creating an integration test project

Writing an integration test

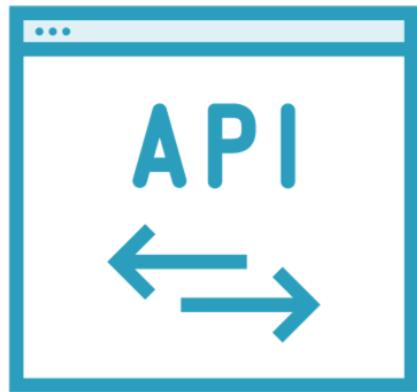
Running an integration test

- Visual Studio
- .NET CLI

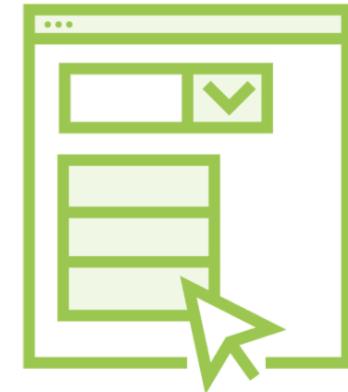
What is the WebApplicationFactory?



Later in This Course



Integration testing ASP.NET Core
web APIs



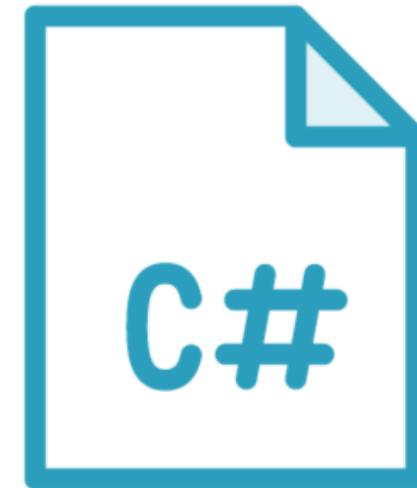
Integration testing ASP.NET Core
UI applications



Course Prerequisites



Some knowledge of ASP.NET Core



Experience with C#



Before We Begin



Follow along: Download the exercise files

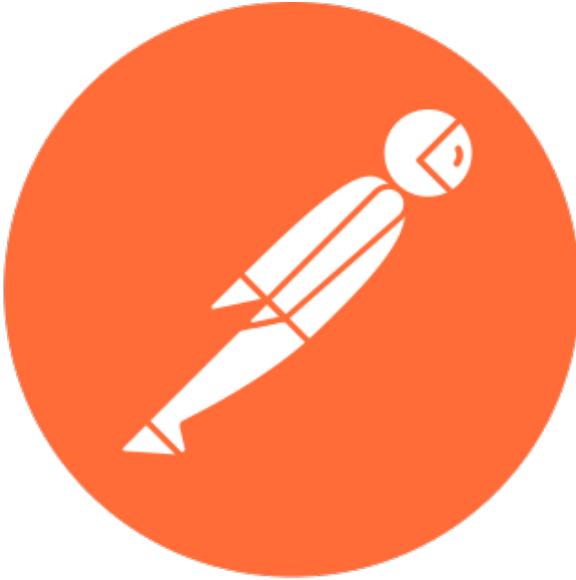


The solution requires the .NET Core 3.1 SDK



I'm using Visual Studio 2019 (16.5.x)





POSTMAN

postman.com



ASP.NET Core and .NET Core 3.1

Released: December 2019



ASP.NET Core and .NET Core 2.1

Released: May 2018



Let's Get Started



Running the Sample Application



Introducing Integration Tests



Testing Pyramid

Complex / Slower



More Expensive

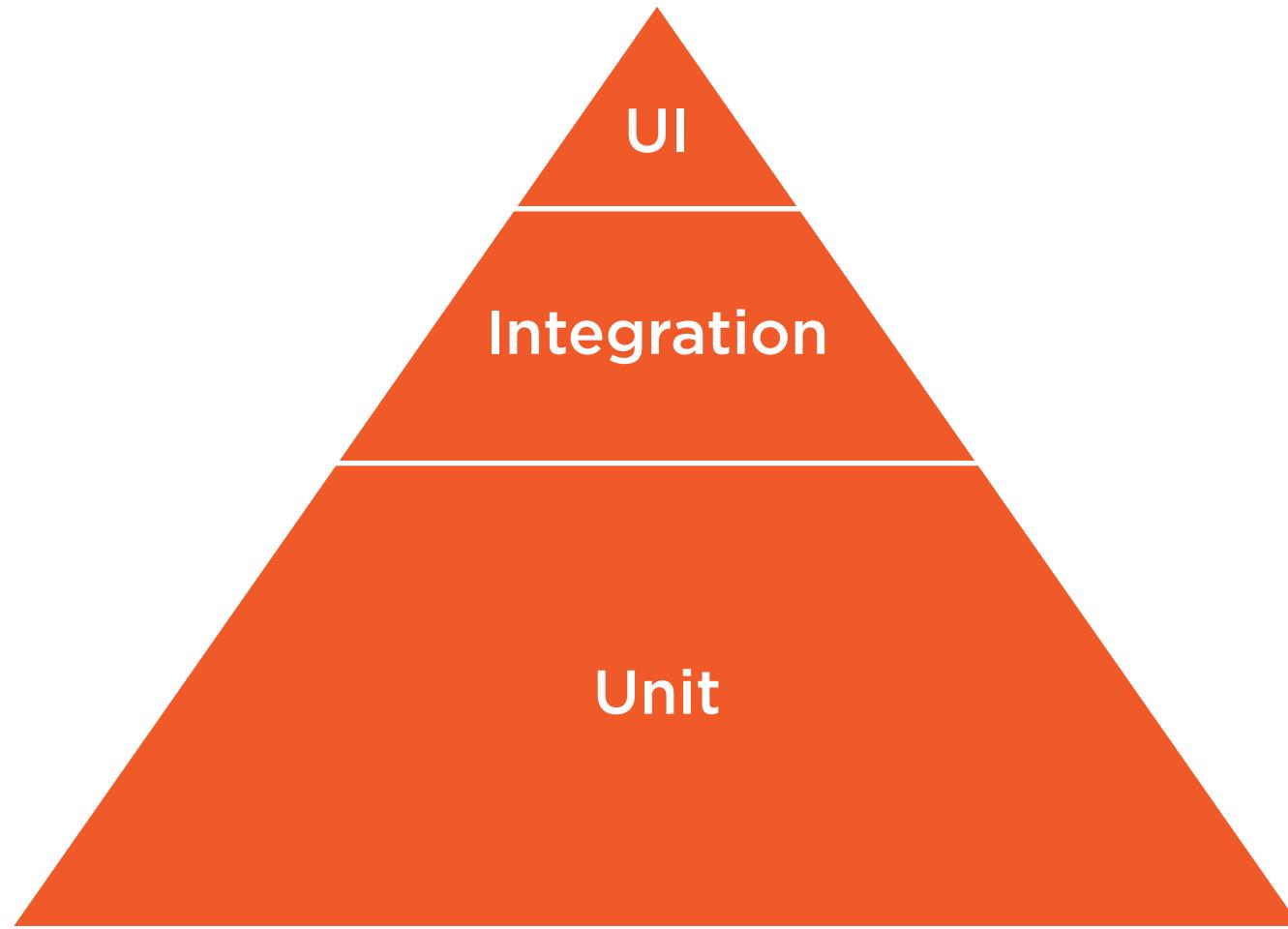


Simpler / Faster

Unit

Integration

UI



Testing Pyramid

Complex / Slower



More Expensive



Simpler / Faster

Unit

Integration

UI



Integration Tests



Test that multiple software components work together



Efficiently cover a large volume of overall functionality



Provide high level assurance of software quality



Unit Tests vs. Integration Tests

Unit Tests

Test a small piece of code (a unit) such as an individual method

Each test has a narrow scope

No dependencies outside of the code under test

Often rely on mocked dependencies

Generally lightweight and quick to run

Integration Tests

Provide an extra layer of testing above unit tests

Test multiple components working together, when integrated

Tests the application more broadly

Rely less on mocks or fakes, preferring to test the real components

Require more set up and teardown

May be slower to run



ASP.NET Core Integration Tests



Traditional Integration Tests



- Performed after development**
- Require deployment of the application**
- May require manual testing steps**
- May be partially or fully automated**



Example Scenario



Given an authenticated account holder

When visiting the statements page

**Then the account holder can view ONLY
their own statements**



Components Under Test



Authentication



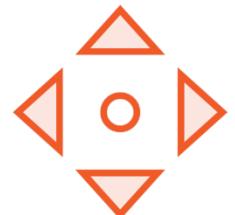
Database or document store



Authorization



Data access layer



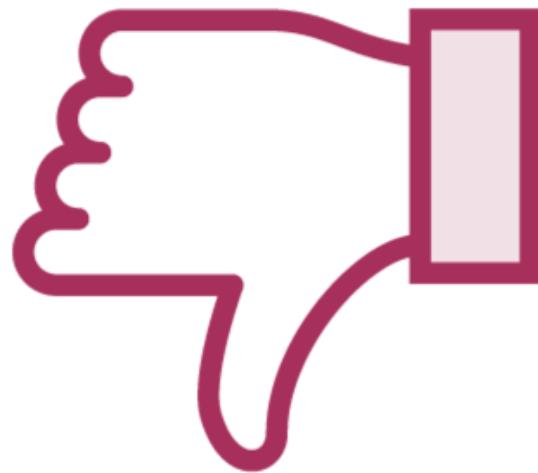
Routing



UI rendering

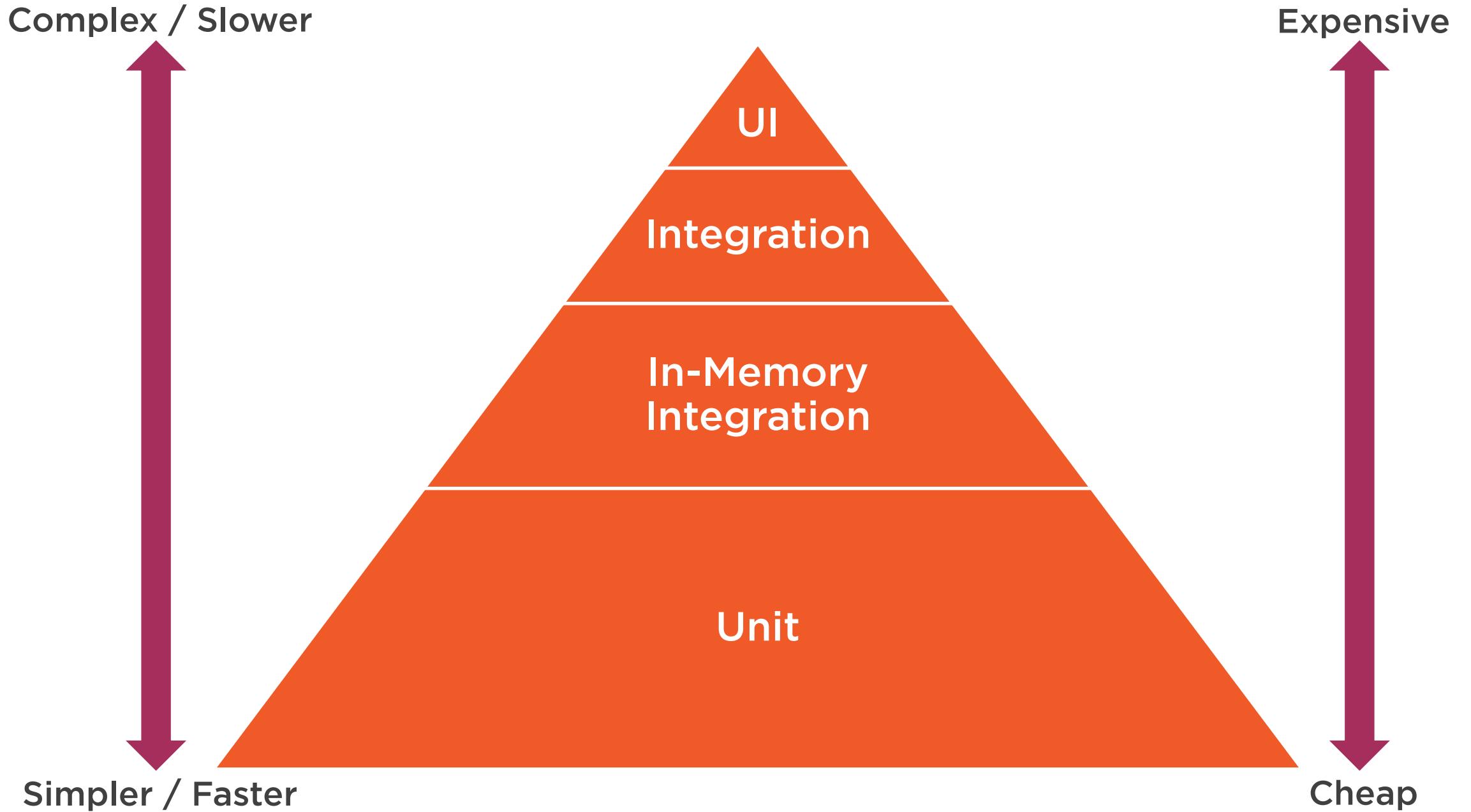


Disadvantages of Traditional Integration Tests

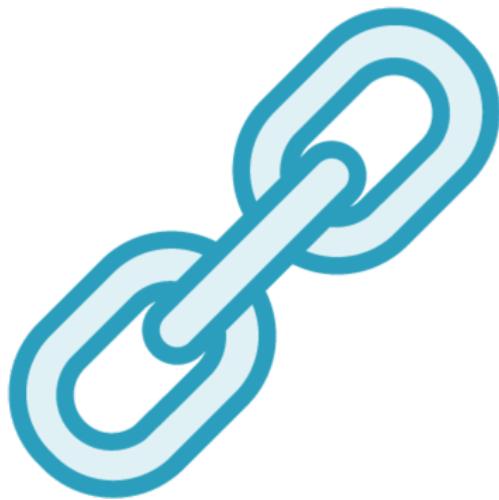


- May require a specialised team**
- Often run black-box**
- Identifying causes of failures is difficult**
- Require test environment and infrastructure**
- Slow to execute**
- Run late in the development lifecycle**
- Delay deploying features to production**





ASP.NET Core Integration Tests



Streamline end-to-end
testing of MVC, Razor
pages, and web API
projects



Host the application
in-memory, using a
TestServer



Library provides
support for easy set
up, teardown,
configuration, and
execution of tests



Advantages of ASP.NET Core Integration Tests



- Written by developers using familiar tools**
- Run often during the development lifecycle**
- Execute very quickly**
- Can be debugged**
- Can be used to apply TDD techniques**
- Require no extra infrastructure**
- Can be automated to run during CI/CD**



Components Under Test



Authentication



Database or document store



Authorization



Data access layer



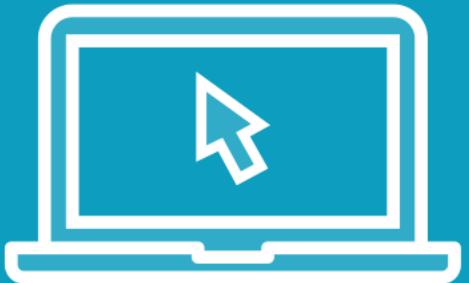
Routing



UI rendering



Demo

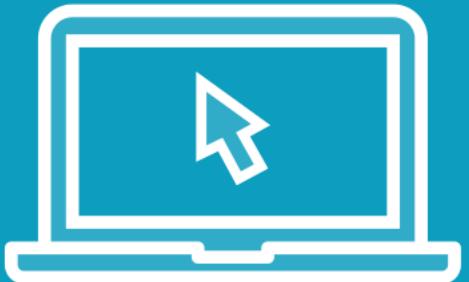


Create an integration test project

- Include required package reference
- Configure other prerequisites



Demo



Create an integration test

- Write a test which verifies that the API runs and a health check responds



Components Being Tested



Web API application starts



Server is running and can handle requests



Required services registered with the dependency injection container



Middleware pipeline is correctly configured



Routing sends requests to the expected endpoint



Recapping the Key Steps



Created a new project using the xUnit template



Updated the project to use the web SDK “Microsoft.NET.Sdk.Web”



Referenced Microsoft.AspNetCore.Mvc.Testing NuGet package



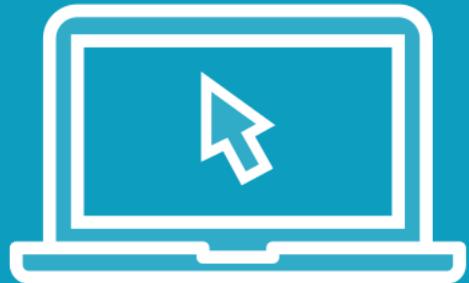
Disabled shadow copying using xunit.runner.json



Created a unit test class using the WebApplicationFactory class fixture



Demo



Run the integration test

- Visual Studio
- .NET CLI





.NET CLI

The .NET Command Line Interface (CLI) is a cross-platform toolchain for developing, building, running and publishing .NET Core applications.



What Is the WebApplicationFactory?



Integration Test Class

HealthcheckTests.cs

```
public class HealthcheckTests : IClassFixture<WebApplicationFactory<Startup>>
{
    private readonly HttpClient _client;

    public HealthcheckTests(WebApplicationFactory<Startup> factory)
    {
        _client = factory.CreateDefaultClient();
    }

    ...
}
```

Class Fixture

An xUnit feature used to create, set up and teardown a shared test class instance, used across all test methods defined in the test class.





By default, xUnit creates a new instance of a test class, for each test method

When using a Class Fixture:

- A single, shared instance is created
- The same test server is used by each test method in the class
- Once tests are complete, it will clean up by calling Dispose (if present)

More efficient when test set up or teardown is expensive

Improves the execution time of tests



Integration Test Class

HealthcheckTests.cs

```
public class HealthcheckTests : IClassFixture<WebApplicationFactory<Startup>>
{
    private readonly HttpClient _client;

    public HealthcheckTests(WebApplicationFactory<Startup> factory)
    {
        _client = factory.CreateDefaultClient();
    }

    ...
}
```



WebApplicationFactory

Bootstraps an application using an in-memory test server for functional, end-to-end integration testing.



WebApplicationFactory



Requires a generic argument, accepting a type from the application under test

- It is typical to use the Startup class

Builds and runs a host, configured to use a TestServer instance

A customized factory can be created via inheritance

- Override virtual methods to control the configuration of the host

Cleans up after all tests have executed



Integration Test Class

HealthcheckTests.cs

```
public class HealthcheckTests : IClassFixture<WebApplicationFactory<Startup>>
{
    private readonly HttpClient _client;

    public HealthcheckTests(WebApplicationFactory<Startup> factory)
    {
        _client = factory.CreateDefaultClient();
    }

    ...
}
```

Summary



- Learned about ASP.NET Core integration tests
- Described the benefits they offer
- Learned how they differ from traditional definitions of integration tests
- Created an integration test project
- Wrote an integration test
- Executed the integration test
- Understood the WebApplicationFactory



Up Next:

Writing Integration Tests for ASP.NET Core Web APIs:
Part 1

