

Setting Up Tests and Controlling Test Execution



Kevin Dockx

Architect

@Kevindockx | www.kevindockx.com



Coming Up



Setting up tests and sharing test context

- Constructor and dispose
- Class fixture
- Collection fixture

Integrating test context with ASP.NET
Core's dependency injection system



Coming Up



Categorizing tests

Skipping tests

Customizing test output



Setting Up Tests and Sharing Test Context



**Constructor and
dispose approach**



Constructor and Dispose

**Set up test context in the constructor,
potentially clean up in Dispose method**

- Context is recreated for each test



Setting Up Tests and Sharing Test Context



**Constructor and
dispose approach**



Class fixture approach



Class Fixture

Create a single test context shared among all tests in the class

- Context is cleaned up after all tests in the class have finished

Use when context creation and clean-up is expensive



Class Fixture

Don't let a test depend on changes made to the context by other tests

- Test must remain isolated
- You don't have control over the order in which tests are run



Setting Up Tests and Sharing Test Context



**Constructor and
dispose approach**



Class fixture approach



**Collection fixture
approach**



Collection Fixture

Create a single test context shared among tests in several test classes

- Context is cleaned up after all tests across classes have finished

Use when context creation and clean-up is expensive



Demo



**Sharing context with the constructor
and dispose approach**



Demo



Sharing context with the class fixture approach



Demo



Sharing context with the collection fixture approach



Integrating Test Context With ASP.NET Core's Dependency Injection System

In ASP.NET Core, dependencies are often resolved via the built-in IoC container

- Can that be integrated with a unit test?**



Integrating Test Context With ASP.NET Core's Dependency Injection System

Newing up dependencies is the preferred approach

- Simple, fast, concise

You might want to integrate with the DI system

- If the class has got a lot of dependencies
- If the dependency tree is large



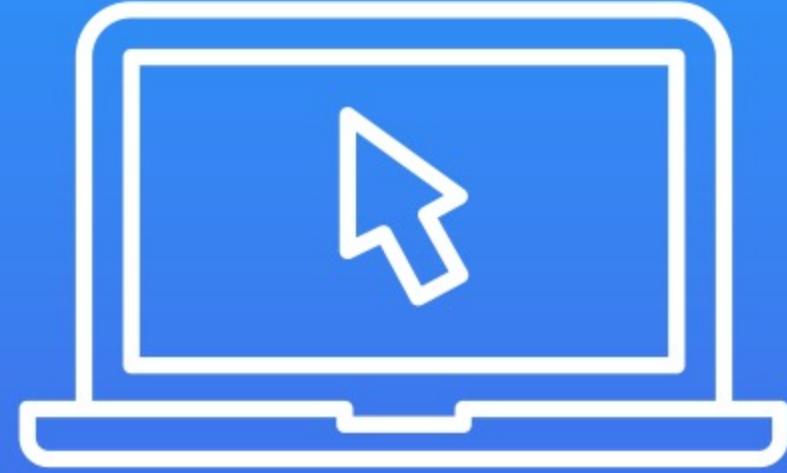
Demo



**Integrating test context with the ASP.NET
Core dependency injection system**



Demo



Categorizing and running subsets of tests



Demo



Skipping tests



Demo



Adding additional test output



Summary



Approaches for sharing test context

- Constructor and dispose
- Class fixture
- Collection fixture

Integrating test context with ASP.NET Core's dependency injection system



Summary



Use `[Trait]` to categorize tests

Use the `Skip` property on `[Fact]` to skip tests

Use `ITestOutputHelper` to log additional diagnostics info



Up Next:

Working with Data-driven Tests

