Testing Components Using bUnit

Gill Cleeren

CTO Xpirit Belgium

@gillcleeren – xpirit.com/gill

Module overview

Understanding unit tests

Writing tests with bUnit

Understanding Unit Tests

A unit test is an automated piece of code that invokes a unit of work in the system and then checks a single assumption about the behavior of that unit of work.

From The Art of Unit Testing, Roy Osherove

Unit Tests

Block of code

Public methods

Isolated and independent

Using Unit Tests

Consistent

Automated

Fast

Why Do We Need Unit Tests?

Find bugs

Change without fear of breaking something

Improve quality

Documentation of code

Parts of a unit test

- Arrange
- Act
- Assert

Commonly used frameworks

- xUnit, Nunit, MSTest
- Moq

A Simple Unit Test

```
public void CanUpdatePiePrice()
{
    // Arrange
    var pie =
        new Pie { Name = "Sample pie", Price = 12.95M };

    // Act
    pie.Price = 20M;

    //Assert
    Assert.Equal(20M, pie.Price);
}
```

Writing Tests with bUnit

Introducing bUnit

Unit testing library for Blazor components

Relies on xUnit (or other) unit testing framework

Not a UI testing library!

Features of bUnit

Testing using C# or Razor files

Semantic HTML comparer

Interacting with components and triggering events

Passing parameters and injecting services

Creating a bUnit Project

bUnit Template

Manual

Demo

Creating the test project in the solution

Writing Tests

*.cs files

*.razor files

```html
<h1>Hello Pluralsight</h1>
```

Sample Component to Test

Component name: HelloPluralsight.razor

bUnit Testing Code

```
@code
{
  [Fact]
  public void HelloWorldComponentRendersCorrectly()
  {
    //Arrange
    using var ctx = new TestContext();

    //Act
    var cut = ctx.Render(@<HelloPluralsight />);

    //Assert
    cut.MarkupMatches(@<h1>Hello Pluralsight</h1>);
  }
}
```

Demo

Testing our components

More Advanced Testing Scenarios

Event handlers

IJSRuntime emulation

Faking authorization

Mocked HttpClient

Demo

Testing components with injected services

Summary

bUnit offers a simple but powerful way to test components

Razor-based tests offer the easiest approach

Up next:

Sharing code between Blazor client and server