

**Blazor Testing**

**Course Number:** ASPNC-118
**Duration:** 2 days

**Overview**

This Blazor training course teaches attendees the fundamental principles and best practices for testing Blazor applications. Participants learn to organize and execute tests using C# and JavaScript specifically for Blazor applications, including isolated testing with mocks, stubs, and fakes.

The course focuses on programming and executing unit and integration tests for Razor Components, as well as E2E tests for Blazor web pages to enhance the robustness and reliability of Blazor applications.

**Prerequisites**

Students need HTML, CSS, JavaScript, and C# programming experience. Prior experience with Blazor is required. If your attendees do not yet have this expertise, please let us know so that we can customize your class to cover the necessary prerequisite(s).

**Materials**

All students receive comprehensive courseware covering all topics in the course. Courseware is distributed via GitHub through documentation and extensive code samples. Students practice the topics covered through challenging hands-on lab exercises.

**Software Needed on Each Student PC**

Students will need a free, personal GitHub account to access the courseware. This course can be taught with Visual Studio, Visual Studio Code, or JetBrains Rider. Students will need permission to install the selected IDE on their computers. Also, students will need permission to install NuGet Packages. If students are unable to configure a local environment, a cloud-based environment can be provided.

**Objectives**

* Learn the principles and practices of testing Blazor applications
* Review the differences between unit, integration, and end-to-end (E2E) testing
* Practice how to organize C# and JavaScript tests
* Understand what needs to be tested for Blazor Applications
* Enable testing in isolation with mocks, stubs, and fakes
* Program and run unit and integration tests for Razor Components
* Program and run E2E tests for Blazor web pages

**Outline**

* Introduction
* Kinds of Testing
	+ Unit Tests
	+ Integration Tests
	+ E2E Testing
	+ Automated vs. Manual Testing
	+ Testing & DevOps
* Testing Parts
	+ Tests
	+ Test Suites
	+ Assertions
	+ Setup/Teardown
	+ Mocks, Fakes, Stubs
	+ Arrange, Act, Assert
	+ Test Frameworks
	+ Test Runners
	+ Code Coverage
* Overview of .NET Core and Testing
	+ XUnit
	+ Test Runners:  Command-Line, Visual Studio, Visual Studio Code, JetBrains Rider
	+ Testing with XUnit and the Selected IDE (see above)
	+ Mocking with Moq
	+ Fluent Assertions
	+ BUnit
* Test C# and Razor code with xUnit
	+ What is xUnit?
	+ Testing Framework
	+ Test Parallelism
	+ Shared Test Context
	+ Facts vs. Theory
	+ Assertions
	+ Integration with Visual Studio
	+ Debugging Unit Tests in Visual Studio
	+ Debugging Unit Tests in Visual Studio Code
* Razor Component Unit and Integration Testing
	+ What Should be tested on a Razor Component?
	+ What is bUnit?
	+ Using bUnit with xUnit
	+ Setup and define components under tests in C# or Razor syntax
	+ Verify outcome using semantic HTML comparer
	+ Interact with and inspect components
	+ Trigger event handlers
	+ Provide cascading values
	+ Inject services
	+ Mock IJsRuntime
	+ Perform snapshot testing
* JavaScript Testing
	+ Is JavaScript Unit Testing a Thing?
	+ Benefits of Unit Testing JavaScript
	+ Challenges of Unit Testing JavaScript
	+ Testing JavaScript with Jest
	+ Test Suites
	+ Tests
	+ Assertions
	+ Mocks and Spies
* JavaScript Testing within a Blazor Project
	+ Configure Unit Testing for a JavaScript/TypeScript project
	+ Organize Code for Efficient Testing
	+ Mocking the DOM
	+ Testing DOM Manipulation Code
	+ Testing AJAX Code
	+ Running Tests
	+ Debugging Tests
* E2E Testing Overview
	+ What is End-To-End Testing?
	+ Selenium WebDriver
	+ Program and Run E2E tests with C#
	+ Finding Elements in the DOM Tree
	+ Page Objects
* Conclusion