CLOUD CONTRAPTIONS
COMPUTER PROGRAMMING & TRAINING

To discuss this course and customizations:
Call: 434-509-5680 or Email: sales@cloudcontraptions.com

Test-Driven Development with .NET 8 Blazor and Web API

Duration

5 days

Description

Join our comprehensive course and become a proficient practitioner of Test-Driven Development (TDD). You'll not only learn the principles and best practices of TDD but also apply them to ASP.NET Core 8 and JavaScript/TypeScript code, gaining hands-on experience. Dive into the intricacies of unit testing and end-to-end (E2E) testing, honing your skills in organizing C# and JavaScript unit tests effectively. Understand the nuances of testing for Blazor WebAssembly and ASP.NET Core Web APIs applications, ensuring robust and reliable code.

Discover the art of testing in isolation with mocks, stubs, and fakes, enabling you to create thorough and accurate tests. Master the craft of unit testing for both client-side Blazor WebAssembly and server-side Web APIs code, ensuring the quality of your applications. Finally, learn how to seamlessly integrate testing into your local development tools and CI/CD pipelines, whether using Azure DevOps or GitHub, to streamline your development process and deliver high-quality software. Join us on this journey to becoming a skilled TDD practitioner in the world of web development.

Objectives

- Learn the principle and best practices of Test-Driven Development
- Apply TDD in ASP.NET and JavaScript/TypeScript code
- Explore the parts of unit testing and E2E testing
- Practice how to organize C# and JavaScript unit tests
- Understand what needs to be tested for Blazor and Web API Applications
- Enable testing in isolation with mocks, stubs, and fakes
- Unit test client-side Blazor code
- Unit test server-side Web API code
- Integrate testing into local development tools and CI/CD pipelines (Azure DevOps or GitHub)

Prerequisites

Students need HTML, CSS, JavaScript, and C# programming experience. Experience with Blazor and Web APIs is highly recommended.

Training Materials

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

Software Requirements

Students will need a free, personal GitHub account to access the courseware. Student will need permission to install Visual Studio 2022 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.

Outline

- Introduction
  - What is Test-Driven Development?
  - Benefits of TDD
  - Challenges of TDD
- Principles of TDD
  - Three Laws of TDD
  - Clean Tests
  - One Assert Per Test
  - Five Rules: FIRST
    - Fast
    - Independent
    - Repeatable
    - Self-Validating
    - Timely
  - Red, Green, Refactor Technique
- 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
  - Testing Frameworks
    - MSTest
    - NUnit
    - XUnit
  - Test Runners
    - Command-Line
    - Visual Studio
    - Visual Studio Code
  - Testing Libraries
    - Mocking with Moq
    - Fluent Assertions
    - BUnit
- 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

- ASP.NET Web API Test-Driven Development
  - What Should be Tested on an MVC application?
  - Implementing a REST Service with a Web API
  - Integrate Test Projects into a Solution
  - Testing Controllers
  - Testing APIs
  - Integration Testing of APIs
- ASP.NET Blazor Test-Driven Development
  - 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
- Mocking Databases (choose 1)
  - Entity Framework
  - Dapper
- JavaScript Test-Driven Development
  - Is JavaScript Unit Testing a Thing?
  - Benefits of Unit Testing JavaScript
  - Challenges of Unit Testing JavaScript
- JavaScript Testing Frameworks Overview
  - Jest
  - Jasmine
  - Mocha
- JavaScript Test Runners Overview
  - Jest
  - Karma (deprecated but will cover for legacy projects)
- JavaScript E2E Testing Overview
  - What is End-To-End Testing?

  - Selenium WebDriver
  - Cypress
- JavaScript TDD with BDD
  - What is Behavior-Driven Development?
  - Coding Unit Tests around Behavior vs Functions
  - Given-When-Then
- JavaScript Unit Testing (will be covered with Jest or Jasmine)
  - Test Suites
  - Tests
  - Assertions
  - Mocks and Spies
- JavaScript E2E Testing (will be covered with Selenium)
  - Finding Elements in the DOM Tree
  - Page Objects
- Testing JavaScript Code Frameworks
  - Plain Vanilla JavaScript (can use Jest or Jasmine)
    - 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
- Code Coverage
  - What is Code Coverage?
  - What can be understood from Code Coverage?
  - What are the limitations of Code Coverage?
  - Generate Code Coverage reports for ASP.NET C# code
  - Generate Code Coverage reports for JavaScript code
  - Integrate code coverage into CI/CD systems (Azure DevOps or GitHub Actions)