protechtraining.com

Full Stack Web Programming with Blazor WebAssembly and ASP.NET Core Web API

Course Summary

Description

This Full Stack Web Programming with Blazor WebAssembly and ASP.NET Core Web API training teaches you how to build UI apps using the same component-based patterns popularized by libraries such as Angular and React, but with C#. Attendees also learn server-side coding using ASP.NET Core Web APIs and SignalR to provide data for their Blazor WebAssembly applications.

Objectives

After taking this course, students will be able to:

- Understand the Blazor platform
- Build UIs with components
- Utilize data binding and event handling
- Compose components
- Build Blazor pages and configure routing
- Deploy a Blazor WebAssembly application to production
- Consume Server Data via REST APIs and SignalR (WebSockets)
- Unit test Blazor apps
- Unit test server-side code

Topics

- Introduction
- Blazor WebAssembly Application
- Razor Components and Data Binding
- Composing Razor Components
- Razor Component Forms
- Razor Component Pages
- Using Server Data
- Interacting with JavaScript
- Unit Testing

Audience

This course is designed for students withing to learn how to build UI apps using the same component-based patterns popularized by libraries such as Angular and React, but with C#.

Prerequisites

Before taking this course, students should have C# programming experience, and HTML, CSS, and JavaScript development experience.

Duration

Five days

Full Stack Web Programming with Blazor WebAssembly and ASP.NET Core Web API

Course Outline

I. *Introduction*
   A. What is Blazor?
   B. Blazor Hosting Models
   C. Blazor Server vs. Blazor WebAssembly
   D. What is WebAssembly?
   E. Browser Compatibility
   F. WebAssembly vs. JavaScript
   G. How does .NET Core / C# run in a web browser?

II. *Blazor WebAssembly Application*
   A. Project Template
   B. Create a New Application
   C. Hosting Blazor WebAssembly with a ASP.NET Core MVC Server
   D. Configuration
   E. Dependency Injection
   F. Environments
   G. Logging
   H. Handling Errors
   I. Debugging WebAssembly

III. *Razor Components and Data Binding*
   A. What is a Component?
   B. Creating a Data Model
   C. Binding the Data Model to the HTML
   D. Passing Arbitrary Attributes
   E. Handling Events
   F. Manually Trigger State Updates and Re-rendering

IV. *Composing Razor Components*
   A. Decompose a Component into Smaller Components
   B. One-Way Data Binding
   C. Two-Way Data Binding
   D. Pass Data from a Parent Component to a Child Component using Parameters
   E. Pass Data from a Child Component to a Parent Component using Event Callbacks
   F. Use Keys to Optimize Performance
   G. Use Refs to Access DOM Elements
   H. Razor Component Libraries
   I. Razor Component Design Patterns
      1. Parameters are Immutable
      2. Lift State Up
      3. Managing State in General

V. *Razor Component Forms*
   A. What is the purpose of Form?
   B. Collecting Data using a Form, Input, Select, and TextArea Elements
   C. Explore Form Element Two-Data Binding
   D. Build Forms with the Blazor Edit Form Razor Component
   E. Explore the Concept of the Edit Context
   F. Use the Specialized Edit Form Controls
      1. Input Text
      2. Input TextArea
      3. Input Select
      4. Input Number
      5. Input Checkbox
      6. Input Date
   G. Applying Validation to the Form
   H. Decorating the View Model with Validation Attributes
   I. Code Custom Validation Attributes

VI. *Razor Component Pages*
   A. What is the Page model?
   B. Differences between Razor Pages and Razor Components
   C. Using a Razor Component as a Page
   D. Explore the Router Component
   E. Configuring Page Routing
   F. Route to Components from Multiple Assemblies
   G. Using Route Parameters
   H. Using the Query String
   I. Applying Authorization to a Razor Component Page
   J. Using Authorization within the Component Tree

Course Outline (cont'd)

VII. Using Server Data

A. ASP.NET Core MVC Web API
   1. What is ASP.NET Core MVC?
   2. What is a REST API?
   3. What is an API Controller?
   4. Injecting the Http Client
   5. Exploring the Http Client
   6. Calling a REST API from a Blazor Component using the HttpClient
   7. Build a REST API with ASP.NET Core MVC
   8. Implementing Authentication and Authorization

B. SignalR
   1. What is SignalR?
   2. What are Web Sockets?
   3. Understand Two-Way Data Flow with SignalR
   4. Use SignalR to communicate between Razor Components and ASP.NET Core server

VIII. Interacting with JavaScript

A. What is the JavaScript Interop?
B. When is JavaScript needed?
C. Synchronous vs. Asynchronous Calls
D. How to call a JavaScript function from a Component
E. How to call C# code from JavaScript
F. Calling Static Methods
G. Calling Instance Methods
H. Organizing JavaScript Code within a Blazor WebAssembly App
I. Explore JavaScript Ecosystem
J. Client-Side Libraries
K. NPM & Yarn
L. Webpack
M. Useful Libraries

IX. Unit Testing

A. What is Unit Testing?

B. Principles of Unit Testing
   1. Defining a Unit
   2. Setup/Teardown
   3. Testing in Isolation
   4. Determining What to Test
   5. Code Coverage
   6. Test Frameworks
   7. Stubs, Mocks and Spies

C. xUnit
   1. What is xUnit?
   2. Testing Framework
   3. Facts vs. Theory
   4. Assertions
   5. Integration with Visual Studio

D. Razor Components
   1. What Should be Tested on a Razor Component?
   2. What is bUnit?
   3. Using bUnit with xUnit
   4. Setup and define components under tests in C# or Razor syntax
   5. Verify outcome using semantic HTML comparer
   6. Interact with and inspect components
   7. Trigger event handlers
   8. Provide cascading values
   9. Inject services
   10. Mock IJsRuntime
   11. Perform snapshot testing

E. ASP.NET Core Web API
   1. What Should be Tested on a Web API?
   2. Testing Controllers
   3. Testing APIs
   4. Integration Testing of APIs