

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

**Course Number:** ASPNC-112
**Duration:** 5 days

**Overview**

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.

**Prerequisites**

* C# programming experience
* HTML, CSS, and JavaScript development experience

**Materials**

All Blazor training students receive comprehensive courseware.

**Software Needed on Each Student PC**

* Windows 10 or later with at least 8 GB RAM
* Visual Studio 2019 or later
* .NET Core 3.1 or later SDK
* LocalDB or another version of SQL Server
* Postman application
* Additional lab files that Accelebrate provides

**Objectives**

* 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

**Outline**

* Introduction
	+ What is Blazor?
	+ Blazor Hosting Models
	+ Blazor Server vs. Blazor WebAssembly
	+ What is WebAssembly?
	+ Browser Compatibility
	+ WebAssembly vs. JavaScript
	+ How does .NET Core / C# run in a web browser?
* Blazor WebAssembly Application
	+ Project Template
	+ Create a New Application
	+ Hosting Blazor WebAssembly with a ASP.NET Core MVC Server
	+ Configuration
	+ Dependency Injection
	+ Environments
	+ Logging
	+ Handling Errors
	+ Debugging WebAssembly
* Razor Components and Data Binding
	+ What is a Component?
	+ Creating a Data Model
	+ Binding the Data Model to the HTML
	+ Passing Arbitrary Attributes
	+ Handling Events
	+ Manually Trigger State Updates and Re-rendering
* Composing Razor Components
	+ Decompose a Component into Smaller Components
	+ One-Way Data Binding
	+ Two-Way Data Binding
	+ Pass Data from a Parent Component to a Child Component using Parameters
	+ Pass Data from a Child Component to a Parent Component using Event Callbacks
	+ Use Keys to Optimize Performance
	+ Use Refs to Access DOM Elements
	+ Razor Component Libraries
	+ Razor Component Design Patterns
		- Parameters are Immutable
		- Lift State Up
		- Managing State in General
* Razor Component Forms
	+ What is the purpose of Form?
	+ Collecting Data using a Form, Input, Select, and TextArea Elements
	+ Explore Form Element Two-Data Binding
	+ Build Forms with the Blazor Edit Form Razor Component
	+ Explore the Concept of the Edit Context
	+ Use the Specialized Edit Form Controls
		- Input Text
		- Input TextArea
		- Input Select
		- Input Number
		- Input Checkbox
		- Input Date
	+ Applying Validation to the Form
	+ Decorating the View Model with Validation Attributes
	+ Code Custom Validation Attributes
* Razor Component Pages
	+ What is the Page model?
	+ Differences between Razor Pages and Razor Components
	+ Using a Razor Component as a Page
	+ Explore the Router Component
	+ Configuring Page Routing
	+ Route to Components from Multiple Assemblies
	+ Using Route Parameters
	+ Using the Query String
	+ Applying Authorization to a Razor Component Page
	+ Using Authorization within the Component Tree
* Using Server Data
	+ ASP.NET Core MVC Web API
		- What is ASP.NET Core MVC?
		- What is a REST API?
		- What is an API Controller?
		- Injecting the Http Client
		- Exploring the Http Client
		- Calling a REST API from a Blazor Component using the HttpClient
		- Build a REST API with ASP.NET Core MVC
		- Implementing Authentication and Authorization
	+ SignalR
		- What is SignalR?
		- What are Web Sockets?
		- Understand Two-Way Data Flow with SignalR
		- Use SignalR to communicate between Razor Components and ASP.NET Core server
* Interacting with JavaScript
	+ What is the JavaScript Interop?
	+ When is JavaScript needed?
	+ Synchronous vs. Asynchronous Calls
	+ How to call a JavaScript function from a Component
	+ How to call C# code from JavaScript
	+ Calling Static Methods
	+ Calling Instance Methods
	+ Organizing JavaScript Code within a Blazor WebAssembly App
	+ Explore JavaScript Ecosystem
		- Client-Side Libraries
		- NPM & Yarn
		- Webpack
		- Useful Libraries
* Unit Testing
	+ What is Unit Testing?
	+ Principles of Unit Testing
		- Defining a Unit
		- Setup/Teardown
		- Testing in Isolation
		- Determining What to Test
		- Code Coverage
		- Test Frameworks
		- Stubs, Mocks and Spies
	+ xUnit
		- What is xUnit?
		- Testing Framework
		- Facts vs. Theory
		- Assertions
		- Integration with Visual Studio
	+ Razor Components
		- 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
	+ ASP.NET Core Web API
		- What Should be Tested on a Web API?
		- Testing Controllers
		- Testing APIs
		- Integration Testing of APIs
* Conclusion