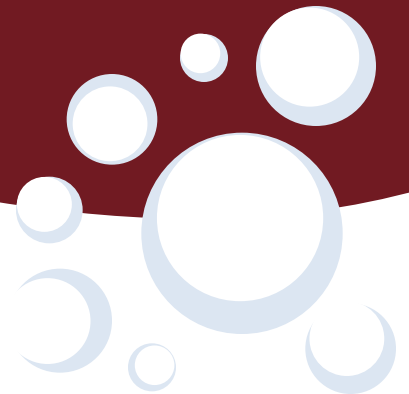


Modern Full-Stack with .NET 8 and JavaScript



Modern Full-Stack with .NET 8 and JavaScript


Agenda


- Introduction
- Overview of Project Types
- Modern C#
- Multi-Project Solutions
- Modern JavaScript
- JavaScript Frameworks
- Hands-On Case-Study Project

Modern Full-Stack with .NET 8 and JavaScript

Introduction

- Evolution of the .NET Platform
- .NET SDKs and Runtimes
- IDE Choices

- 
- 2002 .NET 1.0, C# 1.0, Visual Studio .NET
 - 2003 .NET 1.1, Visual Studio 2003
 - 2004 Mono 1.0
 - 2005 .NET 2.0, C# 2.0, Visual Studio 2005
Generics, Nullable Value Types
 - 2006 .NET 3.0, Mono 1.2
WPF, WCF, WF
 - 2007 .NET 3.5, C# 3.0, Visual Studio 2008
LINQ, Anonymous Types, Lambda Expressions, Extension Methods, Implicit Typing
 - 2008 Entity Framework 1.0
 - 2009 ASP.NET MVC 1.0
 - 2010 .NET 4.0, C# 4.0, ASP.NET MVC 2, Visual Studio 2010
Named / Optional Arguments, Dynamic Binding
 - 2012 .NET 4.5, C# 5.0, Mono 3.0, ASP.NET MVC 4, Visual Studio 2012
Asynchronous Members (async / await)
 - 2013 .NET 4.5.1, ASP.NET MVC 5, Visual Studio 2013
SignalR 1.0

- 
- 2015 .NET 4.6, C# 6.0, Mono 4.0, Visual Studio 2015, Visual Studio Code 1.0
Expression Bodied Members, Null Propagator, String Interpolation
 - 2016 Xamarin Acquisition, .NET Core 1.0, .NET Standard 1.0
Entity Framework Core 1.0
 - 2017 .NET 4.7, .NET Core 2.0, C# 7.0, Visual Studio 2017
ASP.NET Razor Pages, Out Variables, Tuples, Ref Locals and Returns
 - 2018 GitHub Acquisition, .NET Standard 2.0
Blazor Server
 - 2019 .NET 4.8, .NET Core 3.0, C# 8.0, Visual Studio 2019
gRPC, Default Interface Methods, Using Declarations, Nullable Reference Types
 - 2020 .NET 5, C# 9.0
Blazor WebAssembly, Records, Init Only Setters, Top-Level Statements
 - 2021 .NET 6, C# 10.0, Visual Studio 2022
.NET MAUI
 - 2022 .NET 7, C# 11.0
Standard-Term Support (STS) release primarily focused on performance

2023



.NET 8, C# 12

*Long-Term Support (LTS) release [will be supported until November 2026]
Primary constructors, collection expressions, runtime improvements,
preview of .NET Aspire, render modes in Blazor, and more*

Introduction

Evolution of the .NET Platform

- .NET 1.0 included ASP.NET Web Forms
 - Had the potential to be cross-platform but was only officially supported on Windows
- Current version of this variant is 4.8 and now referred to as ".NET Framework"
- Will be supported for many years to come (end of support for .NET 3.5 SP1 is October 2028)

Introduction

Evolution of the .NET Platform

- The ASP.NET MVC web application framework was introduced in 2009
- Initially presented as an alternative to Web Forms (not a replacement)
- Accompanied by a related framework for building services called Web API

Introduction

Evolution of the .NET Platform

- In 2016, Microsoft introduced a new variant of .NET called .NET Core
- Many components were completely rewritten
- Fully supported on Windows, macOS, and Linux
- Included a subset of the functionality provided by .NET Framework
 - Focused on web-based workloads (web UIs and services)
- Merged MVC and Web API into the core framework

Introduction

Evolution of the .NET Platform

- The version of .NET Core after 3.1 became the "main line" for .NET and was labeled .NET 5.0
- Supports development of Windows Forms and WPF applications that run on Windows
- The ASP.NET framework in .NET still includes the name "Core" to avoid confusion with previous versions of ASP.NET MVC

Introduction

Evolution of the .NET Platform

- The entire .NET platform is made available as open-source
- Community contributions are encouraged via pull requests
 - Thoroughly reviewed and tightly controlled by Microsoft

github.com/dotnet

Introduction

.NET SDKs and Runtimes

- .NET Runtime
 - Different version for each platform
 - Provides assembly loading, garbage collection, JIT compilation of IL code, and other runtime services
 - Includes the dotnet tool for launching applications
- ASP.NET Core Runtime
 - Includes additional packages for running ASP.NET Core applications
 - Reduces the number of packages that you need to deploy with your application

Introduction

.NET SDKs and Runtimes

- .NET SDK
 - Includes the .NET runtime for the platform
 - Additional command-line tools for compiling, testing, and publishing applications
 - Contains everything needed to develop .NET applications (with the help of a text editor)

Introduction

.NET SDKs and Runtimes

- Each version of .NET has a lifecycle status
 - Standard Term Support (STS) – Includes the latest features with a support period of 18 months
 - Long Term Support (LTS) – Has an extended support period of three years
 - Preview – Not supported for production use
 - Out of support – No longer supported

dotnet.microsoft.com/download

Introduction

IDE Choices

- Visual Studio is available for Windows and macOS
 - Full-featured IDE
- Visual Studio Code is available for Windows, macOS, and Linux
 - Includes IntelliSense and debugging features
 - Thousands of extensions are available for additional functionality

visualstudio.microsoft.com

Introduction

IDE Choices

- JetBrains also offers an IDE for .NET development called Rider
- Available for Windows, macOS, and Linux
- Includes advanced capabilities in the areas of refactoring, unit testing, and low-level debugging

www.jetbrains.com/rider

Modern Full-Stack with .NET 8 and JavaScript

Overview of Project Types

- ASP.NET MVC and Razor Pages
- Minimal APIs
- Blazor
- Class Libraries
- .NET MAUI
- Disadvantages of Project Templates

Overview of Project Types

ASP.NET MVC and Razor Pages

- ASP.NET Core supports two different architectural styles
 - Model-View-Controller (MVC)
 - Razor Pages
- Everything can be done with either approach
- The two styles can be mixed in the same project

Overview of Project Types

ASP.NET MVC and Razor Pages

- With MVC, incoming requests are routed to a controller
- The controller interacts with the model data and determines what response to return
- When using a View, the controller passes the model data to the view
 - Using ViewData or the Model<T> property

Overview of Project Types

ASP.NET MVC and Razor Pages

- Razor Pages is a more page-based architectural style
- By using the @page directive, a view can act like an MVC action and handle requests directly without a separate controller
- View can specify a PageModel class that is used to provide data to the view
 - Takes over the responsibilities handled by the controller and view model in an MVC style app

Overview of Project Types

Minimal APIs

- Simplified approach for building HTTP APIs
- Route and endpoint functionality defined in the pipeline configuration
- No need for a separate controller

```
var app = builder.Build();  
  
app.MapGet("/users/{userId}/books/{bookId}",  
    (int userId, int bookId) => $"User: {userId}, Book: {bookId}");  
  
app.Run();
```

Overview of Project Types

Blazor

- Component-based framework for implementing back-end and front-end web application functionality in C#
- Components can execute on the server and provide client-side interactivity via a persistent WebSocket connection
- Components can execute on the client by using a version of the .NET CLR implemented in WebAssembly
- Native client apps can use a hybrid approach to embed functionality implemented as Blazor components
 - .NET MAUI, WPF, Windows Forms, etc.

Overview of Project Types

.NET MAUI

- .NET Multi-platform App UI (MAUI) is a cross-platform framework for creating native mobile and desktop apps with C# and XAML
- Apps that can run on Android, iOS, macOS, and Windows from a single code-base
- Evolution of Xamarin.Forms

Overview of Project Types

.NET MAUI

- Android apps use the normal CLR and JIT compiler
- iOS apps are fully ahead-of-time (AOT) compiled from C# into native ARM assembly code
- macOS apps use Mac Catalyst
- Windows apps use the WinUI 3 library (native UI platform component that ships with the Windows App SDK)

Overview of Project Types

Class Libraries

- With so many different application frameworks and hosting options for C# code, it is important for framework-neutral code to be independent and reusable
- .NET class libraries can be used to componentize functionality into modules that can be used by multiple applications
- If compatibility with .NET Framework is required, define functionality in a .NET Standard 2.0 library

Overview of Project Types

Disadvantages of Project Templates

- The project templates included with the .NET SDK aim to provide developers with a functional starting point
- Can be a valuable learning resource
- Templates often include third-party libraries
 - Cause templates to become "opinionated"
 - Library versions were current at the time the template was created

Modern Full-Stack with .NET 8 and JavaScript

Modern C#

- Evolution of the C# Language
- Value Types and Reference Types
- Nullability
- Immutability
- Record Types
- Exceptions vs. Result
- Designing for Concurrency
- Deferred Execution

Modern C#

Evolution of the C# Language

- C# 1.0 was released in January 2002 alongside .NET 1.0
- Language has undergone many changes over the years
- Some features still exist for backward compatibility but should no longer be used (e.g., ArrayList)
- Many features found in functional programming languages have been added to C# over the years
 - Higher-order functions, closures, tuples, pattern matching, and more
- One thing that has not changed in C# is that every type is either a value type or a reference type

Modern C#

Value Types and Reference Types

- Value types
 - Equality involves the actual data values
 - Have a default value (cannot be null)
 - Cannot participate in inheritance
- Reference types
 - Equality involves referential identity
 - Defaults to null
 - Supports object-oriented concepts (inheritance, polymorphism, etc.)

Modern C#

Value Types and Reference Types

- All types in C# are copied upon assignment but it is important to recognize what is being copied
 - Did you make a copy of a value or a copy of a reference?
 - Critically important when evaluating thread safety (shared state)
- The String type is defined as a reference type but is immutable and can therefore be safely treated as if it was a value type

Modern C#

Nullability

- Accessing a null value is the number one cause of .NET application crashes (NullReferenceException)
- One way to prevent this is to never allow anything to be null
 - Some programming languages have taken this approach
- Value types cannot be null but a nullable value type can be used
 - Uses the Nullable<T> structure that can contain a value of the underlying type or the value of null

```
Nullable<int> maybeNumber;
```

```
int? maybeNumber;
```

Modern C#

Nullability

- `Nullable<T>` requires the developer to explicitly handle the possibility that the value may be null

```
if (maybeNumber is null) {  
    DoSomethingDifferent();  
}
```

- Leads to improved safety and turns what would be a run-time error into a compile-time error

Modern C#

Nullable Reference Types

- C# 8 introduced the concept of nullable reference types (enabled by default starting with .NET 6)
- When enabled, the compiler will treat all reference types as not allowing null
 - Enables null-state analysis for generating compiler warnings
- For a variable that should allow a null value, the same syntax used for nullable value types is used

```
Person? maybePerson;
```

- Is only used for compile-time analysis (not present in generated code)

Modern C#

Nullable Reference Types

- Null-conditional operator

```
string? str = person.Address?.Street;
```

- Null-coalescing operator

```
Person p = person ?? new Person();
```

- Null-forgiving operator

```
string str = person.Address!.Street;
```

Modern C#

Immutability

- Something else that can help with safety and to avoid unexpected side-effects is immutability
- It is safe to access state from multiple threads if the state is immutable (read-only)
- By convention, structs in C# are immutable (methods that appear to modify a struct instead return a new struct)

Modern C#

Record Types

- New kind of user-defined type in C#
- Value semantics for equality
- Concise syntax for creating a reference type with immutable properties
- Good fit for things like view models and DTOs
- Built-in support for cloning a record via the with keyword

```
var person = new Person("Joe", "Smith");  
Person brother = person with { FirstName = "Bill" };
```

Modern C#

Exceptions vs. Result

- C# does not have checked exceptions
- Sometimes difficult to know when a method might throw an exception
- The LanguageExt.Core NuGet package adds `Option<T>` and `Result<T>`
 - Will become an official feature when C# adds type unions

Modern C#

Designing for Concurrency

- Concurrency is a powerful technique to...
 - Improve server scalability
 - Improve user experience (responsiveness and feedback)
 - Implement parallel "first one wins" patterns
 - Perform CPU-bound operations faster

Modern C#

Designing for Concurrency

- When performing IO-bound operations (database access, web service calls, etc.), it is a best practice to perform that work asynchronously
- Allows for the efficient use of thread resources
 - Thread pool threads can be used to handle other incoming requests while the IO-bound operation is in progress
 - Improves the scalability of a web application

Modern C#

Designing for Concurrency

- Asynchronous requests from the client to the server can be used to improve the user experience
- Ajax in JavaScript or HttpClient in C# with Blazor

Modern C#

Deferred Execution

- Language Integrated Query (LINQ) has become a significant part of modern C# development
- LINQ queries that produce a collection leverage deferred execution
 - Evaluation of the query is delayed until its realized value is actually required
 - Implemented internally with C#'s yield-return statement

Modern Full-Stack with .NET 8 and JavaScript

Multi-Project Solutions

- Source Control Repositories
- Project References
- Unit Testing

Multi-Project Solutions

Source Control Repositories

- Using a source control repository is essential when multiple developers are working on a project
 - Pull requests, code reviews, version tagging, etc.
- Can also be very useful when working locally
 - Branching, discarding changes, etc.

Multi-Project Solutions

Source Control Repositories

- Git is currently the most popular source control solution
- Support included in Visual Studio, VS Code, and Rider
- There are several different workflows that can be used with Git

Multi-Project Solutions

Source Control Repositories

- Centralized Workflow
 - One "master" branch that everyone pulls from and pushes to
- Feature Branch Workflow
 - Separate branch for each feature
 - Feature branch merged into master when complete
- Forking Workflow
 - Developers fork the main repository and then submit pull requests when work is complete
 - Popular with open-source projects

Multi-Project Solutions

Source Control Repositories

- Pull Request Workflow
 - Developers create branches for their work and submit pull requests when changes are ready for review
 - Team members review the changes and provide feedback
 - Can be used in conjunction with other workflows

Multi-Project Solutions

Source Control Repositories

- Gitflow Workflow
 - Separate branches for "master", "develop", "feature", "release", and "hotfix"
 - Feature branches are created from the development branch and merged back in
 - Release branch created from development

Multi-Project Solutions

Source Control Repositories

- For a multi-project .NET solution, a single "monorepo" can be used
 - Solution file(s) and all project files are in one repository
- Alternatively, a separate repository can be used for each project
 - Developers can manage their own solution file(s) based on the project(s) they are working on

Multi-Project Solutions

Project References

- Projects within a .NET solution will often reference each other (via a relative path)
- When using separate repositories, developers will need to be aware of the dependencies and the necessary folder structure

Multi-Project Solutions

Unit Testing

- Unit tests should always be kept in a separate project (code is not shipped with the product)
- Options include...
 - One unit test project that covers an entire monorepo
 - Multiple unit test projects for different functional areas
 - Every .NET project has a companion unit test project

Modern Full-Stack with .NET 8 and JavaScript

Modern JavaScript

- Introduction
- Host Environments
- Strict Mode
- Statements
- Types
- Numbers
- Strings
- Object Type
- Functions
- Classes

Modern Full-Stack with .NET 8 and JavaScript

Modern JavaScript

- Asynchronous JavaScript
- HTML Script Tag
- Web Security Model
- Events

Modern JavaScript

Introduction

- When you would like to execute code on the client-side (browser), JavaScript is necessary
- WebAssembly and Blazor are gaining in popularity, but some JavaScript is often still required
 - Blazor is also not appropriate in all scenarios

Modern JavaScript

Introduction

- High-level, dynamic, interpreted programming language
- Variables are untyped
- "JavaScript" is a trademark now owned by Oracle
- The standardized version of the language has the name "ECMAScript" (often abbreviated ES)
- ES6 (2015) was a major release and added many new features

Modern JavaScript

Host Environments

- The original and most popular host environment for JavaScript is the web browser
- Introduced in 2010, Node provides another stand-alone host environment

Modern JavaScript

Strict Mode

- ES5 and later let scripts opt in to strict mode
- Alters the semantics of JavaScript to improve resiliency and the visibility of errors

```
// Whole-script strict mode syntax
"use strict";
const v = "Hi! I'm a strict mode script!";
```

- JavaScript modules are automatically in strict mode

Modern JavaScript

Statements

- Like many languages, JavaScript uses the semicolon to separate statements
- JavaScript will add an implicit semicolon if the next non-space character cannot be interpreted as a continuation of the current statement

```
let x
x
=
2
console.log(x)
```

```
let x; x = 2; console.log(x);
```

Modern JavaScript

Types

- Primitive types include number, string, boolean, null, and undefined
- Everything else is an object or an array
- Object types are mutable and primitive types are immutable
- Constants and variables allow you to use names to refer to values

```
const x = 2;  
let y = 5;
```

Modern JavaScript

Types

- JavaScript liberally converts values from one type to another
- The equality operator (==) will perform type conversions
- The strict equality operator (===) will not

Modern JavaScript

Numbers

- The Number type in JavaScript is 64-bits in size and is used to represent integers and approximate real numbers
- JavaScript does not raise errors in cases of overflow, underflow, or division by zero
 - Special values of Infinity, -Infinity, and NaN (not a number) are used

Modern JavaScript

Strings

- A string literal can be defined using a matched pair of single quotes, double quotes, or backticks

```
let s1 = "This";  
let s2 = 'is';  
let s3 = `a test`;
```

- Backticks are commonly used to allow for the easy embedding of the others (which are used in text and HTML)

Modern JavaScript

Object Type

- An object in JavaScript is a collection of name/value pairs

```
let book = {  
  title: "Introduction to JavaScript",  
  pub: 2024  
};
```

- The `JSON.stringify()` method converts a JavaScript value to a JSON string

```
{  
  title: "Introduction to JavaScript",  
  pub: 2024  
}
```

- Use `JSON.parse()` to construct an object from a JSON string

Modern JavaScript

Functions

- A function is a named and parameterized block of JavaScript

```
function addOne(x) {  
  return x + 1;  
}
```

- A function can be assigned to a variable

```
let square = function(x) {  
  return x * x;  
}  
  
let r = square(2);
```

Modern JavaScript

Functions

- ES6 added a shorthand syntax known as arrow functions

```
const plusOne = x => x + 1;
```

Modern JavaScript

Classes

- Classes are templates for creating objects
- In JavaScript, classes are really just "special functions" that create objects

```
class Rectangle {  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
  calcArea() {  
    return this.height * this.width;  
  }  
}
```

```
const square = new Rectangle(10, 10);  
console.log(square.calcArea);
```

Modern JavaScript

Classes

- Fields do not have to be declared explicitly unless you want the fields to be private

```
class Rectangle {  
  #height;  
  #width;  
  constructor(height, width) {  
    this.height = height;  
    this.width = width;  
  }  
}
```

Modern JavaScript

Classes

- The extends keyword is used to create a class as a child of another constructor

```
class Animal {  
  constructor(name) {  
    this.name = name;  
  }  
  speak() {  
    console.log(`${this.name} makes a noise.`);  
  }  
}  
  
class Dog extends Animal {  
  constructor(name) {  
    super(name);  
  }  
  speak() {  
    console.log(`${this.name} barks.`);  
  }  
}
```

Modern JavaScript

Modules

- When JavaScript files are loaded into an HTML page via a script tag, by default, they are loaded into global scope
- Global scope in a web page is defined by the window

Modern JavaScript

Modules

- Starting with ES6, each JavaScript file can represent a module that defines its own scope
- Constants, variables, functions, and classes defined within a file are private to that module unless explicitly exported

```
export const PI = Math.PI;

export function degreesToRadians(d) { return d * PI / 180; }

export class Circle {
  constructor(r) { this.r = r; }
  area() { return PI * this.r * this.r; }
}
```

Modern JavaScript

Modules

- You import values that have been exported by other modules with the import keyword

```
import { mean, stddev } from "./stats.js";
```

```
import * as stats from "./stats.js";
```

- For a JavaScript file to be treated as a module by the web browser, type="module" must be used

```
<script type="module">import "./main.js";</script>
```

Modern JavaScript

Modules

- A dynamic import can be used to asynchronously load additional JavaScript only when it is required

```
async analyzeData(data) {  
  let stats = await import("./stats.js");  
  return {  
    average: stats.mean(data),  
    stddev: stats.stddev(data)  
  };  
}
```

Modern JavaScript

Asynchronous JavaScript

- Asynchronous behavior is very common in modern browser-based applications
- Implemented in many places via callbacks

```
let poll = setInterval(checkForUpdates, 10000);
```

```
let okay = document.querySelector('#okayButton');  
okay.addEventListener('click', saveData);
```

Modern JavaScript

Asynchronous JavaScript

- JavaScript includes newer features to make asynchronous code easier to write
 - Promises: Introduced in ES6, are objects that represent a "not yet available" result of an asynchronous operation
 - async / await: Keywords added in ES2017 to provide a new alternative syntax for promise-based code

Modern JavaScript

Asynchronous JavaScript

- Promises help to improve asynchronous code
 - Simpler to read and understand (vs. many nested callbacks)
 - Easier error handling
- Many options become available for a function that returns a promise

```
fetch(url).then(displayProducts).catch(handleProductsError);
```

Modern JavaScript

Asynchronous JavaScript

- `async` and `await` dramatically simplify the use of promises
- Allows for asynchronous code that looks more like synchronous code

```
async function getProducts() {  
  let response = await fetch(url);  
  let products = await response.json();  
  return products;  
}
```

- An `async` function returns a promise
- `await` can be used with a function that returns a promise

Modern JavaScript

HTML Script Tag

- The `<script>` tag is the primary way to load JavaScript into an HTML page

```
<script src="cart.js"></script>
```

- Tag must be explicitly closed (`<script/>` not supported)
- The language and type attributes are unnecessary (unless to specify that the script is a module)

Modern JavaScript

HTML Script Tag

- `defer` attribute will cause a script to execute after the document has been fully loaded and parsed

```
<script defer src="cart.js"></script>
```

- Multiple scripts with `defer` will execute in the order they appear in the document
- `async` attribute will cause a script to execute as soon as possible but will not block document parsing while being downloaded

```
<script async src="cart.js"></script>
```


Modern JavaScript

Web Security Model

- The origin of a document is defined as the protocol, host, and port of the URL from which the document was loaded
- Scripts can only read the properties of windows and documents that have the same origin as the document that contains the script
- Scripts can also only make HTTP requests to the same origin from which the containing document was loaded (unless the target server has CORS configured properly)

Modern JavaScript

Web Security Model

- Cross-Origin Resource Sharing (CORS) allows servers to decide which origins they are willing to serve
- Browsers honor the CORS headers returned by a server

Modern JavaScript

Events

- There are multiple ways to register an event handler
 - Set an event handler attribute

```
<button onclick="console.log('hello');">Click me</button>
```

- Set a property of the event target

```
window.onload = function() { ... }
```

- Call `addEventListener()`

```
let b = document.querySelector("#theButton");  
b.addEventListener("click", () => { console.log('hello'); });
```

Modern Full-Stack with .NET 8 and JavaScript

JavaScript Frameworks

- TypeScript
- jQuery and jQuery UI
- Angular
- React
- Vue.js
- Svelte
- Alpine.js
- HTMLX

JavaScript Frameworks

TypeScript

- A strongly-typed language built on top of JavaScript
- Provides more develop-time static analysis
- Converts to pure JavaScript
- Many JavaScript frameworks provide specific support for TypeScript and some are also written in it (e.g., Angular)

JavaScript Frameworks

jQuery and jQuery UI

- jQuery is a JavaScript library designed to simplify DOM tree transversal / manipulation, event handling, animations, and Ajax
- Although still very popular, new features have been added to JavaScript itself that address many of the shortcomings that made jQuery so attractive (e.g., `fetch()`)
- jQuery UI is a collection of GUI widgets, animated visual effects, and themes implemented with jQuery
- jQuery (and Bootstrap) are included by default in the ASP.NET Core project templates

JavaScript Frameworks

Angular

- TypeScript-based single-page web application framework
- Uses a component-based architecture that allows developers to build encapsulated, reusable user interface elements
- Supports two-way data binding

JavaScript Frameworks

React

- JavaScript library for front-end development
- Component-based architecture and declarative views
- Uses one-way data binding and a virtual DOM

JavaScript Frameworks

Vue.js

- Model-View-ViewModel front end JavaScript framework for building user interfaces and single-page applications
- Extends basic HTML elements to encapsulate reusable code

JavaScript Frameworks

Svelte

- Component-based front-end framework and language
- Compiles HTML templates to specialized code that manipulates the DOM directly

JavaScript Frameworks

Alpine.js

- Lightweight JavaScript micro framework
 - 15 attributes, 6 properties, and 2 methods
- Described as "jQuery for the modern web"

```
<div x-data="{ open: false }">  
  <button x-on:click="open != open">Toggle</button>  
  <div x-show="open">  
    ...  
  </div>  
</div>
```

JavaScript Frameworks

HTMX

- Front-end JavaScript library that extends HTML with custom attributes
- Enables the use of Ajax directly in HTML with a hypermedia-driven approach

```
<button hx-post="/clicked" hx-swap="outerHTML">  
  Click Me  
</button>
```

Modern Full-Stack with .NET 8 and JavaScript

Hands-On Case-Study Project

- Requirements
- Architectural Choices
- Entities and Business Logic
- Data Persistence, EF, and LINQ
- Server Generated Content
- Client-Side Interactivity
- Building a RESTful API
- Authorization and Bearer Tokens
- Blazor
- Using JavaScript within a Blazor Application