C# .Net 12.0

OOPS with VS 2022

.Net 8

Outline

- 1. C# 12 -Improvements and new features
- 2. ASP.NET Core 8 Improvements and new features
- 3. Entity Framework Core 8 Improvements and new features
- 4. Dynamic Profile-Guided Optimization (PGO)
- 5. .NET Aspire
- 6. AI features - Extended AI and ML Support
- 7. Improved Diagnostics and Observability
- 8. Native AOT Support
- 9. Extended API Authoring
- 10. Blazor improvements

C# 12 -Improvements and new features

C# 12

Installing Visual Studio 2022

- We get 30 days trial version from the Microsoft site:
- https://my.visualstudio.com/Downloads?PId=6542
- Login using your Microsoft id.
- Download the installer from the above site.
- We will use Community Edition, which is the full edition of VS2022.
- Step by step follow the installer, selecting the required modules.

New Features of C# 12

- The release of C# 12, bundled with .NET 8, has introduced several exciting features that enhance the language’s expressiveness and performance.
- **Collection Expressions**
- **Primary Constructors**
- **Inline Arrays**
- **Default Lambda Parameters**
- **ref readonly Parameters**
- **Alias Any Type with using**
- **Experimental Attribute**

Collection Expressions

- This new syntax allows for the creation of collections like arrays, lists, and spans (Span is a struct in C#. A new feature introduced in C# 7.2 that allows for the representation of a contiguous region of arbitrary memory as a first-class object. This can be useful when working with large amounts of data, as it allows for more efficient memory usage and can improve performance.) in a more concise way.
- We can use the spread operator .. to inline elements from one collection into another, streamlining the process of combining multiple collections.
- It allows us to directly specify the elements of collections in a more concise and expressive manner.

Traditional approach to defining an array, a span, and a list

```
using System;
using System.Collections.Generic;
class Program {
    static void Main(string[] args)   {
        int[] a = {1, 2, 3, 4, 5, 6, 7, 8}; // Create an array
        List<string> b = new List<string>{"one", "two", "three"}; // Create a list
        Span<int> c = new Span<int>(a);  // Create a span
        int[][] twoD = {new int[]{1, 2, 3}, new int[]{4, 5, 6}, new int[]{7, 8, 9}}; // Create a jagged 2D array:
        // Create a jagged 2D array from variables:
        //int[] row0 = [1, 2, 3];   int[] row1 = [4, 5, 6];   int[] row2 = [7, 8, 9];
        int[][] twoDFromVariables = twoD;
        // Printing values
        Console.WriteLine("Array a:");
        foreach (var item in a) Console.WriteLine(item);
        Console.WriteLine("\nList b:");
        foreach (var item in b) Console.WriteLine(item);
        Console.WriteLine("\nSpan c:");
        foreach (var item in c)   Console.WriteLine(item);
        Console.WriteLine("\n2D array twoD:");
        foreach (var row in twoD) {
            foreach (var item in row) Console.Write(item + " ");
            Console.WriteLine();
        }
        Console.WriteLine("\n2D array twoDFromVariables:");
        foreach (var row in twoDFromVariables)   {
            foreach (var item in row) Console.Write(item + " ");
            Console.WriteLine();
        }
    }
}
```

```
Array a:
1
2
3
4
5
6
7
8

List b:
one
two
three

Span c:
1
2
3
4
5
6
7
8

2D array twoD:
1 2 3
4 5 6
7 8 9

2D array
twoDFromVariables:
1 2 3
4 5 6
7 8 9
```

New c# 12 collection to defining an array, a span, and a list

```
using System;
using System.Collections.Generic;
class Program {
    static void Main(string[] args)   {
        int[] a = [1, 2, 3, 4, 5, 6, 7, 8]; // Create an array
        List<string> b = ["one", "two", "three"]; // Create a list
        Span<char> c = ['a', 'b', 'c', 'd', 'e', 'f', 'h', 'i'];  // Create a span
        int[][] twoD = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]; // Create a jagged 2D array:
        // Create a jagged 2D array from variables:
        int[] row0 = [1, 2, 3];   int[] row1 = [4, 5, 6];   int[] row2 = [7, 8, 9];
        int[][] twoDFromVariables = [row0, row1, row2];
        // Printing values
        Console.WriteLine("Array a:");
        foreach (var item in a) Console.WriteLine(item);
        Console.WriteLine("\nList b:");
        foreach (var item in b) Console.WriteLine(item);
        Console.WriteLine("\nSpan c:");
        foreach (var item in c)   Console.WriteLine(item);
        Console.WriteLine("\n2D array twoD:");
        foreach (var row in twoD) {
            foreach (var item in row) Console.Write(item + " ");
            Console.WriteLine();
        }
        Console.WriteLine("\n2D array twoDFromVariables:");
        foreach (var row in twoDFromVariables)   {
            foreach (var item in row) Console.Write(item + " ");
            Console.WriteLine();
        }
    }
}
```

```
Array a:
1
2
3
4
5
6
7
8

List b:
one
two
three

Span c:
a
b
c
d
e
f
h
i

2D array twoD:
1 2 3
4 5 6
7 8 9

2D array
twoDFromVariables:
1 2 3
4 5 6
7 8 9
```

New c# 12 collection using spread operator

```
using System;
using System.Collections.Generic;

class Program
{
    static void Main(string[] args)
    {
        // Create a array:
        int[] row0 = [1, 2, 3];
        int[] row1 = [4, 5, 6];
        int[] row2 = [7, 8, 9];
        int[] single = [.. row0, .. row1, .. row2];

        // Printing values
        foreach (var element in single)
        {
            Console.Write($"{element}, ");
        }
    }
}
```

```
1, 2, 3, 4, 5, 6, 7, 8, 9,
```

Primary Constructors

- A significant addition to C# 12, primary constructors enable more concise and clear syntax by allowing constructors to be declared inline with the type declaration. This feature is applicable to various types including class, struct, record class, and record struct, and is particularly useful for initializing member fields or properties and facilitating dependency injection.
- Primary constructors streamline the declaration of constructors inline with the type, making the code more concise and readable.
- Primary Constructors is a feature that was already present in C# 9 but it was restricted to record types, and now with C# 12 this feature was extended to any class and struct.
- Primary Constructor is an easier way to create a constructor for your class or struct, by eliminating the need for explicit declarations of private fields and constructor bodies that only assign parameter values to those fields. With Primary Constructor you can add parameters to the class declaration and use these values in the class body.
- Primary Constructors work well with classes that have many properties, as they reduce the amount of boilerplate code required to initialize them. Additionally, using Primary Constructors makes your code more readable and reduces the risk of errors caused by missing or incorrectly ordered initialization statements.

Traditional Code

```
using System;
public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public DateTime HireDate { get; set; }
    public decimal Salary { get; set; }
    public Employee(string firstName, string lastName, DateTime hireDate, decimal salary)  {
        FirstName = firstName;
        LastName = lastName;
        HireDate = hireDate;
        Salary = salary;
    }
    public override string? ToString() {
        return "\nFirstName: " + FirstName + " LastName: " + LastName + " HireDate: " + HireDate + " Salary: " + Salary;
    }
}
class Program {
    static void Main(string[] args)     {
        var employee = new Employee("John", "Doe", new DateTime(2024, 1, 1), 50000);
        Console.WriteLine(employee);
    }
}
```

```
FirstName: John LastName: Doe HireDate: 01-01-2024
00:00:00 Salary: 50000
```

C#12 Code : Primary Constructors

```
using System;
public class Employee(string firstName, string lastName, DateTime hireDate, decimal
salary)
{
    public string FirstName { get; init; } = firstName;
    public string LastName { get; init; } = lastName;
    public DateTime HireDate { get; init; } = hireDate;
    public decimal Salary { get; init; } = salary;
public override string? ToString() {
        return "\nFirstName: " + FirstName + " LastName: " + LastName + " HireDate: " +
HireDate + " Salary: " + Salary;
    }
}
class Program {
    static void Main(string[] args)    {
        var employee = new Employee("John", "Doe", new DateTime(2024, 1, 1), 50000);
        Console.WriteLine(employee);
    }
}
```

```
FirstName: John LastName: Doe HireDate: 01-
01-2024 00:00:00 Salary: 50000
```

Primary Constructors for Structs

- Similar to primary constructors for classes, structs can also have primary constructors, making them more concise and easier to use.

Example

```
using System;
namespace ConApp1222 {
    public struct Point(int x, int y)    {
        public int X { get; } = x;
        public int Y { get; } = y;
        //Deconstruction: The Deconstruct method allows the struct to be deconstructed into its components.
        public void Deconstruct(out int x, out int y)
        {   x = X;   y = Y;    }
        public override string ToString() => $"Point({X}, {Y})";
    }
    class Program   {
        static void Main(string[] args)       {
            // Create a Point instance using the primary constructor
            Point point = new Point(3, 4);
            // Access properties
            Console.WriteLine($"X: {point.X}, Y: {point.Y}");
            // Deconstruct the point
            var (x, y) = point;
            Console.WriteLine($"Deconstructed: X = {x}, Y = {y}");
            // Use the ToString method
            Console.WriteLine(point);
        }
    }
}
```

```
X: 3, Y: 4
Deconstructed: X = 3, Y = 4
Point(3, 4)
```

Deconstruction, introduced in C# 7.0

- Deconstruction was introduced in C# 7.0.
- This feature allows you to easily break down tuples and objects into individual variables, providing a more readable and concise way to handle multiple values returned from methods or properties.
- Key Features Introduced with Deconstruction in C# 7.0
  - Deconstruction of Tuples: This allows tuples to be broken down into individual variables.
  - Deconstruct Method for Custom Types: You can define a Deconstruct method in your own classes and structs, enabling custom types to support deconstruction.
- **Advantages of Deconstruction**
  - **Improved Readability**: Deconstruction can make code more readable by reducing the need for verbose property access.
  - **Simplified Code**: It simplifies the assignment of multiple variables from a single object or tuple.
  - **Consistent Syntax**: Provides a consistent and familiar syntax for working with tuples and custom types.

Example

```
using System;
namespace ConApp1222 {
    class Program   {
        static void Main(string[] args)        {
            var point = (x: 3, y: 4);
            (int x, int y) = point;
            Console.WriteLine($"X: {x}, Y: {y}"); //X: 3, Y: 4
        }
    }
}
```

Inline Arrays

- Inline arrays are struct-based, fixed-length arrays that provide a performance boost, particularly in scenarios involving arrays of structures. This feature enables you to work with memory buffers more efficiently, without needing unsafe code.
- Inline arrays are fixed-size, struct-based arrays that improve performance by reducing allocations and copying.
- One advantage of this is that structs are value types, and memory for struct instances are usually allocated on the stack (unless subjected to ‘boxing,’ which is the process of converting a value type to a reference type, or if they are fields within a class or elements of an array), which results in more efficient memory usage, while arrays are reference types, and their memory is stored on the heap.

Example

```
using System;
namespace Console12App1  {
    struct XY { public int X; public int Y;
        public override string ToString() {  return "(" + X + ", " + Y + ")";  }
    }
    [System.Runtime.CompilerServices.InlineArray(4)] //Max-Size : 4
    struct Point  { // Inline array for geometric operations
        public XY xy { get; set; }
    }
    class Program    {
        static void Main(string[] args)     {
            XY xy1 = new XY { X = 2, Y = 3 }; XY xy2 = new XY { X = 5, Y = 7 };
            Point p = new Point(); // Demonstrate geometric operations with inline array
            p[0] = xy1; p[1] = xy2;
            double distance = Math.Sqrt(Math.Pow(p[0].X - p[1].X, 2) + Math.Pow(p[0].Y - 
p[1].Y, 2));
            Console.WriteLine($"Distance between points: {p[0]} and {p[1]} is {distance}");
        }
 }  }
```

```
Distance between points: (2, 3) and (5, 7) is 5
```

Lambda Expression Improvements

- C# 12 introduces several enhancements to lambda expressions that make them more powerful and expressive.
- **Key Lambda Expression Improvements in C# 12**
  - **Return Types for Lambdas**
  - **Attributes on Lambdas**
  - **Default Parameter Values for Lambdas**

Return Types for Lambdas

- C# 12 introduces the ability to explicitly specify return types for lambda expressions.
- This enhancement allows developers to provide more clarity about what a lambda returns, especially when the return type is complex or cannot be easily inferred by the compiler.
- Explicit return types can also be useful for documentation and code readability.
- **Benefits of Specifying Return Types**
  - **Clarity**: Explicit return types make it clear what type the lambda expression returns, improving code readability.
  - **Safety**: Helps prevent errors by ensuring the return type is what you expect.
  - **Inference Limitations**: In some cases, the compiler might not be able to infer the return type correctly, especially with more complex logic. Explicit return types solve this problem.
- By specifying return types for lambdas, you can make your code more robust and maintainable, especially when dealing with complex expressions or when the return type is not immediately obvious. This feature is a valuable addition to C# 12, enhancing the language's expressiveness and usability.

Example

```
using System;
namespace ConApp1222 {
    class Program   {
        static void Main(string[] args)        {
            // Lambda with explicit return type for addition
            Func<int, int, int> add = (int x, int y) => (x + y);
            int additionResult = add(3, 4);
            Console.WriteLine($"Addition Result: {additionResult}");
            // Lambda with explicit return type for multiplication
            Func<int, int, int> multiply = (int x, int y) => (x * y);
            int multiplicationResult = multiply(3, 4);
            Console.WriteLine($"Multiplication Result: {multiplicationResult}");
            // Lambda with explicit return type for determining if a number is even
            Func<int, bool> isEven = (int number) => (number % 2 == 0);
            bool isEvenResult = isEven(4);
            Console.WriteLine($"Is 4 even? {isEvenResult}");
            // Lambda with explicit return type for a more complex calculation
            Func<double, double, double> complexCalculation = (double a, double b) =>  {
                double result = Math.Pow(a, b) + Math.Sqrt(a * b);
                return result;
            };
            double complexResult = complexCalculation(3.0, 4.0);
            Console.WriteLine($"Complex Calculation Result: {complexResult}");
        }
    }
}
```

Attributes on Lambdas

- In C# 12, you can now apply attributes directly to lambda expressions.
- This feature allows you to add metadata to lambdas, which can be useful for various purposes such as diagnostics, controlling compiler behavior, or providing additional information about the lambda.
- **Benefits of Applying Attributes to Lambdas**
  - **Enhanced Metadata**: Adding attributes to lambdas allows you to provide additional information and control behavior at a granular level.
  - **Conditional Compilation**: Using attributes like Conditional, you can control whether certain lambdas are included in the compiled output based on compilation symbols.
  - **Custom Behaviors**: Custom attributes can enable specific behaviors, logging, or other diagnostics tailored to your needs.
- By applying attributes to lambdas, you can make your code more flexible and powerful, leveraging the additional metadata and behavior control these attributes provide.
- This feature in C# 12 opens up new possibilities for managing and utilizing lambda expressions in your applications.

Example

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program  {
        static void Main(string[] args)       {
#if DEBUG
            // Lambda with a Conditional attribute
            Action logDebug =  () =>  Console.WriteLine("This is a debug message.");
            logDebug();
#endif
            // Lambda with a custom attribute
            Action logCustom = [MyCustom] () =>  Console.WriteLine("This is a custom attributed lambda.");
            logCustom();
        }
        // Custom attribute definition
        [AttributeUsage(AttributeTargets.Method | AttributeTargets.Delegate)]
        public class MyCustomAttribute : Attribute  {   }
    }
}
// This is a debug message.
// This is a custom attributed lambda.
```

Default Lambda Parameters

- C# 12 introduces the ability to define default values for parameters in lambda expressions. This feature brings added flexibility and expressiveness to lambda expressions, making them more concise and adaptable.
- C# 12 allows specifying default values for lambda expression parameters, offering more flexibility.
- **Benefits of Default Parameter Values in Lambdas**
  - **Simplified Code**: Default parameter values reduce the need for method overloads or additional logic to handle default values.
  - **Improved Readability**: Code becomes more readable and maintainable by clearly indicating default values within the lambda expression.
  - **Enhanced Flexibility**: Allows for more flexible lambda expressions that can be used in various contexts without requiring explicit arguments.
- By using default parameter values in lambda expressions, you can make your code more concise and easier to work with, leveraging the new capabilities introduced in C# 12.
- In C# 12, you can specify default parameter values for lambda expressions, but at least one parameter must be provided without a default value. This ensures that the lambda expression is correctly interpreted by the compiler.

Example

```
using System;
namespace Console12App1
{
    class Program
    {
        static void Main(string[] args)
        {
            var addNumbers = (int x = 12, int y = 1) => x + y;
            //Func<int, int, int> addNumbers = (int x=12, int y=1) => (x + y);

            Console.WriteLine(addNumbers()); // Output : 13
            Console.WriteLine(addNumbers(5)); // Output: 6
            Console.WriteLine(addNumbers(5, 5)); // Output: 10
        }
    }
}
```

Collection Literals

- Collection literals provide a shorthand way to initialize collections, making the code more readable and reducing the amount of boilerplate code.
- In C# 12, collection literals are introduced as a language feature, allowing you to initialize certain types of collections using a concise syntax directly in your code. This feature enhances readability and reduces verbosity when initializing collections.
- Types of Collection Literals
  - Lists (List<T>):
    - Example: List<int> numbers = [1, 2, 3, 4, 5];
  - Immutable Lists (ImmutableList<T> from System.Collections.Immutable):
    - Example: ImmutableList<int> numbers = [1, 2, 3, 4, 5].ToImmutableList();
  - Arrays:
    - Example: int[] numbers = [1, 2, 3, 4, 5];
  - Sets (HashSet<T>):
    - Example: HashSet<int> numbers = {1, 2, 3, 4, 5};
  - Immutable Sets (ImmutableHashSet<T> from System.Collections.Immutable):
    - Example: ImmutableHashSet<int> numbers = {1, 2, 3, 4, 5}.ToImmutableHashSet();
  - Dictionaries (Dictionary<TKey, TValue>):
    - Example: Dictionary<string, int> ages = {"Alice" => 30, "Bob" => 25, "Charlie" => 35};
  - Immutable Dictionaries (ImmutableDictionary<TKey, TValue> from System.Collections.Immutable):
    - Example: ImmutableDictionary<string, int> ages = {"Alice" => 30, "Bob" => 25, "Charlie" => 35}.ToImmutableDictionary();

Collection Literals

- **Benefits of Collection Literals**
  - **Conciseness**: Collection literals provide a more compact and readable syntax for initializing collections.
  - **Clarity**: The intent of initializing a collection is clearer and less error-prone.
  - **Reduced Boilerplate**: Avoids the need for explicit constructors or initialization methods.
- **Compatibility and Requirements**
  - Collection literals are available starting from C# 12 and require .NET 6 or newer. Ensure that your development environment and project configuration support these versions for optimal usage.
  - This feature significantly improves the developer experience when dealing with collections in C#, making the language more expressive and reducing the verbosity of initialization code.

Enhanced Pattern Matching

- In C# 12, pattern matching has been enhanced with new features and improvements that make it more powerful and easier to use. Pattern matching allows you to write clearer and more concise code when checking for the shape and properties of data. Let's explore the enhanced pattern matching features introduced in C# 12.
- Enhanced Pattern Matching Features in C# 12
- 1. Parenthesized Patterns :
  - You can now use parentheses to group patterns and apply logical operators (&&, ||, !) to combine patterns more effectively.
- 2. Not Patterns
  - The not keyword allows you to negate a pattern.
- 3. Relational Patterns
  - You can now use relational patterns (<, <=, >, >=, ==, !=) to perform comparisons within patterns.
- 4. Improved Type Patterns
  - You can use type patterns with constants and switch on multiple types.
- Benefits of Enhanced Pattern Matching
  - Conciseness: Patterns are more concise and readable.
  - Flexibility: More powerful with support for logical operators and relational patterns.
  - Deconstruction: Easily destructure objects and match their properties.
  - Type Safety: Improved type checking and safety.
- These enhancements in pattern matching make C# code more expressive and reduce boilerplate, improving both readability and maintainability of your code. They are available starting from C# 12 and require .NET 6 or newer. Ensure that your development environment and project configuration support these versions for optimal usage.

Example: Parenthesized Patterns

```
using System;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static string GetColorName(Color color)     {
            return color switch      {
                (0, 0, 0) => "Black",
                (255, 0, 0) => "Red",
                (0, 255, 0) => "Green",
                (0, 0, 255) => "Blue",
                _ => "Unknown"
            };
        }
        public record Color(int Red, int Green, int Blue);
        public static void Main()    {
            Color redColor = new Color(255, 0, 0);
            Console.WriteLine(GetColorName(redColor)); // Output: Red
        }
    }
}
```

Example: Not Patterns

```
using System;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static string GetOppositeColorName(Color color)        {
            return color switch            {
                //not (0, 0, 255) => "Not Blue",
                not (255, 0, 0) => "Not Red",
                not (0, 255, 0) => "Not Green"
                //_ => "Unknown"
            };
        }
        public record Color(int Red, int Green, int Blue);
        public static void Main()        {
            Color greenColor = new Color(0, 255, 0);
            Console.WriteLine(GetOppositeColorName(greenColor)); // Output: Not Green
        }
    }
}
```

Example: Relational Patterns

```
using System;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static string GetAgeGroup(int age)     {
            return age switch      {
                < 0 => "Invalid age",
                >= 0 and <= 12 => "Child",
                > 12 and <= 19 => "Teenager",
                > 19 and <= 65 => "Adult",
                _ => "Senior"
            };
        }
        public static void Main()    {
            int age = 25;
            Console.WriteLine(GetAgeGroup(age)); // Output: Adult
        }
    }
}
```

Example: Improved Type Patterns

```
using System;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static string DescribeObject(object obj)     {
            return obj switch      {
                0 => "Zero",
                string s => $"String of length {s.Length}",
                int i => $"Integer: {i}",
                bool b => $"Boolean: {b}",
                _ => "Unknown"
            };
        }
        public static void Main()       {
            object[] objects = { 0, "hello", 42, true, new List<int>() };
            foreach (var obj in objects)       {
                Console.WriteLine($"{obj} is {DescribeObject(obj)}");
            }
        }
    }
}
```

ref readonly Parameters

- Building upon the ref parameters feature, C# 12 introduces ref readonly parameters, which allow passing parameters by reference in a read-only context. This is particularly useful for methods that need the memory location of an argument without modifying it.
- ref readonly parameters enable passing arguments by reference in a read-only context, enhancing performance and safety.
- With that said, you might be wondering, why this ref readonly feature was added, if it was already possible to do something similar using the in keyword?
  - The reason for this, as it is in Microsoft's Docs it's for "APIs which capture or return references from their parameters would like to disallow rvalues and also enforce some indication at the callsite that a reference is being captured. ref readonly parameters are ideal in such cases as they warn if used with rvalues or without any annotation at the callsite.".

Why Ref Readonly Parameters?

- Why Ref Readonly Parameters?
  - Safety First: Restrict accidental modifications of passed references, leading to more predictable and reliable code.
  - Performance Boost: Avoid unnecessary copying of values compared to traditional ref parameters.
  - API Design Elegance: Clearly indicate your intent to only read passed references, enhancing code readability and maintainability.
- The Magic Inside
  - Declare with 'ref readonly': Just like ref, but with the readonly keyword adding the safety guard.
  - Read Freely: Access the parameter's value as you would within the method/function.
  - Modify at Your Own Risk: Attempting to modify the value inside the method will result in a compiler error.
- When to Embrace Them?
  - Passing large structures without unnecessary copying (e.g., reading sensor data).
  - Exposing readonly fields through methods while preventing unwanted changes (e.g., calculating statistics from a dataset).
  - Implementing interfaces/protocols requiring read-only access (e.g., iterating through a collection).
- Comparison with Similar Options
  - Ref vs. Ref Readonly: ref allows modification, while ref readonly enforces read-only access. Choose ref readonly for clarity and safety when writing methods/functions that don't need to modify passed references.
  - In vs. Ref Readonly: Both restrict modification, but in prevents copying and creates a temporary copy within the method. Ref readonly avoids copying and allows accessing the actual parameter. Use it for passing primitives efficiently or when the actual reference shouldn't be exposed within the method.

ref vs in vs ref readonly

- Imagine you require a reference to a value instead of a value as an argument to a method, but the struct you want to pass is a readonly struct. You have three options as of right now to pass by reference:
  - The ref parameter: This is not an option for you, as you are passing a readonly struct which cannot be modified, but because you could technically modify a ref parameter, the compiler must generate a copy of the readonly struct to guarantee, that the readonly struct is not modified.
  - The in parameter: This one is inapplicable in your case, because while it avoids copying the entire struct, and prohibits modificiation, there is no guarantee, that your parameter will actually be passed by reference - you can omit the in keyword when passing the parameter, to pass it by value. This means, that if you need to have a reference for an unsafe method for example, this would not work. The reason for the in keyword, is not for a parameter to be passed by readonly reference, but to avoid allocation. This often means passing a parameter by reference, but is not required.
  - The readonly ref parameter solves the above posed issue: It is guaranteed by a compiler warning, that the value must be passed by reference and there will not be a protection copy, because you cannot modify the parameter.
- To conclude: Only the readonly ref parameter will work, if you need to have a reference to a readonly struct. The in parameter does not guarantee that the parameter is passed by reference, and the ref parameter allows for modification, what means that the compiler has to copy the object to ensure, that it is not modified.

Callsite rules

Callsite annotation	ref parameter	ref readonly parameter	in parameter	out parameter
ref	Allowed	Allowed	Warning	Error
in	Error	Allowed	Allowed	Error
out	Error	Error	Error	Allowed
No annotation	Error	Warning	Allowed	Error

Value kind	ref parameter	ref readonly parameter	in parameter	out parameter
rvalue	Error	Warning	Allowed	Error
lvalue	Allowed	Allowed	Allowed	Allowed

Where lvalue means a variable (i.e., a value with a location; does not have to be writable/assignable) and rvalue means any kind of value.

Example

```
using System;
namespace Console12App1  {
    class Program    {
        static int sum(in int x, ref int y, ref readonly int z)   {
            // x++; // x is an in parameter, so readonly
            y++; //possible as y is ref, so y is reference parameter
            // z++; // z is a ref readonly parameter, so called as ref but works like in
            return x + y + z;
        }
        static void Main(string[] args)     {
            int x = 10, y = 20, z = 30;
            var res = sum(in x, ref y, ref z);
            Console.WriteLine(res); // Output: 61
        }
} }
```

Alias Any Type with using

- The using directive has been extended to allow aliases for any type, not just named types. This means you can create semantic aliases for complex types like tuples, arrays, pointers, or other unsafe types.
- Extending the using directive to alias any type, including complex types like tuples and arrays.

Why is alias any type useful?

- Alias any type is useful for several reasons. First, it can help you reduce the amount of code you need to write, and make your code more readable and maintainable. For example, if you have a complex type that you use frequently in your code, such as a tuple type, an array type, or a pointer type, you can create a simple alias for it, and use the alias instead of the full type name. This can make your code more concise and clear, and avoid repetition and typos.
- Second, it can help you create semantic aliases for types that have no names, or have names that are not meaningful or descriptive. For example, if you have a tuple type that represents a point, such as (int x, int y), you can create an alias for it, such as Point, and use the alias to express the meaning and intention of the type. This can make your code more self-documenting and understandable, and avoid confusion and ambiguity.
- Third, it can help you create aliases for types that are not available in your current scope, or are difficult to access or import. For example, if you have a type that is defined in another assembly, or in another namespace, or in another project, you can create an alias for it, and use the alias to refer to the type without having to qualify it with the full name or use a using directive. This can make your code more portable and modular, and avoid conflicts and dependencies.
- How to use alias any type?
  - To use alias any type, you need to use the using alias directive, and specify the new name and the type you want to alias. The syntax is as follows:
    - using alias = type;
  - where alias is the new name you want to create, and type is any type you want to alias.

Limitations and rules of alias any type

- There are some limitations and rules you need to be aware of when using alias any type. Here are some of them:
  - You can only create an alias for a type in the same scope where the type is available. For example, you cannot create an alias for a local variable type, or a method parameter type, or a lambda parameter type, because they are not in scope outside of their declaration.
  - You can only create an alias for a type at the namespace level or the compilation unit level. For example, you cannot create an alias for a type inside a class, a struct, an interface, a delegate, an enum, or a record, because they are not valid places for a using directive.
  - You can only create one alias for a type in the same scope. For example, you cannot create two aliases for the same type in the same namespace, or in the same compilation unit, because they would conflict with each other.
  - You can only use an alias for a type in the same scope where the alias is declared. For example, you cannot use an alias for a type in a different namespace, or in a different compilation unit, or in a derived class, or in an external assembly, because they are not visible or accessible from there.
  - You cannot use an alias for a type as a type argument for another generic type.

Example

```
using System;
using Person = System.Collections.Generic.Dictionary<string, object>;
using Point = (int x, int y);
namespace Console12App1  {
    class Program    {
        static void Main(string[] args)     {
            // System.Collections.Generic.Dictionary<string, object> p = new
System.Collections.Generic.Dictionary<string, object>();
            Person p = new Person();
            p["Name"] = "John Doe";
            p["Age"] = 25;
            Console.WriteLine(p["Name"] + " is of age " + p["Age"]);// John Doe is of age
25
            // (int x, int y) pt = (1, 2);
            Point pt = (1, 2);
            Console.WriteLine(pt.x); // 1
            Console.WriteLine(pt.y); // 2
        }
 }  }
```

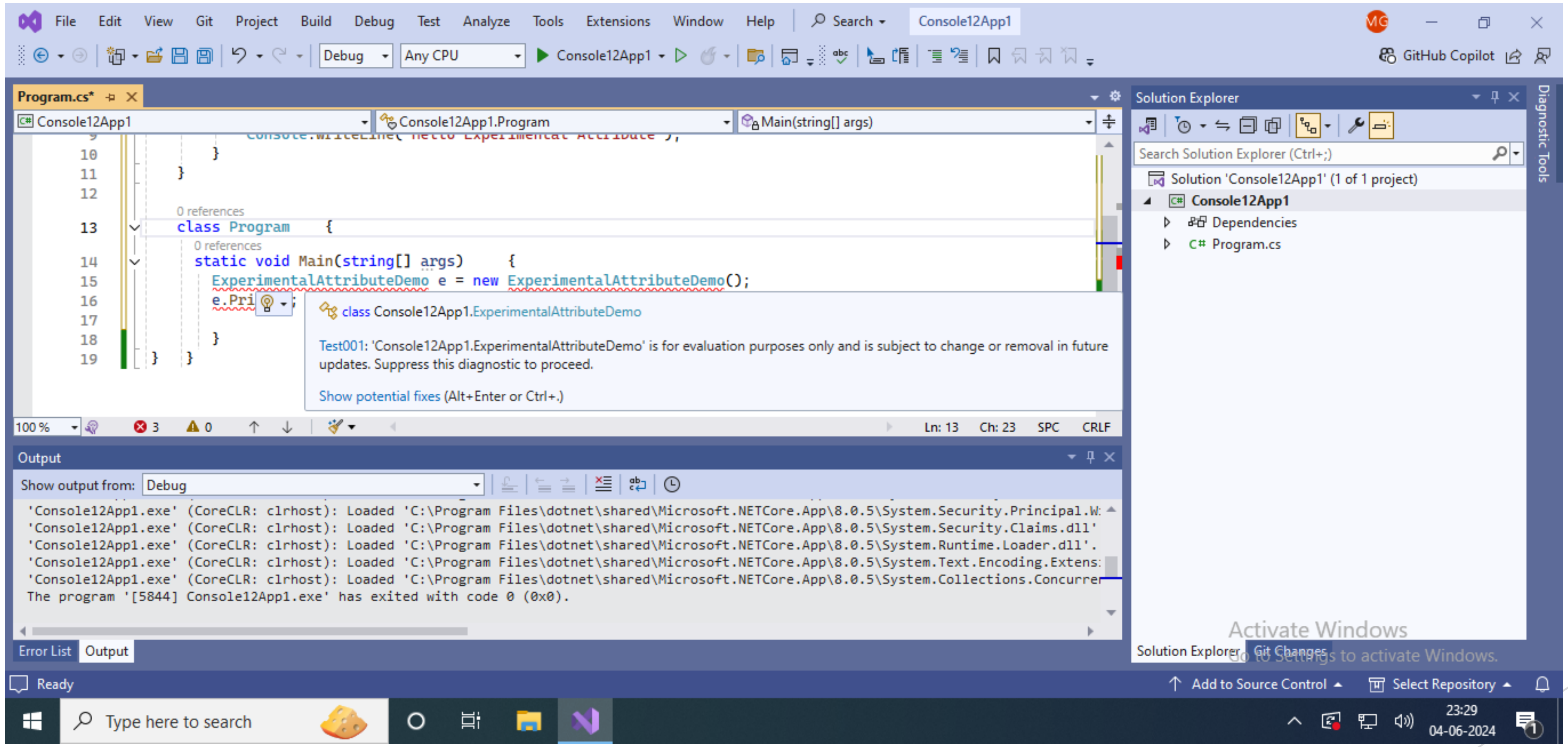
Experimental Attribute

- This new attribute can be used to mark types, methods, or assemblies as experimental. The compiler issues a warning when accessing methods or types annotated with this attribute, making it clearer which parts of your code are experimental and potentially subject to change.
- The compiler issues an error when someone tries to use the method or the type marked Experimental.
- In case we want to further test the experiment we can add:
  - #pragma warning disable Test001
- And later post testing the values, restore:
  - #pragma warning restore Test001

Example

```
using Console12App1;
using System;
using System.Diagnostics.CodeAnalysis;
namespace Console12App1  {
    [Experimental(diagnosticId: "Test001")]
    public class ExperimentalAttributeDemo   {
        [Experimental(diagnosticId: "Test002")]
        public void Print()   {
            Console.WriteLine("Hello Experimental Attribute");
        }
    }
    class Program    {
      static void Main(string[] args)     {
#pragma warning disable Test001
            ExperimentalAttributeDemo e = new ExperimentalAttributeDemo();
#pragma warning disable Test002
            e.Print(); // output : Hello Experimental Attribute
#pragma warning restore Test001
        }
    }  }
```

Example..



```
        }
    }

    0 references
    class Program    {
        0 references
        static void Main(string[] args)     {
            ExperimentalAttributeDemo e = new ExperimentalAttributeDemo();
            e.Pri ;

        }
 }   }
```

class Console12App1.ExperimentalAttributeDemo

Test001: 'Console12App1.ExperimentalAttributeDemo' is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed.

Show potential fixes (Alt+Enter or Ctrl+.)

Output — Show output from: Debug

```
'Console12App1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.Security.Principal.W:
'Console12App1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.Security.Claims.dll'
'Console12App1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.Runtime.Loader.dll'.
'Console12App1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.Text.Encoding.Extens:
'Console12App1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.Collections.Concurrer
The program '[5844] Console12App1.exe' has exited with code 0 (0x0).
```

Improved Interpolated Strings

- In C# 12, there are improvements to interpolated strings, which make them more flexible and powerful. Interpolated strings allow you to embed expressions directly into string literals, making string formatting easier and more readable. Let's explore the improvements introduced in C# 12 for interpolated strings.
- 1. Formatted Interpolated Strings
  - In C# 12, you can apply format strings directly within interpolated strings. This feature simplifies the process of formatting values inside interpolated strings without using additional string manipulation methods.
- 2. Interpolated String with Conditional Expressions
  - You can use conditional expressions (? :) directly inside interpolated strings, allowing you to conditionally include parts of the string based on a condition.
- 3. Interpolated String with Null-conditional Operator
  - You can use the null-conditional operator (?.) directly inside interpolated strings to safely access members of objects that may be null.
- 4. Interpolated String with Index and Range
  - In C# 12, you can use index and range expressions (^ and ..) directly inside interpolated strings to access parts of strings or arrays.
- Benefits of Improved Interpolated Strings
  - Simplicity: Interpolated strings now support more features directly, reducing the need for additional string manipulation.
  - Readability: Code is more readable and concise, making it easier to understand and maintain.
  - Flexibility: You can embed complex expressions and conditions directly inside interpolated strings.
- These improvements in C# 12 enhance the capabilities of interpolated strings, making them more powerful and easier to use for string formatting and text manipulation tasks. They are available starting from C# 12 and require .NET 6 or newer. Ensure that your development environment and project configuration support these versions for optimal usage.

Example: Formatted Interpolated Strings

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static void Main()       {
            string name = "Alice";
            int age = 30;

            // Old way
            string oldMessage = $"Hello, {name}! You are {age} years old.";

            // New way with format specifier
            string newMessage = $"Hello, {name}! You are {age:D3} years old."; // D3 ensures age is at
least three digits

            Console.WriteLine(oldMessage); // Output: Hello, Alice! You are 30 years old.
            Console.WriteLine(newMessage); // Output: Hello, Alice! You are 030 years old.
        }
    }
}
```

Example: Interpolated String with Conditional Expressions

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static void Main()        {
            int temperature = 25;
            string warm = "Warm", cold = "Cold";

            string message = $"The weather is {(temperature > 20 ? warm : cold)}. " +
                                 $"The temperature is {temperature} degrees Celsius.";

            Console.WriteLine(message); // Output: The weather is warm. The temperature is 25
degrees Celsius.
        }
    }
}
```

Example: Interpolated String with Null-conditional Operator

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static void Main()        {
            string? name = null;
            int? age = 30;

            string message = $"Hello, {name ?? "Guest"}! " +
                             $"Your age is {age?.ToString() ?? "unknown"}.";

            Console.WriteLine(message); // Output: Hello, Guest! Your age is 30.
        }
    }
}
```

Example: Interpolated String with Index and Range

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222 {
    class Program   {
        public static void Main()       {
            string name = "Alice";
            int age = 30;

            string message = $"Hello, {name[..3]}! " +
                             $"You are {age} years old.";

            Console.WriteLine(message); // Output: Hello, Ali! You are 30 years old.
        }
    }
}
```

Scoped Values

- In C# 12, scoped values are introduced as a new feature to simplify the management of local variables in your code, particularly within small, limited scopes.
- This feature enhances the readability and maintainability of your code by reducing the scope of variables to where they are actually needed.
- Scoped values allow you to declare and initialize variables within a limited scope using a simplified syntax.
- This is particularly useful when you want to isolate variables to a specific block of code, such as a conditional statement or a loop, without having to define them at the method level.
- Scoped values in C# 12 are a useful addition for managing variables within limited scopes, enhancing code readability and maintainability.
- They provide a modern syntax for declaring variables closer to their usage, reducing the risk of unintended side effects caused by variables with broader scopes.
- In C# 12, the scoped modifier is indeed used for ref and ref struct values only, not for general variables.

.. Scoped Values

- Benefits of Scoped Values
  - **Improved Readability**: Variables are declared closer to their use, making it easier to understand their purpose.
  - **Reduced Scope**: Limits the visibility of variables to where they are actually needed, reducing potential bugs caused by unintended modifications.
  - **Easier Maintenance**: Code becomes more maintainable as it is clearer where variables are defined and used.
- Considerations
  - **Scope Limitations**: Scoped variables are limited to the block in which they are defined (e.g., a loop or conditional block). They cannot be accessed outside of that block.
  - **Compatibility**: Scoped values are available starting from C# 12 and require .NET 6 or newer. Ensure that your development environment and project configuration support these versions for optimal usage.

Example: Scoped Values

```
using System;
using System.Diagnostics;
using System.Xml.Linq;
namespace ConApp1222    {
    public ref struct RefStruct   {
        public int Value { get; set; }
    }
    class Program    {
        public static void Main()        {
            // Using scoped with ref struct
            scoped RefStruct refStruct = new RefStruct { Value = 10 };
            refStruct.Value = 20;

            Console.WriteLine($"Value: {refStruct.Value}"); // Output: Value: 20        }
        }
    }
}
```

ASP.NET Core 8 Improvements and new features

C#12

ASP.NET Core 8: Improvements and New Features

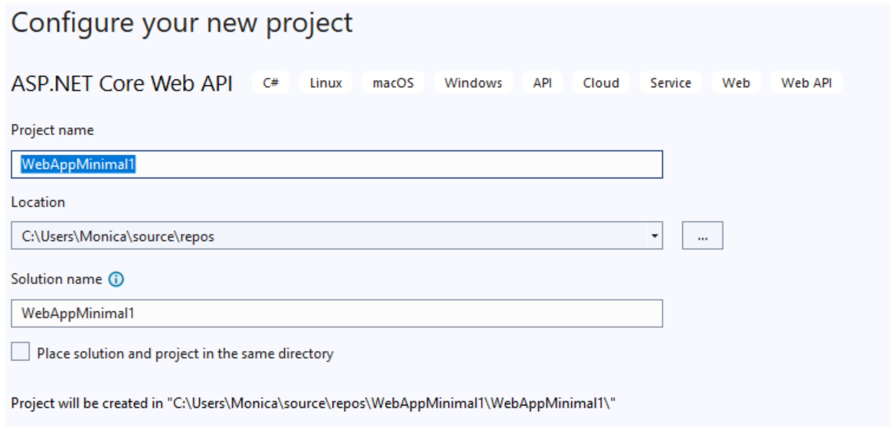
- ASP.NET Core 8 brings several enhancements and new features that aim to improve performance, developer productivity, and the overall experience of building web applications. Below are some of the key improvements and new features in ASP.NET Core 8.
- 1. Enhanced Minimal APIs
- 2. Native AOT Support
- 3. Improved Blazor Features
- 4. Enhanced Performance and Scalability
- 5. Security Enhancements
- 6. Developer Productivity Improvements
- ASP.NET Core 8 brings a plethora of improvements and new features that enhance performance, security, and developer productivity.
- From enhanced minimal APIs and Native AOT support to improved Blazor capabilities and advanced security features, ASP.NET Core 8 continues to be a robust and versatile framework for building modern web applications.
- Whether you are building APIs, web apps, or hybrid apps, these enhancements will help you deliver high-quality, performant, and secure solutions.

Enhanced Minimal APIs

- Improved Routing and Parameter Binding:
  - Minimal APIs in ASP.NET Core 8 offer more intuitive routing and parameter binding, making it easier to define and work with HTTP endpoints.
- OpenAPI and Swagger Integration:
  - Enhanced support for OpenAPI and Swagger simplifies the process of documenting and testing APIs.
- Filter Support:
  - You can now use filters with minimal APIs to implement cross-cutting concerns such as logging, authentication, and validation.

Example

- Create a new Project in VS 2022 with template: ASP.NET Core Web API
- Add OpenAPI : Microsoft.AspNetCore.OpenApi from NuGet packages
- For Minimal API, we need to configure services and middleware to support Minimal APIs. So Edit Program.cs
- Run the Application in IIS Express



Configure your new project

ASP.NET Core Web API — C# Linux macOS Windows API Cloud Service Web Web API

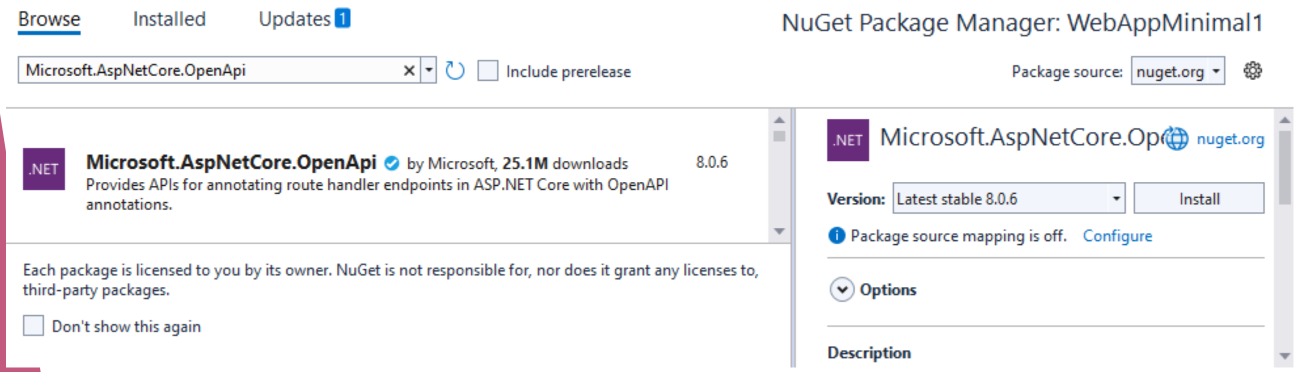
Project name: WebAppMinimal1

Location: C:\Users\Monica\source\repos

Solution name: WebAppMinimal1

☐ Place solution and project in the same directory

Project will be created in "C:\Users\Monica\source\repos\WebAppMinimal1\WebAppMinimal1\"



Browse | Installed | Updates 1 — NuGet Package Manager: WebAppMinimal1

Search: Microsoft.AspNetCore.OpenApi ☐ Include prerelease — Package source: nuget.org

Microsoft.AspNetCore.OpenApi by Microsoft, 25.1M downloads — 8.0.6
Provides APIs for annotating route handler endpoints in ASP.NET Core with OpenAPI annotations.

Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, third-party packages.

☐ Don't show this again

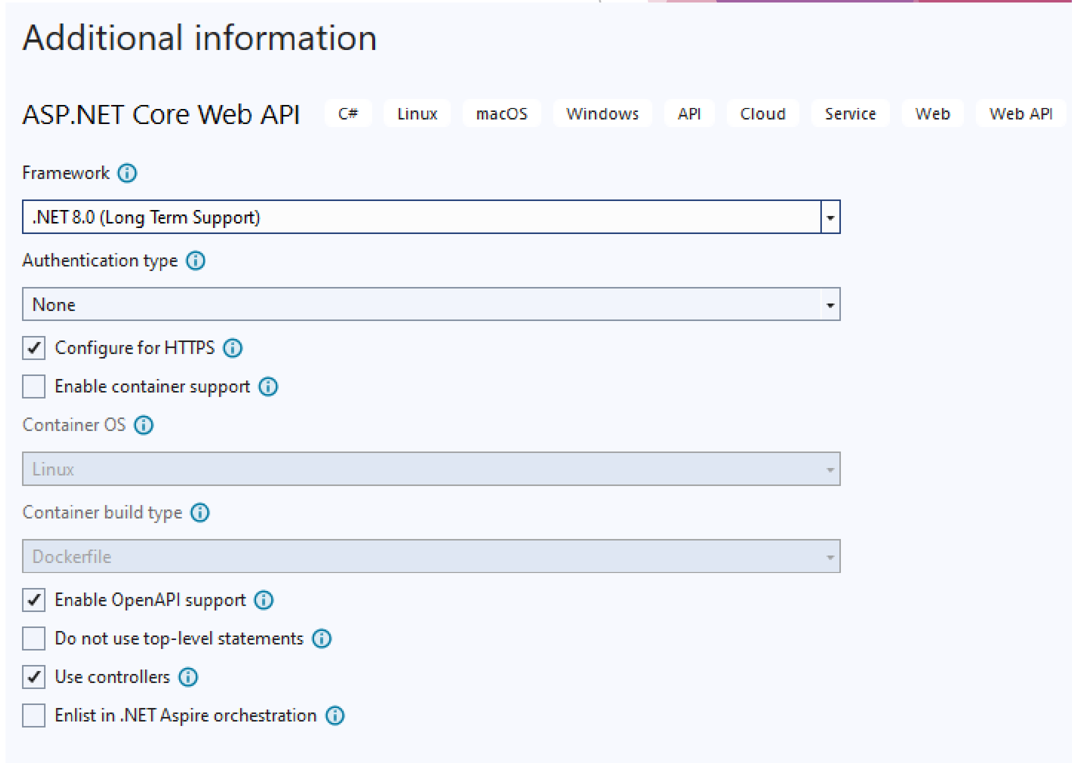
Microsoft.AspNetCore.Op… nuget.org

Version: Latest stable 8.0.6 — Install

Package source mapping is off. Configure

Options

Description



Additional information

ASP.NET Core Web API — C# Linux macOS Windows API Cloud Service Web Web API

Framework: .NET 8.0 (Long Term Support)

Authentication type: None

☑ Configure for HTTPS

☐ Enable container support

Container OS: Linux

Container build type: Dockerfile

☑ Enable OpenAPI support

☐ Do not use top-level statements

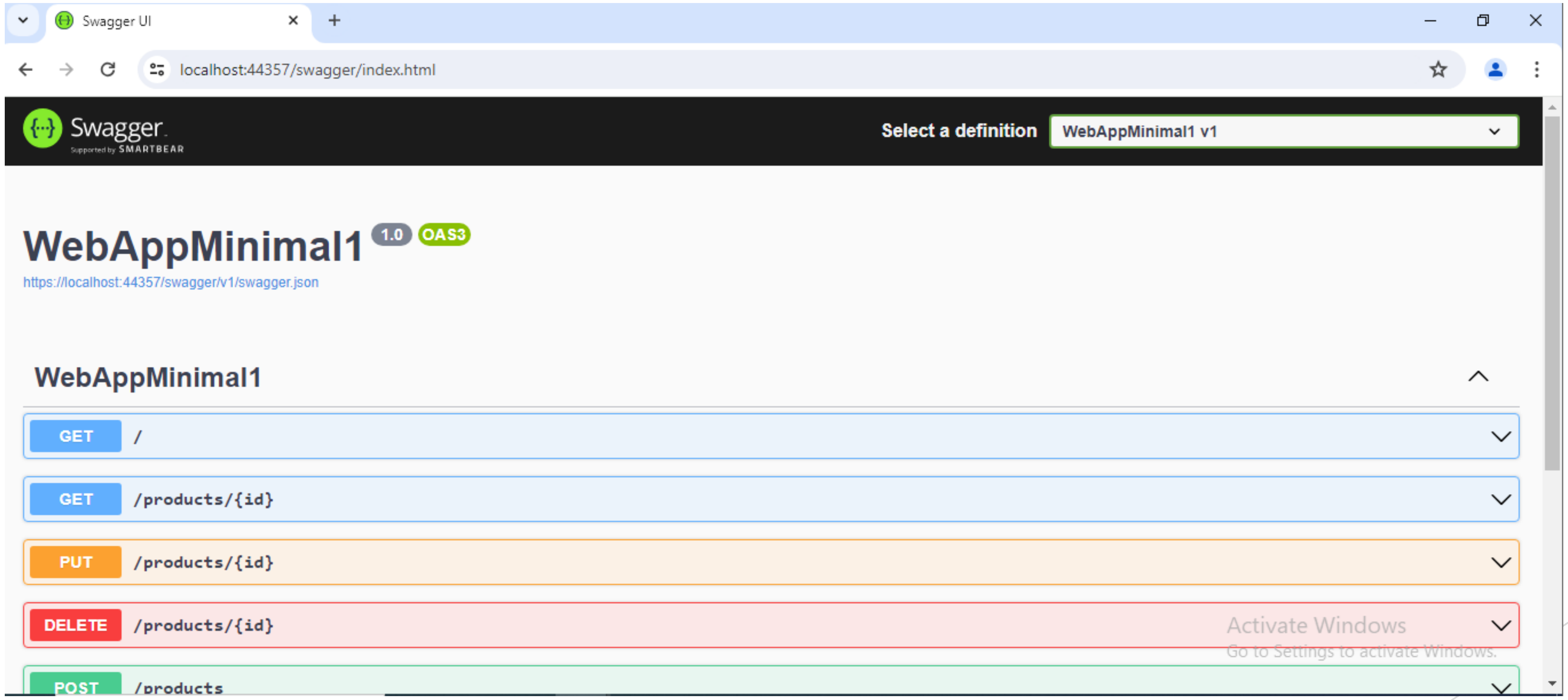
☑ Use controllers

☐ Enlist in .NET Aspire orchestration

Program.cs

```
var builder = WebApplication.CreateBuilder(args);
builder.Services.AddEndpointsApiExplorer();// Add services to the container.
builder.Services.AddSwaggerGen();
var app = builder.Build();
if (app.Environment.IsDevelopment()) {  // Configure the HTTP request pipeline.
    app.UseSwagger();
    app.UseSwaggerUI();
}
app.UseHttpsRedirection();
app.MapGet("/", () => "Hello World!");
app.MapGet("/products/{id}", (int id) =>
{ return Results.Ok(new { Id = id, Name = "Sample Product", Price = 100 }); })
.WithName("GetProduct")
.WithOpenApi(); // Generates OpenAPI documentation
app.MapPost("/products", (Product product) => // Process the product
{return Results.Created($"/products/{product.Id}", product);})
.WithName("CreateProduct")
.WithOpenApi(); // Generates OpenAPI documentation
app.MapPut("/products/{id}", (int id, Product updatedProduct) => // Update the product with the given ID
{ return Results.NoContent(); })
.WithName("UpdateProduct")
.WithOpenApi(); // Generates OpenAPI documentation
app.MapDelete("/products/{id}", (int id) =>   // Delete the product with the given ID
{ return Results.NoContent();})
.WithName("DeleteProduct")
.WithOpenApi(); // Generates OpenAPI documentation
app.Run();
record Product(int Id, string Name, decimal Price);
```

Output


Swagger UI

localhost:44357/swagger/index.html

Select a definition WebAppMinimal1 v1

WebAppMinimal1 1.0 OAS3

https://localhost:44357/swagger/v1/swagger.json

WebAppMinimal1

GET /

GET /products/{id}

PUT /products/{id}

DELETE /products/{id}

POST /products

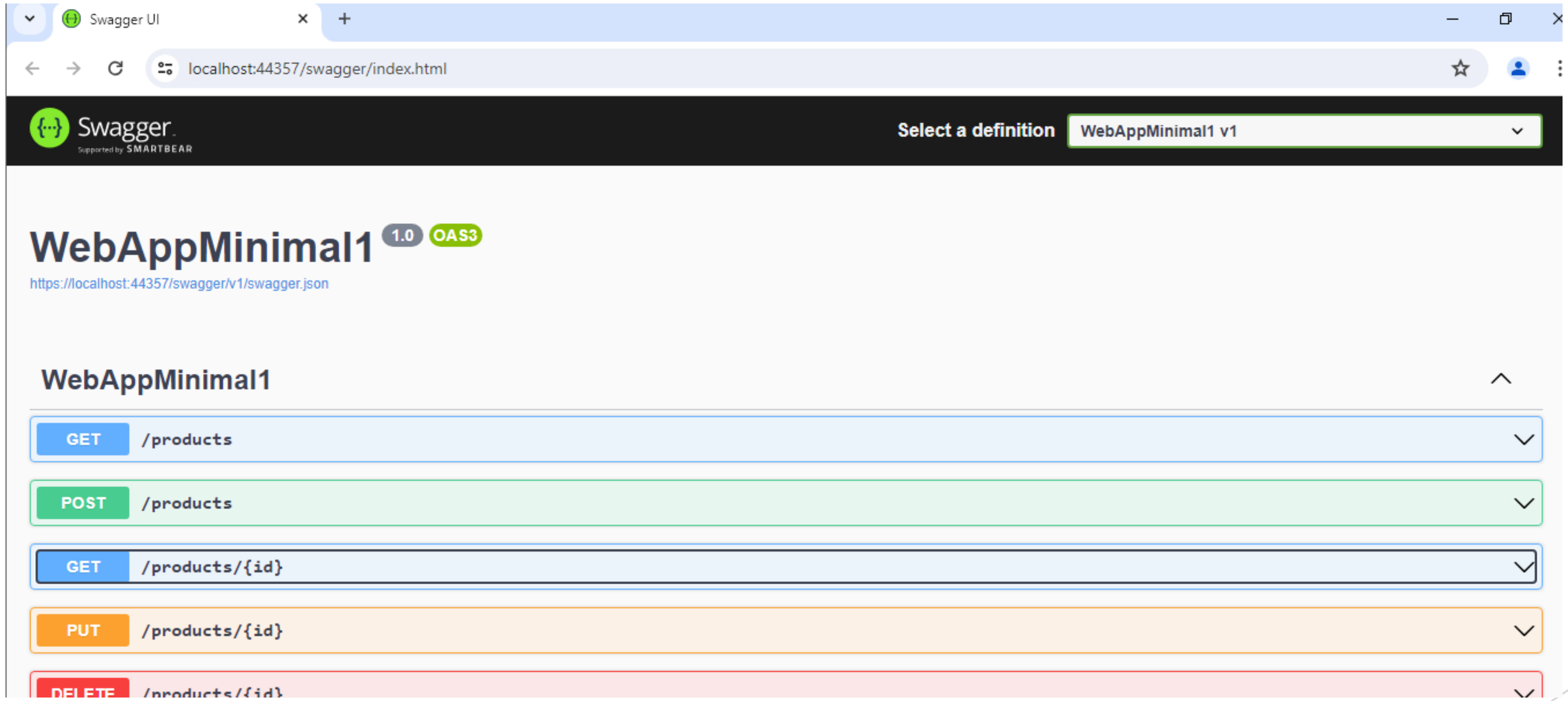
Activate Windows
Go to Settings to activate Windows.

CRUD Operation with Minimal API Program.cs

Program.cs

Output

- Add Products, Delete Products, View a Product, View all Products, Edit a Product



WebAppMinimal1 1.0 OAS3

https://localhost:44357/swagger/v1/swagger.json

WebAppMinimal1

```
GET     /products
POST    /products
GET     /products/{id}
PUT     /products/{id}
DELETE  /products/{id}
```

OpenAPI : Important points

- OpenAPI define a standard for building APIs.
- Swagger tool is commonly used to generate OpenAPI documentation.
- HTML is NOT a valid OpenAPI document format
- The main benefit of using OpenAPI in .NET 8 for API development is Standardized API documentation

Example: CRUD in MVC

- Create a project in .Net8 framework: Asp.Net Web App (MVC)
- Add Models, Controllers, Views for Employee
- Run to check output

Model: Employee.cs

```
namespace Day30MVC.Models
{
    public class Employee
    {
        public string name { get; set; }
        public int id { get; set; }

        public Employee(string name, int id)
        {
            this.name = name;
            this.id = id;
        }

        public Employee() { }

        public override string ToString()
        {
            return "\nEmp: " + name + " with id: " + id;
        }
    }
}
```

Model : EmployeeList.cs

```
using System.Collections.Generic;
using System;
using System.Xml.Linq;

namespace Day30MVC.Models
{
    public class EmployeeList
    {
        public List<Employee> emps = new List<Employee>();
        public EmployeeList(List<Employee> emps) { this.emps = emps; }
        public EmployeeList(Employee emp) { this.emps.Add(emp); }
        public EmployeeList()
        {
            this.AddEmp(new Employee("abc", 101));
            this.AddEmp(new Employee("xyz", 102));
            this.AddEmp(new Employee("abcxyz", 103));
        }
        public void AddEmp(Employee emp) { this.emps.Add(emp); }
        public void AddEmp(string name, int id) { this.emps.Add(new Employee(name,id)); }
        public void DispEmp()
        {
            foreach (Employee e in emps)
            {
                Console.WriteLine(e);
            }
```

EmployeeController

```
using Day30MVC.Models;
using Microsoft.AspNetCore.Mvc;
using System.Xml.Linq;

namespace Day30MVC.Controllers
{
    public class EmployeeController : Controller
    {
        static EmployeeList emps = new EmployeeList();
        public IActionResult Index()
        {
            return View();
        }
        public IActionResult WelcomeEmp(string name)
        {
            TempData["Emp"] = emps.getEmp(name);
            return View();
        }
        public IActionResult WelcomeEmp2(string name)
        {
            TempData["EmpName"] = name;
            return View();
        }
        public IActionResult UpdateEmps(int id, string name)
        {
```

Views: Index

```
<h1>
<a href="/Employee/ListEmp">List of Emp line wise</a>
<br />
<a href="/Employee/ListEmp2">List of Emp as JSON</a>
<Br /><hr />
********For just a welcome to name screen*********
<form action="/Employee/WelcomeEmp2">
Name: <input type="text" name="name" />
<input type="submit" value="WelcomeEmp2" />
</form>
<br />
<hr />
********For Search according to Name**********
<form action="/Employee/WelcomeEmp">
<!-- Name: Monica is a hidden field here.
<input type="hidden" name="name1" value="Monica"/> -->
Name: <input type="text" name="name" />
<input type="submit" value="WelcomeEmp" />
</form>

<br />
<hr />
****For Update***
<form action="/Employee/UpdateEmps">
Id: <input type="text" name="id" /><br />
Name: <input type="text" name="name" /><br />
<input type="submit" value="UpdateEmp" />
</form>
```

Views:AddEmps

- <h1>
-     Employee Added Successfully!
-     <br />
-     <hr />
-     <a href="/Employee/ListEmp">List of Emp line wise</a>
-     <br />
-     <a href="/Employee/Index">Back to Index</a>
- </h1>

Views: DelEmps

- <h1>
- @TempData["msg"] !!
- <br />
- <hr />
- <a href="/Employee/ListEmp">List of Emp line wise</a>
- <br />
- <a href="/Employee/Index">Back to Index</a>
- <br />
- </h1>

Views: ListEmp

```
@model IEnumerable<Day30MVC.Models.Employee>
<h3>
@foreach (var emp in Model)
{
    <div>
        @emp
        <hr />
    </div>
}

<hr />
<br />
<hr />
<a href="/Employee/ListEmp">List of Emp line wise</a>
<br />
<a href="/Employee/Index">Back to Index</a>
<br />
</h3>
```

UpdateEmps

- <h1>
- @TempData["msg"] !!
- <br />
- <hr />
- <a href="/Employee/ListEmp">List of Emp line wise</a>
- <br />
- <a href="/Employee/Index">Back to Index</a>
- <br />
- </h1>

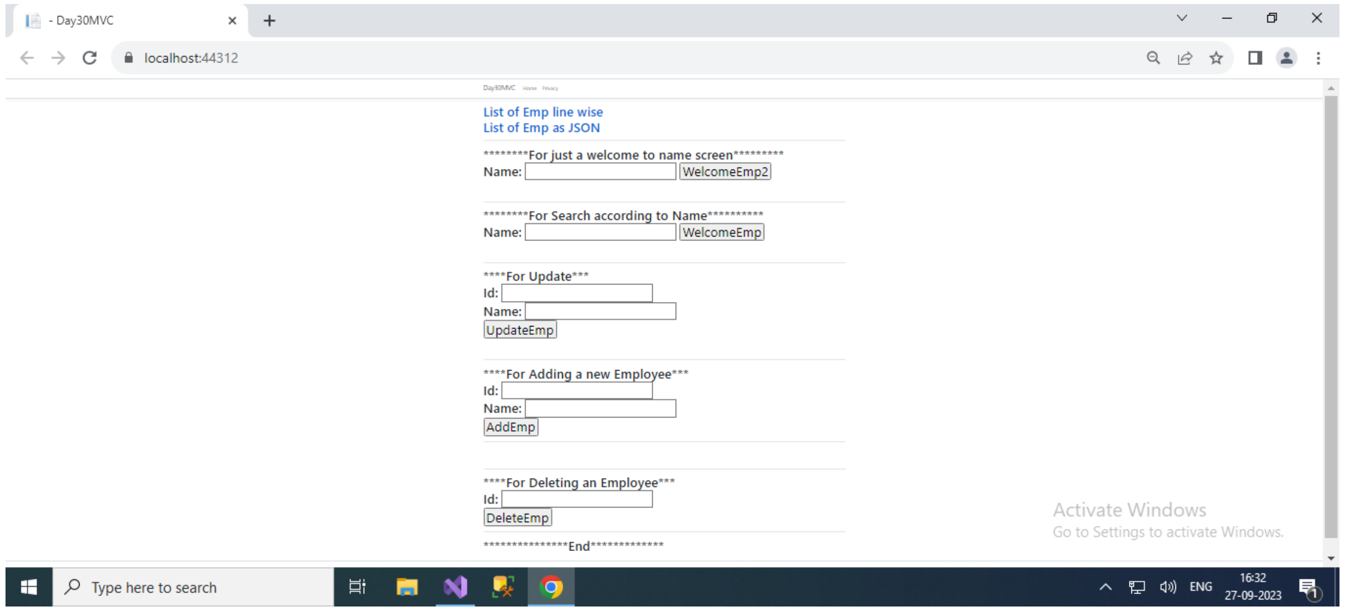
Views: WelcomeEmp

- <h1>
- Welcome @TempData["Emp"] !!
- </h1>

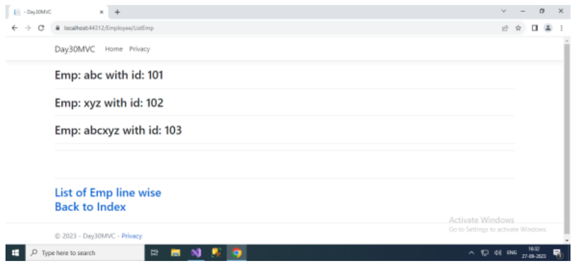
Views: WelcomeEmp2

- <h1>
- Welcome @TempData["EmpName"] !!
- </h1>

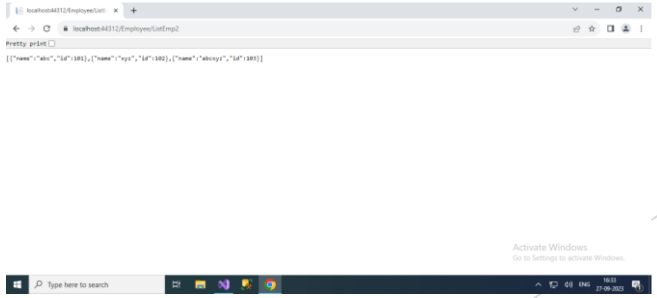
Output



```
List of Emp line wise
List of Emp as JSON
********For just a welcome to name screen*********
Name: [          ] [WelcomeEmp2]

********For Search according to Name*********
Name: [          ] [WelcomeEmp]

****For Update***
Id: [          ]
Name: [          ]
[UpdateEmp]

****For Adding a new Employee***
Id: [          ]
Name: [          ]
[AddEmp]

****For Deleting an Employee***
Id: [          ]
[DeleteEmp]
***************End*************
```



```
Emp: abc with id: 101
Emp: xyz with id: 102
Emp: abcxyz with id: 103

List of Emp line wise
Back to Index

© 2023 - Day30MVC - Privacy
```



```
[{"name":"abc","id":101},{"name":"xyz","id":102},{"name":"abcxyz","id":103}]
```

Native AOT Support

- Native AOT (Ahead-Of-Time) compilation in .NET 8 is a significant enhancement for building high-performance applications.
- **Native AOT** involves compiling .NET code directly to a native binary, improving performance and reducing startup time by eliminating the need for JIT (Just-In-Time) compilation.
- Faster Startup Times:
  - ASP.NET Core 8 applications can leverage Native AOT (Ahead-of-Time) compilation to achieve faster startup times and reduced memory usage.
- Self-contained Executables:
  - Native AOT produces self-contained executables that include all necessary dependencies, improving deployment and portability.
- Supported Scenarios
  - Console Applications: Ideal for command-line tools and utilities.
  - Microservices: Lightweight and fast, suitable for microservice architectures.
  - Blazor WebAssembly: Improved performance for Blazor applications.
- We need to enable Native AOT (Ahead-of-Time) support in .NET 8 using Visual Studio 2022, as of current version.

Key Features of Native AOT

- Performance Improvements:
  - Reduced Startup Time: Applications start faster since there's no JIT compilation.
  - Reduced Memory Usage: No need for JIT code caches, reducing the overall memory footprint.
  - Optimized Execution: Native code execution can be more efficient.
- Deployment Benefits:
  - Self-Contained Executables: Applications are packaged as a single executable, simplifying deployment.
  - Reduced Dependencies: Less dependency on runtime libraries, which simplifies distribution.
- Security Enhancements:
  - Smaller Attack Surface: Native binaries are harder to reverse-engineer compared to IL code.

Configuration and Usage

- Project Setup:
  - Ensure the project targets .NET 8.
  - Add necessary configurations in the project file (.csproj):
  - <PropertyGroup>
  - <PublishAot>true</PublishAot>
  - </PropertyGroup>
- Building the Application:
  - Use the dotnet publish command with the -c Release option to generate the native binary:
  - dotnet publish -c Release
- Runtime Configuration:
  - Native AOT requires runtime configuration to optimize the AOT process.
  - Use the runtimeconfig.json file to specify settings.

Example

- Create a New .NET 8 ASP.NET Core Web App (MVC) Project

- Configure Native AOT in the Project
- Edit the Project File (.csproj):
- <Project Sdk="Microsoft.NET.Sdk">
-   <PropertyGroup>
-     <OutputType>Exe</OutputType>
-     <TargetFramework>net8.0</TargetFramework>
-     <PublishAot>true</PublishAot>
-     <RuntimeIdentifier>win-x64</RuntimeIdentifier> <!-- Change as per your target platform -->
-     <SelfContained>true</SelfContained>
-   </PropertyGroup>
- </Project>
- Ensure the <PublishAot> property is set to true, which instructs the compiler to use Ahead-of-Time compilation.

..Example

- Build and Publish the Application
- Build the Project:
  - Press Ctrl+Shift+B to build the project, ensuring there are no errors.
- Publish the Application:
  - Right-click on your project in the Solution Explorer and select Publish.
  - Follow the steps in the Publish wizard to publish your application.
  - Click Publish button
  - Alternatively, you can use the Package Manager Console or Terminal to publish your application:
    - dotnet publish -c Release -r win-x64 --self-contained
  - Adjust the -r parameter for your specific runtime identifier (e.g., linux-x64, osx-x64).

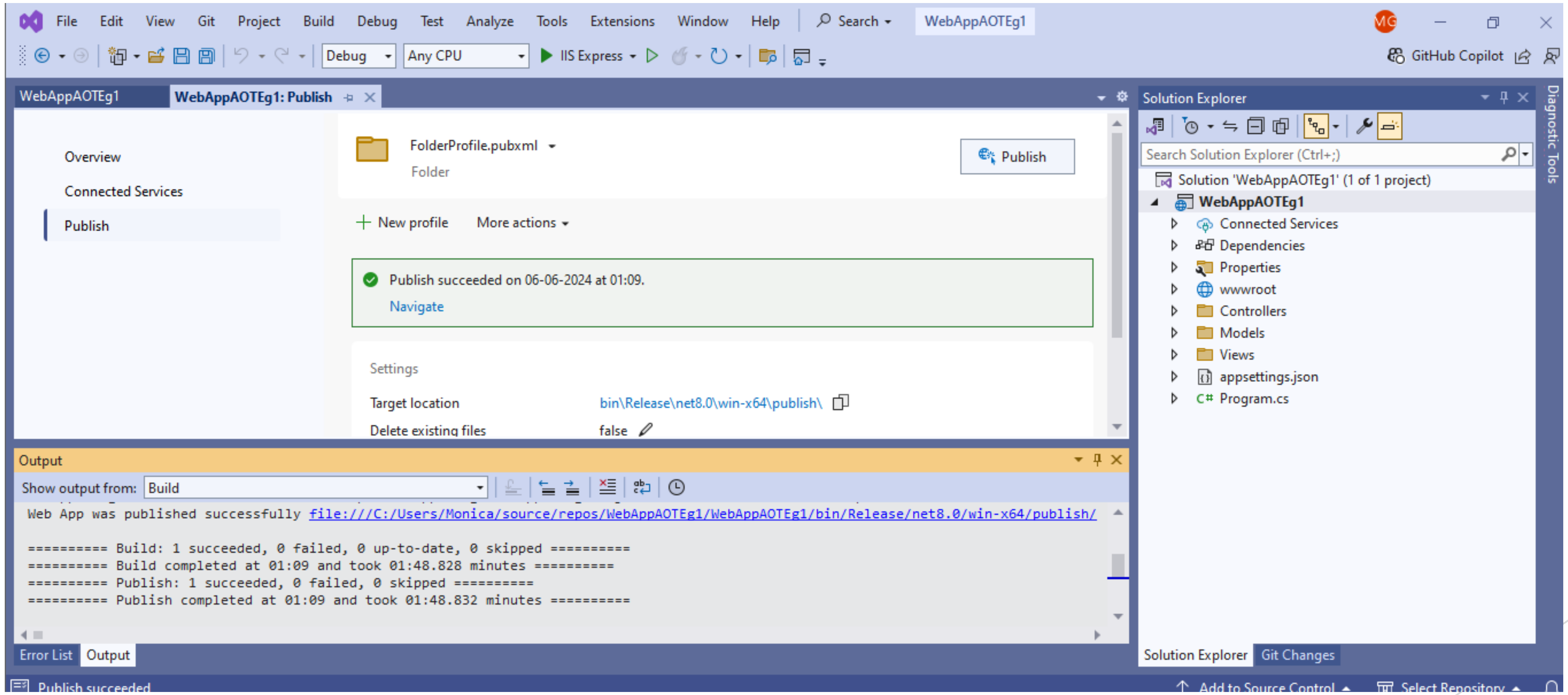
..Example

- Verify the Published Output
- Locate the Published Output:
  - The published output will be located in the bin\Release\net8.0\win-x64\publish directory (adjust the path based on your configuration and runtime identifier).
- Run the Application:
  - Navigate to the published directory and run the executable. The application should start up quickly, demonstrating the performance benefits of Native AOT.

Click on project and add in XML:

```
<Project Sdk="Microsoft.NET.Sdk.Web">

  <PropertyGroup>
    <TargetFramework>net8.0</TargetFramework>
    <Nullable>enable</Nullable>
    <ImplicitUsings>enable</ImplicitUsings>

  <OutputType>Exe</OutputType>

    <PublishAot>true</PublishAot>
    <RuntimeIdentifier>win-x64</RuntimeIdentifier> <!-- Change as per your target platform -->

    <SelfContained>true</SelfContained>

  </PropertyGroup>

</Project>
```

Output: Publish



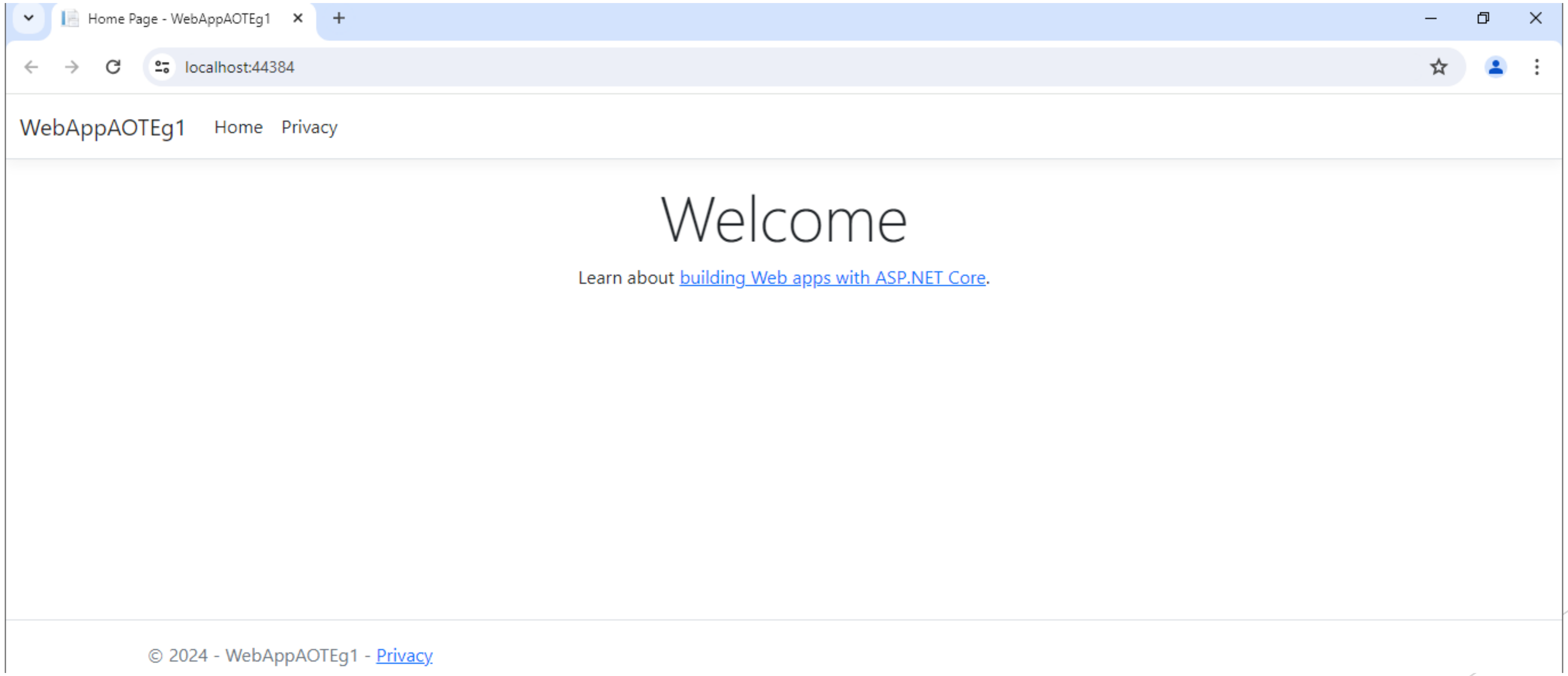
Publish succeeded on 06-06-2024 at 01:09.
Navigate

Settings

Target location	bin\Release\net8.0\win-x64\publish\
Delete existing files	false

```
Web App was published successfully file:///C:/Users/Monica/source/repos/WebAppAOTEg1/WebAppAOTEg1/bin/Release/net8.0/win-x64/publish/

========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
========== Build completed at 01:09 and took 01:48.828 minutes ==========
========== Publish: 1 succeeded, 0 failed, 0 skipped ==========
========== Publish completed at 01:09 and took 01:48.832 minutes ==========
```

Output: Run



WebAppAOTEg1 Home Privacy

Welcome

Learn about building Web apps with ASP.NET Core.

© 2024 - WebAppAOTEg1 - Privacy

Improved Blazor Features

- Blazor Hybrid Apps:
  - Enhanced support for Blazor hybrid apps allows for seamless integration of Blazor components in WPF, Windows Forms, and MAUI applications.
- JavaScript Interoperability:
  - Improved interop with JavaScript enables more efficient communication between Blazor components and JavaScript libraries.
- Performance Enhancements:
  - Various performance improvements in Blazor components reduce latency and enhance the user experience.
- We will do the Practical example in a later session.

Enhanced Performance and Scalability

- .NET 8 brings significant enhancements to performance and scalability, making it a robust choice for developing high-performance applications
- Improved Caching:
  - New and improved caching mechanisms help in optimizing the performance of ASP.NET Core applications.
- Connection Resiliency:
  - Enhanced connection resiliency features ensure that applications can handle transient failures and maintain connectivity.
- HTTP/3 Support:
  - Full support for HTTP/3 enhances performance for web applications by providing faster and more reliable connections.

Key Performance and Scalability Enhancements

- Runtime Performance Improvements:
  - JIT (Just-In-Time) Compiler Enhancements: Improved code generation and optimizations.
  - Tiered Compilation: Enhanced tiered compilation strategies to balance startup time and long-term performance.
- Garbage Collection (GC) Enhancements:
  - Optimized GC Algorithms: Improved algorithms for better memory management.
  - Low-Latency GC: Configurations for applications requiring minimal GC-induced pauses.
- Async and Task Parallel Library (TPL) Improvements:
  - Async Overhead Reduction: Reduced overhead in async/await patterns.
  - Enhanced Task Scheduling: Improvements in task scheduling and execution.
- New APIs and Enhancements:
  - Span<T> and Memory<T>: Enhanced support for high-performance memory operations.
  - Hardware Intrinsics: More intrinsics for leveraging hardware capabilities.
  - I/O Performance Improvements: Optimized file and network I/O operations.
- Profiling and Diagnostics:
  - Improved Profiling Tools: Enhanced profiling tools for better performance diagnostics.
  - EventCounters: More granular event counters for monitoring performance metrics.

Using The Features

- Performance Testing:
  - Conduct thorough performance testing using benchmarking tools like BenchmarkDotNet.
  - Profile applications to identify and address performance bottlenecks.
- Memory Management:
  - Minimize allocations and use memory-efficient structures like Span<T> and ArrayPool<T>.
  - Optimize GC settings based on application requirements.
- Concurrency:
  - Use async programming effectively to avoid blocking threads.
  - Leverage the Task Parallel Library for concurrent operations.
- Code Analysis:
  - Regularly analyze code for performance issues using static analyzers and profilers.
  - Apply best practices for high-performance coding.

Security Enhancements

- ASP.NET in .NET 8 introduces significant security enhancements aimed at providing robust protection for web applications.
- Enhanced Authentication and Authorization:
  - New and improved features for authentication and authorization provide better security and easier implementation of security policies.
- Rate Limiting:
  - Built-in support for rate limiting helps protect your applications from abuse and ensures fair usage of resources.
- Enhanced Data Protection:
  - Improved data protection mechanisms ensure that sensitive information is securely handled and stored.

Key Security Enhancements

- Improved Authentication and Authorization:
  - Support for OAuth 2.1 and OpenID Connect: Enhanced support for modern authentication protocols.
  - Role-based and Policy-based Authorization: Fine-grained control over access to resources.
- Enhanced Data Protection:
  - Data Protection API Enhancements: Improved encryption mechanisms and key management.
  - Secure Data Storage: Enhanced options for secure storage of sensitive data.
- Security by Default:
  - HTTPS Enforcement: Default configuration to enforce HTTPS.
  - SameSite Cookie Policy: Improved default settings for cookie security.
  - CORS (Cross-Origin Resource Sharing) Enhancements: Better default configurations to prevent cross-origin attacks.
- Advanced Threat Protection:
  - Rate Limiting and Throttling: Built-in mechanisms to prevent denial-of-service attacks.
  - Enhanced Logging and Monitoring: Improved logging for security events and integration with monitoring tools.
  - Security Headers: Improved default HTTP security headers.
- Dependency Management:
  - Improved Package Management: Enhanced mechanisms to ensure dependencies are secure and up-to-date.
  - Supply Chain Security: Measures to prevent and mitigate supply chain attacks.

JWT Authentication and Authorization

- In .NET 8, you can use JWT authentication to secure your APIs without a database server.
- Configure Authentication and Authorization:
  - Update the ConfigureServices and Configure methods in Startup.cs to configure JWT authentication and authorization.

How to Secure Apps: Practices

- Regular Security Audits:
  - Conduct regular security audits and penetration tests.
  - Use tools like OWASP ZAP and Burp Suite for vulnerability scanning.
- Secure Coding Practices:
  - Follow secure coding guidelines and practices.
  - Regularly review and update dependencies to fix known vulnerabilities.
- Access Control:
  - Implement least privilege access control.
  - Regularly review and update role-based access policies.
- Monitoring and Logging:
  - Enable detailed logging for security events.
  - Integrate with monitoring tools like Azure Monitor and Splunk.
- Data Encryption:
  - Encrypt sensitive data at rest and in transit.
  - Use strong encryption algorithms and manage keys securely.

Example:

- Create a new ASP.net web project(MVC).
- Install the following from NuGet packages:
  - dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version 8.0.0
  - dotnet add package Microsoft.IdentityModel.Tokens --version 6.15.0
  - dotnet add package System.IdentityModel.Tokens.Jwt --version 6.15.0

Edit program.cs

```
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Hosting;
using System.Reflection.PortableExecutable;
using WebAppAuth1;
namespace WebAppAuth1 {
    public class Program    {
        public static void Main(string[] args)        {
            CreateHostBuilder(args).Build().Run();
        }
        public static IHostBuilder CreateHostBuilder(string[] args) =>
              Host.CreateDefaultBuilder(args)
               .ConfigureWebHostDefaults(webBuilder =>
               {
                   webBuilder.UseStartup<Startup>();
                   webBuilder.UseKestrel(options =>
                   {
                       options.ListenAnyIP(5000, listenOptions =>
                       {
                           listenOptions.UseHttps("certificate.pfx", "apples");
                       });
                   });
               });
    }    }
```

Code startup.cs

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.IdentityModel.Tokens;
using System.Text;
namespace WebAppAuth1   {
    public class Startup    {
        public Startup(IConfiguration configuration)    {    Configuration = configuration;    }
        public IConfiguration Configuration { get; }
        public void ConfigureServices(IServiceCollection services)    {
            var secretKey = Configuration["JwtSettings:SecretKey"];
            services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)     // Add authentication
                    .AddJwtBearer(options =>  {
                        options.TokenValidationParameters = new TokenValidationParameters    {
                            ValidateIssuer = true,
                            ValidateAudience = true,
                            ValidateLifetime = true,
                            ValidateIssuerSigningKey = true,
                            ValidIssuer = "https://localhost:5001", // Replace with your actual issuer URL
                            ValidAudience = "your-api", // Replace with your actual audience
                            IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("apples"))
                        };
                    });
                services.AddAuthorization(); // Add authorization
            services.AddControllersWithViews();// Other services
        }
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env) {
            app.UseAuthentication(); // Other configurations
            app.UseAuthorization();
            app.UseRouting(); // Add this line
            app.UseAuthentication(); // Add this line
            app.UseAuthorization(); // Add this line
            app.UseEndpoints(endpoints =>   // Other middleware
            {           endpoints.MapControllerRoute(
                        name: "default",
                        pattern: "{controller=Home}/{action=Index}/{id?}");
                });
            }    }    }
```

Note that if startup.cs is not there,
add startup.cs as a start class to the project
Add -> new Item -> start class -> startup.cs

Add secret key

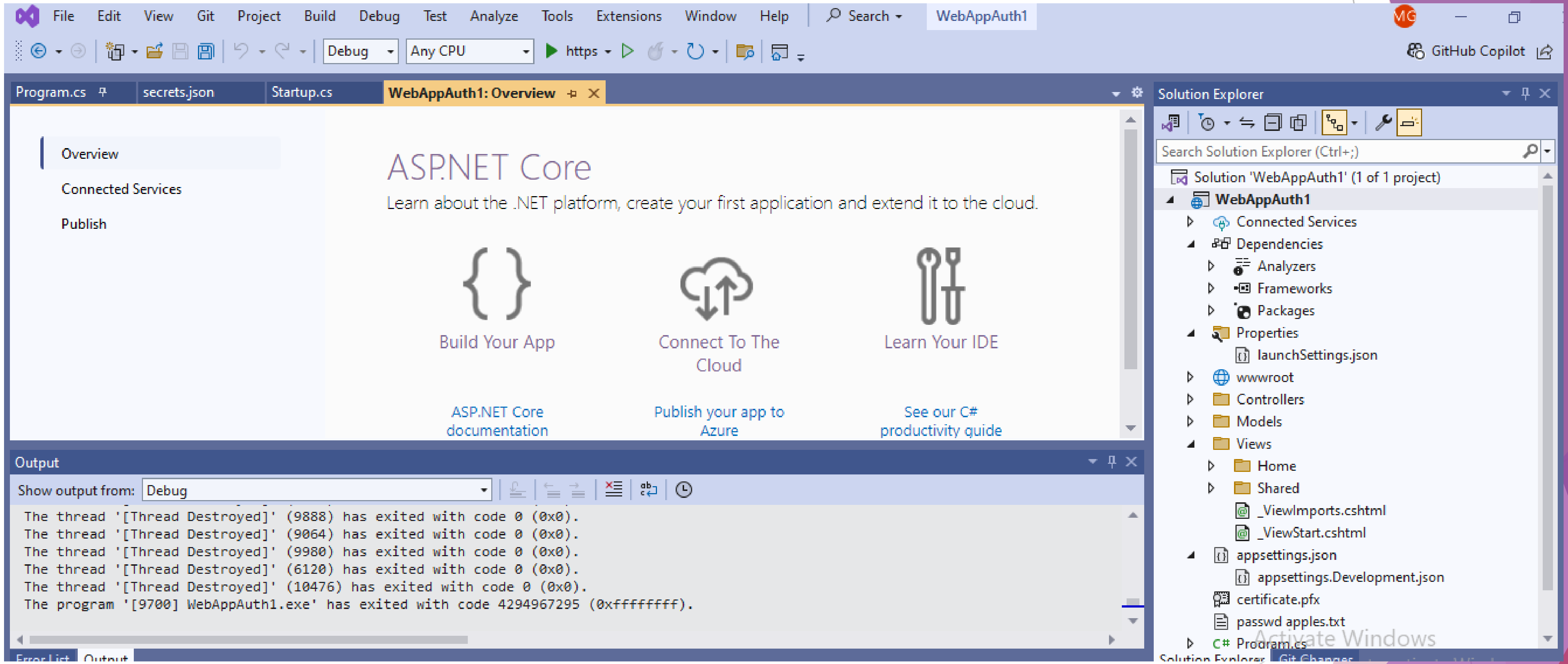
- Right click project-> Manage user secrets-> we get a secret.json generated.
- Add in secret.json:

```
{
  "JwtSettings": {
    "SecretKey": "apples"
  }
}
```

Add Certificate.pfx file downloaded

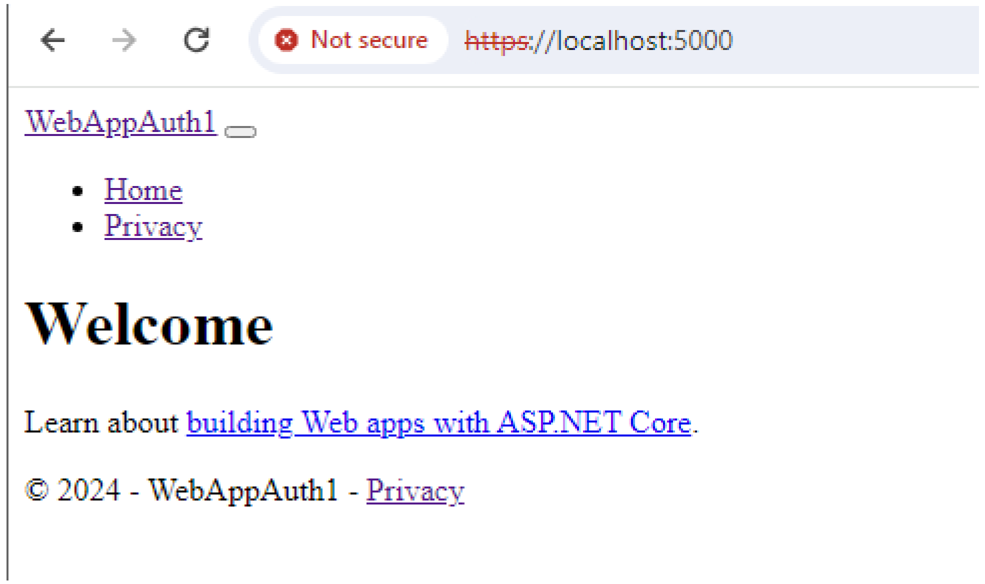
- https://asecuritysite.com/encryption/digitalcert2
- Download the one with apples as password
- Copy the file renamed from fred.pfx to certificate.pfx to the project folder.

Run the Project in HTTPS



```
The thread '[Thread Destroyed]' (9888) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (9064) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (9980) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (6120) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (10476) has exited with code 0 (0x0).
The program '[9700] WebAppAuth1.exe' has exited with code 4294967295 (0xffffffff).
```

Launch the output in chrome



WebAppAuth1

- Home
- Privacy

Welcome

Learn about building Web apps with ASP.NET Core.

© 2024 - WebAppAuth1 - Privacy

Developer Productivity Improvements

- Hot Reload Enhancements:
  - Improved Hot Reload support allows developers to see changes instantly without restarting the application, speeding up the development process.
- Enhanced Debugging and Diagnostics:
  - New tools and features for debugging and diagnostics help identify and resolve issues more quickly.
- Template Improvements:
  - Updated project templates provide a better starting point for new applications, incorporating best practices and new features out of the box.

ASP.NET Core 8 Improvements and new features

- .NET 8 offers enhanced performance, improved diagnostic and observability, expanded cross-platform support, advanced tooling and integration, long-term support (LTS), and much more. ASP.NET Core is the open-source version of ASP.NET. In this article, we will see some of the significant updates and optimizations done in ASP.NET Core in .NET 8.
- 1. Full-stack web UI
  - Blazor is a full-stack web UI framework for developing apps that render content at either the component or page level with:
    - Static Server rendering (also called static server-side rendering, static SSR) to generate static HTML on the server.
    - Interactive Server rendering (also called interactive server-side rendering, interactive SSR) generates interactive components with prerendering on the server.
- 2. New article on class libraries with static server-side rendering (static SSR)
  - Microsoft has added a new article that discusses component library authorship in Razor class libraries (RCLs) with static server-side rendering (static SSR).
- 3. New Article on HTTP Caching Issues
  - A new article discussing HTTP caching issues that can occur when upgrading Blazor apps across major versions and how to address HTTP caching issues has been added in this update.
- 4. New Blazor Web App template
  - The new template provides a single starting point for using Blazor components to build any style of web UI. The template combines the strengths of the existing Blazor Server and Blazor WebAssembly hosting models.

ASP.NET Core 8 Improvements and new features

- 5. New JS initializers for Blazor Web Apps
  - For Blazor Web Apps, a new set of JS initializers are used: beforeWebStart, afterWebStarted, beforeServerStart, afterServerStarted, beforeWebAssemblyStart, and afterWebAssemblyStarted.
- 6. Split of prerendering and integration guidance
  - This time above subjects have been separated into the following new articles, which have been updated for .NET 8:
    - Prerender ASP.NET Core Razor components
    - Integrate ASP.NET Core Razor components into ASP.NET Core apps
- 7. Persist component state in a Blazor Web App
  - You can persist and read component state in a Blazor Web App using the existing PersistentComponentState service. This is useful for persisting component state during prerendering.
- 8. Render Razor components outside of ASP.NET Core
  - One can now render Razor components outside the context of an HTTP request. You can render Razor components as HTML directly to a string or stream independently of the ASP.NET Core hosting environment.
- 9. Root-level cascading values
  - Root-level cascading values can be registered for the entire component hierarchy. Named cascading values and subscriptions for update notifications are supported.
- 10. Content Security Policy (CSP) compatibility
  - Blazor WebAssembly no longer requires enabling the unsafe-eval script source when specifying a Content Security Policy (CSP).

Full-stack web UI

- With the release of .NET 8, Blazor is a full-stack web UI framework for developing apps that render content at either the component or page level with:
  - Static Server rendering (also called static server-side rendering, static SSR) to generate static HTML on the server.
  - Interactive Server rendering (also called interactive server-side rendering, interactive SSR) to generate interactive components with prerendering on the server.
  - Interactive WebAssembly rendering (also called client-side rendering, CSR, which is always assumed to be interactive) to generate interactive components on the client with prerendering on the server.
  - Interactive Auto (automatic) rendering to initially use the server-side ASP.NET Core runtime for content rendering and interactivity. The .NET WebAssembly runtime on the client is used for subsequent rendering and interactivity after the Blazor bundle is downloaded and the WebAssembly runtime activates. Interactive Auto rendering usually provides the fastest app startup experience.
  - Interactive render modes also prerender content by default.

ASP.NET Core Blazor fundamentals

- Some of the concepts are connected to a basic understanding of *Razor components are:*
  - Client and server rendering concepts
  - Static and interactive rendering concepts
  - Razor components
  - Render modes
  - Document Object Model (DOM)

Client and server rendering concepts

- Throughout the Blazor documentation, activity that takes place on the user's system is said to occur on the client or client-side. Activity that takes place on a server is said to occur on the server or server-side.
- The term rendering means to produce the HTML markup that browsers display.
- Client-side rendering (CSR) means that the final HTML markup is generated by the Blazor WebAssembly runtime on the client. No HTML for the app's client-generated UI is sent from a server to the client for this type of rendering. User interactivity with the page is assumed. There's no such concept as static client-side rendering. CSR is assumed to be interactive, so "interactive client-side rendering" and "interactive CSR" aren't used by the industry or in the Blazor documentation.
- Server-side rendering (SSR) means that the final HTML markup is generated by the ASP.NET Core runtime on the server. The HTML is sent to the client over a network for display by the client's browser. No HTML for the app's server-generated UI is created by the client for this type of rendering. SSR can be of two varieties:
  - Static SSR: The server produces static HTML that doesn't provide for user interactivity or maintaining Razor component state.
  - Interactive SSR: Blazor events permit user interactivity and Razor component state is maintained by the Blazor framework.
- Prerendering is the process of initially rendering page content on the server without enabling event handlers for rendered controls. The server outputs the HTML UI of the page as soon as possible in response to the initial request, which makes the app feel more responsive to users. Prerendering can also improve Search Engine Optimization (SEO) by rendering content for the initial HTTP response that search engines use to calculate page rank. Prerendering is always followed by final rendering, either on the server or the client.

Static and interactive rendering concepts

- Razor components are either statically rendered or interactively rendered.
- Static or static rendering is a server-side scenario that means the component is rendered without the capacity for interplay between the user and .NET/C# code. JavaScript and HTML DOM events remain unaffected, but no user events on the client can be processed with .NET running on the server.
- Interactive or interactive rendering means that the component has the capacity to process .NET events via C# code. The .NET events are either processed on the server by the ASP.NET Core runtime or in the browser on the client by the WebAssembly-based Blazor runtime.
- When using a Blazor Web App, most of the Blazor documentation example components require interactivity to function and demonstrate the concepts covered by the articles. When you test an example component provided by an article, make sure that either the app adopts global interactivity or the component adopts an interactive render mode.

Razor components

- Blazor apps are based on Razor components, often referred to as just components. A component is an element of UI, such as a page, dialog, or data entry form. Components are .NET C# classes built into .NET assemblies.
- Razor refers to how components are usually written in the form of a Razor markup page for client-side UI logic and composition. Razor is a syntax for combining HTML markup with C# code designed for developer productivity. Razor files use the .razor file extension.
- Although some Blazor developers and online resources use the term "Blazor components," the documentation avoids that term and universally uses "Razor components" or "components."

Conventions for showing and discussing components

- Project code, file paths and names, project template names, and other specialized terms are in United States English and usually code-fenced.
- Components are usually referred to by their C# class name (Pascal case) followed by the word "component." For example, a typical file upload component is referred to as the "FileUpload component."
- Usually, a component's C# class name is the same as its file name.
- Routable components usually set their relative URLs to the component's class name in kebab-case. For example, a FileUpload component includes routing configuration to reach the rendered component at the relative URL /file-upload. Routing and navigation is covered in ASP.NET Core Blazor routing and navigation.
- When multiple versions of a component are used, they're numbered sequentially. For example, the FileUpload3 component is reached at /file-upload-3.
- Razor directives at the top of a component definition (.razor file) are placed in the following order: @page, @rendermode (.NET 8 or later), @using statements, other directives in alphabetical order. Additional information on Razor directive ordering is found in the Razor syntax section of ASP.NET Core Razor components.
- Access modifiers are used in article examples. For example, fields are private by default but are explicitly present in component code. For example, private is stated for declaring a field named maxAllowedFiles as private int maxAllowedFiles = 3;.
- Component parameter values lead with a Razor reserved @ symbol, but it isn't required. Literals (for example, boolean values), keywords (for example, this), and null as component parameter values aren't prefixed with @, but this is also merely a documentation convention. Your own code can prefix literals with @ if you wish.
- Generally, examples adhere to ASP.NET Core/C# coding conventions and engineering guidelines.

Render modes

- Every component in a Blazor Web App adopts a render mode to determine the hosting model that it uses, where it's rendered, and whether or not it's rendered statically on the server, rendered with for user interactivity on the server, or rendered for user interactivity on the client (usually with prerendering on the server).
- Blazor Server and Blazor WebAssembly apps for ASP.NET Core releases prior to .NET 8 remain fixated on hosting model concepts, not render modes. Render modes are conceptually applied to Blazor Web Apps in .NET 8 or later.
- The following table shows the available render modes for rendering Razor components in a Blazor Web App. Render modes are applied to components with the @rendermode directive on the component instance or on the component definition. It's also possible to set a render mode for the entire app.

Name	Description	Render location	Interactive
Static Server	Static server-side rendering (static SSR)	Server	❌ No
Interactive Server	Interactive server-side rendering (interactive SSR) using Blazor Server	Server	✓ Yes
Interactive WebAssembly	Client-side rendering (CSR) using Blazor WebAssembly†	Client	✓ Yes
Interactive Auto	Interactive SSR using Blazor Server initially and then CSR on subsequent visits after the Blazor bundle is downloaded	Server, then client	✓ Yes

Document Object Model (DOM)

- The Document Object Model (DOM) is a programming interface for web documents. It represents the page so that programs can change the document structure, style, and content.
- The DOM represents the document as nodes and objects; that way, programming languages can interact with the page.
- A web page is a document that can be either displayed in the browser window or as the HTML source. In both cases, it is the same document but the Document Object Model (DOM) representation allows it to be manipulated.
- As an object-oriented representation of the web page, it can be modified with a scripting language such as JavaScript.

New article on class libraries with static server-side rendering (static SSR)

- We've added a new article that discusses component library authorship in Razor class libraries (RCLs) with static server-side rendering (static SSR).

Blazor Mode

- Blazor is a free and open-source web framework that enables developers to create web apps using C# and HTML.
- It is being developed by Microsoft and is part of the .NET ecosystem.
- Blazor has two hosting models: Blazor Server and Blazor WebAssembly.
- Blazor Server offers a way to build web UIs using C# instead of JavaScript.
- Blazor Server apps are hosted on the server and use real-time communication via SignalR to handle UI updates.
- Blazor WebAssembly, on the other hand, takes advantage of WebAssembly to do everything in the browser.

Blazor WebAssembly (WASM) or Blazor Server

- Choosing between Blazor WebAssembly (WASM) and Blazor Server depends on the specific requirements of your application and the trade-offs you are willing to make.
- **Choose Blazor WebAssembly if:**
  - You need offline capabilities.
  - You want to reduce server load.
  - You are building a client-centric application where performance is critical.
  - You prefer the app to be more independent of server connectivity.
- **Choose Blazor Server if:**
  - You want fast initial load times.
  - You need access to full .NET APIs without constraints.
  - Your app requires real-time updates and maintains server-side state.
  - You prefer a simpler deployment model with server-side control.

Blazor WebAssembly (WASM)

- **Pros:**
  - **Client-Side Execution**: The application runs in the browser on WebAssembly, which can result in faster UI responsiveness since interactions don't require round-trips to the server.
  - **Offline Capabilities**: Once downloaded, the application can continue to function offline.
  - **Scalability**: Since the execution is client-side, server load is reduced, potentially leading to better scalability for applications with many users.
  - **No Server Dependency**: Ideal for scenarios where you want the application to run independently of a server after the initial load.
- **Cons:**
  - **Initial Load Time**: The initial download can be large, leading to longer startup times.
  - **Limited Browser Support**: Not all features are available in all browsers.
  - **Resource Constraints**: The browser's memory and processing limitations can restrict what you can do.

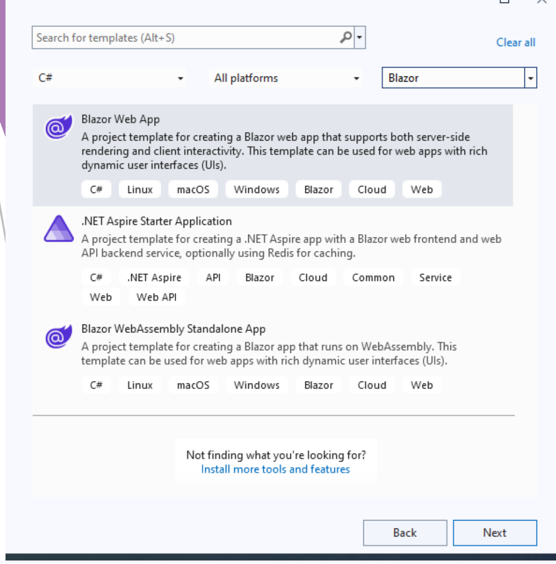
Blazor Server

- **Pros:**
  - **Instant Load**: Since the application is running on the server, the initial load time is usually faster.
  - **Full .NET API Support**: You have full access to .NET APIs without limitations.
  - **Smaller Payloads**: Only the necessary UI changes are sent to the client, which can result in less bandwidth usage for each interaction.
  - **Simpler Debugging**: Debugging is easier since the code runs on the server, and you have access to all server-side resources.
- **Cons:**
  - **Latency**: Every UI interaction requires a round-trip to the server, which can introduce latency.
  - **Server Load**: The server needs to maintain a constant connection to each client, which can become a scalability bottleneck.
  - **No Offline Support**: The app does not work without a constant server connection.

Example: Blazor Web App .NET 8

- Blazor is a .NET frontend web framework that supports both server-side rendering and client interactivity in a single programming model.
- Blazor is a framework for
  - Building interactive client-side web UI with .NET
  - Create rich interactive UIs using C#
  - Share server-side and client-side app logic written in .NET.

- Open VS 2022
- Create a new project
- Choose the template: Blazor Web App
- Give it a name
- Choose Auto (Server and WebAssembly) from Interactive Render Mode
- Choose Per page/component for Interactivity location
- Include Sample pages
- Create

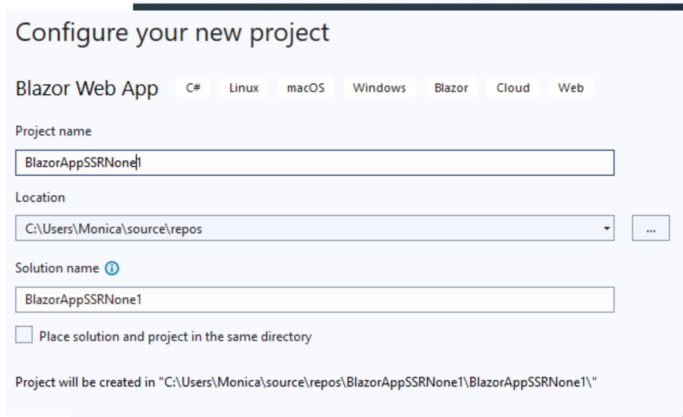


Blazor Web App
A project template for creating a Blazor web app that supports both server-side rendering and client interactivity. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

.NET Aspire Starter Application
A project template for creating a .NET Aspire app with a Blazor web frontend and web API backend service, optionally using Redis for caching.
C# .NET Aspire API Blazor Cloud Common Service Web Web API

Blazor WebAssembly Standalone App
A project template for creating a Blazor app that runs on WebAssembly. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

Not finding what you're looking for?
Install more tools and features


Configure your new project

Blazor Web App C# Linux macOS Windows Blazor Cloud Web

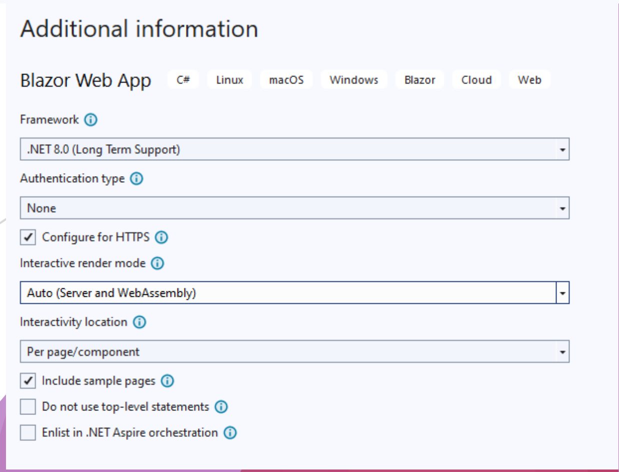
Project name: BlazorAppSSRNone1

Location: C:\Users\Monica\source\repos

Solution name: BlazorAppSSRNone1

☐ Place solution and project in the same directory

Project will be created in "C:\Users\Monica\source\repos\BlazorAppSSRNone1\BlazorAppSSRNone1\"


Additional information

Blazor Web App C# Linux macOS Windows Blazor Cloud Web

Framework: .NET 8.0 (Long Term Support)

Authentication type: None

✓ Configure for HTTPS

Interactive render mode: Auto (Server and WebAssembly)

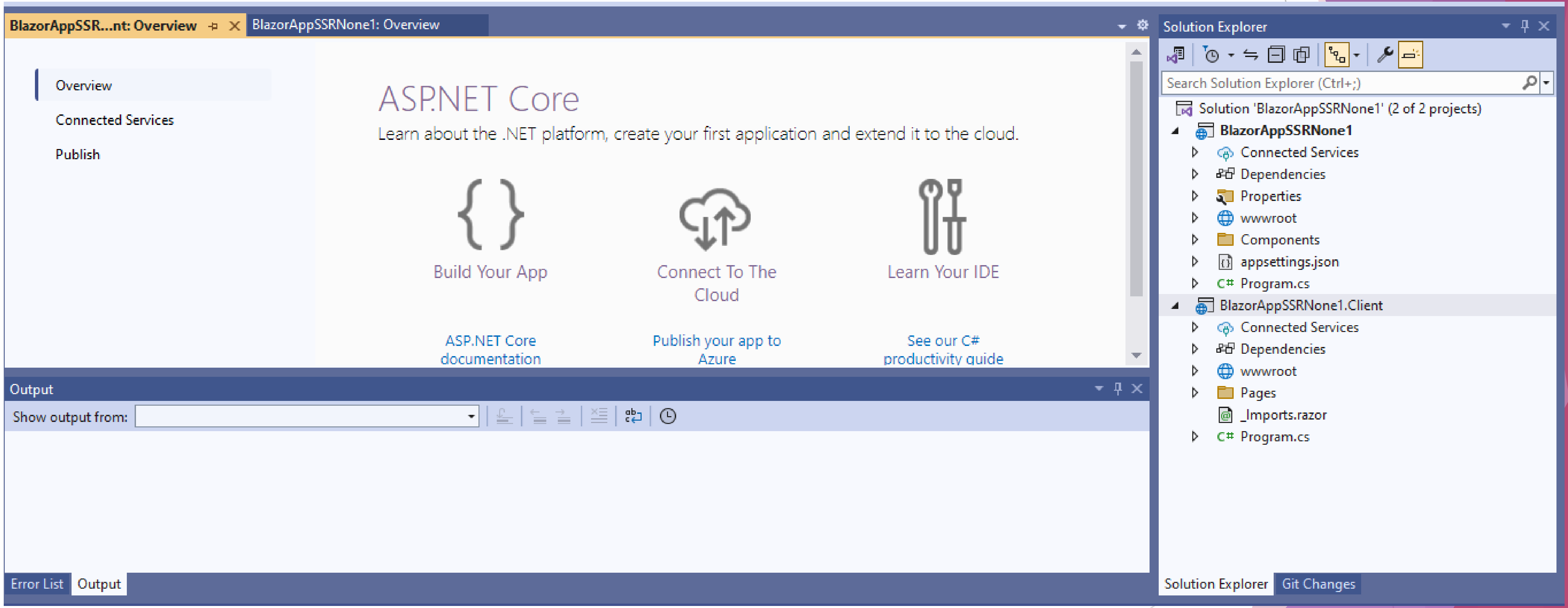
Interactivity location: Per page/component

✓ Include sample pages

☐ Do not use top-level statements

☐ Enlist in .NET Aspire orchestration

Get the new Application



Program.cs

- Check the Program.cs file. The Interactivity is added as Server first, then from Cache of WebAssembly. This is the mode of Auto.
- // Add services to the container.
- builder.Services.AddRazorComponents()
-     .AddInteractiveServerComponents()
-     .AddInteractiveWebAssemblyComponents();

Create a new Component

- Create a folder MyComp in Components folder
- Add a new Razor Component in the folder MyComp. Name it BtnComp.razor

- Edit Home.razor to add BtnComp component.
- We see that the output does not render the button click
- Now add @rendermode="InteractiveServer" to the BtnComp
- We can see that the button is interacting
- Next we can give the component page wise rendermode also by adding @rendermode="InteractiveServer" at BtnComp page rather than the Home page/ BtnComp component.

BtnComp.razor

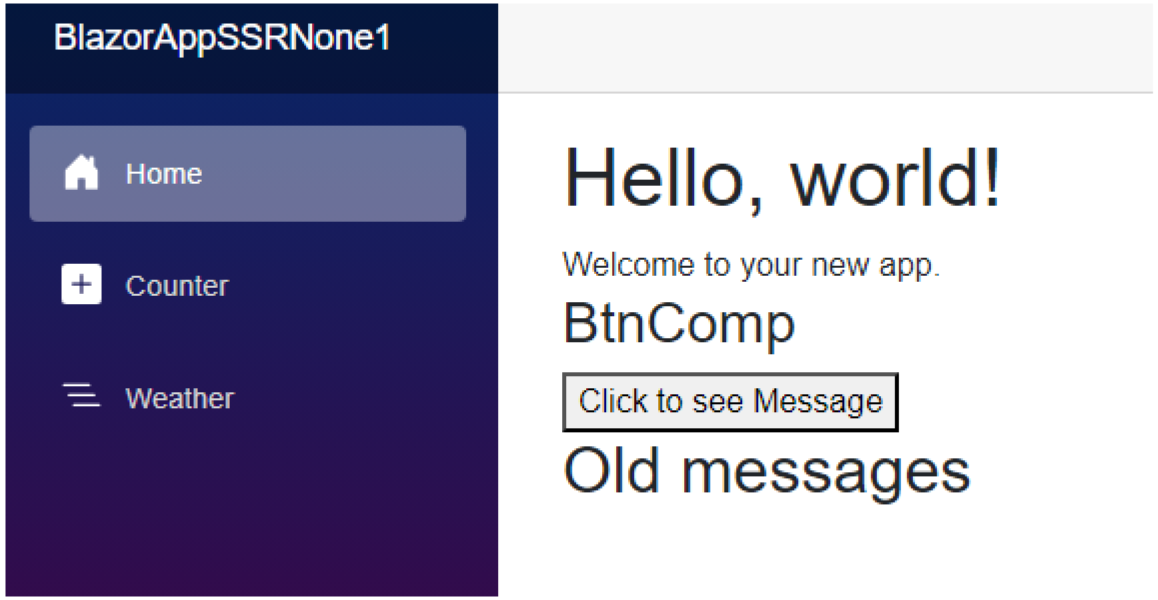
```
<h3>BtnComp</h3>

<button @onclick="changeMsg">Click to see Message</button>
<h2>@msg</h2>

@code {
    string msg="Old messages";
    void changeMsg(){
        msg = msg == "Old messages" ? "This is the new Message post click":"Old messages";
    }

}
```

Home.razor

- @page "/"
- <PageTitle>Home</PageTitle>
- <h1>Hello, world!</h1>
- Welcome to your new app.
- <BtnComp />

Output

- Note that the button click does not Interact with the user


BlazorAppSSRNone1

Hello, world!

Welcome to your new app.

BtnComp

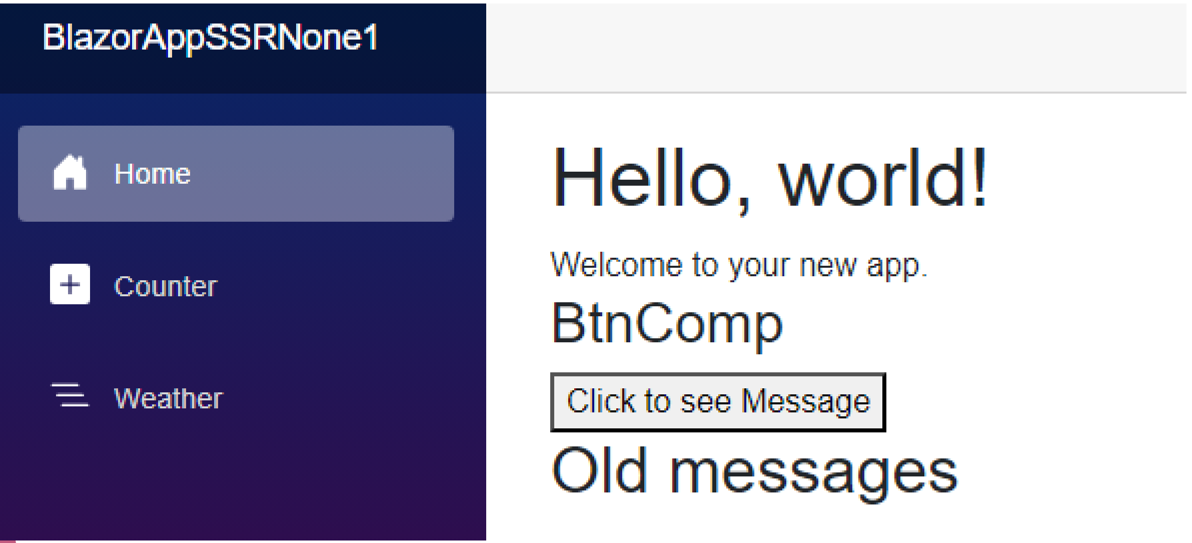
Old messages

Home.razor

- @page "/"
- <PageTitle>Home</PageTitle>
- <h1>Hello, world!</h1>
- Welcome to your new app.
- <BtnComp @rendermode="InteractiveServer"/>

Output

- Now the button is interactive



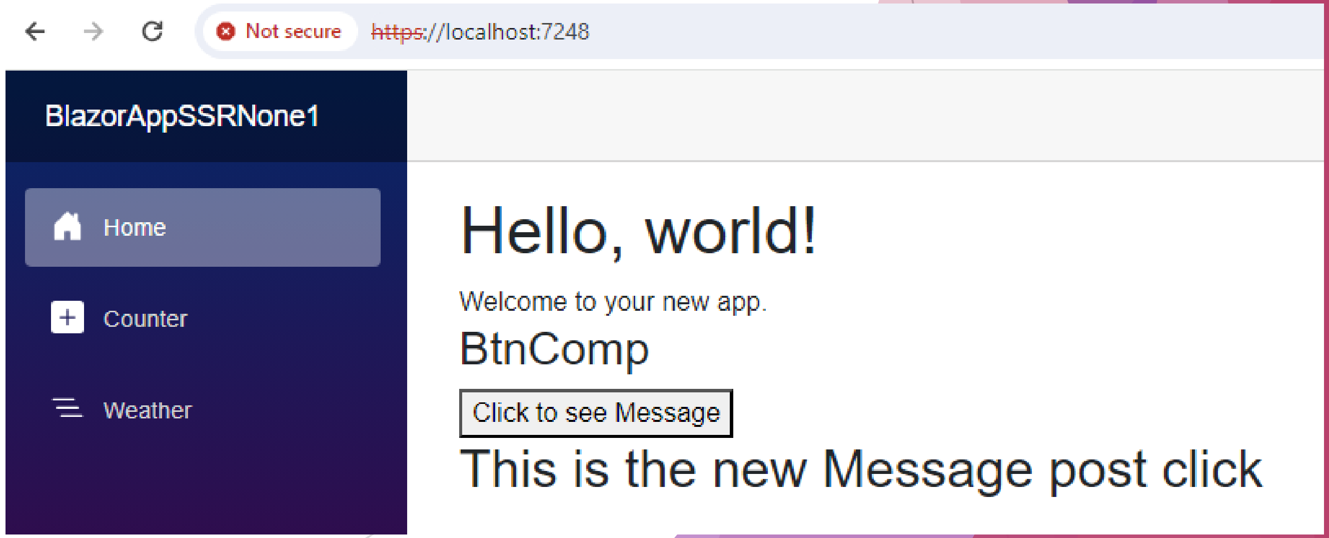
Hello, world!

Welcome to your new app.

BtnComp

Click to see Message

Old messages



Hello, world!

Welcome to your new app.

BtnComp

Click to see Message

This is the new Message post click

Home.razor

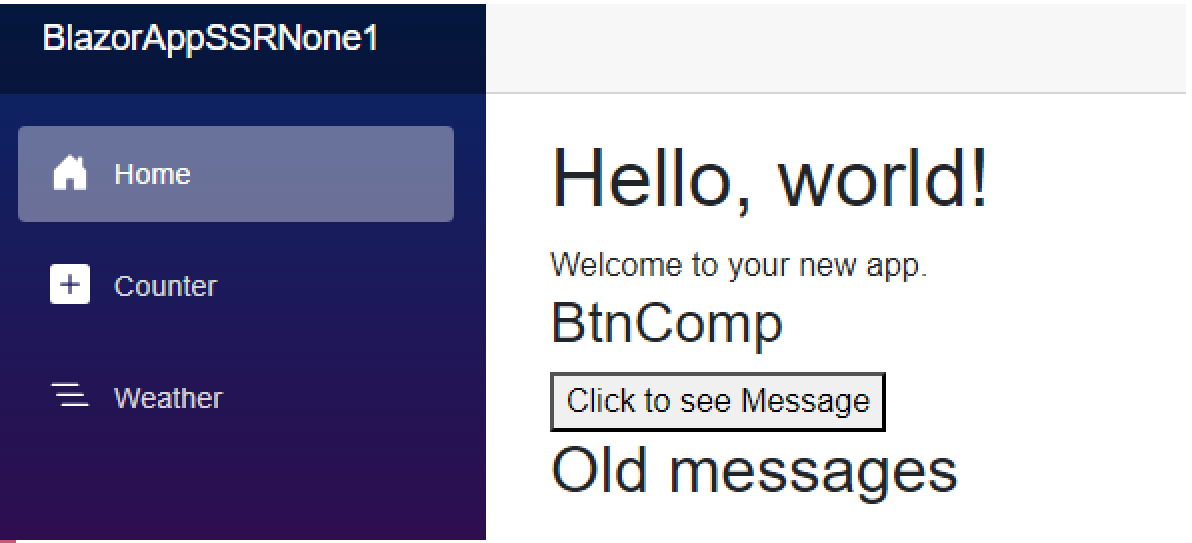
- @page "/"
- <PageTitle>Home</PageTitle>
- <h1>Hello, world!</h1>
- Welcome to your new app.
- <BtnComp />

BtnComp.razor

```
@rendermode InteractiveServer

<h3>BtnComp</h3>

<button @onclick="changeMsg">Click to see Message</button>
<h2>@msg</h2>

@code {
    string msg="Old messages";
    void changeMsg(){
        msg = msg == "Old messages" ? "This is the new Message post click":"Old messages";
    }

}
```

Output

- Now the button is interactive



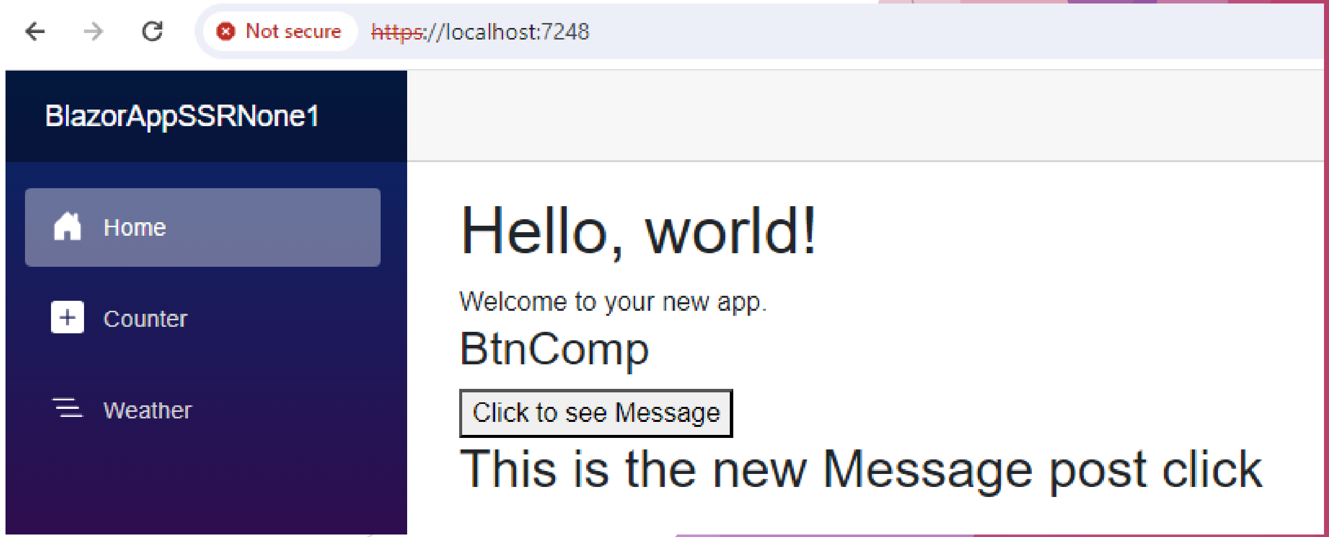
Hello, world!

Welcome to your new app.

BtnComp

Click to see Message

Old messages



Hello, world!

Welcome to your new app.

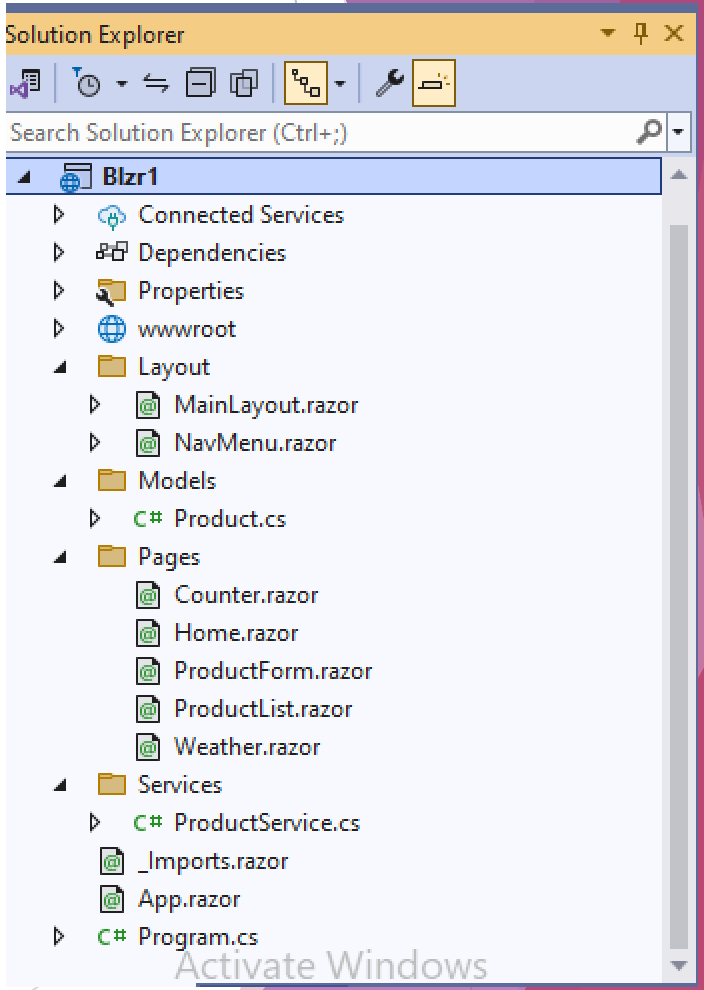
BtnComp

Click to see Message

This is the new Message post click

Example: CRUD in Blazor..in .Net8

- Create a Blazor App: Blazor WebAssembly StandAlone App
- Name it : Blzr1
- Once created,
- Create the folders Models, Services
- Create classes:
  - Models->Product.cs
  - Services->ProductService.cs
- Create Razor Component in Pages folder:
  - ProductForm.razor
  - ProductList.razor
- Edit _imports.razor
- Edit Program.cs
- Edit Layout->NavMenu.razor



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Blzr1
  Connected Services
  Dependencies
  Properties
  wwwroot
  Layout
    MainLayout.razor
    NavMenu.razor
  Models
    Product.cs
  Pages
    Counter.razor
    Home.razor
    ProductForm.razor
    ProductList.razor
    Weather.razor
  Services
    ProductService.cs
  _Imports.razor
  App.razor
  Program.cs
```

Activate Windows

Models->Product.cs

```
namespace Blzr1.Models
{
    public class Product
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public decimal Price { get; set; }
    }
}
```

Services-> ProductService.cs

```
using Blzr1.Models;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Blzr1.Services
{
    public class ProductService
    {
        private List<Product> _products = new List<Product>
        {
            new Product { Id = 1, Name = "Product 1", Price = 10.0M },
            new Product { Id = 2, Name = "Product 2", Price = 20.0M }
        };

        public Task<List<Product>> GetProductsAsync()
        {
            return Task.FromResult(_products);
        }

        public Task<Product> GetProductByIdAsync(int id)
        {
            return Task.FromResult(_products.FirstOrDefault(p => p.Id == id));
        }
```

Program.cs

- using Blzr1;
- using Microsoft.AspNetCore.Components.Web;
- using Microsoft.AspNetCore.Components.WebAssembly.Hosting;
- using Blzr1.Services;

- var builder = WebAssemblyHostBuilder.CreateDefault(args);
- builder.RootComponents.Add<App>("#app");
- builder.RootComponents.Add<HeadOutlet>("head::after");

- builder.Services.AddScoped(sp => new HttpClient { BaseAddress = new Uri(builder.HostEnvironment.BaseAddress) });

- builder.Services.AddScoped<ProductService>();

- await builder.Build().RunAsync();

_imports.razor

- @using System.Net.Http
- @using System.Net.Http.Json
- @using Microsoft.AspNetCore.Components.Forms
- @using Microsoft.AspNetCore.Components.Routing
- @using Microsoft.AspNetCore.Components.Web
- @using Microsoft.AspNetCore.Components.Web.Virtualization
- @using Microsoft.AspNetCore.Components.WebAssembly.Http
- @using Microsoft.JSInterop
- @using Blzr1
- @using Blzr1.Layout
- @using Blzr1.Services
- @using Blzr1.Models

Add in NavMenu.razor

```
<div class="nav-item px-3">
    <NavLink class="nav-link" href="products">
        <span class="oi oi-list-rich" aria-hidden="true"></span> Products
    </NavLink>
</div>
```

Layout-> NavMenu.razor

```
<div class="top-row ps-3 navbar navbar-dark">
    <div class="container-fluid">
        <a class="navbar-brand" href="">Blzr1</a>
        <button title="Navigation menu" class="navbar-toggler" @onclick="ToggleNavMenu">
            <span class="navbar-toggler-icon"></span>
        </button>
    </div>
</div>

<div class="@NavMenuCssClass nav-scrollable" @onclick="ToggleNavMenu">
    <nav class="flex-column">
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="" Match="NavLinkMatch.All">
                <span class="bi bi-house-door-fill-nav-menu" aria-hidden="true"></span> Home
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="counter">
                <span class="bi bi-plus-square-fill-nav-menu" aria-hidden="true"></span> Counter
            </NavLink>
        </div>
        <div class="nav-item px-3">
            <NavLink class="nav-link" href="weather">
                <span class="bi bi-list-nested-nav-menu" aria-hidden="true"></span> Weather
            </NavLink>
```

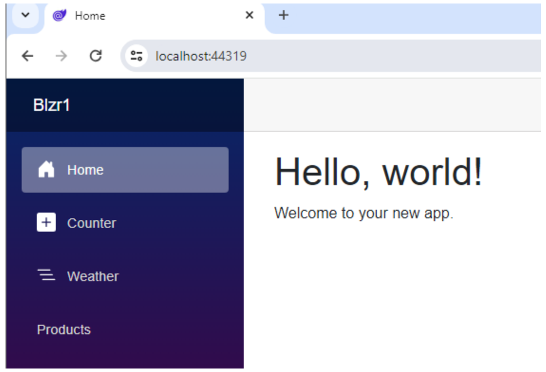
Pages-> ProductList.razor

```
@page "/products"
@inject ProductService ProductService
@inject NavigationManager NavigationManager

<h3>Product List</h3>

<table class="table">
    <thead>
        <tr>
            <th>Name</th>
            <th>Price</th>
            <th></th>
        </tr>
    </thead>
    <tbody>
        @foreach (var product in products)
        {
            <tr>
                <td>@product.Name</td>
                <td>@product.Price</td>
                <td>
                    <button class="btn btn-primary" @onclick="() => EditProduct(product.Id)">Edit</button>
                    <button class="btn btn-danger" @onclick="() => DeleteProduct(product.Id)">Delete</button>
                </td>
            </tr>
```

Pages-> ProductForm.razor

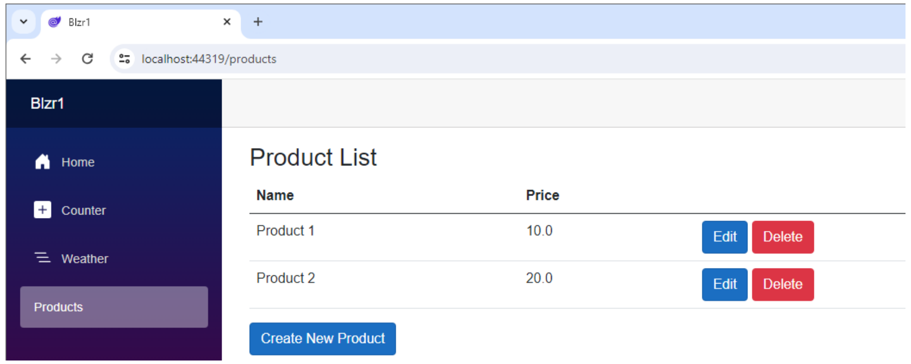
```
@page "/create-product"
@page "/edit-product/{id:int}"
@inject ProductService ProductService
@inject NavigationManager NavigationManager

<h3>@(IsEdit ? "Edit Product" : "Create Product")</h3>

<EditForm Model="product" OnValidSubmit="HandleValidSubmit">
    <DataAnnotationsValidator />
    <ValidationSummary />

    <div class="form-group">
        <label>Name</label>
        <InputText class="form-control" @bind-Value="product.Name" />
    </div>

    <div class="form-group">
        <label>Price</label>
        <InputNumber class="form-control" @bind-Value="product.Price" />
    </div>

    <button type="submit" class="btn btn-primary">@ButtonText</button>
</EditForm>

@code {
```

Output



Hello, world!

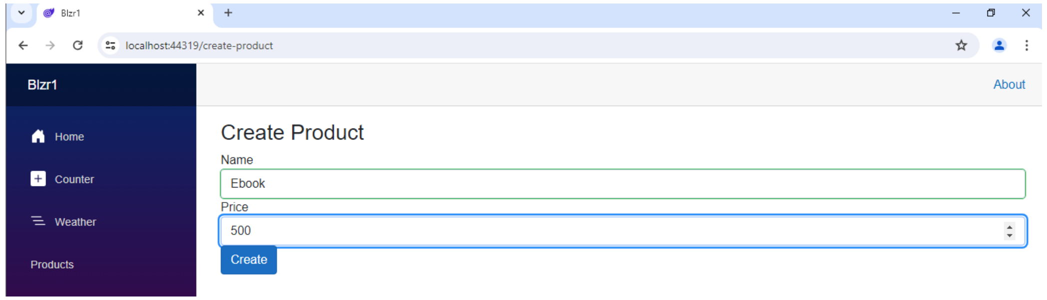
Welcome to your new app.



Product List

Name	Price	
Product 1	10.0	Edit Delete
Product 2	20.0	Edit Delete

Create New Product

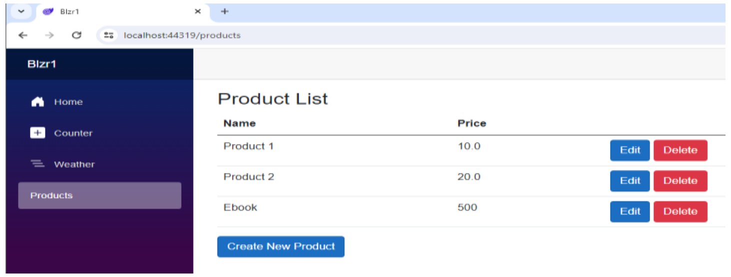


Create Product

Name: Ebook

Price: 500

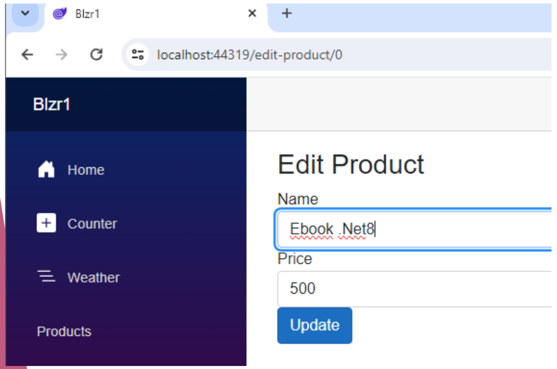
Create



Product List

Name	Price	
Product 1	10.0	Edit Delete
Product 2	20.0	Edit Delete
Ebook	500	Edit Delete

Create New Product

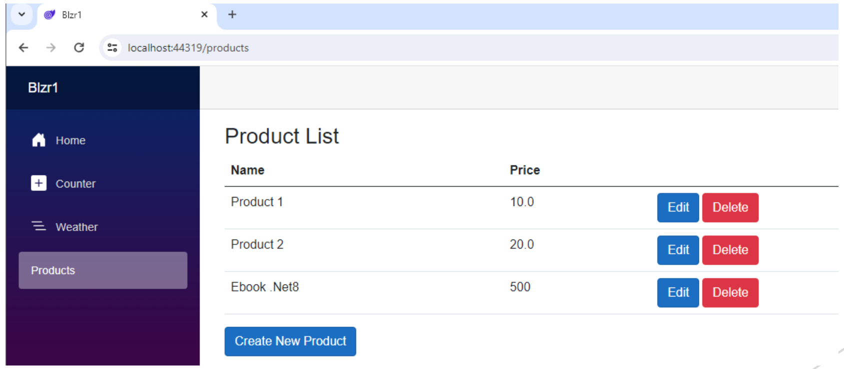


Edit Product

Name: Ebook .Net8

Price: 500

Update



Product List

Name	Price	
Product 1	10.0	Edit Delete
Product 2	20.0	Edit Delete
Ebook .Net8	500	Edit Delete

Create New Product

Entity Framework Core 8 Improvements and new features

.Net 8

Entity Framework Core 8

- Entity Framework Core 8 (EF Core 8) introduces several improvements and new features aimed at enhancing performance, flexibility, and ease of use.
- **Key Improvements and New Features in EF Core 8**
  - **Performance Improvements**: EF Core 8 includes various optimizations to improve the performance of queries and save operations.
  - **JSON Columns**: EF Core 8 supports JSON columns for storing and querying JSON data directly within relational databases.
  - **Bulk Updates and Deletes**: Bulk operations are more efficient, allowing for batch updates and deletes without loading entities into memory.
  - **Temporal Tables**: Enhanced support for temporal tables, allowing you to track historical data changes over time.
  - **Improved LINQ Translations**: Better translation of LINQ queries to SQL, including support for more complex expressions.
  - **New Interceptors**: More options for intercepting database operations, providing better control and customization.

EntityFramework : Step1: Create Database

- Open SSMS
- Get connected
- Create database training;
    - Use training;

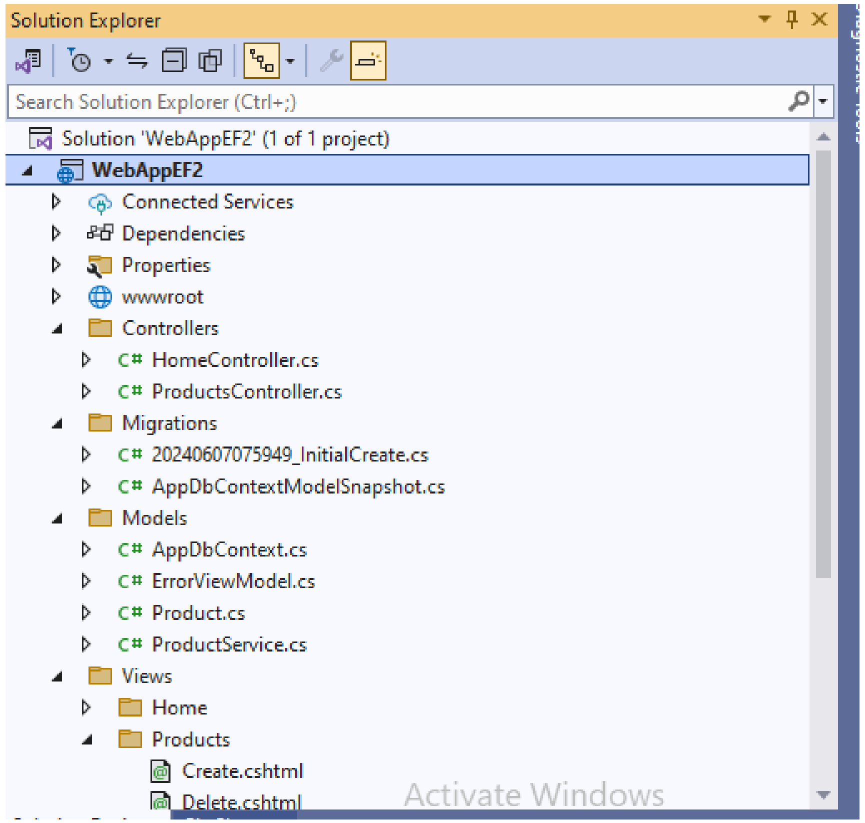
EntityFramework : Step2: Copy the data connection in a notepad

- Data Source = DESKTOP-62JHJ6O\SQLEXPRESS; Initial Catalog = training; Integrated Security = True
- Data Source=DESKTOP-ORPSK1N;Initial Catalog=training;Integrated Security=True
- Data Source=DESKTOP-3833V1F;Initial Catalog=training;Integrated Security=True
- DESKTOP-3833V1F

Step3: Create an ASP.Net Web Application(MVC) : Add : NuGet Packages

- Name the Project : WebApiEF1
- Install packages for Microsoft.EntityFrameworkCore :
  - Install-Package Microsoft.EntityFrameworkCore.SqlServer
  - Install-Package Microsoft.EntityFrameworkCore.Tools
  - Install-Package Swashbuckle.AspNetCore
  - Install-Package Microsoft.AspNetCore.Diagnostics.EntityFrameworkCore
- Edit Program.cs, appSettings.json, Views->Home->Index.cshtml
- Add classes: Models.Product.cs, Models. AppDbContext.cs, ProductService.cs
- Do Migration via tools->PackageManager Console in Nuget PM
- Controller -> Add New Scaffolded item: Controller for Entity Product and DbContext AppDbContext; Create Controller and Views
- Add a View for the EF8 Controller-> Action -> ListUsingEF8() -> Views-> Products -> ListUsingEF8.cshtml
- Edit Views->Products-> Index.cshtml for adding a href for ListUsingEF8 action.

Solution View Post Application Run

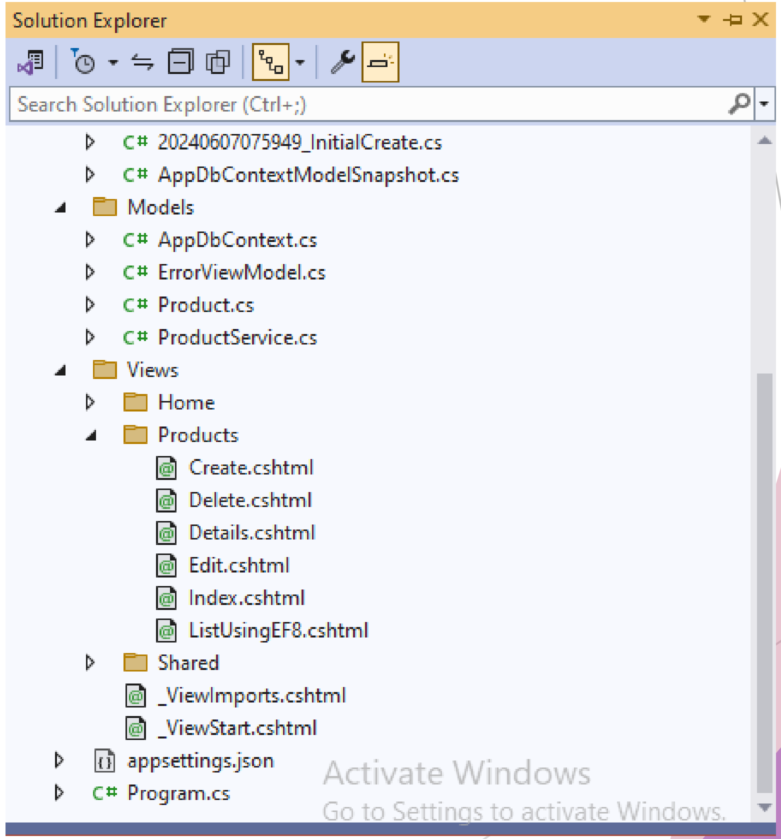


Solution Explorer

Search Solution Explorer (Ctrl+;)

```
Solution 'WebAppEF2' (1 of 1 project)
◢ WebAppEF2
    ▷ Connected Services
    ▷ Dependencies
    ▷ Properties
    ▷ wwwroot
    ◢ Controllers
        ▷ C# HomeController.cs
        ▷ C# ProductsController.cs
    ◢ Migrations
        ▷ C# 20240607075949_InitialCreate.cs
        ▷ C# AppDbContextModelSnapshot.cs
    ◢ Models
        ▷ C# AppDbContext.cs
        ▷ C# ErrorViewModel.cs
        ▷ C# Product.cs
        ▷ C# ProductService.cs
    ◢ Views
        ▷ Home
        ◢ Products
            Create.cshtml
            Delete.cshtml
```

Activate Windows



Solution Explorer

Search Solution Explorer (Ctrl+;)

```
        ▷ C# 20240607075949_InitialCreate.cs
        ▷ C# AppDbContextModelSnapshot.cs
    ◢ Models
        ▷ C# AppDbContext.cs
        ▷ C# ErrorViewModel.cs
        ▷ C# Product.cs
        ▷ C# ProductService.cs
    ◢ Views
        ▷ Home
        ◢ Products
            Create.cshtml
            Delete.cshtml
            Details.cshtml
            Edit.cshtml
            Index.cshtml
            ListUsingEF8.cshtml
        ▷ Shared
          _ViewImports.cshtml
          _ViewStart.cshtml
    ▷ appsettings.json
    ▷ C# Program.cs
```

Activate Windows
Go to Settings to activate Windows.

Models.Product.cs

```
using Microsoft.EntityFrameworkCore;
using System.ComponentModel.DataAnnotations;

namespace WebAppEF2.Models
{
    public class Product
    {
        [Key]
        public int Id { get; set; }
        public string Name { get; set; }
        public int Price { get; set; }
        public string JsonData { get; set; }
        public bool IsDeleted { get; set; } // For soft delete filter
    }
}
```

Program.cs

```
using Microsoft.EntityFrameworkCore;
using WebAppEF2.Models;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllersWithViews();
builder.Services.AddDbContext<AppDbContext>(options =>
            options.UseSqlServer(builder.Configuration.GetConnectionString("SqlServerConnection")));
builder.Services.AddDatabaseDeveloperPageExceptionFilter();
builder.Services.AddRazorPages();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (!app.Environment.IsDevelopment())  {
   app.UseExceptionHandler("/Home/Error");
   // The default HSTS value is 30 days. You may want to change this for production scenarios, see https://aka.ms/aspnetcore-hsts.
   app.UseHsts();
}
app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();
app.UseAuthorization();
app.MapControllerRoute(
   name: "default",
   pattern: "{controller=Home}/{action=Index}/{id?}");
app.Run();
```

appSettings.json

```
{
  "ConnectionStrings": {
    "SqlServerConnection": "Data Source=DESKTOP-3833V1F;Initial Catalog=training;Integrated Security=True;TrustServerCertificate=True"
  },

  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*"
}
```

Models. AppDbContext.cs

```
using Microsoft.EntityFrameworkCore;
namespace WebAppEF2.Models {
    public class AppDbContext : DbContext     {
        public DbSet<Product> Products { get; set; }
        public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
        //protected override void OnConfiguring(DbContextOptionsBuilder optionsBuilder)
        //{
        //    optionsBuilder.UseSqlServer("Data Source=DESKTOP-3833V1F;Initial Catalog=training;Integrated Security=True;TrustServerCertificate=True");
        //}
        protected override void OnModelCreating(ModelBuilder modelBuilder)    {
            modelBuilder.Entity<Product>()
                .HasQueryFilter(p => !p.IsDeleted);
            //modelBuilder.Entity<Product>()
            //    .Property(p => p.JsonData)
            //    .HasColumnType("json");
        }
    }
}
```

Models. ProductService.cs : for EF8 Services

```
using Microsoft.EntityFrameworkCore;
namespace WebAppEF2.Models {
    public class ProductService    {
        private readonly AppDbContext _context;
        public ProductService(AppDbContext context) { _context = context; }
        // Bulk Updates and Deletes.    // EF Core 8 provides more efficient bulk operations.Here’s an example of how to perform bulk
updates:
        public async Task BulkUpdatePricesAsync()      {
            var products = _context.Products.Where(p => p.Price < 10);
            await products.ExecuteUpdateAsync(p => p.SetProperty(p => p.Price, p => p.Price + 1));
        }
        public async Task BulkDeleteProductsAsync()      {   // And for bulk deletes
            var products = _context.Products.Where(p => p.Price < 10);
            await products.ExecuteDeleteAsync();
        }
        // Improved LINQ Translations. // EF Core 8 improves the translation of complex LINQ queries to SQL, making it easier to write
and maintain complex queries.
        public async Task<List<Product>> GetProductsAsync()      {
            var products = await _context.Products
                .Where(p => p.Name.Contains("Ebook")).OrderBy(p => p.Price).ToListAsync();
            return products;
        }
    }
}
```

Views->Home->Index.cshtml

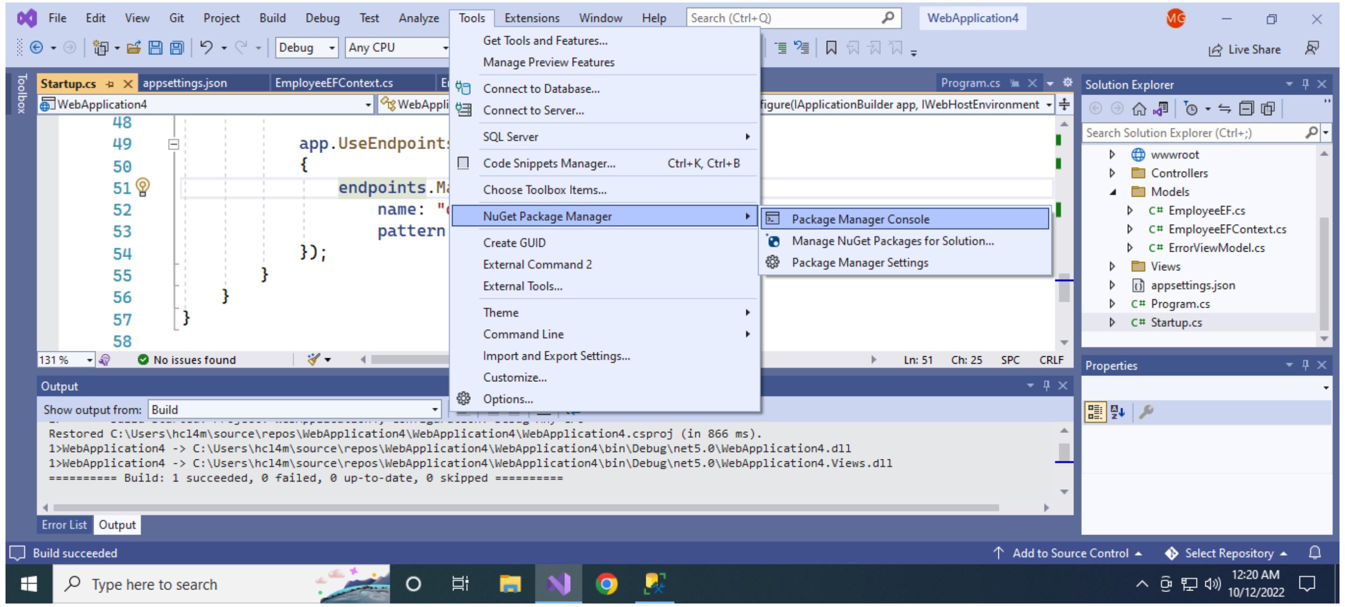
```
@{
    ViewData["Title"] = "Home Page";
}

<div class="text-center">
    <h1 class="display-4">Welcome</h1>
    <p>Learn about <a href="https://learn.microsoft.com/aspnet/core">building Web apps with ASP.NET Core</a>.</p>
<p>click to go to <a href="/Products">Products</a> Index page</p>
</div>
```

Build solution

- Build successfully

Goto Package Manager console to Migrate the table

- Tools-> NuGet Package Manager->Package Manager Console



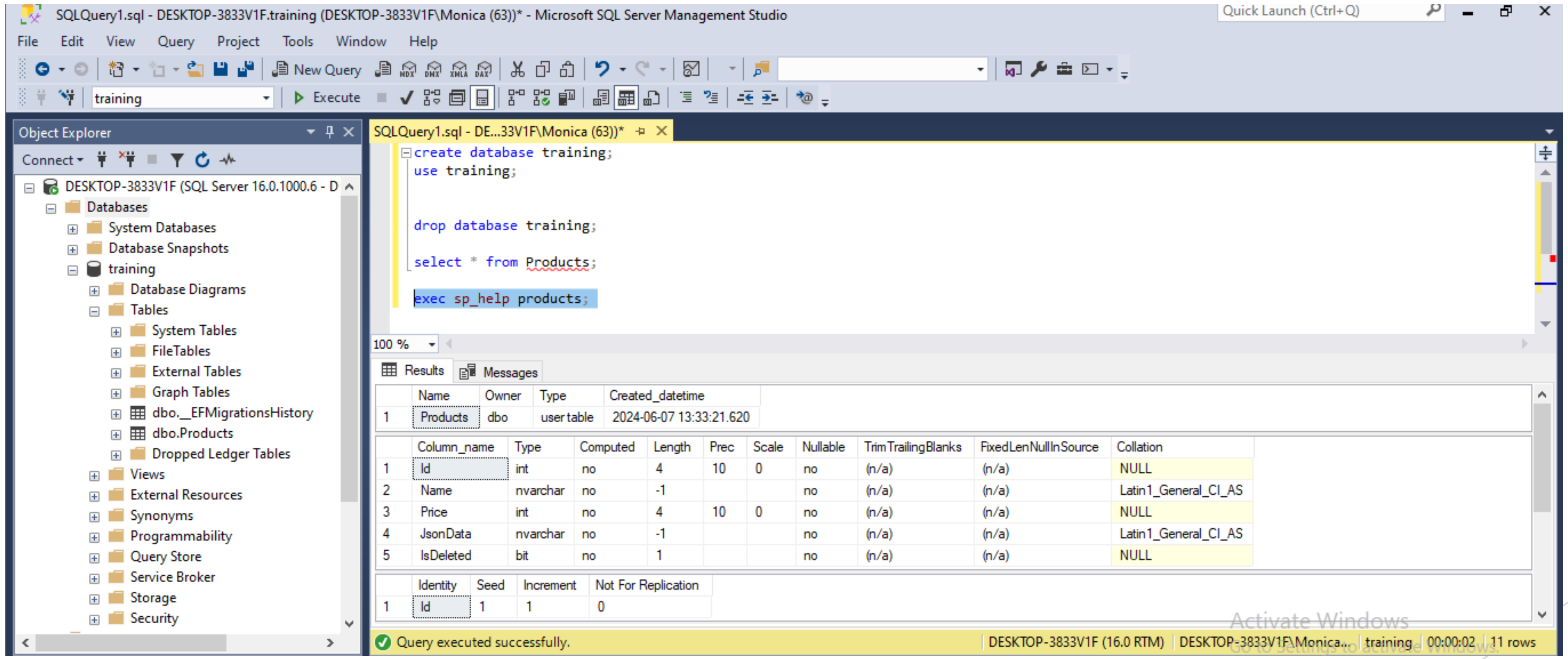
```
app.UseEndpoint
{
    endpoints.M
        name: "
        pattern
});
    }
  }
}
```

```
Restored C:\Users\hcl4m\source\repos\WebApplication4\WebApplication4\WebApplication4.csproj (in 866 ms).
1>WebApplication4 -> C:\Users\hcl4m\source\repos\WebApplication4\WebApplication4\bin\Debug\net5.0\WebApplication4.dll
1>WebApplication4 -> C:\Users\hcl4m\source\repos\WebApplication4\WebApplication4\bin\Debug\net5.0\WebApplication4.Views.dll
========== Build: 1 succeeded, 0 failed, 0 up-to-date, 0 skipped ==========
```

In the Console of Package Manager:

- PM> Add-Migration InitialCreate
  - Build started...
  - Build succeeded.
  - To undo this action, use Remove-Migration.
- PM> Update-Database
  - Build started...
  - Build succeeded.
  - Done.
- PM>

The table is created
select * from products;
exec sp_help products;



```sql
create database training;
use training;


drop database training;

select * from Products;

exec sp_help products;
```

	Name	Owner	Type	Created_datetime
1	Products	dbo	user table	2024-06-07 13:33:21.620

	Column_name	Type	Computed	Length	Prec	Scale	Nullable	TrimTrailingBlanks	FixedLenNullInSource	Collation
1	Id	int	no	4	10	0	no	(n/a)	(n/a)	NULL
2	Name	nvarchar	no	-1			no	(n/a)	(n/a)	Latin1_General_CI_AS
3	Price	int	no	4	10	0	no	(n/a)	(n/a)	NULL
4	JsonData	nvarchar	no	-1			no	(n/a)	(n/a)	Latin1_General_CI_AS
5	IsDeleted	bit	no	1			no	(n/a)	(n/a)	NULL

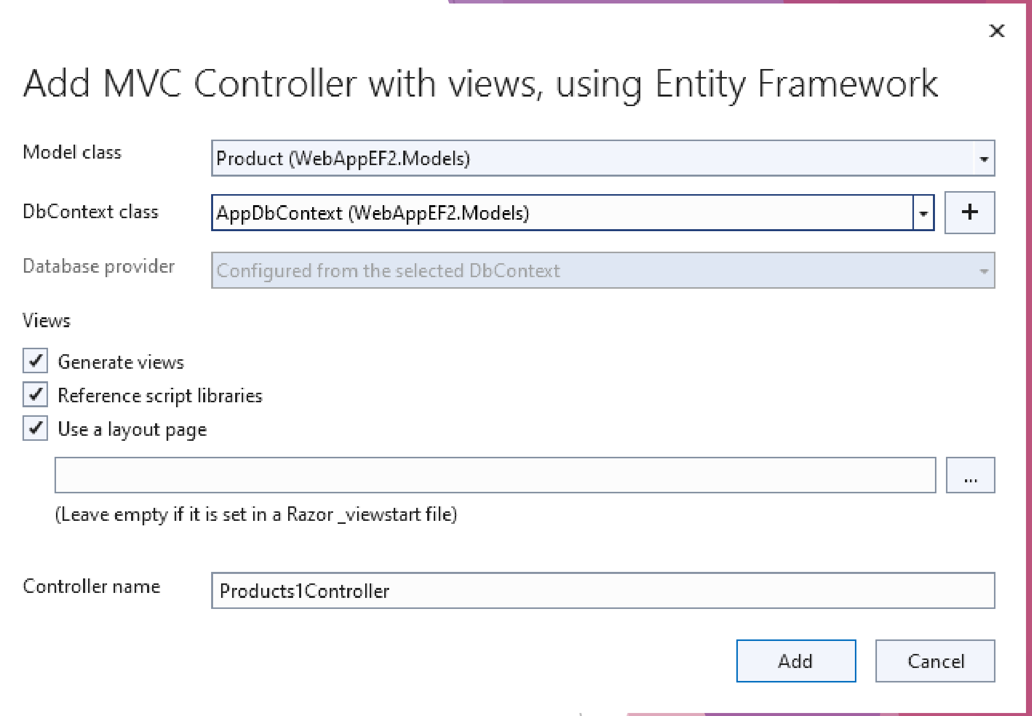
	Identity	Seed	Increment	Not For Replication
1	Id	1	1	0

Query executed successfully. | DESKTOP-3833V1F (16.0 RTM) | DESKTOP-3833V1F\Monica... | training | 00:00:02 | 11 rows

Add Controller+views

- Right-click project
- ->Add
- -> New Scaffolded items..
- ->MVC controller with views using Entity Framework
- ->Add
- Provide Model class: Product
- Data context class: AppDbContext
- ->Add

- All the views etc. are automatically added along
- With the Controller


Add MVC Controller with views, using Entity Framework

Model class: Product (WebAppEF2.Models)

DbContext class: AppDbContext (WebAppEF2.Models)

Database provider: Configured from the selected DbContext

Views

- ✓ Generate views
- ✓ Reference script libraries
- ✓ Use a layout page

(Leave empty if it is set in a Razor _viewstart file)

Controller name: Products1Controller

Add in Products Controller

```
public class ProductsController : Controller {
    private readonly AppDbContext _context;
    private ProductService _productService;
    public ProductsController(AppDbContext context)
    {
        _context = context;
        _productService = new ProductService(_context);
    }
    // GET: Products
    public async Task<IActionResult> Index()
    {
        return View(await _context.Products.ToListAsync());
    }
    // GET: Products/ListUsingEF8/
    public async Task<IActionResult> ListUsingEF8()
    {
        return View(await _productService.GetProductsAsync());
    }
    // GET: Products/Details/5
…
```

Add for EF8 Listing style:

```
// GET: Products/ListUsingEF8/
    public async Task<IActionResult> ListUsingEF8()
    {
        return View(await
_productService.GetProductsAsync());
    }
```

Add a view now for the above action.

View-> Product-> ListUsingEF8.cshtml

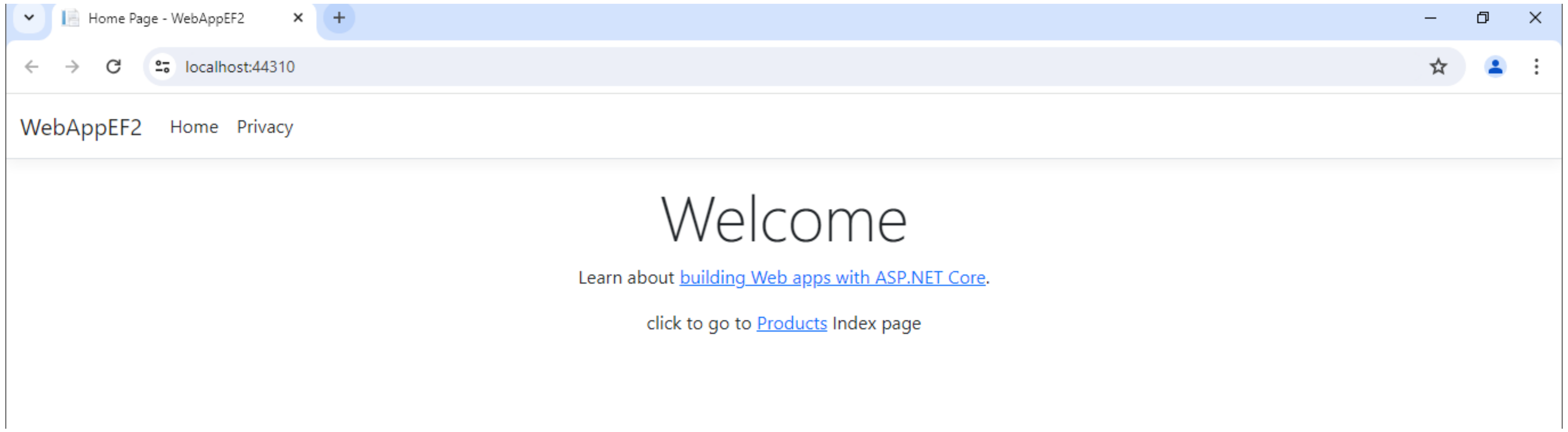
```
@model IEnumerable<WebAppEF2.Models.Product>

@{
    ViewData["Title"] = "Index";
}

<h1>List using EF 8</h1>

<p>
    <a href="/Products">Products/Index</a>
</p>

<p>
    <a asp-action="Create">Create New</a>
</p>
<table class="table">
    <thead>
        <tr>
            <th>
                @Html.DisplayNameFor(model => model.Name)
            </th>
            <th>
                @Html.DisplayNameFor(model => model.Price)
            </th>
            <th>
```

Add in : View-> Product-> Index.cshtml

```
@model IEnumerable<WebAppEF2.Models.Product>

@{
    ViewData["Title"] = "Index";
}

<h1>Index</h1>

<p>
    <a href="/Products/ListUsingEF8">Products/ListUsingEF8 (Only Ebooks)</a>
</p>

<p>
    <a asp-action="Create">Create New</a>
</p>
```

Output

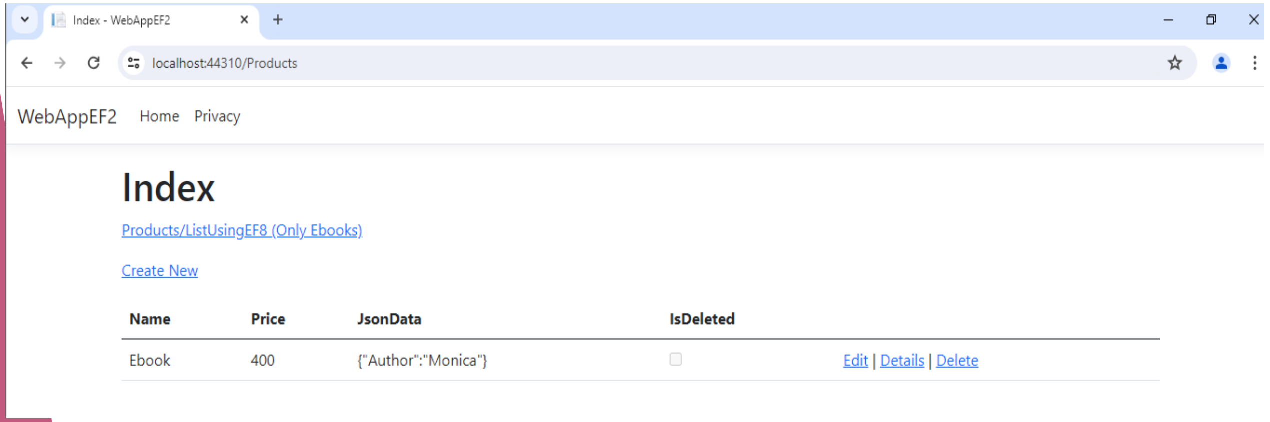
- Run the Project and click on Products link in the home page to get the output
- Check the work of all the functionalities.



Welcome

Learn about building Web apps with ASP.NET Core.

click to go to Products Index page

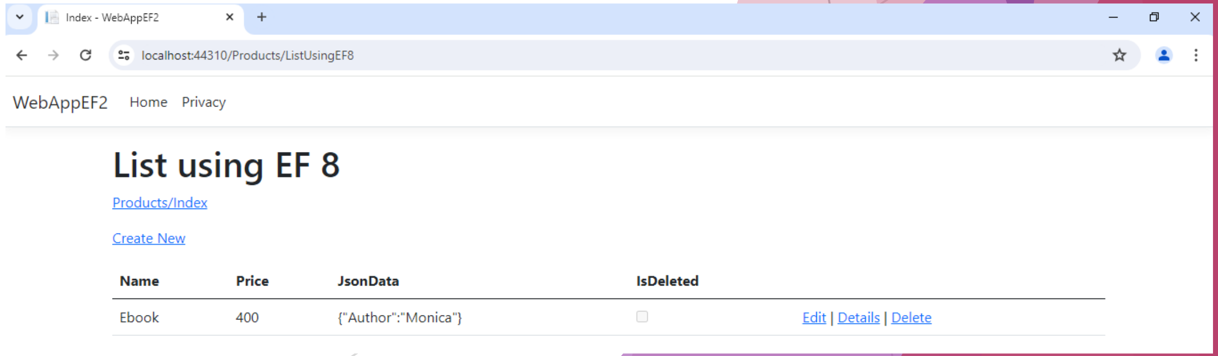


Index

Products/ListUsingEF8 (Only Ebooks)

Create New

Name	Price	JsonData	IsDeleted	
Ebook	400	{"Author":"Monica"}	☐	Edit | Details | Delete



List using EF 8

Products/Index

Create New

Name	Price	JsonData	IsDeleted	
Ebook	400	{"Author":"Monica"}	☐	Edit | Details | Delete

Dynamic Profile-Guided Optimization (PGO)

.Net 8

Dynamic Profile-Guided Optimization (PGO)

- Dynamic Profile-Guided Optimization (PGO) is an advanced optimization technique used by compilers to improve the performance of generated code.
- Dynamic Profile-Guided Optimization is a powerful tool for improving the performance of compiled code, especially in scenarios where runtime behavior can vary significantly.
- By using actual runtime profiling data, PGO enables compilers to make more informed decisions about optimization, leading to faster and more efficient software.
- In .NET 8, Dynamic Profile-Guided Optimization (PGO) is a new feature that aims to improve the performance of .NET applications by leveraging runtime profiling information.
- Dynamic Profile-Guided Optimization (PGO) in .NET 8 represents a significant enhancement to the .NET runtime and compilation pipeline. By leveraging runtime profiling information, .NET applications can achieve improved performance based on actual usage patterns, rather than assumptions made during static compilation. This can lead to faster and more efficient .NET applications across a variety of deployment scenarios.

How Dynamic Profile-Guided Optimization (PGO) Works in .NET 8

- **Compilation with Instrumentation:**
  - During compilation, the .NET 8 compiler (likely Roslyn) instruments the managed code to gather runtime profiling information. This instrumentation can collect data such as:
    - Frequency of method calls.
    - Hot and cold regions of code.
    - Data access patterns.
    - Type information.
- **Execution of Instrumented Code:**
  - The instrumented application is executed with representative inputs or workloads. This phase collects runtime data that reflects the actual behavior of the application when it is running in production or in a representative environment.
- **Profile Data Collection:**
  - As the application runs, the runtime collects profile data, which is stored in profile files. This data helps identify the most frequently executed code paths and data access patterns.
- **Recompilation with Optimization:**
  - Using the profile data collected in the previous step, the .NET 8 compiler recompiles the application with optimizations tailored to the observed runtime behavior. These optimizations can include:
    - Inlining of frequently called methods.
    - Optimizing branch predictions.
    - Data layout optimizations to improve cache locality.
- **Improved Performance:**
  - The resulting optimized code is expected to exhibit improved performance characteristics because it is tailored to the specific usage patterns of the application.

Benefits of Dynamic Profile-Guided Optimization

- **Performance:** PGO optimizes based on real-world usage patterns, leading to potentially significant performance improvements.
- **Efficiency:** Optimizations are specific to the application's actual runtime behavior, reducing unnecessary optimizations and improving efficiency.
- **Flexibility:** Can adapt to different usage scenarios and input data, making it suitable for a variety of .NET applications.
- **Integration:** Built into the .NET 8 compilation pipeline, making it easier to use without additional tooling.

Limitations of Dynamic Profile-Guided Optimization

- **Overhead:** The initial instrumentation and profiling can introduce some overhead, especially for larger applications.
- **Workload Dependency:** Optimizations are based on the workload used for profiling, which may not cover all edge cases.
- **Compilation Time:** PGO involves an additional compilation phase, which can increase build times.

Usage of Dynamic Profile-Guided Optimization

- **Common in Performance-Critical Applications:** Used extensively in high-performance computing, game development, and other areas where performance is crucial.
- **Optimization Threshold:** .NET 8 may use heuristics to determine whether the application is suitable for PGO and whether the overhead of profiling is justified.
- **Compatibility:** PGO is likely compatible with different types of .NET applications, including ASP.NET, desktop applications, and services.
- **Compiler Support:** .NET 8's compiler (Roslyn) will likely provide support for PGO as part of its standard optimizations.

Visual Studio 2022 and Profile-Guided Optimization (PGO)

- Visual Studio 2022 does support Profile-Guided Optimization (PGO) for native C++ applications, but it does not directly support PGO for managed .NET applications like C# or VB.NET. PGO in the context of Visual Studio typically applies to native code compiled with the Visual C++ compiler.
- For managed .NET applications, including C# and VB.NET, profile-guided optimizations are generally handled by the .NET runtime itself, rather than the Visual Studio IDE or compiler. This means that the .NET runtime (like .NET 8) and the JIT (Just-In-Time) compiler within it can use runtime profiling data to optimize the managed code at runtime.
- If you were to use Visual Studio 2022 for developing a managed .NET application (C# or VB.NET), you would typically profile the application using tools like the Performance Profiler included with Visual Studio. This profiler can help identify performance bottlenecks, but the optimizations based on that data would be handled by the .NET runtime's JIT compiler rather than by the Visual Studio compiler directly.
- To summarize, while Visual Studio 2022 supports PGO for native C++ applications, for managed .NET applications, PGO optimizations are handled by the .NET runtime's JIT compiler, with profiling data collected using tools provided by Visual Studio.

For Native C++ Applications (Visual C++) : PGO

- **Instrumentation and Collection:**
  - Visual Studio can instrument native C++ code to collect profiling data. This can include data such as function call frequencies, branch prediction data, and data access patterns.
- **Execution and Data Collection:**
  - After instrumentation, the application is executed with representative workloads or inputs. During this phase, the profiler collects runtime data based on the application's behavior.
- **Recompilation with PGO:**
  - Using the collected profiling data, Visual Studio can perform a recompilation of the native C++ code with optimizations tailored to the observed runtime behavior. This can include optimizations like inlining frequently called functions and optimizing branch predictions.

For Managed .NET Applications (C#, VB.NET)

- **Runtime and JIT Compilation:**
  - Managed .NET applications are typically compiled to IL (Intermediate Language) by the Visual Studio compiler (Roslyn), and then Just-In-Time (JIT) compiled by the .NET runtime at runtime.
- **PGO at Runtime:**
  - Profile-Guided Optimization for managed .NET applications is handled by the .NET runtime's JIT compiler. The JIT compiler can use runtime profiling data to optimize the IL code based on the actual usage patterns and behavior of the application.
- **Integration with .NET Runtime:**
  - Visual Studio 2022 provides tools for managing and debugging .NET applications, but the PGO optimizations themselves are part of the .NET runtime's capabilities rather than being directly integrated into the Visual Studio IDE.

.NET Aspire

.Net8

.NET Aspire

- .NET Aspire represents a modern approach to .NET application development, leveraging the latest features of .NET 8 and Visual Studio 2022.
- By adopting the .NET Aspire project type, developers can build robust, scalable, and maintainable applications more efficiently.
- Whether you are developing web APIs, frontend applications, or background services, .NET Aspire provides a comprehensive framework that supports the best practices and principles of modern software development.
- .NET Aspire is an advanced project template introduced in Visual Studio 2022, designed to provide a comprehensive and structured approach to modern .NET development.
- It encapsulates best practices and integrates various aspects of application development, making it easier for developers to build, maintain, and scale applications.
- This project type includes four distinct projects:
  - Aspire.WebApi
  - Aspire.WebApp
  - Aspire.Worker
  - Aspire.Shared.

Objectives of .NET Aspire

- The primary objectives of the .NET Aspire project type are:
- **Promote Clean Architecture**: By separating concerns across different projects, .NET Aspire encourages a clean architecture where each project has a distinct responsibility.
- **Facilitate Code Reusability**: The Aspire.Shared project allows for common code to be reused across the entire solution, promoting DRY (Don't Repeat Yourself) principles.
- **Streamline Development Workflow**: By providing a predefined structure, developers can focus more on building features rather than setting up and configuring projects from scratch.
- **Enhance Maintainability**: Clear separation of concerns and modular design make it easier to maintain and extend applications over time.
- **Support for Modern Development Practices**: The template supports modern development practices such as microservices, background processing, and web-based UIs.

Components of .NET Aspire: Aspire.WebApi

- The Aspire.WebApi project is the backend of the application. It is responsible for:
  - **Handling Client Requests**: Using controllers to process HTTP requests and return appropriate responses.
  - **Business Logic Implementation**: Services contain the core business logic and ensure that the application behaves as intended.
  - **Data Access**: Repositories abstract the data access layer, making it easier to interact with databases or other data sources.
  - **Data Transfer Objects (DTOs)**: DTOs are used to define how data is sent over the network, providing a layer of abstraction between the API and the underlying data model.
- **Purpose:**
  - The Aspire.WebApi project is designed to serve as the backend of your application. It typically contains RESTful APIs that handle client requests, perform business logic, and interact with databases or other services.
- **Key Features:**
  - **Controllers**: Handles incoming HTTP requests and maps them to appropriate actions.
  - **Services**: Contains business logic and interacts with data repositories.
  - **Repositories**: Manages data access, abstracting the data layer from the business logic.
  - **DTOs (Data Transfer Objects)**: Defines how data is sent and received by the API.

Components of .NET Aspire: Aspire.WebApp

- The Aspire.WebApp project is the frontend web application. It is designed to:
  - **Provide a User Interface**: Using Blazor, React, Angular, or another frontend framework to build a responsive and interactive user interface.
  - **Consume APIs**: Interact with the backend APIs to fetch and manipulate data.
  - **Handle Client-Side Logic**: Implement client-side logic and state management to ensure a smooth user experience.
  - **Routing and Navigation**: Manage routing and navigation within the application, providing a seamless user experience.
- **Purpose:**
  - The Aspire.WebApp project is typically a frontend web application that consumes the APIs provided by Aspire.WebApi. This could be a Single Page Application (SPA) using frameworks like React, Angular, or Blazor.
- **Key Features:**
  - **Components**: Reusable UI components for building the user interface.
  - **Pages**: Represents different views or pages in the application.
  - **Services**: Manages interactions with the backend APIs.
  - **State Management**: Manages the application state, ensuring a consistent user experience.

Components of .NET Aspire: Aspire.Worker

- The Aspire.Worker project is for background services and scheduled tasks. Its main responsibilities are:
  - **Running Background Jobs**: Execute long-running or periodic tasks in the background without blocking the main application.
  - **Processing Queues**: Handle messages from queues, enabling asynchronous processing of tasks.
  - **Scalability**: Improve application scalability by offloading heavy or time-consuming tasks to background workers.
  - **Reliability**: Ensure that critical background tasks are executed reliably, with appropriate error handling and retries.
- **Purpose:**
  - The Aspire.Worker project is designed to run background services or scheduled tasks. This could include processing messages from a queue, running periodic jobs, or handling long-running processes.
- **Key Features:**
  - **Worker Services:** Background services that run independently of user interactions.
  - **Hosted Services:** Services that start with the application and can run tasks in the background.
  - **Queue Processing**: Manages and processes messages from a message queue like Azure Service Bus or RabbitMQ.

Components of .NET Aspire: Aspire.Shared

- The Aspire.Shared project contains shared code and resources that are used across the solution. It typically includes:
  - **Common Models**: Data models that are used by both the API and frontend projects.
  - **Utilities**: Helper classes and methods that provide common functionality, reducing code duplication.
  - **Configurations and Constants**: Centralized configurations and constants that are used throughout the application, ensuring consistency and ease of maintenance.
- **Purpose:**
  - The Aspire.Shared project contains shared code and resources that are used by the other projects in the solution. This includes common models, utilities, and helper functions that promote code reuse and maintainability.
- **Key Features:**
  - **Models**: Shared data models used across different projects.
  - **Utilities**: Helper classes and methods for common functionality.
  - **Configurations**: Shared configuration settings and constants.

Design Principles of .NET Aspire

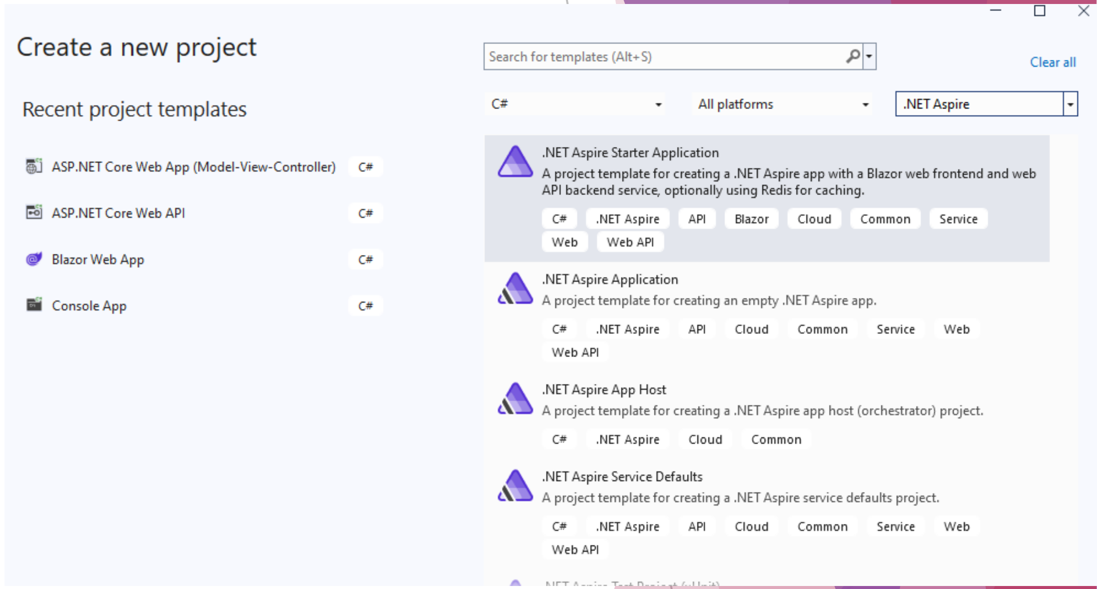
- **Clean Architecture**
  - Clean architecture is a design philosophy that emphasizes the separation of concerns, making applications easier to test, maintain, and scale. .NET Aspire adopts this philosophy by clearly dividing the solution into distinct projects, each with a specific responsibility.
- **Modularity**
  - Modularity refers to designing software in such a way that different parts of the system can be developed, tested, and deployed independently. The .NET Aspire project structure promotes modularity by separating the API, frontend, worker services, and shared components into distinct projects.
- **Reusability**
  - Reusability is the practice of writing code that can be used in multiple places within the application. The Aspire.Shared project embodies this principle by containing common models, utilities, and configurations that can be reused across the entire solution.
- **Scalability**
  - Scalability is the ability of an application to handle increased load without compromising performance. The Aspire.Worker project enhances scalability by offloading resource-intensive tasks to background workers, freeing up the main application to handle user requests more efficiently.

Benefits of Using .NET Aspire

- **Enhanced Developer Productivity**: By providing a structured template, .NET Aspire allows developers to focus on building features rather than setting up projects.
- **Improved Code Quality**: The separation of concerns and modular design promote better organization and higher-quality code.
- **Easier Maintenance**: The clear structure makes it easier to understand, maintain, and extend the application over time.
- **Faster Onboarding**: New developers can quickly get up to speed with the project structure and best practices embedded in the template.
- **Better Performance**: Background processing and efficient API handling contribute to better overall application performance.

Example: Aspire4App1

- Create a new Project of type:
  - .Net Aspire Starter Application
- It contains template for all the four types of .Net Aspire project
- Give it a name: Aspire4App1
- Take all Defaults
- Create


Create a new project

Search for templates (Alt+S) | Clear all

C# | All platforms | .NET Aspire

Recent project templates

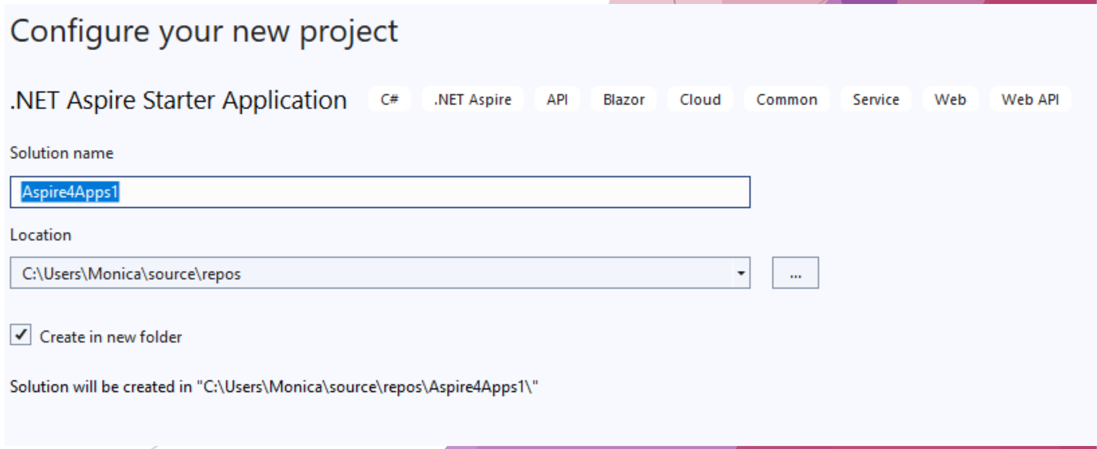
ASP.NET Core Web App (Model-View-Controller)	C#
ASP.NET Core Web API	C#
Blazor Web App	C#
Console App	C#

.NET Aspire Starter Application
A project template for creating a .NET Aspire app with a Blazor web frontend and web API backend service, optionally using Redis for caching.
C# .NET Aspire API Blazor Cloud Common Service Web Web API

.NET Aspire Application
A project template for creating an empty .NET Aspire app.
C# .NET Aspire API Cloud Common Service Web Web API

.NET Aspire App Host
A project template for creating a .NET Aspire app host (orchestrator) project.
C# .NET Aspire Cloud Common

.NET Aspire Service Defaults
A project template for creating a .NET Aspire service defaults project.
C# .NET Aspire API Cloud Common Service Web Web API


Configure your new project

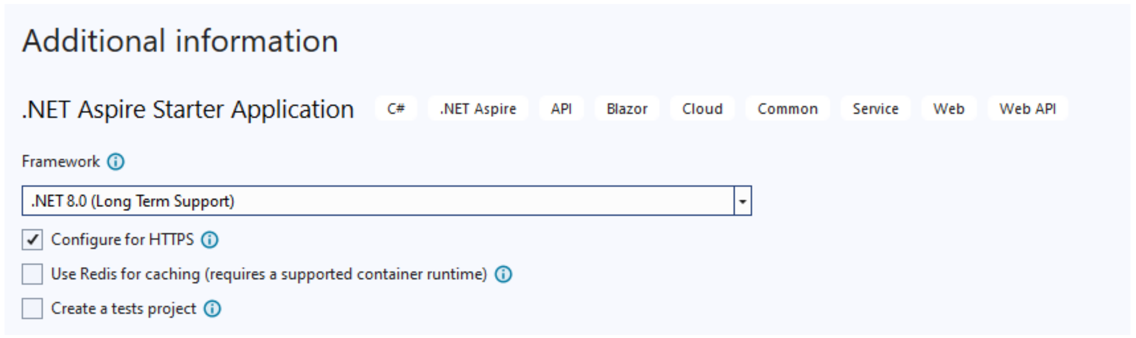
.NET Aspire Starter Application C# .NET Aspire API Blazor Cloud Common Service Web Web API

Solution name
Aspire4Apps1

Location
C:\Users\Monica\source\repos

✓ Create in new folder

Solution will be created in "C:\Users\Monica\source\repos\Aspire4Apps1\"


Additional information

.NET Aspire Starter Application C# .NET Aspire API Blazor Cloud Common Service Web Web API

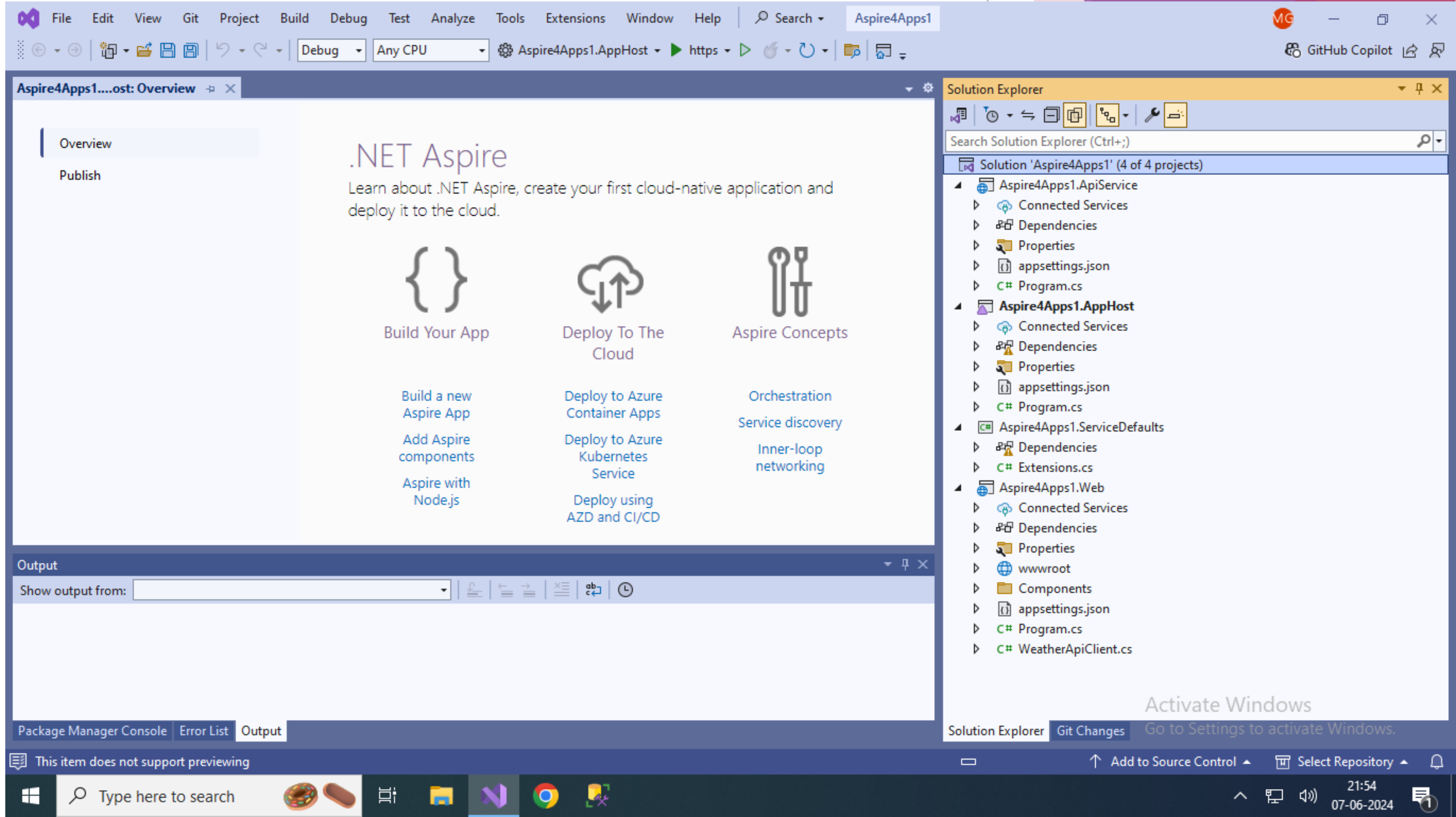
Framework
.NET 8.0 (Long Term Support)

- [✓] Configure for HTTPS
- [ ] Use Redis for caching (requires a supported container runtime)
- [ ] Create a tests project

The Project : Aspire4App1

- The Solution Contains 4 projects:
  - Aspire4App1.ApiServices
    - Manages API endpoints and business logic.
  - Aspire4App1.AppHost
    - Configures and runs the application.
  - Aspire4App1.ServiceDefaults
    - Provides shared logic and default implementations.
  - Aspire4App1.Web
    - Handles the front-end of the application.

The Project has a typical architecture for a web application with distinct layers for API services, application hosting, service defaults, and a web interface.



Breakdown of what each project represents:

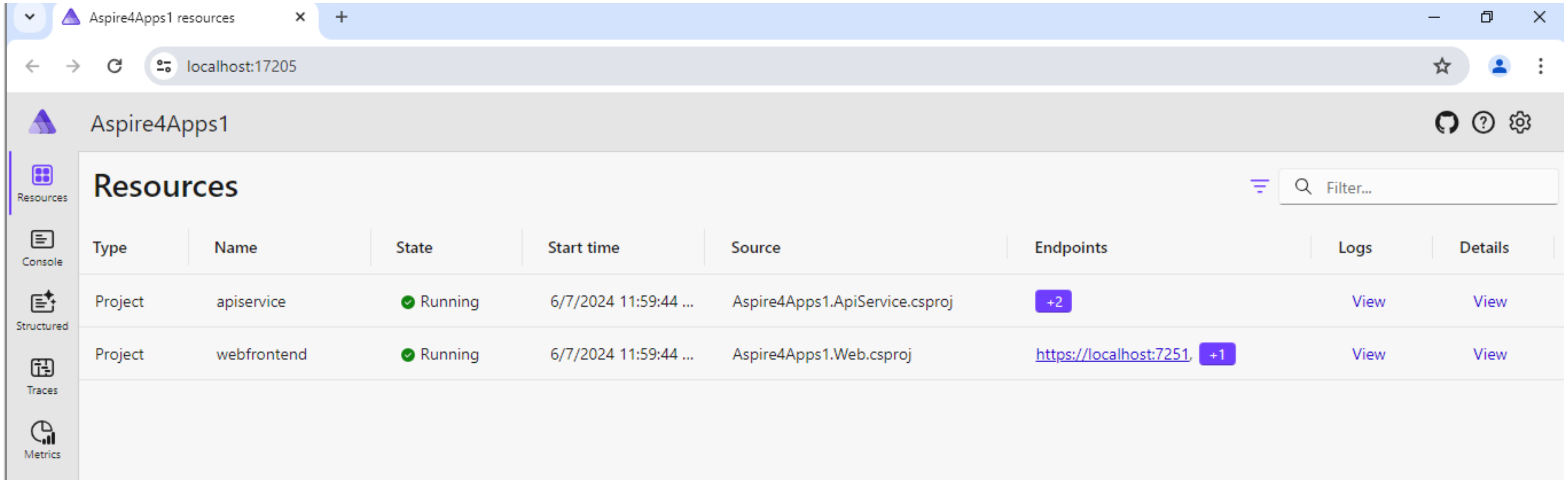
- **Aspire4App1.ApiServices**:
  - **Purpose**: This project is likely responsible for handling the API endpoints of your application. It will contain the business logic that your application exposes as RESTful or other types of APIs.
  - **Contents**: Controllers, models, service interfaces, and possibly business logic implementations.
- **Aspire4App1.AppHost**:
  - **Purpose**: This project acts as the entry point of your application. It configures and starts the application, setting up dependency injection, middleware, and other configurations necessary for the app to run.
  - **Contents**: `Program.cs`, `Startup.cs`, and other configuration files necessary for hosting the application.
- **Aspire4App1.ServiceDefaults**:
  - **Purpose**: This project probably contains default implementations and shared logic that are used across various services in your application. It might include common utilities, default configurations, or base classes that other services extend or use.
  - **Contents**: Utility classes, shared models, common configuration settings, and base service implementations.
- **Aspire4App1.Web**:
  - **Purpose**: This is likely the front-end part of your application, possibly an MVC web application or a single-page application (SPA) that interacts with the backend API services.
  - **Contents**: HTML, CSS, JavaScript, TypeScript files, views, controllers (if it's an MVC application), and front-end logic.

Project Structure

```
├────Aspire4Apps1.ApiService
│   ├────bin──Debug──net8.0
│   ├────obj──Debug──net8.0
│   │         ├────ref
│   │         ├────refint
│   │         └────staticwebassets
│   └────Properties
├────Aspire4Apps1.AppHost
│   ├────bin──Debug──net8.0
│   ├────obj──Debug──net8.0
│   │         ├────Aspire──references
│   │         ├────ref
│   │         └────refint
│   └────Properties
├────Aspire4Apps1.ServiceDefaults
│   ├────bin──Debug──net8.0
│   └────obj─Debug─net8.0
│            ├────ref
│            └────refint
└────Aspire4Apps1.Web
    ├────bin──Debug──net8.0
    ├────Components
    │   ├────Layout
    │   └────Pages
    ├────obj─Debug─net8.0
    │        ├────ref
    │        ├────refint
    │        └────staticwebassets
    ├────Properties
    └────wwwroot─bootstrap
```

Output

https://localhost:17205/

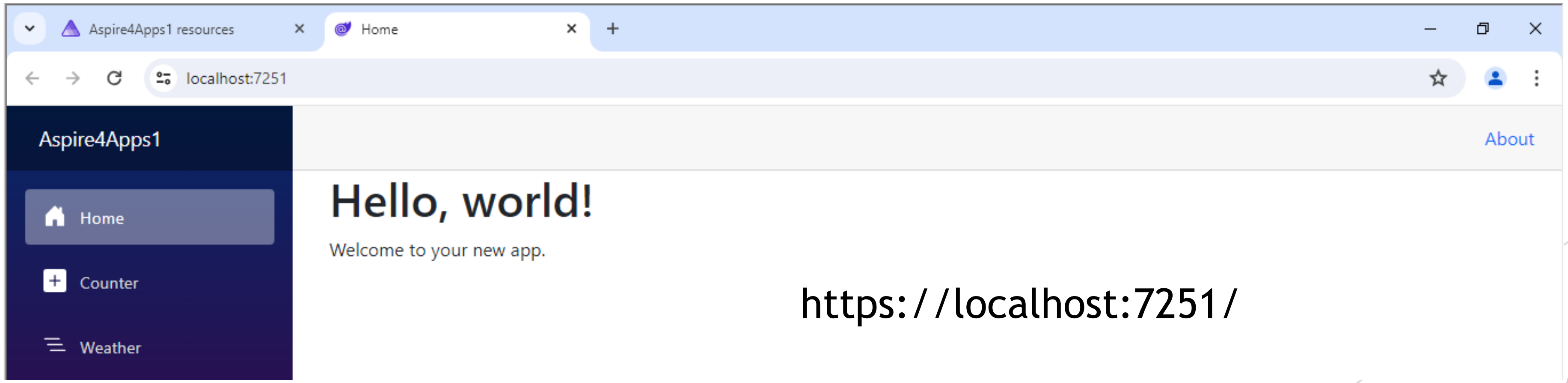

Aspire4Apps1 resources

localhost:17205

Aspire4Apps1

Resources

Type	Name	State	Start time	Source	Endpoints	Logs	Details
Project	apiservice	Running	6/7/2024 11:59:44 ...	Aspire4Apps1.ApiService.csproj	+2	View	View
Project	webfrontend	Running	6/7/2024 11:59:44 ...	Aspire4Apps1.Web.csproj	https://localhost:7251, +1	View	View


Aspire4Apps1 resources

Home

localhost:7251

Aspire4Apps1

About

Hello, world!

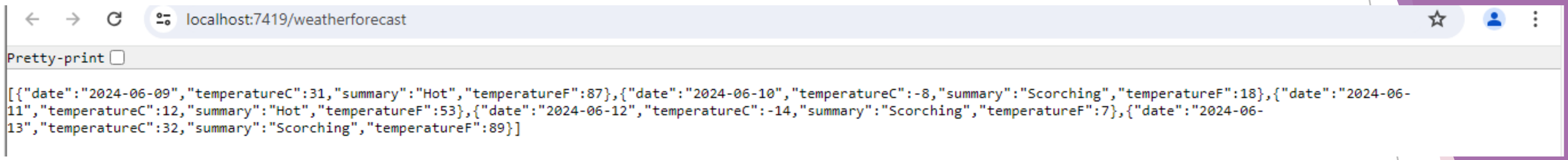
Welcome to your new app.

https://localhost:7251/

..Output

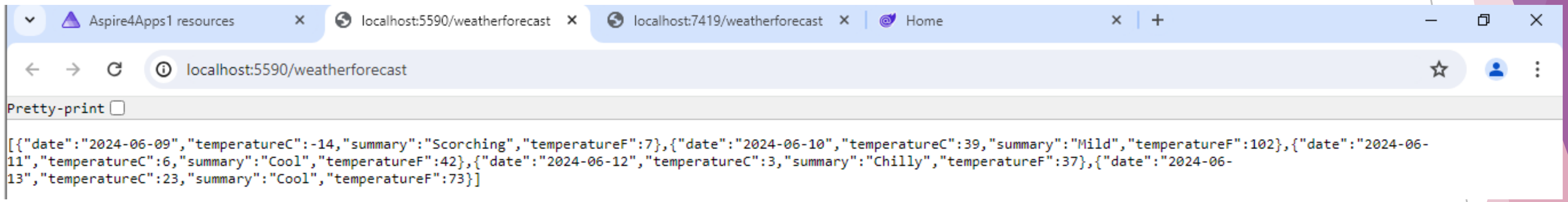
```
[
 {
  "date": "2024-06-09",
  "temperatureC": 31,
  "summary": "Hot",
  "temperatureF": 87
 },
 {
  "date": "2024-06-10",
  "temperatureC": -8,
  "summary": "Scorching",
  "temperatureF": 18
 },
 {
  "date": "2024-06-11",
  "temperatureC": 12,
  "summary": "Hot",
  "temperatureF": 53
 },
 {
  "date": "2024-06-12",
  "temperatureC": -14,
  "summary": "Scorching",
  "temperatureF": 7
 },
 {
  "date": "2024-06-13",
  "temperatureC": 32,
  "summary": "Scorching",
  "temperatureF": 89
 }
]
```

https://localhost:7419/weatherforecast



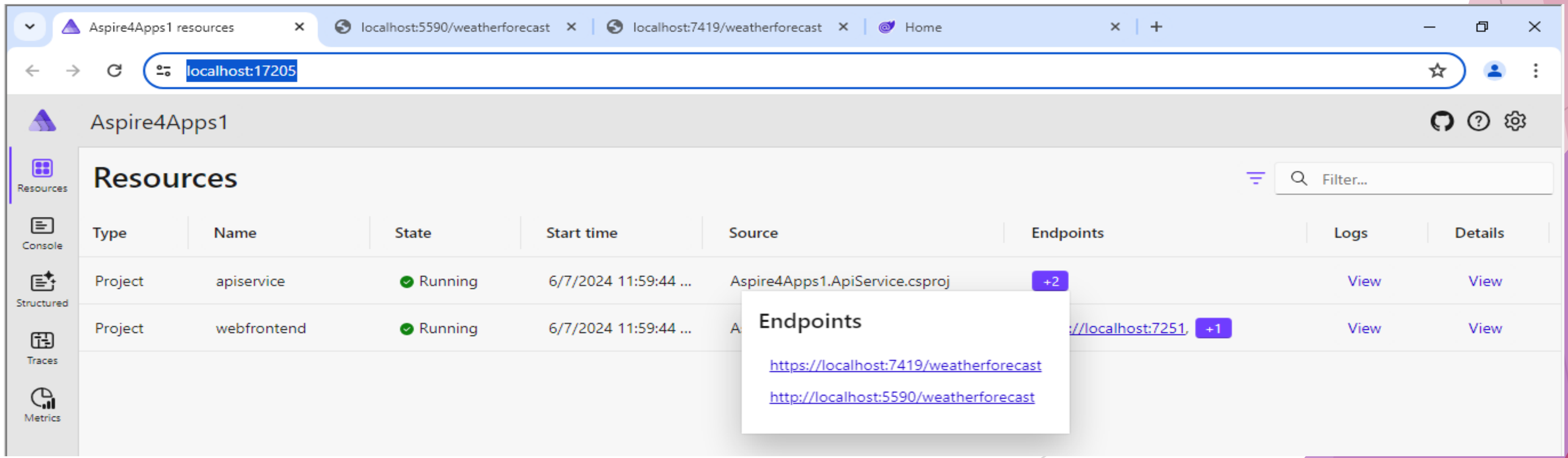
```
[{"date":"2024-06-09","temperatureC":31,"summary":"Hot","temperatureF":87},{"date":"2024-06-10","temperatureC":-8,"summary":"Scorching","temperatureF":18},{"date":"2024-06-
11","temperatureC":12,"summary":"Hot","temperatureF":53},{"date":"2024-06-12","temperatureC":-14,"summary":"Scorching","temperatureF":7},{"date":"2024-06-
13","temperatureC":32,"summary":"Scorching","temperatureF":89}]
```

http://localhost:5590/weatherforecast


Aspire4Apps1 resources
localhost:5590/weatherforecast
localhost:7419/weatherforecast
Home
localhost:5590/weatherforecast
Pretty-print

```
[{"date":"2024-06-09","temperatureC":-14,"summary":"Scorching","temperatureF":7},{"date":"2024-06-10","temperatureC":39,"summary":"Mild","temperatureF":102},{"date":"2024-06-
11","temperatureC":6,"summary":"Cool","temperatureF":42},{"date":"2024-06-12","temperatureC":3,"summary":"Chilly","temperatureF":37},{"date":"2024-06-
13","temperatureC":23,"summary":"Cool","temperatureF":73}]
```

https://localhost:17205/



Resources

Type	Name	State	Start time	Source	Endpoints	Logs	Details
Project	apiservice	Running	6/7/2024 11:59:44 ...	Aspire4Apps1.ApiService.csproj	+2	View	View
Project	webfrontend	Running	6/7/2024 11:59:44 ...	A	://localhost:7251, +1	View	View

Endpoints

https://localhost:7419/weatherforecast

http://localhost:5590/weatherforecast

AI features - Extended AI and ML Support

.Net 8

Extended AI and ML Support in .NET 8

- .NET 8, released in conjunction with Visual Studio 2022, includes several new and enhanced features that bolster AI and machine learning (ML) support.
  - **ML.NET Improvements**: .NET 8 includes enhancements to ML.NET, Microsoft's open-source machine learning framework for .NET. This includes better performance, new algorithms, and improved tooling.
  - **Integration with Azure ML**: Improved integration with Azure Machine Learning services allows developers to build, train, and deploy models more seamlessly within the Azure ecosystem.
  - **ONNX Runtime Integration**: Enhanced support for the Open Neural Network Exchange (ONNX) runtime, which allows developers to run pre-trained models from various frameworks like TensorFlow, PyTorch, and Scikit-learn within .NET applications.
  - **AI-Assisted Development**: Visual Studio 2022 includes AI-assisted development tools like IntelliCode, which provides intelligent code suggestions based on best practices and your coding style.
  - **Automated Machine Learning (AutoML)**: Enhancements in AutoML capabilities in ML.NET simplify the process of model selection and hyperparameter tuning.

ML Library

- Microsoft.ML the **ML .Net Library has:**
  - **MLContext**: The starting point for all ML.NET operations.
  - **IDataView**: Represents the data in ML.NET. Here, we load the sample housing data into an IDataView.
  - **Transforms and Trainers**: The pipeline defines the transformations (e.g., concatenating features) and the training algorithm (e.g., SDCA regression).
  - **Fit**: This method trains the model.
  - **Prediction Engine**: Used to make single predictions on new data.
- .NET 8 and Visual Studio 2022 significantly enhance AI and ML support, making it easier to integrate machine learning into .NET applications.
- With improved tools, better performance, and seamless integration with Azure, developers can build and deploy sophisticated AI models more efficiently.

Example

- The following example demonstrates a basic regression model using ML.NET, showcasing the simplicity and power of the new ML tools.

- Create a Console based project : ConAppML1
- Add NuGet Package:
  - Install-Package Microsoft.ML

- Add Classes:
  - HousingData.cs
  - HousingPricePrediction.cs
- Edit
  - Program.cs

HousingData.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConAppML1
{
    public class HousingData
    {
        public float Size { get; set; }
        public float Price { get; set; }
    }

}
```

HousingPricePrediction.cs

```
using Microsoft.ML.Data;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace ConAppML1
{
    public class HousingPricePrediction
    {
        [ColumnName("Score")]
        public float Price { get; set; }
    }
}
```

Program.cs

```
using ConAppML1;
using Microsoft.ML;
using Microsoft.ML.Data;
namespace ConAppML1{
    public class Program    {
        static void Main(string[] args)        {
            MLContext mlContext = new MLContext();
            // Load data
            var data = new List<HousingData>        {
            new HousingData { Size = 1.1F, Price = 1.2F },
            new HousingData { Size = 1.9F, Price = 2.3F },
            new HousingData { Size = 2.8F, Price = 3.0F }    };
            IDataView trainingData = mlContext.Data.LoadFromEnumerable(data);
            // Define data transformations and model training pipeline
            var pipeline = mlContext.Transforms.Concatenate("Features", nameof(HousingData.Size))
                .Append(mlContext.Regression.Trainers.Sdca(labelColumnName: nameof(HousingData.Price), maximumNumberOfIterations: 100));
            // Train the model
            var model = pipeline.Fit(trainingData);
            // Create prediction engine
            var predictionEngine = mlContext.Model.CreatePredictionEngine<HousingData, HousingPricePrediction>(model);            // Predict
            var sampleData = new HousingData { Size = 2.5F };
            var prediction = predictionEngine.Predict(sampleData);
            Console.WriteLine($"Predicted price for size {sampleData.Size} is {prediction.Price}");
        }
    }
}
```

```
Predicted price for size 2.5 is 2.7618234
```

Integrate AI into .NET 8: ONNX Runtime

- ONNX Runtime: Microsoft.ML.OnnxRuntime has the following features:
  - **ONNX Runtime Setup**: We initialize an InferenceSession with the path to the ONNX model.
  - **Load and Preprocess Image**: The LoadAndPreprocessImage method loads an image, resizes it to 224x224 pixels (common input size for models like MobileNet), and converts it to a tensor.
  - **Create Input Data**: We create the input data as a NamedOnnxValue list. The name "input" should match the input node name in the ONNX model.
  - **Run Inference**: We run the inference session with the input data and get the output.
  - **Post-process Results**: We find the label with the highest score and get the corresponding label name.
- By leveraging pre-trained models and the powerful ONNX runtime, developers can easily add sophisticated AI capabilities to their applications.
- This setup allows for running complex AI models efficiently within .NET applications, making it suitable for a variety of real-world scenarios.
- This example demonstrates how to integrate AI into a .NET 8 application using the ONNX Runtime.

Example :AI

- The following example demonstrates a basic regression model using AI and ML in .NET8, showcasing the simplicity and power of the new ML and AI tools.
- We'll create a simple sentiment analysis model using ML.NET. We'll train a model to classify text as positive or negative.

- Create a Console based project : ConAppAI1
- Add NuGet Package:
  - Install-Package Microsoft.ML
  - Install-Package Microsoft.ML.Data

- Edit
  - Program.cs

SentimentData.cs

```
using System;
namespace ConAppAI1 {
    public class SentimentData  {
        public bool Label { get; set; }
        public string SentimentText { get; set; }
    }
    public class SentimentPrediction : SentimentData  {
        public bool Prediction { get; set; }
        public float Probability { get; set; }
        public float Score { get; set; }
        public static List<SentimentData> mylist = new List<SentimentData>      {
                new SentimentData { SentimentText = "I love this product!" },
                new SentimentData { SentimentText = "This is a great movie." },
                new SentimentData { SentimentText = "I feel amazing!" },
                new SentimentData { SentimentText = "I am very happy with the service." },
                new SentimentData { SentimentText = "I hate this!" },
                new SentimentData { SentimentText = "This is terrible." },
                new SentimentData { SentimentText = "I feel awful." },
                new SentimentData { SentimentText = "I am very disappointed." }
          };
    }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using Microsoft.ML;
using Microsoft.ML.Data;
namespace ConAppAI1 {
    public class Program    {
        static void Main(string[] args)        {
            var mlContext = new MLContext(); // Create a new ML context
            var trainingData = SentimentPrediction.mylist; // Load training data
            var trainingDataView = mlContext.Data.LoadFromEnumerable(trainingData);
            // Define the data preparation and training pipeline
            var pipeline = mlContext.Transforms.Text.FeaturizeText("Features", nameof(SentimentData.SentimentText))
                .Append(mlContext.BinaryClassification.Trainers.SdcaLogisticRegression(labelColumnName: "Label", featureColumnName: "Features"));
            var model = pipeline.Fit(trainingDataView);// Train the model
            // Create a prediction engine
            var predictionEngine = mlContext.Model.CreatePredictionEngine<SentimentData, SentimentPrediction>(model);
            // Test the model
            var testData = new List<SentimentData>        {
                new SentimentData { SentimentText = "I love the new design!" },
                new SentimentData { SentimentText = "This is the worst experience I've ever had." }
            };
            foreach (var item in testData)      {
                var prediction = predictionEngine.Predict(item);
                Console.WriteLine($"Text: {item.SentimentText}");
                Console.WriteLine($"Prediction: {(prediction.Prediction ? "Positive" : "Negative")}");
                Console.WriteLine($"Probability: {prediction.Probability}\nScore: {prediction.Score}");
                Console.WriteLine();
            }       }   }
}
```

Output

- Text: I love the new design!
- Prediction: Negative
- Probability: 0.98396903
- Score: 4.117072

- Text: This is the worst experience I've ever had.
- Prediction: Negative
- Probability: 0.5551207
- Score: 0.22138253

Improved Diagnostics and Observability

.Net 8

Improved Diagnostics and Observability

- .NET 8 brings several enhancements to diagnostics and observability, making it easier for developers to monitor, troubleshoot, and optimize their applications. Here are some of the key improvements:
- **Enhanced Logging and Tracing**
  - **Structured Logging**: .NET 8 improves support for structured logging, enabling more detailed and context-rich log messages. This helps in better understanding application behavior and identifying issues.
  - **OpenTelemetry Integration**: Improved integration with OpenTelemetry, an open-source observability framework, allows for better tracing and monitoring across distributed systems. This includes support for distributed tracing, metrics, and logging.
- **Improved Performance Counters and Metrics**
  - **Extended Metrics**: .NET 8 extends the range of performance counters and metrics available, providing deeper insights into application performance. This includes more granular metrics for CPU usage, memory allocation, and garbage collection.
  - **Custom Metrics**: Developers can define custom metrics more easily, allowing for tailored monitoring solutions specific to their application's needs.
- **Better Exception Handling and Reporting**
  - **Enhanced Exception Logging**: Improved mechanisms for logging exceptions, including more detailed stack traces and contextual information. This helps in quicker identification and resolution of issues.
  - **Error Reporting Tools**: Integration with popular error reporting tools (like Sentry, Application Insights) is enhanced, making it easier to track and analyze application errors in real-time.

..Improved Diagnostics and Observability

- **Diagnostics Tools Improvements**
  - **dotnet-dump**: Enhanced dotnet-dump tool with better diagnostics capabilities, allowing for more comprehensive analysis of memory dumps and application state.
  - **dotnet-trace and dotnet-counters**: Improved functionality and usability of dotnet-trace and dotnet-counters, enabling more effective real-time tracing and performance monitoring.
- **Advanced Profiling Tools**
  - **Profiling APIs**: New and improved profiling APIs enable developers to build custom profiling solutions, providing more flexibility and control over performance analysis.
  - **Integration with Profiler Tools**: Better integration with popular profiler tools like Visual Studio Profiler, JetBrains dotTrace, and others, ensuring a smoother experience for performance tuning.
- **Simplified Diagnostic Configuration**
  - **Centralized Configuration**: .NET 8 introduces more straightforward ways to configure diagnostics settings, allowing for centralized management of logging, tracing, and metrics.
  - **Configuration APIs**: New APIs for dynamically adjusting diagnostic settings at runtime, providing more flexibility in how applications handle diagnostics.

..Improved Diagnostics and Observability

- **Enhanced Security and Privacy for Diagnostics**
  - **Secure Logging**: Improvements in how sensitive information is handled in logs, ensuring that personal or confidential data is appropriately redacted or anonymized.
  - **Compliance and Auditing**: Tools and features to help ensure that diagnostic data collection complies with various regulations and standards, such as GDPR.
- **Improved Observability in Cloud Environments**
  - **Cloud-native Diagnostics**: Enhanced support for cloud-native diagnostics, with better integration with cloud monitoring services like Azure Monitor, AWS CloudWatch, and Google Cloud Monitoring.
  - **Kubernetes Support**: Improved support for observability in Kubernetes environments, including easier integration with tools like Prometheus and Grafana.
- These improvements in .NET 8 significantly enhance the ability to diagnose, monitor, and optimize .NET applications.
- With better tools, integrations, and APIs, developers can achieve deeper insights into their applications' performance and behavior, leading to more robust and efficient software.

Logging Enhancements

- **Structured Logging:**
  - Overview and benefits.
  - Implementation using ILogger.
  - Example: Creating structured logs with contextual information.
- **Integrating with Serilog:**
  - Setting up Serilog.
  - Logging to different sinks (e.g., files, console, databases).
  - Example: Using Serilog for structured logging.

Example1: Structured Logging

- Structured logging allows for logs to be parsed and queried more efficiently by storing them in a structured format (e.g., JSON).
- Implementation: Using the ILogger interface to create structured logs.

- Create an ASP.Net Web App (MVC) project : WebAppLog1
- Edit Program.cs
- Create WeatherForecastController : Right-click on the "Controllers" folder, select "Add" > "Controller". Choose "API Controller - Empty" and name it WeatherForecastController.
- Press F5 to run the application.
- Open a browser and navigate to
- https://localhost:5001/weatherforecast
- (adjust the URL based on your local configuration).
- View Logs: Check the output window in Visual Studio 2022 or the console where the application is running. You should see log messages.

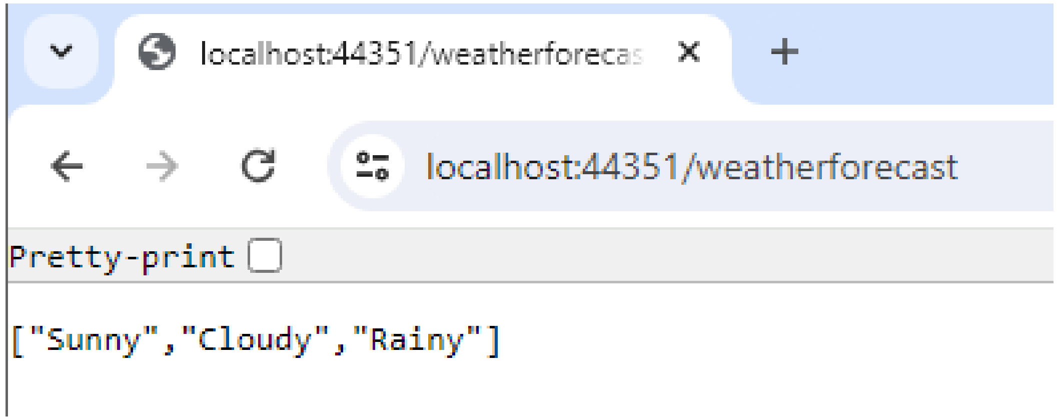
Program.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
var builder = WebApplication.CreateBuilder(args);
// Add services to the container.
builder.Services.AddControllers();
// Configure logging
builder.Logging.ClearProviders(); // Optional: Clear default providers
builder.Logging.AddConsole();
builder.Logging.AddDebug();
builder.Logging.AddEventSourceLogger();
// You can add other logging providers like Serilog, NLog, etc.

var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment())   {
    app.UseDeveloperExceptionPage();
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

WeatherForecastController.cs

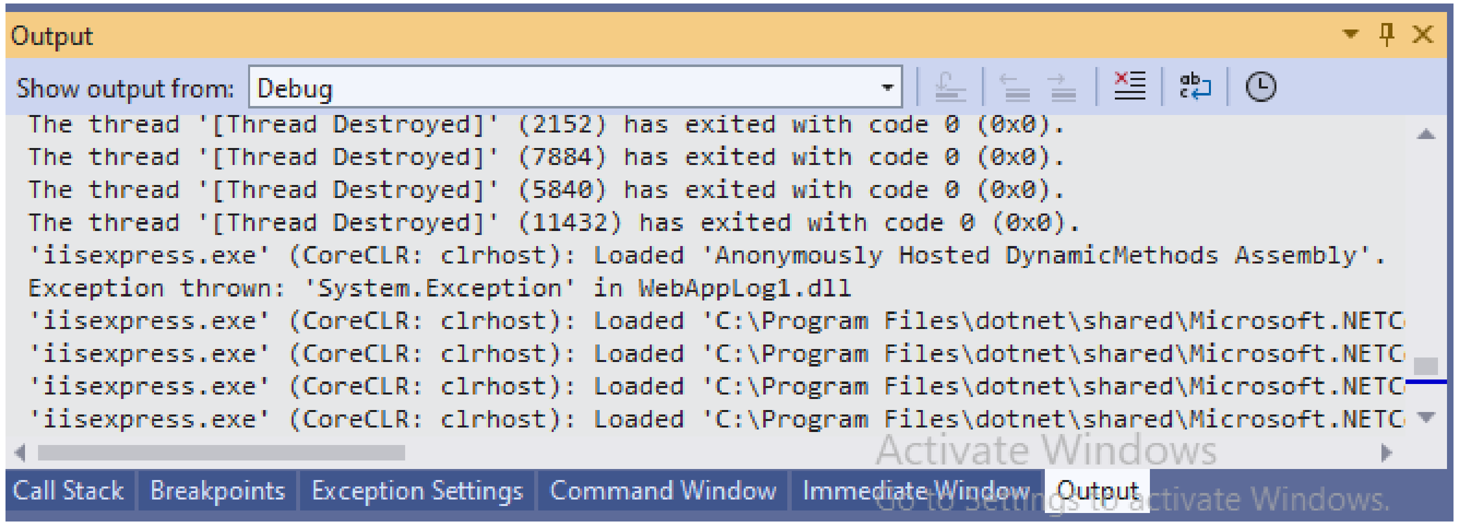
```
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Logging;
using System.Collections.Generic;
namespace MyWebApp.Controllers  {
    [ApiController]
    [Route("[controller]")]
    public class WeatherForecastController : ControllerBase    {
        private readonly ILogger<WeatherForecastController> _logger;
        public WeatherForecastController(ILogger<WeatherForecastController> logger)        {
            _logger = logger;
        }
        [HttpGet]
        public IEnumerable<string> Get()        {
            _logger.LogInformation("WeatherForecastController Get method called.");
            try    {   // Simulate a potential issue
                throw new System.Exception("Sample exception");
            }
            catch (System.Exception ex)        {
                _logger.LogError(ex, "An error occurred while fetching weather data.");
            }
            return new string[] { "Sunny", "Cloudy", "Rainy" };
        }
    }
}
```

Output : https://localhost:44351/weatherforecast



localhost:44351/weatherforecas
localhost:44351/weatherforecast
Pretty-print

```
["Sunny","Cloudy","Rainy"]
```



```
The thread '[Thread Destroyed]' (2152) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (7884) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (5840) has exited with code 0 (0x0).
The thread '[Thread Destroyed]' (11432) has exited with code 0 (0x0).
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'Anonymously Hosted DynamicMethods Assembly'.
Exception thrown: 'System.Exception' in WebAppLog1.dll
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
```

Activate Windows
Go to Settings to activate Windows.

Example2: Structured Logging with Serilog

- For more advanced logging, you can integrate Serilog.
- Install Serilog packages via NuGet:
- dotnet add package Serilog
- dotnet add package Serilog.Extensions.Logging
- dotnet add package Serilog.Sinks.Console
- dotnet add package Serilog.AspNetCore

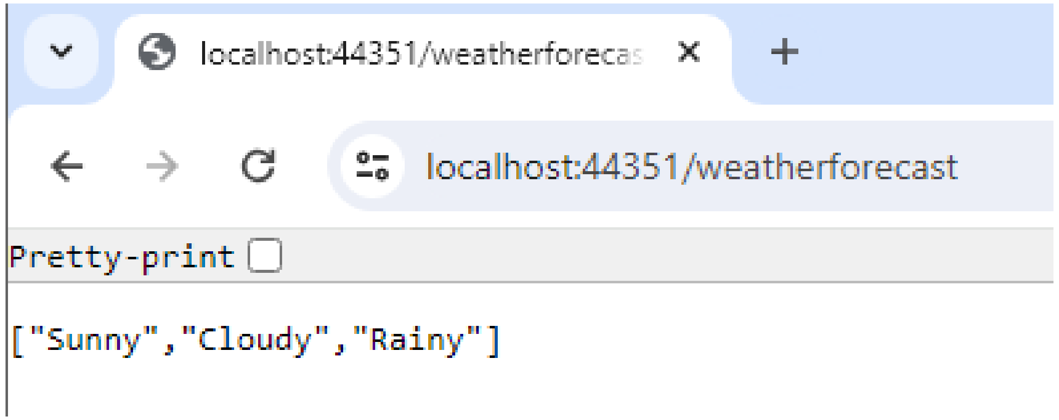
- Edit the ASP.Net Web App (MVC) project : WebAppLog1
- Edit Program.cs

- Press F5 to run the application.
- Open a browser and navigate to
- https://localhost:5001/weatherforecast
- (adjust the URL based on your local configuration).
- View Logs: Check the output window in Visual Studio 2022 or the console where the application is running. You should see log messages.

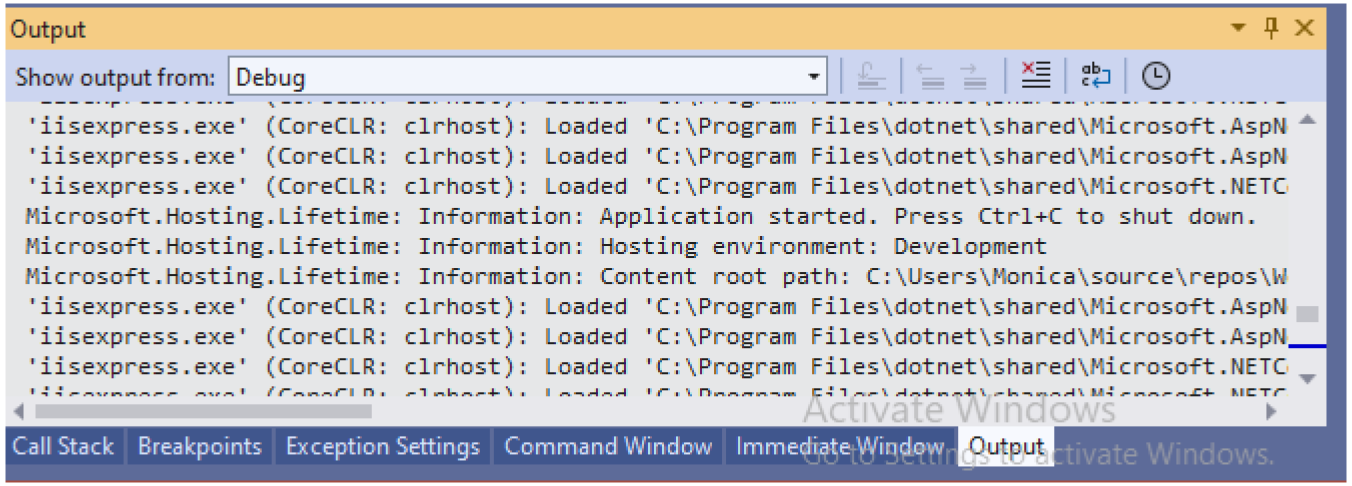
Program.cs

```
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Serilog;
var builder = WebApplication.CreateBuilder(args);

// Configure Serilog
Log.Logger = new LoggerConfiguration()
    .WriteTo.Console()
    .CreateLogger();
builder.Host.UseSerilog();

// Add services to the container.
builder.Services.AddControllers();
var app = builder.Build();
// Configure the HTTP request pipeline.
if (app.Environment.IsDevelopment()){
    app.UseDeveloperExceptionPage();
}
app.UseHttpsRedirection();
app.UseAuthorization();
app.MapControllers();
app.Run();
```

Output : https://localhost:44351/weatherforecast


localhost:44351/weatherforecas
localhost:44351/weatherforecast
Pretty-print

```
["Sunny","Cloudy","Rainy"]
```



```
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.AspN
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.AspN
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
Microsoft.Hosting.Lifetime: Information: Application started. Press Ctrl+C to shut down.
Microsoft.Hosting.Lifetime: Information: Hosting environment: Development
Microsoft.Hosting.Lifetime: Information: Content root path: C:\Users\Monica\source\repos\W
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.AspN
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.AspN
'iisexpress.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETC
```

Performance Metrics and Counters

- In .NET 8, performance metrics and counters are essential tools for monitoring the health and performance of your applications.
- You can use built-in tools like dotnet-counters as well as custom metrics and counters for more granular monitoring.

Example

- **Create a new project as** "ASP.NET Core Web App" and click "Next". Configure your project (name, location, etc.) and select ".NET 8.0" as the framework. Click "Create".
- **Setup Built-in Performance Metrics**
  - .NET provides built-in performance counters that you can monitor using dotnet-counters.
- **Run Your Application:**
  - Start your application from Visual Studio 2022 (Press F5).
- **Open Terminal:**
  - Open a terminal or command prompt.
- **List Available Counters:**
  - Identify the process ID (PID) of your running application. You can find it in Task Manager or by using a command like
    - dotnet tool install --global dotnet-counters
    - dotnet-counters ps
  - List the available counters for the process:
    - dotnet-counters list
- **Monitor Specific Counters:**
  - Monitor specific counters in real-time:
    - dotnet-counters monitor System.Runtime --process-id <process-id>
    - dotnet-counters monitor --process-id <process-id>

..Tools-> Developer Command prompt (After the application is running)

- C:\Users\Monica\source\repos\WebAppLog1>dotnet tool install --global dotnet-counters
- You can invoke the tool using the following command: dotnet-counters
- Tool 'dotnet-counters' (version '8.0.510501') was successfully installed.

```
C:\Users\Monica\source\repos\WebAppLog1>dotnet-counters ps
 12700  iisexpress                          ogram Files\IIS Express\iisexpress.exe  AppLog1" /apppool:"WebAppLog1 AppPool"
  8200  ServiceHub.DataWarehouseHost         t.x64\ServiceHub.DataWarehouseHost.exe  BucketFiltersToAddDumpsToFaults\":[]}"
  4948  ServiceHub.Host.dotnet.x64           net.x64\ServiceHub.Host.dotnet.x64.exe  BucketFiltersToAddDumpsToFaults\":[]}"
 13268  ServiceHub.IdentityHost              dotnet.x64\ServiceHub.IdentityHost.exe  BucketFiltersToAddDumpsToFaults\":[]}"
  5036  ServiceHub.IndexingService           net.x64\ServiceHub.IndexingService.exe  BucketFiltersToAddDumpsToFaults\":[]}"
  9064  ServiceHub.RoslynCodeAnalysisService  rviceHub.RoslynCodeAnalysisService.exe  BucketFiltersToAddDumpsToFaults\":[]}"
  6436  ServiceHub.VSDetouredHost            tnet.x64\ServiceHub.VSDetouredHost.exe  BucketFiltersToAddDumpsToFaults\":[]}"
```

..Tools-> Developer Command prompt (After the application is running)

- C:\Users\Monica\source\repos\WebAppLog1>dotnet-counters list
- Showing well-known counters for .NET (Core) version 6.0 only. Specific processes may support additional counters.
- System.Runtime

cpu-usage	The percent of process' CPU usage relative to all of the system CPU resources [0-100]
working-set	Amount of working set used by the process (MB)
gc-heap-size	Total heap size reported by the GC (MB)
gen-0-gc-count	Number of Gen 0 GCs between update intervals
gen-1-gc-count	Number of Gen 1 GCs between update intervals
gen-2-gc-count	Number of Gen 2 GCs between update intervals
time-in-gc	% time in GC since the last GC
gen-0-size	Gen 0 Heap Size
gen-1-size	Gen 1 Heap Size
gen-2-size	Gen 2 Heap Size
loh-size	LOH Size
poh-size	POH (Pinned Object Heap) Size
alloc-rate	Number of bytes allocated in the managed heap between update intervals
gc-fragmentation	GC Heap Fragmentation

- ……..

..Tools-> Developer Command prompt (After the application is running)

```
C:\Users\Monica\source\repos\WebAppLog1>dotnet-counters monitor --process-id 12700
 % Time in GC since last GC (%)                          0
    Allocation Rate (B / 1 sec)                      32,768
    CPU Usage (%)                                     0.806
    Exception Count (Count / 1 sec)                        0
    GC Committed Bytes (MB)                             17.383
    GC Fragmentation (%)                              27.448
    GC Heap Size (MB)                                 6.735
    Gen 0 GC Budget (MB)                               15
    Gen 0 GC Count (Count / 1 sec)                         0
    Gen 0 Size (B)                                6,36,336
    Gen 1 GC Count (Count / 1 sec)                         0
    Gen 1 Size (B)                                  89,136
    Gen 2 GC Count (Count / 1 sec)                         0
    Gen 2 Size (B)                               13,99,816
    IL Bytes Jitted (B)                           2,94,158
    LOH Size (B)                                    98,384
…….
```

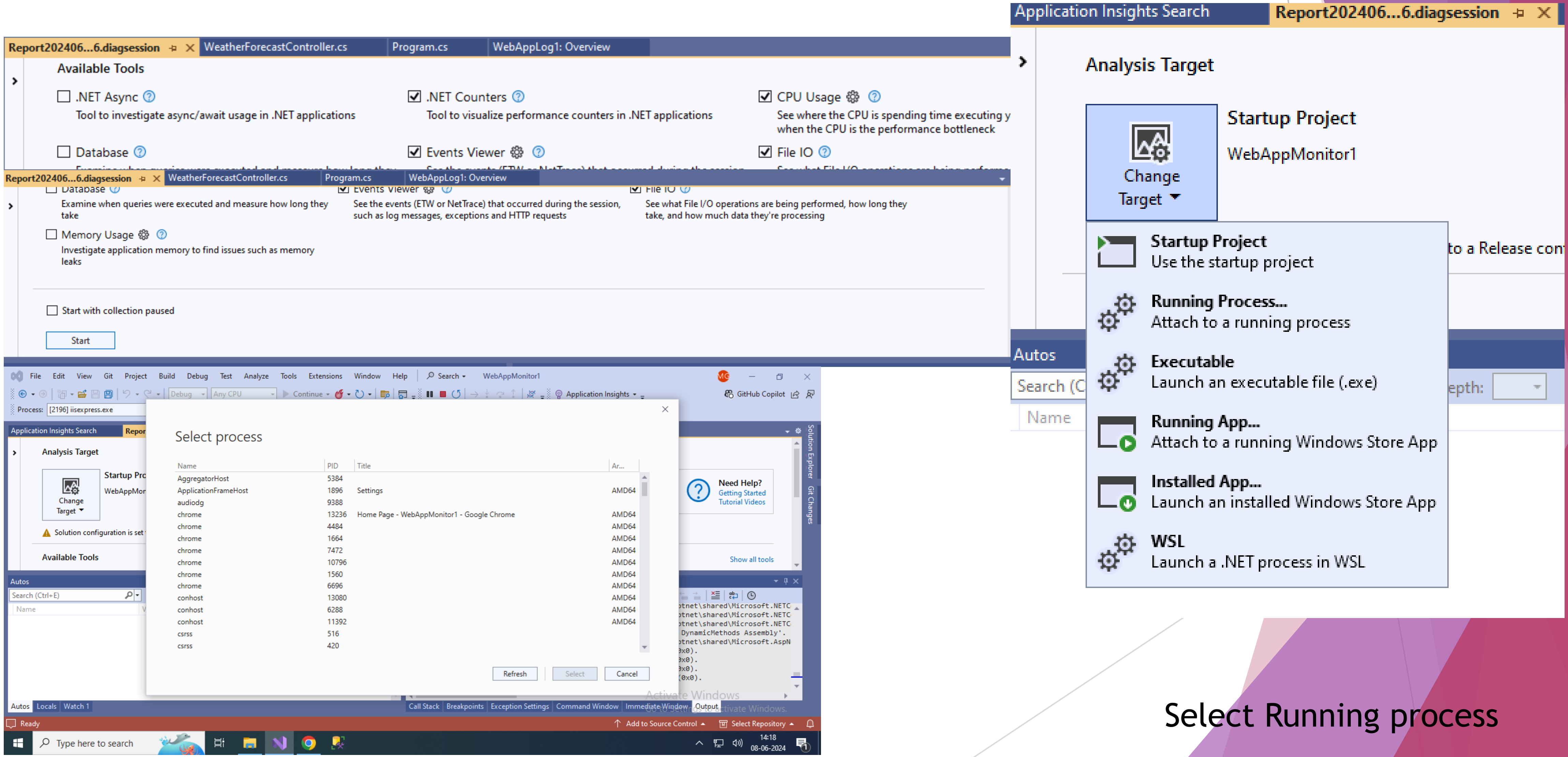
Diagnostics Tools in .NET 8

- In .NET 8, Microsoft has enhanced the diagnostics and performance monitoring tools available for developers.
- These tools are crucial for identifying and resolving performance issues, memory leaks, and other runtime problems in your applications.
- Visual Studio 2022 provides a powerful integrated environment to utilize these tools effectively.

Performance Profiler

- The performance profiler in Visual Studio 2022 helps you analyze CPU and memory usage, identify performance bottlenecks, and optimize your application's performance.
- **Example: Using the Performance Profiler**
- **Start the Profiler**:
  - Run your .NET application in Debug mode (Press F5).
  - Go to `Debug > Performance Profiler` or press `Alt + F2`.
  - Select the type of performance you want to profile (e.g., CPU Usage, Memory Usage).
- **Analyze the Results**:
  - The profiler will capture performance data as you interact with your application.
  - View CPU Usage, Memory Usage, and other metrics.
  - Use the Call Tree, Hot Path, or other views to identify performance bottlenecks.
  - Optimize your code based on the profiler's recommendations.

Output : Tools -> Performance Analysers



Report202406...6.diagsession | WeatherForecastController.cs | Program.cs | WebAppLog1: Overview

Available Tools

- [ ] .NET Async — Tool to investigate async/await usage in .NET applications
- [✓] .NET Counters — Tool to visualize performance counters in .NET applications
- [✓] CPU Usage — See where the CPU is spending time executing y… when the CPU is the performance bottleneck
- [ ] Database — Examine when queries were executed and measure how long they take
- [✓] Events Viewer — See the events (ETW or NetTrace) that occurred during the session, such as log messages, exceptions and HTTP requests
- [✓] File IO — See what File I/O operations are being performed, how long they take, and how much data they're processing
- [ ] Memory Usage — Investigate application memory to find issues such as memory leaks

- [ ] Start with collection paused

Start

File Edit View Git Project Build Debug Test Analyze Tools Extensions Window Help Search WebAppMonitor1

Process: [2196] iisexpress.exe

Application Insights Search | Report202406...6.diagsession

Analysis Target

Change Target — Startup Project: WebAppMonitor1

Solution configuration is set …

Available Tools

Select process

Name	PID	Title	Ar...
AggregatorHost	5384		
ApplicationFrameHost	1896	Settings	AMD64
audiodg	9388		
chrome	13236	Home Page - WebAppMonitor1 - Google Chrome	AMD64
chrome	4484		AMD64
chrome	1664		AMD64
chrome	7472		AMD64
chrome	10796		AMD64
chrome	1560		AMD64
chrome	6696		AMD64
conhost	13080		AMD64
conhost	6288		AMD64
conhost	11392		AMD64
csrss	516		
csrss	420		

Refresh | Select | Cancel

Need Help? Getting Started · Tutorial Videos · Show all tools

```
otnet\shared\Microsoft.NETC
otnet\shared\Microsoft.NETC
otnet\shared\Microsoft.NETC
 DynamicMethods Assembly'.
otnet\shared\Microsoft.AspN
0x0).
0x0).
0x0).
(0x0).
```

Ready · Add to Source Control · Select Repository · 14:18 08-06-2024

Change Target menu:

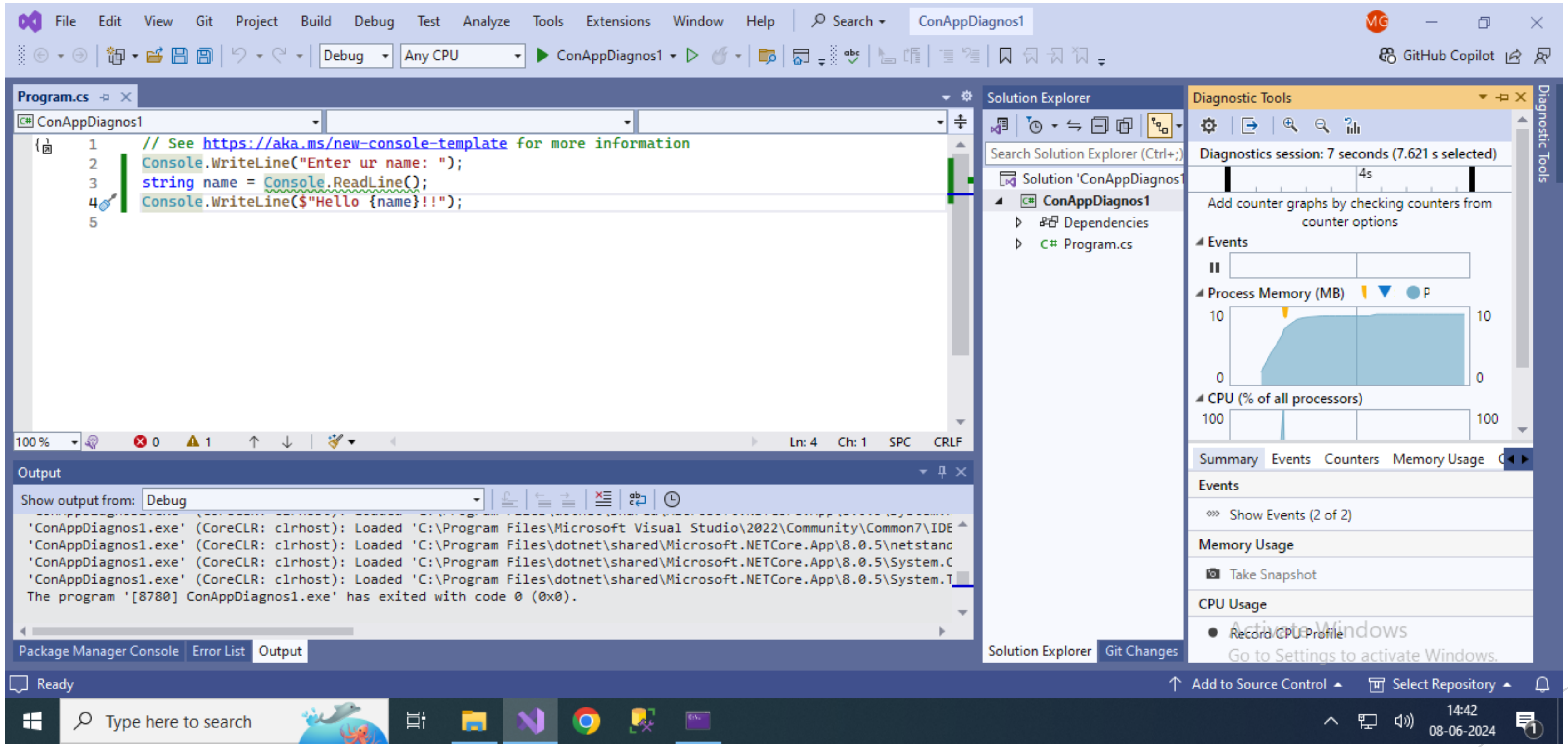
- **Startup Project** — Use the startup project
- **Running Process...** — Attach to a running process
- **Executable** — Launch an executable file (.exe)
- **Running App...** — Attach to a running Windows Store App
- **Installed App...** — Launch an installed Windows Store App
- **WSL** — Launch a .NET process in WSL

Select Running process

Diagnostics Tools Window

- The Diagnostics Tools window in Visual Studio 2022 provides real-time information about your application's CPU, memory, and network usage.
- **Example: Using the Diagnostics Tools Window**
- **Run Your Application:**
  - Start your application in Debug mode (Press F5).
- **Open the Diagnostics Tools Window:**
  - Go to `Debug > Windows > Show Diagnostic Tools` or press `Ctrl + Alt + F2`.
  - The Diagnostics Tools window will show real-time data such as CPU Usage, Memory Usage, and events.
- **Analyze the Data:**
  - Use the timeline to navigate and inspect events during different time periods.
  - Analyze memory usage, CPU usage, and thread activity.

Output



```
// See https://aka.ms/new-console-template for more information
Console.WriteLine("Enter ur name: ");
string name = Console.ReadLine();
Console.WriteLine($"Hello {name}!!");
```

```
'ConAppDiagnos1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\Microsoft Visual Studio\2022\Community\Common7\IDE
'ConAppDiagnos1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\netstanc
'ConAppDiagnos1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.C
'ConAppDiagnos1.exe' (CoreCLR: clrhost): Loaded 'C:\Program Files\dotnet\shared\Microsoft.NETCore.App\8.0.5\System.T
The program '[8780] ConAppDiagnos1.exe' has exited with code 0 (0x0).
```

Extended API Authoring

.Net 8

Extended API Authoring

- Extended API Authoring with .NET 8 and Visual Studio 2022 refers to the process of creating robust and scalable APIs using the latest version of the .NET framework and the integrated development environment provided by Visual Studio 2022.
- Extended API authoring with .NET 8 and Visual Studio 2022 allows developers to leverage the latest advancements in the .NET ecosystem to build high-performance, scalable APIs efficiently.
- By using the latest features in .NET 8 and the enhanced development tools in Visual Studio 2022, developers can create APIs that are not only robust but also maintainable and secure.

.NET 8

- .NET 8 is the latest major release of the .NET framework. It includes several new features and improvements that are beneficial for API development:
- **C# 10**: .NET 8 supports C# 10, which introduces new language features like record types, global using directives, and more concise syntax.
- **Improved Performance**: .NET 8 comes with performance improvements in various areas, including garbage collection and just-in-time (JIT) compilation.
- **Async Streams**: Allows for the consumption and processing of asynchronous data streams.
- **.NET MAUI**: .NET 8 introduces support for .NET Multi-platform App UI (MAUI), which enables building cross-platform applications, including mobile apps, using a single codebase.
- **Nullable Reference Types**: Enhancements in nullability checking to help prevent null reference exceptions.

Visual Studio 2022

- Visual Studio 2022 is the latest version of Microsoft’s integrated development environment (IDE) for building applications in various programming languages, including C# and .NET. It includes several features that aid in API development:
- **Enhanced Performance**: Visual Studio 2022 is optimized for better performance, especially when working with large solutions and during debugging sessions.
- **Improved Code Analysis and Navigation**: Enhanced code analysis tools help identify issues early in the development process, improving code quality and maintainability.
- **Better Git Integration**: Improved Git integration with better diff view, conflict resolution, and pull request management directly within the IDE.
- **Container Development**: Integrated tools for container development, making it easier to develop, test, and deploy applications as Docker containers.

Extended API Authoring Process

- To author an extended API with .NET 8 and Visual Studio 2022, you typically follow these steps:
  - **Create a New Project**: Specify .NET 8 as the target framework-> Name your project and click "Create".
  - **Define Your API**: Define endpoints and their corresponding HTTP methods (GET, POST, PUT, DELETE, etc.)->Define data models (POCO classes).Implement business logic (services).
  - **Use C# 10/12 Features**: Take advantage of new C# 10/12 language features to write more concise and readable code.
  - **Dependency Injection**: Utilize .NET's built-in dependency injection (DI) container for loosely coupled components and easier unit testing.
  - **Async Programming**:Use async/await to handle asynchronous operations efficiently, especially when accessing external services or databases.
  - **Middleware and Filters**:Implement middleware to handle cross-cutting concerns such as exception handling, logging, and authentication.Use action filters to add behavior to controller actions.
  - **Testing**:Write unit tests and integration tests to verify the functionality of your API.Use built-in testing frameworks such as xUnit or NUnit.
  - **Deployment**:Deploy your API to a hosting platform (Azure, AWS, etc.) using tools provided in Visual Studio 2022.

Example

- Create Asp.Net Web API project : WebApiAuth1

- Edit Program.cs to have minimal support
- Run the Application

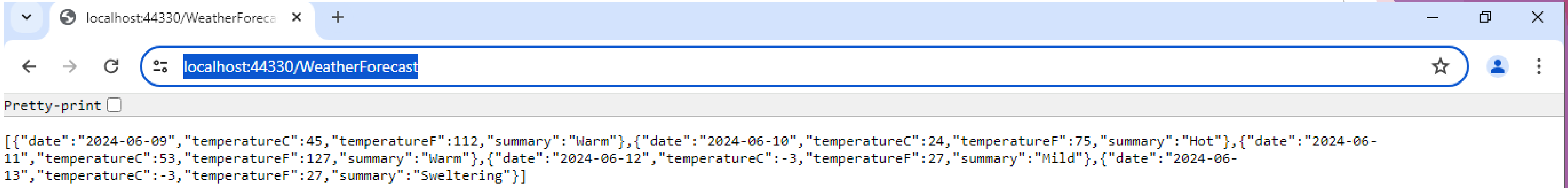
- To deploy the API, right-click on the project name in Solution Explorer.
- Choose Publish.
- Follow the publish wizard to deploy your application to Azure, AWS, or any other hosting provider.

- Check the application in the output

Program.cs

- using Microsoft.AspNetCore.Hosting;
- using Microsoft.Extensions.Hosting;
- var builder = WebApplication.CreateBuilder(args);
- //// Add services to the container.
- builder.Services.AddControllers();
- //// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle
- //builder.Services.AddEndpointsApiExplorer();
- //builder.Services.AddSwaggerGen();
- var app = builder.Build();
- // Configure the HTTP request pipeline.
- if (app.Environment.IsDevelopment())
- {
-     //app.UseSwagger();
-     //app.UseSwaggerUI();
-     app.UseDeveloperExceptionPage();
- }
- app.UseHttpsRedirection();
- //app.UseAuthorization();
- app.MapControllers();
- app.Run();

Output: https://localhost:44330/WeatherForecast


localhost:44330/WeatherForeca
localhost:44330/WeatherForecast
Pretty-print

```
[{"date":"2024-06-09","temperatureC":45,"temperatureF":112,"summary":"Warm"},{"date":"2024-06-10","temperatureC":24,"temperatureF":75,"summary":"Hot"},{"date":"2024-06-
11","temperatureC":53,"temperatureF":127,"summary":"Warm"},{"date":"2024-06-12","temperatureC":-3,"temperatureF":27,"summary":"Mild"},{"date":"2024-06-
13","temperatureC":-3,"temperatureF":27,"summary":"Sweltering"}]
```

Training Ends

Thank you