Gauntlet:
.NET / C#
Tips and Tricks

Kevin Griffin

#1 : Use DateTimeOffset

```
// create a time, but in what timezone?
DateTime meetingTimeInVirginia = new DateTime(2024, 2, 10, 15, 0, 0);

// let's use a Timezone!
DateTimeOffset meetingTimeInVirginia = new DateTimeOffset(2024, 2, 10, 15, 0, 0, TimeSpan.FromHours(-5));
```

#2 : Convert from DateTime to DateTimeOffset

```
// create a time, but in what timezone?
DateTime meetingTimeInVirginia = new DateTime(2024, 2, 10, 15, 0, 0);

TimeZoneInfo easternZone =
    TimeZoneInfo.FindSystemTimeZoneById("Eastern Standard Time");
DateTimeOffset meetingTimeInVirginiaOffset =
    TimeZoneInfo.ConvertTimeToUtc(meetingTimeInVirginia, easternZone);
```

#3 : NodaTime

```
// Define the time zones for Virginia and London
var virginiaZone = DateTimeZoneProviders.Tzdb['America/New_York']
var londonZone = DateTimeZoneProviders.Tzdb['Europe/London']

// Define the local time for the meeting in Virginia
var localMeetingTimeInVirginia = new LocalDateTime(2024, 2, 10, 15, 0) // 3 PM

// Convert the local time in Virginia to a ZonedDateTime
var zonedMeetingTimeInVirginia =
  localMeetingTimeInVirginia.InZoneLeniently(virginiaZone)

// Convert Virginia time to London time
var zonedMeetingTimeInLondon = zonedMeetingTimeInVirginia.WithZone(londonZone)

// Meeting Time in Virginia (Eastern Time): 2024-02-10T15:00:00 America/New_York (-05)
Console.WriteLine("Meeting Time in Virginia (Eastern Time): " + zonedMeetingTimeInVirginia)

// Meeting Time in London (GMT): 2024-02-10T20:00:00 Europe/London (+00)
Console.WriteLine("Meeting Time in London (GMT): " + zonedMeetingTimeInLondon)
```

#4 : DateOnly / TimeOnly

```
// Creating a DateOnly instance for the current date
DateOnly today = DateOnly.FromDateTime(DateTime.Now);

// Returns "02/06/2024"
Console.WriteLine($"Today's Date: {today}");
```

#4 : DateOnly / TimeOnly

```
// Creating a TimeOnly instance for the current date
TimeOnly today = TimeOnly.FromDateTime(DateTime.Now);

// Returns 23:20 (yes, 24-hour time)
Console.WriteLine($"Today's Time: {today}");
// Returns 11:22 PM
Console.WriteLine($"Today's Time: {today:hh:mm tt}");
```

Hi,
I’m Kevin

Kevin Griffin

15-time Microsoft MVP
President, .NET Foundation

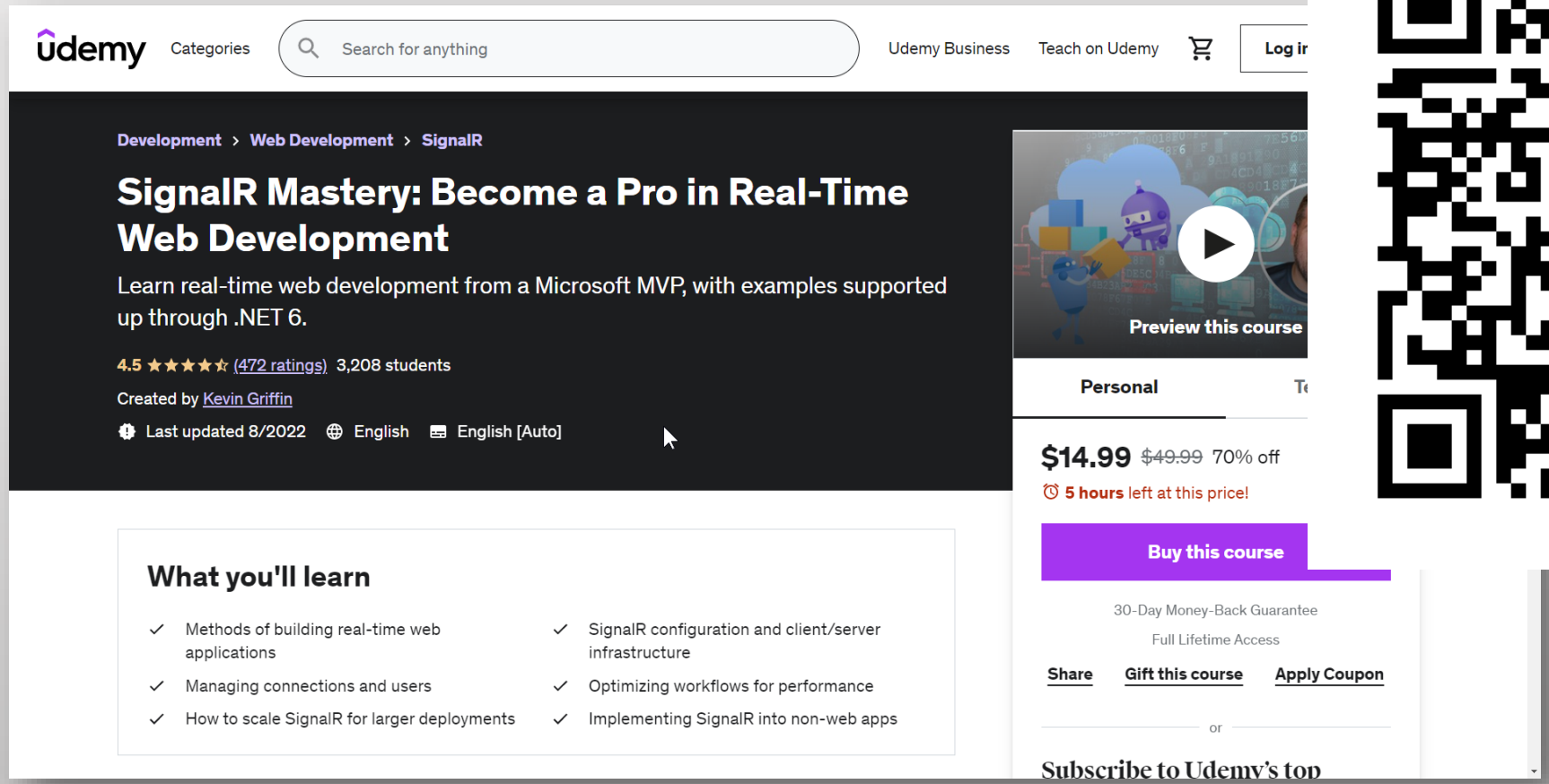
ASP.NET Core, Azure, and Web

consultwithgriff.com
twitter.com/@1kevgriff
bbiz.io/@1kevgriff

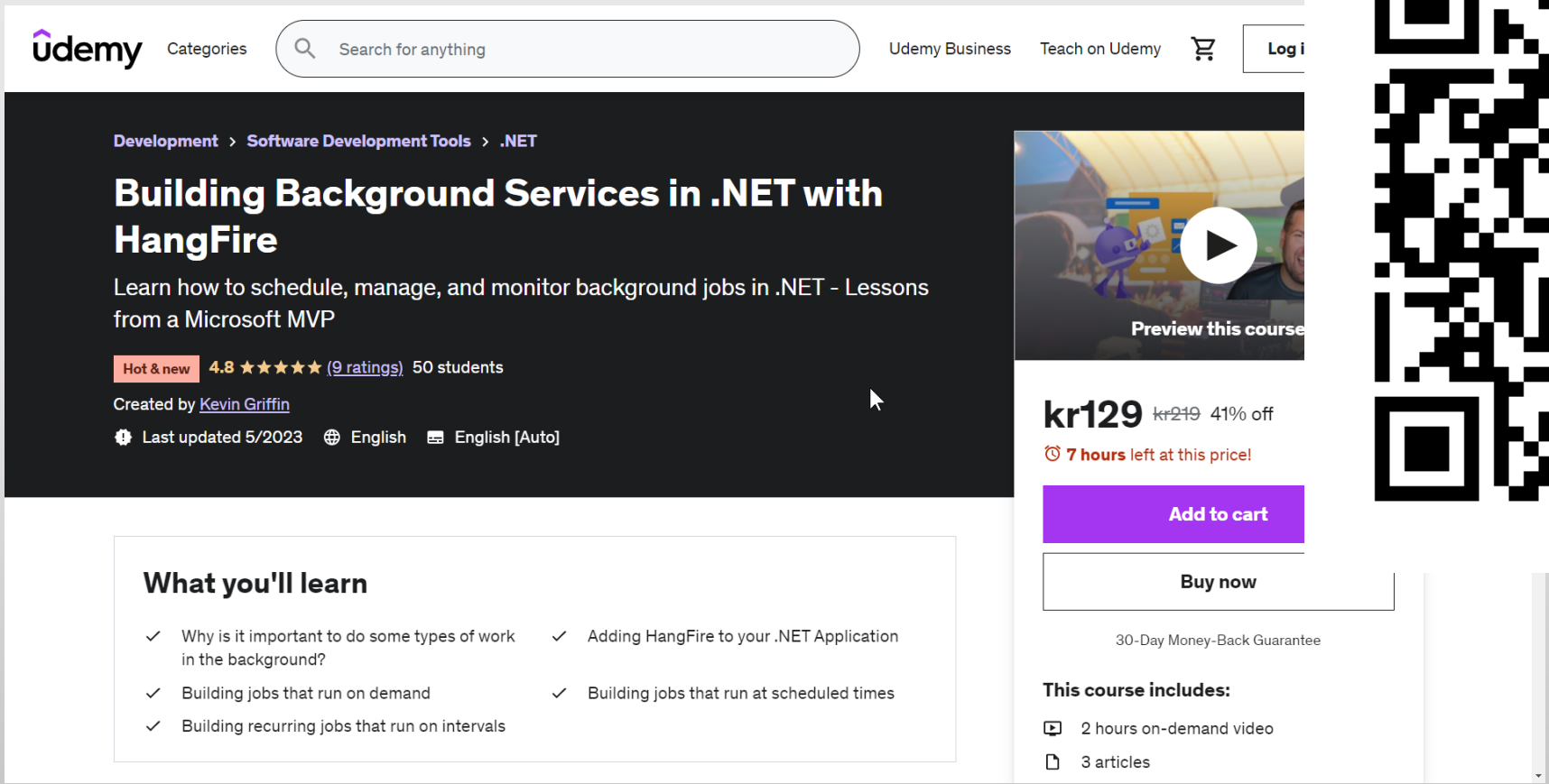
Join the .NET Foundation for FREE!

join.dotnetfoundation.org

Courses


udemy
Categories
Search for anything
Udemy Business
Teach on Udemy
Log ir
Development › Web Development › SignalR
SignalR Mastery: Become a Pro in Real-Time Web Development
Learn real-time web development from a Microsoft MVP, with examples supported up through .NET 6.
4.5 (472 ratings) 3,208 students
Created by Kevin Griffin
Last updated 8/2022
English
English [Auto]
Preview this course
Personal
$14.99 $49.99 70% off
5 hours left at this price!
Buy this course
30-Day Money-Back Guarantee
Full Lifetime Access
Share
Gift this course
Apply Coupon
or
Subscribe to Udemy's top
What you'll learn
Methods of building real-time web applications
SignalR configuration and client/server infrastructure
Managing connections and users
Optimizing workflows for performance
How to scale SignalR for larger deployments
Implementing SignalR into non-web apps

Courses


udemy
Categories
Search for anything
Udemy Business
Teach on Udemy
Log i
Development › Software Development Tools › .NET
Building Background Services in .NET with HangFire
Learn how to schedule, manage, and monitor background jobs in .NET - Lessons from a Microsoft MVP
Hot & new
4.8 ★★★★★ (9 ratings) 50 students
Created by Kevin Griffin
Last updated 5/2023
English
English [Auto]
Preview this course
kr129 kr219 41% off
7 hours left at this price!
Add to cart
Buy now
30-Day Money-Back Guarantee
This course includes:
2 hours on-demand video
3 articles
What you'll learn
Why is it important to do some types of work in the background?
Adding HangFire to your .NET Application
Building jobs that run on demand
Building jobs that run at scheduled times
Building recurring jobs that run on intervals

#5 : Use Patterns

```
public string GetShapeName(Shape shape)
{
    switch (shape)
    {
        case Shape { NumberOfSides: 3, Color: "Red" }: return "Red Triangle";

        case Shape { NumberOfSides: 3, Color: "Blue" }: return "Blue Triangle";

        case Shape { NumberOfSides: 3 }: return "Triangle"; // Default for triangle

        case Shape { NumberOfSides: 4, Color: "Red" }: return "Red Rectangle";

        case Shape { NumberOfSides: 4, Color: "Blue" }: return "Blue Rectangle";

        case Shape { NumberOfSides: 4 }: return "Rectangle";

        // ... more
    }
}
```

#6 : Use Switch Expressions

```
public string GetShapeName(Shape shape)
{
    return shape switch
    {
        { NumberOfSides: 3, Color: "Red" } => "Red Triangle",
        { NumberOfSides: 3, Color: "Blue" } => "Blue Triangle",
        { NumberOfSides: 3 } => "Triangle", // Default for triangle
        { NumberOfSides: 4, Color: "Red" } => "Red Rectangle",
        { NumberOfSides: 4, Color: "Blue" } => "Blue Rectangle",
        { NumberOfSides: 4 } => "Rectangle", // Default for rectangle
        // ... more patterns can be added here
        _ => "Unknown shape" // Default for all other cases
    };
}
```

#6 : Use Switch Expressions

```
private static string GeneratePricingData(ParagEvent src)
{
    return src switch
    {
        { LowPrice: 0, HighPrice: > 0 } => $"{src.HighPrice:C}",
        { LowPrice: > 0, HighPrice: 0 } => $"{src.LowPrice:C}",
        { LowPrice: 0, HighPrice: 0 } => string.Empty,
        _ => $"{src.LowPrice:C} - {src.HighPrice:C}"
    };
}
```

#7 : nameof() not Magic Strings

```
public static string PrintName(string name, int age)
{
    // don't do this
    if (string.IsNullOrWhiteSpace(name))
        throw new ArgumentException("Name cannot be null or empty.",
"name");

    // do this!
    if (string.IsNullOrWhiteSpace(name))
        throw new ArgumentException("Name cannot be null or empty.",
nameof(name));

    return $"{name} is {age} years old.";
}
```

#8 : String Interpolation

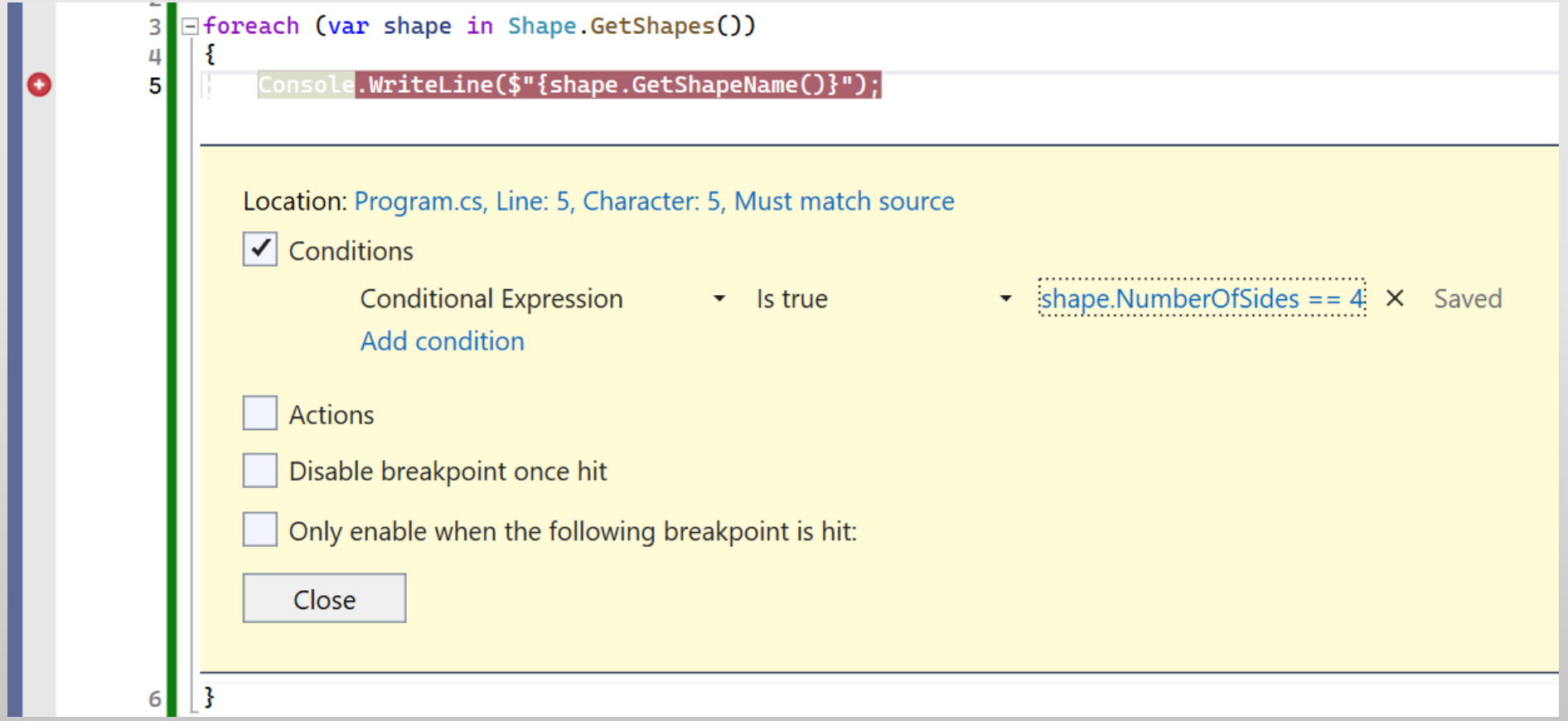
```
public static string PrintName(string name, int age)
{
    // old way
    return string.format("{0} is {1} years old.");

    // new hawtness
    return $"{name} is {age} years old.";
}
```

#9 : Formatting with String Interpolation

```
var standardFormat = currentUtcTime.ToString("yyyy-MM-dd HH:mm:ss");
```

```
var currentUtcTime = DateTime.UtcNow;

// Sample Output: "2024-02-06 15:45:30"
var standardFormat = $"{currentUtcTime:yyyy-MM-dd HH:mm:ss}";

// Sample Output: "2024-02-06T15:45:30.0000000Z"
var iso8601Format = $"{currentUtcTime:O}";

// Sample Output: "Mon, 06 Feb 2024 15:45:30 GMT"
var rfc1123Format = $"{currentUtcTime:R}";

// Sample Output: "2024-02-06T15:45:30"
var sortableFormat = $"{currentUtcTime:s}";

// Sample Output: "Monday, February 06, 2024"
var fullDayMonthFormat = $"{currentUtcTime:dddd, MMMM dd, yyyy}";

// Sample Output: "03:45 PM"
var time12HourFormat = $"{currentUtcTime:hh:mm tt}";

// Sample Output: "Monday, February 6, 2024 3:45:30 PM"
var universalLongTimeFormat = $"{currentUtcTime:U}";

// Sample Output: "Mon, Feb 06, 2024 15:45"
var customAbbreviatedFormat = $"{currentUtcTime:ddd, MMM dd, yyyy
HH:mm}";
```

#10 : Use Debugger.Break()



#10 : Use Debugger.Break()

```
foreach (var shape in Shape.GetShapes())
{
    if (shape.NumberOfSides == 4) Debugger.Break();

    Console.WriteLine($"{shape.GetShapeName()}");
}
```

#11 : Top-Level Statements

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Threading.Tasks;

// look ma! no Program or Main()

var builder = Host.CreateDefaultBuilder(args)
    .ConfigureServices((hostContext, services) =>
    {
        // Register services here
    });

var host = builder.Build();

// Application logic here

await host.RunAsync();
```

#12 : Use .NET Host

```
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using System.Threading.Tasks;

var builder = Host.CreateDefaultBuilder(args)
    .ConfigureServices((hostContext, services) =>
    {
        // Register services here
    });

var host = builder.Build();

// Application logic here

await host.RunAsync();
```

Add Configuration
Add Logger
Add Dependency Injection
(and more!)

#13 : ILogger

```
public class MyService
{
    private readonly ILogger<MyService> _logger;

    public MyService(ILogger<MyService> logger)
    {
        _logger = logger;
    }

    public void DoWork()
    {
      try
      {
        _logger.LogInformation("Doing work in MyService");
      } catch (Exception ex)
      {
        _logger.LogError(ex, "Error while doing work!");
      }
    }
}
```

#14 : Know Your Injection Types

```
var builder = Host.CreateDefaultBuilder(args)
    .ConfigureServices((_, services) =>
    {
        // New Instance Always
        services.AddTransient<MyService>();

        // New Instance Per Request
        services.AddScoped<MyService>();

        // One Instance Forever
        services.AddSingleton<MyService>();
    });
```

#15 : Multiple Injections for Same Type

```
public interface IValidationRule
{
    bool Validate(string input);
}

public class RuleNotNull : IValidationRule
{
    public bool Validate(string input) => !string.IsNullOrEmpty(input);
}

public class RuleLength : IValidationRule
{
    public bool Validate(string input) => input.Length > 5;
}

public class RuleUpperCase : IValidationRule
{
    public bool Validate(string input) => input.All(char.IsUpper);
}
```

```
// Configure Services
services.AddSingleton<IValidationRule, RuleNotNull>();
services.AddSingleton<IValidationRule, RuleLength>();
services.AddSingleton<IValidationRule, RuleUpperCase>();

public class MyService
{
    private readonly IEnumerable<IValidationRule> _rules;

    public MyService(IEnumerable<IValidationRule> rules)
    {
        _rules = rules;
    }

    public bool Validate(string input)
    {
        return _rules.All(rule => rule.Validate(input));
    }
}
```

#16 : Primary Constructors

```
public class MyService(IEnumerable<IValidationRule> rules,
                       ILogger<MyService> logger)
{
    public bool Validate(string input)
    {
        logger.LogInformation("Validating rules...");
        return rules.All(rule => rule.Validate(input));
    }
}
```

#17 : Default Values of Lambda

```
Func<int, int, int> add = (int x, int y = 10) => x + y;

add.Method.GetParameters()[1].DefaultValue; // 10
```

#18 : Use Records for Immutable Data

```
public record Person(string FirstName, string LastName);

var kevin = new Person("Kevin", "Griffin");
kevin.FirstName = "Bob"; // ERROR
```

#19 : Records, but smarter!

```
public record Person(string FirstName, string LastName)
{
    public string FullName => $"{FirstName} {LastName}";
}

var kevin = new Person("Kevin", "Griffin");
// kevin.FullName = "Kevin Griffin"
```

#20 : MinBy/MaxBy

```
List<Employee> employees = new List<Employee>
{
    new Employee("Alice", 30, 70000),
    new Employee("Bob", 28, 80000),
    new Employee("Charlie", 35, 60000),
    new Employee("Diana", 40, 90000)
};

employees.OrderBy(p => p.Salary).First(); // Diana has highest salary
employees.OrderBy(p => p.Salary).Last(); // Charlie has lowest salary
```

#20 : MinBy/MaxBy

```
List<Employee> employees = new List<Employee>
{
    new Employee("Alice", 30, 70000),
    new Employee("Bob", 28, 80000),
    new Employee("Charlie", 35, 60000),
    new Employee("Diana", 40, 90000)
};

var highestPaidEmployee = employees.MaxBy(e => e.Salary); // Diana
var lowestPaidEmployee = employees.MinBy(e => e.Salary); // Charlie
```

#21 : ValueTask over Task

```
public class Example
{
    public async ValueTask<string> GetDataAsync()
    {
        // method sometimes completes synchronously
        string data = CheckDataFromLocalCache();

        if (!string.IsNullOrWhitespace(data))
        {
            return data;
        }
        else
        {
            // Simulate an asynchronous operation
            return await GetData()
        }
    }

    private string CheckDataAvailability()
    {
        // TODO: get cached data
    }
}
```

#22 : Fun with Flags

```
[Flags]
public enum PersonAttributes
{
    None = 0,
    HasBlondeHair = 1 << 0,     // 0000_0001 in binary
    HasBrownHair = 1 << 1,      // 0000_0010 in binary
    HasBlackHair = 1 << 2,      // 0000_0100 in binary
    HasGreyHair = 1 << 3,       // 0000_1000 in binary
    HasNoHair = 1 << 4,         // 0001_0000 in binary
    HasBlueEyes = 1 << 5,       // 0010_0000 in binary
    HasBrownEyes = 1 << 6,      // 0100_0000 in binary
    HasHazelEyes = 1 << 7       // 1000_0000 in binary
}
```

#22 : Fun with Flags

```
// Set Kevin's attributes
PersonAttributes kevinsAttributes =
  PersonAttributes.HasBrownHair | PersonAttributes.HasGreyHair | PersonAttributes.HasBlueEyes;

// Convert Kevin's attributes to an integer for database storage
int kevinsAttributesValue = (int)kevinsAttributes;

// does Kevin have grey
// Check if Kevin has grey hair
if (kevinsAttributesValue.HasFlag(PersonAttributes.HasGreyHair))
{
    Console.WriteLine("Kevin has grey hair.");
}
else
{
    Console.WriteLine("Kevin does not have grey hair.");
}
```

#23 : Tuples instead of objects

```
class Program
{
    static void Main()
    {
        var (a, b) = SwapNumbers(5, 10);
        Console.WriteLine($"After swap: a = {a}, b = {b}");
    }

    static (int, int) SwapNumbers(int x, int y)
    {
        return (y, x);
    }
}
```

#23 : Tuples instead of objects

```
public class Program
{
    public static void Main()
    {
        var projectInfo = GetProjectInfo();
        Console.WriteLine($"Project Name: {projectInfo.Item1}");
        Console.WriteLine($"Number of Developers: {projectInfo.Item2}");
        Console.WriteLine($"Current Version: {projectInfo.Item3}");
        Console.WriteLine($"Estimated Completion: {projectInfo.Item4.ToShortDateString()}");
    }

    private static (string, int, string, DateTime) GetProjectInfo()
    {
        // Example data
        string projectName = "AlphaProject";
        int devCount = 5;
        string version = "1.2.3";
        DateTime estimatedCompletion = new DateTime(2024, 12, 31);

        return (projectName, devCount, version, estimatedCompletion);
    }
}
```

#24 : DataAnnotations

```
using System.ComponentModel.DataAnnotations;

public class User
{
    [Required(ErrorMessage = "Name is required.")]
    [StringLength(50, ErrorMessage = "Name cannot be longer than 50 characters.")]
    public string Name { get; set; }

    [Required(ErrorMessage = "Email is required.")]
    [EmailAddress(ErrorMessage = "Invalid Email Address.")]
    public string Email { get; set; }

    [Range(18, 60, ErrorMessage = "Age must be between 18 and 60.")]
    public int Age { get; set; }

    // Additional properties and methods...
}
```

#24 : DataAnnotations

```
[HttpPost]
public ActionResult CreateUser(User user)
{
    if (ModelState.IsValid)
    {
        // Proceed with saving user, etc.
        return RedirectToAction("Success");
    }

    // Validation failed, return to form
    return View(user);
}
```

#24 : DataAnnotations

```
var validationResults = new List<ValidationResult>();
var context = new ValidationContext(person, serviceProvider: null, items: null);
bool isValid = Validator.TryValidateObject(person, context, validationResults, true);

if (!isValid)
{
    foreach (var validationResult in validationResults)
    {
        Console.WriteLine(validationResult.ErrorMessage);
    }
}
```

#25 : Custom Validations

```
public class ValidReleaseYearAttribute : ValidationAttribute
{
    protected override ValidationResult IsValid(object value, ValidationContext validationContext)
    {
        if (value is int year)
        {
            if (year >= 1900 && year <= DateTime.Now.Year)
            {
                return ValidationResult.Success;
            }
            else
            {
                return new ValidationResult($"Release year must be between 1900 and {DateTime.Now.Year}.");
            }
        }
        return new ValidationResult("Invalid year format.");
    }
}
```

#25 : Custom Validations

```
public class Movie
{
    public int Id { get; set; }

    [Required]
    [StringLength(100)]
    public string Title { get; set; }

    [ValidReleaseYear]
    public int ReleaseYear { get; set; }
}
```

#26 : Span<T>

```
int[] numbers = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 };

// Create a span over a portion of the array.
Span<int> span = new Span<int>(numbers, 2, 5); // Span over elements 3 to 7 (inclusive).

// Modify the span.
for (int i = 0; i < span.Length; i++)
{
    span[i] = span[i] * 2; // Double each element in the span.
}

// { 1, 2, 6, 8, 10, 12, 14, 8, 9, 10 }
```

#27 : yield return

```
// before
public List<int> GetNumbers(int max)
{
    List<int> numbers = new List<int>();
    for (int i = 1; i <= max; i++)
    {
        numbers.Add(i);
    }
    return numbers;
}

// after
public IEnumerable<int> GetNumbers(int max)
{
    for (int i = 1; i <= max; i++)
    {
        yield return i;
    }
}
```

#28 : IHostedService for Background Work

```
namespace Tactic.WindowsService;

public class CleanupFolderService(ILogger<CleanupFolderService> _logger) : BackgroundService
{
    private const string FolderToCheck = "C:\\Temp";
    private const int DelayBetweenChecks = 1000 * 15; // 15 seconds

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        while (!stoppingToken.IsCancellationRequested)
        {
            _logger.LogInformation("Checking folder {FolderToCheck}", FolderToCheck);

            // TODO: delete any files older than 5 seconds in FolderToCheck

            await Task.Delay(DelayBetweenChecks, stoppingToken);
        }

        _logger.LogInformation("Shutting down worker");
    }
}
```

#28 : IHostedService for Background Work

```
IHost host = Host.CreateDefaultBuilder(args)
    .ConfigureServices(services =>
    {
        services.AddHostedService<CleanupFolderService>();
    })
    .Build();

host.Run();
```

#29 : IHostApplicationLifetime

```
public class StatusGeneratorWorker(ILogger<StatusGeneratorWorker> logger, Engine engine, IHostApplicationLifetime lifetime)
    : BackgroundService
{
    private readonly TimeSpan _interval = TimeSpan.FromSeconds(5);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        var validateConnection = await engine.ValidateConnection();
        if (!validateConnection.IsSuccess)
        {
            logger.LogError("Engine is reporting it's unable to connect to database: {message}", validateConnection.Message);
            lifetime.StopApplication(); // This line triggers the shutdown of the application
            return;
        }

        // TODO: other stuff
    }
}
```

#30 : PeriodicTimer

```
public class StatusGeneratorWorker(ILogger<StatusGeneratorWorker> logger, Engine engine, IHostApplicationLifetime lifetime)
    : BackgroundService
{
    private readonly TimeSpan _interval = TimeSpan.FromSeconds(5);

    protected override async Task ExecuteAsync(CancellationToken stoppingToken)
    {
        // OTHER CODE...

        var periodicTimer = new PeriodicTimer(_interval);

        do
        {
            try
            {
                logger.LogInformation("Generating status");
                await engine.CreateStatusUpdateAsync();
            }
            catch (Exception e)
            {
                logger.LogError(e, "Error generating status");
            }

        } while (!stoppingToken.IsCancellationRequested && await periodicTimer.WaitForNextTickAsync(stoppingToken));
    }
}
```

Thanks!