.NET Core 3.0 and C# 8.0

.NET CORE 3.0

- 1. Support WPF
- 2. Support Windows Forms
- 3. Entity Framework 6
- 4. Client-side development with Razor components
- 5. Utf8JsonReader & JsonDocument & Utf8JsonWriter
- 6. EF Core 3.0:
- -EF Core 3 support Azure Cosmos DB;
- -LINQ improvements;
- 7. Event Pipe improvements
- 8. Performance improvements

New JSON API

- Provide high performance
- Remove Json.NET dependency from Asp.net core
- Provider an Asp.net core integration package for Json.NET

Performance new Json Api

Scenario	Speed	Memory
Deserialization	2x faster	Parity or lower
Serialization	1.5x faster	Parity or lower
Document (read-only)	3-5x faster	~Allocation free for sizes < 1 MB
Reader	2-3x faster	~Allocation free (until you materialize values)
Writer	1.3-1.6x faster	~Allocation free

Performance Improvements in .NET Core 3.0

- Span and friends
- Arrays and strings
- Parsing/Formatting
- Regular expressions
- Threading
- Collections
- Networking
- System.IO
- System.Diagnostics.Process
- LINQ
- GC
- JIT

Default interface implementations

- This feature allows you to add a default interface implementation. Therefore, when some class will implement this interface, the implementation of the interface will be optional

```
interface ILogger
{
    void Debug(string message) =>
                Debug.WriteLine(message);
    void Info(string message);
    void Error(string message);
}
```

Pattern matching

- It allows you to deconstruct matching objects, providing access to their data structures :

  - Property expressions;

  - Tuple patterns;

  - Positional patterns;

Tuple and positional patterns

- Tuple patterns allow matching of more than one value in a single pattern matching expression:

```
static string ShowTuplePatterns(DateTime dt) => dt switch
{
    (3, 9, 1996)                => $"My birthday {dt:d}",
    (_, 5, 2014)                => $"SBTech Ukraine birthday {dt:d}",
    (_, 7, 2018)                => $"Start work at SBTech {dt:d}",
    var (_, _, z) when z > 2019 =>  "Future date",
    _                           => $"Today is {dt:d}"
};
```

Property patterns

- The property pattern enables you to match on properties of the object examined

```
public static string Display(object o) => o switch
 {
     Point { X: 0, Y: 0 } p => "origin",
     Point { X: var x, Y: var y } p => $"({x}, {y})",
     _ => "unknown"
 };
```

Indices and Ranges

This function simplifies the syntax for specifying subranges in an array or collection

- System.Index represents an index into a sequence.
- The ^ operator, which specifies that an index is relative to the end of a sequence.
- System.Range represents a sub range of a sequence.
- The Range operator (..), which specifies the start and end of a range as its operands.

Nullable reference types

```
    string text = null; //warning: Converting null literall or possible null value to non-nullable type

    Console.WriteLine(text.Length); //warning: Dereference of a possible null reference
```

Asynchronous stream

- Enumerators which allows support async operations

```
static async Task Main(string[] args)
{
    await foreach(int number in GetAsyncEnumerable())
        Console.WriteLine(number);
}
static async IAsyncEnumerable<int> GetAsyncEnumerable()
{
    for (int i = 0; i <= 10; i++)
     {
        await Task.Delay(1000);
        yield return i;
     }
}
```

Static local functions

```
static void Main(string[] args)
{
     WriteLine(123);
     static void WriteLine<T>(T item) =>
                 Console.WriteLine(item?.ToString());
}
```

Using declarations

- Simplifies the use of the 'using' operator

```
using var writer = new StreamWriter("c:\\some_file.txt");
```

Disposable ref structs

Allows use 'using' pattern with ref struct and readonly ref struct

```
static void Main(string[] args) {
    using var spanList = new SpanList<string>();
}
ref struct SpanList<T>
{
    public void Dispose() => Console.WriteLine($"Dispose
             span list of {typeof(T)}");
}
```