30+ Tips for .NET Developers

Episode 1 : What is .AsNoTracking() and its benefits

While using AsNoTracking

1. The entity is not tracked by the context.

2. EF does not know the state of its entity.

3 Not recommended when you are performing Add/Update/Delete kind operations

4. Improved performance over regular LINQ queries.

5. Only recommended when you are doing read-only operations.

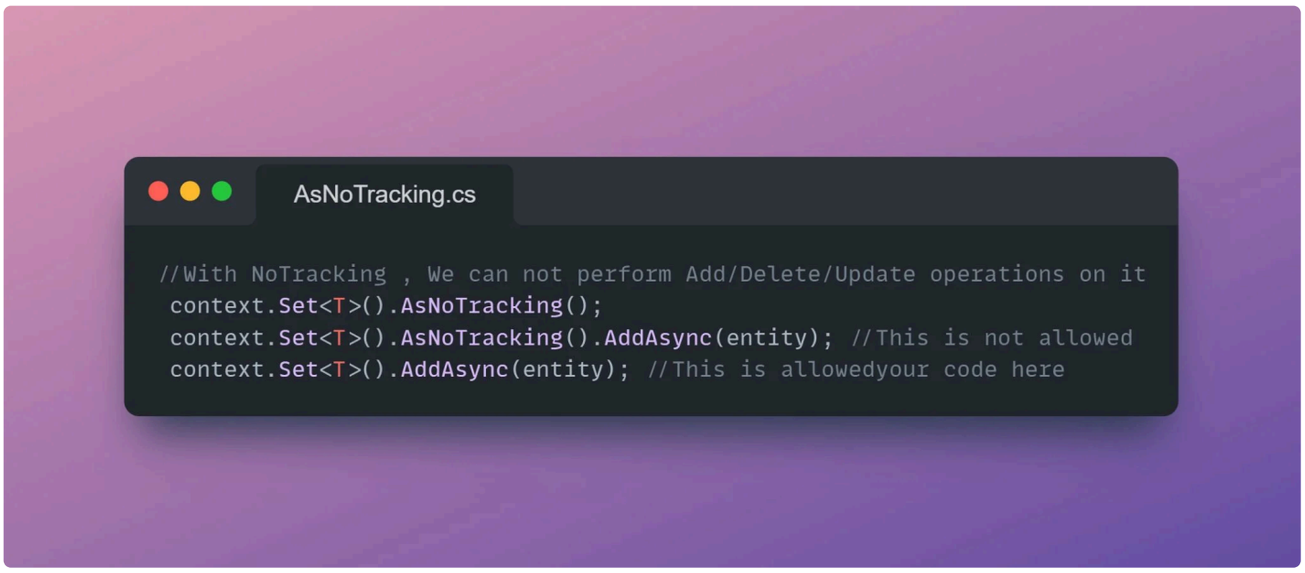
6. Most efficient when we have to retrieve large set of data.

Without AsNoTracking

1. The entity is tracked by the context.

2. EF knows the state of this entity.

3. We can use this entity to save/update and we don't need to set the state of entity again.



Episode 2 : SingleAsync and FirstAsync Methods of LINQ in .NET

1. In .NET, SingleAsync and FirstAsync are methods that can be used to retrieve a single element from a collection of elements.

2. Both methods are similar in that they return the first element in a collection that satisfies a specified condition.
3. The main difference between SingleAsync and FirstAsync is that SingleAsync will throw an exception if there is more than one element in the collection that satisfies the specified condition, while FirstAsync will return the first element it finds and then stop.
4. SingleAsync can be used to ensure that there is exactly one element in the collection that satisfies the condition, while FirstAsync can be used to simply retrieve the first element that satisfies the condition, regardless of how many elements in the collection satisfy the condition.

```
using System.Linq;

// Get the first element that has a value of 5
var result1 = await collection.FirstAsync(x => x.Value == 5);

// Get the only element that has a value of 5, or throw an exception
// if there is more than one element with a value of 5
var result2 = await collection.SingleAsync(x => x.Value == 5);
```

Episode 3 : Basic overview of Monolithic and Microservices applications

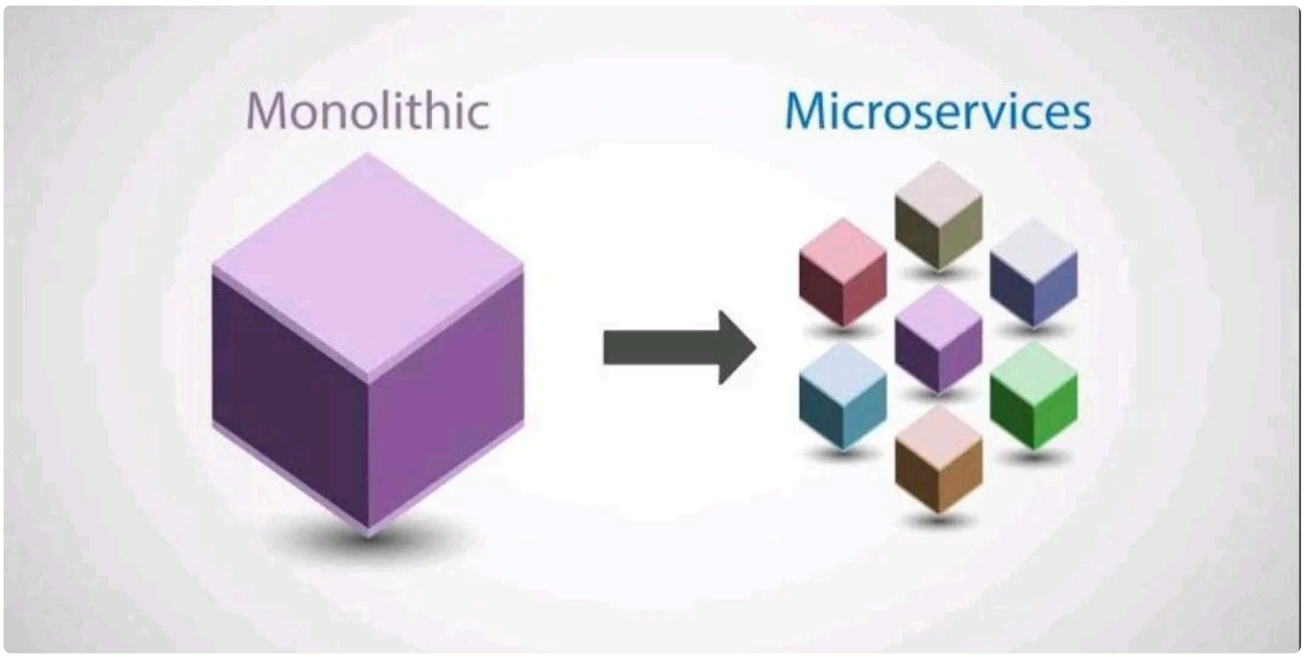
Lets see the difference b/w monolithic and microservices

Monolithic:

1. A monolithic application is a single, self-contained program that consists of a single codebase.
2. It is typically built and deployed as a single unit
3. It tends to be more tightly coupled
4. It can be more brittle, since a failure in one part of the application can affect the entire system

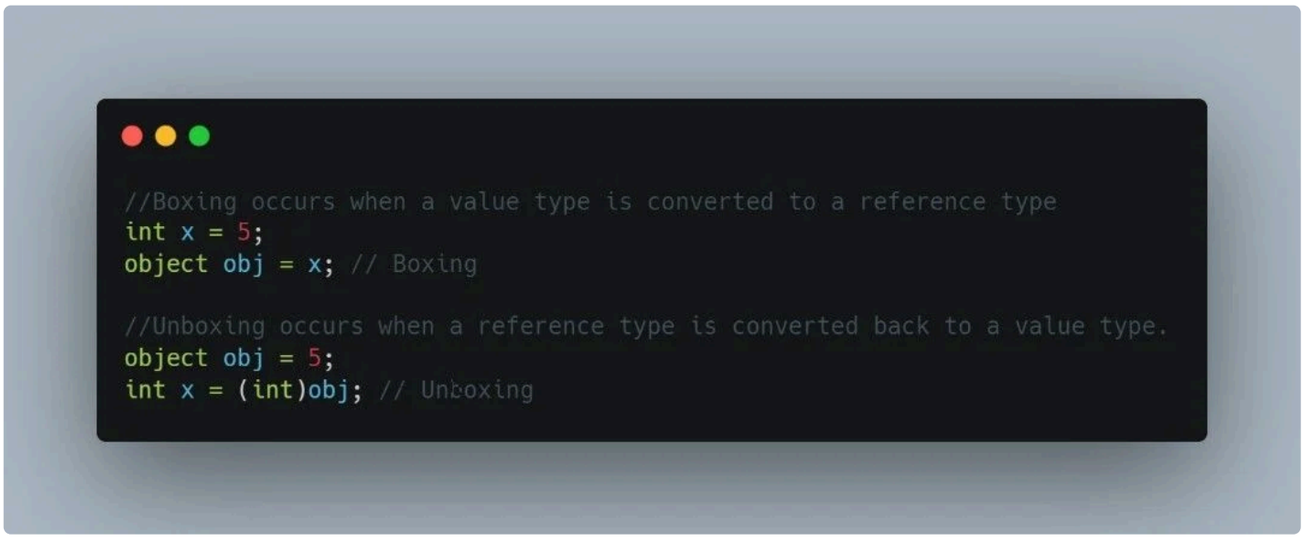
Microservices:

1. A microservices application is a collection of small, independent services that communicate with each other to form a complete application.
2. It is built as a set of independent services that are deployed and managed separately
3. It is flexible and scalable as individual services can be updated and deployed without affecting the entire application.
4. It tends to be loosely coupled.
5. These applications can be more complex to build and manage, since they require coordination among multiple services.
6. They can also be more expensive to run, since each service needs to be individually managed and scaled.



Episode 4 : Difference b/w Boxing and Unboxing in C#

Boxing and **Unboxing** are used to convert value types to reference types and vice versa. When value type is moved to a reference type it's called as **Boxing** . The vice-versa is termed as **Unboxing**.



```
//Boxing occurs when a value type is converted to a reference type
int x = 5;
object obj = x; // Boxing

//Unboxing occurs when a reference type is converted back to a value type.
object obj = 5;
int x = (int)obj; // Unboxing
```

Episode 5 : Benefit of using AsReadOnly Method of List in .NET

Here are some benefits of **AsReadOnly** Method of **List<T>**

1. It gives you read only view of your collection.
2. It allows you to prevent the collection from being modified, either accidentally or intentionally
3. It can improve the performance of your code in some cases because the read-only wrapper provides a more restricted view of the collection.
4. This can be particularly beneficial when working with large collections, where even small performance improvements can make a significant difference.

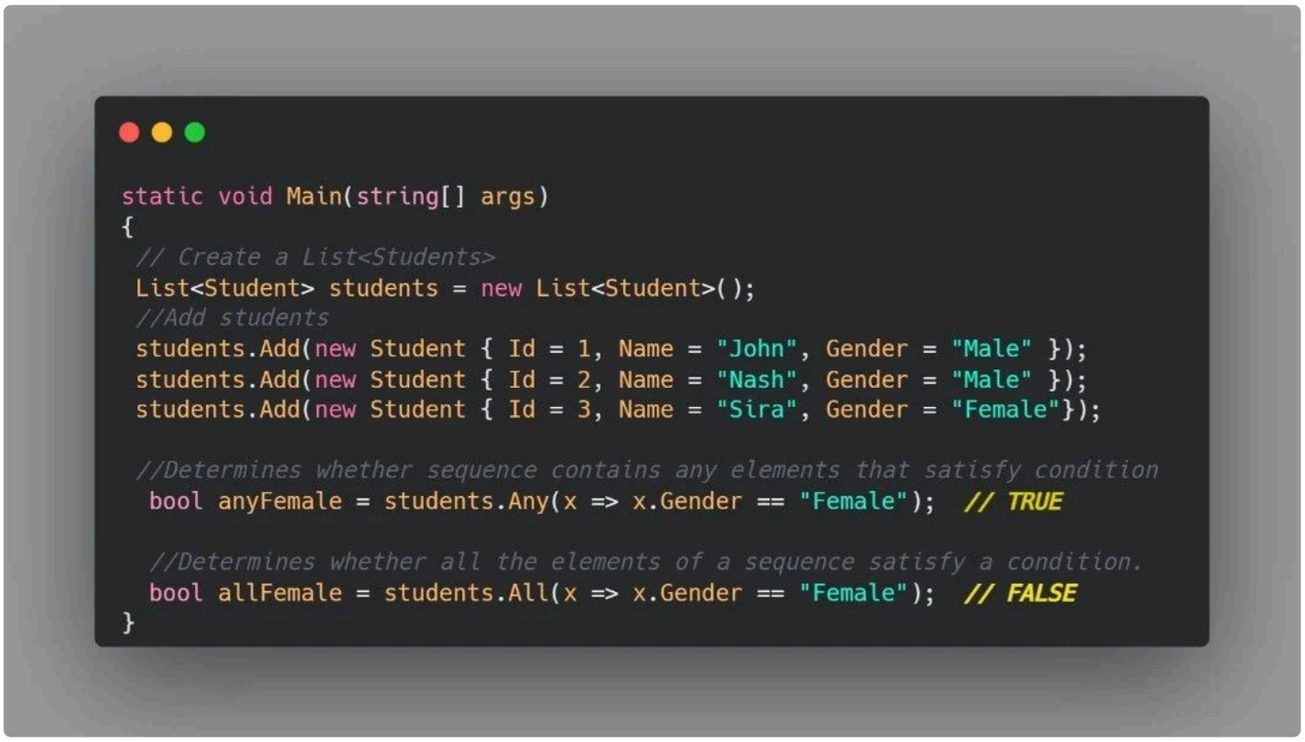
```
static void Main(string[] args)
{
    // Create a List<string>
    List<string> developers = new List<string>() { "Waseem", "Usman"};

    // Create a read-only wrapper for the List<string>
    IReadOnlyList<string> readOnlyDevelopers = developers.AsReadOnly();

    //Add operations is allowed on List
    developers.Add("Milan");

    //Add operation is not allowed on readonly List
     readOnlyDevelopers.Add("Stefan"); //ERROR
}
```

Episode 6 : Difference b/w Any and All Method for Collection in .NET

Any vs All Extension Method of List<T>

1.The **All** method checks if all elements of a list satisfy a specified condition.

2.The **Any** method checks if any elements of a list satisfy a specified condition.



```
static void Main(string[] args)
{
 // Create a List<Students>
 List<Student> students = new List<Student>();
 //Add students
 students.Add(new Student { Id = 1, Name = "John", Gender = "Male" });
 students.Add(new Student { Id = 2, Name = "Nash", Gender = "Male" });
 students.Add(new Student { Id = 3, Name = "Sira", Gender = "Female"});

 //Determines whether sequence contains any elements that satisfy condition
  bool anyFemale = students.Any(x => x.Gender == "Female");  // TRUE

  //Determines whether all the elements of a sequence satisfy a condition.
  bool allFemale = students.All(x => x.Gender == "Female");  // FALSE
}
```

Episode 7 : Lazy Loading vs Eager Loading in EntityFramework

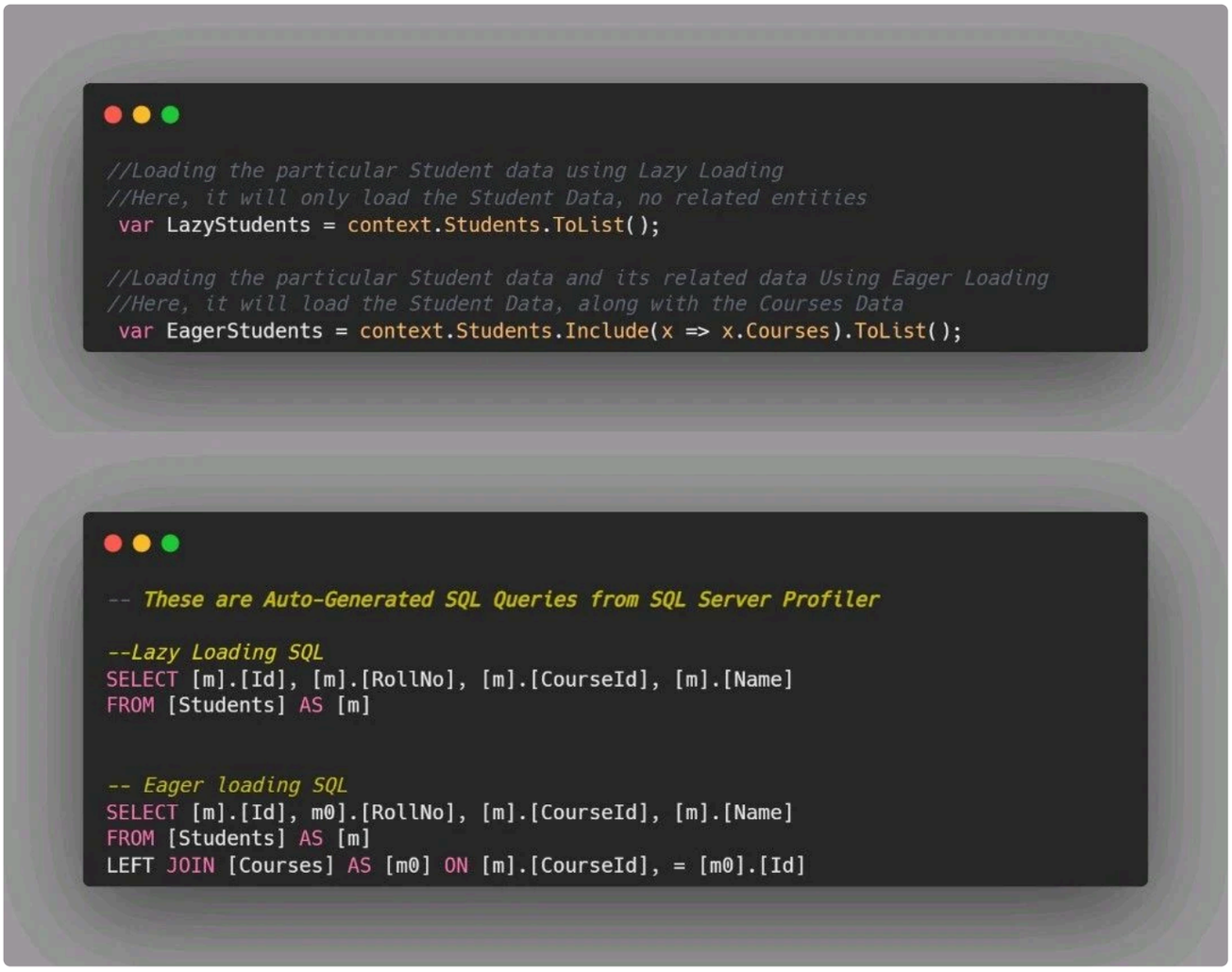
Lazy Loading (LL)

1. Lazy Loading is a process where EF loads the related entities on demand.
2. It is the default behavior of EF
3. It delays the loading of related entities until you specifically request it.
4. You can go with it when you are sure that you are not using the related entities Instantly.
5. The number of round trips to the database is more as for each master entity data, it will issue a separate SQL query to get the child-related entity data.
6. It simply uses the SELECT Statement without any join.
7. If you are not interested in related entities or the related entities are not used instantly, then you can use it.

Eager Loading (EL)

1. Eager loading is a Process where EF loads the related entities along with the main entity
2. EF will not execute separate SQL queries for loading the related entities.
3. All the entities are loaded from the database with a single query saving bandwidth and server CPU time.
4. You can go with EL when you are sure that you will be using the related entities with the main entity everywhere
5. It is a good practice to reduce the number of SQL queries to be sent to the database server to fetch the related entities
6. It will use SQL Joining to join the related tables with the main table and then return the Main entity data along with the related entities.
7. If you are interested in related entities used instantly in your application, then you need to go with it.

Note : If you want to check the **SQL Generated Query** when a LINQ query executes then you can check it by clicking on **Tools → SQL Server Profiler** in SQL Server Management Studio.



```
//Loading the particular Student data using Lazy Loading
//Here, it will only load the Student Data, no related entities
 var LazyStudents = context.Students.ToList();

//Loading the particular Student data and its related data Using Eager Loading
//Here, it will load the Student Data, along with the Courses Data
 var EagerStudents = context.Students.Include(x => x.Courses).ToList();
```

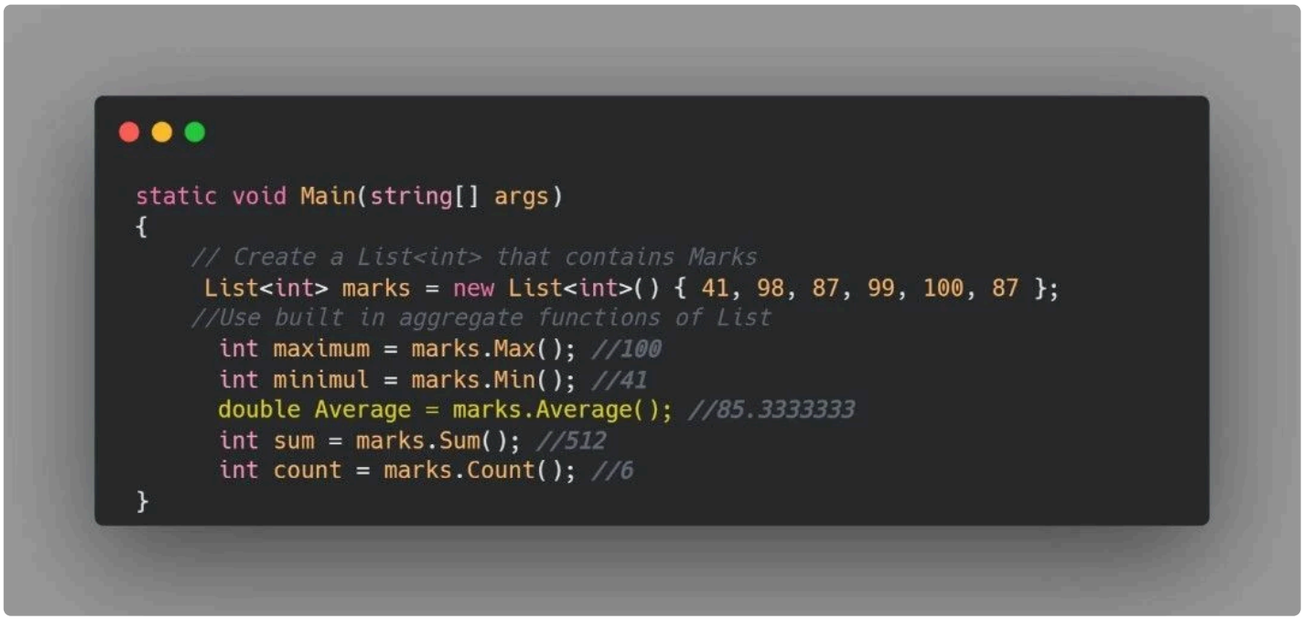
```
-- These are Auto-Generated SQL Queries from SQL Server Profiler

--Lazy Loading SQL
SELECT [m].[Id], [m].[RollNo], [m].[CourseId], [m].[Name]
FROM [Students] AS [m]


-- Eager loading SQL
SELECT [m].[Id], m0].[RollNo], [m].[CourseId], [m].[Name]
FROM [Students] AS [m]
LEFT JOIN [Courses] AS [m0] ON [m].[CourseId], = [m0].[Id]
```

Episode 8 : Aggregate function over List by Default Provided

You would be familiar with aggregate functions from SQL, let's see how to use Entity Framework Queryable Extension Methods for aggregate functions over the List

1. Sum
2. Average
3. Minimum
4. Maximum
5. Count

.NET 6.0 has 12 overloads of these all methods for all numeric data types used in code ⏬



```
static void Main(string[] args)
{
    // Create a List<int> that contains Marks
     List<int> marks = new List<int>() { 41, 98, 87, 99, 100, 87 };
    //Use built in aggregate functions of List
      int maximum = marks.Max(); //100
      int minimul = marks.Min(); //41
      double Average = marks.Average(); //85.3333333
      int sum = marks.Sum(); //512
      int count = marks.Count(); //6
}
```

Episode 9 : Difference b/w Include and ThenInclude in Entity Framework

Include and ThenInclude are two methods in EF that can be useful for improving the performance of a query by reducing the number of databases round trips.

Include is used for eager loading that's why related entities come in a single query and database round trips are reduced.

Main difference b/w them is level, include is used for **Single Level** travelling along entities and ThenInclude is helpful in **Multilevel Retrieval**.

```
using (var context = new MyDBContext())
{
 //Retrieve all customers and their related invoices we have to use the Include Method.
  var SingleLevelCustomers = context.Customers
    .Include(c => c.Invoices).ToList();

//Drill down thru relationships to include multiple levels of related data using the
ThenInclude method
  var MultiLevelCustomers = context.Customers
        .Include(i => i.Invoices).ThenInclude(it => it.Items)
        .ToList();

}
```

Episode 10 : Use ToQueryString() Extension method while debugging

ToQueryString is a custom extension method that converts IQueryable to SQL Query at the back-end side, especially helpful for debugging. ⏬

```
using (var context = new Context())
{
  //Generates a string representation of the query used.
  //This string may not be suitable for direct execution is intended only for use in debugging.
  string SqlQuery = context.Cities.ToQueryString();
}
```

```
-- SQL returned from ToQueryString()
SELECT [m].[Id], [m].[Description], [m].[DistrictId], [m].[Name]
FROM [City] AS [m]
```

Episode 11 : How to avoid DbContext threading issues in Entity Framework

When EF Core detects an attempt to use a DbContext instance concurrently, you'll see an InvalidOperationException with a message like this:

"A second operation started in this context before a previous operation was completed."

There are few ways that can help you to avoid threading issues in EF.

1. Use a separate DbContext instance for each thread. This ensures that each thread has its own DbContext instance, which means that there is no shared state between threads and no potential for threading conflicts.

Problem: Costly in terms of memory usage

2. Use a thread-safe DbContext wrapper. In this approach, you would create a wrapper class for DbContext that uses synchronization techniques, such as the lock keyword to ensure that only one thread can access the DbContext instance at a time.

Problem: It can impact performance because threads may have to wait for access to the DbContext instance.

3. Use **asynchronous** methods that allows you to write code that can run concurrently on multiple threads. It can help improve the performance of your application by allowing multiple operations to be executed concurrently. We can use **await** and **async** to perform asynchronous operation.

```
//We need to put await in asynchronous operations and async,Task with Method
public async Task DoSomething()
{
    using (var context  = new DbContext())
    {
        var results = await context.Entity<T>.FindAsync(Id);
    }
}
```

Episode 12 : How to register Open Generics in .NET Core Dependency Injection

If you have a generic interface and its generic implementation like we mostly do when we make a generic repository for CRUD operations and you want to register its dependency injection at startup, then there is a simple way of registering the DI.

1. For .NET 3.1 register in ConfigureServices in **Stratup.cs**

2. For latest versions of .NET(6.0 ,7.0) register in **Program.cs**

```
public interface IGenericRepository<T> where T : class
 {
     Task<T?> FindAsync(int id);
 }
```

```
  public class GenericRepository<T> : IGenericRepository<T> where T : class
    {
        protected readonly SomeContext context;

        public GenericRepository(SomeContext _context)
        {
            context = _context;
        }

        public async Task<T?> FindAsync(int id)
        {
            return await context.Set<T>().FindAsync(id);
        }

    }
```

```
//This will not work
services.AddScoped<IGenericRepository<T>,GenericRepository<T>>(); //ERROR

//THIS WILL WORK :)
services.AddScoped(typeof(IGenericRepository<>), typeof(GenericRepository<>));
```

Episode 13 : What are CORS and how to enable them in .NET at API Level

CORS stands for Cross Origin Resource sharing, so what exactly is cross origin.

These two URLs have the same origin: https://my-site-no-1.com/Get/HakunaMatata https://my-site-no-1.com/Get/AllIsWell

These URLs have different origins https://my-site-no-1.com/Get/HakunaMatata http://my-site-no-1.net/Get/AllIsWell

To facilitate requests from different origins you need to enable CORS in .NET.

In .NET 6 by using the combination of these methods you can enable CORS as per your requirement.

AllowAnyOrigin: This policy allows requests from any origin.

WithOrigins: This policy allows requests from specific origins. You can specify one or more origins as arguments to this method.

AllowAnyHeader: This policy allows requests with any header.

WithHeaders: This policy allows requests with specific headers. You can specify one or more headers as arguments to this method.

AllowAnyMethod: This policy allows requests with any HTTP method (e.g., GET, POST, PUT, DELETE).

WithMethods: This policy allows requests with specific HTTP methods. You can specify one or more methods as arguments to this method.

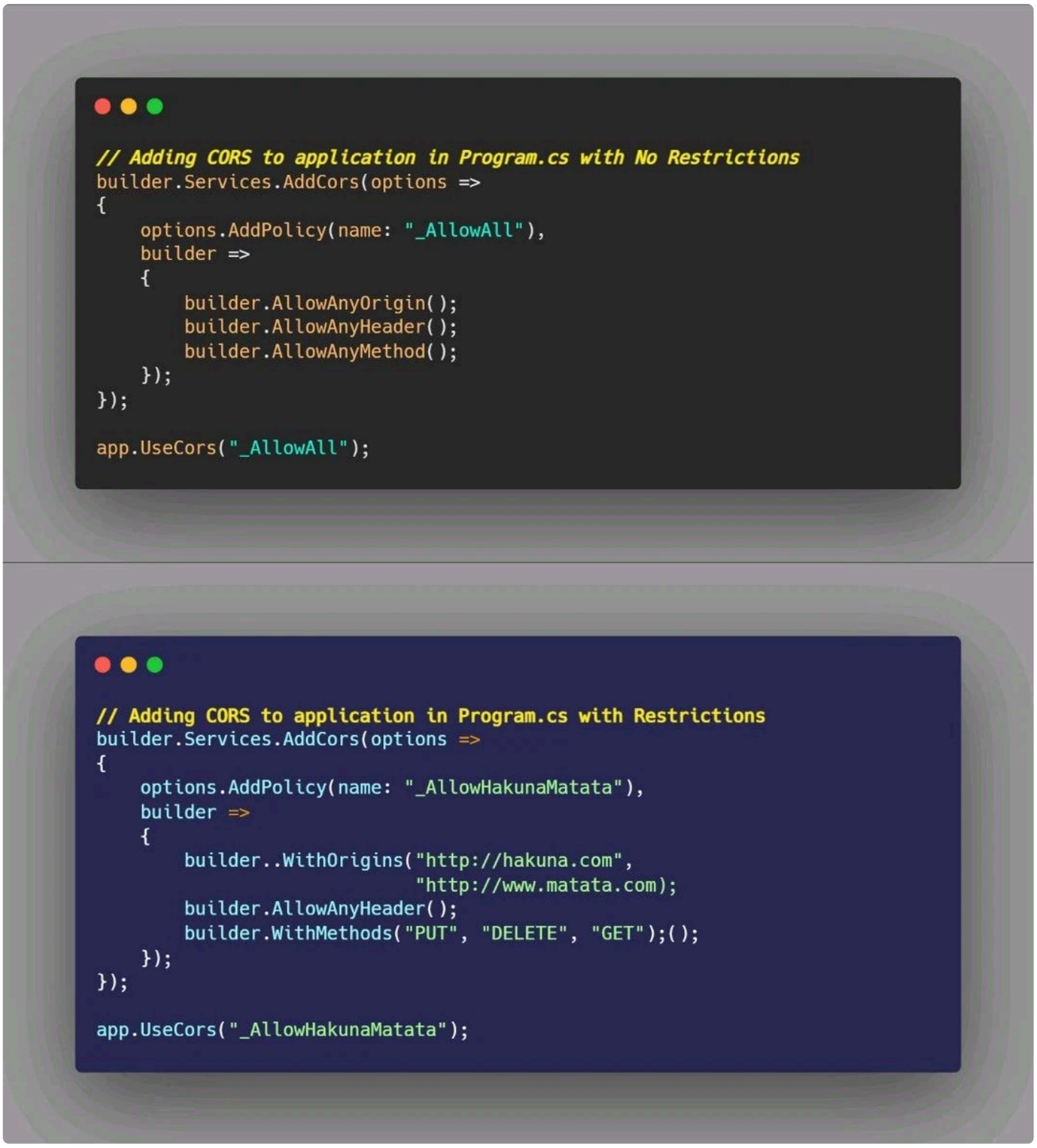
Few Things to Keep in mind

✔️ CORS is not a security feature. CORS is a W3C standard that allows a server to relax the same-origin policy.

✔️ An API isn't safer by allowing CORS.

✔️ It's a way for a server to allow browsers to execute a cross-origin request that otherwise would be forbidden.

✔️ Browsers without CORS can't do cross-origin requests.



```
// Adding CORS to application in Program.cs with No Restrictions
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "_AllowAll"),
    builder =>
    {
        builder.AllowAnyOrigin();
        builder.AllowAnyHeader();
        builder.AllowAnyMethod();
    });
});

app.UseCors("_AllowAll");
```

```
// Adding CORS to application in Program.cs with Restrictions
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "_AllowHakunaMatata"),
    builder =>
    {
        builder..WithOrigins("http://hakuna.com",
                             "http://www.matata.com);
        builder.AllowAnyHeader();
        builder.WithMethods("PUT", "DELETE", "GET");();
    });
});

app.UseCors("_AllowHakunaMatata");
```

Episode 14 : Common Middlewares in .NET API

Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:

Chooses whether to pass the request to the next component in the pipeline.

Can perform work before and after the next component in the pipeline.

Here are some common types of middleware that might be used in a .NET API program:

1. Routing

2. Exception handling

3. Authentication and authorization
4. CORS (Cross-Origin Resource Sharing)
5. Response compression
6. Request validation
7. Response caching 8.Static file serving

Routing middleware: This middleware is responsible for determining which endpoint should handle a particular request based on the request's path and method.

Exception handling middleware: This middleware is responsible for catching and handling exceptions that occur during the processing of a request.

Authentication and authorization middleware: This middleware is responsible for verifying that a request is from an authenticated and authorized user.

CORS (Cross-Origin Resource Sharing) middleware: This middleware is responsible for adding the necessary headers to allow a browser to make cross-origin requests to the API.

Response compression middleware: This middleware is responsible for compressing the response payload in order to reduce the size of the response and improve performance.

Request validation middleware: This middleware is responsible for validating incoming requests to ensure that they conform to the expected format.

Response caching middleware: This middleware is responsible for caching responses in order to reduce the load on the server and improve performance.

Static file serving middleware: This middleware is responsible for serving static files, such as HTML, CSS, and JavaScript files, from the file system.

It's important to note that the order in which middleware is added to the pipeline can be important, as the middleware will be executed in the order in which it is added. For example, if the authentication middleware is added before the routing middleware, the routing middleware will not be executed until the authentication middleware has completed.

Episode 15 : Response Compression in .NET Core and how to configure its middleware

Network bandwidth is a limited resource. Reducing the size of the response usually increases the responsiveness of an app, often dramatically. One way to reduce payload sizes is to compress an app's responses.

What is Response Compression Response compression is a technique that can be used to reduce the size of HTTP responses, which can improve the performance of a web application by reducing the amount of data that needs to be transmitted over the network.

Benefits of Response Compression

1.Improved performance: Compressing the response can reduce the amount of data that needs to be transmitted over the network, which can lead to faster page load times and a better user experience.

2.Reduced bandwidth usage: By compressing the response, you can reduce the amount of data that is transmitted over the network, which can lead to reduced bandwidth usage and lower costs for hosting and bandwidth.

3.Better SEO: Search engines take page load times into account when ranking websites, so a faster loading website may rank higher in search results.

How to configure Response Compression Middleware in .NET We can configure it using the .NET middleware, AddResponseCompression.

.NET also Provides built in providers for compression we can configure their options as per our need.

1. BrotliCompressionProvider Using it a text file response at 2,044 bytes was compressed to ~979 bytes.
2. GzipCompressionProvider Using it a Scalable Vector Graphics (SVG) image response at 9,707 bytes was compresses to ~4,459 bytes

Sample Application: https://bit.ly/3G3rslj

Compression Levels

Optimal - The compression operation should be optimally compressed, even if the operation, it takes a longer time to complete.

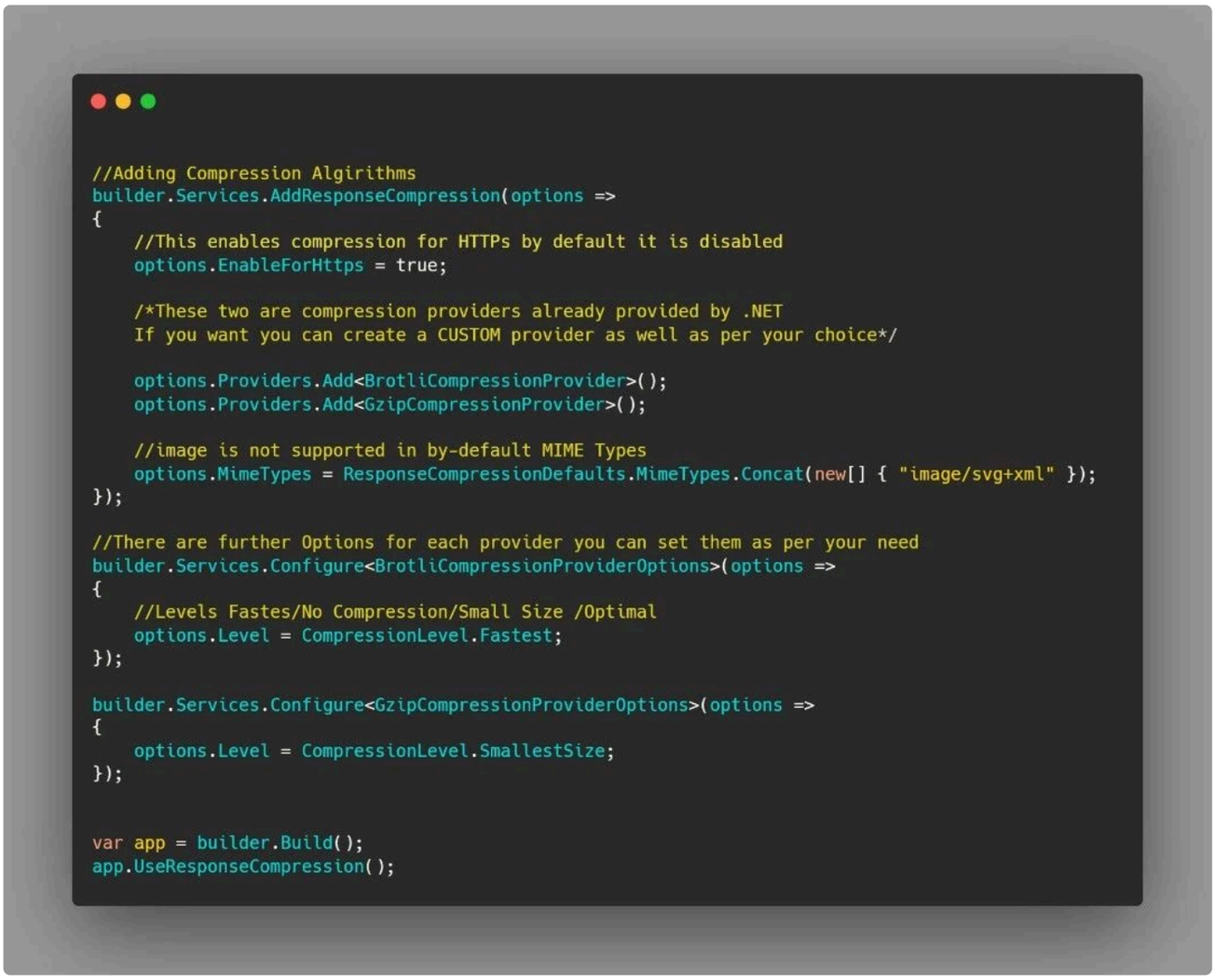
Fastest - The compression operation should complete as quickly as possible, even if the resulting file is not optimally compressed.

No Compression - No compression should be performed on the file.

Smallest Size - The compression operation should create output as small as possible, even if the operation takes a longer time to complete.

MIME Types by Default Supported By .NET Providers

1. text/plain
2. text/css
3. application/javascript
4. text/html
5. application/xml
6. text/xml
7. application/json
8. text/json
9. application/wasm



```
//Adding Compression Algirithms
builder.Services.AddResponseCompression(options =>
{
    //This enables compression for HTTPs by default it is disabled
    options.EnableForHttps = true;

    /*These two are compression providers already provided by .NET
    If you want you can create a CUSTOM provider as well as per your choice*/

    options.Providers.Add<BrotliCompressionProvider>();
    options.Providers.Add<GzipCompressionProvider>();

    //image is not supported in by-default MIME Types
    options.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(new[] { "image/svg+xml" });
});

//There are further Options for each provider you can set them as per your need
builder.Services.Configure<BrotliCompressionProviderOptions>(options =>
{
    //Levels Fastes/No Compression/Small Size /Optimal
    options.Level = CompressionLevel.Fastest;
});

builder.Services.Configure<GzipCompressionProviderOptions>(options =>
{
    options.Level = CompressionLevel.SmallestSize;
});


var app = builder.Build();
app.UseResponseCompression();
```

Episode 16 : Count() vs TryGetNonEnumeratedCount() and Which one is better ?

Count()

This method will make an enumeration to calculate the count of elements.

Available in all versions of .NET

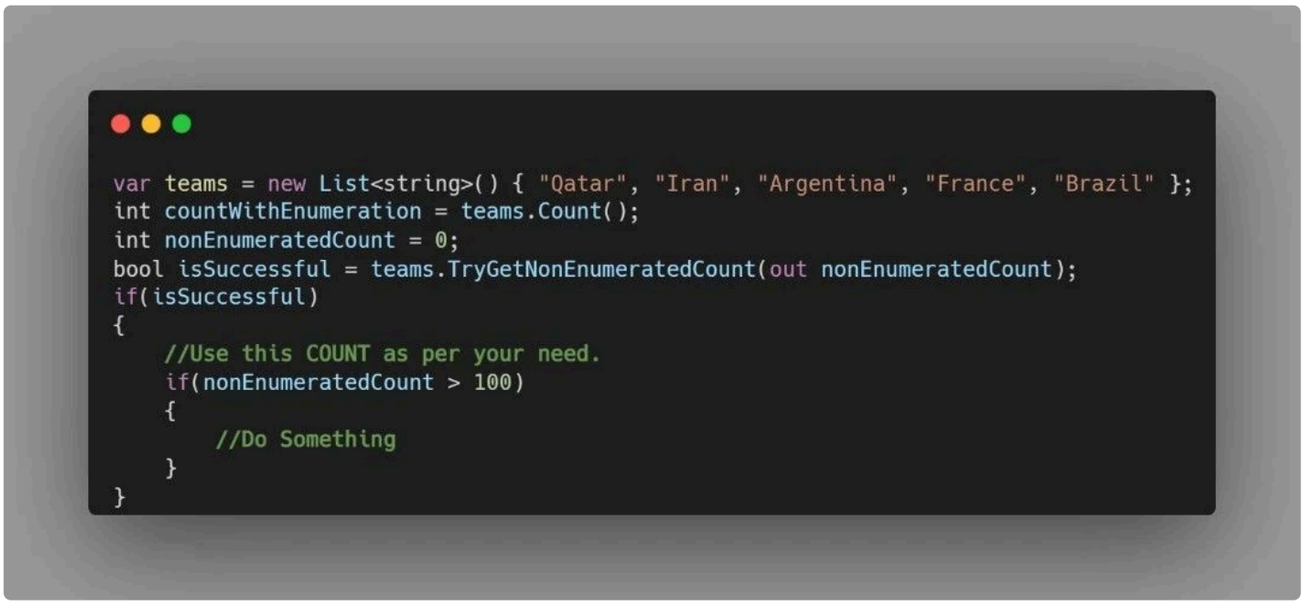
TryGetNonEnumeratedCount()

This attempts to determine the number of elements in a sequence without forcing an enumeration.

It is available in .NET 6 and .NET 7

It is available only on the ICollection interface.

It is typically a constant-time operation, but ultimately this depends on the complexity characteristics of the underlying collection's implementation.



```
var teams = new List<string>() { "Qatar", "Iran", "Argentina", "France", "Brazil" };
int countWithEnumeration = teams.Count();
int nonEnumeratedCount = 0;
bool isSuccessful = teams.TryGetNonEnumeratedCount(out nonEnumeratedCount);
if(isSuccessful)
{
    //Use this COUNT as per your need.
    if(nonEnumeratedCount > 100)
    {
        //Do Something
    }
}
```

Episode 17 : Everything about Rate Limiting in .NET

Rate limiting is a technique used to control the amount of incoming and outgoing traffic to a network or service. It is often used to protect servers and other resources from being overwhelmed by too many requests, or to prevent abuses such as distributed denial of service (DDoS) attacks.

How it works? Rate limiting works by setting a limit on the number of requests that a client can make to a server within a specified time period. If the client exceeds the rate limit, the server will return an error, typically an HTTP status code 429 (Too Many Requests), to the client.

Benefits of Rate Limiting?

1. Protecting against denial-of-service attacks of specific types.
2. Maintaining service availability.
3. Reducing resource consumption.
4. Detecting & blocking maliciousbehavior.
5. Improving user experience.

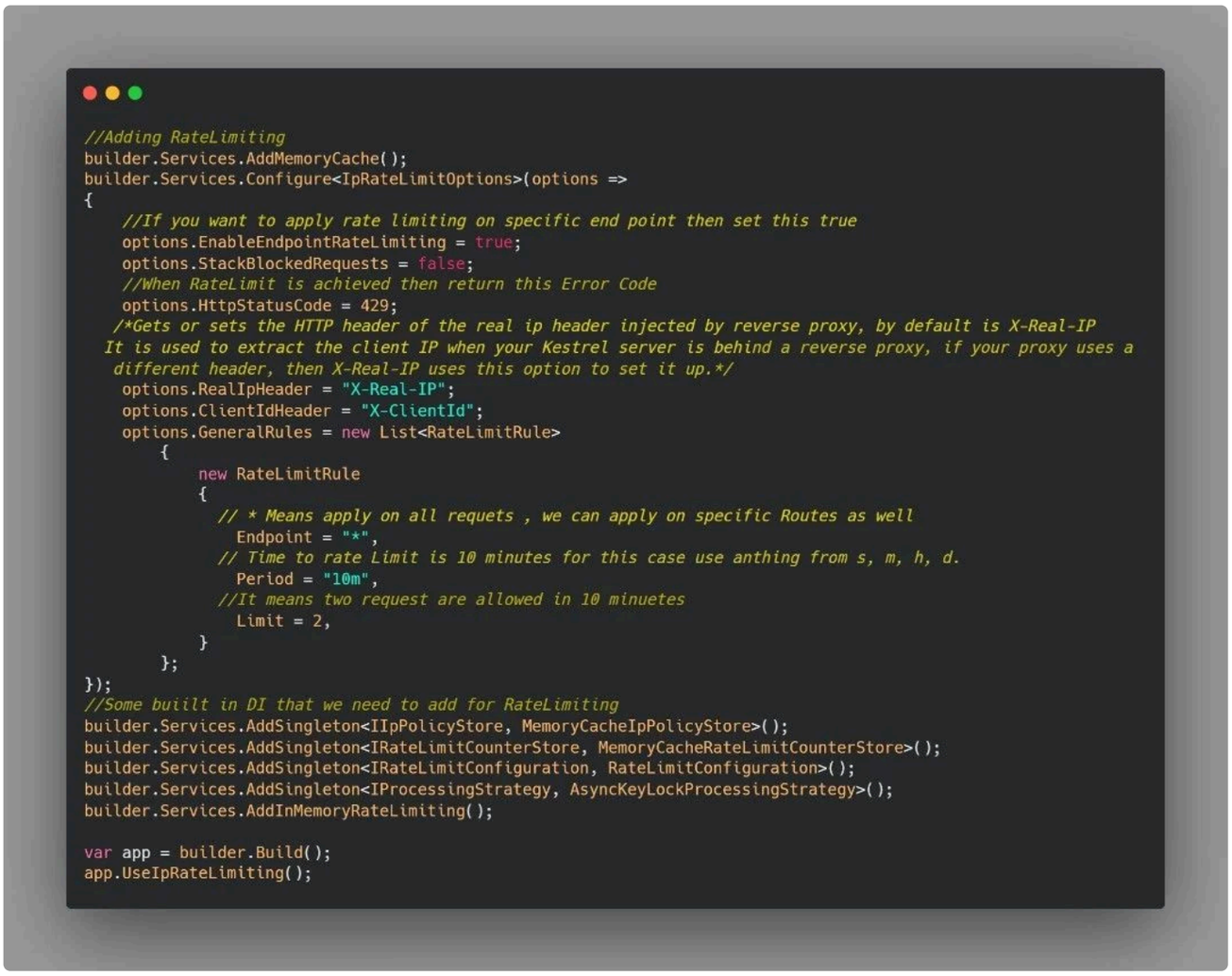
Cons of Rate Limiting? Rate limiting cannot distinguish between good and bad traffic, it will just look into IP and number of requests, so in some cases by changing the IP address attack is still possible.

At which point we can apply Rate Limiting?

1. Network Edge
2. Application Layer
3. Database Layer
4. Service Level

How can I verify that Rate Limiting has been added? You can check from response headers of your request there would be complete information that is your remaining limit, what is limit time etc. if your Rate Limiting has successfully configured.

How to add Rate Limiting in .NET Application? We can apply rate limiting on Application Layer in our project using Asp Net Core Rate Limit NuGet Package ⏬



```
//Adding RateLimiting
builder.Services.AddMemoryCache();
builder.Services.Configure<IpRateLimitOptions>(options =>
{
    //If you want to apply rate limiting on specific end point then set this true
    options.EnableEndpointRateLimiting = true;
    options.StackBlockedRequests = false;
    //When RateLimit is achieved then return this Error Code
    options.HttpStatusCode = 429;
   /*Gets or sets the HTTP header of the real ip header injected by reverse proxy, by default is X-Real-IP
  It is used to extract the client IP when your Kestrel server is behind a reverse proxy, if your proxy uses a
   different header, then X-Real-IP uses this option to set it up.*/
    options.RealIpHeader = "X-Real-IP";
    options.ClientIdHeader = "X-ClientId";
    options.GeneralRules = new List<RateLimitRule>
        {
            new RateLimitRule
            {
              // * Means apply on all requets , we can apply on specific Routes as well
                Endpoint = "*",
              // Time to rate Limit is 10 minutes for this case use anthing from s, m, h, d.
                Period = "10m",
              //It means two request are allowed in 10 minuetes
                Limit = 2,
            }
        };
});
//Some built in DI that we need to add for RateLimiting
builder.Services.AddSingleton<IIpPolicyStore, MemoryCacheIpPolicyStore>();
builder.Services.AddSingleton<IRateLimitCounterStore, MemoryCacheRateLimitCounterStore>();
builder.Services.AddSingleton<IRateLimitConfiguration, RateLimitConfiguration>();
builder.Services.AddSingleton<IProcessingStrategy, AsyncKeyLockProcessingStrategy>();
builder.Services.AddInMemoryRateLimiting();

var app = builder.Build();
app.UseIpRateLimiting();
```

Episode 18 : A basic visit to Response Caching along with its implementation in .NET

Response caching is a technique for storing the responses of an API or web application in a cache so that they can be served faster to subsequent requests. The responses are stored in a cache with a key that uniquely identifies them, and the cache has a limited size and a policy for removing items when it becomes full.

Benefits of Response Caching?

1. Improved performance by reducing the load on the server.
2. Reduced server load, as it can serve the cached response instead of generating a new one.
3. Reduced bandwidth usages it reduces the amount of data that needs to be transferred between the server and the client.

4. Improved security as it can reduce the number of requests that reach the server, reducing the risk of certain types of attacks.

On which requests we can apply Response Caching

1. Get
2. Head

Few constraints for Response Caching 1.The request must result in a server response with a 200 (OK) status code.

2.Response Caching Middleware must be placed before middleware that require caching. For more information, see ASP.NET Core Middleware.

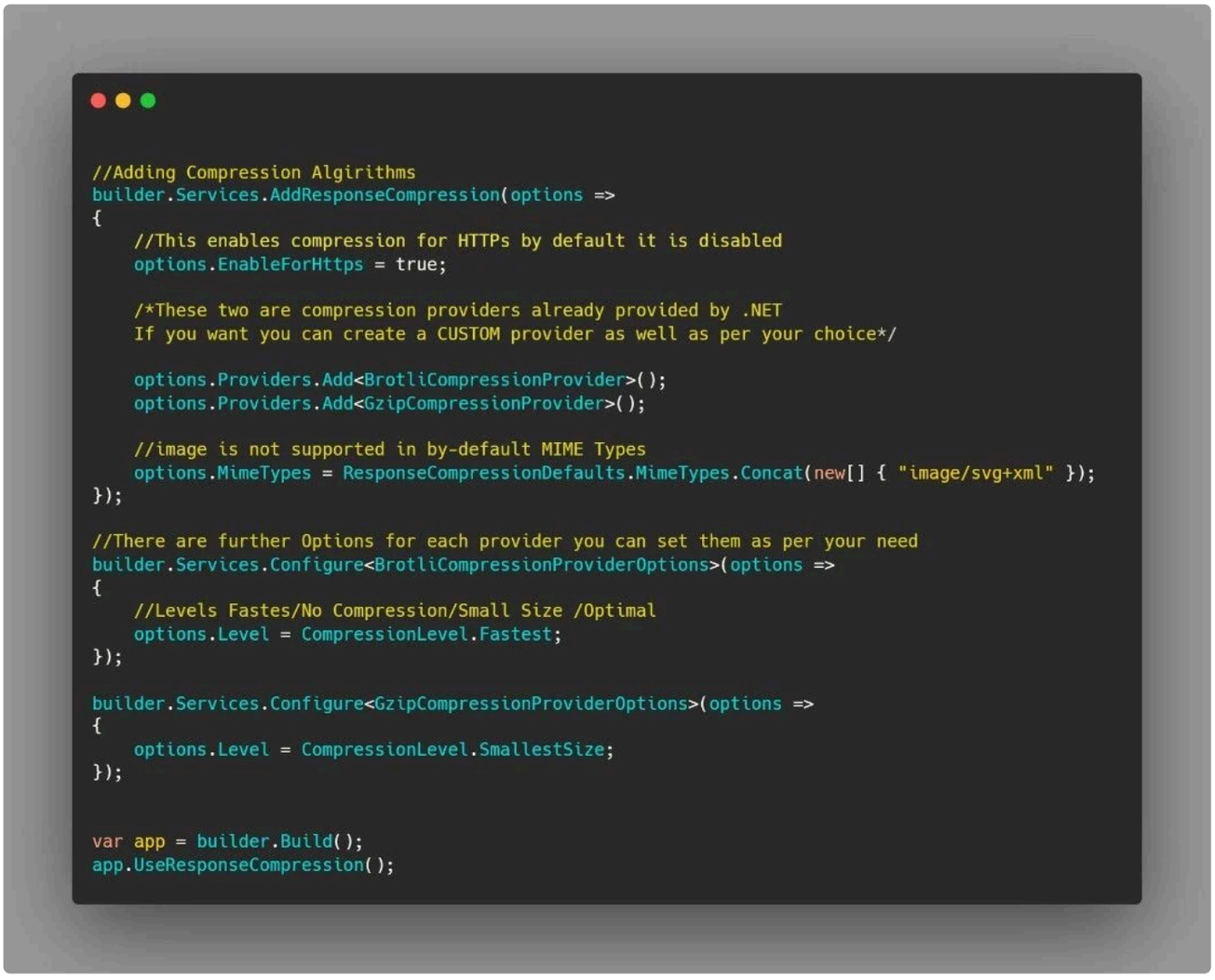
3.The Authorization header must not be present.

4. Cache-Control header parameters must be valid, and the response must be marked public and not marked private.
5. The Content-Length header value (if set) must match the size of the response body.

Some real-world examples of Response Caching

1. News website
2. E-commerce websites In your application you can apply response caching to those requests whose response is changed after a time and you are sure about it.

How can I verify it? Create an application apply caching and then request it from Postman and set time to 60 minutes, you will notice that only first request will reach the controller, after that even if you try request will not reach controller.

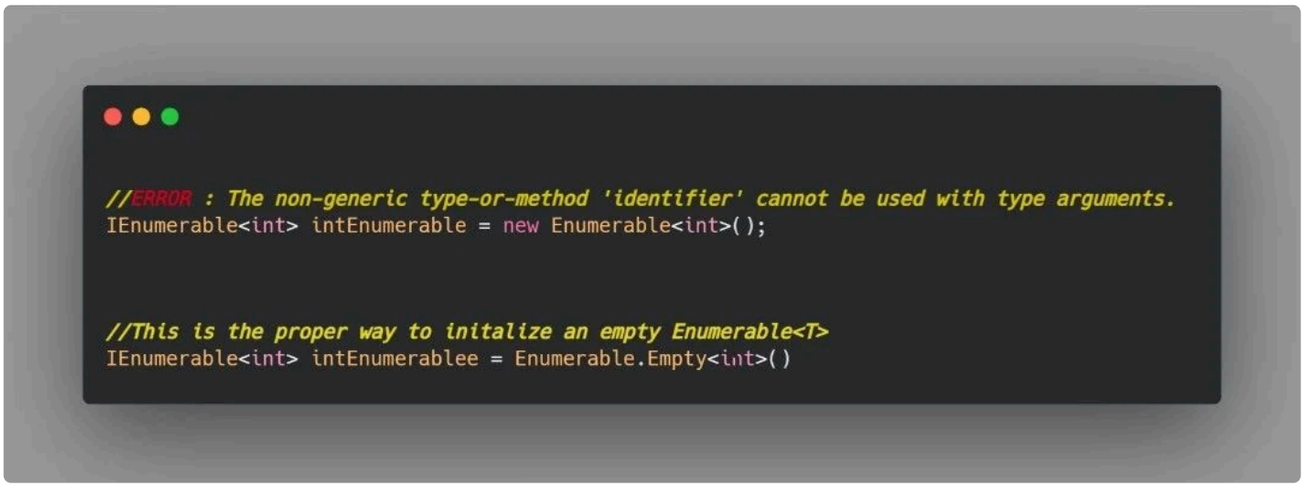


```
//Adding Compression Algirithms
builder.Services.AddResponseCompression(options =>
{
    //This enables compression for HTTPs by default it is disabled
    options.EnableForHttps = true;

    /*These two are compression providers already provided by .NET
    If you want you can create a CUSTOM provider as well as per your choice*/

    options.Providers.Add<BrotliCompressionProvider>();
    options.Providers.Add<GzipCompressionProvider>();

    //image is not supported in by-default MIME Types
    options.MimeTypes = ResponseCompressionDefaults.MimeTypes.Concat(new[] { "image/svg+xml" });
});

//There are further Options for each provider you can set them as per your need
builder.Services.Configure<BrotliCompressionProviderOptions>(options =>
{
    //Levels Fastes/No Compression/Small Size /Optimal
    options.Level = CompressionLevel.Fastest;
});

builder.Services.Configure<GzipCompressionProviderOptions>(options =>
{
    options.Level = CompressionLevel.SmallestSize;
});


var app = builder.Build();
app.UseResponseCompression();
```

Episode 19 : Do you know how to initialize an Empty Enumerable in .NET ?

✔️ .NET provides us the Empty() method to initialize an empty Enumerable of any type.

✔️ This method is useful for passing an empty sequence to a user defined method that takes an IEnumerable 🔃



Episode 20 : Dependency Injection Explained in .NET

This post crossed 100K views on my LinkedIn

Dependency injection involves providing a class with its required dependencies from an external source rather than having the class create them itself.

This helps to decouple the object creation process from the caller, leading to a more modular and flexible system.

In other words, it allows a class to focus on its core functionality rather than worrying about how to create and manage its dependencies.

Why do we need dependency injection? By using DI classes are decoupled from each other so you make changes at one place and it is reflected all over the places.

How to implement dependency injection?

✅ In .NET 6 we implement DI in Program.cs class by using builder.Services

✅ For previous versions of .NET, to implement DI we need to add the service in “ConfigureServices” method which is in Startup.cs file

Different ways of implementing Dependency Injection There are three ways of doing DI:

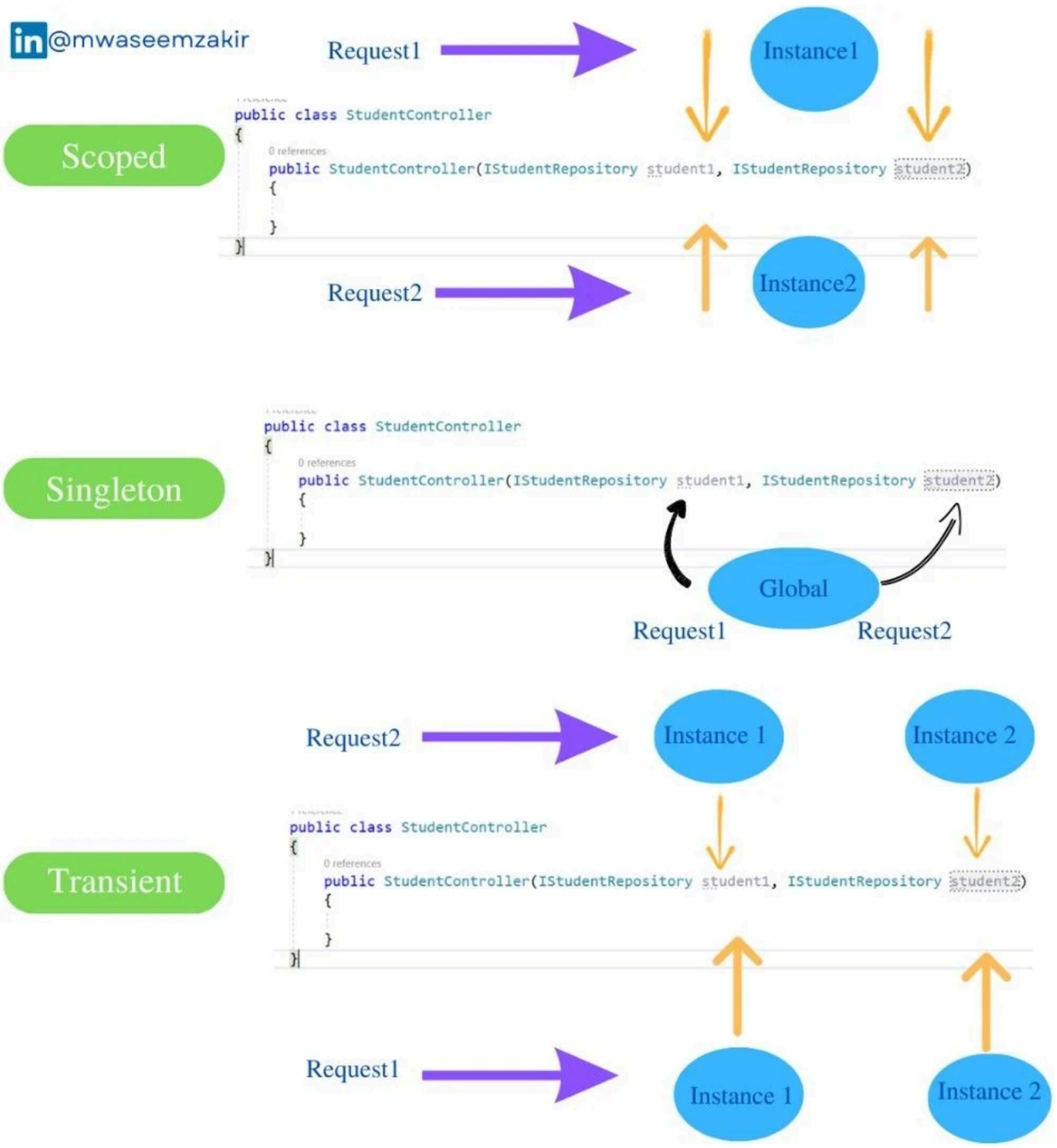
1.Scoped ➡️ It will create an instance per scope, if we are in same scope same instance would be used. Whenever we go out of scope new instance would be created.

2. Transient ➡️ It creates new instances every time its injected.
3. Singleton ➡️ It instantiates one global object for all requests coming to the server from any user.

Benefits of Dependency Injection

1.Improved testability: Dependency injection makes it easier to write unit tests for your code, as you can easily substitute mock objects for the real dependencies.

2. Enhanced flexibility: By injecting dependencies from the outside, you can easily change the implementation of a class's dependencies without having to modify the class itself. This makes it easier to adapt your application to changing requirements.
3. Increased modularity: Dependency injection encourages the use of small, single-purpose classes that are easy to test and reuse in different contexts. This can lead to a more modular and maintainable codebase.
4. Better separation of concerns: Dependency injection helps to separate the concerns of different parts of your application, making it easier to understand and maintain the code.
5. Enhanced decoupling: Dependency injection promotes loose coupling between classes, which can make your application more resilient to change and easier to test.



Episode 21 : IEnumerable vs IQueryable in .NET

IEnumerable and IQueryable interfaces are both used to work with collections of data and both support LINQ (Language Integrated Query).

IQueryable

✅ IQueryable executes queries on the server side.

✅ It is designed specifically to work with LINQ

✅ It extends IEnumerable, which means it includes all of the functionality of IEnumerable.

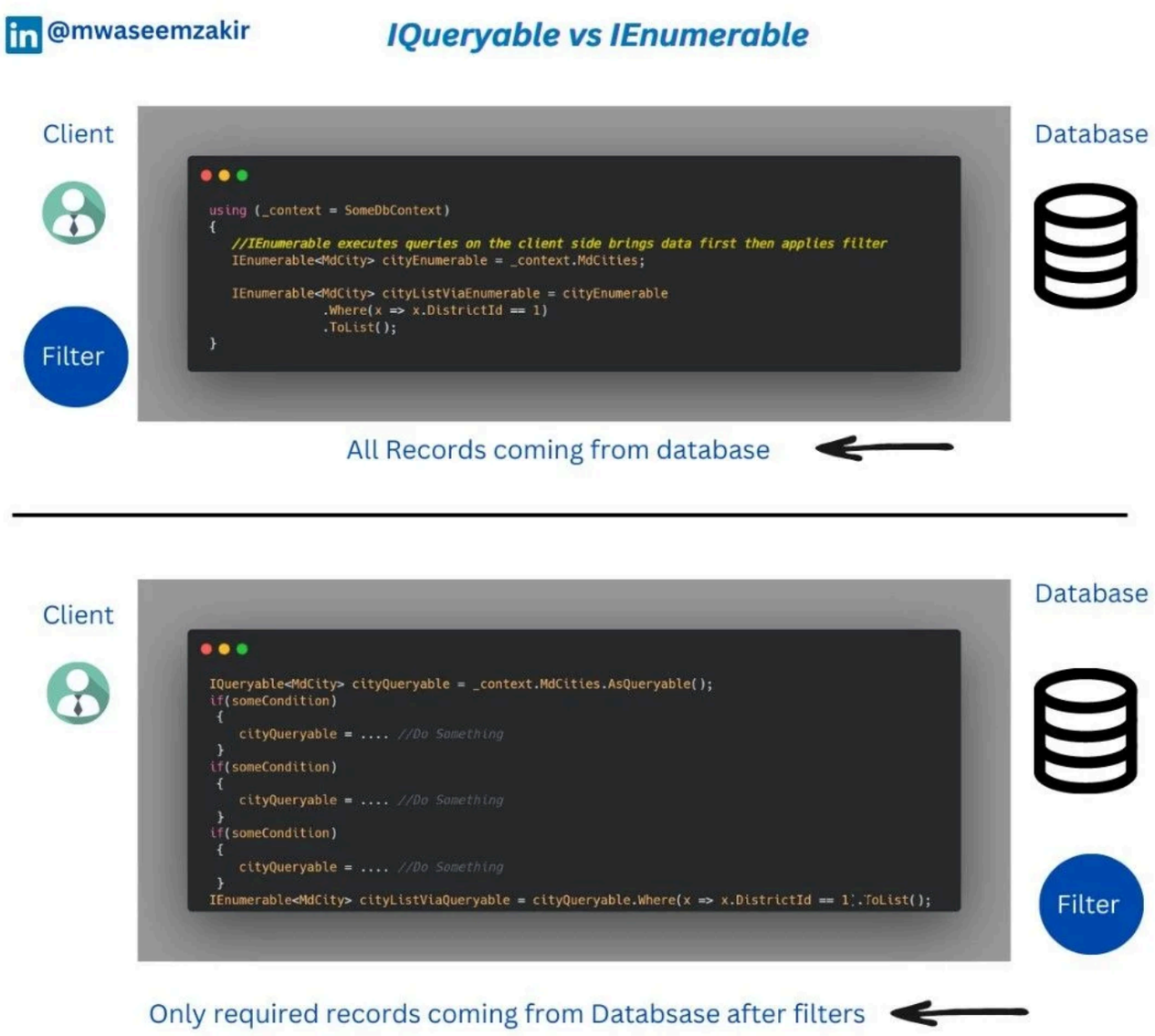
✅ It can be more efficient when working with large data

✅ It can be more helpful when you have to apply a lot of filtrations, you can apply all filters on Queryable and when you are done you can convert data to desired collection

IEnumerable ✔ IEnumerable executes queries on the client side.

✔ It is generally used to work with in-memory data collections.

Explanation with code 🔽



Episode 22 : Why is it generally considered good practice to keep Dependency Injections in separate class !

It is good practice to move all of your dependency injection work in a separate class instead of filling your Program.cs with all Injections. Here are some benefits of this approach

1. By keeping dependency injections in separate classes, you can better adhere to the principle of separation of concerns. So, your Program.cs is just focusing on configuration and its DI

class headache to manage dependencies.

2. Keeping dependency injections in separate classes can make it easier to maintain the application over time. For any change in DI's, you would not be changing the Program.cs rather you would just change the desired dependency injection class

3. For better understanding we can create different DI classes that would be dealing with similar dependencies.

Episode 23 : Difference b/w GetType() and typeOf() Methods in .NET

Both TypeOf and GetType help you to get the type with a little difference

✅ typeof gets the type from a class while GetType gets type from an object.

✅ The GetType method is used to retrieve the type of an object at runtime, while the typeof operator is used to retrieve the type of an object at compile-time.

I have described one example where typeof can prove helpful

Episode 24 : Difference b/w VAR and DYNAMIC keyword in C#

VAR ✅ VAR is early binded (statically checked)

✅ It looks at your right-hand side data and then during compile time it decides the left-hand side data type

✅ Use of var makes your code more readable, simplified and reduces the typing.

DYNAMIC ✔ Dynamic is late binded (dynamically evaluated).

✔ It is used to work on dynamic objects.

✔️ It helps us when we are not sure about the data type of objects, saves us from a lot of data type-oriented checking (because the compiler ignores them).

💭 There is a debate over var or proper type. I would like to hear your opinion, many people say that we should use var instead of proper type, but some say that if you are familiar with type then why would you go for var. 📝⏬

```
//Developer known type is List<Developer>
var developerKnown = new List<Developer>();

//unknownDeveloper is of dynamic type
dynamic unknownDeveloper = new List<Developer>();

//No such method exists , but this line will not give compiler error
unknownDeveloper.DynamicIsSillyFellow();

public class Developer
{
    public int RollNo { get; set; }
    public string Name { get; set; } = string.Empty;
    public int Age { get; set; }
}
```

Episode 25 : StringBuilder vs string in C#

Difference b/w String and StringBuilder

StringBuilder

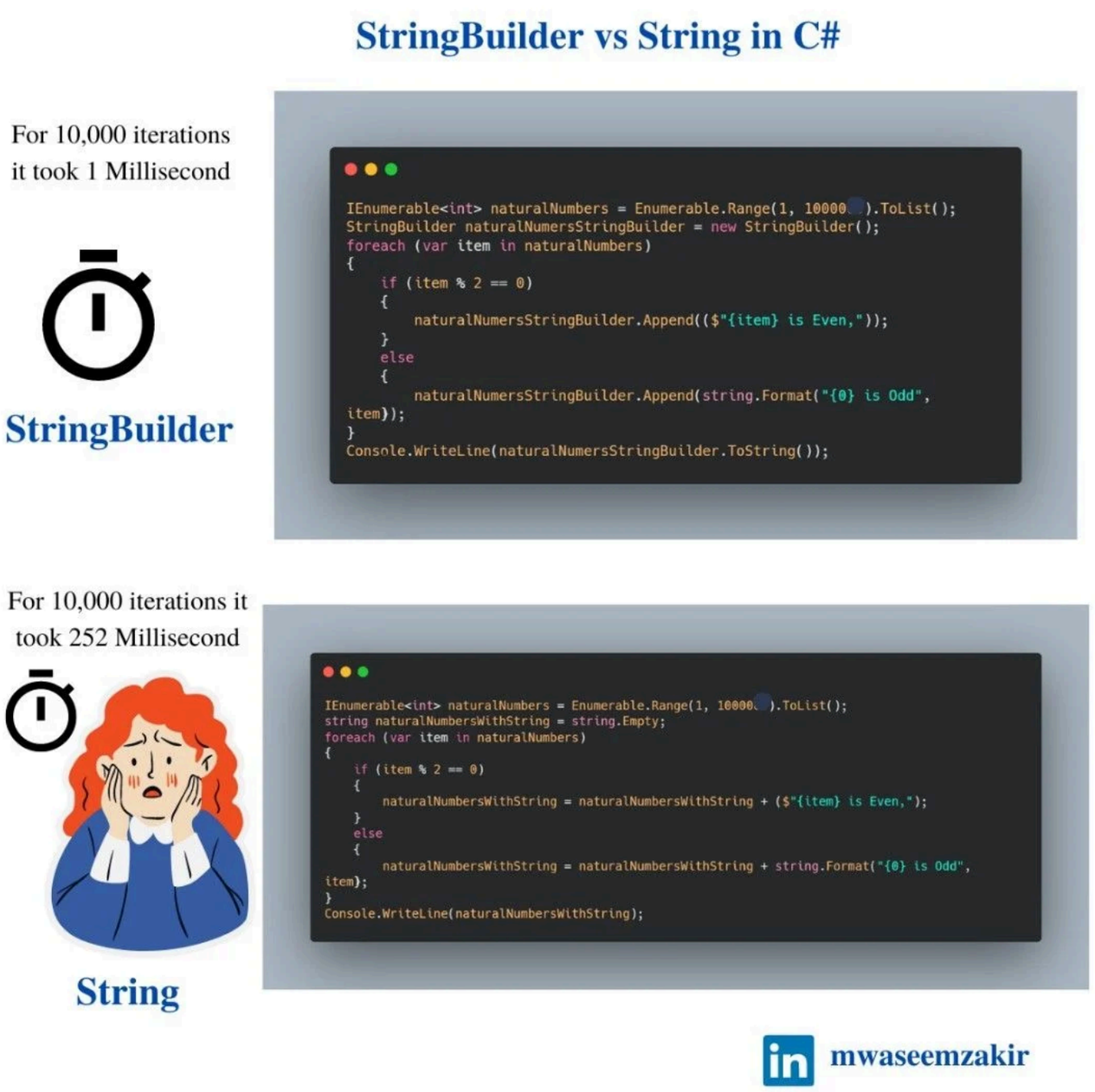
1. StringBuilder is mutable
2. StringBuilder will only create one object on heap and every time it would be updated with new value even if you append/insert 1 million values.

String

1. String is immutable.
2. Every time when we update data in string it creates a new instance of object. So, if you update value 1K times it will create 1K new instances.

Time difference over 10,000 iterations A good rule of thumb is to use strings when you aren't going to perform operations like(Append/Remove) repetitively, use StringBuilder for vice versa.

I took 10,000 iterations and checked the difference for that see the difference in picture. ⏬



Episode 26 : Arrays vs ArrayList in C#

Difference b/w Array and Arraylist and which one is faster.

Array

1. Arrays are fixed in size.
2. It is strongly typed, in other words when you create an array it can store only one data type.

Array List

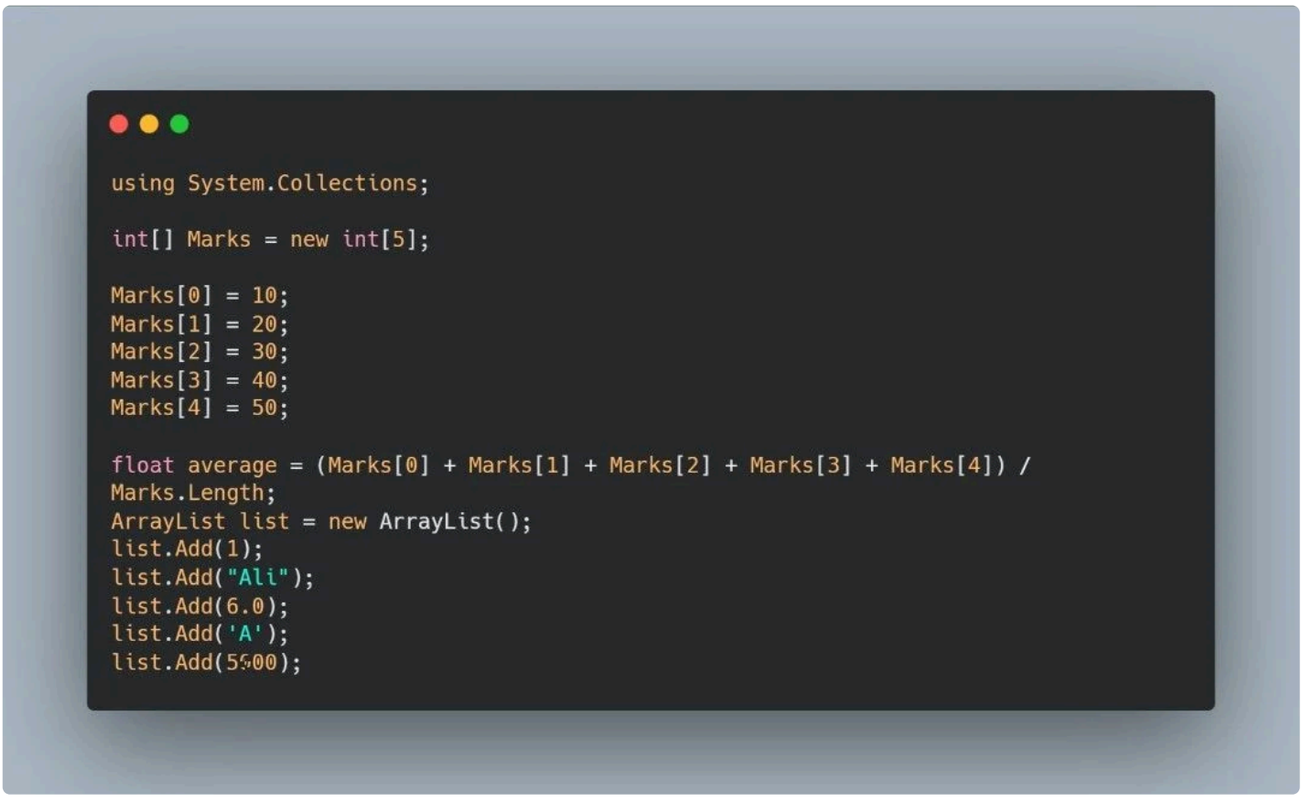
1. It is a collection from System.Collection in .NET
2. It is dynamically resizable.
3. It can store any data type

Which one is faster and Why ? Array list takes any data type which leads to boxing and unboxing. As arrays are strongly typed, they do not do boxing and unboxing. So, arrays are faster as compared to array lists.

When boxing and unboxing happens the data needs to jump from stack memory to heap and vice-versa which is a bit of memory intensive process. As a good practice avoid boxing and unboxing wherever possible.

Recap of Boxing and Unboxing Boxing and Unboxing are used to convert value types to reference types and vice versa. When the value type is moved to a reference type it's called Boxing. The vice-versa is termed as Unboxing.

When should we use Arraylist or Array? Overall, Arraylist is a more flexible and convenient choice when you need to work with a collection of objects that can change size over time, while an array is a good choice when you need to work with a fixed-size collection of elements of the same data type.



```
using System.Collections;

int[] Marks = new int[5];

Marks[0] = 10;
Marks[1] = 20;
Marks[2] = 30;
Marks[3] = 40;
Marks[4] = 50;

float average = (Marks[0] + Marks[1] + Marks[2] + Marks[3] + Marks[4]) /
Marks.Length;
ArrayList list = new ArrayList();
list.Add(1);
list.Add("Ali");
list.Add(6.0);
list.Add('A');
list.Add(5600);
```

Episode 27 : Extension Methods in C#

An extension method is a special kind of static method that allows you to "add" methods to an existing type without modifying the type itself.

You already know about extension method let me remind you, you might have used method ToString() that is extension Method for string

Benefits of using extension methods

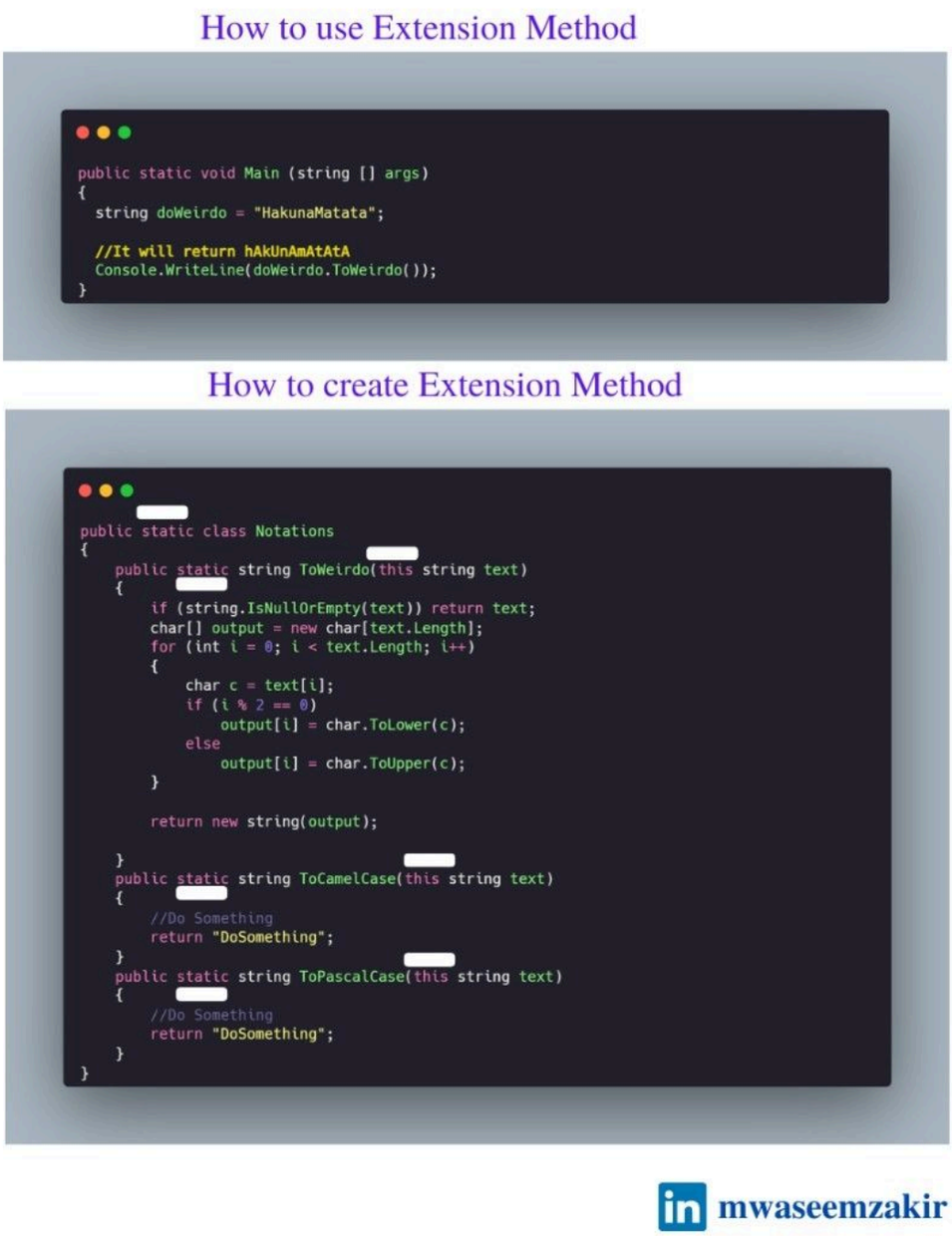
1. Extension methods allow you to reuse code across multiple types without having to create a new subclass or interface for each type.

2. They reduce code duplication and improve the maintainability of your code.

3. They can make your code more readable by allowing you to define methods that are directly related to the type they are extending
4. These methods are used extensively in the LINQ (Language Integrated Query) library.
5. They are very simple to develop just a static method of static class and in parameter you add this keyword.

How to create Extension Method? Extension methods are defined as static methods in a static class, and use the "this" keyword to specify the type they are extending.

Some scenerio Suppose you are creating some game and your text should be in some weird notations that all characters should be lower case followed by upper case, so it's better to create an extension method for string and then use it throughout the application.



Episode 28 : Common design principles you should keep in mind while developing applications

✅ **Separation of concerns** This principle asserts that software should be separated based on the kinds of work it performs.

Consider a media player application that has a feature to create and save playlists. The application has logic to retrieve a list of songs from the user's library and logic to organize the songs into playlists. The behavior for retrieving the list of songs should be separate from the behavior for creating the playlists, since these are separate concerns.

✅ **Encapsulation** Encapsulation is a way to protect the data inside an object from being changed by code outside of that object.

In other words, it helps to keep the internal state of an object hidden from the rest of the program. Instead of allowing other parts of the program to directly access and change the data inside an object, we should provide specific methods (getter/setter) that can be used to manipulate the data in a controlled way.

✅ **Single responsibility** The single responsibility principle is a concept in software development that says that each object or component in a program should only have one job or responsibility.

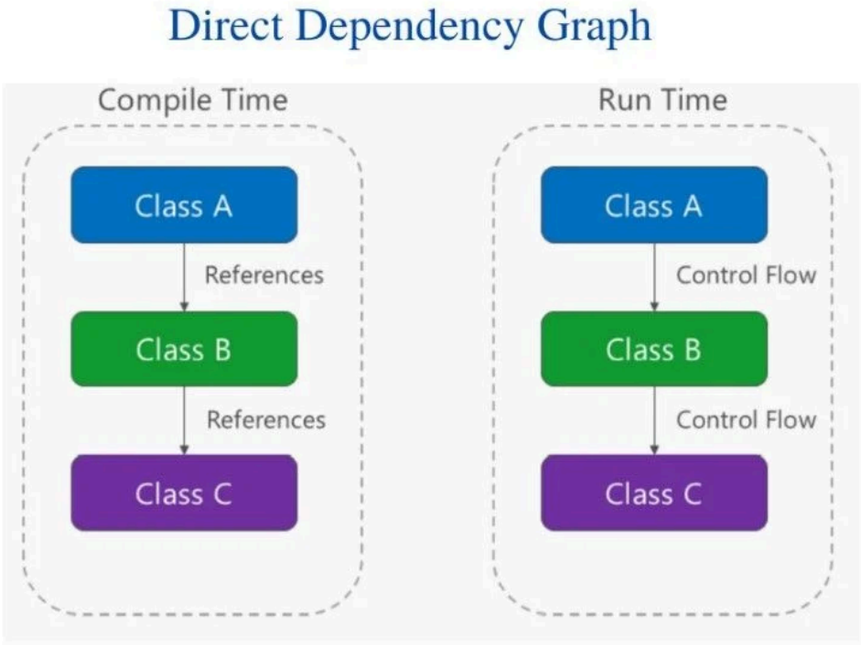
For example, the user interface should be responsible for presenting information to the user, while the data access layer should be responsible for storing and retrieving data. The business logic, which is the part of the program that does the important work, should be kept in its own section so it can be tested and changed without affecting other parts of the program.

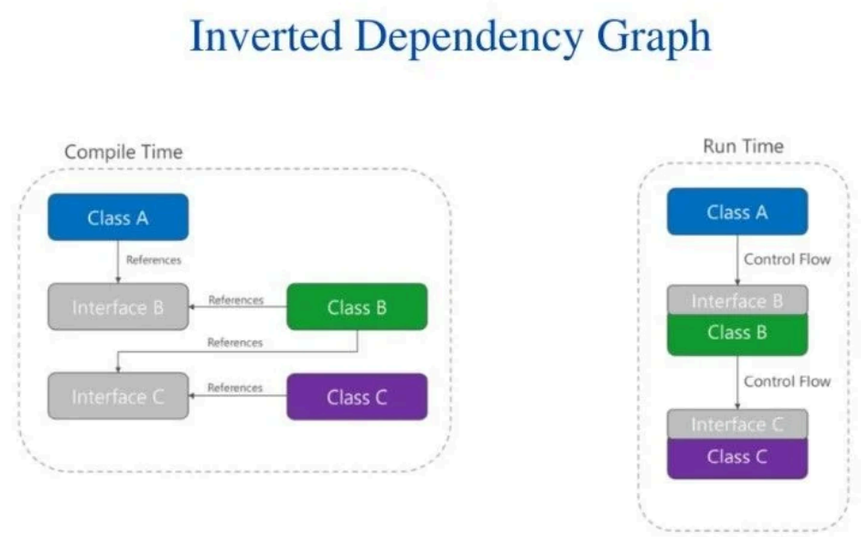
✅ **Don't repeat yourself (DRY)** The application should avoid specifying behavior related to a particular concept in multiple places as this practice is a frequent source of errors.

Avoid binding together behavior that is only coincidentally repetitive. For example, just because two different constants both have the same value, that doesn't mean you should have only one constant, if conceptually they're referring to different things. Duplication is always preferable to coupling to the wrong abstraction.

✅ **Dependency Inversion** The direction of dependency within the application should be in the direction of abstraction, not implementation details.

The practice of dependency injection is made possible by following the dependency inversion principle. See the difference of graph when Dependency Inversion is applied.





Episode 29 : Sealed keyword in C#

When applied to a class, the sealed modifier prevents other classes from inheriting from it.

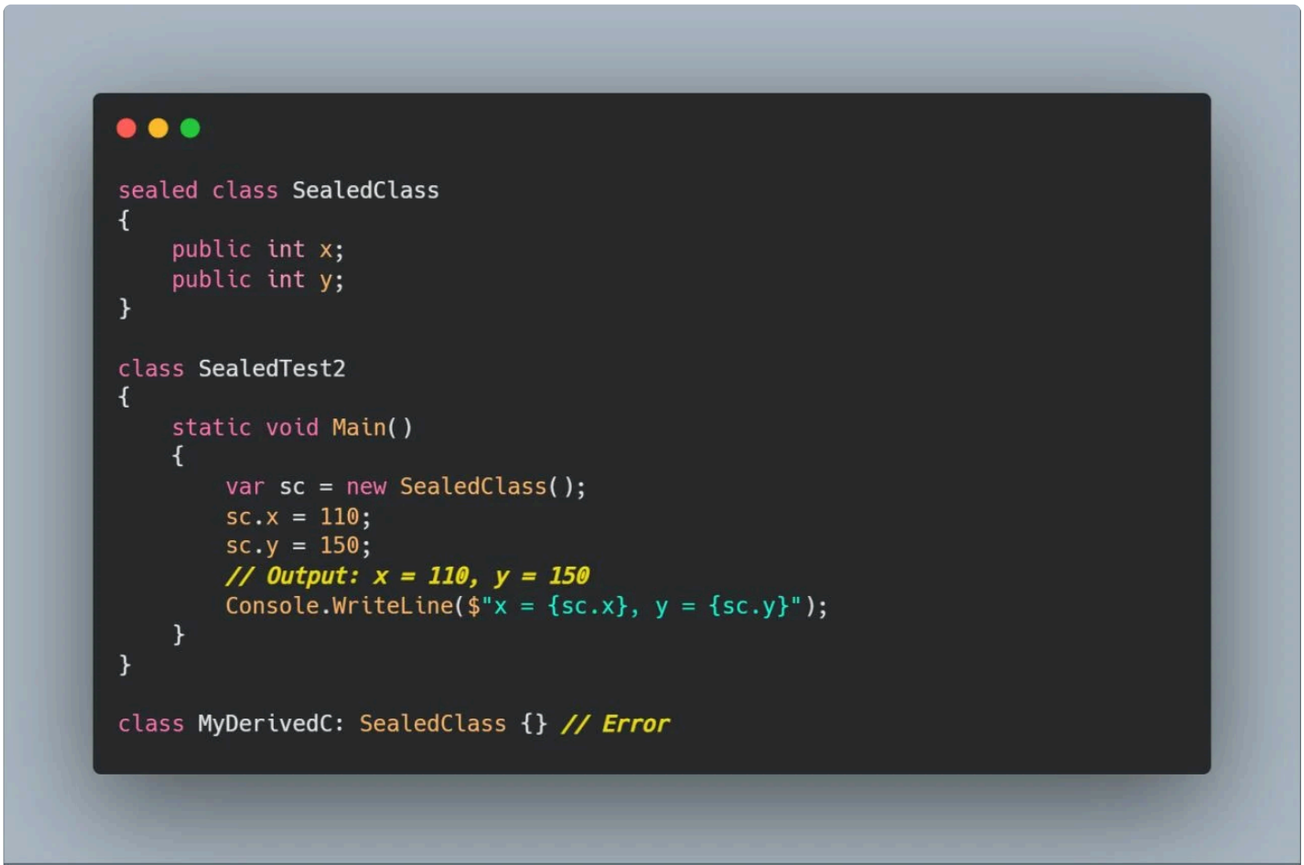
When applied to a method or property, the sealed modifier must always be used with override because structs are implicitly sealed, they cannot be inherited.

When should we make our class sealed

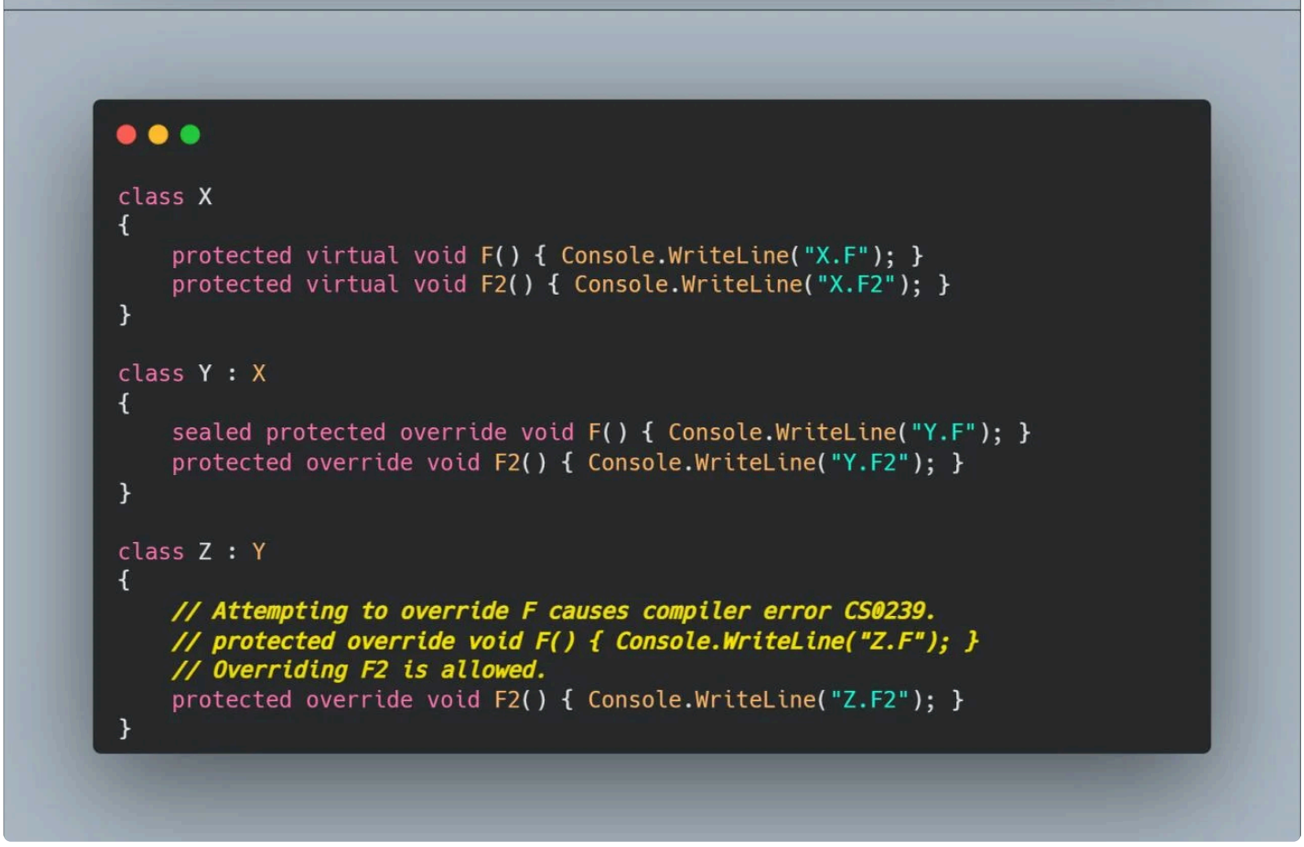
1. It ensures that a class cannot be subclasses / used as base class, either to maintain the integrity of the class's design or to improve performance.
2. When you want to create a class that can be used as a singleton, to ensure that only one instance of the class can be created (Although defining a class does not ensure that it would become singleton, we need to take care for few more things for singleton).

How making a class sealed improves the performance A sealed class does not have to worry about executing code in derived classes that may override members of the sealed class at runtime It reduces the number of methods calls that need to be made at runtime. When a method is called

on an object, the runtime must determine which method to execute by checking the object's type and searching up the inheritance hierarchy until it finds a matching method.



```
sealed class SealedClass
{
    public int x;
    public int y;
}

class SealedTest2
{
    static void Main()
    {
        var sc = new SealedClass();
        sc.x = 110;
        sc.y = 150;
        // Output: x = 110, y = 150
        Console.WriteLine($"x = {sc.x}, y = {sc.y}");
    }
}

class MyDerivedC: SealedClass {} // Error
```



```
class X
{
    protected virtual void F() { Console.WriteLine("X.F"); }
    protected virtual void F2() { Console.WriteLine("X.F2"); }
}

class Y : X
{
    sealed protected override void F() { Console.WriteLine("Y.F"); }
    protected override void F2() { Console.WriteLine("Y.F2"); }
}

class Z : Y
{
    // Attempting to override F causes compiler error CS0239.
    // protected override void F() { Console.WriteLine("Z.F"); }
    // Overriding F2 is allowed.
    protected override void F2() { Console.WriteLine("Z.F2"); }
}
```

Episode 30 : String Interpolation vs Verbitam Identifier vs Raw String Literal

Let's see the difference b/w them

String interpolation using $ The $ special character identifies a string literal as an interpolated string. An interpolated string is a string literal that might contain interpolation expressions. String interpolation provides a more readable, convenient syntax to format strings. It's easier to read than string composite formatting.

Verbatim Identifier - @ The @ special character serves as a verbatim identifier. It can be used in the following ways

1. To enable C# keywords to be used as identifiers.
2. It helps to specify a string that should be interpreted literally, without any escape characters being processed.

Raw string literal - """ A raw string literal (Available in C# 11.0) starts and ends with a minimum of three double quote (") characters.

1. It can span multi lines
2. Like @ it can also specify a string that should be interpreted literally
3. It can combine work with interpolated string
4. It can be helpful in designing server-side emails as well

While working with them following rules should be kept in mind.

1. Both opening/closing quote characters must be on their own line.
2. Any whitespace to the left of the closing quotes is removed from all lines of the raw string literal.
3. Whitespace following the opening quote on the same line is ignored.
4. Whitespace only lines following the opening quote are included in the string literal.

mwaseemzakir

Raw String Literal
"""

```
var html = """
            <nz-form-item>
            <nz-form-label  nzRequired [nzSpan]="7"></nz-form-label>
            <nz-form-control nzErrorTip="Enter Title" [nzSpan]="8">
                <input class="input-type-one w-100" formControlName="title" placeholder="Title"/>
            </nz-form-control>
            </nz-form-item>
        """;
```

String Interpolation
$

```
string name = "Muhammad Waseem";
var date = DateTime.Now;
// Composite formatting:
Console.WriteLine("Hello, {0}! Today is {1}, it's {2:HH:mm} now.", name, date.DayOfWeek,
date);

// String interpolation:
Console.WriteLine($"Hello, {name}! Today is {date.DayOfWeek}, it's {date:HH:mm} now.");
```

Verbitam
@

```
string withVerbitam = @"This is a \nstring";
string withputVerbitam = "This is a \nstring";

Prints :-  This is a \nstring
Console.WriteLine(withVerbitam);
 Prints : - This is a
            string
Console.WriteLine(withputVerbitam);


string[] @for = { "John", "James", "Joan", "Jamie" };
for (int @while = 0; @while < @for.Length; @while++)
{
    Console.WriteLine($"Here is your gift, {@for[@while]}!");
}

// The example displays the following output:
//     Here is your gift, John!
//     Here is your gift, James!
//     Here is your gift, Joan!
//     Here is your gift, Jamie!
```

Whenever you're ready , there are 3 ways I can help you

1. Book me for 1 hour 1:1 Session to clear all of your doubts about how to start LinkedIn Content Creation Journey , reach me at email or LinkedIn Inbox
2. Promote yourself to **8000+** subscribers by sponsoring my newsletter, reach me at any of my social media account
3. Become a **Patron** and get access to 150+ .NET Questions and Answers , I add 2-5 new questions every week.

PLEASE DON'T FORGET TO LEAVE 5 STARS , IF YOU LIKED THIS BOOK

Special Offers

1. **A Clean Architecture Course:** Every .NET developer should take, if you are planning to learn about clean architecture then grab this opportunity and get 10% discount using my promo code : **MUWAS**
2. **Ultimate ASP.NET Core Web API Second Edition - Premium Package:** Use my promo code **9s6nuez** for 10% discount

H A K U N A - M A T A T A ☺