Ping authen

Service

Block diagramme

Design principals

Serilog

C#

Number array -odd number square

3 consecutive odd even true

Capitalized first element from string

Can we write try with finally only-yes

If exception happens in catch will it execute finally – yes

If Exception in catch it will always go to Exception ex

Can try has multiple catch-yes

Collections

Generics

Generic and nongeneric collection

Delegates

Async await

Class template to make it immutable

Shallow copy , Deep copy in c#

Sealed static readonly extension methods , linq ,entity framework , sql- indexes nonclusterd index how indexes internally work

Oops

Dynamic garbage collection multithreading task parallelism

Middleware dependency injection hosting environment lifetime of services,configurations -app settings ,hosting environment how you read configuration development environment

Dotnetcore execution lifecycle

Web servers inproc outproc kestrel server,custom middleware what is process

Web Api

API gateway,Rest Api SOAP API,CLIENT SERVER Architecture

Logging in WEB API(Generic logger,Custom logger),HttpMethods jsonpath patch different,Custom Model Binding,custom validation,return types web apis,2 series-200,4 series,Security of Web APIS

Idempotency in Web API,Ratelimiting factor in Web API,Routing in WEB API,API versioning

DEVEOPS PIPELINE RELEASE PIPELINE,CODE COVERAGE,CODE COMPLEXITY

How you will make class as immutable

Making a class immutable in C# ensures that its instances cannot be modified after they are created. Here’s how you can achieve this:

**Steps to Create an Immutable Class:**

1. **Use readonly fields** – Declare fields as private readonly so they can only be set during object creation.
2. **Initialize fields via a constructor** – Assign values to fields through a constructor, preventing modification after initialization.
3. **Avoid setters** – Properties should be read-only (get only) and not have set accessors.
4. **Prevent modifications from outside** – If your class has collections, expose them as read-only (ReadOnlyCollection<T> or return copies).
5. **Ensure deep immutability** – Any reference-type fields should also be immutable or return defensive copies.

**Example:**

public class ImmutablePerson

{

public string Name { get; }

public int Age { get; }

public ImmutablePerson(string name, int age)

{

Name = name;

Age = age;

}

}

**Handling Collections:**

If your class contains collections, return a read-only version:

public class ImmutableTeam

{

private readonly List<string> members;

public ImmutableTeam(IEnumerable<string> members)

{

this.members = new List<string>(members); // Defensive copy

}

public IReadOnlyList<string> Members => members.AsReadOnly();

}

**Benefits of Immutability:**

* **Thread Safety** – No unexpected modifications in a multi-threaded environment.
* **Predictability** – Easier debugging and reasoning.
* **Reliability** – Helps avoid unintended side effects.

Would you like more examples or explanations?

**Access Modifiers in C#**

Access modifiers in C# define the **visibility** and **accessibility** of classes and their members. They help enforce **encapsulation** by restricting access to certain parts of the code.

**Types of Access Modifiers**

| **Modifier** | **Accessibility** |
| --- | --- |
| **public** | Accessible from anywhere in the program. |
| **private** | Accessible only within the same class. |
| **protected** | Accessible within the same class and derived classes. |
| **internal** | Accessible within the same assembly. |
| **protected internal** | Accessible within the same assembly and derived classes in other assemblies. |
| **private protected** | Accessible within the same class and derived classes in the same assembly. |
| **file** | Accessible only within the same file (introduced in C# 11). |