Contents

[Entity Framework 1](#_Toc210135383)

[LINQ 6](#_Toc210135384)

[OOPS 6](#_Toc210135385)

[General Questions 7](#_Toc210135386)

[Threading/ Async Await / Task 15](#_Toc210135387)

[ASP.NET 15](#_Toc210135388)

[Web API 16](#_Toc210135389)

[JWT 16](#_Toc210135390)

# Entity Framework

Question 1: Difference between code-first, database-first, and model-first approaches?

1. **Code-First** → You write **C# classes** first → EF creates database from them.
2. **Database-First** → In **Database First Approach**, the **database is already created** (tables, relationships, keys, etc.). We use Scaffold-DbContext command to generate **models and DbContext** from tables.



1. **Model-First** → You design a **visual model (EDMX)** → EF generates both classes and database. **EF Core does not support Model-First (EDMX designer)** — it only supports **Code-First** and **Database-First**. Model-First was only in **older EF (up to EF6)**.

Question 2: Explain eager loading vs lazy loading vs explicit loading?

1. **Eager Loading** → Related data is loaded **immediately together** with the main data (using Include).It uses joins.
2. **Lazy Loading** → Related data is loaded **only when you access it later** (needs EF Core Proxies and virtual keyword with navigation property). It don’t uses joins.
3. **Explicit Loading** → You **manually tell EF** to load related data (using Load). Not use joins.

(context.Entry(...).Collection(...).Load()).

1. **Which to use?** → In real projects we **prefer Eager Loading**, because it clearly fetches everything in one go and avoids surprises. **Lazy Loading** can cause many hidden queries, so it’s rarely used in big projects. **Explicit Loading** is used only when we want full manual control, but not for everyday cases.

Example- one to many (one student have many course but course will have single student)

A computer screen shot of text

AI-generated content may be incorrect.

Eager Loading (Include) - I want Student + all his Courses immediately in one query.

A black background with white text

AI-generated content may be incorrect. A black background with white text

AI-generated content may be incorrect.

Explicit Loading (Load) First give me Student, later I’ll decide whether to fetch his Courses.

A computer screen shot of a program code

AI-generated content may be incorrect. A computer screen with white text

AI-generated content may be incorrect.

Lazy Loading Don’t fetch related data until I actually try to use the navigation property. EF will **auto-fetch** when you first access student.Courses

To Enable Lazy loading follow steps:

1. Install 
2. Set this in program.cs. UseLazyLoadingProxies();

A computer screen shot of a computer code

AI-generated content may be incorrect.

1. Mark navigation properties as virtual

A computer screen shot of white text

AI-generated content may be incorrect.

A computer screen shot of a program

AI-generated content may be incorrect. A computer screen shot of a program

AI-generated content may be incorrect.

Question 3: What's the difference between DbContext and ObjectContext?

1. DbContext → DbContext is the main class in Entity Framework Core that connects our C# code with the database. It manages the database connection, tracks changes of entities, and lets us query or save data.
2. ObjectContext → Older, heavier, low-level API (used in EF 4.x/5/6).
3. Today, we mostly use DbContext.

Question 4: How do you handle transactions in EF Core?

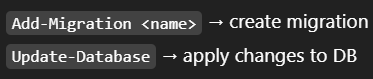
We handle transactions in EF Core using Database.BeginTransaction(). If all operations succeed, we commit; otherwise, we rollback to keep data safe. a transaction makes sure a group of database operations are all done successfully or none at all.

A screen shot of a computer program

AI-generated content may be incorrect.

Question 5: What are migrations in EF Core and how do you use them?

1. Migrations in EF Core are a way to keep the database schema in sync with our model classes. Whenever we change our C# models, migrations update the database without dropping data.
2. Common commands



Question 6: *Explain Change Tracking in EF Core*?

* Change Tracking is a feature in EF Core where the DbContext keeps track of all the changes made to entities (objects) retrieved from the database. It monitors whether an entity is **added, modified, or deleted**, so that when SaveChanges() is called, EF Core knows which SQL statements to generate and execute.
* By default, all entities retrieved via DbContext are **tracked**.
* Change tracking improves efficiency for updates but uses **more memory**.

A screenshot of a computer program

AI-generated content may be incorrect.

Question 7: *What is the difference between AsNoTracking vs Tracking queries?*

* **Tracking**: EF Core watches changes → needed for update/delete → slower for large data.
* **AsNoTracking**: EF Core does not watch → read-only queries → faster, less memory.

A black screen with white text

AI-generated content may be incorrect.

Question 8: *What is Concurrency Handling in EF Core?*

* control ensures that **two users don’t overwrite each other’s changes** when updating the same record. EF Core provides **Optimistic Concurrency**, which assumes conflicts are rare and checks **before saving**.

**How EF Core does it internally**

* EF Core uses a concurrency token (commonly a RowVersion column [Timestamp] / RowVersion).
* When SaveChanges() is called, EF Core checks whether the token in the database matches the token in your entity. EF Core generates a SQL UPDATE statement with a WHERE clause that includes the original RowVersion. And we get token when we fetch entity.
* If **no rows are affected**, EF Core throws DbUpdateConcurrencyException.

A screen shot of a computer

AI-generated content may be incorrect.

**Steps to implement concurrency control in EF Core**

* Add a RowVersion column of rowversion type ssms will auto manage in your table



* Update your entity model:

A computer screen shot of a program code

AI-generated content may be incorrect.

* **Use try-catch to handle conflicts**:
* A screen shot of a computer program

  AI-generated content may be incorrect.

Question 9: *How SQL Server (SSMS) handles concurrency in stored procedures?*

In SQL Server, concurrency can be handled using:

* **Optimistic concurrency** → check before update (like EF Core) – ( eg with RowVersion)

A screenshot of a computer program

AI-generated content may be incorrect. A screenshot of a computer program

AI-generated content may be incorrect.

* **Pessimistic concurrency** → lock the row/table while working. *UPDLOCK + ROWLOCK* → SQL Server locks the row so no one else can update it until the transaction commits.

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

# LINQ

Question 1: *Difference between IEnumerable, IQueryable, ICollection, List*

IEnumerable –

* *What it is:* Base interface for looping over collections. Read-only. Filters run in memory.
* *Cannot:* Modify items or access by index.
* *Why we need it:* To loop through collections using foreach or simple LINQ in memory.

IQueryable –

* ***What it is:*** Extends IEnumerable for database querying. Queries are translated to SQL and executed on DB. Executes queries on the database, fetching only required data.
* ***Cannot:*** Modify collection directly or use index access.
* ***Why we need it:*** To fetch **only required data** from DB instead of loading everything in memory.

ICollection<T> -

* *What it is:* Extends IEnumerable with Add, Remove, Count. Works in memory.
* *Cannot:* Query databases or support index access.
* *Why we need it:* To **modify** and **manage** in-memory collections.

IList<T>

* *What it is:* Extends ICollection, supports index-based access.
* *Why we need it:* When position-based access (list[0]) is required.

**List<T>**

* *What it is:* A concrete class implementing IList, ICollection, IEnumerable. Supports Add, Remove, Indexing, Sorting.
* *Cannot:* Query databases. Only in memory.
* *Why we need it:* When we need a full-featured collection with indexing + modification.

A screen shot of a computer program

AI-generated content may be incorrect. A computer screen shot of a program code

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect. A computer screen shot of a program

AI-generated content may be incorrect.

A computer screen shot of a black screen

AI-generated content may be incorrect. A computer screen shot of white text

AI-generated content may be incorrect.

No where no orderby in IEnumerable

A black background with white text

AI-generated content may be incorrect.A black background with white text

AI-generated content may be incorrect.

A screen shot of a computer program

AI-generated content may be incorrect. A screen shot of a computer program

AI-generated content may be incorrect.

# OOPS

Question 1: *Why do you use an Interface?*

In real projects, when we don’t use interfaces, our code becomes **tightly coupled**.

For Example: Suppose I have to **send OTP to users**. At first, I only support **SMS**: A computer screen shot of text

AI-generated content may be incorrect. A computer screen shot of text

AI-generated content may be incorrect.

“In real projects, if I don’t use an interface, my class becomes ***tightly coupled***. For example, if my OtpService directly uses SMS, tomorrow if business says ‘send OTP via WhatsApp’, I must keep editing the same class again and again. ***With an interface, my service doesn’t care how the OTP is sent — SMS, Email, WhatsApp — I just plug in the required object using Dependency Injection***. This makes the code flexible, testable, and easy to maintain.”

Question 2: *Difference between abstract class vs interface?*

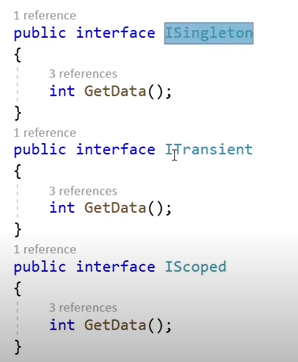
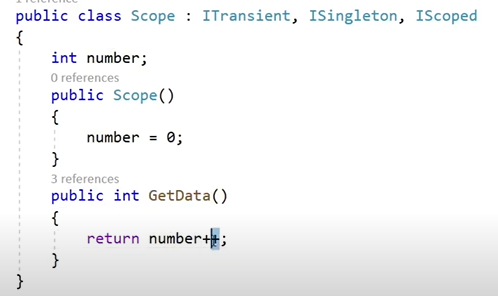
# General Questions

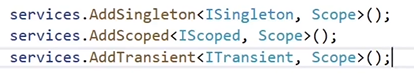
Question 1: *What is Dependency Injection? Why should you use it?*

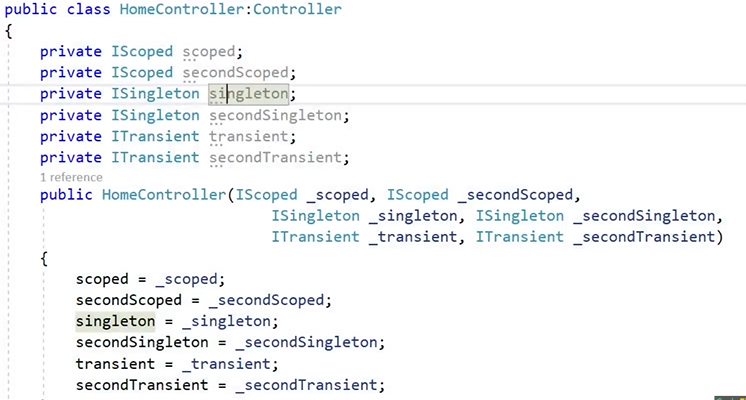
DI means instead of my class creating objects itself, I get those objects from outside. This keeps the code **loosely coupled (decouple), flexible, and easy to test.**

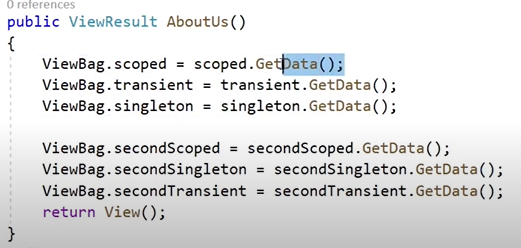
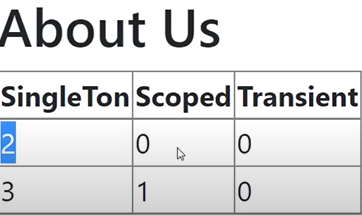
Lifetimes in DI

1. **Transient:** When object is requested, new instance will be created every time.
2. **Scoped:** When object is requested, a new instance will be created and will return the same object through out the life of the http request.
3. **Singleton:** When object is requested, a new instance will be created and will be returned same object throughout of the life of the application.

** **

****

****

** **

Use case

* **Transient:** We usually use this for small helper or utility services, like a password generator, where we want a fresh object every time it is called.
* **Scoped:** The most common example is DbContext, where we want the same instance to be reused throughout a single HTTP request, but not shared between different requests.
* **Singleton:** We mostly use this for services like ILogger or MemoryCache, where one shared instance is enough for the entire lifetime of the application.

**A screenshot of a computer program

AI-generated content may be incorrect.**

If a class doesn’t hold state(Value), Transient, Scoped, or Singleton behave the same. Lifetimes matter only for objects that store data between calls. Lifetime affects **stateful objects**, not stateless methods that generate fresh values each time.

Question 2: *What is the difference between Decryption and Hashing?*

Decryption is reversible — for example, I can encrypt a message using AES and then decrypt it with the key to get the original text. Hashing(non reversible) is one-way — for example, I store a password as a SHA256 hash; when the user logs in, I hash their input and compare hashes to verify it. So, decryption is for reading data, hashing is for verifying data without storing the original.

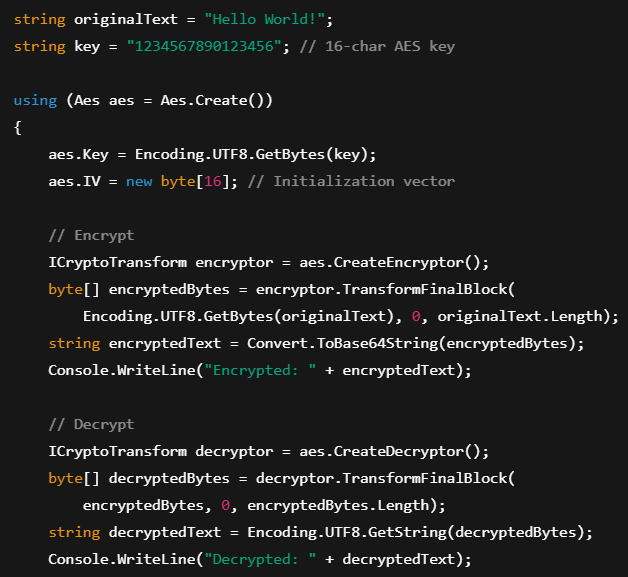
SHA-256 hashes passwords for secure storage, and AES encrypts messages so only someone with the key can read them.

A computer screen with white and green text

AI-generated content may be incorrect. A black and white text

AI-generated content may be incorrect.

**AES** stands for **Advanced Encryption Standard**. AES is a symmetric encryption( where the **same key** is used to **encrypt and decrypt** data) algorithm that securely encrypts and decrypts data using the same key.



This code encrypts and decrypts a message using AES. IV is the Initialization Vector that adds randomness to encryption. ICryptoTransform represents the encryptor or decryptor, and TransformFinalBlock actually performs the encryption or decryption on the byte data. Finally, we convert bytes to string to display the encrypted and decrypted text.

IV is like a salt for encryption — it adds randomness. Here it’s empty for simplicity, but in real projects we use a random IV so the same message doesn’t always encrypt to the same output.



Question 3: *What is a Delegate? Can you name some built-in delegates?*

A delegate is just a variable that can point to a method. Instead of always fixing which method to call, I can pass the method itself and call it later. This makes code more flexible.

A computer screen shot of a program

AI-generated content may be incorrect.

Question 4: *Explain Func, Action, and Predicate.*

1. Action :
   1. **What it is:** Points to a method that **does not return anything (void)**.
   2. **Use:** When you just want to perform an action.
2. Func :
   1. **What it is:** Points to a method that **returns a value**.
   2. **Use:** When you want to process something and get a result.
3. Predicate:
   1. **What it is:** Special version of Func that **always returns a bool**.
   2. **Use:** When you want to check a condition (true/false).

A computer screen with many colorful text

AI-generated content may be incorrect.

Question 4: *Difference between .NET Framework, .NET Core, and .NET 5/6/7.*

**.Net Framework –**

* Microsoft’s first platform, launched in 2002. It’s Windows-only and mainly used in legacy enterprise apps like WebForms, WCF, WinForms, WPF.
* Slow, because it loads the **whole big package** even if you need just one small feature.
* No CLI- Only through IDE
* Used for desktop apps (WinForms, WPF) and web apps (ASP.NET MVC, Web Forms).

**.Net Core 3.X –**

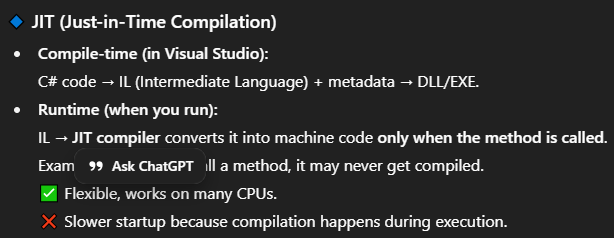
* came in 2016 as a modern, open-source, cross-platform version.
* Faster, because it loads only what you need (modular).
* Used for modern web apps, APIs, microservices, and console apps.

**.NET –**

* This unified platform combines the best of Framework, Core, and Xamarin.
* Fastest, because it has new optimizations like better memory handling, better GC (Garbage Collection) and AOT compilation.
* Supports desktop, web, cloud, mobile, IoT, AI, and gaming.

Question 4.1: *JIT VS AOT*

JIT compiles IL into machine code at runtime (just in time), while AOT compiles everything into machine code before runtime (ahead of time).

 A black and white screen with white text

AI-generated content may be incorrect.

“You can use AOT **to precompile the app**, so it runs directly without JIT.In .NET 7/8+, you enable AOT by setting **<PublishAot>true</PublishAot>** in the project file and publishing. It produces a native executable that runs without JIT.”

***By default .NET uses JIT, but if you enable NativeAOT during publish, it will produce a fully compiled native app without JIT, while GC still handles memory.***

A black background with white text

AI-generated content may be incorrect. A black screen with white text

AI-generated content may be incorrect.

**JIT =** IL → machine code at runtime  
**AOT =** IL → machine code before runtime

Question 4.2: *Explain IL.* *Benefit of compiling into IL code*

IL is a partially compiled code. IL is the CPU-independent code that C# or other .NET languages are compiled into before running. At runtime, the JIT compiler converts IL into machine code. For example, your app compiles to MyApp.dll (.dll is IL), which you can run cross-platform using dotnet MyApp.dll. (.dll = IL code we can run mvc and exe using MyApp.dll in linux server as well.)

**C# code --> IL -->(JIT converts to) --> Native macine language.**

**The main benefit of IL** is cross-platform support — the same compiled code can run on Windows, Linux, or macOS. With JIT, IL is converted to machine code at runtime. With AOT, IL is skipped and C# is compiled directly into machine code for the target platform.

Question 5: *Explain CLR, CTS, CLS.*

**CLR (Common Language Runtime)** is the engine that runs .NET apps. It executes your code, manages memory with the Garbage Collector, handles security and exceptions, and converts IL to machine code using JIT — or if you use AOT, it runs the precompiled native code directly.

**Main Functions:**

* **JIT Compilation** → Converts IL to machine code at runtime.
* **Memory Management** → Garbage Collector cleans unused objects.
* **Security & Exception Handling** → Protects from bad code and handles errors.

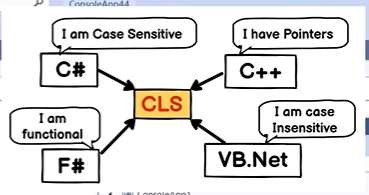
**CTS** As .net provide multiple language support so CTS (Common type system) ensures that data types defined in two different languages get compiled to a common data type.

Example int in C# and Integer in VB.NET both map to System.Int32, so a method returning int in C# can be used in VB.NET as Integer.

A computer screen with text on it

AI-generated content may be incorrect. A black background with white text

AI-generated content may be incorrect.

**CLS (Common Language Specification)** is a set of rules that ensures .NET code is usable across all .NET languages. If your code follows CLS, other languages like VB.NET, F#, or C# can access your classes, methods, and properties without compatibility issues. 

Question 6: *Difference between Value Types vs Reference Types.*

Value types contain data directly, reference types contain an address pointing to data.

*A screenshot of a computer

AI-generated content may be incorrect.* *A screenshot of a computer

AI-generated content may be incorrect.*

Question 7: *Boxing and Unboxing in C#.*

1. **Boxing** - Converting a **Value Type** into a **Reference Type (object)**. Happens **automatically (implicit)**. value → object (implicit)
2. **Unboxing** - Converting the **Reference Type (object)** back to a **Value Type**. Needs **explicit cast**. object → value (explicit)

A black background with white text

AI-generated content may be incorrect.

Question 7.1: *Difference between Array and ArrayList.*

A black screen with white text

AI-generated content may be incorrect.

**Why Boxing and Unboxing were Needed?**

* In early .NET (before Generics List<T>), collections like ArrayList could only store object.
* To put a value type (int, bool, double) inside, it had to be boxed (converted to object).
* When retrieving, it had to be unboxed (cast back to value type).

A black screen with white text

AI-generated content may be incorrect. A black screen with white text

AI-generated content may be incorrect.

Question 8: *What are Generics in C# and their benefits?*

**Definition:** Generics allow you to create **type-safe classes, methods, interfaces, or delegates** without specifying the actual data type upfront (means in advance it means **you don’t need to decide the data type when writing the class, method, or interface; it can be specified later when you actually use it**.).

**How it works:** You define a placeholder **<T>** for the data type, and when you use it, you decide the type (like int, string, or a custom class).

**Benefits**

* **Type-Safe** → Compiler checks the type at compile time, no runtime errors.

A computer screen shot of a check mark and a check mark

AI-generated content may be incorrect.

* **Performance** → Avoids boxing/unboxing.
* **Reusability** → One generic class/method works for all types.

A screen shot of a computer code

AI-generated content may be incorrect. A screen shot of a computer code

AI-generated content may be incorrect.

**Constraints** in generic control what types can be used in generics. Example ***where T : class*** **controls** the type allowed in Repository<T>. Only **reference types** (like string, Person, arrays) are allowed.

A screen shot of a computer program

AI-generated content may be incorrect.

Question 9: *How Garbage Collection works in the .NET CLR?*

In .NET, **Garbage Collection** is the process where the **CLR (Common Language Runtime)** automatically frees up memory by deleting objects that are no longer used by your program.

***When Does GC Run?***

* When system memory is low.
* When Gen 0 heap is full.
* When you explicitly force it with GC.Collect() (not recommended in real apps).

Why .Net Uses Genrations?

.NET uses generational garbage collection to improve performance by dividing objects based on their lifespan:

* **Generation 0:** New objects. Collected **most frequently**, because most short-lived objects die quickly.
* **Generation 1:** Objects that survived a Gen 0 collection. Collected **less often**.
* **Generation 2:** Long-lived objects that survived multiple collections. Collected **rarely**, only when memory pressure is high.

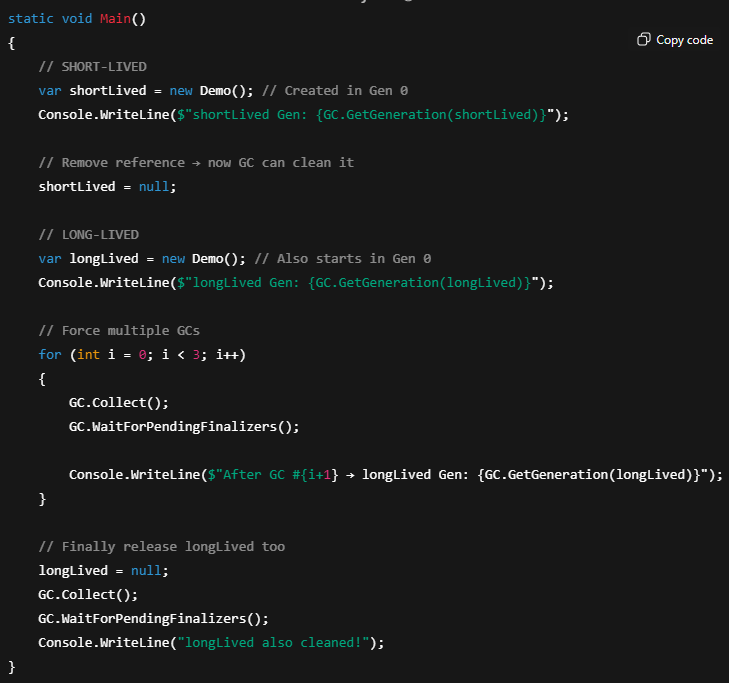
“Survived a collection” means: object was still in use when GC ran, so it was not deleted and moved to higher generation. If obj is **still in use** (reference exists), it **survives Gen 0 collection** → promoted to Gen 1. If we had done obj = null; → it would **not survive** → collected immediately.

A screen shot of a computer program

AI-generated content may be incorrect.

**How generations helps:** This selective scanning makes garbage collection much faster and more efficient.

* GC does not scan the entire heap every time.
* Normally it scans Gen 0 only → fast, because most objects die young.
* If Gen 0 collection is not enough → it scans Gen 1.
* If memory is still low → it scans Gen 2 (expensive).

A screenshot of a computer

AI-generated content may be incorrect.

The 3 Steps of GC: **Mark → Sweep → Compact**

**Mark -** GC starts from roots (local variables, static fields, CPU registers). It marks all objects that are still referenced. Here obj1 is marked as alive. obj2 is not referenced → will be garbage.

**A black background with white text

AI-generated content may be incorrect.**

**Sweep -** After marking, GC finds all unreferenced objects and removes them from memory.

**Compact -** After sweep, memory might have gaps. GC moves remaining objects together to avoid fragmentation and make allocation faster.

**Fragmentation:** Fragmentation is when memory has many small free blocks scattered. Even if total free memory is large, a program may fail to allocate a large object because there is no single continuous block available.

****

**Defragmentation** GC shifts live objects together to make memory continuous:

****

***“In .NET, Garbage Collection frees memory by deleting unused objects. Objects start in Gen 0, if they survive a collection, they move to Gen 1, then Gen 2. GC first collects Gen 0 when it gets full, so it doesn’t scan the whole heap every time. The process has three steps: Mark (find live objects in Gen 0 or the target generation), Sweep (remove dead ones), and (rearrange memory to remove fragmentation). This makes memory management fast and efficient.”***

*Question 10: Difference between Equals() vs == in C#.*

*Question 11:* *Difference between ref, out, and in parameters in C#.*

Question 10: *How to implement Caching (In-Memory, Distributed, Redis).*

# Threading/ Async Await / Task

Question 1: *What is a Thread?*

Question 2: *If an async method is executed, does it create a new thread under the hood?*

Question 3: *How do you stop two different threads from accessing the same method?*

Question 4: *Task vs ValueTask vs IAsyncEnumerable.*

Question 5: *What are async/await and how do they improve performance?*

# ASP.NET

Question 1: *What is Middleware? Can you name some built-in middleware?*

Middleware is **code that runs in the request pipeline** (every request passes through it) and can **process requests and responses**. Every incoming request passes through the middleware components in the order they are registered. Middleware can perform tasks like handling authentication, logging, routing, exception handling, or modifying the response before it reaches the client.

Examples: **Authentication, Authorization, StaticFiles, ExceptionHandling, Routing, CORS**.

Question 2: *What is an Anti-Forgery Token? Why do you use it?*

Question 3: *What is CORS?*

Question 4: *How do you transfer data from a Controller to a View?*

Question 5: *Difference between Controller vs ControllerBase.*

Question 6: *Explain filters and its types. Action Filters, Result Filters, Exception Filters.*

Question 7: *Difference between TempData, ViewData, ViewBag.*

Question 8: *What is Routing in MVC vs .NET Core.*

Question 9: *Explain Model Binding and Validation.*

*Question 10: How to implement global exception handling in ASPNET Core?*

*Question 11: How does Middleware pipeline work in .NET Core?*

*Question 12: Explain Microservices architecture and how to implement it in.NET Core.*

Question 13: *Explain MVC Architecture?*

*Question 14: How does Routing work?*

*Question 15: Role of Startup.cs?*

*Question 16: Tag Helpers vs HTML Helpers?*

*Question 17: What is a ViewModel?*

*Question 18: What is Kestrel Web Server?*

*Question 19: IActionResult vs ActionResult?*

*Question 20: Exception Handling approaches?*

*Question 21: Razor Pages vs MVC?*

*Question 22: Consuming Web APls via HttpClient?*

*Question 23: Configuration using appsettings.json & IConfiguration?*

# Web API

Question 1: *What is REST?*

Question 2: *If REST is stateless, why is it called a RESTful API?*

Question 3: *What are API Versioning strategies?*

# JWT

Question 1: *How do you validate a JWT without using built-in methods?*

Question 2: *Explain JWT Authentication & Authorization flow?*

Question 3: *Difference between Authentication vs Authorization.*