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?

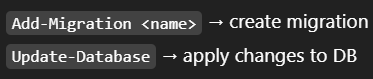
We handle transactions in EF Core using Database.BeginTransaction(). If all operations succeed, we commit; otherwise, we rollback to keep data safe. a transaction makes sure a group of database operations are all done successfully or none at all.

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?*

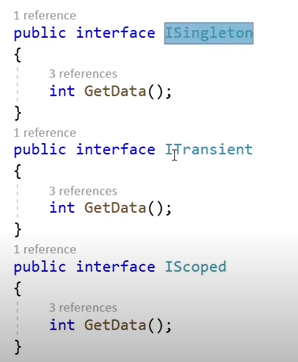
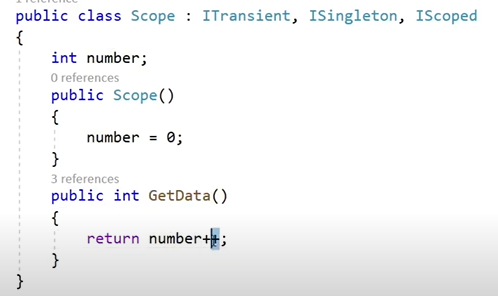
# General Questions

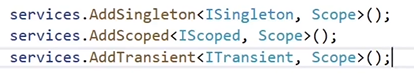
Question 1: *What is Dependency Injection? Why should you use it?*

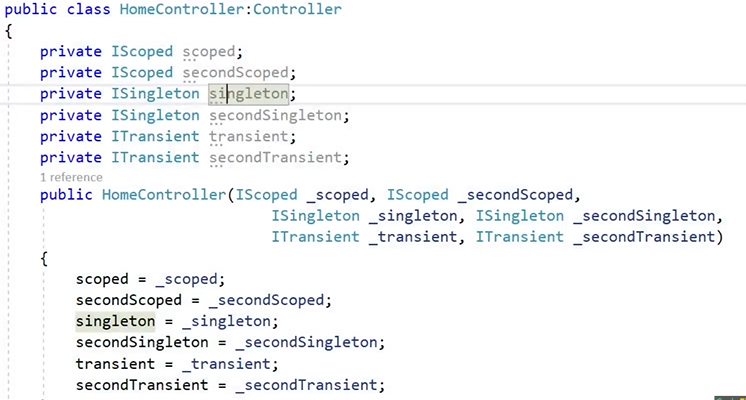
DI means instead of my class creating objects itself, I get those objects from outside. This keeps the code **loosely coupled (decouple), flexible, and easy to test.**

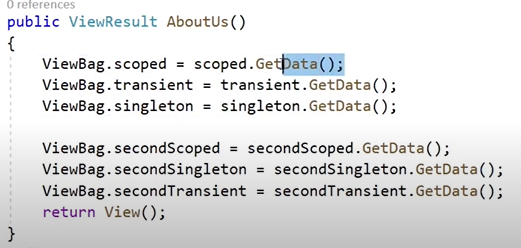
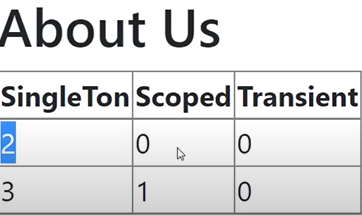
Lifetimes in DI

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.

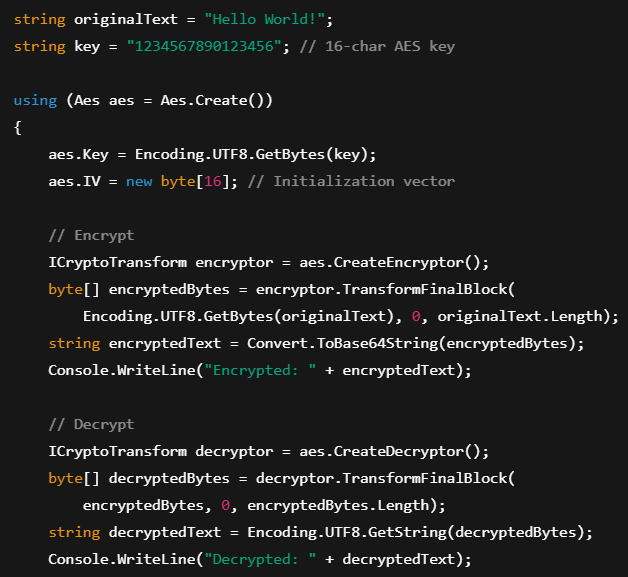
SHA-256 hashes passwords for secure storage, and AES encrypts messages so only someone with the key can read them.

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 –**

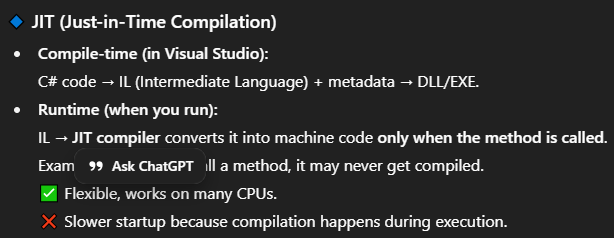
* came in 2016 as a modern, open-source, cross-platform version.
* Faster, because it loads only what you need (modular).
* Used for modern web apps, APIs, microservices, and console apps.

**.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.

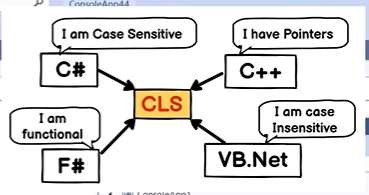
**CTS** As .net provide multiple language support so CTS (Common type system) ensures that data types defined in two different languages get compiled to a common data type.

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.

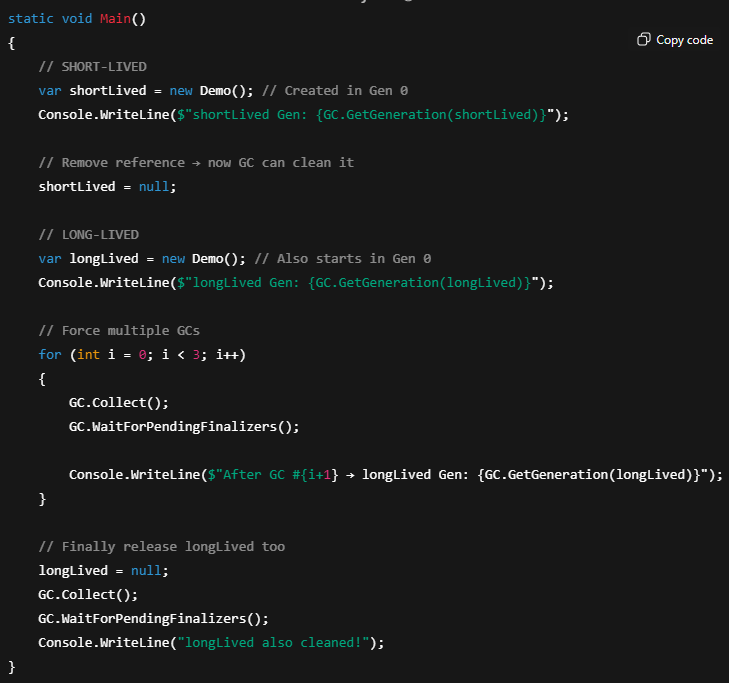
“Survived a collection” means: object was still in use when GC ran, so it was not deleted and moved to higher generation. If obj is **still in use** (reference exists), it **survives Gen 0 collection** → promoted to Gen 1. If we had done obj = null; → it would **not survive** → collected immediately.

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#.*

* **Equals()** → *is* ***virtual method*** *from object class. By default, checks reference, but can be* ***overridden*** *to check values.* (In C#, **every class implicitly inherits from System.Object** (if you don’t specify any base class).
* **==** → **operator**. By default, checks reference for reference types, value for value types. Can be **overloaded**.
* ***“String special case*** *→ Both Equals() and == are already implemented to compare content (characters).”*
* ***Custom classes*** *→ You must override Equals() and operator overload == if you want value-based comparison.*

A screenshot of a computer

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

AI-generated content may be incorrect.

A computer screen shot of a program code

AI-generated content may be incorrect.

A computer screen with colorful text

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

AI-generated content may be incorrect.

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

* **ref** = I already have a value, I want method to **update it**.
* **out** = I don’t have a value, method will **create/return it**.
* **in** = I **already have a value**, I want the method to **use it read-only** (cannot modify).

A computer screen with colorful text

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

AI-generated content may be incorrect.

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? Ways to create middlewares?*

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.

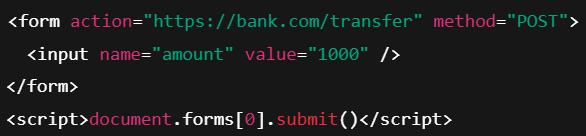
Examples: **Authentication, Authorization, StaticFiles, ExceptionHandling, Routing, CORS**.

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

* An **Anti-Forgery Token** is a **security feature** in ASP.NET used to protect against **CSRF (Cross-Site Request Forgery) attacks**.
* ***“An Anti-Forgery Token is a security feature in ASP.NET that ensures a request comes from the user’s own browser, not from another site(not a malicious site), by validating a token stored in both the form (or header) and a cookie.”***

**What is CSRF -** CSRF (Cross-Site Request Forgery/ XSRF) is when a malicious site tricks a logged-in user’s browser into sending a request (with the user’s cookies) to your site, performing actions the user didn’t intend.

1. User logs into bank.com — browser holds an auth cookie.
2. Attacker hosts evil.com with HTML like:



1. When the user visits evil.com, the browser auto-submits the form to bank.com **and the browser attaches the bank cookie automatically.**

***Question evil.com cannot set or append cookies for bank.com. How CSRF still happens?***

* Even though evil.com **cannot set bank.com cookies**, it **can make the browser send a request to bank.com.**
* The browser **automatically attaches bank.com cookies** because the request is for bank.com.

***Question Could a malicious site generate an antiforgery token using @Html.AntiForgeryToken() and then include a hidden field so the browser sends a valid token-cookie pair to the real site?***

No — @Html.AntiForgeryToken() on evil.com cannot grab or attach bank.com’s cookie/token.

* **Same‑Origin Policy:** JavaScript/pages on evil.com cannot read cookies or DOM from bank.com.
* **Different token issuance:** @Html.AntiForgeryToken() on evil.com will generate a token tied to evil.com (signed with evil.com’s keys). That token will never match what bank.com expects.
* **When a user visits evil.com and evil.com posts to bank.com:** the browser will include the bank.com auth cookie automatically — but it will **not** include any bank.com token from evil.com’s page (because there isn’t one the attacker can read/generate dom property). The hidden field on evil.com is useless to bank.com, so bank.com rejects the request.

How AntiForgery works under the hood?

**Incase of ASP.NET Razor View:-**

ASP.NET gives a **secret token** for every form/request. When ASP.NET generates a token, it actually creates **two linked token**:

1. **One is stored in a hidden field in the form (e.g. \_\_RequestVerificationToken)..**
2. **One is stored in a cookie (on your browser, domain-locked → attacker site can’t read it).**

“*When a server renders a form, it gives two connected tokens: one in a cookie and one hidden in the form. On submit, the server checks they match and are valid. An attacker can’t read the cookie or fake the signature, so fake requests are blocked.”*

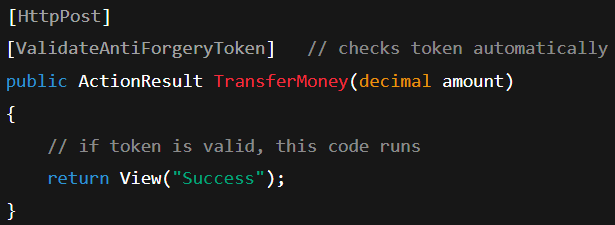
A screenshot of a computer program

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

AI-generated content may be incorrect.

**Browser Cookie **

**On POST request**

* Server reads hidden field token. Server reads cookie token.
* Uses machineKey/DataProtection keys to validate. If they match (were issued together), request is valid.
* 

**An anti-forgery token with a salt is very secure because the token is random.**

A screenshot of a computer program

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

AI-generated content may be incorrect.

With Angular

* **Server issues a cookie token** (when Angular app loads or user logs in (maybe /api/home or /login)). Example: cookie name XSRF-TOKEN. Browser stores it automatically.

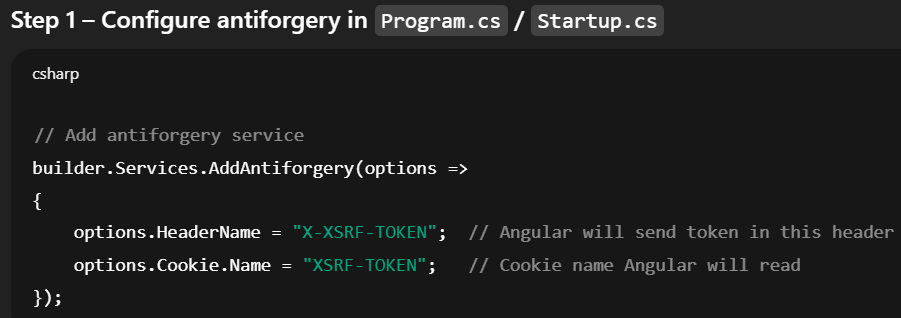


* **Angular’s HttpClient automatically reads this cookie** and sends it in a **request header** (X-XSRF-TOKEN).



* **Server receives header + cookie** → validates just like before. If header == cookie token match → request allowed.

**Setup in ASP.NET Core backend**

Angular side

Angular **already has built-in support** for XSRF:

* By default, Angular looks for a cookie named XSRF-TOKEN.
* It will automatically add a header X-XSRF-TOKEN for every HttpClient request.

A computer screen shot of code

AI-generated content may be incorrect.

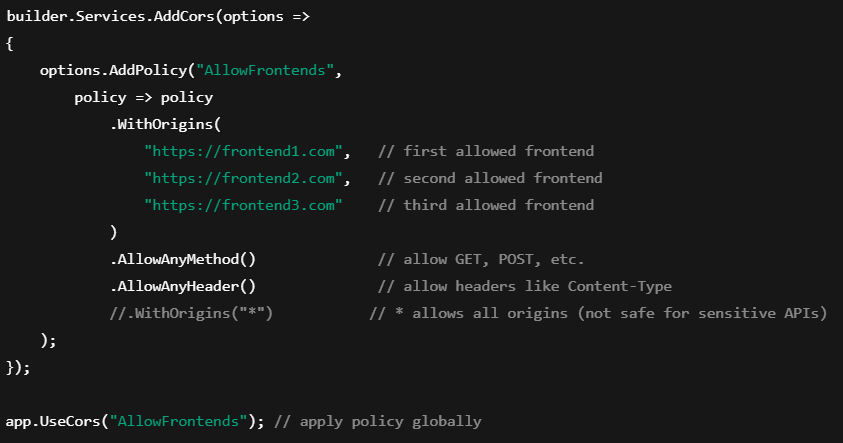
*“In Razor pages, cookie token and form token are two different values that must match on the server. In Angular + ASP.NET Core, the cookie and header both carry the same token value. Angular automatically copies the cookie into a header, and the server validates them.”*

Question 3: *What is CORS?*

***“CORS is a browser security feature that controls which websites can call your API. The browser checks with the server first (sometimes sending a preflight OPTIONS request) and only allows the request if the server permits it.”***

* CORS, or Cross-Origin Resource Sharing, solves a browser security problem. Browsers block requests from one website to another by default, to prevent unauthorized or malicious access. For example, if my frontend at ***frontend.com*** tries to call an API at ***api.com***, the browser blocks it unless the API explicitly allows it.
* When a frontend makes a cross-origin request, the browser first checks if the API allows it. For some requests like POST, it sends a ***preflight OPTIONS request***. The API responds with headers like Access-Control-Allow-Origin to tell the browser which origins and methods are allowed. If the origin is allowed, the browser lets the request go through. If not, it blocks the request.
* CORS is enforced by the browser, so it only affects frontend calls like in SPAs (Angular, React, Vue). Server-to-server communication doesn’t need CORS because browsers aren’t involved.
* For example, in ASP.NET Core, we can whitelist an origin like this:

1. \* = allows all origins, but not safe for APIs with sensitive data or credentials.
2. Always prefer whitelisting trusted frontend URLs.



**Problem it solves and How it works step by step**

1. You visit https://myfrontend.com and try to call an API at <https://myapi.com>.
2. **Browser sees request to another domain** → “Hmm, this is cross-origin.”
3. **Browser sends a “preflight” request** (OPTIONS) to ask the server: “Hey, can I come with GET/POST from my site?”
4. **Server responds with allowed methods/origins**.
5. **Browser checks response** → if allowed, sends the actual request.
6. If **server doesn’t allow**, browser **blocks request** for security.

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

We can transfer data from Controller → View in **four main ways:**

1. **Using ViewBag** *(Dynamic property)*

A black screen with white text

AI-generated content may be incorrect.

1. **Using ViewData** *(Dictionary type)*

A black screen with text and symbols

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

AI-generated content may be incorrect.

1. **Using TempData** *(For redirect scenarios)*

A computer screen with text

AI-generated content may be incorrect.

1. **Using a Strongly Typed Model** *(Best & preferred way)*

A black screen with white text

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

AI-generated content may be incorrect.

Question 5: *Difference between Controller vs ControllerBase.*

ControllerBase - **ControllerBase** is the **base class** for all **API controllers** in ASP.NET Core.  
It provides the **core features** needed to handle HTTP requests — like Ok(), BadRequest(), NotFound(), ModelState, and Request/Response — **but no view-related features**.

A screenshot of a computer program

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

AI-generated content may be incorrect.

**Controller - Controller inherits from ControllerBase and adds view-support features — like View(), PartialView(), and ViewData — for returning HTML pages.**

Controller = ControllerBase + + **ViewBag/ViewData/TempData + View/PartialView rendering**

A screenshot of a computer program

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

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

AI-generated content may be incorrect.

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

A **filter** is a piece of code that runs **before or after an action method** in ASP.NET MVC/Core.  
Filters let you **inject reusable logic** like authentication, authorization, logging, caching, or exception handling **without changing the action method itself**.

1. ***Authorization Filter -*** Checks if the user is allowed to do something.

Built-in filters:

* + [Authorize] → restrict access to authenticated users.
  + [AllowAnonymous] → allow access even if [Authorize] is applied globally.
  + [Authorize(Roles="Admin")] → role-based access

A black background with white text

AI-generated content may be incorrect.

1. ***Action Filter*** - Runs code **before and after** the action method,use for Logging.

***[OutputCache]*** → cache action results. [OutputCache(Duration=60)] caches the **entire HTTP response** of the action for 60 seconds.During that period, **all requests get the same response** without executing the controller again.After 60 seconds, the cache expires and the controller executes again to refresh the cache.This is especially useful for expensive actions like DB queries, API calls, or heavy computations.

**Type:** **Both Action Filter and Result Filter**

* + - **Action Filter:** Can short-circuit the action — if cached response exists, the action **doesn’t even run**.
    - **Result Filter:** Can modify or cache the **result before sending it to client**.

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.

A screenshot of a computer

AI-generated content may be incorrect.

***[ValidateAntiForgeryToken]*** → prevent CSRF attacks.

A screenshot of a computer program

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

AI-generated content may be incorrect.

1. ***Result Filter*** - Runs **before sending the response** to the user.
   1. [OutputCache] (also works here)
   2. [ContentType]

A computer screen with text

AI-generated content may be incorrect.

1. ***Exception Filter*** - Handles errors if something goes wrong in the action.
   1. [HandleError] → catch exceptions and show friendly error pages.

A screenshot of a computer screen

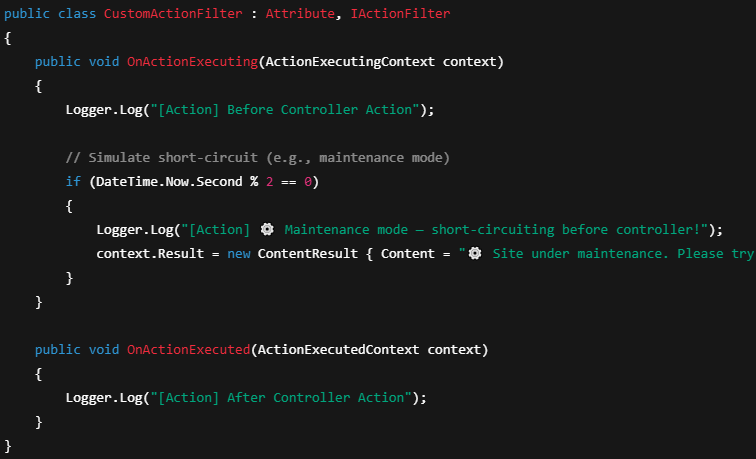
AI-generated content may be incorrect.

1. ***Resource Filter*** - Runs **very early**, even before the action starts.

A computer screen shot of a program

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

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

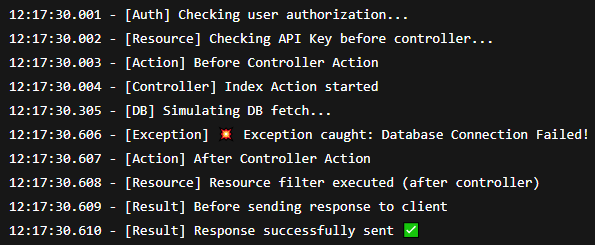
AI-generated content may be incorrect.

A screen shot of a computer program

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

AI-generated content may be incorrect.

Exception Filter Short-Circuit GET /api/demo/index?apikey=123



Short-circuiting means the filter ends the request early, so the action method never runs.  
When a filter sets context.Result, ASP.NET Core sees it and **skips running the action method**, returning that result immediately. A black background with white text

AI-generated content may be incorrect.

Question 7: *What is [ApiController] Attribute?*

The [ApiController] attribute is used in ASP.NET Core to make building Web APIs easier.  
It automatically handles model validation, infers binding sources, and returns standardized error responses. When applied, you don’t need to manually check ModelState.IsValid.  
It tells ASP.NET Core, “this controller is for API calls, not MVC views.”

Main Features of [ApiController]

1. ***Automatic 400 responses***  - If ModelState is invalid → it automatically returns **400 Bad Request**, no need to write if(!ModelState.IsValid) manually.

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. A screen shot of a computer

AI-generated content may be incorrect.

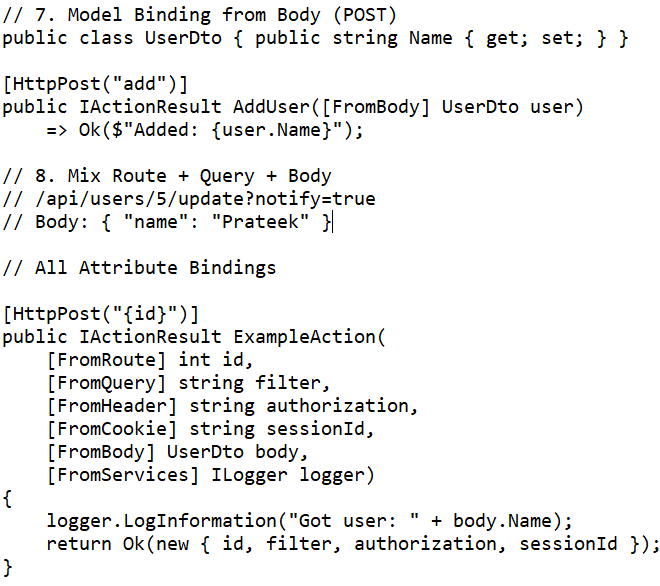
1. ***Binding source inference-*** ASP.NET Core automatically figures out where to read values from (e.g., [FromBody], [FromQuery], [FromRoute]). Example: for complex types → it assumes **body**, for simple types → **query or route**. ASP.NET Core automatically infers:

* UserModel model → comes from body ([FromBody])
* int page → comes from query ([FromQuery])

Without [ApiController], you’d have to decorate explicitly:



A screenshot of a computer program

AI-generated content may be incorrect. 

1. ***Adds ModelStateInvalidFilter-*** **Type:** IActionFilter. After **model binding**, before action executes. You don’t need to write if(!ModelState.IsValid) return BadRequest();.

What it does A black screen with white text

AI-generated content may be incorrect.

1. ***Standardizes Error Responses-*** Instead of returning raw ModelState, it returns a standardized JSON object with fields like errors, status, and traceId: ModelStateInvalidFilter uses ValidationProblemDetails object.

***A computer screen with white text

AI-generated content may be incorrect.***

1. ***Automatic HTTP 400 for Parameter Binding Failures***- API expects an int id in route, but user sends "abc". A black background with red and yellow text

   AI-generated content may be incorrect.

A screenshot of a computer program

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

AI-generated content may be incorrect.

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

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

*Routing in ASP.NET Core means deciding which controller and action will handle a request. There are two types — attribute and conventional routing. In attribute routing, we put route paths directly on controller methods using attributes like [Route] or [HttpGet]. So, each route is fixed and directly points to that method — it’s mostly used in Web APIs. In conventional routing, we define a common pattern like {controller}/{action}/{id?} in Program.cs, and ASP.NET Core matches the URL and then figures out which controller and action to call. The main difference is — attribute routing is explicit and already knows which method to run, while conventional routing follows a pattern and decides at runtime.*

**Common Routing Flow**

When a request comes, e.g., products/details/5

* **Kestrel** receives request → passes to middleware pipeline.
* **Middleware pipeline** processes request (UseRouting(), UseAuthentication(), UseAuthorization()).
* UseRouting() → **matches URL against routing table** → fills HttpContext.GetRouteData().

1. **At Startup — Builds Routing Table-** When your app starts, ASP.NET Core scans for all possible routes.
2. **At Runtime — Matches Incoming URL-** The Routing Middleware (UseRouting()) goes through this table. It finds the first matching pattern and extract value.

 A screen shot of text

AI-generated content may be incorrect.

* Route Data Saved in HttpContext - Now ASP.NET Core attaches this route data to the current HTTP request —that’s why it’s called **HttpContext.** **You can access it in code like this:**

A black background with white text

AI-generated content may be incorrect.

* UseEndpoints() → executes the matched endpoint.
* MVC **creates controller instance via DI** → executes action → returns response.

Attribute Routing (MapControllers())

A computer screen with white text

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

AI-generated content may be incorrect.

A screenshot of a black screen

AI-generated content may be incorrect.

* UseRouting() matches /api/products/5 → finds endpoint → stores route values id=5.
* UseEndpoints() executes **precompiled delegate** → MVC creates ProductsController → calls Get(5).
* **No runtime lookup** needed; endpoint already points to the method.

Conventional Routing (MapControllerRoute())

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.

A screenshot of a phone

AI-generated content may be incorrect.

A screen shot of a computer

AI-generated content may be incorrect.

**Mixed Routing** - ASP.NET Core merges attribute and conventional routes into the same routing table. Attribute route takes priority because it is explicit.

A screen shot of a computer code

AI-generated content may be incorrect.A screenshot of a black screen

AI-generated content may be incorrect.

A screenshot of a computer program

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

AI-generated content may be incorrect.

*Question 14: How does Routing work?*

1. **Kestrel** receives the request.
2. **HttpContext** is created and passed into the **middleware pipeline**.
3. The **Routing Middleware** (UseRouting) checks all registered routes in routing table and tries to **match the URL**.
4. Once matched, it **stores route data** (like controller, action, id) in HttpContext.
5. Then **UseEndpoints** runs, which **executes the matched endpoint** (e.g., your controller action).

A screen shot of a computer program

AI-generated content may be incorrect.

Question 9: *Explain Model Binding and Validation.*

***Model Binding*** - Model Binding is the process in ASP.NET Core that **takes data from the HTTP request** (like route, query string, form, or body) and **creates .NET objects or parameter values** for your controller action.*Binding Sources: [FromRoute], [FromQuery], [FromBody], [FromForm]*

A screen shot of a computer program

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

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

AI-generated content may be incorrect.

***Validation*** - Once model binding is done, ASP.NET Core can automatically validate the model using **Validation Attributes** [Required], [StringLength], [Range], [EmailAddress].And fill modelstate if we have use apicontroller it will automatically through erorr with 400 message.

A screenshot of a computer code

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

AI-generated content may be incorrect.

If you are using [ApiController], it automatically returns:

A black screen with white text

AI-generated content may be incorrect.

A close-up of a computer code

AI-generated content may be incorrect.

All Attribute with custom validation message and custom validation of age:

To write custom validation, **inherit from ValidationAttribute for single-property rules or implement IValidatableObject on the model for multi-property/complex validation**.

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 program

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

AI-generated content may be incorrect.*

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

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

*In .NET Core, the middleware pipeline processes each HTTP request step-by-step in the order added, where each middleware can handle, modify, pass, or short-circuit the request before sending the response back.*

* The middleware pipeline is built inside Program.cs using methods like Use(), Run(), and Map().
* Each middleware component receives an HttpContext and a RequestDelegate (next).
* The pipeline is **executed in the same order it’s registered**.

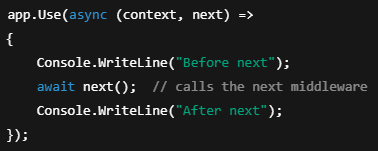
Execution Flow

* **Request enters Kestrel** → HttpContext created.
* **Pipeline executes sequentially** — each middleware either calls next() or ends the flow.
* **Terminal middleware** (Run) generates the final response.
* **Response flows back** — all “after next()” code executes in reverse order.

**Middleware Methods in ASP.NET Core — Use(), Run(), and Map()**

**Use() — Add Middleware (with next)**

Used to **add middleware** that can do something *before* and *after* the next middleware runs.



**Run() — Terminal Middleware (ends pipeline)**

Used to define the **final middleware** that produces the **response** and **does not call next()**.If you put **multiple Run() middlewares**, only the **first one that matches** the request will execute — **others are ignored** because the pipeline stops. **Only “First Run” executes.**

A screen shot of a computer program

AI-generated content may be incorrect.

Map() — Branch the Pipeline

Used to **branch** the middleware pipeline based on the request **path**. Each branch has its **own mini-pipeline**. If no Map() matches, the request continues down the **main pipeline**.

A computer screen shot of a program code

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

AI-generated content may be incorrect.

A screenshot of a computer program

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

AI-generated content may be incorrect.

A screenshot of a computer program

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

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

AI-generated content may be incorrect.

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

*Question 15: What is HttpContext?*

***When a request enters Kestrel, ASP.NET Core creates a new HttpContext object that represents that single HTTP request and response. It travels through the middleware pipeline and gives access to everything about the request — like URL, headers, user, cookies, and response — so middleware and controllers can read or modify it before it’s sent back to the client.***

For example, using HttpContext, we can get the logged-in user or read request data inside our controller or middleware.

*A screenshot of a computer program

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

AI-generated content may be incorrect.*

Question 13: *How do you start a Web API or MVC project in .NET Core?*

Minimal setup to start projects

A screenshot of a computer program

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

AI-generated content may be incorrect.

WebApplicationBuilder (builder)

* Prepares the app **before it runs**.
* Holds **services (DI)**, **configuration**, **logging**, and **host settings**.
* You register everything your app will need here.
* Think of it as the **blueprint** of your application.

WebApplication (app)

* Created by builder.Build().
* Represents the **running application**.
* You attach **middleware** and endpoints to it.
* app.Run() starts the server to **listen for HTTP requests**.
* Think of it as the **constructed house ready to serve requests**.

app.Run() starts the server loop, listening for HTTP requests and passing each request into the pipeline with HttpContext.

builder = blueprint/setup, app = running application serving requests.

Question 13: *Explain MVC Architecture?*

*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.*