Here are detailed answers to hard-level .NET interview questions:

👷💏.NET Architecture & CLR Internals

1. CLR Memory & GC: - CLR divides memory into stack and managed heap. - GC uses generations (Gen 0, 1, 2) to manage memory. - To reduce GC pressure: use object pooling, avoid large object allocations, and minimize allocations in high-frequency code paths.

2. Value vs Reference Types: - Value types are stored on the stack or inline (structs), reference types on the heap (class). - Boxing wraps a value type in a reference, unboxing reverses it. This has a performance cost due to heap allocation and GC.

3. Async without Await: - The method runs synchronously until it hits an awaitable. No compiler-generated state machine is created unless `await` is used. - Can lead to unexpected behavior.

4. JIT vs AOT: - JIT compiles IL to native code at runtime. - AOT compiles IL to native code at publish-time (used in NativeAOT). - Tiered compilation balances fast startup (Tier 0) with optimized code (Tier 1).

5. Task vs Thread vs ThreadPool: - `Thread`: Dedicated OS thread. Heavyweight. - `ThreadPool`: Managed threads reused for efficiency. - `Task`: Abstraction over ThreadPool with scheduler support. Ideal for async and parallel work.

👨‍💻ASP.NET Core & Web APIs

6. Request Pipeline: - Configured via `Startup.cs` or `Program.cs`. - Middleware are invoked in order. Examples: Authentication, Logging, Routing.

7. Rate Limiting: - Use `AspNetCoreRateLimit` or custom middleware. - Consider Redis for distributed rate limiting.

8. Model Binding vs Validation: - Binding: Maps HTTP data to C# objects. - Validation: Uses data annotations or `IValidatableObject`. - You can write custom binders or filters.

9. Global Exception Handling: - Use `UseExceptionHandler`, `DeveloperExceptionPage`, or custom middleware. - Log using `ILogger`.

10. Filters vs Middleware: - Filters (Action, Result, Exception): Tied to MVC pipeline. - Middleware: Lower level, reusable across apps.

🏊Security

11. API Security Best Practices: - Use HTTPS, validate inputs, sanitize output. - Use authentication (OAuth, JWT). - Prevent CSRF via antiforgery tokens. - Prevent XSS via HTML encoding.

12. OAuth2 vs OpenID Connect: - OAuth2: Authorization. - OpenID Connect: Identity layer on top of OAuth2. - Use JWT Bearer Authentication middleware.

13. DataProtection APIs: - Encrypt data like cookies, tokens. - Supports key rotation, can store keys in Azure, Redis.

14. Role & Policy Authorization: - `[Authorize(Roles = "Admin")]` - Policies via `AddAuthorization` and `AuthorizationHandler`.

👦Multithreading & Async

15. ConfigureAwait(false): - Prevents capturing the `SynchronizationContext`, avoiding deadlocks in UI apps. - In ASP.NET Core, it's often unnecessary because there's no context.

16. ThreadPool & TaskScheduler: - ThreadPool manages a pool of threads for reuse. - `TaskScheduler` determines how tasks are scheduled (e.g., concurrent vs sequential).

17. Parallel vs WhenAll: - `Parallel.ForEach`: CPU-bound. - `Task.WhenAll`: I/O-bound.

18. Avoiding Deadlocks: - Don’t block async code (`.Result`, `.Wait()`). - Use async all the way.

👬EF Core

19. EF Core Performance Pitfalls: - N+1 queries, lazy loading abuse. - Use eager loading (`Include`), `AsNoTracking`, batching.

20. Change Tracking: - Tracks entity state for updates. - Use `AsNoTracking()` for read-only queries to boost performance.

21. AsNoTracking vs ExecuteSqlRaw: - `AsNoTracking`: Read-only, no overhead. - `ExecuteSqlRaw`: Raw SQL queries; bypass LINQ.

22. Migrations: - `Add-Migration`, `Update-Database`. - Avoid auto migrations in production. - Use idempotent scripts.

👧🏻Design Patterns & Principles

23. DI in .NET Core: - Built-in container with `AddSingleton`, `AddScoped`, `AddTransient`. - Autofac allows child containers, property injection.

24. Design Patterns: - Factory: Create objects (e.g., ILoggerFactory). - Strategy: Swap logic at runtime (e.g., payment gateways). - Decorator: Extend behavior (e.g., logging service).

25. SOLID Example: - Violation: A class that validates, saves, and logs data. - Fix: Split into `IValidator`, `IRepository`, `ILogger`.

🤱🏾Tricky Concepts

26. Hosted Services Fail Silently: - Exception in background service isn't reported. - Use `try/catch`, log, use `IHostApplicationLifetime`.

27. Span<T> & Memory<T>: - Stack-only slices of memory. - Used in high-performance code (e.g., file parsing).

28. Caching Strategies: - MemoryCache: Fast, per instance. - Redis: Distributed. - Response caching: Cache entire response.

29. API Versioning: - URI (/v1/api/...), headers, or query strings. - Use `Microsoft.AspNetCore.Mvc.Versioning`.

30. Plugin Architecture: - Load assemblies dynamically via `AssemblyLoadContext`. - Use interfaces/ contracts for isolation.

Let me know if you want: - Code examples - Cloud-native (Azure/AWS) .NET questions - Real-time SignalR or Blazor questions - System Design for .NET apps