Here is a categorized list of **interview questions** tailored for a **10+ years experienced .NET Full Stack Developer cum Architect**. These cover **backend, frontend, architecture, DevOps, cloud, and soft skills**, reflecting the responsibilities of a technical leader or solution architect.

## 🔷 **.NET Backend (C#, ASP.NET Core)**

1. **How do you implement Clean Architecture in a .NET Core application? Explain with folder structure.**
2. **What are the advantages of using Dependency Injection in .NET Core?**
3. **Explain how you handle exception handling and logging in a production-grade API.**
4. **What is Middleware in ASP.NET Core? Can you build a custom middleware?**
5. **Explain the difference between Task, async/await, and thread in C#.**
6. **What’s your strategy for securing Web APIs (JWT, OAuth2, etc.)?**
7. **How do you handle performance bottlenecks in your API?**
8. **What is your approach for versioning Web APIs in ASP.NET Core?**

## 🔷 **Entity Framework & Data Access**

1. **Explain Code First vs Database First approach in EF Core. Which one do you prefer and why?**
2. **What strategies do you follow to improve EF Core query performance?**
3. **How do you implement Unit of Work and Repository patterns in EF Core?**
4. **What are shadow properties and global query filters in EF Core?**

## 🔷 **Frontend (Angular/React + TypeScript)**

1. **How do you structure a large-scale Angular or React application?**
2. **How do you handle state management in your front-end apps? (e.g., NgRx, Redux, Context API)**
3. **Explain the component lifecycle in Angular/React.**
4. **How do you implement lazy loading and module federation?**
5. **How do you ensure accessibility (a11y) and performance in frontend applications?**

## 🔷 **Architecture & Design**

1. **Can you explain how you designed a recent enterprise-scale system end to end?**
2. **What is your approach to Domain-Driven Design (DDD)? Give a real scenario.**
3. **Explain the difference between monolith, microservices, and modular monolith architecture.**
4. **How do you make architecture decisions? Can you give an example of a trade-off you made?**
5. **What are some of the architectural patterns you've implemented (CQRS, Event Sourcing, etc.)?**

## 🔷 **Design Patterns**

1. **Which design patterns have you frequently used in your solutions? Why?**
2. **How have you implemented Singleton, Factory, Strategy, or Mediator pattern in C#?**
3. **Explain a real-world scenario where you used the Repository + Unit of Work pattern.**
4. **What is the difference between Adapter and Decorator pattern with code examples in .NET?**

## 🔷 **Microservices**

1. **How do you design microservices communication: REST vs messaging (e.g., RabbitMQ, Kafka)?**
2. **How do you handle distributed transactions or data consistency in microservices?**
3. **What’s your strategy for service discovery and API Gateway (e.g., Ocelot, YARP)?**
4. **How do you ensure resilience and fault tolerance in microservices? (e.g., Polly, retries, circuit breaker)**

## 🔷 **Azure / Cloud-Native**

1. **How do you deploy .NET apps in Azure? What services do you commonly use?**
2. **Can you explain the design of a scalable architecture using Azure App Service, Azure Functions, and Azure Service Bus?**
3. **How do you manage secrets and configurations securely in Azure?**
4. **What is your CI/CD strategy using Azure DevOps/GitHub Actions?**

## 🔷 **DevOps, CI/CD, Testing**

1. **How do you automate builds, testing, and deployments for a .NET solution?**
2. **What tools and processes do you use for code quality checks and static code analysis?**
3. **How do you write unit and integration tests for controllers and services?**
4. **What is your approach to containerization using Docker with .NET Core apps?**

## 🔷 **Performance, Scalability & Security**

1. **How do you identify and resolve memory leaks or GC pressure in .NET Core?**
2. **How do you handle high throughput and horizontal scalability in your architecture?**
3. **What are your top security practices in .NET and frontend apps (OWASP, CSRF, XSS)?**

## 🔷 **Soft Skills & Leadership**

1. **How do you handle conflict in architectural decisions within your team?**
2. **Can you describe a situation where your system failed in production? How did you handle it?**
3. **How do you mentor junior developers or onboard new team members?**
4. **How do you estimate complex features and manage deadlines?**
5. **How do you communicate technical decisions to non-technical stakeholders?**

Would you like answers for any of these questions? Or want these in **mock interview format**, **quiz format**, or with **C# code** for design patterns and architecture questions?

Here is a categorized list of **interview questions** tailored for a **10+ years experienced .NET Full Stack Developer cum Architect**. These cover **backend, frontend, architecture, DevOps, cloud, and soft skills**, reflecting the responsibilities of a technical leader or solution architect.

## 🔷 **.NET Backend (C#, ASP.NET Core)**

1. **How do you implement Clean Architecture in a .NET Core application? Explain with folder structure.**
2. **What are the advantages of using Dependency Injection in .NET Core?**
3. **Explain how you handle exception handling and logging in a production-grade API.**
4. **What is Middleware in ASP.NET Core? Can you build a custom middleware?**
5. **Explain the difference between Task, async/await, and thread in C#.**
6. **What’s your strategy for securing Web APIs (JWT, OAuth2, etc.)?**
7. **How do you handle performance bottlenecks in your API?**
8. **What is your approach for versioning Web APIs in ASP.NET Core?**

## 🔷 **Entity Framework & Data Access**

1. **Explain Code First vs Database First approach in EF Core. Which one do you prefer and why?**
2. **What strategies do you follow to improve EF Core query performance?**
3. **How do you implement Unit of Work and Repository patterns in EF Core?**
4. **What are shadow properties and global query filters in EF Core?**

## 🔷 **Frontend (Angular/React + TypeScript)**

1. **How do you structure a large-scale Angular or React application?**
2. **How do you handle state management in your front-end apps? (e.g., NgRx, Redux, Context API)**
3. **Explain the component lifecycle in Angular/React.**
4. **How do you implement lazy loading and module federation?**
5. **How do you ensure accessibility (a11y) and performance in frontend applications?**

## 🔷 **Architecture & Design**

1. **Can you explain how you designed a recent enterprise-scale system end to end?**
2. **What is your approach to Domain-Driven Design (DDD)? Give a real scenario.**
3. **Explain the difference between monolith, microservices, and modular monolith architecture.**
4. **How do you make architecture decisions? Can you give an example of a trade-off you made?**
5. **What are some of the architectural patterns you've implemented (CQRS, Event Sourcing, etc.)?**

## 🔷 **Design Patterns**

1. **Which design patterns have you frequently used in your solutions? Why?**
2. **How have you implemented Singleton, Factory, Strategy, or Mediator pattern in C#?**
3. **Explain a real-world scenario where you used the Repository + Unit of Work pattern.**
4. **What is the difference between Adapter and Decorator pattern with code examples in .NET?**

## 🔷 **Microservices**

1. **How do you design microservices communication: REST vs messaging (e.g., RabbitMQ, Kafka)?**
2. **How do you handle distributed transactions or data consistency in microservices?**
3. **What’s your strategy for service discovery and API Gateway (e.g., Ocelot, YARP)?**
4. **How do you ensure resilience and fault tolerance in microservices? (e.g., Polly, retries, circuit breaker)**

## 🔷 **Azure / Cloud-Native**

1. **How do you deploy .NET apps in Azure? What services do you commonly use?**
2. **Can you explain the design of a scalable architecture using Azure App Service, Azure Functions, and Azure Service Bus?**
3. **How do you manage secrets and configurations securely in Azure?**
4. **What is your CI/CD strategy using Azure DevOps/GitHub Actions?**

## 🔷 **DevOps, CI/CD, Testing**

1. **How do you automate builds, testing, and deployments for a .NET solution?**
2. **What tools and processes do you use for code quality checks and static code analysis?**
3. **How do you write unit and integration tests for controllers and services?**
4. **What is your approach to containerization using Docker with .NET Core apps?**

## 🔷 **Performance, Scalability & Security**

1. **How do you identify and resolve memory leaks or GC pressure in .NET Core?**
2. **How do you handle high throughput and horizontal scalability in your architecture?**
3. **What are your top security practices in .NET and frontend apps (OWASP, CSRF, XSS)?**

## 🔷 **Soft Skills & Leadership**

1. **How do you handle conflict in architectural decisions within your team?**
2. **Can you describe a situation where your system failed in production? How did you handle it?**
3. **How do you mentor junior developers or onboard new team members?**
4. **How do you estimate complex features and manage deadlines?**
5. **How do you communicate technical decisions to non-technical stakeholders?**

Would you like answers for any of these questions? Or want these in **mock interview format**, **quiz format**, or with **C# code** for design patterns and architecture questions?

Perfect! We’ll proceed **section by section**, displaying each one here with **minimum 15 questions and expected answers**. As requested, **React** will be emphasized more, and **Angular** will have a separate section. **Azure** will be the main cloud focus.

### 🔷 Section 1: .NET Backend (C#, ASP.NET Core)

Here are 15+ high-quality backend interview questions with expected answers:

#### **1. What are the key benefits of using ASP.NET Core over traditional ASP.NET?**

**Answer:**

* Cross-platform support (Windows, Linux, macOS)
* Modular architecture via middleware
* Improved performance and lightweight
* Built-in dependency injection
* Unified programming model for MVC/Web API
* Side-by-side versioning

#### **2. Explain Middleware in ASP.NET Core.**

**Answer:** Middleware is software that's assembled into an app pipeline to handle requests and responses. Examples: authentication, logging, exception handling. Middleware runs in sequence and can short-circuit or pass the request to the next.

#### **3. How do you register and use a service in ASP.NET Core's built-in DI container?**

**Answer:**

services.AddScoped<IMyService, MyService>(); // in Startup.cs

Then inject in constructor:

public class MyController : ControllerBase {

private readonly IMyService \_service;

public MyController(IMyService service) {

\_service = service;

}

}

#### **4. What is the difference between AddTransient, AddScoped, and AddSingleton?**

**Answer:**

* Transient: new instance every time
* Scoped: one instance per request
* Singleton: single instance for entire app lifetime

#### **5. How do you secure Web APIs using JWT in ASP.NET Core?**

**Answer:**

* Use Microsoft.AspNetCore.Authentication.JwtBearer
* Configure authentication in Startup.cs
* Add [Authorize] attribute to endpoints
* Validate tokens using symmetric or asymmetric keys

#### **6. What is Model Binding and Model Validation?**

**Answer:**

* **Model Binding**: Converts HTTP data (query string, route, form) into .NET objects.
* **Model Validation**: Uses attributes like [Required], [StringLength] to validate input.

#### **7. What are Filters in ASP.NET Core?**

**Answer:** Filters are used to inject logic at different stages of request processing (before/after action/method). Types: Authorization, Action, Result, Exception filters.

#### **8. How do you handle global exception handling?**

**Answer:** Use custom middleware or UseExceptionHandler in Startup.cs. Also log exceptions using ILogger or external providers like Serilog.

#### **9. What are Minimal APIs in .NET 6+?**

**Answer:** Lightweight way to create APIs with minimal setup:

app.MapGet("/hello", () => "Hello World!");

Useful for microservices and small apps.

#### **10. How do you implement versioning in Web APIs?**

**Answer:** Use Microsoft.AspNetCore.Mvc.Versioning. Options include:

* URL path versioning (/api/v1/resource)
* Query string
* Header versioning

#### **11. What’s the role of** Program.cs **and** Startup.cs **in .NET Core?**

**Answer:**

* Program.cs: Entry point; bootstraps host.
* Startup.cs: Configures services and middleware pipeline.

#### **12. What is the difference between synchronous and asynchronous action methods?**

**Answer:**

* Sync: blocks threads during I/O
* Async: releases thread while waiting, more scalable

public async Task<IActionResult> GetDataAsync() => await \_service.GetAsync();

#### **13. How do you implement health checks in ASP.NET Core?**

**Answer:** Add services.AddHealthChecks() and map it using:

app.MapHealthChecks("/health");

#### **14. What are some common security threats in APIs and how to prevent them?**

**Answer:**

* SQL Injection → Parameterized queries/EF Core
* XSS → Encode output
* CSRF → Anti-forgery tokens
* Authentication & Authorization → Use JWT, scopes, claims

#### **15. What’s the difference between API Gateway and Backend-for-Frontend (BFF)?**

**Answer:**

* **API Gateway**: Centralized entry point for multiple services.
* **BFF**: Custom gateway per frontend (mobile/web), tailored to frontend needs.

#### **16. How do you load configuration from multiple sources in ASP.NET Core?**

**Answer:** ASP.NET Core supports hierarchical configuration from:

* appsettings.json
* Environment variables
* Azure Key Vault
* Command-line args

builder.Configuration

.AddJsonFile("appsettings.json")

.AddEnvironmentVariables();

Would you like me to proceed with **Section 2: Entity Framework & Data Access** next?

Great! Here's the next section.

### 🔷 Section 2: Entity Framework Core & Data Access (15+ Questions with Answers)

#### **1. What is the difference between EF Core and EF 6?**

**Answer:**

* EF Core is cross-platform, lightweight, and supports async out of the box.
* EF 6 is older, Windows-only, and not actively developed.
* EF Core supports features like shadow properties, global query filters, better LINQ translation.

#### **2. What is the Code First approach in EF Core?**

**Answer:**

* You define your domain models in C# classes.
* EF Core generates the database schema from these classes using migrations.

#### **3. How do you configure relationships (one-to-many, many-to-many) in EF Core?**

**Answer:** Using Fluent API or Data Annotations:

modelBuilder.Entity<Order>()

.HasOne(o => o.Customer)

.WithMany(c => c.Orders)

.HasForeignKey(o => o.CustomerId);

#### **4. How do you perform database migrations in EF Core?**

**Answer:**

1. Add-Migration MigrationName
2. Update-Database

These generate and apply schema changes based on model differences.

#### **5. What are global query filters in EF Core?**

**Answer:** Used to filter entities globally, e.g., soft delete:

modelBuilder.Entity<Product>()

.HasQueryFilter(p => !p.IsDeleted);

#### **6. What is the difference between** AsNoTracking() **and tracked queries?**

**Answer:**

* AsNoTracking() disables change tracking — better performance for read-only queries.
* Tracked queries keep change information for updating entities.

#### **7. How do you implement Unit of Work and Repository patterns with EF Core?**

**Answer:** Encapsulate data access logic:

public interface IUnitOfWork {

IProductRepository Products { get; }

Task<int> CompleteAsync();

}

DbContext.SaveChangesAsync() acts as Unit of Work.

#### **8. What is Lazy Loading vs Eager Loading vs Explicit Loading?**

**Answer:**

* **Eager**: Include() loads related data.
* **Lazy**: EF loads data when accessed (need proxies).
* **Explicit**: Load manually via .Load() method.

#### **9. How do you seed data in EF Core?**

**Answer:** Use HasData() in OnModelCreating:

modelBuilder.Entity<Category>().HasData(

new Category { Id = 1, Name = "Electronics" }

);

#### **10. How do you handle concurrency in EF Core?**

**Answer:** Add a concurrency token (e.g., RowVersion) and decorate it:

[Timestamp]

public byte[] RowVersion { get; set; }

EF throws DbUpdateConcurrencyException when conflict occurs.

#### **11. What’s the role of DbContext and DbSet?**

**Answer:**

* DbContext: Represents session with DB, tracks changes.
* DbSet<T>: Represents a table and allows querying and updating.

#### **12. How do you handle transactions in EF Core?**

**Answer:**

using var transaction = await context.Database.BeginTransactionAsync();

// perform operations

await transaction.CommitAsync();

Or use ambient transaction with SaveChanges() if all operations are in same context.

#### **13. What is a shadow property in EF Core?**

**Answer:** A property not defined in your C# class but mapped to a DB column:

modelBuilder.Entity<Order>().Property<DateTime>("CreatedAt");

#### **14. How do you optimize performance in EF Core queries?**

**Answer:**

* Use AsNoTracking() for read-only
* Project with Select() instead of full entity loading
* Index your database columns
* Avoid N+1 problems by using Include()

#### **15. How do you execute raw SQL in EF Core?**

**Answer:**

var result = context.Products

.FromSqlRaw("SELECT \* FROM Products WHERE IsActive = 1")

.ToList();

Be cautious of SQL injection—use parameters.

#### **16. How do you test code that uses EF Core?**

**Answer:**

* Use InMemory DB or SQLite for unit tests.
* Mock the repository/DbContext interfaces in unit tests.

#### **17. What is Value Conversion in EF Core?**

**Answer:** Maps C# types to DB types when there's no direct match.

modelBuilder.Entity<User>()

.Property(e => e.Status)

.HasConversion<string>();

#### **18. How do you map enums in EF Core?**

**Answer:** By default, enums are stored as integers. To store as strings:

modelBuilder.Entity<Order>()

.Property(o => o.Status)

.HasConversion<string>();

Would you like me to continue with **Section 3: Frontend (React + TypeScript)** next?

Perfect! As requested, here's the next section:

### 🔷 Section 3: Frontend – **React + TypeScript** (Minimum 15 Questions with Answers)

These questions focus on enterprise-level React applications, TypeScript usage, and frontend architecture patterns, suitable for an architect-level candidate.

#### **1. How do you structure a large-scale React + TypeScript application?**

**Answer:** Structure by **feature or domain**:

/src

/features

/orders

OrderList.tsx

orderSlice.ts

types.ts

/shared

components/

hooks/

Use folders for: **components**, **hooks**, **store**, **services**, **types**, **utils**. Enforce boundaries and reusability.

#### **2. What are React hooks? Name a few and their purposes.**

**Answer:** Hooks allow using state and lifecycle features in functional components. Common ones:

* useState() – local state
* useEffect() – side effects
* useContext() – consume context
* useMemo() – memoization
* useCallback() – memoize functions
* useRef() – persistent mutable value

#### **3. How do you manage global state in a React app?**

**Answer:** Options:

* **Redux Toolkit** – for large-scale state with middleware
* **Context API** – for simple global state
* **Recoil / Zustand / Jotai** – modern alternatives
* **React Query / SWR** – for server state and caching

#### **4. What’s the difference between controlled and uncontrolled components?**

**Answer:**

* **Controlled**: Form state managed by React (useState)
* **Uncontrolled**: Uses DOM refs to get form values Controlled gives more control, validation, and reactivity.

#### **5. How do you type props and state in TypeScript for a React component?**

**Answer:**

type Props = { title: string; count: number };

const MyComponent: React.FC<Props> = ({ title, count }) => {

return <h1>{title} ({count})</h1>;

};

#### **6. What is memoization in React and when do you use** useMemo **and** React.memo**?**

**Answer:** Used to avoid unnecessary recalculations/rerenders:

* React.memo: Memoize entire component
* useMemo: Memoize computed values
* useCallback: Memoize functions

#### **7. How do you handle API calls in React?**

**Answer:** Use:

* fetch / axios inside useEffect()
* async/await with error handling
* Use abstraction layer (e.g., services)
* Use React Query for caching, retries, polling

#### **8. How do you implement routing in React apps?**

**Answer:** Using **React Router**:

<BrowserRouter>

<Routes>

<Route path="/home" element={<Home />} />

<Route path="/details/:id" element={<Details />} />

</Routes>

</BrowserRouter>

#### **9. How do you optimize performance in React apps?**

**Answer:**

* Lazy loading routes/components
* Code splitting with React.lazy
* Memoization: useMemo, React.memo
* Avoid prop drilling
* Virtualization (e.g., react-window)
* Efficient list rendering with key prop

#### **10. What are React Portals and where are they used?**

**Answer:** Portals allow rendering children into a DOM node outside the parent hierarchy. Used for modals, tooltips, dialogs.

#### **11. How do you manage forms in React?**

**Answer:** Options:

* Native state + validation
* Formik with Yup for schema-based validation
* react-hook-form for performant form management

#### **12. How do you write unit tests for React components?**

**Answer:** Using:

* **Jest** – test runner & assertion library
* **React Testing Library** – simulate real user interactions

render(<MyComponent />);

expect(screen.getByText("Hello")).toBeInTheDocument();

#### **13. How do you create a reusable component library?**

**Answer:**

* Create base components (Button, Input, etc.)
* Use TypeScript types & props
* Publish via private npm or as a mono-repo package
* Use Storybook for documentation

#### **14. How do you handle error boundaries in React?**

**Answer:** Use class components with componentDidCatch or create reusable wrapper:

class ErrorBoundary extends React.Component {

state = { hasError: false };

static getDerivedStateFromError() {

return { hasError: true };

}

}

#### **15. How do you implement dark mode or theme switching?**

**Answer:**

* Use Context API or CSS variables
* Toggle class on <html> or <body>
* Store user preference in localStorage

#### **16. How do you enforce code quality in React projects?**

**Answer:**

* Use **ESLint**, **Prettier**
* Setup **TypeScript strict mode**
* Configure **Husky** & **lint-staged** for pre-commit checks
* Add CI pipeline for static checks

#### **17. How do you implement lazy loading for components and routes?**

**Answer:**

const MyComponent = React.lazy(() => import('./MyComponent'));

Wrap in Suspense:

<Suspense fallback={<Loader />}><MyComponent /></Suspense>

#### **18. What are compound components in React?**

**Answer:** A design pattern where components work together (e.g., <Tabs>, <Tab>, <TabPanel>) and share implicit state.

Let me proceed next with:

### ✅ Section 4: Angular (Frontend)

Then we'll proceed to:

* Architecture & Design
* Design Patterns
* Microservices
* Azure
* DevOps
* Performance/Security
* **Cloud Architecture Patterns** (your special section)

Shall I continue with **Angular** next?

Excellent! Here is the next section:

### 🔷 Section 4: **Angular (Frontend)** – 15+ Interview Questions with Answers

#### **1. What is the difference between Angular and React in terms of architecture?**

**Answer:**

* **Angular** is a full-fledged **framework** with everything built-in (DI, routing, forms).
* **React** is a **library** focused on UI; additional libraries are needed for state, routing, etc. Angular uses **two-way data binding**, whereas React uses **one-way binding**.

#### **2. What is Angular’s component lifecycle?**

**Answer:** Angular provides lifecycle hooks:

* ngOnInit: after component init
* ngOnChanges: when input changes
* ngAfterViewInit: after view loads
* ngOnDestroy: before component destroys

#### **3. How does Angular handle Dependency Injection (DI)?**

**Answer:** Angular uses hierarchical DI. Services can be registered at:

* Root level: available app-wide
* Component/module level: scoped

@Injectable({ providedIn: 'root' })

export class MyService {}

#### **4. What are modules in Angular?**

**Answer:** Modules (@NgModule) group components, directives, pipes, and services:

* AppModule: root
* FeatureModule: per feature
* SharedModule: common components
* CoreModule: singleton services

#### **5. What is the difference between Template-driven and Reactive forms?**

**Answer:**

* **Template-driven**: Easy, declarative, for simple forms
* **Reactive forms**: Programmatic, testable, scalable, with better validation control

#### **6. How do you handle state management in Angular?**

**Answer:** Options:

* **NgRx**: Redux-style
* **RxJS Subjects**
* **Component state + service layer**
* **Akita**, **NGXS**: modern alternatives

#### **7. What are Observables in Angular?**

**Answer:** Observables (from RxJS) allow reactive programming. They:

* Emit data over time
* Are used for HTTP requests, events
* Support operators (map, filter, mergeMap)

#### **8. How do you lazy-load modules in Angular?**

**Answer:** Configure route-level lazy loading:

{ path: 'admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) }

#### **9. How do you optimize Angular app performance?**

**Answer:**

* Lazy loading
* AOT compilation
* Tree shaking
* OnPush change detection
* Avoid memory leaks in subscriptions
* Virtual scroll for large lists

#### **10. What is Change Detection Strategy in Angular?**

**Answer:** Default strategy runs on every event. OnPush only triggers when inputs change or events fire:

@Component({

changeDetection: ChangeDetectionStrategy.OnPush

})

#### **11. How do you communicate between components?**

**Answer:**

* Parent → Child: @Input()
* Child → Parent: @Output() EventEmitter
* Sibling: Shared service with Subject or BehaviorSubject

#### **12. What are Pipes and how are they used?**

**Answer:** Pipes transform data in templates:

{{ amount | currency:'INR' }}

You can create custom pipes using PipeTransform.

#### **13. How do you handle HTTP requests in Angular?**

**Answer:** Using HttpClient:

this.http.get<Product[]>('/api/products')

.subscribe(data => this.products = data);

Use interceptors for headers, logging, error handling.

#### **14. How do you secure Angular routes?**

**Answer:** Using **route guards**:

* CanActivate, CanDeactivate, Resolve
* Implement logic in a service and return boolean or Observable

#### **15. How do you test Angular components and services?**

**Answer:** Using **Jasmine + Karma** (default):

* Use TestBed to set up test module
* Use mocks/stubs for services
* Use fixture.detectChanges() to trigger rendering

#### **16. What is Ahead-of-Time (AOT) compilation?**

**Answer:** AOT compiles templates at **build time**, reducing runtime errors and improving performance. Used by default in production builds.

#### **17. What is Angular Universal?**

**Answer:** Angular Universal enables **server-side rendering** (SSR), improving SEO and load speed. Renders HTML on the server before sending to client.

#### **18. What is the role of** ngZone **in Angular?**

**Answer:** NgZone triggers Angular change detection. You can run code outside Angular zone to avoid performance hits:

this.zone.runOutsideAngular(() => { ... });

Next up will be:

### 🔷 Section 5: Architecture & Design

Let me know if you'd like to proceed. After that, I’ll include:

* Design Patterns (with C#)
* Microservices
* Azure
* DevOps
* Performance & Security
* **Cloud Architecture** (Special section)

Perfect. Let's proceed.

### 🔷 Section 5: **Architecture & Design** (15+ Questions with Answers)

These questions focus on software/system design decisions, architecture patterns, scalability, modularization, and practical trade-offs relevant for a full stack architect.

#### **1. What is Clean Architecture and how do you implement it in a .NET Core project?**

**Answer:** Clean Architecture separates concerns into layers:

* **API/UI Layer**
* **Application Layer** (Use Cases)
* **Domain Layer** (Entities, Interfaces)
* **Infrastructure Layer** (DB, external APIs)

Each layer depends only on inner layers (Dependency Rule). Use interfaces to abstract infrastructure.

#### **2. What’s the difference between Monolithic, Microservices, and Modular Monolith architectures?**

**Answer:**

* **Monolith**: All components in a single deployable unit
* **Microservices**: Independent services per business domain
* **Modular Monolith**: Modularized monolith with clear boundaries and isolated logic

#### **3. What is the Domain-Driven Design (DDD) approach?**

**Answer:** DDD focuses on the **domain model** and aligns software structure with business logic. Key concepts:

* Entities, Value Objects
* Aggregates
* Repositories
* Bounded Contexts
* Ubiquitous Language

#### **4. What is the difference between an Entity and a Value Object?**

**Answer:**

* **Entity**: Has identity (Id) and lifecycle
* **Value Object**: Immutable and compared by values (e.g., Address)

#### **5. How do you identify and define a bounded context?**

**Answer:** By aligning with business domains. Each context should:

* Own its data and model
* Expose contracts/interfaces
* Not share persistence with other contexts

#### **6. What is the CQRS pattern and when do you use it?**

**Answer:** **CQRS** separates **read** and **write** models:

* Command: modifies state
* Query: reads data Used when:
* Read and write workloads differ
* Complex domain logic
* Event sourcing is needed

#### **7. What is the difference between layered, hexagonal, and onion architectures?**

**Answer:**

* **Layered**: Linear dependencies (UI → BL → DAL)
* **Hexagonal (Ports & Adapters)**: Emphasizes input/output ports with adapters
* **Onion**: Domain at the center, with dependencies flowing inward

#### **8. What are the SOLID principles?**

**Answer:**

* **S**: Single Responsibility
* **O**: Open/Closed
* **L**: Liskov Substitution
* **I**: Interface Segregation
* **D**: Dependency Inversion

They guide maintainable and testable architecture.

#### **9. How do you design a multi-tenant architecture?**

**Answer:** Approaches:

* Shared DB, Shared Schema (cheapest, least isolation)
* Shared DB, Separate Schemas
* Separate DB per tenant (costly, best isolation) Use tenant ID in context; enforce isolation and security.

#### **10. How do you handle cross-cutting concerns (e.g., logging, validation)?**

**Answer:**

* Middleware (ASP.NET Core)
* Filters (e.g., ExceptionFilter)
* Decorator pattern or Interceptors (for validation, caching)

#### **11. How do you make architecture decisions in a team setting?**

**Answer:**

* Evaluate trade-offs (scalability, cost, complexity)
* Write ADRs (Architecture Decision Records)
* Involve stakeholders and tech leads
* Prototype if needed

#### **12. How do you design APIs for external consumers (clients/partners)?**

**Answer:**

* Use RESTful principles
* Version the API
* Include proper documentation (OpenAPI/Swagger)
* Secure with OAuth2/JWT
* Rate-limiting and throttling for public use

#### **13. How do you ensure scalability and high availability in your designs?**

**Answer:**

* Stateless services with load balancers
* Caching (Redis)
* Horizontal scaling
* Queue-based async processing
* Redundancy and replication in databases

#### **14. What is Event Sourcing and when is it useful?**

**Answer:** Store state changes (events), not current state. Rebuild state by replaying events. Useful for:

* Audit logs
* Temporal querying
* CQRS scenarios

#### **15. How do you handle versioning in microservices?**

**Answer:**

* URI versioning (/v1/resource)
* Header-based versioning
* Separate service instances per version Always ensure backward compatibility.

#### **16. What is the role of an API Gateway in microservice architecture?**

**Answer:**

* Entry point for all clients
* Handles routing, auth, throttling
* Offloads cross-cutting concerns from services Examples: Azure API Management, Ocelot, YARP

#### **17. How do you enforce modularity in large applications?**

**Answer:**

* Use feature folders or modules
* Hide implementation behind interfaces
* Define clear contracts
* Follow "high cohesion, low coupling"

#### **18. What’s your approach for fault-tolerant system design?**

**Answer:**

* Retry with exponential backoff
* Circuit breaker pattern (e.g., Polly)
* Graceful degradation
* Monitoring + alerts (Application Insights, Prometheus)

Would you like me to continue with:

👉 **Section 6: Design Patterns (C# examples)** next?

Excellent. Here's the next section:

### 🔷 Section 6: **Design Patterns in C# (Creational, Structural, Behavioral)**

**15+ Interview Questions with Answers and Code**

#### 🔹 **Creational Patterns**

#### **1. What is the Singleton pattern?**

**Answer:** Ensures a class has only one instance and provides a global point of access to it.

**Code:**

public sealed class Logger {

private static readonly Logger \_instance = new Logger();

private Logger() {}

public static Logger Instance => \_instance;

}

#### **2. What is the Factory Method pattern?**

**Answer:** Defines an interface for creating an object but lets subclasses decide the type.

**Code:**

public interface INotification { void Notify(); }

public class EmailNotification : INotification {

public void Notify() => Console.WriteLine("Email sent");

}

public class SmsNotification : INotification {

public void Notify() => Console.WriteLine("SMS sent");

}

public class NotificationFactory {

public static INotification Get(string type) =>

type == "email" ? new EmailNotification() : new SmsNotification();

}

#### **3. What is the Abstract Factory pattern?**

**Answer:** Provides an interface for creating families of related objects without specifying their concrete classes.

**Code:**

public interface IUIFactory {

IButton CreateButton();

}

public class WindowsFactory : IUIFactory {

public IButton CreateButton() => new WindowsButton();

}

public class MacFactory : IUIFactory {

public IButton CreateButton() => new MacButton();

}

#### **4. What is the Builder pattern?**

**Answer:** Builds a complex object step-by-step.

**Code:**

public class Car {

public string Engine { get; set; }

public string Color { get; set; }

}

public class CarBuilder {

private Car car = new Car();

public CarBuilder SetEngine(string e) { car.Engine = e; return this; }

public CarBuilder SetColor(string c) { car.Color = c; return this; }

public Car Build() => car;

}

#### 🔹 **Structural Patterns**

#### **5. What is the Adapter pattern?**

**Answer:** Allows incompatible interfaces to work together.

**Code:**

public class LegacyPrinter {

public void PrintOld() => Console.WriteLine("Old printer");

}

public interface IPrinter {

void Print();

}

public class Adapter : IPrinter {

private LegacyPrinter \_legacy;

public Adapter(LegacyPrinter legacy) => \_legacy = legacy;

public void Print() => \_legacy.PrintOld();

}

#### **6. What is the Decorator pattern?**

**Answer:** Adds new behavior to objects without altering their structure.

**Code:**

public interface IMessage {

string GetContent();

}

public class TextMessage : IMessage {

public string GetContent() => "Hello";

}

public class EncryptedMessage : IMessage {

private IMessage \_message;

public EncryptedMessage(IMessage msg) => \_message = msg;

public string GetContent() => Encrypt(\_message.GetContent());

private string Encrypt(string text) => Convert.ToBase64String(System.Text.Encoding.UTF8.GetBytes(text));

}

#### **7. What is the Facade pattern?**

**Answer:** Provides a simplified interface to a complex system.

**Code:**

public class OrderProcessor {

public void PlaceOrder() {

new Inventory().CheckStock();

new Payment().Process();

}

}

#### **8. What is the Proxy pattern?**

**Answer:** Provides a surrogate or placeholder for another object to control access.

**Code:**

public interface IService {

void Execute();

}

public class RealService : IService {

public void Execute() => Console.WriteLine("Executing...");

}

public class ProxyService : IService {

private RealService \_realService;

public void Execute() {

Console.WriteLine("Logging...");

\_realService = new RealService();

\_realService.Execute();

}

}

#### 🔹 **Behavioral Patterns**

#### **9. What is the Strategy pattern?**

**Answer:** Defines a family of algorithms and makes them interchangeable.

**Code:**

public interface ISortStrategy {

void Sort(List<int> list);

}

public class QuickSort : ISortStrategy {

public void Sort(List<int> list) => Console.WriteLine("QuickSort");

}

public class BubbleSort : ISortStrategy {

public void Sort(List<int> list) => Console.WriteLine("BubbleSort");

}

public class SortContext {

private ISortStrategy \_strategy;

public SortContext(ISortStrategy strategy) => \_strategy = strategy;

public void Execute(List<int> data) => \_strategy.Sort(data);

}

#### **10. What is the Observer pattern?**

**Answer:** Allows a subject to notify observers when its state changes.

**Code:**

public interface IObserver {

void Update(string message);

}

public class EmailObserver : IObserver {

public void Update(string msg) => Console.WriteLine($"Email: {msg}");

}

public class Subject {

private List<IObserver> \_observers = new();

public void Subscribe(IObserver obs) => \_observers.Add(obs);

public void Notify(string msg) {

foreach (var o in \_observers) o.Update(msg);

}

}

#### **11. What is the Command pattern?**

**Answer:** Encapsulates a request as an object.

**Code:**

public interface ICommand {

void Execute();

}

public class LightOnCommand : ICommand {

public void Execute() => Console.WriteLine("Light On");

}

public class Remote {

public void Submit(ICommand cmd) => cmd.Execute();

}

#### **12. What is the Mediator pattern?**

**Answer:** Centralizes communication between objects.

**Code:**

public class ChatRoom {

public void ShowMessage(string user, string msg) =>

Console.WriteLine($"{user}: {msg}");

}

#### **13. What is the Chain of Responsibility pattern?**

**Answer:** Passes requests through a chain of handlers.

**Code:**

public abstract class Handler {

protected Handler next;

public void SetNext(Handler h) => next = h;

public abstract void Handle(int level);

}

#### **14. What is the Template Method pattern?**

**Answer:** Defines the skeleton of an algorithm, allowing steps to be overridden.

**Code:**

public abstract class ReportGenerator {

public void Generate() {

Header();

Body();

Footer();

}

protected abstract void Header();

protected abstract void Body();

protected abstract void Footer();

}

#### **15. What is the State pattern?**

**Answer:** Allows an object to alter its behavior when its internal state changes.

Let me continue with **Section 7: Microservices** next. Then we’ll move to Azure, DevOps, Cloud Architectures, and Logical Thinking. Shall I proceed?

Great! Let's continue.

### 🔷 Section 7: **Microservices Architecture** (15+ Questions with Answers)

These questions focus on distributed system principles, microservice patterns, communication, deployment, and practical challenges.

#### **1. What are microservices?**

**Answer:** Microservices are small, independently deployable services focused on a specific business capability. Each service owns its data and communicates via APIs or messaging.

#### **2. What are the main benefits of microservices?**

**Answer:**

* Independent development and deployment
* Scalability per service
* Fault isolation
* Technology flexibility
* Better team autonomy

#### **3. What are the challenges of microservices?**

**Answer:**

* Distributed data management
* Inter-service communication
* Network latency and security
* Testing complexity
* Monitoring and tracing

#### **4. How do microservices communicate?**

**Answer:**

* **Synchronous**: REST, gRPC
* **Asynchronous**: Messaging (e.g., Azure Service Bus, RabbitMQ, Kafka)

#### **5. What is service discovery and how do you implement it?**

**Answer:** Service discovery allows dynamic lookup of service instances. In Azure:

* Use **Azure API Management** or **App Gateway**
* In Kubernetes: use DNS or service registry (e.g., Consul)

#### **6. What is an API Gateway and its role in microservices?**

**Answer:** An API Gateway is a single entry point that:

* Routes requests
* Handles auth, rate limiting, logging
* Transforms responses Examples: Azure API Management, Ocelot, YARP

#### **7. How do you handle distributed transactions in microservices?**

**Answer:**

* **Avoid** them; use eventual consistency
* Use **SAGA pattern** (choreography or orchestration)
* Implement compensating actions

#### **8. What is the SAGA pattern?**

**Answer:** A SAGA is a sequence of local transactions coordinated through:

* **Choreography**: Events trigger next service
* **Orchestration**: Central coordinator commands services

#### **9. What is eventual consistency?**

**Answer:** In distributed systems, it means all services will eventually reflect the updated state, even if not immediately. Achieved via message queues, retries.

#### **10. How do you secure microservices?**

**Answer:**

* OAuth2/JWT for authentication
* API Gateway for token validation
* Mutual TLS between services
* Validate input and restrict access by roles

#### **11. How do you handle versioning in microservices?**

**Answer:**

* URI versioning (e.g., /v1/orders)
* Headers (e.g., Accept: application/vnd.api.v1)
* Deploy multiple versions side-by-side

#### **12. How do you monitor microservices?**

**Answer:**

* Centralized logging (Serilog, ELK, Azure Monitor)
* Distributed tracing (OpenTelemetry, Azure App Insights)
* Health checks (/health) and metrics endpoints

#### **13. What is circuit breaker pattern?**

**Answer:** Prevents a failing service from being overwhelmed:

* Opens the circuit after N failures
* Closes after cooldown Tools: Polly in .NET, Steeltoe for Spring

#### **14. How do you deploy microservices?**

**Answer:**

* Azure App Services (independent)
* Containers (Docker) + AKS (Kubernetes)
* Serverless (Azure Functions per service)

#### **15. How do you handle configuration across services?**

**Answer:**

* Centralized config service (e.g., Azure App Configuration)
* Environment variables
* Key Vault for secrets

#### **16. How do you perform inter-service communication in .NET?**

**Answer:**

* Synchronous: HttpClient, Refit, gRPC
* Asynchronous: Azure Service Bus, Event Grid, Kafka

#### **17. What is database per service pattern?**

**Answer:** Each microservice has its **own DB schema**. Avoid shared DBs. Helps enforce service boundaries and autonomy.

#### **18. How do you ensure idempotency in microservices?**

**Answer:** Design operations so repeated requests have the same effect:

* Use idempotency keys (e.g., payment requests)
* Ensure DB operations are guarded with unique constraints

✅ Next up:

### 🔷 Section 8: Azure (Cloud-Native)

Shall I proceed?

Excellent! Here's the next section:

### 🔷 Section 8: **Azure Cloud (Cloud-Native Architecture)**

**15+ Interview Questions with Answers**

This section focuses on Azure services, architecture choices, best practices, and design decisions relevant to a .NET full stack developer cum architect.

#### **1. What are the key components of Azure for building cloud-native applications?**

**Answer:**

* **Azure App Service / Azure Functions** – App hosting
* **Azure SQL / Cosmos DB / Storage** – Data persistence
* **Azure Key Vault** – Secrets and keys
* **Azure Service Bus / Event Grid** – Messaging
* **Azure API Management** – API Gateway
* **Azure Monitor / App Insights** – Monitoring
* **Azure DevOps / GitHub Actions** – CI/CD

#### **2. What is Azure App Service and when should you use it?**

**Answer:** PaaS offering to host web apps, APIs, and background jobs. Use when:

* You want managed infrastructure
* Need quick deployment and scaling
* Running ASP.NET Core or Node.js apps

#### **3. What is Azure Functions and when is it suitable?**

**Answer:** Serverless compute that runs small code blocks in response to events. Use for:

* Event-driven workflows
* Background jobs
* Microservices functions (stateless)

#### **4. What is the difference between Azure App Service and Azure Functions?**

**Answer:** | Feature | App Service | Azure Functions | |----------------|------------------|----------------------| | Hosting | Full web app | Event-driven methods | | Lifecycle | Long-running | Short execution | | Cost model | Reserved pricing | Consumption-based |

#### **5. How do you secure secrets and connection strings in Azure?**

**Answer:** Use **Azure Key Vault** and access them via:

* Managed Identity
* Azure App Configuration
* Environment variables at runtime

#### **6. How do you implement API Gateway on Azure?**

**Answer:** Use **Azure API Management**:

* Provides throttling, versioning, routing, caching, transformations
* Fronts microservices and externalizes policies

#### **7. What is Azure Service Bus and how is it different from Event Grid?**

**Answer:**

* **Service Bus**: Enterprise messaging with queues/topics (supports ordering, retries)
* **Event Grid**: Pub/Sub for lightweight event-driven scenarios (high throughput, low latency)

#### **8. What is Azure B2C and when should you use it?**

**Answer:** Azure AD B2C is an identity provider for apps used by **external users** (customers). Supports:

* Social logins (Google, Facebook)
* Custom user flows
* Multi-tenancy

#### **9. What is Azure AD and how is it used in enterprise apps?**

**Answer:** Azure AD is an identity and access management service:

* Used for employee login (SSO)
* Secure APIs with OAuth2 / OpenID Connect
* RBAC with roles and policies

#### **10. How do you design for high availability in Azure?**

**Answer:**

* Use **Availability Zones** and **regions**
* Deploy across multiple instances
* Use **Azure Front Door / Traffic Manager** for failover
* Design stateless services

#### **11. How do you implement autoscaling in Azure App Services?**

**Answer:**

* Define scale rules (CPU, memory, queue length)
* Use horizontal scaling (add instances)
* Configure in Azure Portal or ARM templates

#### **12. How do you store large files in Azure?**

**Answer:** Use **Azure Blob Storage**:

* Supports hot/cold/archive tiers
* Can generate **SAS tokens** for secure file access
* Easily integrated with Azure CDN

#### **13. How do you monitor and trace issues in Azure-hosted apps?**

**Answer:** Use **Application Insights** and **Azure Monitor**:

* Tracks exceptions, request logs, performance
* Supports distributed tracing for microservices

#### **14. What is Azure App Configuration and how is it different from Key Vault?**

**Answer:**

* **App Configuration**: Manages app settings, feature flags, dynamic config
* **Key Vault**: Stores secrets, keys, and certificates securely

Use both for centralized, secure config management.

#### **15. What is Azure Cosmos DB and its key features?**

**Answer:** Globally distributed NoSQL DB supporting:

* Multiple APIs (SQL, MongoDB, Cassandra)
* Partitioning and global replication
* Tunable consistency levels
* SLA for latency, availability, throughput

#### **16. How do you handle deployment in Azure?**

**Answer:**

* Use **Azure DevOps Pipelines** or **GitHub Actions**
* ARM/Bicep/Terraform for infrastructure as code
* Blue-green or Canary deployments for safe releases

#### **17. What is Managed Identity and why is it important?**

**Answer:** Allows your app to access Azure resources without storing secrets:

* Use with Key Vault, Storage, etc.
* Eliminates credential management

✅ Coming up next:

### 🔷 Section 9: DevOps, CI/CD & Testing

Then:

* Performance & Security
* Cloud Architecture Patterns (Bonus Section)
* Logical/Aptitude Thinking Questions (Final Section)

Shall I proceed to **DevOps**?

Got it! I’ll include **mature, scenario-based logical and critical thinking questions** in the final section to reflect the seniority of your experience.

Now, let's proceed with the next section:

### 🔷 Section 9: **DevOps, CI/CD, and Testing** (15+ Questions with Answers)

#### **1. What is CI/CD and why is it important in modern software delivery?**

**Answer:** CI/CD stands for **Continuous Integration** and **Continuous Deployment/Delivery**. It automates:

* Building
* Testing
* Releasing

Benefits:

* Faster delivery
* Early bug detection
* Consistent deployments

#### **2. What tools do you use for CI/CD in Azure?**

**Answer:**

* **Azure DevOps Pipelines**
* **GitHub Actions**
* **YAML pipelines**
* Integration with **Docker**, **Kubernetes**, **ARM/Bicep**, **Terraform**

#### **3. What’s the difference between release and deployment?**

**Answer:**

* **Deployment**: Moving code to an environment
* **Release**: Making the feature available to users

Decouple release from deployment using feature flags.

#### **4. How do you implement a secure CI/CD pipeline?**

**Answer:**

* Store secrets in Azure Key Vault
* Use service connections with least privilege
* Enable approvals and gated releases
* Sign artifacts
* Scan for vulnerabilities (SAST, DAST)

#### **5. What is Infrastructure as Code (IaC)?**

**Answer:** IaC manages infrastructure using code instead of manual setup.

Tools:

* **ARM Templates**, **Bicep** (Azure)
* **Terraform** (cross-cloud)

Benefits:

* Repeatability
* Version control
* Auditability

#### **6. What is the difference between Bicep and ARM templates?**

**Answer:**

* **ARM**: JSON-based, verbose
* **Bicep**: DSL over ARM, simpler syntax, easier to maintain

Both compile to ARM under the hood.

#### **7. How do you manage multi-environment deployments (dev, test, prod)?**

**Answer:**

* Use parameterized YAML templates
* Use variable groups and environment-specific settings
* Use release gates and approvals
* Use separate resource groups or subscriptions

#### **8. What is Canary Deployment and how do you implement it in Azure?**

**Answer:** Deploy new version to a subset of users, gradually increase traffic.

In Azure:

* **App Service Deployment Slots**
* **Traffic splitting** via slots or Azure Front Door

#### **9. What are Azure Deployment Slots?**

**Answer:** App Service feature that allows:

* Deploying to a **staging slot**
* Testing in production-like environment
* **Swapping** to production with zero downtime

#### **10. How do you version and package your .NET applications?**

**Answer:**

* Use **SemVer** (1.2.3)
* Generate build versions via CI
* Package using dotnet pack (NuGet) or Docker image tags

#### **11. How do you write effective unit tests in .NET?**

**Answer:**

* Follow **AAA** pattern (Arrange-Act-Assert)
* Mock dependencies using **Moq**
* Test behavior and edge cases
* Use **xUnit**, **NUnit**, or **MSTest**

#### **12. What is the difference between unit, integration, and E2E tests?**

**Answer:**

* **Unit**: Isolated, test one function/class
* **Integration**: Multiple components together (e.g., DB + API)
* **E2E**: Full system from UI to DB

#### **13. How do you mock dependencies in unit tests?**

**Answer:** Use libraries like **Moq**:

var mockService = new Mock<IMyService>();

mockService.Setup(s => s.GetData()).Returns("test");

#### **14. How do you handle test data for integration testing?**

**Answer:**

* Use **InMemory DB** for EF Core
* Seed test data per test run
* Use Docker containers for full DB if needed

#### **15. How do you measure code quality and test coverage?**

**Answer:**

* Use **SonarQube**, **Coverlet**, or **dotCover**
* Setup coverage thresholds in pipelines
* Integrate quality gates in PR validation

#### **16. How do you handle secrets in CI/CD pipelines?**

**Answer:**

* Never store secrets in source control
* Store in **Azure Key Vault**
* Link Key Vault to pipeline variables using service principal or managed identity

#### **17. What is shift-left testing and why is it important?**

**Answer:** Shift testing earlier in the SDLC to detect issues early:

* Run unit and static analysis on pull requests
* Shortens feedback loop
* Reduces cost of fixing bugs

Next up:

### 🔷 Section 10: Performance, Security & Optimization

Then we’ll move to:

* **Section 11: Cloud Architecture Patterns**
* **Section 12: Logical/Aptitude Thinking (mature, thought-provoking)**

Ready to proceed?

Great! Here's the next section:

### 🔷 Section 10: **Performance, Security & Optimization**

**15+ Interview Questions with Expected Answers**

This section focuses on architect-level concerns like app optimization, scalability, secure design, and reliability.

#### **1. How do you improve the performance of a .NET Web API?**

**Answer:**

* Use **async/await** for I/O operations
* Enable **response caching**
* Avoid blocking calls and excessive logging
* Reduce serialization overhead
* Use **compiled queries** in EF Core
* Implement **pagination** for large data sets

#### **2. How do you improve frontend performance (React/Angular)?**

**Answer:**

* Lazy load components and routes
* Use **OnPush** (Angular) or **memoization** (React)
* Minify JS/CSS
* Use code splitting
* Avoid unnecessary re-renders

#### **3. How do you reduce the payload in API responses?**

**Answer:**

* Use **DTOs** instead of returning full entities
* Support **query parameters** to select fields
* Use **GZIP compression**
* Implement **pagination** and **filtering**

#### **4. What are some caching strategies you’ve used?**

**Answer:**

* **In-memory caching** (MemoryCache)
* **Distributed caching** (Redis)
* **Output caching** for views/responses
* Cache invalidation using time-based or event-based rules

#### **5. How do you detect and resolve memory leaks in .NET applications?**

**Answer:**

* Use **dotMemory**, **dotTrace**, or **Visual Studio Profiler**
* Look for unreferenced objects in memory
* Dispose of IDisposable objects
* Avoid static events that prevent GC

#### **6. How do you protect APIs from abuse or brute-force attacks?**

**Answer:**

* Implement **rate limiting** (via API Gateway or middleware)
* Use **OAuth2 / JWT** with expiration
* Detect anomalies using logging and alerts

#### **7. How do you implement input validation and prevent injection attacks?**

**Answer:**

* Use libraries like **FluentValidation**
* Validate all inputs, including headers and query strings
* Use **parameterized queries** in SQL to avoid SQL injection

#### **8. What is Cross-Site Scripting (XSS) and how do you prevent it?**

**Answer:** XSS is injecting malicious scripts into web pages. Prevent by:

* Encoding output in HTML
* Avoiding dangerous user input
* Using security libraries like **DomPurify**

#### **9. What is CSRF and how do you protect against it?**

**Answer:** CSRF tricks the user’s browser into making a request they didn’t intend.

**Prevention:**

* Use anti-CSRF tokens (with SameSite cookie settings)
* Validate origin headers
* Use **ASP.NET Core’s built-in CSRF protection**

#### **10. How do you ensure secure authentication and authorization in your applications?**

**Answer:**

* Use **Azure AD**, **IdentityServer**, or **OAuth2**
* Protect APIs with **JWT Bearer tokens**
* Use **role-based or policy-based** authorization
* Store passwords hashed with **PBKDF2**, **bcrypt**

#### **11. How do you handle large-scale logging without affecting performance?**

**Answer:**

* Use **structured logging** (Serilog, NLog)
* Write logs asynchronously
* Push logs to external services (Azure Monitor, Seq, ELK)

#### **12. What are the OWASP Top 10 vulnerabilities you defend against?**

**Answer:**

* SQL Injection
* XSS
* CSRF
* Insecure Deserialization
* Broken Authentication
* Security Misconfiguration
* Insufficient Logging & Monitoring
* Broken Access Control
* Using Vulnerable Components
* Sensitive Data Exposure

#### **13. What tools do you use for performance testing?**

**Answer:**

* **BenchmarkDotNet** (C# micro-benchmarking)
* **Postman / JMeter / k6** (API load testing)
* **Lighthouse** for frontend
* Azure Load Testing

#### **14. How do you design for scalability and high performance?**

**Answer:**

* Stateless services
* Load balancing
* Caching at multiple layers (API, DB, frontend)
* Async processing (queues)
* Database sharding/partitioning

#### **15. What is rate limiting and how is it implemented?**

**Answer:** Limits the number of requests a user can make in a time window.

**Implementation:**

* Middleware (e.g., AspNetCoreRateLimit)
* API Gateway policies
* Redis-backed counters

#### **16. How do you design for graceful degradation?**

**Answer:**

* Provide fallback content or features
* Use **circuit breakers** for downstream failures
* Notify users appropriately
* Retry or queue the failed action

✅ Next up:

### 🔷 Section 11: Cloud Architecture Patterns (Enterprise)

Followed by:

### 🔷 Section 12: Logical / Aptitude Thinking (Mature, Senior-Level)

Shall I continue with **Cloud Architecture Patterns**?

Excellent! Here's the next section:

### 🔷 Section 11: **Cloud Architecture Patterns**

**15+ Interview Questions with Answers (Enterprise Level)**

These questions focus on how you use cloud-native design patterns and principles to build scalable, resilient, and efficient systems in Azure and other cloud platforms.

#### **1. What is the difference between vertical and horizontal scaling?**

**Answer:**

* **Vertical scaling**: Add more resources (CPU, RAM) to a single server
* **Horizontal scaling**: Add more servers to distribute load Horizontal is more suitable for cloud-native and microservices architecture.

#### **2. What is the Twelve-Factor App and why is it important?**

**Answer:** A methodology for building cloud-native apps:

* Codebase in version control
* Config via environment
* Backing services as attached resources
* Stateless processes
* Logs as event streams
* Dev/prod parity

Used to build portable and scalable SaaS apps.

#### **3. What is the Circuit Breaker pattern and when do you use it?**

**Answer:** Temporarily halts calls to a failing service to prevent cascading failures.

* Helps avoid overwhelming a dependency
* Common in Polly (for .NET), Azure App Gateway with custom rules

#### **4. What is the Retry pattern and how does it differ from Circuit Breaker?**

**Answer:** **Retry** tries a failed operation again after a delay. **Circuit Breaker** blocks calls after consecutive failures.

They are often used together: Retry first, then break circuit if persistent failure.

#### **5. Explain the Bulkhead pattern.**

**Answer:** Isolates resources or functionality into separate containers (like compartments in a ship), so one failure doesn’t bring down the entire system.

Azure example:

* Separate services across App Service Plans
* Use container orchestration with limits

#### **6. What is the Queue-Based Load Leveling pattern?**

**Answer:** Uses a message queue (e.g., Azure Service Bus) between producer and consumer to:

* Buffer spikes
* Decouple services
* Prevent overloading

#### **7. What is the Event Sourcing pattern?**

**Answer:** Stores state as a sequence of events. The current state is derived by replaying these events. Useful when:

* Full audit is needed
* Complex domain logic
* Integration with external systems via event replay

#### **8. What is the Command and Query Responsibility Segregation (CQRS) pattern?**

**Answer:** Separates read and write logic:

* Commands change state
* Queries read data Improves scalability and maintainability in complex systems.

#### **9. What is the Strangler Fig pattern?**

**Answer:** Gradually replaces parts of a legacy system by building new functionality in parallel and routing traffic selectively. Useful during modernization or cloud migration.

#### **10. What is the Fan-Out/Fan-In pattern?**

**Answer:** Fan-out: split tasks across multiple services/functions Fan-in: combine results after all are complete

Example: Azure Durable Functions with WhenAll to parallelize work.

#### **11. How do you design a multi-tenant SaaS app in Azure?**

**Answer:**

* Shared DB with tenant identifier
* Shared DB with separate schemas
* Isolated DB per tenant (high isolation) Use Azure AD B2C for user management and access control.

#### **12. What is eventual consistency and how do you achieve it in cloud apps?**

**Answer:** In distributed systems, not all nodes reflect the same state instantly. Achieved via:

* Queues
* Eventual sync
* Retries
* Background workers

#### **13. What is API Gateway pattern and how do you implement it in Azure?**

**Answer:** A single entry point for multiple backend services. Provides:

* Routing
* Auth
* Throttling
* Caching

**Azure API Management** is the go-to solution for this.

#### **14. What is Auto-Scaling and how do you configure it in Azure?**

**Answer:** Automatically adjusts resources based on:

* CPU usage
* Queue length
* Custom metrics

Implemented in:

* App Services
* Azure Functions (Consumption plan)
* Azure Kubernetes Service (AKS)

#### **15. What are Azure Availability Zones and why use them?**

**Answer:** Physically separate datacenters within a region:

* Used for high availability
* Protects against datacenter-level failure
* Recommended for production apps

#### **16. What is Geo-Replication and how do you use it in Azure?**

**Answer:** Replicates data across regions:

* Enabled in Azure Blob Storage, Cosmos DB
* Provides disaster recovery and faster read performance for global apps

#### **17. How do you handle secrets and sensitive config in a distributed system?**

**Answer:**

* Use **Azure Key Vault**
* Access via **Managed Identity**
* Avoid environment variables for secrets

✅ Next up is the **final section**:

### 🔷 Section 12: Logical, Aptitude & Thought-Provoking Questions

(Senior-level reasoning, system thinking, and decision-making)

Proceed?

Perfect! Let's now conclude with the final section tailored for **experienced professionals**:

### 🔷 Section 12: Logical Thinking, Aptitude & Scenario-Based Reasoning

**(15+ Mature-Level Interview Questions with Answers)**

These questions test your **decision-making**, **critical thinking**, **business understanding**, and **architectural intuition** — not just your coding skills.

#### **1. You're asked to rewrite a legacy monolithic app. Would you choose microservices? Why or why not?**

**Answer:** Not necessarily. Microservices add complexity (distributed system, DevOps overhead). I'd first analyze:

* Business domains and team structure
* Performance/scalability needs
* Change frequency per module  
  If justified, I'd consider **modular monolith** first, then evolve to microservices.

#### **2. Your team wants to use the latest JS framework in production. What would you consider before approval?**

**Answer:**

* Team’s expertise
* Community support & maturity
* Tooling and integration with .NET backend
* Long-term maintainability
* Security & stability Trendy isn’t always right for production.

#### **3. A stakeholder demands real-time analytics for a dashboard. What’s your thought process?**

**Answer:**

* Define what “real-time” means (sub-second, near real-time?)
* Consider Azure SignalR, Event Hubs, WebSockets
* Ensure backend supports streaming
* Handle backpressure and latency gracefully Balance user experience vs. system cost/complexity.

#### **4. A production bug occurs only under high load. What steps do you take?**

**Answer:**

* Reproduce using load testing
* Enable detailed telemetry (App Insights, logs)
* Profile CPU, memory usage
* Analyze logs for race conditions, thread starvation
* Consider caching, async calls, scaling

#### **5. Your team is split on two designs. How do you make the final call?**

**Answer:**

* Facilitate a design review
* List trade-offs (maintainability, performance, team skill)
* Run a spike (prototype both)
* Document with ADR (Architecture Decision Record)
* Consensus where possible; otherwise, decision backed by data

#### **6. How do you approach a project with unclear or changing requirements?**

**Answer:**

* Use agile/iterative delivery
* Focus on building a flexible, modular system
* Define MVP and build incrementally
* Prioritize stakeholder communication and feedback loops

#### **7. A customer says “the system is slow.” How do you debug?**

**Answer:**

* Ask for scenario, metrics, timestamps
* Check logs, request timings, DB performance
* Use telemetry (App Insights) to isolate bottleneck
* Categorize: network, API, DB, frontend

#### **8. You notice your junior devs are copy-pasting code. What do you do?**

**Answer:**

* Coach them on **DRY** principles and abstraction
* Introduce code reviews and pair programming
* Set architectural standards
* Offer examples and reusable libraries/components

#### **9. How do you balance business urgency with technical debt?**

**Answer:**

* Track tech debt formally (backlog)
* Prioritize fixes based on impact (performance, security)
* Negotiate with product team to include refactoring in sprints
* Present long-term risk of ignoring debt

#### **10. Your system is under DDoS attack. What is your immediate plan?**

**Answer:**

* Enable rate limiting and geo-fencing via WAF/API Gateway
* Monitor traffic via Azure Front Door/App Gateway
* Scale out services temporarily
* Engage with security team

#### **11. How do you handle disagreement with another architect?**

**Answer:**

* Focus on facts, not ego
* Understand their perspective and constraints
* Back decisions with data, POCs, benchmarks
* Agree to disagree if both approaches are valid

#### **12. You need to redesign a system to reduce Azure cost. What’s your approach?**

**Answer:**

* Identify high-cost resources (App Insights, App Service Plans, DBs)
* Right-size compute (use consumption-based plans)
* Apply auto-scaling and stop unused environments
* Optimize storage tiering (hot → cool)

#### **13. A new dev joins your team. How do you onboard them effectively?**

**Answer:**

* Provide system overview, architecture docs, code walkthroughs
* Assign a small, low-risk feature
* Pair program for initial days
* Encourage questions and retrospectives

#### **14. How do you justify a refactoring effort to a non-technical stakeholder?**

**Answer:**

* Relate it to business value: speed, stability, maintainability
* Share specific risks or inefficiencies
* Use cost-of-change curve or past bugs as data points

#### **15. You’re asked to add a new feature with tight deadline. How do you ensure quality?**

**Answer:**

* Define scope clearly (avoid scope creep)
* Add unit + integration tests for new code
* Use feature toggles if needed
* Deploy via staging slot and monitor post-release

#### **16. In an architecture decision meeting, what are the key dimensions you evaluate?**

**Answer:**

* Scalability
* Maintainability
* Security
* Performance
* Team readiness
* Cost (operational + developmental)
* Vendor lock-in

#### **17. How do you design a system that can handle unexpected future growth?**

**Answer:**

* Design for scale-out
* Use queues for async processing
* Externalize config
* Use loosely coupled services (avoid tight dependencies)

#### **18. What would you do if a cloud provider increases cost drastically for a key service?**

**Answer:**

* Analyze usage & dependencies
* Optimize configurations
* Evaluate hybrid/multi-cloud alternatives
* Benchmark against other services/providers

✅ That concludes your **full-stack architect-level interview prep set**, covering:

* ✅ .NET, C#, Backend
* ✅ React & Angular (in earlier sections)
* ✅ Azure Cloud
* ✅ System Design
* ✅ DevOps
* ✅ Design Patterns
* ✅ Security & Performance
* ✅ Logical & Decision-Making Thinking

Would you like this entire set exported to a **Word or PDF document** now? Or do you want any section refined or expanded further?

Great! Here's the new section:

### 🔷 Section 13: **Coding Interview Questions**

**15+ Practical C# Coding Problems with Explanations and Expected Solutions**

These problems are tailored for a **10+ years experienced .NET developer/architect**, focusing on logic, data structures, algorithms, and clean architecture.

#### **1. Reverse a string without using built-in functions**

**Problem:** Write a method that reverses a string.

**Solution:**

public string ReverseString(string input)

{

char[] result = new char[input.Length];

for (int i = input.Length - 1, j = 0; i >= 0; i--, j++)

result[j] = input[i];

return new string(result);

}

#### **2. Check if a string is a palindrome**

**Problem:** Ignore case and non-alphanumeric characters.

**Solution:**

public bool IsPalindrome(string input)

{

var cleaned = new string(input.Where(char.IsLetterOrDigit).ToArray()).ToLower();

return cleaned == new string(cleaned.Reverse().ToArray());

}

#### **3. Detect duplicate elements in an array**

**Problem:** Return the first duplicate.

**Solution:**

public int? FirstDuplicate(int[] nums)

{

HashSet<int> seen = new();

foreach (var num in nums)

{

if (!seen.Add(num)) return num;

}

return null;

}

#### **4. FizzBuzz (Clean Code Version)**

**Problem:** Print numbers 1 to 100. Replace:

* Multiples of 3 → Fizz
* Multiples of 5 → Buzz
* Both → FizzBuzz

**Solution:**

public void FizzBuzz()

{

for (int i = 1; i <= 100; i++)

{

string output = "";

if (i % 3 == 0) output += "Fizz";

if (i % 5 == 0) output += "Buzz";

Console.WriteLine(output == "" ? i : output);

}

}

#### **5. Implement a basic LRU Cache**

**Problem:** Limit capacity, remove least recently used item.

**Solution:**

public class LRUCache<K, V>

{

private readonly int \_capacity;

private readonly Dictionary<K, LinkedListNode<(K key, V val)>> \_map;

private readonly LinkedList<(K key, V val)> \_list;

public LRUCache(int capacity)

{

\_capacity = capacity;

\_map = new();

\_list = new();

}

public V Get(K key)

{

if (!\_map.ContainsKey(key)) return default;

var node = \_map[key];

\_list.Remove(node);

\_list.AddFirst(node);

return node.Value.val;

}

public void Put(K key, V value)

{

if (\_map.ContainsKey(key)) \_list.Remove(\_map[key]);

else if (\_map.Count >= \_capacity)

{

var last = \_list.Last;

\_list.RemoveLast();

\_map.Remove(last.Value.key);

}

var node = new LinkedListNode<(K, V)>((key, value));

\_list.AddFirst(node);

\_map[key] = node;

}

}

#### **6. Find the longest substring without repeating characters**

**Problem:** Return the length of the longest substring without repeating characters.

**Solution:**

public int LengthOfLongestSubstring(string s)

{

var set = new HashSet<char>();

int left = 0, maxLen = 0;

for (int right = 0; right < s.Length; right++)

{

while (set.Contains(s[right]))

set.Remove(s[left++]);

set.Add(s[right]);

maxLen = Math.Max(maxLen, right - left + 1);

}

return maxLen;

}

#### **7. Merge two sorted arrays**

**Solution:**

public int[] MergeSortedArrays(int[] a, int[] b)

{

int i = 0, j = 0;

List<int> result = new();

while (i < a.Length && j < b.Length)

{

result.Add(a[i] < b[j] ? a[i++] : b[j++]);

}

result.AddRange(a[i..]);

result.AddRange(b[j..]);

return result.ToArray();

}

#### **8. Implement a custom LINQ-style** Where **filter**

**Solution:**

public static IEnumerable<T> MyWhere<T>(this IEnumerable<T> source, Func<T, bool> predicate)

{

foreach (var item in source)

{

if (predicate(item)) yield return item;

}

}

#### **9. Count the frequency of each word in a paragraph**

**Solution:**

public Dictionary<string, int> WordFrequency(string text)

{

var words = text.Split(' ', StringSplitOptions.RemoveEmptyEntries);

return words

.Select(w => w.ToLowerInvariant())

.GroupBy(w => w)

.ToDictionary(g => g.Key, g => g.Count());

}

#### **10. Implement Dependency Injection manually (mini container)**

**Solution:**

public class Container

{

private Dictionary<Type, Type> \_map = new();

public void Register<TContract, TImpl>() where TImpl : TContract

{

\_map[typeof(TContract)] = typeof(TImpl);

}

public TContract Resolve<TContract>()

{

var implType = \_map[typeof(TContract)];

return (TContract)Activator.CreateInstance(implType);

}

}

#### **11. Check if a binary tree is balanced**

**Solution:**

public bool IsBalanced(TreeNode root)

{

return Height(root) != -1;

}

private int Height(TreeNode node)

{

if (node == null) return 0;

int left = Height(node.left);

int right = Height(node.right);

if (left == -1 || right == -1 || Math.Abs(left - right) > 1) return -1;

return Math.Max(left, right) + 1;

}

#### **12. Convert a number to Roman numeral**

**Solution:**

public string ToRoman(int num)

{

var map = new (int val, string sym)[] {

(1000, "M"), (900, "CM"), (500, "D"), (400, "CD"),

(100, "C"), (90, "XC"), (50, "L"), (40, "XL"),

(10, "X"), (9, "IX"), (5, "V"), (4, "IV"), (1, "I")

};

var result = new StringBuilder();

foreach (var (val, sym) in map)

{

while (num >= val)

{

result.Append(sym);

num -= val;

}

}

return result.ToString();

}

#### **13. Find the intersection of two arrays**

**Solution:**

public int[] Intersect(int[] nums1, int[] nums2)

{

var set = new HashSet<int>(nums1);

return nums2.Where(n => set.Remove(n)).ToArray();

}

#### **14. Write a method to flatten a nested list**

**Example:** [1, [2, [3, 4]], 5] → [1, 2, 3, 4, 5]

**Solution:**

public IEnumerable<int> Flatten(IEnumerable<object> nested)

{

foreach (var item in nested)

{

if (item is int val)

yield return val;

else if (item is IEnumerable<object> sublist)

foreach (var sub in Flatten(sublist))

yield return sub;

}

}

#### **15. Serialize and deserialize a custom object to/from JSON**

**Solution:**

public class Person { public string Name; public int Age; }

public string Serialize(Person p) => JsonSerializer.Serialize(p);

public Person Deserialize(string json) =>

JsonSerializer.Deserialize<Person>(json);

✅ This section includes algorithmic, architectural, and practical real-world tasks.  
Would you like a **separate set of system design/coding case studies** or a **PDF/Word export** of the full content now?

Excellent. Below is a curated list of **medium to complex non-DSA coding questions** designed specifically for a **Tech Lead or Architect-level** C#/.NET full-stack professional. These focus on real-world architectural thinking, clean code, design patterns, SOLID principles, and advanced system-level practices — **not simple algorithms or data structures**.

## 🔷 Section 13-A (Revised): **Non-DSA Coding Questions for Tech Lead (C#)**

**Medium to Complex – Real-World Architecture, Design, and Implementation**

### ✅ **1. Implement a plugin-based architecture to support runtime extension loading**

**Scenario:**  
You are designing a system where new business logic components (plugins) can be added at runtime without restarting the app.

**Expected Solution:**

* Use AssemblyLoadContext (in .NET Core) or MEF (Managed Extensibility Framework)
* Load DLLs from a folder dynamically and invoke specific interfaces

public interface IPlugin

{

string Name { get; }

void Execute();

}

public class PluginLoader

{

public IEnumerable<IPlugin> LoadPlugins(string path)

{

var plugins = new List<IPlugin>();

foreach (var file in Directory.GetFiles(path, "\*.dll"))

{

var assembly = Assembly.LoadFrom(file);

var types = assembly.GetTypes().Where(t => typeof(IPlugin).IsAssignableFrom(t) && !t.IsInterface);

foreach (var type in types)

{

plugins.Add((IPlugin)Activator.CreateInstance(type));

}

}

return plugins;

}

}

### ✅ **2. Build a simple rules engine (strategy pattern)**

**Scenario:**  
Implement business rules dynamically for discount calculation without changing core logic.

**Expected Solution:** Use IStrategy interface and load implementations dynamically or via DI.

public interface IDiscountStrategy

{

decimal Calculate(decimal amount);

}

public class NewCustomerDiscount : IDiscountStrategy

{

public decimal Calculate(decimal amount) => amount \* 0.9m;

}

public class PremiumCustomerDiscount : IDiscountStrategy

{

public decimal Calculate(decimal amount) => amount \* 0.8m;

}

public class DiscountService

{

private readonly IDiscountStrategy \_strategy;

public DiscountService(IDiscountStrategy strategy)

{

\_strategy = strategy;

}

public decimal ApplyDiscount(decimal amount) => \_strategy.Calculate(amount);

}

### ✅ **3. Build a lightweight event bus for in-memory communication**

**Scenario:**  
You want components to communicate without tight coupling.

**Expected Concepts:** Observer/Event Aggregator pattern.

public class EventAggregator

{

private readonly Dictionary<Type, List<Delegate>> \_handlers = new();

public void Subscribe<T>(Action<T> handler)

{

var type = typeof(T);

if (!\_handlers.ContainsKey(type))

\_handlers[type] = new List<Delegate>();

\_handlers[type].Add(handler);

}

public void Publish<T>(T eventData)

{

var type = typeof(T);

if (\_handlers.TryGetValue(type, out var handlers))

foreach (var handler in handlers)

((Action<T>)handler)(eventData);

}

}

### ✅ **4. Implement centralized exception logging middleware with correlation ID**

**Expected Concepts:**

* Custom Middleware
* Use of ILogger<>
* Inject IHttpContextAccessor to pass CorrelationId

public class ErrorLoggingMiddleware

{

private readonly RequestDelegate \_next;

private readonly ILogger<ErrorLoggingMiddleware> \_logger;

public ErrorLoggingMiddleware(RequestDelegate next, ILogger<ErrorLoggingMiddleware> logger)

{

\_next = next;

\_logger = logger;

}

public async Task Invoke(HttpContext context)

{

try

{

await \_next(context);

}

catch (Exception ex)

{

var correlationId = context.TraceIdentifier;

\_logger.LogError(ex, "Exception with Correlation ID: {Id}", correlationId);

context.Response.StatusCode = 500;

await context.Response.WriteAsync($"Error: {correlationId}");

}

}

}

### ✅ **5. Design a scheduled background job system (clean code)**

**Scenario:**  
Implement recurring background jobs using IHostedService or Quartz.NET for business reports or email dispatching.

public class ReportJob : BackgroundService

{

private readonly ILogger<ReportJob> \_logger;

public ReportJob(ILogger<ReportJob> logger) => \_logger = logger;

protected override async Task ExecuteAsync(CancellationToken stoppingToken)

{

while (!stoppingToken.IsCancellationRequested)

{

\_logger.LogInformation("Running report job at: {time}", DateTimeOffset.Now);

// do some work

await Task.Delay(TimeSpan.FromMinutes(10), stoppingToken);

}

}

}

### ✅ **6. Build an extensible audit logging framework**

**Expected Design:**

* Use a IAuditSink interface
* Write to DB, file, or external system interchangeably

public interface IAuditSink

{

Task WriteAsync(AuditEntry entry);

}

public class FileAuditSink : IAuditSink

{

public Task WriteAsync(AuditEntry entry) =>

File.AppendAllTextAsync("audit.txt", JsonSerializer.Serialize(entry) + "\n");

}

public class AuditEntry

{

public string Actor { get; set; }

public string Action { get; set; }

public DateTime Timestamp { get; set; } = DateTime.UtcNow;

}

### ✅ **7. Dynamic querying with Expression Trees**

**Scenario:**  
You want to build dynamic filters (like user search filters) based on client input.

public IQueryable<T> ApplyFilter<T>(IQueryable<T> source, string property, object value)

{

var param = Expression.Parameter(typeof(T));

var prop = Expression.Property(param, property);

var body = Expression.Equal(prop, Expression.Constant(value));

var lambda = Expression.Lambda<Func<T, bool>>(body, param);

return source.Where(lambda);

}

### ✅ **8. Build a decorator pattern for adding logging around service calls**

public interface IOrderService

{

void PlaceOrder(string orderId);

}

public class OrderService : IOrderService

{

public void PlaceOrder(string orderId) => Console.WriteLine($"Order {orderId} placed");

}

public class LoggingDecorator : IOrderService

{

private readonly IOrderService \_inner;

public LoggingDecorator(IOrderService inner) => \_inner = inner;

public void PlaceOrder(string orderId)

{

Console.WriteLine("Logging before");

\_inner.PlaceOrder(orderId);

Console.WriteLine("Logging after");

}

}

### ✅ **9. Enforce clean architecture separation with command handlers**

**Scenario:**  
Use CQRS + MediatR-style handlers

public class CreateOrderCommand : IRequest<bool>

{

public string OrderId { get; set; }

}

public class CreateOrderHandler : IRequestHandler<CreateOrderCommand, bool>

{

public Task<bool> Handle(CreateOrderCommand request, CancellationToken cancellationToken)

{

// business logic

return Task.FromResult(true);

}

}

### ✅ **10. Build a validation pipeline using the Chain of Responsibility**

public abstract class ValidationHandler

{

protected ValidationHandler Next;

public void SetNext(ValidationHandler next) => Next = next;

public virtual void Handle(string input)

{

Next?.Handle(input);

}

}

public class NullCheckHandler : ValidationHandler

{

public override void Handle(string input)

{

if (string.IsNullOrEmpty(input)) throw new ArgumentNullException(nameof(input));

base.Handle(input);

}

}

Would you like **10 more** of these types of **mature, architectural coding problems**?  
Or should we now add/revisit a **refined DSA section tailored for architects**?

Great! Let’s continue with **10 more medium-to-complex, non-DSA coding questions** suitable for a **Tech Lead/Architect role**, focused on enterprise-grade, real-world .NET/C# scenarios.

## 🔷 Section 13-A (Continued): **Non-DSA Coding Questions for Tech Lead – Part 2**

### ✅ **11. Implement a multi-tenant aware service layer**

**Scenario:**  
You are building a SaaS system with multiple tenants. You need to isolate logic and data per tenant dynamically.

public interface ITenantContext

{

string TenantId { get; }

}

public class TenantMiddleware

{

private readonly RequestDelegate \_next;

public TenantMiddleware(RequestDelegate next) => \_next = next;

public async Task Invoke(HttpContext context, ITenantContext tenantContext)

{

var tenantId = context.Request.Headers["X-Tenant-ID"].FirstOrDefault();

if (!string.IsNullOrWhiteSpace(tenantId))

((TenantContext)tenantContext).TenantId = tenantId;

await \_next(context);

}

}

public class TenantContext : ITenantContext

{

public string TenantId { get; set; }

}

### ✅ **12. Design a custom model binder to parse complex query strings into objects**

**Scenario:**  
Your API needs to accept nested filter conditions like filter[status]=active&filter[priority]=high.

public class FilterModelBinder : IModelBinder

{

public Task BindModelAsync(ModelBindingContext context)

{

var result = new FilterDto();

foreach (var key in context.HttpContext.Request.Query.Keys)

{

if (key.StartsWith("filter["))

{

var prop = key.Replace("filter[", "").Replace("]", "");

typeof(FilterDto).GetProperty(prop, BindingFlags.IgnoreCase | BindingFlags.Public | BindingFlags.Instance)

?.SetValue(result, context.HttpContext.Request.Query[key]);

}

}

context.Result = ModelBindingResult.Success(result);

return Task.CompletedTask;

}

}

### ✅ **13. Implement a scoped correlation ID logger for distributed tracing**

**Scenario:**  
You want consistent trace IDs across logs in a single request pipeline.

public class CorrelationIdMiddleware

{

private const string Header = "X-Correlation-ID";

private readonly RequestDelegate \_next;

public CorrelationIdMiddleware(RequestDelegate next) => \_next = next;

public async Task Invoke(HttpContext context)

{

if (!context.Request.Headers.ContainsKey(Header))

context.Request.Headers[Header] = Guid.NewGuid().ToString();

context.Response.Headers[Header] = context.Request.Headers[Header];

using (LogContext.PushProperty("CorrelationId", context.Request.Headers[Header]))

{

await \_next(context);

}

}

}

### ✅ **14. Implement dynamic localization using resource files and middleware**

**Scenario:**  
Your application should support dynamic culture switching via header Accept-Language.

public class LocalizationMiddleware

{

private readonly RequestDelegate \_next;

public LocalizationMiddleware(RequestDelegate next) => \_next = next;

public async Task Invoke(HttpContext context)

{

var cultureQuery = context.Request.Headers["Accept-Language"].ToString();

if (!string.IsNullOrWhiteSpace(cultureQuery))

{

var culture = new CultureInfo(cultureQuery);

CultureInfo.CurrentCulture = culture;

CultureInfo.CurrentUICulture = culture;

}

await \_next(context);

}

}

### ✅ **15. Build an extensible factory with dependency injection**

**Scenario:**  
Create different implementations based on enum/strategy value.

public enum NotificationType { Email, SMS }

public interface INotificationSender

{

void Send(string message);

}

public class EmailSender : INotificationSender

{

public void Send(string msg) => Console.WriteLine("Email: " + msg);

}

public class SMSSender : INotificationSender

{

public void Send(string msg) => Console.WriteLine("SMS: " + msg);

}

public class NotificationFactory

{

private readonly IServiceProvider \_provider;

public NotificationFactory(IServiceProvider provider) => \_provider = provider;

public INotificationSender Create(NotificationType type) =>

type switch

{

NotificationType.Email => \_provider.GetRequiredService<EmailSender>(),

NotificationType.SMS => \_provider.GetRequiredService<SMSSender>(),

\_ => throw new NotImplementedException()

};

}

### ✅ **16. Implement dynamic role-based endpoint visibility in a Blazor or React SPA**

**Scenario:**  
Backend sends a list of allowed features per user. Frontend hides buttons/pages based on roles.

**Backend API (ASP.NET Core):**

[HttpGet("features")]

public IActionResult GetFeaturesForUser()

{

var roles = User.Claims.Where(c => c.Type == ClaimTypes.Role).Select(r => r.Value);

var features = FeatureRegistry.GetFeaturesForRoles(roles);

return Ok(features);

}

### ✅ **17. Build a rate limiter using memory cache**

**Scenario:**  
Limit API to 100 calls per IP per minute.

public class RateLimitingMiddleware

{

private readonly RequestDelegate \_next;

private readonly IMemoryCache \_cache;

public RateLimitingMiddleware(RequestDelegate next, IMemoryCache cache)

{

\_next = next;

\_cache = cache;

}

public async Task Invoke(HttpContext context)

{

var key = context.Connection.RemoteIpAddress.ToString();

var count = \_cache.GetOrCreate(key, entry =>

{

entry.AbsoluteExpirationRelativeToNow = TimeSpan.FromMinutes(1);

return 0;

});

if (count >= 100)

{

context.Response.StatusCode = 429;

await context.Response.WriteAsync("Rate limit exceeded");

return;

}

\_cache.Set(key, ++count);

await \_next(context);

}

}

### ✅ **18. Implement Domain Event Dispatching in a Clean Architecture app**

**Scenario:**  
After placing an order, you want to trigger notifications without tightly coupling logic.

public class OrderPlacedEvent : IDomainEvent

{

public Order Order { get; }

public OrderPlacedEvent(Order order) => Order = order;

}

public class OrderPlacedHandler : IDomainEventHandler<OrderPlacedEvent>

{

public Task Handle(OrderPlacedEvent domainEvent)

{

// Send email/notify

return Task.CompletedTask;

}

}

Use a mediator (like MediatR) or publish via event dispatcher injected into domain service.

### ✅ **19. Build a JSON patch updater using** JsonPatchDocument<T>

**Scenario:**  
Allow partial updates to entities via HTTP PATCH

[HttpPatch("{id}")]

public async Task<IActionResult> Patch(int id, [FromBody] JsonPatchDocument<Order> patch)

{

var order = await \_context.Orders.FindAsync(id);

if (order == null) return NotFound();

patch.ApplyTo(order, ModelState);

await \_context.SaveChangesAsync();

return Ok(order);

}

### ✅ **20. Design a system for on-demand PDF generation using headless browser**

**Scenario:**  
Export UI as PDF (React/Angular app) from server-side using Puppeteer or wkhtmltopdf

public async Task<IActionResult> GeneratePDF(string url)

{

var process = new Process

{

StartInfo = new ProcessStartInfo

{

FileName = "wkhtmltopdf",

Arguments = $"{url} output.pdf",

RedirectStandardOutput = true,

UseShellExecute = false,

CreateNoWindow = true

}

};

process.Start();

process.WaitForExit();

return File(System.IO.File.ReadAllBytes("output.pdf"), "application/pdf", "report.pdf");

}

✅ **Next Steps:** Would you now like the **DSA coding section revised for advanced Tech Lead/Architect level**? Or should I bundle everything so far into a **downloadable format (PDF/Word)**?