**MIDDLEWARE**  
*“Middleware in ASP.NET Core is a component in the HTTP request pipeline that can handle requests and responses. Each middleware piece can perform actions both before and after the next component executes, which allows you to build modular, layered behavior in your application. For example, you can add middleware for logging, authentication, error handling, routing, and more.*

*In ASP.NET Core, middleware is added in the* Program.cs *or previously in* Startup.cs*, using methods like* UseRouting()*,* UseAuthentication()*, and custom* app.Use(...) *calls. The order in which you add them is really important because it directly affects how requests are processed. I've even implemented custom middleware—for instance, one that logs request execution time or appends headers for security.”*

**Ex; of custom middleware**  
**app.Use(async (context, next) =>**

**{**

**var path = context.Request.Path;**

**Console.WriteLine($"Request for {path} started.");**

**await next();**

**Console.WriteLine($"Request for {path} ended.");**

**});**

**What are FILTERS**

*A filter in ASP.NET Core is a component that allows code to run at specific stages in the request processing pipeline of an MVC or Web API application. Filters provide a declarative way to handle cross-cutting concerns like logging, validation, exception handling, caching, or authorization.*

*For example, I’ve used* IActionFilter *and* IExceptionFilter *to log the execution of controller actions and to catch unhandled exceptions globally. They can be applied globally, at the controller level, or action level depending on the scope required. ASP.NET Core also supports asynchronous versions of filters, which is helpful when working with I/O-bound operations.*

*In contrast to middleware, which operates at the app level, filters are tightly integrated into the MVC pipeline, meaning they have full access to route data, model state, and controller context.”*

**END POINT FILTER**

*“An Endpoint Filter in ASP.NET Core is a feature introduced in .NET 7 that allows you to attach reusable logic directly to minimal API endpoints. They function like localized middleware, meaning they let you intercept, modify, or validate requests and responses at the endpoint level.*

*I've used them to handle input validation, authorization checks, and even response shaping. They're especially useful in minimal API scenarios where traditional MVC filters don't apply. Since they execute before and after the endpoint handler, they give you fine-grained control without affecting the entire request pipeline.*

*Compared to middleware, which operates globally, endpoint filters target specific routes, which keeps the app lean and modular—perfect for microservices or performance-focused APIs.”*

**DEPENDENCY INJECTION**

*“Dependency Injection (DI) is a design pattern used in ASP.NET Core to achieve loose coupling between components. It allows the system to provide the dependencies a class needs, rather than the class creating them internally. This makes the application easier to test, maintain, and extend.*

*For example, instead of creating a* DbContext *or service instance directly inside a controller, I inject it via the constructor. ASP.NET Core’s built-in IoC container handles the lifecycle and resolves the dependency at runtime.*

*Here’s a typical use case I’ve implemented: in* Program.cs*, I register a service—say* IOrderService*—using* AddScoped()*, and inject it into a controller via its constructor. This makes testing and mocking much easier and keeps my code modular and clean.”*

**// Program.cs**

**builder.Services.AddScoped<IOrderService, OrderService>();**

**// Inside Controller**

**private readonly IOrderService \_service;**

**public OrdersController(IOrderService service)**

**{**

**\_service = service;**

**}**

*“In ASP.NET Core, services can be registered with three lifetimes: Transient, Scoped, and Singleton—each defines how and when instances are created.*

* **Transient**: A new instance is created *every time* it's requested. Best for lightweight, stateless services.
* **Scoped**: A single instance is created *per HTTP request*. Useful for services that maintain request-level state, like DbContext.
* **Singleton**: Only *one instance* is created and reused for the app’s entire lifetime. Best for stateless and thread-safe operations.”\*

### **1. What is the difference between .NET Framework and .NET Core/.NET 6+?**

**Answer:** .NET Framework is Windows-only and monolithic, while .NET Core (now unified as .NET 6/7/8) is cross-platform, modular, and optimized for cloud-native development.

### **2. What is Dependency Injection in ASP.NET Core?**

**Answer:** DI is a design pattern where dependencies are provided to a class rather than created inside it. ASP.NET Core has built-in support via IServiceCollection.

### **3. What are Middleware and their role in ASP.NET Core?**

**Answer:** Middleware are components in the request pipeline that can handle, modify, or short-circuit HTTP requests/responses. Examples: UseRouting, UseAuthentication, custom logging middleware.

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

**Answer:**

* Scoped: One instance per HTTP request
* Transient: New instance every time it's requested
* Singleton: One instance for the app's lifetime

### **5. What is Model Binding and Model Validation in ASP.NET Core?**

**Answer:** Model Binding maps HTTP request data to action parameters. Validation uses data annotations like [Required], [Range], and can be extended with custom validators.

## **Frontend (Angular/JavaScript)**

### **6. What is the difference between Angular services and components?**

**Answer:** Components manage UI and templates, while services handle business logic and data access. Services are injected using Angular’s DI system.

### **7. What are Observables and how are they used in Angular?**

**Answer:** Observables (from RxJS) are used for asynchronous data streams—like HTTP calls. They support operators like map, filter, and switchMap.

### **8. How do you handle form validation in Angular?**

**Answer:** Using ReactiveFormsModule or FormsModule, with validators like Validators.required, Validators.email, and custom validators.

## **🗃️ Database & ORM**

### **9. What is Entity Framework Core and how does it work?**

**Answer:** EF Core is an ORM that maps .NET objects to database tables. It supports LINQ queries, migrations, and change tracking.

### **10. What is the difference between Code First and Database First?**

**Answer:**

* **Code First**: Define models in code, generate DB via migrations
* **Database First**: Scaffold models from an existing DB

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

**Answer:** Using DbContext.Database.BeginTransaction() or TransactionScope for atomic operations.