🔷 SECTION 1: ASP.NET CORE FUNDAMENTALS & ARCHITECTURE 🔷

1️⃣ What is ASP.NET Core?

🎯 **ASP.NET Core is**:

A **cross-platform, open-source** framework
Used to build:

- Web APIs
- MVC applications
- Microservices

Designed for **high performance, modularity, and cloud readiness**

📌 It replaces the old **ASP.NET Framework**.

🔹 One-liner (MEMORIZE)

"ASP.NET Core is a cross-platform, high-performance framework for building modern web applications and APIs."

2️⃣ Why ASP.NET Core? (WHY WAS IT CREATED)

Old ASP.NET:

- Windows-only
- Heavy
- Hard to scale

ASP.NET Core:

- Lightweight
- Fast
- Runs anywhere

🎯 ASP.NET Core was introduced to:

- Be **cross-platform** (Windows, Linux, macOS)
- Improve **performance**
- Support **cloud & microservices**
- Provide **built-in dependency injection**
- Have a **modular middleware pipeline**

🔁 Q: Is ASP.NET Core just a version upgrade?
❌ No — it's a complete rewrite.

3️⃣ ASP.NET Core vs ASP.NET Framework

ASP.NET Core	**ASP.NET Framework**
Cross-platform	Windows only
High performance	Slower
Middleware pipeline	System.Web
Built-in DI	Limited DI
Cloud-ready	Legacy

📌 Don't say "Core is better" — say why.

4️⃣ Key Architectural Principles

🔹 **Modular Architecture**

🧠You only use what you need.

- Features are added via NuGet packages
- Smaller memory footprint
- Better maintainability

🔹 **Middleware-based Pipeline**

- No System.Web
- Everything is middleware
- Full control over request flow

🔹 **Built-in Dependency Injection**

- First-class DI support
- Promotes clean architecture
- Improves testability

🔹 **Unified MVC & Web API**

🧠One framework for both APIs and web apps.

- No separate Web API framework
- Controllers serve both JSON and views

5️⃣ Hosting & Web Server (IMPORTANT)

🧠ASP.NET Core apps don't directly listen to the internet. They use a web server.

🎯ASP.NET Core uses:

- Kestrel (cross-platform web server)
- Can be hosted behind:
  - IIS
  - Nginx
  - Apache

📌 Kestrel is fast but **not edge-secure.**

🔁 Q: Can Kestrel be used directly in production?

✔ Yes, with proper hardening

✔ Often behind reverse proxy

6️⃣ Application Startup (VERY IMPORTANT)

🧠When the app starts:

- Services are registered
- Middleware pipeline is built

🎯Startup has two major phases:

- **Service registration**
- **Middleware configuration**

In modern ASP.NET Core (6+):

`Program.cs` handles everything

7️⃣ **Environment Support**

🎯 ASP.NET Core supports:

- Development
- Staging
- Production

Used for:

- Logging levels
- Error handling
- Configurations

📌 Environment-specific behavior is built-in.

8️⃣ **Configuration System**

🧠 Settings come from many places:

- JSON
- Environment variables
- Command line

Configuration is:

- Hierarchical
- Provider-based
- Environment-aware

📌 Order matters — last provider wins.

9️⃣ **Performance Focus** (WHY COMPANIES LOVE IT)

🎯 ASP.NET Core:

- Asynchronous by default
- Minimal allocations
- Optimized pipeline
- Scales well under load

🔟 Common Interview Traps 🚨

- ❌ “ASP.NET Core runs only on IIS”
- ❌ “It’s just MVC framework”
- ❌ “Middleware and filters are same”
- ❌ “Core is only for APIs”

🧠 **FINAL INTERVIEW SUMMARY (MEMORIZE)**

“ASP.NET Core is a modern, cross-platform, high-performance framework designed to build scalable web applications and APIs using a modular middleware pipeline, built-in dependency injection, and cloud-ready architecture.”

🔧 HANDS-ON 1: ASP.NET CORE FUNDAMENTALS

🧱 STEP 0: Create a Minimal ASP.NET Core App

```
dotnet new webapi -n FundamentalsDemo
cd FundamentalsDemo
```

Run it: `dotnet run`

Open browser: `https://localhost:<port>/weatherforecast`

✅ App is running — this proves Kestrel + ASP.NET Core pipeline are working.

🧱 STEP 1: Understand Project Structure (INTERVIEW FAVORITE)

```
FundamentalsDemo
|— Controllers
|   └— WeatherForecastController.cs
|— Program.cs
|— appsettings.json
|— Properties/launchSettings.json
```

📌 ASP.NET Core uses a minimal startup model where `Program.cs` configures services and middleware.

🧱 STEP 2: Program.cs – HEART OF ASP.NET CORE

Open `Program.cs`:

```
var builder = WebApplication.CreateBuilder(args);
```

What this does:

🧠 Creates the application builder — sets up DI, config, logging.

🎯 This initializes the host, configuration system, logging, and dependency injection container.

Register Services: `builder.Services.AddControllers();`

📌 Meaning:

- Adds MVC services
- Registers controllers into DI container

Interview line: Services are registered before building the app.

Build the App: `var app = builder.Build();`

📌 Meaning:

- DI container is finalized
- Middleware pipeline is ready to be configured

🧱 STEP 3: Middleware Pipeline (VERY IMPORTANT)

```
app.UseHttpsRedirection();
```

🧠 Redirects HTTP → HTTPS

🎯 Middleware executes in order and forms the request pipeline.

```
app.UseAuthorization();
```

🎯 Authorization middleware enforces access policies after authentication.

```
app.MapControllers();
```

🧠 Maps incoming HTTP requests to controllers.

🎯 Endpoint routing maps routes to controller actions.

Final Line: `app.Run();`

📌 Interview trap: This is terminal middleware — nothing runs after this.

🧱 STEP 4: Controller Anatomy

Open `WeatherForecastController.cs`

```
[ApiController]
[Route("[controller]")]
public class WeatherForecastController : ControllerBase
{
```

📌 Interview Explanation:

- `[ApiController]` enables:
  - Automatic model validation
  - Binding source inference
- `ControllerBase` is used for APIs (no views)

Action Method

```
[HttpGet]
public IEnumerable<WeatherForecast> Get()
```

📌 Attribute routing defines HTTP verb and endpoint mapping.

🧱 STEP 5: Request Flow (MENTAL MODEL)

When you hit /weatherforecast

```
Browser → Kestrel → Middleware pipeline → Routing → Controller action → Response
```

📌 Every request passes through `Kestrel → middleware → endpoint → response`.

🧱 STEP 6: Environment Awareness

Open `launchSettings.json`

```
"environmentVariables": {
  "ASPNETCORE_ENVIRONMENT": "Development"
}
```

📌 ASP.NET Core behavior changes based on environment (Dev, Staging, Prod).

🧱 STEP 7: appsettings.json (Configuration)

```
Json
{
  "Logging": {
    "LogLevel": {
      "Default": "Information"
    }
  }
}
```

📌 Configuration is provider-based and hierarchical.

🧱 STEP 8: Minimal API (MODERN INTERVIEW BONUS)

Add in `Program.cs` before `app.Run()`:

```
app.MapGet("/ping", () => "API is alive");
```

Call: `GET /ping`

📌 Minimal APIs are lightweight endpoints suitable for microservices.

🧠 FINAL INTERVIEW SUMMARY (MEMORIZE)

"An ASP.NET Core application starts in Program.cs where services are registered into the DI container and middleware is configured to form the request pipeline. Requests are handled by Kestrel, processed through middleware, routed to controllers or minimal APIs, and returned as responses."