ASP.NET Core - Practice Sheet

10 Q&A on pipeline, DI lifetimes, Minimal API, auth, and testing

Q1. What is the ASP.NET Core request pipeline and how do you add middleware?

A: The pipeline is a chain of middleware delegates in Program.cs. app.UseRouting(), app.UseAuthentication(), and app.UseAuthorization() must be in that order. app.Use(async (ctx, next) => {...; await next(ctx)}) inserts custom middleware.

Q2. What are the three DI service lifetimes and what are captive dependency risks?

A: Singleton lives for the app lifetime. Scoped lives per HTTP request. Transient is created fresh each injection. A captive dependency occurs when a Singleton injects a Scoped service the Scoped service is never disposed correctly. ASP.NET Core detects this at startup.

Q3. How does ASP.NET Core attribute routing work?

A: [Route('api/users')] on a controller and [HttpGet('{id:int}')] on an action define the full route template. Route constraints like :int, :guid, and :minlength(3) validate segments. [ApiController] enforces model binding and returns 400 on validation failure.

Q4. What is the difference between [FromBody], [FromQuery], and [FromRoute]?

A: [FromBody] reads from the JSON request body (only one per action). [FromQuery] reads query string parameters. [FromRoute] reads named route template segments. Without explicit attributes, [ApiController] infers: complex types from body, simple types from query/route.

Q5. What is Minimal API in ASP.NET Core 6+ and when should you use it?

A: Minimal API uses lambda-based route handlers (app.MapGet('/users/{id}', (int id) => ...)) instead of controllers. It is less structured but has lower overhead and works well for microservices or simple CRUD APIs where the MVC controller ceremony is unnecessary.

Q6. How do you add JWT Bearer authentication to ASP.NET Core?

A: Call builder.Services.AddAuthentication().AddJwtBearer(opts => {opts.TokenValidationParameters = new() {ValidIssuer=..., IssuerSigningKey=...}}). Add app.UseAuthentication() and app.UseAuthorization() in the pipeline. Protect endpoints with

Q7. What is the Options pattern in ASP.NET Core configuration?

A: Define a class MySettings with properties. Call builder.Services.Configure<MySettings>(builder.Configuration.GetSection('MySettings')). Inject IOptions<MySettings> (singleton snapshot), IOptionsSnapshot<MySettings> (scoped, reloads per request), or

Q8. What is Entity Framework Core and how does Add-Migration work?

A: EF Core is Microsoft's ORM. DbContext represents a session with the database. Define DbSet<T> properties for each entity. dotnet ef migrations add AddUserTable generates a migration C# file based on the difference between the model and the last migration.

Q9. What are action filters in ASP.NET Core and what can they do?

A: IActionFilter runs OnActionExecuting (before the action) and OnActionExecuted (after). Use them for logging, validation, or caching. [ResponseCache], [ValidateAntiForgeryToken] are built-in filters. Register globally in AddControllers(opts => opts.Filters.Add(...)).

Q10. How do you write integration tests for ASP.NET Core?

A: Use WebApplicationFactory<Program> to spin up the app in-process. Call factory.CreateClient() to get an HttpClient that sends requests to the test server. Override services with factory.WithWebHostBuilder(b => b.ConfigureTestServices(...)). No real port is needed.