Interview Questions & Answers: JWT Authentication, Clean Architecture, and ASP.NET Core

1 Authentication & Authorization

1. **What is JWT and how does it work in ASP.NET Core?**
   JWT (JSON Web Token) is a compact, URL-safe means of representing claims to be transferred between two parties. In ASP.NET Core, JWTs are generated on successful login and contain encoded user claims (e.g., username, role). The token is returned to the client and included in the Authorization header (Bearer) in subsequent requests.

2. **How do you implement role-based authorization in ASP.NET Core?**
   You can implement role-based authorization by adding roles to the JWT token as claims and using the `[Authorize(Roles = "RoleName")]` attribute on controllers or actions.

3. **What is the difference between [Authorize] and [AllowAnonymous]?**

   - `[Authorize]` restricts access to authenticated users.
   - `[AllowAnonymous]` overrides `[Authorize]` to allow public access.

4. **How do you assign roles to users and use them in JWT?**
   On login, map user roles (from DB) to the token using `ClaimTypes.Role`. For example:
   `new Claim(ClaimTypes.Role, RoleHelper.GetRoleName(user.RoleId))`

5. **How do you secure endpoints for specific roles using [Authorize(Roles = "...")]?**
   Example:
   `[Authorize(Roles = "Admin,Staff")]`
   This allows only Admin and Staff users to access the endpoint.

6. **What is a Claim in JWT? How do you use claims in authorization?**
   Claims are key-value pairs in the JWT that contain user-specific information (e.g., ID, role). They're used by ASP.NET Core middleware to determine user identity and access rights.

7. **Can you handle multiple roles in an attribute? How?**
   Yes. Use comma-separated values in `[Authorize(Roles = "Admin,Student")]`.

2 Clean Architecture / Layered Architecture

8. **Explain the Clean Architecture and how you structured your project.**
   Clean Architecture separates concerns into layers:

- API for controllers
- Application for services and validation
- Core for interfaces and entities
- Infrastructure for data access

9. **Where do you place your business logic controller, service, or repository? Why?**
   Business logic goes into the Service layer to keep controllers thin and repositories focused on data access.

10. **What is the purpose of using DTOs? How do you map them to entities?**
    DTOs (Data Transfer Objects) simplify and secure data transfer. Mapping can be manual or via libraries like AutoMapper.

11. **How do you keep your controllers clean and minimal?**
    Use DTOs, delegate business logic to the service layer, and handle validation in validators.

3 Validation

12. **How do you validate user input in your API?**
    Using FluentValidation. Validators are defined for each DTO and injected via constructor.

13. **What is FluentValidation? How is it better than Data Annotations?**
    FluentValidation is a flexible, clean way to apply validations with logic. It supports custom rules and is easier to test and maintain.

14. **How do you write custom validators (e.g., age $\geq$ 18)?**
    Example:
    `RuleFor(x => x.StaffDob).Must(BeAtLeast18YearsOld).WithMessage("Must be 18+");`

15. **How do you inject validators and handle validation failures?**
    Inject `IValidator<T>` in the repository or service and call `ValidateAsync(dto)`. Throw `ValidationException` if not valid.

4 Repository & Service Pattern

16. **Why do you use Repository and Service layers?**
    To separate database access (Repository) from business logic (Service), ensuring clean, maintainable code.

17. **What's the difference between Repository and Service layers?**

- Repository handles DB operations
- Service handles business logic and orchestration

18. **Where do you place your database logic and why?**
    In the repository layer, to isolate EF Core dependencies.

19. **How do you inject and use repositories in services?**
    Via constructor injection of repository interfaces.

5 Entity Framework Core (EF Core)

20. **How do you check if a record already exists in the database before inserting (e.g., email)?**
    Use `AnyAsync()` in repository:
    ```
    if (await _context.Users.AnyAsync(u => u.Email == dto.Email))
    ```

21. **How do you use async/await in EF Core operations?**
    EF Core supports async methods like `ToListAsync()`, `AddAsync()`, `SaveChangesAsync()`.

22. **How do you configure relationships between entities? (If applicable)**
    Use Fluent API or data annotations in `OnModelCreating`.

23. **How do you manage migrations and database updates?**
    Use commands:
    ```
    dotnet ef migrations add InitialCreate
    dotnet ef database update
    ```

6 General .NET Core Questions

24. **What are middleware and how do you configure JWT middleware?**
    Middleware handles request/response pipeline. JWT middleware is added using:
    `services.AddAuthentication(...).AddJwtBearer(...)`

25. **How do you read configuration settings (e.g., JWT key, issuer)?**
    Use:
    `_configuration.GetSection("JwtSettings")["Key"]`

26. **How do you secure your API?**
    Use HTTPS, JWT Authentication, Role-based Authorization, and Input Validation.

27. **How do you handle errors and return proper HTTP status codes (e.g., 403, 401, 400)?**
    Use middleware and structured responses. Return `Unauthorized()`, `Forbidden()`, or `BadRequest()`.

28. **What are IHttpContextAccessor and ClaimsPrincipal used for?**
    Used to access user information from the current HTTP request, including JWT claims.

7 Bonus / Advanced Concepts

29. **What are enums and how do you convert int roles to enum names?**
    Enums are named constants. Use helper:
    `Enum.GetName(typeof(UserRole), roleId)`

30. **How do you write helper classes for reusable logic like role mapping?**
    Create a static helper:
    `public static class RoleHelper { public static string GetRoleName(int roleId) => ...; }`

31. **How would you implement refresh tokens?**
    Store refresh tokens in DB, return it with access token, and issue new access token on valid refresh request.

32. **How would you implement custom authorization policies?**
    Use `AddAuthorization` to register policies and apply them using `[Authorize(Policy = "PolicyName")]`.