200+ Full Stack Developer

Interview Questions and answers

Sai Reddy 5/4/25 Interview Tips

200+ Full Stack Interview Q&A

(0–2 years experience)

1. What is Angular? How is it different from AngularJS?

   Angular is a TypeScript-based framework used to build SPA (Single Page Applications).
   AngularJS is based on JavaScript and follows the MVC pattern.
   Angular is component-based while AngularJS uses controllers and scopes.
   Angular uses TypeScript, supports modular architecture, and has better performance.
   Example: Angular (v2+) uses @Component, AngularJS uses controller.

2. What are components in Angular?

   Components are building blocks of Angular apps.
   Each component has an HTML template, TypeScript class, and optional CSS.
   Components control views and handle user interactions.
   They are defined using @Component decorator.
   Example:
   @Component({ selector: 'app-hello', template: '<h1>Hello</h1>' }) export class HelloComponent {}

3. What is a module in Angular?

   A module groups related components, services, and directives.
   Defined using @NgModule decorator.
   Every Angular app has at least one root module (AppModule).
   Modules can be feature-specific and imported as needed.
   Example:
   @NgModule({ declarations: [AppComponent], imports: [BrowserModule] }) export class AppModule {}

4. What is data binding? Explain the types.

   Data binding connects the component and the template.
   Types: Interpolation, Property binding, Event binding, Two-way binding.
   Helps sync UI and model easily.
   Enables dynamic view updates.
   Example: {{name}}, [src]="imageUrl", (click)="onClick()", [(ngModel)]="name"

5. What is interpolation in Angular?

   Interpolation binds component data to the view.
   It uses {{ }} syntax to display variables or expressions.
   It's one-way from component to template.
   Mainly used in templates for string rendering.
   Example: **<h1>Hello {{ username }}</h1>**

6. What is a directive? Name types with examples.

   Directives add behavior to DOM elements.
   Three types: Structural (*ngIf, *ngFor), Attribute (ngClass, ngStyle), Custom.
   Structural changes the DOM layout.
   Attribute changes appearance or behavior.
   Example: **<div *ngIf="isVisible">Visible **

7. What is a service in Angular?

Services provide reusable business logic or data operations.
They are injected into components using dependency injection.
Services are defined as classes and decorated with @Injectable().
Promotes code reuse and separation of concerns.
Example:

```
@Injectable() export class UserService { getUsers() { return ['Tom', 'Jerry']; } }
```

8. How do you communicate between components?

Use @Input() to pass data from parent to child.
Use @Output() and EventEmitter to send data from child to parent.
Use services with Observables for sibling communication.
Helps create reusable and decoupled components.
Example:

```
@Input() title: string; @Output() clicked = new EventEmitter<string>();
```

9. What is dependency injection in Angular?

It's a design pattern where dependencies are provided rather than created.
Angular's injector provides services to components or other services.
Promotes loose coupling and testability.
Use constructor injection to get services.
Example:

```
constructor(private userService: UserService) {}
```

10. What are lifecycle hooks? Name a few.

Lifecycle hooks are special Angular methods triggered during different stages of a component's life: creation, change detection, rendering, and destruction.
Angular provides multiple hooks like:

```
ngOnInit(), ngOnChanges(), ngDoCheck()
ngAfterContentInit(), ngAfterContentChecked()
ngAfterViewInit(), ngAfterViewChecked(), ngOnDestroy()
```

They help in initializing data, reacting to changes, checking updates, or cleaning up.
Example:

```
ngAfterViewInit() { console.log('View initialized'); }
```

11. What is the purpose of ngOnInit()?

It's a lifecycle hook called after the component's constructor.
Used to initialize data or fetch from services.
Preferred over constructors for Angular logic.
Called once the component's inputs are set.
Example:

```
ngOnInit() { this.users = this.userService.getUsers(); }
```

12. What is ngIf, ngFor, and ngSwitch?

ngIf: conditionally adds/removes DOM elements.
ngFor: loops over a collection to render elements.
ngSwitch: switch-case rendering for elements.
All are structural directives.
Example:

```
<div *ngIf="isLoggedIn">Welcome</div>
<li *ngFor="let item of items">{{item}}</li>
```

13. What is two-way data binding? How is it implemented?

Two-way data binding synchronizes data between class and template.
Achieved using [(ngModel)] syntax.
Requires FormsModule.
Changes in input reflect in model and vice versa.
Example:
`<input [(ngModel)]="username"> {{ username }}`

14. What is Angular CLI and how do you use it?

Angular CLI is a command-line tool to create and manage Angular apps.
It provides commands to generate components, services, modules, etc.
Also used for building, testing, and serving the app.
Saves time and enforces standards.
Example: `ng generate component login or ng serve`

15. How do you handle forms in Angular (Template-driven vs Reactive)?

Template-driven forms use HTML and directives like ngModel.
Reactive forms use FormControl and FormGroup in code.
Reactive forms are more scalable and testable.
Template-driven is easier for simple use-cases.
Example:
`this.form = new FormGroup({ name: new FormControl('') });`

16. How do you make HTTP calls in Angular?

Use HttpClient service to make API calls.
Inject HttpClient in the component or service.
Supports GET, POST, PUT, DELETE methods.
Must import HttpClientModule.
Example:
`this.http.get('api/users').subscribe(data => this.users = data);`

17. What is HttpClientModule?

It's an Angular module to enable HTTP communication.
Must be imported in AppModule to use HttpClient.
Offers a simplified and observable-based HTTP API.
It replaces the older HttpModule.
Example:
`imports: [HttpClientModule]`

18. What is the purpose of routing in Angular?

Routing allows navigation between different views/components.
Managed using RouterModule and route definitions.
Enables SPA behavior without full page reloads.
Improves user experience and performance.
Example:
`{ path: 'home', component: HomeComponent }`

19. How do you pass parameters to routes?

Use route parameters in the route path (:id).
Access them using ActivatedRoute.
Use routerLink to navigate with params.

Use paramMap to get the parameter values.

Example:

```
this.route.snapshot.paramMap.get('id');
```

20. What are pipes in Angular?

Pipes transform data in templates.
Built-in pipes: date, uppercase, currency, async.
You can also create custom pipes.
They are used with | symbol.

Example: `{{ birthday | date:'longDate' }}`

2–5 years' experience

1. Template-Driven vs Reactive Forms

Template-driven forms use HTML and are easier for simple forms; reactive forms use TypeScript and are scalable.
Reactive forms offer better testability and dynamic control.
Template-driven uses ngModel, while reactive uses FormControl, FormGroup.
Validation in reactive forms is more flexible and powerful.

Example:

```
new FormControl('Angular', Validators.required);
```

2. What is Change Detection in Angular?

Angular's Change Detection updates the DOM when data changes.
It checks component state and rerenders affected views.
Uses Zone.js to track async events.
ChangeDetectorRef and OnPush strategy help optimize it.

Example:

```
this.cd.detectChanges(); // manually trigger change detection
```

3. How do you optimize performance in Angular apps?

Use OnPush strategy, lazy loading, and pure pipes.
Avoid unnecessary watchers and use trackBy in *ngFor.
Detach change detection when not needed.
Minimize DOM manipulation and bundle size.

Example:

```
*ngFor="let item of items; trackBy: trackById"
```

4. Subject vs BehaviorSubject vs ReplaySubject

- Subject: emits only after subscription.
- BehaviorSubject: emits latest value to new subscribers.
- ReplaySubject: replays specified number of previous values.
  Used for state sharing and event streams.

  Example:

  ```
  const subject = new BehaviorSubject<number>(0);
  ```

5. How does Angular handle memory leaks?

Unsubscribe from observables to prevent memory leaks.
Use ngOnDestroy() and takeUntil() or async pipe.
Detach unused DOM and clear intervals.
Track memory using Chrome DevTools.

Example:

```
this.destroy$.next(); this.destroy$.complete();
```

6. What is Lazy Loading?

Lazy loading loads modules only when needed.
Reduces initial bundle size and improves load time.
Configured in routing using loadChildren.
Each feature module has its own routing.

Example:

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

7. What are ViewChild and ContentChild?

@ViewChild gets reference to a child in the template.
@ContentChild accesses projected content from another component.
Used for DOM or component access.
Can call child methods or read values.

Example:

```
@ViewChild('myDiv') div: ElementRef;
```

8. What are Observables and RxJS?

Observables handle async data streams in Angular.
RxJS provides operators for managing them.
Used for HTTP, event handling, and reactive patterns.
Can be chained and transformed with operators.

Example:

```
this.http.get('/api/data').subscribe(data => console.log(data));
```

9. Error Handling with HttpClient

Use .pipe() with catchError() to handle errors.
Global handling can be done with interceptors.
Return user-friendly messages from service.
Can retry or log errors as needed.

Example:

```
this.http.get('/api/data').pipe(catchError(error => of([])))
```

10. Authentication and Authorization in Angular

Use JWT tokens for authentication.
Store tokens in localStorage or cookies.
Protect routes using guards.
Use interceptors to attach auth headers.

Example:

```
req = req.clone({ setHeaders: { Authorization: `Bearer ${token}` } });
```

11. What is a Route Guard?

Guards prevent unauthorized access to routes.
Types: CanActivate, CanActivateChild, CanLoad, CanDeactivate.
Return true/false or Observable/Promise.
Used for role-based access.

Example:

```
canActivate() { return this.authService.isLoggedIn(); }
```

12. AOT vs JIT Compilation

- AOT: Compiles during build, faster runtime.
- JIT: Compiles in the browser at runtime.
  AOT reduces bundle size and detects errors early.

Use ng build --prod for AOT.

Example:

```
ng build --aot
```

13. Custom Pipes

Transform data in templates.
Create using @Pipe() decorator and implement PipeTransform.
Useful for formatting dates, currencies, etc.
Reusable across components.

Example:

```
@Pipe({ name: 'capitalize' }) transform(value: string) { return value.toUpperCase(); }
```

14. Share Data Between Unrelated Components

Use a shared service with RxJS Subject.
Emit and subscribe to data in different components.
Avoid tight coupling.
Also can use global state management tools.

Example:

```
this.sharedService.sendData('Message');
```

15. Debug Angular Applications

Use Chrome DevTools and Angular DevTools extension.
Use console.log() and breakpoints.
Use ngOnInit to track flow.
Enable source maps for readable code.

Example:

```
console.log('Component loaded');
```

16. Folder Structure Best Practices

Organize by feature (feature modules).
Keep shared, core, services, models in separate folders.
Use consistent naming.
Lazy load heavy modules.

Example:

```
/src/app/
  /shared/
  /core/
  /auth/
```

17. Angular Interceptors

Interceptors modify HTTP requests/responses globally.
Used for adding auth tokens, logging, or handling errors.
Implement HttpInterceptor.
Provide in providers[].

Example:

```
intercept(req, next) { return next.handle(modifiedReq); }
```

18. HTTP Loader Implementation

Use an interceptor to trigger a loader before requests.
Display spinner while request is active.
Hide it in finalize() of the request.

Use shared loader service or component.

Example:

```
return next.handle(req).pipe(finalize(() => this.loader.hide()));
```

19. What are Standalone Components?

Standalone components don't require NgModules.
Declared with standalone: true.
Imported directly in routes or other components.
Introduced in Angular 14+.

Example:

```
@Component({ standalone: true, selector: 'app-header', templateUrl: './header.html' })
```

20. What is the scope?

In Angular, a scope is an object that refers to the application model. It is a context in which expressions can be executed. These scopes are grouped hierarchically, comparable to the DOM structure of the application. A scope aids in the propagation of various events and the monitoring of expressions.

5–15 years' experience :

1. How do you architect large-scale Angular applications?

Divide the app into core, shared, feature, and utility modules.
Use lazy loading for features, implement route guards and services per module.
Enforce coding standards and use consistent folder structure.
Use state management (e.g., NgRx) to handle app-wide data.

Example: Split an e-commerce app into Product, Cart, and User modules with lazy loading.

2. How do you implement state management (NgRx, Akita, Services)?

Use NgRx for predictable, centralized state via actions, reducers, and selectors.
Akita simplifies entity and store management.
Services with RxJS can be used for smaller apps.
Always unsubscribe properly and use selectors for performance.

Example: Use NgRx to manage shopping cart state with actions like addToCart and removeFromCart.

3. What are the SOLID principles in Angular projects?

S: Single responsibility — components should do one thing.
O: Open/closed — use inheritance or DI to extend.
L: Liskov substitution — base types can be replaced with subtypes.
I: Interface segregation — keep interfaces small.
D: Dependency inversion — use DI tokens and services.

Example: Separate data fetching logic into a service to keep the component clean (SRP).

4. How do you implement lazy loading with preloading strategies?

Use Angular router's loadChildren for lazy loading.
Apply PreloadAllModules or custom preloading strategies.
Improves initial load time while preloading other routes.
Use canLoad guards to secure lazy routes.

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

5. Explain the role of a shared module vs core module.

Core module: singleton services, guards, and config used app-wide (import only in AppModule).
Shared module: common components, directives, and pipes (import in multiple modules).

Prevents duplicate service instances and simplifies reusability.
Improves structure and separation of concerns.
Example: **SharedModule contains ButtonComponent; CoreModule has AuthService.**

6. How would you implement role-based access in Angular?

Use route guards like CanActivate to restrict routes based on user roles.
Store user roles in AuthService and check them in the guard.
Optionally hide UI elements based on roles.
Use JWT claims for backend validation.
Example: **Allow only admin users to access /admin-dashboard.**

7. How do you secure Angular applications?

Use HTTPS, sanitize user input, and avoid direct DOM manipulation.
Implement route guards and JWT-based auth.
Use CSP headers and prevent XSS/CSRF attacks.
Hide sensitive keys using environment files.
Example: **Use Angular's DomSanitizer to sanitize URLs in [src].**

8. How do you perform SSR (Server Side Rendering) using Angular Universal?

Use Angular Universal to render the first page on the server.
Improves SEO and time-to-interactive for slow networks.
Install @nguniversal/express-engine and set up Express server.
Render HTML on the server using renderModuleFactory.
Example: **Run ng add @nguniversal/express-engine to add SSR to Angular.**

9. What is the difference between Renderer2 and ElementRef?

ElementRef accesses native DOM directly (not safe).
Renderer2 abstracts DOM manipulation and supports server-side rendering.
Use Renderer2 for better security and portability.
Avoid ElementRef for critical DOM operations.
Example: **this.renderer.setStyle(this.el.nativeElement, 'color', 'blue').**

10. Explain dynamic component loading in Angular.

Use ViewContainerRef.createComponent() to load a component dynamically.
Use ComponentFactoryResolver (pre Angular 13) or createComponent() in Angular 13+.
Useful for modal dialogs or plugins.
Ensure the component is added in the entryComponents (before Angular 9).
Example: **Load a ModalComponent into a ViewContainerRef at runtime.**

11. How do you manage application-wide settings/configuration?

Use environment.ts files for static config per environment.
Create a ConfigService to load dynamic config at startup via APP_INITIALIZER.
Use DI tokens for constants.
Externalize settings in JSON files if needed.
Example: **Load base API URL from environment.prod.ts.**

12. How do you optimize Angular for SEO?

Use Angular Universal for SSR.
Set meta tags and titles using Title and Meta services.
Use robots.txt and sitemap.xml.
Ensure semantic HTML and structured data.
Example: **this.meta.updateTag({ name: 'description', content: 'Best online courses' });**

13. How do you track performance and errors in Angular (Sentry/AppInsights)?

Integrate Sentry or Application Insights in main.ts.
Use interceptors to log HTTP errors.
Track navigation timing, component load, and user actions.
Use custom error handler to capture global errors.
Example: Send exception to Sentry using Sentry.captureException(error) in global handler.

14. What is the role of environment-specific configurations?

Separate configs for dev, prod, staging using environment.ts files.
Handled during build using Angular CLI.
Avoids exposing secrets and enables feature toggles.
Customize logging and error tracking per environment.
Example: Use environment.apiUrl to switch between dev and prod endpoints.

15. How do you structure a multi-tenant Angular app?

Use different modules/services per tenant.
Load tenant-specific configs during bootstrap.
Use dynamic theming and layout per tenant.
Separate authentication and authorization per tenant.
Example: Load tenant-config.json using APP_INITIALIZER and adjust behavior accordingly.

Team Management & Testing

16. How do you implement unit testing and integration testing in Angular?

Use Jasmine and Karma for unit testing.
Test components, services, and pipes in isolation.
Use HttpTestingController for HTTP tests.
Use TestBed for setting up component modules.
Example: Test UserService by mocking HttpClient.

17. What are best practices for writing testable Angular code?

Keep components small and focused.
Use DI for services and avoid tightly coupled logic.
Separate UI and logic.
Mock dependencies in tests.
Example: Inject AuthService via constructor, not create inside component.

18. How do you manage a team of Angular developers?

Set clear code standards and folder structure.
Use Git branching strategies and code reviews.
Encourage modular development and CI/CD pipelines.
Organize regular sync-ups and KT sessions.
Example: Use Angular Style Guide and enforce via lint rules.

19. How do you handle tech debt in Angular applications?

Track tech debt as tasks or backlog items.
Refactor incrementally during feature development.
Prioritize based on impact and cost.
Use tools like SonarQube or ESLint to detect issues.
Example: Migrate deprecated Angular APIs during minor version upgrades.

20. How do you ensure code quality in Angular projects?

Use linting, unit testing, and code reviews.
Enforce CI pipelines with coverage thresholds.

Follow Angular style guide and modular practices.
Monitor performance and bundle sizes.
Example: **Set up GitHub Actions to run ng lint, ng test, and ng build --prod.**

Security 5–10 years' experience

1. How do you prevent XSS (Cross Site Scripting) in Angular?
   Angular auto-sanitizes data binding expressions, preventing XSS.
   Avoid using [innerHTML] unless necessary.
   Use DomSanitizer only when you're confident about the input source.
   Escape user inputs and never trust raw HTML from users.
   Example: **Use {{ userInput }} instead of injecting HTML directly.**
2. What are the risks of using innerHTML and how can you mitigate them?
   innerHTML can inject scripts leading to XSS attacks.
   Angular warns about this by sanitizing untrusted content.
   To allow safe HTML, use DomSanitizer.bypassSecurityTrustHtml() carefully.
   Validate and clean inputs before rendering.
   Example: **<div [innerHTML]="trustedHtml"> after sanitizing.**
3. How does Angular handle CSRF (Cross Site Request Forgery)?
   Angular does not handle CSRF on its own; it's a backend responsibility.
   Use tokens (e.g., anti-CSRF tokens) and validate them server-side.
   Angular can send credentials via withCredentials: true.
   Tokens are validated per session/request.
   Example: **HttpClient.post(url, data, { withCredentials: true }).**
4. How do you secure API requests from Angular frontend?
   Use HTTPS, JWT tokens, and authorization headers.
   Avoid exposing secrets or sensitive data in frontend code.
   Use interceptors to attach tokens.
   Always validate requests server-side.
   Example: **Use Authorization: Bearer <token> in headers.**
5. What is DOM sanitization in Angular and how is it used?
   Angular cleans untrusted values to prevent malicious code execution.
   DomSanitizer can override defaults for safe content.
   Used mainly with [innerHTML], iframes, URLs, etc.
   Should be used cautiously.
   Example: **this.sanitizer.bypassSecurityTrustHtml(htmlContent).**
6. How do you implement JWT authentication securely in Angular?
   Store JWTs in sessionStorage or memory, not localStorage.
   Use interceptors to send tokens in headers.
   Use refresh tokens securely.
   Verify tokens on the backend.
   Example: **req.clone({ setHeaders: { Authorization: Bearer ${token} } }).**
7. How do you protect routes based on roles and permissions?
   Use canActivate route guards with AuthService.
   Check user roles from decoded JWT or state.
   Redirect unauthorized users to a forbidden page.
   Avoid hardcoding roles in frontend.
   Example: **canActivate: [RoleGuardService].**

8. How do you store and secure tokens on the client side?

Use sessionStorage for better security, or in-memory for sensitive data.
Avoid localStorage due to XSS risks.
Consider rotating tokens periodically.
Use refresh tokens securely.
Example: **sessionStorage.setItem('token', jwt).**

9. What is an Angular interceptor and how do you use it for security headers or token refresh?

Interceptors modify outgoing requests.
They attach headers like Authorization or refresh expired tokens.
They handle centralized error logging too.
Useful for implementing retries.
Example: **Add token using req.clone({ setHeaders }).**

10. How do you handle security in dynamic script or style injections?

Avoid injecting scripts/styles unless essential.
Use DomSanitizer if you must trust input.
Host trusted scripts and load them safely.
Never inject user-controlled URLs.
Example: **this.sanitizer.bypassSecurityTrustResourceUrl(url).**

11. What are best practices for handling user input in Angular forms?

Use form validations: Validators.required, etc.
Sanitize or escape user-generated content.
Avoid binding user input directly to DOM.
Prevent script injection via custom inputs.
Example: **<input [formControl]="nameControl"> with validators.**

12. How do you protect against clickjacking in Angular apps?

Set X-Frame-Options: DENY or Content-Security-Policy.
Disable iframe embedding of your site.
Backend (e.g., Nginx/Express) should handle headers.
Angular cannot protect alone.
Example: **res.setHeader('X-Frame-Options', 'DENY');.**

13. What Content Security Policy (CSP) should be used with Angular apps?

Define CSP headers to allow scripts/styles only from trusted sources.
Restrict script-src, style-src, and img-src.
Prevent inline scripts unless hashed.
Always test CSP before enforcing.
Example: **Content-Security-Policy: default-src 'self'; script-src 'self'.**

Deployment

1. What steps do you take before deploying an Angular app to production?

Run ng build --configuration production.
Use AOT, enable optimizations and tree-shaking.
Minify assets and check performance.
Set correct environment file.
Example: **ng build --configuration production.**

2. What is AOT (Ahead-of-Time) compilation and why is it important?

AOT compiles Angular HTML and TypeScript before browser loads.
Improves performance and reduces app size.

Prevents template errors early.
Used by default in production builds.
Example: **AOT is enabled via ng build --prod.**

3. How do you manage environment-specific settings during deployment?

Use angular.json file to configure environments.
Create environment.prod.ts, environment.dev.ts, etc.
Use fileReplacements in build configuration.
Access settings via environment.variableName.
Example: **environment.apiUrl.**

4. What is Angular Universal and how is it used in deployment (SSR)?

Angular Universal enables Server-Side Rendering (SSR).
Improves SEO and initial page load time.
Requires Node.js backend.
Use @nguniversal/express-engine.
Example: **ng add @nguniversal/express-engine.**

5. How do you handle deployment to CDN/static hosts (e.g., Netlify, Firebase, AWS S3)?

Build the app and upload dist folder contents.
Set correct base href and routing fallback.
Use S3 static hosting or Firebase Hosting.
Netlify supports Angular out-of-the-box.
Example: **Deploy dist/ to S3 with public read access.**

6. How do you reduce Angular bundle size for faster load times?

Use lazy loading, AOT, and tree-shaking.
Avoid large libraries; use ES modules.
Remove unused imports/components.
Use ng build --configuration production.
Example: **Use --stats-json to analyze bundle.**

7. What is lazy loading and how does it impact deployment?

Lazy loading loads modules on demand, not at startup.
Reduces initial bundle size.
Improves performance and load time.
Implemented via route-level module splits.
Example: **{ path: 'admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) }.**

8. How do you configure Angular apps behind a reverse proxy (Nginx, Apache)?

Set base href correctly.
Use Nginx to serve index.html on fallback.
Forward API requests to backend.
Set proper CORS headers.
Example (Nginx): **try_files $uri $uri/ /index.html;.**

9. How do you handle 404s in Angular SPA during deployment?

Use catch-all fallback to index.html.
Configure server (Nginx, Apache, etc.) to redirect unknown routes.
SPA routing takes over on client-side.
Ensure route paths are set correctly.
Example: **fallback: index.html in Netlify.**

10. How do you monitor errors and performance after deployment? (e.g., Sentry, AppInsights)

Use external tools like Sentry or Azure App Insights.
Track JavaScript errors, user behavior, and performance.
Install SDK and initialize it in main.ts.
Use interceptors to log HTTP errors.
Example: **Sentry.captureException(error) in ErrorHandler.**

11. What CI/CD pipelines do you use to deploy Angular apps?

Use GitHub Actions, GitLab CI, or Azure DevOps.
Automate build, test, and deployment.
Run ng build --prod and deploy dist/.
Configure caching and artifacts.
Example: **GitHub Action pushing to Firebase.**

12. How do you cache Angular assets using service workers or PWA features?

Use @angular/pwa to enable service worker.
Caches static files and assets.
Improves offline support and performance.
Define cache rules in ngsw-config.json.
Example: **ng add @angular/pwa.**

13. How do you handle versioning and rollback of Angular apps in production?

Use CI/CD tools with versioned builds.
Maintain backup builds in S3 or Firebase.
Rollback by pointing to previous version.
Include app version in environment.ts.
Example: **v1.0.3 in footer fetched from config.**

14. What are the common issues you faced during Angular app deployment and how did you resolve them?

Routing 404s, base href issues, CORS errors, large bundles.
Resolved with server fallback, correct base paths, and interceptors.
Configured CORS on backend APIs.
Used ng build optimizations.
Example: **Fixed AWS S3 403 by enabling public read on index.html.**

.Net Core Interview QA Authentication & Authorization

1. How do you implement JWT authentication in .NET Core Web API?

Use Microsoft.AspNetCore.Authentication.JwtBearer middleware.
Validate tokens using the TokenValidationParameters.
Store claims like roles and username in the JWT.
Attach JWTs to the Authorization header.
Example: **services.AddAuthentication().AddJwtBearer(...).**

2. How do you implement role-based authorization in .NET Core?

Use [Authorize(Roles = "Admin")] on controllers/actions.
Assign roles via Claims during token generation.
Use User.IsInRole("Admin") in code for dynamic checks.
Configure roles in Identity or your own user store.
Example: **[Authorize(Roles = "Manager")].**

3. What is the difference between Claims-based and Role-based Authorization?

Role-based checks user roles like "Admin".

Claims-based checks any user attribute (email, dept, etc.).
Claims provide more flexibility and detail.
Use both together for fine-grained control.
Example: **ClaimTypes.Email, ClaimTypes.Role.**

4. How do you secure sensitive configuration (like DB passwords or API keys)?

Use User Secrets for development and Azure Key Vault for production.
Avoid storing secrets in appsettings.json.
Inject via environment variables or secret managers.
Use IConfiguration to access securely.
Example: builder.Configuration["DbPassword"].

OWASP & Secure Coding Practices

5. How do you protect against SQL Injection in .NET Core?

Use parameterized queries or ORMs like EF Core.
Avoid string concatenation in SQL.
Validate and sanitize user inputs.
Enable query logging to detect unusual activity.
Example: **context.Users.FromSqlRaw("SELECT * FROM Users WHERE Name = @name", name).**

6. How do you prevent Cross Site Scripting (XSS) in .NET Core Web API?

Avoid returning raw HTML or JavaScript from APIs.
Validate and encode user inputs/output on frontend.
Use libraries like System.Text.Encodings.Web.
Use Content-Type: application/json.
Example: **Return clean JSON, not HTML: return Ok(new { message = userInput }).**

7. How do you prevent Cross Site Request Forgery (CSRF) in .NET Core MVC?

Use [ValidateAntiForgeryToken] attribute.
Tokens are auto-included in Razor forms.
CSRF protection isn't needed in pure APIs (if CORS + auth is configured).
Use SameSite cookies to enhance protection.
Example: **<form asp-antiforgery="true">.**

8. What is CORS and how do you configure it in .NET Core?

CORS (Cross-Origin Resource Sharing) allows frontend apps to access APIs on different domains.
Use app.UseCors() with policies for origin, headers, and methods.
Enable only trusted domains.
Avoid AllowAnyOrigin in production.
Example: **builder.Services.AddCors(options => options.AddPolicy("MyPolicy", ...)).**

9. How do you log securely in .NET Core?

Avoid logging sensitive data like passwords or tokens.
Use structured logging (e.g., Serilog, Seq, ELK).
Mask PII (Personally Identifiable Info).
Use log levels wisely: Error, Info, Debug.
Example: **logger.LogInformation("User {UserId} logged in", userId).**

10. How do you mitigate brute-force or dictionary attacks on login APIs?

Limit failed attempts using rate limiting or lockout mechanisms.
Use Identity's LockoutOptions.

Implement CAPTCHA or 2FA.
Use logging and alerts for suspicious behavior.
Example: **Lockout.MaxFailedAccessAttempts = 5.**

Encryption & Data Protection

11. How do you encrypt sensitive data in .NET Core?

Use System.Security.Cryptography for AES/RSA.
Use ASP.NET Core's DataProtectionProvider for storing tokens or cookies.
Store encrypted values in the database.
Never hardcode keys—use secure stores.
Example: **dataProtector.Protect("sensitiveData").**

12. What is the ASP.NET Core Data Protection API?

It helps encrypt and protect data like cookies, tokens, etc.
Keys are stored securely (filesystem, Azure, etc.).
Used automatically in Identity for cookie auth.
You can create custom protectors.
Example: **IDataProtector protector = provider.CreateProtector("purpose").**

13. How do you secure cookies in .NET Core?

Use HttpOnly, Secure, and SameSite=Strict attributes.
Set cookies only over HTTPS.
Avoid storing sensitive data in cookies.
Use Identity or Data Protection for cookies.
Example: options.Cookie.HttpOnly = true.

API Security

14. How do you implement API rate limiting in .NET Core?

Use middleware like AspNetCoreRateLimit.
Limit requests per IP, token, or endpoint.
Prevent DoS attacks and abuse.
Use policies based on roles or user types.
Example: **GeneralRules: [{ Endpoint: "*", Limit: 100, Period: "1m" }].**

15. How do you validate incoming requests for tampering or injection?

Use model validation with [Required], [StringLength], etc.
Use custom validators or middleware.
Reject malformed or suspicious inputs.
Log validation failures.
Example: **[Required, MaxLength(50)] public string Name { get; set; }.**

16. How do you implement IP whitelisting or blocking in .NET Core?

Use middleware to inspect HttpContext.Connection.RemoteIpAddress.
Allow/block IPs from configuration.
Use reverse proxy headers if behind a load balancer.
Example: **Check IP in middleware: if (!allowedIps.Contains(ip)) return 403;.**

Advanced Concepts

17. What is OAuth2 and how is it used in .NET APIs?

OAuth2 is a framework for token-based authorization.
Used with IdentityServer, Azure AD, or external providers.
Scopes define resource access.

Tokens (access/refresh) are used to access protected APIs.

Example: **Use Azure AD with AddMicrosoftIdentityWebApi().**

18. What is OpenID Connect? How is it different from OAuth2?

OpenID Connect is an identity layer on top of OAuth2.
OAuth2 handles authZ; OIDC handles authN (identity).
Used to get user info like name, email, etc.
.NET supports via AddOpenIdConnect().

Example: **services.AddAuthentication().AddOpenIdConnect(...).**

19. How do you implement secure file uploads in .NET Core?

Validate file types and extensions.
Scan files for malware.
Limit file size via model binding and config.
Store files outside web root.

Example: **if (!allowedExtensions.Contains(fileExt)) return BadRequest();.**

20. How do you protect .NET APIs hosted on the cloud (e.g., Azure)?

Use HTTPS, API Management, rate limiting, and WAF (Web App Firewall).
Monitor logs via App Insights.
Secure secrets with Azure Key Vault.
Enable identity providers like Azure AD.

Example: **Protect with API Management policies.**

Dapper Interview Questions

1. How do you prevent SQL Injection in Dapper?

Always use parameterized queries, never interpolate strings.
Dapper supports named parameters with @ syntax.
Avoid dynamic SQL unless absolutely necessary.
Sanitize inputs where needed.

Example: **conn.Query("SELECT * FROM Users WHERE Email = @email", new { email }).**

2. How do you securely handle sensitive data (e.g., passwords) with Dapper?

Store only password hashes (not raw passwords).
Use strong hashing algorithms like bcrypt or PBKDF2.
Dapper just moves data—security must be applied in logic.
Do not log passwords or expose them via API.

Example: **user.PasswordHash = HashPassword(user.Password);.**

3. How do you prevent over-posting or mass assignment with Dapper?

Manually map only required fields during insert/update.
Avoid auto-mapping entire request objects.
Use DTOs with limited properties.
Sanitize user input and whitelist fields.

Example: **Only map Name, Email, not entire User object.**

4. How do you secure connection strings when using Dapper?

Store them in appsettings.json or environment variables.
Encrypt in production using Azure Key Vault or AWS Secrets Manager.
Never hardcode credentials.
Use IConfiguration to fetch securely.

Example: **var connStr = _config.GetConnectionString("DefaultConnection");.**

5. How do you prevent data leakage in Dapper queries?

Limit selected columns—don't use SELECT *.
Paginate large queries to avoid excessive data.
Use role-based filtering in SQL.
Avoid exposing internal DB structure.
Example: SELECT Id, Name FROM Users WHERE Role = @role.

6. How do you implement secure logging with Dapper?

Avoid logging SQL with sensitive parameters.
Use structured logging with masked fields.
Catch and log only expected exceptions.
Use centralized log tools like Serilog/ELK.
Example: **logger.LogWarning("User login failed for {Email}", email);.**

7. How do you handle SQL Exceptions securely in Dapper?

Wrap Dapper calls in try-catch blocks.
Return sanitized error messages to clients.
Log details only internally (not to the frontend).
Avoid exposing DB/server errors.
Example: **catch (SqlException ex) { logger.LogError(ex, "..."); return 500; }.**

Dapper Deployment-Related Interview Questions

1. How do you deploy a .NET application that uses Dapper?

Publish using dotnet publish -c Release.
Include config files for connection strings.
Ensure DB schema matches queries used.
Use CI/CD tools (GitHub Actions, Azure DevOps).
Example: **Deploy via Docker to Azure App Service with pre-configured connection string.**

2. How do you manage environment-based DB connections in Dapper?

Use environment-specific appsettings.{Environment}.json.
Load config via IConfiguration.
Use secrets for production DBs.
Switch connection strings via environment variable.
Example: **builder.Configuration.GetConnectionString("ProdDb").**

3. How do you test Dapper queries before deployment?

Use integration tests with in-memory or test databases.
Write unit tests using mocked IDbConnection.
Validate raw SQL using SQL Profiler.
Ensure safe fallbacks for empty/null values.
Example: **Test GetUsersByRole("Admin") against a test DB.**

4. How do you handle DB schema changes in production with Dapper?

Use tools like FluentMigrator or DbUp.
Keep schema versioning scripts in source control.
Run migrations during deployment via CI/CD.
Backup DB before applying scripts.
Example: **Add AddColumn migration using DbUp.**

5. How do you ensure Dapper query performance before deployment?

Run SQL in SSMS with SET STATISTICS IO ON.
Use indexes and avoid N+1 queries.

Cache frequent results (e.g., Redis, MemoryCache).
Use profiling tools like MiniProfiler.
Example: **Add index on Email column if queried frequently.**

6. How do you enable rollback if a deployment using Dapper fails?

Version SQL scripts with rollback support.
Use DB transactions to group changes.
Have pre/post-deployment backups.
Deploy with blue-green or staging strategies.
Example: Wrap queries in using (var tx = conn.BeginTransaction()).

7. What are the challenges of deploying Dapper-based apps to cloud?

Cloud latency impacts raw SQL calls—optimize queries.
Secure cloud DB credentials via secret stores.
Ensure cloud DB firewall rules allow your app.
Monitor DB connections and pool limits.
Example: **Use Azure Key Vault to store SqlConnection secrets.**

ADO.NET interview questions

1. What is the difference between ExecuteReader, ExecuteScalar, and ExecuteNonQuery?

ExecuteReader returns multiple rows (used for SELECT).
ExecuteScalar returns a single value (e.g., count).
ExecuteNonQuery performs DML (INSERT, UPDATE, DELETE).
Choose based on operation type to optimize performance.
Example: **cmd.ExecuteScalar() to get total user count.**

2. How do you manage database connections efficiently in ADO.NET?

Use using blocks to auto-dispose connections.
Leverage connection pooling by default in .NET.
Avoid keeping connections open for long periods.
Ensure error handling closes connections on failure.
Example:

```
using (SqlConnection conn = new SqlConnection(connStr)) { conn.Open(); /*...*/ }
```

3. What are DataSet and DataReader? When to use each?

DataReader is fast, forward-only, read-only – best for large, fast reads.
DataSet is in-memory, supports disconnected architecture, relations.
Use DataReader when performance is key.
Use DataSet when you need offline or editable data.
Example: **DataSet for report generation.**

4. How do you handle transactions in ADO.NET?

Use SqlTransaction to group multiple commands atomically.
Begin, Commit, or Rollback as needed.
Always wrap in try-catch-finally.
Ensures data consistency.
Example:

```
var tran = conn.BeginTransaction(); cmd.Transaction = tran;
```

5. How do you call stored procedures in ADO.NET?

Set CommandType = CommandType.StoredProcedure.
Add parameters via cmd.Parameters.AddWithValue().
Execute the command normally.

Great for performance and security.

Example:

```
cmd.CommandType = CommandType.StoredProcedure; cmd.CommandText = "GetUsers";
```

ADO.NET Security Questions

6. How do you prevent SQL Injection in ADO.NET?

Use parameterized queries.
Avoid string concatenation in SQL.
Validate and sanitize inputs.
Prefer stored procedures where applicable.

Example:

```
cmd.Parameters.AddWithValue("@Email", email);
```

7. How do you secure database connection strings?

Store in appsettings.json or web.config.
Use encryption or secure vaults (Azure Key Vault).
Do not hardcode credentials.
Avoid exposing connection strings in logs.

Example: **Read via ConfigurationManager.ConnectionStrings["MyDb"].**

8. How do you securely handle exceptions in ADO.NET?

Use try-catch blocks.
Do not show raw DB errors to end-users.
Log detailed errors internally only.
Use custom error messages externally.

Example:

```
catch (SqlException ex) { log.Error(ex); throw new AppException("DB error"); }
```

9. How do you prevent over-posting with ADO.NET?

Manually map input fields to query parameters.
Do not map entire objects blindly.
Use strict validation and field whitelisting.
Avoid auto-mapping tools unless configured tightly.

Example: **Only insert allowed fields from DTO.**

Performance & Deployment

10. How do you improve performance in ADO.NET applications?

Use SqlDataReader for large data.
Keep connection lifespan short.
Use connection pooling and indexing.
Minimize round trips to the DB.

Example: **Batch inserts instead of individual ones.**

11. What are common deployment concerns with ADO.NET apps?

Ensure production DB access is secured.
Use retry logic for transient faults.
Monitor connection pool usage.
Externalize and encrypt sensitive config.

Example: **Use Polly for retry and timeout logic.**

12. How do you prevent connection pool exhaustion?

Close/dispose all connections properly.
Avoid blocking threads holding open connections.

Use correct pool sizing in connection string.
Monitor usage and adjust under load.
Example: **Set Max Pool Size=100 in connection string.**

13. How do you use asynchronous ADO.NET operations?

Use async methods like OpenAsync(), ExecuteReaderAsync().
Improves scalability in web apps.
Use await with proper exception handling.
Avoid blocking calls.
Example:

```
await conn.OpenAsync(); var reader = await cmd.ExecuteReaderAsync();
```

14. How do you cache data fetched with ADO.NET?

Use MemoryCache or distributed cache (Redis).
Cache only frequent, read-heavy queries.
Invalidate cache on updates.
Improves performance and reduces DB load.
Example: **Store result of GetProducts() in cache.**

15. How do you audit DB changes using ADO.NET?

Use DB triggers or explicit insert into audit tables.
Capture username, timestamp, action type.
Wrap audit logic inside transaction.
Consider async logging to avoid delays.
Example: **Call InsertAuditLog(userId, action) after update.**

SQL Server Security Interview Questions

1. How do you implement row-level security in SQL Server?

Row-Level Security (RLS) filters data based on user identity.
It uses Security Policies and Predicate Functions.
Ideal for multi-tenant or sensitive data isolation.
Users only see rows they are allowed to access.
Example:

```
CREATE SECURITY POLICY RowFilter
ADD FILTER PREDICATE dbo.fn_RLSPredicate(UserId) ON dbo.Orders
WITH (STATE = ON);
```

2. What is Transparent Data Encryption (TDE)?

TDE encrypts data at rest (data files, backups).
Helps protect stolen database files.
Uses Database Encryption Key (DEK) and a certificate.
Requires SQL Server Enterprise or Standard Edition.
Example:

```
CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE MyServerCert;
```

3. How do you manage SQL Server logins and roles securely?

Use least privilege principle.
Create custom database roles, not just db_owner.
Avoid using sa or sysadmin unnecessarily.

Use Windows Authentication when possible.

Example:

```
CREATE LOGIN readUser WITH PASSWORD = 'Strong@123';
CREATE USER readUser FOR LOGIN readUser;
EXEC sp_addrolemember 'db_datareader', 'readUser';
```

4. What is the difference between SQL Server Authentication and Windows Authentication?

Windows Authentication is more secure (uses Active Directory).

SQL Authentication stores credentials in SQL Server.

Windows authentication supports Kerberos, Group Policies.

SQL Auth may be needed for external apps or cross-domain use.

Example: **Use Windows Auth for internal apps.**

5. How do you audit security changes in SQL Server?

Use SQL Server Audit, fn_get_audit_file(), or triggers.

Can track login attempts, role changes, schema access.

Store audits in a secure location.

Use Audit Specifications at server/database level.

Example:

```
CREATE SERVER AUDIT LoginAudit TO FILE (FILEPATH = 'C:\AuditLogs\');
CREATE SERVER AUDIT SPECIFICATION LoginAuditSpec
FOR SERVER AUDIT LoginAudit
ADD (FAILED_LOGIN_GROUP);
```

6. How do you protect against SQL Injection in stored procedures?

Use parameterized queries and avoid dynamic SQL.

Sanitize and validate user inputs.

Use QUOTENAME() and sp_executesql for dynamic logic.

Avoid string concatenation inside procedures.

Example:

```
EXEC sp_executesql N'SELECT * FROM Users WHERE UserId = @id',
N'@id INT', @id = 1;
```

7. What is Always Encrypted in SQL Server?

Encrypts sensitive data (e.g., SSN) client-side.

SQL Server cannot read data, only client with keys can.

Uses Column Master Key (CMK) and Column Encryption Key (CEK).

Great for PII compliance.

Example: **Encrypt SSN column using Always Encrypted in SSMS.**

8. How do you implement SSL encryption for SQL Server connections?

Install an SSL certificate on the server.

Enable encryption using SQL Server Configuration Manager.

Force encryption for clients.

Protects data-in-transit from sniffing.

Example:

Set Force Encryption = Yes under SQL Server Network Configuration.

9. How do you manage database permissions for developers vs DBAs?

Use separation of duties.

Developers: read/write/test roles; no schema/control access.

DBAs: sysadmin for deployment, monitoring.

Use schemas and roles to isolate areas.

Example:

Grant EXEC only on specific procs to devs.

10. How do you detect and block brute-force login attempts?

Use SQL Server Audit or log monitoring.
Track failed login counts from IP/user.
Configure Windows Group Policy or firewall to block IPs.
Consider enabling account lockout policies.
Exa**mple**:
Monitor xp_readerrorlog for repeated login failures.

Entity Framework Interview Questions

1. EF6 vs EF Core

EF Core is cross-platform, lightweight, and faster than EF6.
EF Core supports LINQ improvements, shadow properties, and batch updates.
EF6 is mature but limited to .NET Framework and lacks modern features.
EF Core has better performance and supports NoSQL providers.
Example: **EF Core allows `HasData()` seeding, EF6 does not.**

2. Repository & Unit of Work in EF Core

Repository abstracts data access; UoW manages transactions.
Create generic repository interfaces and a unit of work class.
Inject `DbContext` into repositories for centralized management.
Promotes testability and separation of concerns.
Example: **`IRepository<T>.Add(T entity)` calls `_dbContext.Set<T>().Add(entity)`.**

3. DbContext Lifecycle in Web App

DbContext is scoped per HTTP request in ASP.NET Core.
Use `AddDbContext<>()` with `ServiceLifetime.Scoped`.
Avoid long-lived DbContexts to prevent memory leaks.
Dispose DbContext automatically via DI.
Example**: Injected `ApplicationDbContext` in controller is disposed after request.**

4. Handling Concurrency Conflicts

Use `[Timestamp]` or concurrency tokens in EF models.
Catch `DbUpdateConcurrencyException` and resolve manually or retry.
EF compares original vs current values before save.
Concurrency ensures last writer wins or user confirms.
Example: **Add a `byte[] RowVersion` property with `[Timestamp]`.**

5. Change Tracking in EF

EF tracks entity changes automatically for updates.
Use `AsNoTracking()` for read-only queries to boost performance.
Disable tracking globally if needed for performance.
Tracked entities are marked Modified, Added, or Deleted.
Example: **`_context.Users.AsNoTracking().First()` won't be tracked.**

6. Soft Deletes with EF

Add `IsDeleted` bool to entities and filter queries globally.
Use EF Core query filters for soft delete behavior.
Avoids physical deletion but hides soft-deleted data.
Maintain audit and restore options.

Example: `modelBuilder.Entity<User>().HasQueryFilter(u => !u.IsDeleted);`

7. Shadow Properties

Properties not defined in class but tracked in model.
Configured using Fluent API in EF Core.
Used for audit fields like CreatedBy or ModifiedDate.
Accessible via `context.Entry(entity).Property("PropertyName")`.
Example: `builder.Property<DateTime>("LastUpdated");`

8. Structuring Large Domain Models

Use modular `DbContext`, split model files using Fluent API.
Apply Feature Folders or Bounded Context design.
Separate read/write models using CQRS.
Avoid bloated context files and tight coupling.
Example: **Use `OnModelCreatingPartial()` for external configuration classes.**

9. HasData() vs Seed()

`HasData()` seeds data during `migrations`.
`Seed()` in EF6 was part of `DbInitializer`, EF Core removed it.
`HasData()` is used in `OnModelCreating`.
Good for static reference data.
Example: `modelBuilder.Entity<Role>().HasData(new Role { Id=1, Name="Admin" });`

10. Migrations Across Environments

Use environment-specific migrations and configurations.
Apply migrations during CI/CD with tools like `dotnet ef`.
Track migration history using `_EFMigrationsHistory`.
Avoid applying dev-only changes to prod.
Example: **Use `dotnet ef database update --environment Production`.**

11. What are common performance issues in EF and how do you address them?

Issues include N+1 queries, untracked entities, over-fetching data, and missing indexes.
Use eager loading, projections (DTOs), and pagination.
Disable tracking with AsNoTracking() for read-only data.
Optimize SQL with indexes and avoid unnecessary joins.
Example: Use .Select(x => new UserDto { x.Id, x.Name }) instead of fetching full entity.

12. Explain the N+1 query problem and how to solve it in EF.

N+1 occurs when querying parent and lazy-loading children, causing multiple SQL calls.
Fix using eager loading (Include) or projection.
Reduces round trips and improves performance.
Use profiling tools to detect N+1 patterns.
Example: context.Orders.Include(o => o.OrderItems) solves N+1 problem.

13. When and how should you use AsNoTracking()?

Use it for read-only operations to avoid tracking overhead.
Improves performance in list/reporting queries.
Tracked entities consume memory and CPU.
Don't use AsNoTracking() if you plan to update entities.
Example: _context.Products.AsNoTracking().ToList() for readonly display.

14. How do you use compiled queries in EF Core?

Compiled queries improve performance by caching query plans.

Use EF.CompileQuery() or EF.CompileAsyncQuery().
Ideal for frequently run queries in high-traffic apps.
Avoids overhead of LINQ-to-SQL translation every time.
Example: **_getProducts = EF.CompileQuery((MyDb ctx) => ctx.Products.Where(p => p.IsActive));**

15. How can you batch multiple commands in a single SaveChanges()?

EF Core supports batching inserts/updates/deletes automatically.
Call SaveChanges() once after multiple changes.
Reduces DB round trips and improves throughput.
Keep DbContext lifetime short to avoid stale tracking.
Example: **Add 10 users, then call _context.SaveChanges() once.**

16. What's the impact of lazy vs eager vs explicit loading?

Lazy loads related data on access—may cause N+1 issues.
Eager loading fetches data using Include()—better for full object graphs.
Explicit loading fetches manually using .Entry().Collection().Load().
Use eager loading for known dependencies and lazy for rare access.
Example: **_context.Orders.Include(o => o.Customer).ToList() = eager.**

17. How do you log and profile SQL queries generated by EF Core?

Use built-in logging with ILoggerFactory or tools like EF Profiler.
Enable logging in DbContextOptionsBuilder.
Can also log to console or Application Insights.
Helpful for optimizing queries and detecting performance issues.
Example: **optionsBuilder.LogTo(Console.WriteLine) in OnConfiguring().**

18. What is the best way to use projection (DTOs) with LINQ in EF Core?

Project only needed fields using Select() into DTOs.
Avoid loading full entity graph for simple views.
Boosts performance and avoids circular references.
Keeps separation between domain and UI models.
Example: **Select(x => new UserDto { x.Id, x.Name }) instead of ToList() on full entity.**

19. How do you protect against SQL injection with EF Core?

EF Core uses parameterized queries internally to prevent SQL injection.
Avoid raw SQL or validate inputs strictly when using FromSqlRaw().
Use FromSqlInterpolated() instead of string concatenation.
Never expose user input directly in query strings.
Example: **context.Users.FromSqlInterpolated($"SELECT * FROM Users WHERE Name = {name}").**

20. What are the risks of dynamic LINQ and how do you secure it?

Dynamic LINQ allows runtime string-based expressions—risk of injection.
Always validate or sanitize expressions from user input.
Prefer strongly-typed LINQ or restricted dynamic expressions.
Use expression trees for safe dynamic query building.
Example: **Use System.Linq.Dynamic.Core with predefined field whitelist.**

21. How do you handle sensitive data (encryption or secure columns) in EF?

Encrypt data before saving and decrypt after reading using value converters.
Avoid storing plain-text passwords or sensitive info.
Store encryption keys securely (e.g., Azure Key Vault).

Use custom converters in OnModelCreating().

Example: builder.Property(p => p.SSN).HasConversion(encrypt, decrypt);

22. How do you implement and manage transactions in EF Core?

Use DbContext.Database.BeginTransaction() for manual control.
SaveChanges() is transactional by default for a single context.
Use TransactionScope for cross-context or DB operations.
Always commit or rollback explicitly in manual transactions.

Example:

```
using var tx = context.Database.BeginTransaction();
context.Add(...);
context.SaveChanges();
tx.Commit();
```

23. How can you ensure audit logging in EF (created/modified info)?

Override SaveChanges() or use ChangeTracker to update audit fields.
Set CreatedBy, CreatedOn, ModifiedBy, ModifiedOn.
Use an IAuditableEntity interface or base entity.
You can also log changes externally for compliance.

Example:

```
entry.Property("ModifiedOn").CurrentValue = DateTime.UtcNow;
```

24. How do you mock DbContext and DbSet for unit testing?

Use interfaces or mock libraries like Moq to mock DbSet.
Use Queryable mocks to simulate LINQ queries.
Avoid mocking EF internals—test services using in-memory DbContext.
Focus on mocking at the repository/service layer.

Example: var mockSet = new Mock<DbSet<User>>(); mockSet.As<IQueryable<User>>()...

25. How do you test LINQ queries with an in-memory database?

Use UseInMemoryDatabase in DbContextOptionsBuilder.
Allows fast tests without actual DB dependency.
In-memory provider may behave differently than SQL Server.
Best for basic CRUD and logic validation.

Example: options.UseInMemoryDatabase("TestDB");

26. What are alternatives to EF Core's InMemory provider for integration tests?

Use SQLite in-memory mode for closer SQL Server behavior.
It supports transactions and relational features.
Good for testing real queries and foreign keys.
Prefer SQLite for integration tests; InMemory for unit tests.

Example: options.UseSqlite("Filename=:memory:");

27. What challenges have you faced during EF migrations in production?

Issues include data loss, downtime, and schema mismatch.
Use backups, scripts, and pre-prod validations.
Split destructive and additive changes.
Avoid automatic migrations on production startup.

Example: Use dotnet ef migrations script to review before deploying.

28. How do you rollback EF migrations safely?

Use Remove-Migration during development if not applied.
For applied migrations, create reverse scripts manually.

Consider custom rollback scripts for critical data.
Use versioned migrations for predictable state.
Example: **dotnet ef migrations script from_mig to_mig --output rollback.sql**

29. How do you migrate a legacy ADO.NET app to EF Core gradually?

Start with adding EF in parallel using Database-First or Reverse Engineering.
Wrap legacy queries in services and refactor module-wise.
Introduce EF-based models gradually to avoid full rewrite.
Ensure same connection and transaction strategy.
Example: **Use Scaffold-DbContext to generate EF models from DB.**

30. How do you ensure EF migration compatibility with CI/CD?

Run dotnet ef migrations script in build pipelines.
Validate DB scripts with pre-prod testing.
Use tools like Flyway or DbUp for migration orchestration.
Store migration files in source control.
Example: **Azure DevOps runs DB migration stage before API deployment.**

Azure Basics (2 Years Experience)

1. What is Microsoft Azure?

Azure is a cloud computing platform by Microsoft offering IaaS, PaaS, and SaaS.
It provides scalable services like VMs, databases, AI, DevOps, and storage.
Users pay only for what they use.
Azure supports hybrid and multi-cloud scenarios.
Example: **Host a .NET Web API on Azure App Service.**

2. What are Azure Resource Groups?

A Resource Group is a container that holds related resources (VMs, DBs, etc.).
It helps manage, monitor, and deploy resources as a group.
All resources in a group share the same lifecycle.
You can apply RBAC and policies at the group level.
Example: **A resource group for your "Employee Portal" project.**

3. What is the difference between Azure App Service and Azure Function?

Azure App Service is for hosting web apps, REST APIs, etc.
Azure Functions are event-driven, used for small background tasks.
App Service is ideal for full-featured web hosting.
Functions are serverless and autoscale on demand.
Example: **Use App Service for the UI, Azure Function for file uploads.**

4. How does Azure handle high availability?

Azure provides availability zones and regions for redundancy.
Load balancers, auto-scaling, and failover systems help maintain uptime.
You can deploy services in active-active or active-passive mode.
SLA guarantees vary by service type.
Example: **Host VMs in different availability zones for 99.99% uptime.**

5. What is Azure Blob Storage?

Blob Storage is used for storing unstructured data like images, videos, and files.
It supports Hot, Cool, and Archive tiers for cost control.
Access via SDK, REST API, or Azure Portal.
Blob types: block, append, and page blobs.
Example: **Store profile pictures in Azure Blob Storage.**

6. How do you secure an Azure Web App?

Use HTTPS, Azure Front Door, and Authentication (AAD).
Apply IP restrictions, private endpoints, and app-level firewalls.
Enable DDoS Protection and Web Application Firewall (WAF).
Use Managed Identity for secure resource access.
Example: **Restrict access to the app only from corporate IPs.**

7. What is Azure Key Vault?

Key Vault securely stores secrets, keys, and certificates.
Apps can access secrets without hardcoding credentials.
It integrates with Azure services and supports RBAC.
Key Vault supports audit logging and soft-delete.
Example: **Store a DB connection string in Key Vault.**

8. What is Azure Virtual Network (VNet)?

VNet is Azure's private network for resources like VMs and DBs.
It supports subnets, firewalls, peering, and VPN gateways.
Allows secure communication between Azure and on-premises.
Provides network isolation and control.
Example: **Connect your Web App and SQL DB within a single VNet.**

9. What is Azure Monitor and Application Insights?

Azure Monitor collects metrics and logs for all Azure resources.
Application Insights monitors app performance and exceptions.
Both support dashboards, alerts, and diagnostics.
They integrate with DevOps and Log Analytics.
Example: **Use App Insights to monitor response times in your API.**

10. How do you deploy applications to Azure?

Use Azure Portal, Azure CLI, GitHub Actions, or Azure DevOps.
Supports ARM templates, Bicep, or Terraform for IaC.
Continuous Deployment automates deployments after code push.
Set up deployment slots for staging and testing.
Example: **Use GitHub Actions to deploy an Angular app to Azure Static Web Apps.**

C# Interview Questions (6–12 Years Experience)

1. What is the output of the following code?

```
string a = "hello";
string b = "hello";
Console.WriteLine(object.ReferenceEquals(a, b));
```

Expected Output: **True**

Explanation: String interning in C# makes both point to the same reference.

2. What will be the output?

```
int x = 0;
Console.WriteLine(x++ + ++x);
```

Output: **2**

Explanation: x++ returns 0, then ++x increments to 2 → 0 + 2 = 2.

3.

4. What does this code do?

```
int[] arr = {1, 2, 3, 4, 5};
var result = arr.Where(x => x % 2 == 0).Select(x => x * x);
```

Result: [4, 16]

Explanation: Filters even numbers and squares them.

4. What is the difference between these two lines?

```
var a = new List<int>();
IEnumerable<int> b = new List<int>();
```

Explanation: a is mutable List; b only allows iteration (read-only view).

Use Case: IEnumerable is good for abstraction and LINQ.

5. What will be printed here?

```
string str = null;
Console.WriteLine(str?.Length ?? 0);
```

Output: 0

Explanation: Null-conditional operator avoids exception, uses fallback ??.

6. Does this compile? If yes, what does it print?

```
static void Main()
{
   Print();
   void Print() => Console.WriteLine("Hello");
}
```

Output: Hello

Explanation: Local function inside Main is valid in C# 7+.

7. Predict output:

```
int x = 5;
object y = x;
x = 10;
Console.WriteLine(y);
```

Output: 5

Explanation: Boxing copies value type to object; later changes to x don't affect y.

8. What happens here?

```
Task.Run(() => throw new Exception("Test"));
Console.WriteLine("Done");
```

Output: Done, but exception is unobserved unless awaited.

Best Practice: Always await or handle background task exceptions.

9. Difference in behavior:

```
StringBuilder sb1 = new StringBuilder("Hi");
StringBuilder sb2 = sb1;
sb2.Append(" there");
Console.WriteLine(sb1.ToString());
```

Output: Hi there

Explanation: StringBuilder is a reference type; both sb1 and sb2 point to the same object.

10. What's the output?

```
var list = new List<int> {1, 2, 3};
foreach (var i in list)
{
   Console.WriteLine(i);
   list.Add(4); // Modifying during iteration
}
```

Result: Throws InvalidOperationException

Explanation: You can't modify a collection while iterating over it.

Thanks for reading all these. Follow Sai Reddy for more such posts.