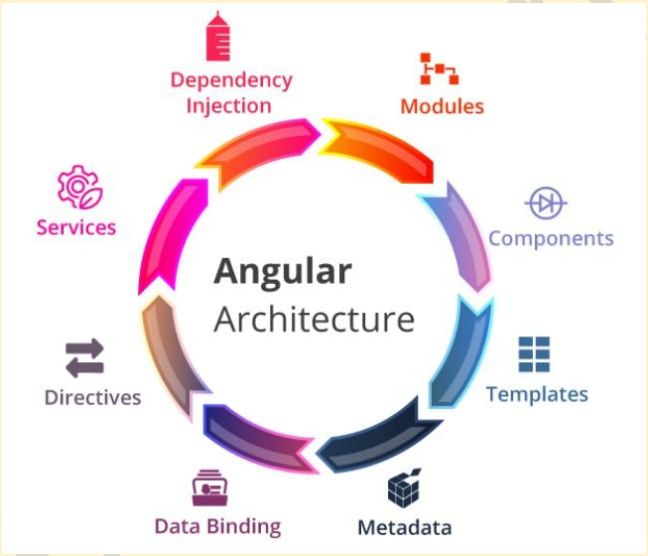

100+ Most Asked Angular

Interview Questions and Answers

Sai Reddy

6/20/25

Interview Tips

100+ Most Asked Angular Interview Q&A

0–2 years Angular interview questions

1. What is Angular and how is it different from AngularJS?

Angular is a modern TypeScript-based front-end framework, whereas AngularJS is JavaScript-based. Angular uses a component-based architecture while AngularJS uses controllers and scopes. Angular is faster and supports mobile, unlike AngularJS.

Example: @Component({ selector: 'app-root', templateUrl: './app.component.html' })

Angular uses decorators and ES6+ features for building apps.

2. **Explain the architecture of an Angular application.**

   Angular architecture includes modules, components, templates, services, and routing. Components control views, and services handle logic. Modules group related components.

   Example: @NgModule({ declarations: [AppComponent], imports: [BrowserModule] })

   The root module is AppModule.

3. **What is a component in Angular?**

   A component controls a part of the UI with a class, template, and style. It's defined using @Component decorator.

   Example:

```
@Component({ selector: 'app-hello', template: '<h1>Hello</h1>' })
export class HelloComponent {}
```

4. **How do you create and use services in Angular?**

   Services are created using @Injectable() and used for business logic or HTTP calls.

   Example:

```
@Injectable()
export class DataService {}
constructor(private ds: DataService) {}
```

5. **What is two-way data binding?**

   It allows synchronization of data between the component and template using [(ngModel)].

   Example:

```
<input [(ngModel)]="name">
<p>{{name}}</p>
```

6. **What are directives in Angular? Name a few built-in directives.**

   Directives modify DOM behavior. Built-in directives include *ngIf, *ngFor, ngClass.

   Example:

```
<p *ngIf="isVisible">Visible</p>
```

7. **How does Angular handle forms (Template vs Reactive)?**

   Template-driven forms use HTML with [(ngModel)], reactive forms use FormGroup, FormControl.

   Example (Reactive):

```
this.form = new FormGroup({ name: new FormControl('') });
```

8. **What is the role of ngOnInit() in a component?**

   ngOnInit() is a lifecycle hook that runs after component initialization.

   Example:

ngOnInit() { this.getData(); }

9. **What is a module in Angular?**
   A module is a container for components, directives, and services. Defined using @NgModule.
   Example:

@NgModule({ declarations: [AppComponent], imports: [BrowserModule] })

10. **How do you handle events in Angular templates?**
    Use (event) binding in the template to trigger component methods.
    Example:

<button (click)="sayHello()">Click</button>

11. **What is dependency injection?**
    It's a design pattern to provide dependencies via constructor.
    Example:

constructor(private myService: MyService) {}

12. **How do you apply conditional rendering in Angular templates?**
    Use *ngIf to conditionally render elements.
    Example:

<div *ngIf="isLoggedIn">Welcome

13. **What is the purpose of the *ngFor directive?**
    It loops through arrays and renders elements.
    Example:

<li *ngFor="let item of items">{{item}}

14. **How does routing work in Angular?**
    Angular uses RouterModule to navigate between components.
    Example:

const routes = [{ path: 'home', component: HomeComponent }];

15. **What is HttpClientModule and how is it used?**
    It's used to make HTTP calls. Import it in AppModule.
    Example:

this.http.get('api/data').subscribe();

16. **How do you call REST APIs from Angular?**
    Use HttpClient service and get(), post() methods.
    Example:

this.http.get('url').subscribe(data => this.result = data);

17. **What are pipes in Angular?**
    Pipes transform data in templates.
    Example:

{{ today | date:'shortDate' }}

18. **What is the difference between @Input and @Output?**
    @Input passes data to child, @Output sends event to parent.
    Example:

@Input() name: string; @Output() clicked = new EventEmitter();

19. **What is interpolation in Angular?**
    It binds data from class to template using {{ }}.
    Example:

```
<p>{{ title }}</p>
```

20. **What are lifecycle hooks in Angular?**
    They are methods that run at different stages (e.g. ngOnInit, ngOnDestroy).
    Example:

```
ngOnDestroy() { console.log("Cleaned up!"); }
```

21. **What is a template reference variable?**
    It's a variable that gives access to a DOM element.
    Example:

```
<input #nameRef> <button (click)="log(nameRef.value)">Log</button>
```

22. **How do you apply custom CSS styles to a component?**
    Use styleUrls or inline styles in @Component.
    Example:

```
@Component({ styleUrls: ['./app.component.css'] })
```

23. **What are the main files generated by Angular CLI?**
    Files include app.component.ts, app.module.ts, main.ts, etc.
    Example:
    ng new my-app creates these files.
24. **How do you install Angular and create a new project?**
    Use Angular CLI: npm install -g @angular/cli then ng new app-name.
    Example:

```
ng serve
```

25. **What is the use of ngIf directive?**
    It conditionally includes/excludes elements in the DOM.
    Example:

```
<div *ngIf="isVisible">Visible Block</div>
```

3–6 Years Experience

1. How does Angular's change detection mechanism work?

Angular uses zone.js to detect async operations and trigger change detection. It compares component model and view using the component tree. Default strategy checks all components; OnPush checks only when inputs change.

```
@Component({ changeDetection: ChangeDetectionStrategy.OnPush })
```

2. What is the difference between ViewChild and ContentChild?

ViewChild accesses elements inside a component's view, while ContentChild accesses projected content. ViewChild is for template DOM; ContentChild is for ng-content.

```
@ViewChild('inputRef') input; @ContentChild('contentRef') content;
```

3. Explain the use of Reactive Forms and FormBuilder.

Reactive Forms give control via code. FormBuilder simplifies group/control creation. You can validate dynamically and handle complex form logic.

```
this.form = this.fb.group({ name: ['', Validators.required] });
```

4. What are Observables and how does Angular use RxJS?

Observables represent asynchronous streams. Angular uses them for HTTP, forms, events. RxJS provides operators like map, filter, switchMap.

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

5. How do you handle HTTP errors in Angular?

Use RxJS catchError inside pipe() to catch errors. Inject HttpClient and use throwError to propagate error.

this.http.get('url').pipe(catchError(err => throwError(() => err)));

6. What is lazy loading and how do you implement it?

Lazy loading loads modules on demand using routes. Improves load time by splitting bundles.

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

7. How can you optimize an Angular application's performance?

Use OnPush change detection, lazy loading, trackBy, and AOT. Avoid unnecessary bindings, use memoization.

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

8. Explain route guards and their types.

Guards control access to routes. Types: CanActivate, CanDeactivate, Resolve, CanLoad.

canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean

9. What is the difference between ngOnChanges and ngDoCheck?

ngOnChanges responds to input property changes. ngDoCheck allows custom change detection.

ngOnChanges() {} ngDoCheck() { console.log('Checked'); }

10. How do you write and run unit tests for components/services?

Use Jasmine and Karma. Use TestBed to configure module. Write specs using it(), expect().

it('should create', () => { expect(component).toBeTruthy(); });

11. How does Angular handle dependency injection hierarchies?

Angular creates injector hierarchies. Root-level services are singleton across app. Component providers override parent.

providers: [MyService] // in component

12. What are modules and sub-modules in Angular?

Modules group features. Sub-modules encapsulate domain logic like AdminModule.

@NgModule({ declarations: [AdminComponent], imports: [CommonModule] })

13. Explain dynamic component loading in Angular.

Use ViewContainerRef and ComponentFactoryResolver. Load component at runtime.

vcRef.createComponent(this.cfr.resolveComponentFactory(MyComponent));

14. What is Ahead-of-Time (AOT) compilation?

AOT compiles Angular HTML & TypeScript during build, not browser. Reduces bundle size, improves load.

ng build --aot

15. How do you secure Angular routes using AuthGuard?

Implement CanActivate and check login status before navigation.

canActivate(): boolean { return this.authService.isLoggedIn(); }

16. What is Angular Universal and why is it used?

Angular Universal allows server-side rendering. Boosts SEO and improves first load time.

Sai Reddy
saireddy-dotnetfs

npm run build:ssr && npm run serve:ssr

17. Explain the async pipe and its usage.

async subscribes to Observables and returns emitted values. Auto-unsubscribes.

{{ data$ | async }}

18. What is the purpose of ng-template and ng-container?

ng-template defines template blocks; ng-container groups without extra DOM.

<ng-template #tmpl> Hi </ng-template>

<ng-container *ngIf="true">Visible</ng-container>

19. What is the difference between Promise and Observable?

Promise emits once, Observable emits multiple values. Observables support cancelation.

Observable.of(1,2,3).subscribe();

Promise.resolve(1).then();

20. How does Angular handle cross-component communication?

Use services with Subject, @Input/@Output, or EventEmitter.

this.subject.next(data); this.subject.asObservable().subscribe();

21. What is a resolver in Angular routing?

Resolvers fetch data before route activation. Ensures data is ready.

resolve(): Observable<Data> { return this.api.getData(); }

22. How do you use interceptors for HTTP calls?

Use HttpInterceptor to intercept requests/responses globally.

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> { return next.handle(req); }

23. What are custom pipes and how do you create them?

Create pipes with @Pipe() and implement transform().

@Pipe({name: 'capitalize'})

transform(value: string): string { return value.toUpperCase(); }

24. Explain the use of trackBy in *ngFor.

trackBy improves performance by tracking identity. Prevents full DOM re-render.

<li *ngFor="let item of items; trackBy: trackByFn">{{item.name}}

25. What is differential loading?

Angular builds separate bundles for modern and legacy browsers. Improves performance.

"target": "es2015", "browserslist": [...]

Let me know if you want the next set for **6–10 years**, or want these in **PDF/Word format** for distribution.

6–10 Years Experience

1. Explain the full bootstrapping process in Angular.

Angular bootstraps via main.ts, calling platformBrowserDynamic().bootstrapModule(AppModule). Angular initializes modules, injectors, and the root component. This process binds Angular to the index.html.

platformBrowserDynamic().bootstrapModule(AppModule);

2. How does Angular handle memory leaks?

Memory leaks are prevented by unsubscribing from Observables, destroying components properly, and avoiding DOM leaks. Use takeUntil, async pipe, and ngOnDestroy.

ngOnDestroy() { this.sub.unsubscribe(); }

3. What are the best practices for scalable Angular architecture?

Use modular structure (core, shared, feature), lazy loading, smart/dumb components, and central state management. Follow SOLID and separation of concerns.

/app/core /app/shared /app/features

4. How do you create a shared module and why is it useful?

SharedModule contains common components, pipes, and modules reused across the app. Import it in feature modules.

@NgModule({ exports: [CommonModule, MyPipe] }) export class SharedModule {}

5. How does Angular Universal affect SEO?

Angular Universal pre-renders HTML on the server, making it crawlable by search engines. It improves SEO and initial load performance.

npm run build:ssr && npm run serve:ssr

6. What is NgRx and how does it differ from services?

NgRx uses Redux pattern for global state management with immutability. Services use imperative logic; NgRx is reactive and centralized.

store.dispatch(loadData()); store.select(getData$).subscribe();

7. How do you structure a large enterprise Angular app?

Split into core, shared, feature, and routing modules. Use state management and strict coding standards.

/src/app/features /core /shared /state

8. What is the Renderer2 service in Angular?

Renderer2 provides platform-independent DOM manipulation, ensuring compatibility across environments.

constructor(private renderer: Renderer2) {}

this.renderer.setStyle(el, 'color', 'red');

9. What are the implications of OnPush change detection?

OnPush improves performance by checking only on @Input change or events. Helps avoid unnecessary re-renders.

@Component({ changeDetection: ChangeDetectionStrategy.OnPush })

10. How does SSR (Server-Side Rendering) work with Angular?

Angular renders HTML on server and sends to browser. Browser then takes over with hydration. Used via Angular Universal.

ng add @nguniversal/express-engine

11. What are view encapsulation strategies in Angular?

ViewEncapsulation options are Emulated (default), ShadowDom, None. It controls how CSS is scoped.

@Component({ encapsulation: ViewEncapsulation.Emulated })

12. How do you handle feature toggling in Angular apps?

Use config service or environment file to toggle features at runtime. Conditional checks in template and logic.

*ngIf="featureService.isEnabled('featureX')"

13. How do you implement progressive web app (PWA) features in Angular?

Use @angular/pwa to add service workers, offline support, manifest, and caching.

ng add @angular/pwa

14. What is a BehaviorSubject and when would you use it?

BehaviorSubject emits the latest value to new subscribers. Ideal for shared state across components.

subject = new BehaviorSubject<string>('init'); subject.next('new');

15. How do you handle access control across Angular modules?

Use route guards with AuthService and roles. Apply canActivate at route level.

canActivate(): boolean { return user.hasRole('admin'); }

16. How to integrate Angular with WebSockets?

Use WebSocketSubject from RxJS or native WebSocket API. Handle messages via observables.

socket = new WebSocket('ws://localhost'); socket.onmessage = e => ...

17. What is dynamic form creation in Angular?

Use FormGroup, FormControl, and FormArray to build forms programmatically. Useful for runtime-generated fields.

this.form = new FormGroup({ name: new FormControl('') });

18. How do you manage state in Angular without NgRx?

Use services with BehaviorSubject, local state in components, and RxJS for communication.

state$ = new BehaviorSubject<any>(null); state$.next(data);

19. How can Angular be used with micro frontends?

Use module federation (Webpack 5), lazy-loaded routes, or web components. Each Angular app acts as a self-contained unit.

loadChildren: () => import('remote/AppModule').then(m => m.AppModule)

20. What are template-driven vs reactive-driven validations?

Template-driven uses HTML with directives; reactive uses TypeScript logic. Reactive is more testable and scalable.

form = new FormGroup({ name: new FormControl('', Validators.required) });

21. What is Angular schematics and how do you use it?

Schematics automate Angular code generation and project scaffolding. Use CLI or create custom schematics.

ng generate component my-comp

22. Explain DI tokens and multi-providers.

DI tokens are used to inject values by key. Multi-providers allow multiple providers under same token.

{ provide: 'CONFIG', useValue: {...}, multi: true }

23. How to handle complex nested routing scenarios?

Use children routes and router-outlet nesting. Helps manage sub-features in modules.

{ path: 'admin', children: [{ path: 'users', component: UsersComponent }] }

24. How to test HTTP interceptors?

Use HttpClientTestingModule and provide the interceptor. Mock requests using HttpTestingController.

httpMock.expectOne('api/test').flush({});

25. What are custom decorators, and how do you implement them?

Custom decorators are functions that modify class/method behavior. Useful for metadata and reusable logic.

function Log(target, key) { console.log('Called: ' + key); }

10–15 Years Experience

1. Design an Angular architecture for an enterprise-grade app.

Use a modular structure with core, shared, and feature modules. Use lazy loading, state management (NgRx/Akita), services for business logic, and configuration via environments.

/app/core /app/shared /app/modules/auth /app/modules/dashboard

2. How do you manage cross-cutting concerns in Angular?

Use interceptors for HTTP, services for logging, route guards for auth, and base components. Apply decorators or base classes to centralize logic.

@Injectable() export class AuthInterceptor implements HttpInterceptor {}

3. How do you implement internationalization (i18n) in Angular?

Use Angular's @angular/localize or ngx-translate for dynamic language loading. Store keys in JSON files and load based on user selection.

translate.use('fr'); {{ 'HOME.TITLE' | translate }}

4. How does Angular interact with third-party libraries?

Install the library via npm, import required modules, and use in components or services. Integrate via directives or wrappers if needed.

import * as moment from 'moment'; moment().format()

5. What is the role of NgModuleFactoryLoader?

It was used for loading modules lazily before Ivy. Now deprecated; Ivy simplifies lazy loading via dynamic import syntax.

loadChildren: () => import('./feature/feature.module').then(m => m.FeatureModule)

6. How do you profile and optimize Angular performance?

Use Chrome DevTools, Angular Profiler, source-map explorer, and Lighthouse. Optimize change detection, bundle sizes, lazy loading, and avoid memory leaks.

npm run build --prod --stats-json

7. How do you manage configuration and environment-based settings in Angular?

Use environment.ts files and inject values using APP_INITIALIZER or services. Avoid hardcoded values.

environment.apiUrl; // used in services

8. How do you implement advanced lazy loading with route preloading?

Use custom PreloadingStrategy for selective module preloading. Improves UX by loading modules in the background.

canLoad: () => this.auth.canLoadModule()

9. What are Angular Zones and how do they work?

Angular uses Zone.js to detect async events and trigger change detection. NgZone allows running code inside/outside zones for performance.

this.ngZone.runOutsideAngular(() => { heavyTask(); });

10. How do you customize Angular CLI with builders?

Extend CLI functionality using custom builders in angular.json. Useful for custom build steps, asset processing, etc.

"architect": { "build": { "builder": "@custom/builder:build" } }

11. How to create a plugin architecture in Angular?

Expose plugins as modules or dynamic components. Use Angular Module Federation or runtime loading with injector tokens.

loadComponent(pluginName) { this.vc.createComponent(pluginFactory); }

12. What's the best approach to bundle splitting in Angular?

Use lazy-loaded modules, differential loading, and loadChildren. Keep shared modules small to prevent duplication.

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

13. How do you handle API versioning in Angular frontend?

Create versioned services or URL-based handling. Use interceptors to add version headers or base path.

this.http.get(`/api/v2/users`);

14. What is the Angular Ivy compiler and its advantages?

Ivy is the default compiler from Angular 9. It enables smaller bundles, better debugging, faster builds, and better tree shaking.

platformBrowser().bootstrapModule(AppModule); // Ivy compatible

15. How do you create reusable libraries in Angular?

Use ng generate library to create packages within Angular workspace. Export components, services, and publish to npm.

ng generate library shared-ui

16. How do you handle long-running tasks in Angular apps?

Use Web Workers or async services to offload processing. Show loaders and use RxJS observeOn for UI-thread safety.

this.zone.runOutsideAngular(() => doHeavyTask());

17. How do you secure Angular applications in production?

Use Angular CLI production build, disable debug tools, use CSP headers, sanitize inputs, and prevent XSS with Angular security APIs.

ng build --prod --source-map=false

18. How does Angular manage concurrent HTTP requests?

Use forkJoin, combineLatest, or mergeMap for parallel or sequential requests with RxJS.

forkJoin([this.http.get(url1), this.http.get(url2)]).subscribe(...)

19. What are hybrid Angular apps (AngularJS + Angular)?

Apps using both AngularJS and Angular via UpgradeModule. Enables gradual migration from AngularJS to Angular.

@NgModule({ imports: [UpgradeModule] }) export class AppModule {}

20. How do you version control Angular features and modules?
Use feature branches, semantic versioning for libraries, and maintain changelogs. Use Git tags or GitFlow strategy.

git tag v1.2.0; git checkout -b feature/xyz

21. How do you track analytics and telemetry in Angular apps?
Use services like Google Analytics, Azure App Insights. Inject tracking in services, interceptors, or directives.

gtag('event', 'page_view', { page_path: this.router.url });

22. How can you integrate Angular with enterprise authentication providers (e.g., Azure AD)?
Use libraries like MSAL or Auth0 SDK, configure routes and tokens, and manage session lifecycle.

msalInstance.loginRedirect(); // Microsoft identity platform

23. How do you build a multi-tenant Angular application?
Dynamically load tenant configuration via APP_INITIALIZER, route users to tenant-specific views, and isolate data.

loadConfig(tenantId) => fetch(`/config/${tenantId}.json`)

24. How do you coordinate multiple Angular teams?
Split app into feature modules or micro frontends, enforce coding guidelines, use shared libs, and central versioning.

/feature1 /feature2 /libs /tools — separate team ownership

25. What is your CI/CD strategy for Angular deployment?
Use pipelines (GitHub Actions, Azure DevOps), automate lint/test/build, versioning, and deploy via Docker/CDN/Cloud.

ng build --prod && firebase deploy || docker build -t myapp .

15–20 Years Experience (Architect & Leadership Level)

1. How do you lead Angular development in large-scale distributed teams?
Use a modular monorepo, define contracts, enforce code reviews, follow CI/CD, and establish weekly sync-ups. Encourage code ownership and version control discipline.

/libs/shared /apps/admin /apps/customer // Managed by independent teams

2. What architectural patterns do you recommend for Angular?
Use clean architecture with core, shared, feature modules, facades for abstraction, and NgRx for state. Separate UI, logic, and services.

Facade => Component => UI Layer (Smart + Dumb components)

3. How do you balance SSR, CSR, and pre-rendering in Angular?

Use Angular Universal for SSR, SPA for user interaction, and Scully for static pre-rendering. Mix as per use case.

ng add @nguniversal/express-engine

4. What is your approach to domain-driven design with Angular?

Group features into bounded contexts, isolate domain logic in services/models, and use shared kernel only when needed.

/domain/user/user.module.ts /domain/inventory/inventory.module.ts

5. How do you enforce coding standards in enterprise Angular apps?

Use ESLint, Prettier, Husky for pre-commit hooks, and shared configs in monorepo. Enforce reviews and automated linting.

"lint": "eslint src/**/*.ts --fix"

6. How do you handle performance regression in Angular at scale?

Track bundle size, use Lighthouse and WebPageTest, automate tests via CI, and compare builds.

ng build --stats-json && webpack-bundle-analyzer dist/stats.json

7. How do you implement centralized logging in Angular?

Create a LoggingService, send logs to ELK or Azure Monitor via HTTP interceptor and capture global errors.

window.onerror = (msg, url, line) => this.logger.log(msg);

8. What's your strategy for maintaining multiple Angular versions across teams?

Use Nx monorepo with version constraints or split repositories with independent pipelines. Document upgrade paths.

projects: { teamA: v15, teamB: v14 }, libraries shared via npm

9. How do you design Angular modules for extensibility?

Expose configurable modules, use forRoot pattern, abstract via tokens, and use dynamic components or plugin loaders.

@NgModule({ providers: [{ provide: CONFIG_TOKEN, useValue: config }] })

10. What are your best practices for Angular library publishing?

Follow SemVer, build with ng-packagr, add peerDependencies, write README, and test in a consumer app.

ng build shared-lib && npm publish dist/libs/shared-lib

11. How do you handle accessibility (a11y) across all Angular components?

Use Angular CDK a11y utilities, ARIA roles, and run audits using axe or Lighthouse. Train devs and write a11y tests.

<mat-form-field aria-label="Email"> <input type="email"> </mat-form-field>

12. Describe your Angular app architecture for micro frontend integration.

Use Module Federation or Web Components. Each micro frontend owns routing and state isolated, shared via event bus.

bootstrap().then(() => defineCustomElement(MyComponent));

13. How do you assess tech debt in Angular projects?

Use code coverage reports, complexity analysis, static code tools, and periodic audits. Log known debts in backlog.

npm run lint && sonar-scanner -Dsonar.projectKey=myApp

14. How do you conduct Angular code reviews?

Use checklists (naming, modularity, test), CI tools for lint/build, peer reviews, and pull request templates.

- [ ] Tests Added - [ ] Follows Module Guidelines - [ ] Lint Passed

15. What's your approach to managing Angular release cycles?

Follow semantic versioning, freeze before release, use beta/staging environments, and tag releases in Git.

git tag -a v1.3.0 -m "Feature release"; git push origin --tags

16. How do you maintain security in enterprise Angular apps?

Enforce HTTPS, sanitize inputs, use CSP, JWT expiration checks, and avoid client secrets. Add SAST tools.

DomSanitizer.bypassSecurityTrustHtml(safeHtml)

17. How do you integrate Angular with GraphQL and Apollo?

Use Apollo Angular Client, define schemas, queries, and manage cache/state using Apollo Link.

apollo.watchQuery({ query: gql`{ books { title } }` }).valueChanges.subscribe(...)

18. How do you optimize Angular for low bandwidth or offline usage?

Use Angular PWA, lazy loading, compression, caching via service workers and indexedDB for offline state.

ng add @angular/pwa

19. How do you evaluate third-party Angular component libraries?

Check community support, a11y compliance, last release date, license, tree-shakability, and bundle size impact.

npm run build && source-map-explorer dist/main.js

20. How do you structure your Angular teams for productivity?

Organize by feature or vertical slice, assign tech leads per module, use shared libraries and weekly demos.

Team A: /user, Team B: /dashboard, Shared: /ui-lib

21. How do you train junior Angular developers?

Use pair programming, mentorship, code katas, internal wikis, and assign small modules with increasing complexity.

Week 1: Components | Week 2: Services | Week 3: Forms | Week 4: RxJS

22. How do you apply S.O.L.I.D. principles in Angular architecture?

Split components/services by responsibility, use DI for open/closed, interfaces for substitution, facades for isolation.

interface Logger { log(msg: string): void } // DIP via InjectionToken

23. How do you handle legacy AngularJS to Angular migrations?

Use UpgradeModule to bootstrap hybrid app, migrate components gradually, test at each phase.

platformBrowserDynamic().bootstrapModule(AppModule).then(() => upgrade.bootstrap(document.body, ['legacyApp']));

24. How do you ensure quality and test coverage in Angular?

Use Karma/Jasmine, add e2e with Cypress, enforce thresholds via angular.json, and integrate into CI pipelines.

"codeCoverage": true, "thresholds": { "statements": 80 }

25. What is your DevOps strategy for Angular with Docker and Kubernetes?

Use multi-stage Docker builds, deploy static files to NGINX, use Helm for K8s config, and auto-scale via HPA.

FROM node:18 AS build RUN ng build --prod FROM nginx:alpine COPY --from=build /dist/app /usr/share/nginx/html

Sai Reddy
saireddy-dotnetfs