Shailesh Diwanji

Part 2

30 Advanced Angular Interview Questions

Part 2 - For Experienced Candidates

Deep Dive into Advanced Concepts & Real-World Scenarios

31. What is the difference between forRoot() and forChild() in Angular modules?

forRoot():

- Called once in the root AppModule
- Provides services at application level (singleton)
- Configures providers for the entire application
- Example: RouterModule.forRoot(routes)

forChild():

- Called in feature/lazy-loaded modules
- Provides directives, components, pipes without services
- Doesn't register services (prevents multiple instances)
- Example: RouterModule.forChild(routes)

This pattern ensures services remain singleton while allowing module reuse across features.

32. Explain Angular's Ivy renderer and its advantages over View Engine.

Ivy is Angular's next-generation compilation and rendering pipeline (default since Angular 9).

Advantages:

- **Smaller bundle sizes:** Tree-shaking removes unused code more effectively
- **Faster compilation:** Incremental compilation and better build times
- **Better debugging:** More readable generated code
- **Improved type checking:** Stricter template type checking
- **Locality principle:** Components compile independently
- **Dynamic component loading:** No need for ComponentFactoryResolver
- **Better testing:** TestBed improvements

33. What are Angular Schematics and how do you create custom schematics?

Schematics are code generators that create, modify, or refactor Angular code through templates and rules.

Use cases:

- Generate components with company-specific structure
- Add configurations to existing projects
- Perform migrations and updates
- Enforce coding standards

Creating custom schematics:

- Use @angular-devkit/schematics package
- Define schema.json for options
- Implement Rule functions with Tree manipulation
- Test using SchematicTestRunner

34. Explain the difference between @Self, @SkipSelf, @Optional, and @Host decorators.

These decorators modify dependency injection behavior:

- **@Self:** Only looks in the current component's injector, throws error if not found
- **@SkipSelf:** Skips current component's injector, searches parent hierarchy
- **@Optional:** Returns null if dependency not found instead of throwing error
- **@Host:** Looks up to the host component only, used with content projection

These can be combined: @Optional() @SkipSelf() for flexible injection patterns.

35. What is Angular's ExpressionChangedAfterItHasBeenCheckedError and how do you fix it?

This error occurs when a value changes during change detection in development mode (which runs CD twice).

Common causes:

- Changing parent state from child component during initialization
- Modifying data in lifecycle hooks that affect parent
- Using getters that return different values on each call

Solutions:

- Use setTimeout to defer changes to next tick
- Move logic to appropriate lifecycle hook (ngAfterViewInit)
- Use ChangeDetectorRef.detectChanges() manually
- Restructure data flow to avoid circular dependencies
- Use async operations properly

36. Explain Angular's ViewEncapsulation modes.

ViewEncapsulation determines how component styles are scoped:

- **Emulated (default):** Scopes CSS using attribute selectors ([_ngcontent-xxx]). Styles don't leak but can be overridden from outside.
- **None:** No encapsulation, styles are global. Can affect other components.
- **ShadowDom:** Uses native Shadow DOM. True encapsulation, styles completely isolated. Limited browser support considerations.

Choose based on style isolation needs and browser support requirements.

37. What are Angular Elements and when would you use them?

Angular Elements packages Angular components as custom elements (Web Components) that work in any HTML environment.

Use cases:

- Integrate Angular components in non-Angular apps
- Create reusable widget libraries
- Micro-frontends architecture
- CMS integration (WordPress, Drupal)
- Legacy application migration

Uses @angular/elements package with createCustomElement() to convert components to custom elements.

38. Explain the difference between Observables and Promises.

Promises:

- Single value, resolves once
- Eager execution (starts immediately)
- Not cancellable
- No operators for transformation

Observables:

- Stream of values over time
- Lazy execution (starts on subscription)
- Cancellable (unsubscribe)
- Rich operator library (map, filter, debounce, etc.)
- Can emit multiple values
- Better for complex async operations

39. What is the purpose of APP_INITIALIZER token?

APP_INITIALIZER is a multi-provider token that executes functions during application bootstrap, before the app initializes.

Use cases:

- Load configuration from server before app starts
- Initialize authentication state
- Load essential data (translations, feature flags)
- Set up third-party libraries

Returns Promise or Observable. App won't initialize until all initializers complete. Critical for ensuring dependencies are ready before app renders.

40. Explain Angular's inject() function and its advantages.

inject() (Angular 14+) is a function-based alternative to constructor injection.

Advantages:

- Can be used in functions, not just constructors
- Works in factory functions and class field initializers
- Enables functional composition patterns
- Cleaner code for multiple injections
- Better tree-shaking potential

Example: `private authService = inject(AuthService);`

Can only be called in injection context (constructor, field initializer, factory function).

41. What are Higher-Order Observables and how do you handle them?

Higher-Order Observables are Observables that emit other Observables (Observable of Observables).

Flattening operators:

- **mergeMap:** Subscribes to all inner observables concurrently
- **switchMap:** Cancels previous, subscribes to latest only
- **concatMap:** Queues inner observables, subscribes sequentially
- **exhaustMap:** Ignores new inner observables while one is active

Common scenario: HTTP requests triggered by user input. Choose operator based on desired behavior (cancel, queue, or concurrent).

42. Explain Angular's EnvironmentInjector and its use cases.

EnvironmentInjector (Angular 14+) is a standalone injector for creating and managing components outside the main component tree.

Use cases:

- Dynamic component creation with proper DI context
- Testing isolated components
- Micro-frontend scenarios
- Plugin architectures

Works with createComponent() to provide injection context for dynamically created components. Replaces older ComponentFactoryResolver pattern.

43. What is the difference between [class.x], [ngClass], and [className]?

- **[class.active]="condition":** Toggles single class based on boolean. Most performant for single class.
- **[ngClass]:** Adds/removes multiple classes. Accepts object, array, or string. More flexible but slight overhead.
- **[className]="value":** Sets entire class attribute, replacing all classes. Rarely used, can cause issues with Angular's class management.

Best practice: Use [class.x] for single toggles, [ngClass] for multiple conditional classes.

44. Explain the role of platformBrowserDynamic() and platformBrowser().

- **platformBrowserDynamic():** Used with JIT compilation. Includes the compiler, bootstraps application in browser with runtime compilation.
- **platformBrowser():** Used with AOT compilation. Smaller bundle, no compiler included, faster bootstrap.

Modern applications use platformBrowser() with AOT for production. platformBrowserDynamic() mainly for development with JIT.

45. What are Angular's build optimization techniques?

Build-time optimizations:

- **AOT compilation:** Pre-compile templates
- **Tree-shaking:** Remove unused code
- **Minification:** Reduce file sizes
- **Code splitting:** Break into smaller chunks
- **Lazy loading:** Load modules on-demand
- **Differential loading:** Serve modern/legacy bundles
- **Build optimizer:** Angular-specific optimizations
- **Source maps:** Disable for production
- **Budget limits:** Set bundle size thresholds

Configure in angular.json under production configuration.

46. Explain Angular's ModuleWithProviders pattern.

ModuleWithProviders is a type that includes both a module and its associated providers, typically returned by static methods like forRoot().

Structure:

```
ModuleWithProviders<MyModule> { ngModule: MyModule, providers: [...] }
```

Purpose:

- Separates module configuration from instantiation
- Allows passing configuration to modules
- Enables singleton services in forRoot()
- Provides flexibility for module consumers

Essential pattern for creating reusable library modules.

47. What is the difference between stateful and stateless components?

Stateful (Smart/Container) components:

- Manage application state
- Interact with services
- Handle business logic
- Pass data to child components
- Aware of the application context

Stateless (Dumb/Presentational) components:

- Receive data via @Input()
- Emit events via @Output()
- No service dependencies
- Highly reusable
- Easier to test
- Use OnPush change detection

Best practice: Maximize stateless components for better maintainability.

48. Explain Angular's DestroyRef and how it simplifies cleanup.

DestroyRef (Angular 16+) provides a way to register cleanup callbacks without implementing ngOnDestroy.

Advantages:

- Cleaner syntax than ngOnDestroy
- Works in functions, not just class methods
- Can be injected anywhere
- Multiple callbacks supported
- Enables functional cleanup patterns

Example: `destroyRef.onDestroy(() => subscription.unsubscribe());`

Particularly useful with inject() function and composition patterns.

49. What are Angular's async validators and how do they differ from sync validators?

Synchronous validators:

- Return validation errors immediately
- Used for simple validation rules
- Example: required, minLength, pattern

Asynchronous validators:

- Return Promise or Observable
- Used for server-side validation (username availability, email uniqueness)
- Run after sync validators pass
- Should include debouncing to avoid excessive requests
- Form shows pending state during validation

Implement AsyncValidator interface, return null for valid or error object for invalid.

50. Explain the difference between cold and hot observables.

Cold Observables:

- Start producing values when subscribed
- Each subscriber gets independent execution
- Examples: HTTP requests, interval, range
- Data producer created inside observable

Hot Observables:

- Produce values regardless of subscriptions
- All subscribers share same execution
- Examples: DOM events, Subjects, WebSockets
- Data producer outside observable

Convert cold to hot using share(), shareReplay(), or publish() operators.

51. What is Angular's TransferState and when is it used?

TransferState transfers data from server-side rendered application to client, avoiding duplicate HTTP requests.

Use case:

- With Angular Universal (SSR)
- Server fetches data and renders HTML
- Data is serialized and transferred to client
- Client reuses data instead of re-fetching

Benefits:

- Faster initial render
- Reduced server load
- Better user experience
- Prevents flickering during hydration

52. Explain Angular's multi-provider tokens.

Multi-provider tokens allow multiple values to be associated with a single injection token, returned as an array.

Configuration: `{ provide: TOKEN, useValue: value, multi: true }`

Use cases:

- HTTP interceptors (multiple interceptors)
- APP_INITIALIZER (multiple initialization functions)
- Validators (combining multiple validation rules)
- Plugin systems
- Event listeners/handlers

All registered providers are collected into an array for injection.

53. What are Angular's compilation contexts and template syntax?

Template expressions execute in component context with specific rules:

Restrictions:

- No assignments (except in event bindings)
- No new keyword
- No chaining expressions with ; or ,
- No increment/decrement operators
- No bitwise operators

Template reference variables: Use # or ref- to create local variables. Scope limited to template.

Template expressions: Should be simple, side-effect free, and idempotent for predictable change detection.

54. Explain Angular's attribute directives vs structural directives.

Attribute Directives:

- Change appearance or behavior of elements
- Examples: ngClass, ngStyle, custom directives
- Don't modify DOM structure
- Can be multiple on same element

Structural Directives:

- Add or remove DOM elements
- Examples: *ngIf, *ngFor, *ngSwitch
- Use * prefix (syntactic sugar for ng-template)
- Only one structural directive per element
- Work with TemplateRef and ViewContainerRef

55. What is Angular's router state and how do you access it?

Router state represents the current state of the router including activated routes, parameters, and data.

Access methods:

- **ActivatedRoute:** Access current route params, query params, data, fragments
- **Router.events:** Observable stream of navigation events
- **RouterState:** Tree of activated routes
- **RouterStateSnapshot:** Snapshot of router state at specific moment

Common use cases: Reading route parameters, accessing route data, responding to navigation events, implementing breadcrumbs.

56. Explain Angular's compilation modes: Full and Partial.

Full Compilation:

- Used for applications
- Generates complete compiled code
- All dependencies must be present

Partial Compilation (Ivy):

- Used for libraries
- Generates partially compiled code
- Final compilation done by consuming application
- Better compatibility across Angular versions
- Smaller library bundles

Configured in tsconfig with compilationMode: "partial" for libraries.

57. What are the different types of route matching strategies?

pathMatch strategies:

- **'prefix' (default):** Matches if URL starts with path. Used for most routes.
- **'full':** Matches only if URL exactly equals path. Essential for empty path routes and redirects.

Example issue: Redirect with path: '' and pathMatch: 'prefix' would match all routes, causing infinite redirects.

Best practice: Always use pathMatch: 'full' for empty path routes and redirects.

58. Explain Angular's TestBed and its role in testing.

TestBed is Angular's primary testing utility that creates and configures a testing module.

Key methods:

- **configureTestingModule():** Configure testing module with components, providers
- **createComponent():** Create component instance for testing
- **inject():** Get service instances from testing injector
- **overrideComponent/Provider:** Replace dependencies for testing

Features: Provides real Angular environment, handles DI, supports async testing, enables component interaction testing.

59. What is the difference between fakeAsync, async, and done() in testing?

fakeAsync:

- Synchronous testing of async code
- Use tick() to advance time
- Use flush() to exhaust all async tasks
- Doesn't work with real XHR requests

async:

- Waits for all async operations
- Use fixture.whenStable() to wait
- Works with real async operations

done():

- Manual control over test completion
- Call done() when test is finished
- More verbose but flexible

60. Explain Angular's differential loading and browser support strategy.

Differential loading serves different bundles to different browsers based on their capabilities.

How it works:

- Builds two sets of bundles: ES2015+ and ES5
- Modern bundles: smaller, faster (for modern browsers)
- Legacy bundles: larger, polyfilled (for old browsers)
- Browser automatically loads appropriate version

Configuration: Set browserslist in package.json

Benefits: Smaller bundles for modern browsers, backward compatibility, automatic based on user agent.