Shailesh Diwanji

PART 3

30 Expert Angular Interview Questions

Part 3 - Advanced Concepts & Architecture

Real-World Scenarios & Performance Optimization

61. How do you implement custom form control with ControlValueAccessor?

A complete implementation requires:

- **Implement ControlValueAccessor interface** with four methods:
  - writeValue(value: any): Set value from model to view
  - registerOnChange(fn): Store callback for value changes
  - registerOnTouched(fn): Store callback for touch events
  - setDisabledState(disabled): Handle disabled state
- **Provide NG_VALUE_ACCESSOR token** in component providers
- **Call onChange()** when internal value changes
- **Call onTouched()** when component is touched/blurred

This allows your custom component to work seamlessly with Angular forms (both template-driven and reactive).

62. Explain Angular's compilation process from code to browser.

With AOT (Production):

- TypeScript compilation to JavaScript
- Template compilation to TypeScript
- Angular compiler generates component factories
- Tree-shaking removes unused code
- Bundling and minification
- Browser loads and executes compiled code

With JIT (Development):

- TypeScript compilation to JavaScript
- Browser loads Angular compiler
- Templates compiled at runtime in browser
- Components instantiated dynamically

AOT moves compilation from runtime to build time for better performance.

63. What are the different strategies for state management in Angular?

Options:

- **Component State:** @Input/@Output, simple for small apps
- **Service with BehaviorSubject:** Shared state via services, good for medium apps
- **NgRx:** Redux pattern with actions, reducers, effects. Best for large, complex apps with predictable state
- **Akita:** Simpler than NgRx, entity-based state management
- **NGXS:** Redux alternative with less boilerplate
- **Signals:** Built-in reactive state (Angular 16+)
- **RxAngular:** Reactive state management with RxJS

Choose based on app complexity, team expertise, and scalability needs.

64. Explain the concept of Facade pattern in Angular services.

Facade pattern provides a simplified interface to complex subsystems.

In Angular:

- Create a facade service that coordinates multiple services
- Components interact only with facade, not individual services
- Hides complexity of state management, API calls, business logic
- Provides clear, focused API for components

Benefits:

- Decouples components from complex logic
- Easier testing (mock single facade)
- Better maintainability
- Clear separation of concerns

65. How do you handle error globally in Angular applications?

Multiple approaches:

- **ErrorHandler:** Implement custom ErrorHandler to catch all errors globally. Override handleError() method.
- **HTTP Interceptor:** Catch HTTP errors centrally, handle 401, 403, 500 errors uniformly.
- **Router Error Events:** Subscribe to router events to catch navigation errors.
- **RxJS catchError operator:** Handle observable errors in streams.

Best practice: Combine approaches - use ErrorHandler for unhandled errors, interceptors for HTTP, and specific error handling where needed. Log errors to monitoring service (Sentry, LogRocket).

66. What is the difference between QueryList and ViewChildren?

ViewChildren: Decorator that queries elements and returns a QueryList

QueryList: The collection type returned by ViewChildren/ContentChildren

QueryList features:

- Live collection that updates when elements change
- Provides Observable via changes property
- Array-like with methods: map, filter, forEach, toArray
- Properties: first, last, length

You use ViewChildren to get a QueryList, which you then iterate or observe for changes.

67. Explain Angular's hydration process in SSR applications.

Hydration is the process of attaching Angular application to server-rendered HTML.

Process:

- Server renders HTML with data
- Browser receives and displays static HTML (fast FCP)
- Angular bootstraps and "hydrates" the HTML
- Event listeners attached, components become interactive
- Angular preserves existing DOM instead of re-creating it

Angular 16+ non-destructive hydration: Reuses server DOM, prevents flickering, improves performance. Enable with provideClientHydration().

68. How do you implement virtual scrolling and why is it important?

Virtual scrolling (via CDK ScrollingModule) renders only visible items in viewport.

Implementation:

- Import ScrollingModule from @angular/cdk/scrolling
- Use cdk-virtual-scroll-viewport component
- Specify itemSize for fixed height items
- Use *cdkVirtualFor instead of *ngFor

Why important:

- Dramatically improves performance for large lists
- Reduces DOM nodes (only renders ~20-30 items)
- Constant performance regardless of list size
- Smooth scrolling experience

69. What are Angular Material's theming capabilities?

Angular Material uses Sass-based theming system.

Key concepts:

- **Palettes:** Define primary, accent, warn colors
- **Themes:** Light or dark theme with color palettes
- **Typography:** Define font families, sizes, weights
- **Density:** Compact, default, or comfortable spacing

Implementation:

- Define theme with mat.define-light-theme()
- Include component themes with @include
- Create multiple themes for different sections
- Use CSS variables for dynamic theming

70. Explain the concept of projection slots in ng-content.

Multiple projection slots allow different content in different locations.

Implementation:

- **Named slots:** <ng-content select=".header"></ng-content>
- **Element selector:** <ng-content select="header"></ng-content>
- **Attribute selector:** <ng-content select="[slot=footer]"></ng-content>
- **Default slot:** <ng-content></ng-content> (catches unprojected content)

Enables flexible, reusable component layouts (cards, dialogs, layouts).

71. How do you implement authentication with JWT in Angular?

Complete flow:

- **Login:** Send credentials, receive JWT token
- **Store token:** localStorage/sessionStorage (or memory for security)
- **HTTP Interceptor:** Attach token to all requests in Authorization header
- **AuthGuard:** Protect routes, check token validity
- **Token refresh:** Implement refresh token mechanism
- **Handle 401:** Redirect to login on unauthorized
- **Logout:** Clear token, navigate to login

Security considerations: Store tokens securely, implement token expiration, use HTTPS, validate tokens server-side.

72. What is the purpose of InjectionToken and when do you use it?

InjectionToken creates unique tokens for dependency injection without classes.

Use cases:

- Inject primitive values (strings, numbers, objects)
- Inject configuration objects
- Avoid naming collisions
- Type-safe injection of non-class dependencies

Example: `export const API_URL = new InjectionToken<string>('api.url');`

Provides type safety and prevents accidental string-based injection errors. Better than string tokens used in AngularJS.

73. Explain the difference between @Attribute and @Input decorators.

@Input():

- Binds to component property
- Dynamic, reactive to changes
- Triggers change detection
- Works with any expression
- Higher performance cost

@Attribute():

- Reads HTML attribute once at initialization
- Static, no change detection
- String values only
- Better performance for static data
- Used in constructor injection

Use @Attribute for static, unchanging values to improve performance.

74. How do you implement caching strategies for HTTP requests?

Strategies:

- **In-memory cache:** Store responses in service Map/object, check before making request
- **Time-based expiration:** Cache with timestamp, invalidate after timeout
- **shareReplay operator:** Cache observable results, share among subscribers
- **HTTP Interceptor:** Implement caching logic centrally
- **Service Worker:** Cache at network level

Considerations: Cache invalidation strategy, memory limits, cache key generation, handling errors.

75. What is Angular's preloading strategy and how do you customize it?

Preloading loads lazy modules in background after initial load.

Built-in strategies:

- **NoPreloading:** Default, no preloading
- **PreloadAllModules:** Preload all lazy modules immediately

Custom strategy:

- Implement PreloadingStrategy interface
- Override preload() method
- Use route data to conditionally preload
- Register in RouterModule.forRoot()

Example: Preload based on user role, network speed, or route priority.

76. Explain Angular's debug utilities and debugging techniques.

Debug utilities:

- **ng.probe():** Get component instance from element
- **ng.getComponent():** Get component instance (Ivy)
- **ng.applyChanges():** Trigger change detection manually
- **Angular DevTools:** Browser extension for profiling, change detection visualization

Techniques:

- Enable source maps for debugging TypeScript
- Use Angular CLI with --source-map flag
- Augury/Angular DevTools for component inspection
- Performance profiling with Chrome DevTools

77. How do you implement file upload with progress tracking?

Implementation:

- Use HttpClient with reportProgress: true option
- Create FormData with file
- Subscribe to upload events (HttpEventType)
- Track upload progress percentage
- Handle complete, error events

Events to handle:

- HttpEventType.UploadProgress: Calculate percentage
- HttpEventType.Response: Upload complete
- Errors: Show error message, allow retry

Can implement pause/resume with proper backend support using Range headers.

78. What is the difference between BrowserModule and CommonModule?

BrowserModule:

- Import once in root AppModule only
- Re-exports CommonModule
- Provides browser-specific services (Title, Meta, etc.)
- Required for browser applications
- Includes app initialization logic

CommonModule:

- Import in feature modules
- Provides common directives (ngIf, ngFor, pipes)
- Platform-agnostic
- Doesn't provide services

Never import BrowserModule in feature modules, use CommonModule instead.

79. Explain the concept of Micro-frontends with Angular.

Micro-frontends split large applications into smaller, independently deployable apps.

Implementation approaches:

- **Module Federation:** Webpack 5 feature for runtime module sharing
- **Angular Elements:** Package components as Web Components
- **Iframe-based:** Simple but limited communication
- **Single-SPA:** Framework for micro-frontend orchestration

Benefits: Independent deployment, team autonomy, technology diversity, scalability.

Challenges: Shared dependencies, styling conflicts, increased complexity, routing coordination.

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

Built-in i18n:

- Mark strings with i18n attribute in templates
- Extract messages: ng extract-i18n
- Translate XLIFF files for each locale
- Build separate bundles per locale
- Deploy locale-specific apps

Runtime i18n (ngx-translate):

- Single build, runtime language switching
- JSON translation files
- Use translate pipe or service
- Dynamic language loading

Choose based on needs: compile-time for performance, runtime for flexibility.

81. What are Angular's animation callbacks and states?

Animation callbacks:

- **(@trigger.start):** Fired when animation starts
- **(@trigger.done):** Fired when animation completes
- Both provide AnimationEvent with details

States:

- Define named states with state()
- Transition between states with transition()
- Special states: void (element not in DOM), * (wildcard)
- :enter (void => *), :leave (* => void) aliases

Use callbacks to coordinate animations or trigger side effects.

82. Explain the concept of component composition vs inheritance.

Composition (Preferred):

- Build components from smaller, reusable pieces
- Use @Input/@Output for communication
- Content projection for flexibility
- Services for shared logic
- More flexible, easier to test

Inheritance:

- Extend base component class
- Share lifecycle hooks, methods
- Can lead to tight coupling
- Harder to test and maintain

Best practice: Favor composition. Use inheritance sparingly for true "is-a" relationships.

83. How do you implement retry logic for failed HTTP requests?

RxJS operators:

- **retry(count):** Retry specified number of times immediately
- **retryWhen():** Custom retry logic with delay
- **Exponential backoff:** Increase delay between retries

Implementation pattern:

- Use retryWhen with delayWhen for timed retries
- Combine with take/takeWhile to limit attempts
- Add conditional logic (retry only on specific errors)
- Implement in HTTP interceptor for global retry

Consider user experience: show retry count, allow manual retry, handle eventual failure gracefully.

84. What is Angular's CDK (Component Dev Kit) and its key modules?

CDK provides behavior primitives for building UI components without styling.

Key modules:

- **Accessibility:** Focus management, ARIA support, keyboard navigation
- **Layout:** Breakpoint observer, responsive layouts
- **Overlay:** Floating panels (tooltips, dialogs, menus)
- **Portal:** Dynamic content rendering
- **Scrolling:** Virtual scrolling, scroll monitoring
- **Drag & Drop:** Draggable elements, drop zones
- **Table:** Data table primitives
- **Stepper:** Step-by-step workflow

Build custom components with Material-like behavior without Material styling.

85. Explain Angular's router auxiliary routes and named outlets.

Auxiliary routes allow multiple router outlets in the same component.

Implementation:

- Define named outlets: <router-outlet name="sidebar"></router-outlet>
- Configure routes with outlet property
- Navigate: router.navigate([{ outlets: { sidebar: ['path'] } }])
- Multiple independent routes active simultaneously

Use cases:

- Sidebar navigation independent of main content
- Modal dialogs with routing
- Multi-panel layouts

Each outlet has its own navigation stack and history.

86. How do you implement optimistic UI updates in Angular?

Optimistic updates show changes immediately before server confirmation.

Pattern:

- Update local state/UI immediately
- Send request to server
- On success: Keep changes (maybe update with server data)
- On error: Revert changes, show error message

Implementation:

- Store previous state before update
- Use RxJS operators (tap for side effects)
- catchError to revert on failure
- Consider using state management (NgRx) for complex scenarios

Improves perceived performance and user experience.

87. What are the security best practices in Angular applications?

Built-in protections:

- **XSS prevention:** Automatic sanitization of untrusted values
- **CSRF protection:** Use XSRF-TOKEN with HttpClient
- **Content Security Policy:** Restrict resource loading

Best practices:

- Never use bypassSecurityTrust* unless absolutely necessary
- Validate and sanitize user input
- Use HTTPS everywhere
- Implement proper authentication/authorization
- Keep Angular and dependencies updated
- Use Angular's built-in sanitization (DomSanitizer)
- Avoid eval() and Function constructor
- Implement proper CORS policies

88. Explain Angular's platform-server and server-side APIs.

platform-server enables Angular applications to run on server (Node.js) for SSR.

Key APIs:

- **platformServer:** Server platform for bootstrapping
- **renderModule:** Renders module to HTML string
- **ServerModule:** Provides server-specific implementations
- **SERVER_TRANSITION_ID:**

88. Explain Angular's platform-server and server-side APIs.

platform-server enables Angular applications to run on server (Node.js) for SSR.

Key APIs:

- **platformServer:** Server platform for bootstrapping
- **renderModule:** Renders module to HTML string
- **ServerModule:** Provides server-specific implementations
- **SERVER_TRANSITION_ID:** Coordinates transition from server to client

Server-specific considerations:

- No DOM APIs (window, document unavailable)
- Use PLATFORM_ID to detect environment
- Handle absolute URLs for HTTP requests
- Use TransferState for data sharing

89. How do you implement pagination in Angular applications?

Approaches:

- **Server-side pagination:** Request specific page from API (page number, size). Better for large datasets.
- **Client-side pagination:** Load all data, paginate locally using slice pipe or logic.

Implementation considerations:

- Track current page, total pages, page size
- Calculate offset: (page - 1) * pageSize
- Use Angular Material Paginator or custom component
- Update route params for bookmarkable pages
- Handle loading states and errors
- Consider infinite scroll for mobile

90. Explain the difference between declarative and imperative approaches in Angular.

Declarative (Reactive):

- Define what should happen, not how
- Use RxJS observables and operators
- Async pipe in templates
- Data streams that automatically update
- Less imperative code, more reactive
- Example: data$ = this.service.getData();

Imperative:

- Explicitly control flow and state
- Subscribe and manually assign to properties
- More control but more boilerplate
- Must handle subscriptions/unsubscriptions

Best practice: Prefer declarative approach with async pipe for cleaner, more maintainable code.