Beginner Angular Interview Questions:

1. **What is Angular, and how is it different from AngularJS?**
   - Discuss the differences between AngularJS (Angular 1.x) and Angular 2+ versions, such as the use of TypeScript, modularity, etc.
2. **Explain the structure of an Angular application.**
   - Components, Modules, Services, Directives, etc.
3. **What are components in Angular? How do you create them?**
   - Explanation of Angular components, their lifecycle, and how to create them using `ng generate component`.
4. **What is data binding in Angular? Explain its different types.**
   - Two-way binding, property binding, event binding, and interpolation.
5. **What are Angular directives? Explain the types of directives available in Angular.**
   - Structural (e.g., `*ngIf`, `*ngFor`) and Attribute Directives (e.g., `ngClass`, `ngStyle`).
6. **What is Angular Router? How do you configure routing in Angular?**
   - Routing setup with `RouterModule`, `routes`, and the role of `router-outlet`.
7. **What are Angular services, and how do you create and inject a service?**
   - Explanation of services, dependency injection, and using `@Injectable()`.
8. **What are pipes in Angular? How are custom pipes created?**
   - Built-in pipes like `DatePipe`, `CurrencyPipe`, `AsyncPipe`, and creating custom pipes using the `Pipe` decorator.
9. **What are observables in Angular? How are they different from promises?**
   - Use of observables with `RxJS`, differences between observables and promises.
10. **What is lazy loading in Angular? How do you implement lazy loading of modules?**
    - Concepts of lazy loading modules and its benefits for performance using Angular's `loadChildren` feature.
11. **What is Angular's `ngOnInit` lifecycle hook?**
    - Explanation of lifecycle hooks, particularly `ngOnInit`, and their significance in component initialization.
12. **What is the difference between `ViewChild` and `ContentChild`?**
    - Explanation of `ViewChild` and `ContentChild` decorators, and when to use each.
13. **Explain the purpose of Angular forms. What are the differences between template-driven and reactive forms?**
    - Comparison between template-driven forms and reactive forms and their use cases.
14. **How does Angular handle error handling?**
    - Using `ErrorHandler` class, custom error handling with services, `HttpClient` error handling.
15. **What is Angular's change detection mechanism? How does it work?**

- Concept of Zones and Change Detection Strategy.

Advanced Angular Interview Questions:

1. **What is Angular Ivy? How does it improve performance and bundle size in Angular?**
   - Discuss Angular Ivy's rendering engine, AOT (Ahead-of-Time) compilation, and how it impacts bundle sizes and performance.
2. **What are Angular modules (`NgModule`)? How do you organize an Angular app using feature modules and shared modules?**
   - Structuring large Angular apps with core modules, shared modules, and lazy-loaded feature modules.
3. **How do you implement Angular `NgRx` for state management? Explain its architecture.**
   - Explanation of Redux-like architecture, actions, reducers, effects, and selectors in `NgRx`.
4. **What are dynamic components in Angular? How do you create and load dynamic components?**
   - Use of `ComponentFactoryResolver` and `ViewContainerRef` for creating dynamic components.
5. **Explain Angular's Dependency Injection (DI) mechanism in depth.**
   - Providers, multi-providers, injection tokens, hierarchical injectors, and use of `@Inject()` decorator.
6. **How do you optimize the performance of an Angular application?**
   - Lazy loading, preloading strategies, AOT compilation, OnPush change detection, using `trackBy` in `ngFor`, efficient RxJS operators, tree-shaking, and third-party library optimizations.
7. **What are Angular interceptors? How do you use them to modify HTTP requests or responses?**
   - Creating interceptors using `HttpInterceptor` for logging, authentication, or modifying HTTP requests.
8. **What is a resolver in Angular routing? How do you use it to prefetch data?**
   - Implementing a resolver to resolve data before routing completes using `Resolve` interface.
9. **Explain how Angular's `ViewEncapsulation` works. What are the different types?**
   - Default, Emulated, Shadow DOM, and None. How each affects CSS scoping.
10. **How do you handle large datasets in Angular?**
    - Implementing pagination, infinite scrolling, virtual scrolling using Angular CDK, and performance considerations.
11. **What is the `RouterModule.forRoot()` vs `RouterModule.forChild()` in Angular?**
    - The difference between the two and when to use them.
12. **What is differential loading in Angular, and how does it benefit modern browsers?**

    - Explanation of how Angular builds modern and legacy bundles for performance optimization.

13. **What is Angular Universal? How do you use it for server-side rendering (SSR)?**
    - Explanation of server-side rendering using Angular Universal and its benefits for SEO and performance.
14. **How does Angular handle cross-component communication?**
    - Techniques like input/output properties, shared services, `@ViewChild`, event emitters, and state management tools like `NgRx` or `BehaviorSubject`.
15. **Explain the usage of `ng-template`, `ng-container`, and `ng-content`.**
    - Discuss the differences between these structural directives and their use cases in templates and content projection.
16. **How do you manage memory leaks in Angular applications?**
    - Identifying and preventing memory leaks using `unsubscribe` for observables, `takeUntil`, `AsyncPipe`, and `ngOnDestroy()`.
17. **Explain the role of AOT (Ahead-of-Time) compilation in Angular. How does it affect performance?**
    - How AOT works and its impact on performance, bundle size, and catching template errors early.
18. **What is a custom structural directive in Angular? How do you create one?**
    - Creating a custom structural directive using the `Directive` decorator and manipulating the DOM with `TemplateRef` and `ViewContainerRef`.
19. **How do you handle routing guards in Angular?**
    - Explanation of `CanActivate`, `CanDeactivate`, `Resolve`, and `CanLoad` guards and their practical usage in securing routes.
20. **What are zone.js and its significance in Angular?**
    - Explaining how `zone.js` helps with Angular's change detection and common use cases for running tasks outside the Angular zone for performance benefits.
21. **How do you handle complex forms in Angular?**
    - Managing complex forms with `FormArray`, custom validators, asynchronous validators, dynamic form controls, and form grouping using Angular's Reactive Forms.
22. **What are custom validators in Angular, and how do you implement them?**
    - Creating both synchronous and asynchronous custom validators using `ValidatorFn` and `AsyncValidatorFn`.
23. **How does Angular handle routing animations?**
    - Implementing route-based animations using Angular's `@angular/animations` module, defining animations in routing transitions, and optimizing performance.
24. **Explain the `ControlValueAccessor` interface in Angular. How do you use it to create custom form controls?**
    - Detailed discussion on how to implement a custom form control that integrates seamlessly with Angular's forms API.
25. **What is `NgUpgrade`? How do you migrate an AngularJS application to Angular?**

- Explanation of hybrid applications and how `NgUpgrade` helps in progressively upgrading an AngularJS app to Angular without a full rewrite.

26. **What are Angular preloading strategies, and how do you implement them?**
    - Discussion of different preloading strategies (`NoPreloading`, `PreloadAllModules`, or custom strategies) and their use in optimizing lazy-loaded modules.
27. **What is `Renderer2` in Angular, and how does it differ from direct DOM manipulation?**
    - Explanation of how `Renderer2` is used to safely manipulate DOM elements in a platform-agnostic way, avoiding direct DOM manipulation.
28. **Explain the difference between `OnPush` and `Default` change detection strategies.**
    - Discussion on how `OnPush` change detection can improve performance and when to use it over the default strategy.
29. **What is `ElementRef` in Angular, and why should you avoid using it directly?**
    - Discussion on the risks of directly manipulating DOM elements with `ElementRef` and why it's recommended to use `Renderer2` instead for better platform support.
30. **Explain how the `AsyncPipe` works. What are its advantages?**
    - Using the `AsyncPipe` to automatically subscribe to observables and promises, handle unsubscription, and clean up efficiently.
31. **What is a service worker in Angular? How do you implement it for PWA support?**
    - Explanation of Progressive Web Apps (PWA) in Angular and how to use Angular's built-in service worker support (`@angular/service-worker`) for offline capabilities and caching strategies.
32. **What is Angular CLI's `ng build --prod` flag? How does it optimize a production build?**
    - Explanation of production build optimizations such as AOT compilation, tree-shaking, minification, dead code elimination, and differential loading.
33. **Explain dependency injection (DI) with multiple providers in Angular.**
    - Managing multiple providers in Angular, how multi-providers work, and scenarios where you might need multiple instances of a service.
34. **How does `HttpClient` handle interceptors? Can you chain multiple interceptors?**
    - Explanation of how interceptors can be chained in Angular's `HttpClientModule` to modify requests or responses and scenarios where multiple interceptors are useful (e.g., logging, error handling, authentication).
35. **What is a `BehaviorSubject`, and how does it differ from a regular `Subject` in RxJS?**
    - Differences between `Subject`, `BehaviorSubject`, `ReplaySubject`, and `AsyncSubject` in RxJS, focusing on the state management benefits of `BehaviorSubject`.

36. **How do you optimize Angular applications for better performance in a large-scale enterprise application?**
    - Detailed discussion on performance optimization techniques like:
    - Lazy loading modules
    - OnPush change detection
    - Using `trackBy` with `ngFor`
    - Using `ngZone.runOutsideAngular()` for non-critical tasks
    - Optimizing RxJS subscriptions (e.g., `takeUntil`, `unsubscribe`)
    - Memory management and profiling with browser dev tools.
37. **What are the major differences between AOT (Ahead-of-Time) and JIT (Just-in-Time) compilation in Angular?**
    - Advantages of AOT over JIT in production builds, including faster load times, early template error detection, and smaller bundle sizes.
38. **What is `DeferLoading` in Angular? How does it improve performance?**
    - Explanation of deferring non-critical content to be loaded later, either through lazy loading components or loading modules asynchronously to improve the perceived performance of the app.
39. **How do you integrate Web Workers into an Angular application?**
    - Using Web Workers to offload heavy computations to a separate thread, improving UI responsiveness. How to use Angular CLI to generate and work with Web Workers.
40. **Explain how Angular ensures security with the DOM via sanitization.**
    - How Angular automatically protects against cross-site scripting (XSS) attacks with its built-in sanitization for templates and the `DomSanitizer` service.
41. **What are `Zones` in Angular, and how do they affect performance? How can you run code outside Angular's zone?**
    - Detailed discussion on `Zone.js`, how it powers Angular's change detection, and scenarios for improving performance by running code outside the zone using `ngZone.runOutsideAngular()`.
42. **How would you approach testing in an Angular application?**
    - Explanation of unit testing with `Karma` and `Jasmine`, mocking services, `HttpClient`, and components, using `TestBed` for component testing, and writing end-to-end tests using `Protractor` or `Cypress`.
43. **What is differential loading, and how does it work in Angular?**
    - Explanation of how differential loading builds different bundles for modern (ES2015+) and legacy (ES5) browsers to optimize load times for modern browsers while maintaining compatibility.
44. **Explain how to handle multiple environments in Angular.**
    - Configuring environment-specific variables using Angular's `src/environments` directory and the `fileReplacements` feature in `angular.json` for different environments (e.g., development, production, staging).
45. **How does Angular handle internationalization (i18n) and localization (l10n)?**

    - Steps for implementing internationalization (i18n) using Angular's built-in i18n tools, configuring different languages, and setting up translations for dynamic content.
46. **How do you use `RouterModule.forRoot()` to configure a guard for protecting a child route in Angular?**
    - Explanation of how to set up route guards (e.g., `CanActivate`, `CanDeactivate`, `CanLoad`) in routing configuration and how to protect child routes.
47. **What are Angular zones, and how do they work with change detection?**
    - Discussion on the use of `zone.js` to track asynchronous operations and propagate changes through Angular's change detection mechanism.
48. **Explain `ng-content` and how to use content projection in Angular.**
    - How content projection allows you to project external content into a component using `ng-content`, along with its advantages in building reusable components.
49. **How do you manage multiple API calls and combine their results using RxJS operators?**
    - Handling multiple asynchronous operations (e.g., `forkJoin`, `combineLatest`, `zip`, `mergeMap`) and combining results using RxJS for efficient HTTP request management.
50. **How do you handle large file uploads in Angular?**
    - Discuss strategies like chunked uploads, background uploads, progress bars, and error handling for large file uploads using Angular and `HttpClient`.

What are the common patterns for handling state management in Angular?

- Discuss common approaches such as using services with `BehaviorSubject`, `NgRx`, and `Akita`. Compare them based on the use case.

How does Angular handle immutability? How would you ensure that your app follows immutability best practices?

- Explanation of immutability concepts, usage of immutable libraries (`Immutable.js`), or manual techniques (e.g., using `Object.assign()` or the spread operator).

What is the role of `Injector` in Angular? How does it differ from `NgModule`?

- Deep dive into how the `Injector` works, its role in Angular's DI system, and the relationship with `NgModule` in terms of providing services.

How do you handle concurrency in Angular using RxJS?

- Discuss RxJS operators like `mergeMap`, `switchMap`, `concatMap`, and how to choose between them to manage concurrency and sequencing of asynchronous operations.

What are `asynchronous validators` in Angular? How do you implement them?

- Explanation of how to use asynchronous validators, how to integrate them with reactive forms, and real-world use cases such as validating unique usernames.

What is a module federation, and how can it be implemented in Angular?

- Explanation of micro frontends using module federation in Angular with Webpack 5 and how to share code across multiple Angular apps without duplication.

What is `Injector` Hierarchy in Angular, and how does it affect service instance creation?

- Discuss the hierarchical structure of injectors in Angular (component-level vs module-level injectors) and how they affect singleton services and scoped service instances.

How do you deal with memory leaks in Angular?

- Techniques like unsubscribing from observables, cleaning up event listeners, using `takeUntil`, `AsyncPipe`, and profiling the application using Chrome DevTools or Angular DevTools to detect memory leaks.

Explain `HttpClientModule`'s features such as `interceptors`, handling request headers, and retry mechanisms.

- Comprehensive discussion on interceptors for modifying requests, adding headers, retries with `retry()` or `retryWhen()` operators, and global error handling.

What is `Angular Schematics`, and how can you create a custom schematic?

- Overview of Angular Schematics and how to create custom templates or automate tasks such as scaffolding components, modules, or services.

How would you handle authorization and authentication in an Angular application?

- Discussion on JWT authentication, implementing guards (`CanActivate`, `CanLoad`), and interceptors for token injection. Integrating third-party solutions like Auth0 or Firebase Authentication.

How does `NgZone` work, and what is the purpose of `ngZone.runOutsideAngular()`?

- Detailed explanation of how Angular uses zones for change detection, and scenarios where running code outside of Angular's zone (e.g., animations or high-frequency events) can improve performance.

What are the differences between `Subject`, `BehaviorSubject`, `ReplaySubject`, and `AsyncSubject` in RxJS?

- Discuss practical scenarios where each type of subject is useful, focusing on how they emit values and store history for subscribers.

How do you implement route resolvers to prefetch data before route activation?

- Explanation of how to use the `Resolve` interface to fetch data from APIs before navigating to a route and the benefits of this approach.

Explain the purpose and benefits of `tree-shaking` in Angular.

- Detailed discussion of how Angular's AOT compilation and bundlers (like Webpack) perform tree-shaking to remove unused code and reduce bundle sizes.

What are pure and impure pipes in Angular? What are the performance implications of each?

- Explanation of how pure pipes are recalculated only when inputs change, whereas impure pipes are recalculated on every change detection cycle, and when to use one over the other.

What is a `Zone.js` and how does it relate to Angular's change detection?

- Discuss how `Zone.js` allows Angular to detect asynchronous operations (e.g., promises, event listeners) and how this affects automatic change detection.

How do you handle API pagination in Angular using `HttpClient` and RxJS?

- Techniques for handling large data sets with paginated API results using RxJS operators like `mergeMap`, `concatMap`, and how to build an efficient infinite scrolling or pagination system.

How do you configure multi-language support (i18n) in Angular?

- Explanation of Angular's built-in internationalization support (`@angular/localize`), configuring translation files, and lazy-loading language-specific modules.

What is differential loading, and why is it important in Angular applications?

- How Angular uses differential loading to serve modern ES2015+ code to modern browsers while delivering legacy ES5 code to older browsers.

How would you design an Angular app for offline support using Service Workers?

- Explanation of how to configure service workers using `@angular/service-worker`, managing app shell caching, and building progressive web apps (PWAs) for offline functionality.

How do you prevent duplicate HTTP requests in Angular?

- Techniques such as caching responses using RxJS operators (`shareReplay`), using interceptors to avoid duplicate requests, or debouncing requests in case of user interaction.

How would you handle large media (images or videos) uploads in Angular?

- Techniques such as chunking file uploads, handling progress bars, retry mechanisms, and optimizing uploads with `FormData` and `HttpClient`.

What are decorators in Angular, and how are they implemented in TypeScript?

- Explanation of how Angular's decorators (`@Component`, `@Injectable`, `@Input`, etc.) work with TypeScript's metadata and reflection APIs.

How do you secure an Angular application against common web vulnerabilities like XSS, CSRF, and Clickjacking?

- Techniques for handling security in Angular, such as Angular's built-in DOM sanitization, CSRF tokens with APIs, and securing routes with guards.

How does Angular Universal handle server-side rendering (SSR), and what are the benefits of SSR?

- Explanation of how Angular Universal provides SSR to improve SEO, perceived performance, and faster initial page load by pre-rendering pages on the server.

What are the different methods of communication between components in Angular?

- Discuss different strategies such as `@Input()` and `@Output()` bindings, shared services with RxJS subjects, and parent-child component communication using `ViewChild` and `ContentChild`.

What is lazy loading, and how can you implement it in an Angular app?

- Explanation of lazy loading modules using `loadChildren` in routing, the performance benefits of lazy loading, and how to preload critical modules.

How do you handle error handling globally in an Angular application?

- Using `ErrorHandler` service for centralized error handling, creating custom error handling strategies, and integrating global error handling with logging services or reporting tools like Sentry.

What are `Dynamic Components` in Angular, and how do you load them at runtime?

- Explanation of dynamic component creation using `ComponentFactoryResolver` or the newer `ViewContainerRef.createComponent()` method in Angular 13+.

How do you set up testing in an Angular project, and how would you write a unit test for a service that makes HTTP calls?

- Setting up Angular testing using `Karma` and `Jasmine`, mocking HTTP calls with `HttpTestingController`, and writing test cases with `TestBed`.

How do you handle long-running tasks or background processing in Angular?

- Approaches for handling long-running tasks, such as using Web Workers for CPU-intensive tasks or RxJS for managing background tasks without blocking the UI.

How would you optimize an Angular app for a mobile-first experience?

- Discussion on responsive design using CSS frameworks like Bootstrap or Angular Material, performance optimization techniques (like lazy loading, image compression, and viewport-specific data loading), and PWA features.

What is WebSocket communication in Angular, and how do you implement it?

- Discuss how to set up WebSocket communication using Angular's `WebSocketSubject` from `RxJS` and real-time applications where it is commonly used (e.g., chat apps, live data feeds).

How does Angular handle dependency injection in child modules vs. root modules?

- Explanation of the difference between providing a service at the root module level (`providedIn: 'root'`) vs child modules, and how singleton services work in Angular's hierarchical DI system.

What are `forwardRef()` and `Optional()` in Angular, and when would you use them?

- Explanation of the `forwardRef()` function for handling circular dependencies in Angular's dependency injection system and how `Optional()` allows injecting services that may or may not exist.

How would you implement an Angular application with micro frontends architecture?

- Discussion of strategies for micro frontends, including module federation, sharing Angular libraries across multiple apps, and the pros and cons of monolithic vs micro frontend architectures.

Explain the concept of feature modules in Angular and how to structure large applications using them.

- Discuss how to organize an Angular app into multiple feature modules, separating concerns for better maintainability, lazy loading, and scalability.

What are Angular animations, and how can you implement complex animations using `@angular/animations`?

- Overview of Angular animations using `trigger()`, `state()`, `transition()`, and `animate()`. Implementing complex animation sequences, route transition animations, and performance considerations.

How do you handle data persistence across route changes or page reloads in Angular?

- Techniques like using local storage, session storage, `IndexedDB`, or Angular service worker's caching mechanisms for persistent data storage.

What is the purpose of `ApplicationRef` in Angular, and when would you use it?

- Explanation of `ApplicationRef` for manually controlling change detection, dynamic component loading, or detecting when Angular has stabilized after change detection.

How would you handle file uploads and downloads securely in an Angular application?

- Techniques for handling file uploads using `FormData`, displaying progress bars, securing downloads with tokens, and ensuring files are sanitized to prevent security vulnerabilities like XSS.

How do you implement client-side caching of HTTP requests in Angular, and when would you invalidate cached data?

- Explanation of caching strategies using RxJS `shareReplay()`, `CachingInterceptor`, and invalidation strategies based on time (TTL), manual triggers, or versioning data.

What is a `ContentChild` decorator in Angular, and how is it different from `ViewChild`?

- Discussion of the `ContentChild` decorator for accessing projected content in a component and how it differs from `ViewChild`, which accesses child components or DOM elements in the view.

How do you implement optimistic updates in Angular?

- Explanation of implementing optimistic UI updates by updating the UI before the server response is received, handling rollbacks if the server fails, and using RxJS to manage state transitions.

What are the strategies to prevent race conditions in Angular applications?

- Discuss techniques to prevent race conditions, such as using RxJS operators (`concatMap`, `exhaustMap`), promises with `async/await`, and ensuring proper sequencing of API calls.

What are providers in Angular, and what are the different types of providers (`useClass`, `useValue`, `useExisting`, `useFactory`)?

- Explanation of Angular's DI system and the various ways to provide services using the `provide` syntax, differences between `useClass`, `useValue`, `useExisting`, and `useFactory`.

What is the role of `HostListener` and `HostBinding` in Angular, and how do they enhance component behavior?

- Detailed explanation of `HostListener` for handling DOM events within a component and `HostBinding` for binding properties to host elements, useful for creating highly reusable components.

How do you implement a custom event emitter in Angular?

- Explanation of how to create custom events using `@Output()` with `EventEmitter` and how to propagate events up the component tree in a controlled manner.

How would you design a reusable Angular component library for internal use? - Steps for creating a modular, reusable component library, configuring `ng-packagr` for Angular packaging, and ensuring the library is easily consumable in multiple applications with proper versioning.

How does Angular handle nullish coalescing and optional chaining in templates? - Discuss the use of nullish coalescing (`??`) and optional chaining (`?.`) operators in Angular templates to prevent null/undefined errors while rendering data in components.

How would you implement skeleton loading in Angular? - Explanation of how to implement skeleton loaders to improve the perceived performance of applications while loading content asynchronously, and integrating it into UI components.

What is the `Renderer2` service in Angular, and when would you use it instead of direct DOM manipulation? - Detailed explanation of the `Renderer2` service for manipulating DOM elements in a platform-agnostic way, such as for web workers, SSR, or ensuring compatibility with native mobile environments.

How does Angular detect and manage changes in deeply nested objects or arrays? - Discussion on how Angular's default change detection only detects changes in object references, and strategies like immutability, using `OnPush` change detection strategy, or `trackBy` for optimizing performance.

Explain the concept of `PreloadingStrategy` in Angular, and how can you customize preloading modules? - Overview of Angular's built-in `PreloadAllModules` strategy for lazy-loaded modules, and how to create custom preloading strategies to optimize performance based on app requirements.

How do you handle circular dependencies in Angular modules or services? - Techniques for resolving circular dependencies using `forwardRef()`, refactoring module/service dependencies, and ensuring modularity in large-scale applications.

How would you structure and architect an enterprise-grade Angular application? - Discussion of best practices for structuring large Angular applications, organizing feature and core modules, following the SOLID principles, and using state management (like NgRx or Akita) to handle complex workflows.

What are `pure` and `impure pipes` in Angular, and how do you decide when to use each? - Explanation of the differences between pure and impure pipes, performance implications, and use cases where an impure pipe is necessary (e.g., for dynamic or changing data).

What are the different testing strategies you follow for an Angular application (unit, integration, e2e)? - Overview of unit testing using `Karma`/`Jasmine`, integration testing with `TestBed`, and end-to-end (e2e) testing using `Protractor` or `Cypress`, and strategies for test-driven development (TDD) in Angular.

How would you implement a highly performant Angular application that handles real-time data streaming (e.g., WebSockets)? - Explanation of real-time data handling using WebSockets or SignalR with RxJS operators to handle asynchronous data streams, ensuring scalability and performance optimizations for real-time applications.

How do you configure Angular to work with different environments (e.g., development, staging, production)? - Overview of using Angular's environment configurations (`src/environments`), setting up different builds using `angular.json`, and managing different environment-specific settings like API URLs or feature flags.

How would you implement SSR (Server-Side Rendering) with Angular Universal to improve SEO and performance? - Explanation of how Angular Universal works for SSR, the benefits for SEO and faster initial page loads, and steps to set up SSR in an existing Angular application.

What are the performance implications of the `OnPush` change detection strategy in Angular? - Discuss how `OnPush` optimizes change detection by checking component inputs for changes by reference and when it's appropriate to use `OnPush` in comparison to the default `ChangeDetectionStrategy.Default`.

How does Angular's Ivy compiler improve application performance and development? - Explanation of how Ivy, Angular's default rendering engine, improves tree-shaking, build times, smaller bundle sizes, and enables more advanced features like AOT compilation and faster rendering.

What is the purpose of `ngTemplateOutlet` and how do you use it in Angular? - Discuss how to use `ngTemplateOutlet` for dynamic template rendering in Angular components, creating flexible, reusable UI components that can accept various templates as input.

How do you debug Angular applications efficiently? - Techniques for debugging Angular applications using Chrome DevTools, Angular DevTools, source maps, `console.log()`, and more advanced strategies such as performance profiling, heap snapshots, and memory leak detection.

What is the purpose of the `ControlValueAccessor` interface, and how do you implement a custom form control? - Detailed explanation of `ControlValueAccessor` for creating custom form controls that integrate seamlessly with Angular's reactive forms or template-driven forms, and how to bind values and handle validation.

What are the best practices for handling API errors in Angular applications? - Techniques for managing errors globally in Angular using HTTP interceptors, error services, displaying user-friendly error messages, and retrying failed requests with `retry()` or `retryWhen()` operators from RxJS.

How would you handle nested forms in Angular? - Explanation of managing nested forms using `FormGroup`, `FormArray`, and how to dynamically add/remove form controls while maintaining validation and performance.

What are the differences between `ViewContainerRef` and `ComponentFactoryResolver` in Angular? - Discussion of how to dynamically load components at runtime using `ViewContainerRef` to insert components into the DOM, and `ComponentFactoryResolver` to create components dynamically with full lifecycle management.

How do you handle large datasets in Angular for optimized rendering and user experience? - Techniques such as virtual scrolling using `cdk-virtual-scroll-viewport`, pagination, and infinite scrolling to efficiently manage large datasets, improving performance.

What are the best practices for managing global state in large Angular applications? - Discussion of state management libraries like `NgRx`, `Akita`, or `NGXS`, and how to structure global state to handle actions, reducers, effects, and selectors to ensure scalability and maintainability.

How would you set up lazy loading in Angular for feature modules, and how does it impact application performance? - Explanation of lazy loading modules using `loadChildren` in the Angular router, benefits in terms of reduced initial bundle sizes, and optimizing route-based loading for faster app performance.

What is the difference between `canActivate`, `canDeactivate`, and `canLoad` guards in Angular? - Detailed explanation of how route guards protect navigation and the specific use cases for each, with examples for implementing authentication or role-based access control (RBAC).

How do you handle memory leaks in Angular applications, and what tools or techniques can you use to debug them? - Explanation of common causes of memory leaks (e.g., subscriptions not being unsubscribed), techniques for using RxJS `takeUntil`, `unsubscribeOnDestroy`, or Angular's `async` pipe, and how to debug using Chrome DevTools.

Explain Angular's hierarchical dependency injection system and how it impacts service scope across different modules and components. - Overview of Angular's DI system, where providers can be registered at different levels (root, feature module, component), and how to control the scope of services (singleton, component-scoped, etc.).

How do you handle dynamic form creation in Angular where the structure of the form is not known at compile time? - Explanation of using `FormBuilder` and `FormArray` for dynamic forms, creating controls dynamically based on API responses, and handling validation and dynamic form groups.

What is Angular's `$event` object, and how can it be used in event binding? - Discuss the usage of `$event` in templates to capture and pass event data (e.g., `click`, `input`, `change`), and how to leverage it in custom event handling.

How does Angular's change detection mechanism work under the hood? - Deep dive into Angular's change detection system (zone.js), the difference between dirty checking and the actual implementation, how `ApplicationRef.tick()` works, and scenarios where manual change detection (`ChangeDetectorRef`) is necessary.

How would you implement drag-and-drop functionality in Angular? - Using the Angular CDK `DragDropModule` to implement drag-and-drop features, handling events like `dropped`, customizing placeholders, and creating reordering logic within lists.

How does the Angular Ahead-of-Time (AOT) compilation process work, and what are the advantages of using AOT? - Explanation of the Angular AOT compiler, its advantages over Just-in-Time (JIT) compilation, such as smaller bundle sizes, faster rendering, and template error detection at build time.

What is `NgZone` in Angular, and how do you use it to optimize performance? - Explanation of how `NgZone` helps in managing Angular's change detection mechanism, how to execute code outside of Angular's zone using `runOutsideAngular()` for performance-critical operations, and re-entering change detection when necessary.

How would you create and handle custom structural directives in Angular (e.g., like `*ngIf`)? - Explanation of creating custom structural directives using the `TemplateRef` and `ViewContainerRef` to conditionally render DOM elements or apply custom logic to Angular templates.

What is Angular's content projection (`ng-content`), and how does it differ from transclusion in AngularJS? - Explanation of content projection in Angular, how to pass and display dynamic content in child components, single-slot vs multi-slot content projection, and its advantages over AngularJS transclusion.

How would you implement an Angular service worker for caching and offline functionality? - Overview of Progressive Web App (PWA) features in Angular using `@angular/service-worker`, how to configure caching strategies, handling offline capabilities, and ensuring automatic updates with background sync.

Explain the difference between static and dynamic components in Angular. How do you create and insert dynamic components at runtime? - Explanation of how static components are defined in templates, whereas dynamic components are instantiated using `ComponentFactoryResolver` or Angular's Ivy renderer, and how to manage their lifecycle.

What is the `Injector` service in Angular, and how can it be used for dynamically injecting services? - Detailed explanation of the `Injector` service, how it allows for creating instances of services dynamically and can be used for advanced DI use cases like plugin systems or injecting dependencies outside of Angular components.

What are the best practices for securing Angular applications against XSS and CSRF attacks? - Explanation of Angular's built-in security features (sanitization, `DomSanitizer`, and `HttpClient` XSRF token handling), and external security measures such as Content Security Policy (CSP) and OAuth 2.0 for authentication and authorization.

How would you handle global error handling in Angular using `ErrorHandler`? - Explanation of creating a global error handling service by extending Angular's `ErrorHandler` class, logging errors to external services (e.g., Sentry), and displaying user-friendly error messages.

How do you manage multiple themes in an Angular application? - Techniques for supporting multiple themes by dynamically loading stylesheets, using CSS variables or SCSS, and ensuring the themes are applied consistently across components using Angular Material's theming support or custom solutions.

What are Angular's lifecycle hooks, and in what order are they executed? - Detailed explanation of Angular's component lifecycle hooks (`ngOnChanges`, `ngOnInit`, `ngDoCheck`, etc.) and the sequence in which they are called. Understanding when and why to use each hook.

How would you implement two-way data binding in Angular without using the built-in `[(ngModel)]` syntax? - Explanation of how to implement two-way data binding manually using `@Input()` and `@Output()` decorators, along with event emitters to synchronize data between parent and child components.

What is Angular Universal, and how would you configure it for server-side rendering (SSR)? - Discuss how Angular Universal enables SSR for SEO and performance benefits, steps to configure it, and handling issues like lazy loading, API calls, and state rehydration in SSR.

How do you handle complex form validations in Angular using custom validators? - Explanation of synchronous and asynchronous custom validators for Angular forms, how to compose multiple validators, and manage cross-field validation.

How would you prevent over-fetching and under-fetching of data in Angular applications? - Discuss using GraphQL or REST best practices, applying data querying techniques like pagination, filtering, and using appropriate HTTP methods to minimize data transfer and ensure efficient fetching.

What is Angular's Ivy engine, and how has it changed the way components and directives are compiled and rendered? - Overview of Ivy, Angular's default rendering engine, its benefits like improved debugging, tree shaking, faster compilation, and smaller bundle sizes.

How do you implement real-time notifications in Angular using WebSockets? - Explanation of setting up WebSockets in Angular using RxJS, how to integrate with backend services, and ensuring reliable communication for real-time updates such as chat applications, live notifications, or collaborative editing.

What is the role of `providedIn: root` in Angular services, and how does it impact tree-shaking? - Discuss the meaning of `providedIn: 'root'` in Angular services for automatic service registration and its impact on tree-shaking by removing unused services from the final bundle.

How would you implement route-based dynamic breadcrumbs in Angular? - Techniques for implementing breadcrumbs by leveraging Angular's router and `ActivatedRoute` to create dynamic breadcrumb trails based on route parameters and hierarchies.

How do you ensure that Angular applications are accessible (WCAG compliant)? - Best practices for making Angular apps accessible by following Web Content Accessibility Guidelines (WCAG), using semantic HTML, ARIA attributes, keyboard navigation, and testing accessibility with tools like Axe.

What is the role of `BehaviorSubject` in Angular applications, and how does it differ from `Subject`? - Explanation of how `BehaviorSubject` retains the latest value for new subscribers, its use cases (such as state management), and differences compared to `Subject`, which only emits to existing subscribers.

How would you implement role-based authorization in Angular applications? - Techniques for securing routes and UI elements based on user roles using Angular guards, `canActivate`/`canLoad`, and how to integrate role-based authorization with JWT tokens or OAuth.

What is the purpose of `ngZone.runOutsideAngular()`, and when should it be used? - Explanation of how `runOutsideAngular()` helps in optimizing performance by preventing unnecessary change detection cycles, especially in scenarios like large animations or third-party libraries.

How would you integrate Angular with a headless CMS (e.g., Contentful, Strapi)? - Discuss how to connect Angular applications to headless CMSs via APIs, handling data fetching, and optimizing performance and caching strategies.

Scenario-Based Questions:

1. **How would you migrate a large-scale Angular application from Angular 7 to Angular 14?**
    - Walk through the step-by-step approach, from reading the official upgrade guide to testing and handling deprecated features.
2. **You have a component with a large number of event listeners (e.g., scroll, mouse movements), and performance is slowing down. How would you optimize it?**

    - Use of `throttleTime()`, `debounceTime()` from RxJS, running code outside of Angular’s zone, or removing unnecessary event listeners.
3. **An Angular application’s initial load time is too slow. What steps would you take to reduce this?**
    - Discuss lazy loading, AOT, preloading strategies, differential loading, tree-shaking, and minification.