# Angular interview questions

Angular interview questions and answers

give questions and answers from basic, intermediate and advanced

please cover all concepts in question and answers, expecting minimux 500+ questions

Creating a **complete list of 500+ Angular interview questions and answers** spanning basic, intermediate, and advanced topics is a very large task, but I can definitely get you started with a **comprehensive, structured, and categorized** set. Then you can ask me to continue in batches or for specific topics.

---

## Angular Interview Questions & Answers

### Basic Level (Fundamentals)

### 1. What is Angular?

**Answer:**
Angular is a TypeScript-based open-source front-end web application framework developed by Google. It allows developers to build single-page client applications using HTML and TypeScript.

### 2. What are components in Angular?

**Answer:**
Components are the basic building blocks of Angular applications. Each component consists of an HTML template, a TypeScript class that defines behavior, and CSS styles.

### 3. What is data binding in Angular?

**Answer:**
Data binding is the mechanism to coordinate parts of a template with parts of a component. It can be one-way or two-way, such as interpolation (`{{}}`), property

binding (`[property]`), event binding (`(event)`), and two-way binding (`[(ngModel)]`).

## 4. Explain Angular Modules.

**Answer:**
Modules (NgModules) organize an Angular app into cohesive blocks of functionality. Each module can contain components, directives, pipes, and services.

## 5. What is a directive?

**Answer:**
Directives are classes that add behavior to elements in Angular applications. They can be structural (e.g., `*ngIf`, `*ngFor`) or attribute directives (e.g., `ngClass`, `ngStyle`).

## 6. What is the difference between `*ngIf` and `[hidden]`?

**Answer:**
`*ngIf` adds or removes the element from the DOM based on the condition, while `[hidden]` just toggles the visibility by applying CSS `display: none`.

## 7. What is dependency injection in Angular?

**Answer:**
Dependency Injection (DI) is a design pattern used to supply a component or service with its dependencies rather than creating them directly.

## 8. What are Angular lifecycle hooks?

**Answer:**
Lifecycle hooks are methods that Angular calls during different stages of a component's life, such as `ngOnInit()`, `ngOnChanges()`, `ngOnDestroy()`, etc.

## 9. What is the Angular CLI?

**Answer:**
Angular CLI is a command-line interface tool that helps to initialize, develop, scaffold, and maintain Angular applications easily.

## 10. What is the difference between AngularJS and Angular?

**Answer:**
AngularJS (version 1.x) is based on JavaScript, uses MVC architecture, and is less

performant. Angular (version 2+) is rewritten in TypeScript, uses components and modules, and offers better performance and scalability.

---

## Intermediate Level

### 11. What are services in Angular?

**Answer:**
Services are classes used to share data or logic across components. They are typically injected via dependency injection.

### 12. What is RxJS and how does Angular use it?

**Answer:**
RxJS (Reactive Extensions for JavaScript) is a library for reactive programming using Observables. Angular uses RxJS extensively for handling asynchronous data streams, such as HTTP requests or events.

### 13. What is an Observable?

**Answer:**
An Observable is a stream of data that can emit multiple values over time, which can be subscribed to for asynchronous programming.

### 14. What is the difference between an Observable and a Promise?

**Answer:**
A Promise handles a single asynchronous event while Observables can handle multiple events over time and can be canceled.

### 15. Explain Angular Routing.

**Answer:**
Routing enables navigation between different views or components in an Angular application. The Router maps URLs to components and manages navigation state.

### 16. What is lazy loading in Angular?

**Answer:**
Lazy loading is a technique to load Angular modules only when required (on demand), which improves the application’s load time.

### 17. What is the difference between template-driven and reactive forms?

**Answer:**

Template-driven forms are simpler and use directives in the template. Reactive forms are more powerful and use explicit, programmatic control over the form model and validation.

### 18. What are pipes in Angular?

**Answer:**

Pipes transform data in templates before displaying it. Angular provides built-in pipes (e.g., `DatePipe`, `CurrencyPipe`), and custom pipes can be created.

### 19. How does Angular handle security?

**Answer:**

Angular provides built-in sanitization, protects against cross-site scripting (XSS), uses template security, and has tools like the DomSanitizer service.

### 20. What is the purpose of `trackBy` in `*ngFor`?

**Answer:**

`trackBy` helps Angular track items efficiently by a unique identifier, improving performance during list updates.

---

## Advanced Level

### 21. How does Angular's change detection work?

**Answer:**

Angular uses a zone-based change detection mechanism to detect changes and update the DOM. It runs change detection when async events occur, and developers can configure change detection strategies like `Default` and `OnPush`.

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

**Answer:**

AOT compiles Angular templates and components during build time, producing faster rendering and smaller bundle sizes.

### 23. Explain Angular Universal.

**Answer:**

Angular Universal is a server-side rendering (SSR) solution for Angular, which improves performance and SEO by rendering the app on the server.

## 24. What are Angular decorators?

**Answer:**

Decorators are functions that add metadata to classes, properties, or methods to tell Angular how to process them. Examples include `@Component()`, `@NgModule()`, and `@Injectable()`.

## 25. How do you create a custom structural directive?

**Answer:**

By creating a directive with the `@Directive()` decorator and using the `TemplateRef` and `ViewContainerRef` to add or remove elements from the DOM.

## 26. What is the difference between `forRoot()` and `forChild()` methods in routing modules?

**Answer:**

`forRoot()` configures the router at the app root level and should be used once. `forChild()` configures routes in feature modules.

## 27. What are Zones in Angular?

**Answer:**

Zones are execution contexts that help Angular track async operations and trigger change detection when needed.

## 28. How does Angular handle internationalization (i18n)?

**Answer:**

Angular supports i18n using marked templates, translation files, and tools to generate localized builds.

## 29. What are entry components in Angular?

**Answer:**

Entry components are components that are loaded dynamically and aren’t referenced in templates.

## 30. Explain dependency injection hierarchical injectors in Angular.

**Answer:**

Angular uses a hierarchical dependency injection system where child injectors can override services provided by parent injectors.

---

## More Topics to Cover (for later expansion)

- Angular animation system
- Angular Material & CDK
- Performance optimization techniques
- Unit testing with Jasmine & Karma
- End-to-end testing with Protractor / Cypress
- State management with NgRx / Akita
- Angular Elements (Web Components)
- Web Workers in Angular
- Security best practices and OWASP threats
- Advanced RxJS operators and usage
- Custom form validators
- HttpClient interceptors
- Angular compiler options & build optimizations
- Angular platform-server vs platform-browser
- Angular schematics and CLI customization

---

### How do you want to proceed?

- Should I continue adding questions by category?
- Do you want me to generate a downloadable file?
- Or focus on specific Angular domains (e.g., RxJS, Routing, Forms)?

I can generate the next 100 questions with detailed answers or focus on your priorities!

generate the next 100 questions with detailed answers

Got it! Here are the **next 100 Angular interview questions with detailed answers**, continuing from the basics to intermediate and advanced topics to build a strong, thorough knowledge base.

---

# Angular Interview Questions 11–110

---

## 11. What is the purpose of the `@NgModule` decorator?

**Answer:**

`@NgModule` defines an Angular module and contains metadata about the module including declarations (components, directives, pipes), imports (other modules), providers (services), and bootstrap components.

---

## 12. How do you create and use a service in Angular?

**Answer:**

You create a service by generating a class and decorating it with `@Injectable()`. Then you provide it in a module or component, and inject it via the constructor where needed.

```typescript
@Injectable({ providedIn: 'root' })
export class MyService {
  getData() { return ... }
}
```

Inject in component:

```typescript
constructor(private myService: MyService) {}
```

---

## 13. Explain the difference between `providers` array in `@NgModule` and `@Component`.

**Answer:**

`providers` in `@NgModule` registers services at the module (singleton) level. In `@Component`, providers are scoped only to that component and its children, creating separate instances.

---

## 14. What is the Angular Template Syntax?

**Answer:**

Angular template syntax includes interpolation ( `{{}}` ), property binding ( `[property]` ), event binding ( `(event)` ), two-way binding ( `[(ngModel)]` ), and structural directives ( `*ngIf` , `*ngFor` ).

---

## 15. How do you implement two-way data binding?

**Answer:**

Using the `[(ngModel)]` directive which combines property binding and event binding to sync data between component and template.

```html
<input [(ngModel)]="name" />
```

---

## 16. What are structural directives?

**Answer:**

Structural directives change the DOM layout by adding/removing elements. Examples: `*ngIf` , `*ngFor` , `*ngSwitch` .

---

## 17. What are attribute directives?

**Answer:**

Attribute directives change the appearance or behavior of an element without changing the DOM layout, e.g., `ngClass` , `ngStyle` .

---

## 18. How does Angular handle forms?

**Answer:**

Angular supports two types: Template-driven (simpler, declarative) and Reactive forms (more scalable, programmatic). Both use `FormsModule` or `ReactiveFormsModule` .

---

## 19. How do you create a reactive form?

**Answer:**

Import `ReactiveFormsModule` and create a `FormGroup` and `FormControl` in the component:

```typescript
this.myForm = new FormGroup({
  name: new FormControl(''),
  email: new FormControl('')
});
```

Bind in template:

```html
<form [formGroup]="myForm">
  <input formControlName="name" />
</form>
```

---

## 20. What are Observables and how do you use them in Angular?

**Answer:**

Observables are streams of async data. You subscribe to them to receive data, e.g., HTTP requests:

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

---

## 21. What is the role of `async` pipe in Angular templates?

**Answer:**

`async` pipe automatically subscribes to an Observable or Promise and returns the latest value. It also unsubscribes when the component is destroyed.

```html
```

```
<div *ngIf="data$ | async as data">{{ data }}</div>
```

---

## 22. How do you handle HTTP requests in Angular?

**Answer:**

Use the `HttpClient` service from `@angular/common/http` module for HTTP methods like GET, POST, PUT, DELETE. It returns Observables.

---

## 23. What is the difference between `Subject` and `BehaviorSubject` in RxJS?

**Answer:**

`Subject` is an Observable and Observer. `BehaviorSubject` holds the latest emitted value and emits it immediately to new subscribers.

---

## 24. How do you protect routes in Angular?

**Answer:**

Use route guards (`CanActivate`, `CanDeactivate`) to check conditions before navigating.

Example:

```typescript
canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
  return this.authService.isLoggedIn();
}
```

---

## 25. What is Angular routing lazy loading?

**Answer:**

Loading feature modules only when needed via the router using `loadChildren`. This improves initial load time.

---

## 26. What is the difference between `@Input()` and `@Output()` decorators?

**Answer:**

`@Input()` passes data from parent to child component.

`@Output()` emits events from child to parent using `EventEmitter`.

---

## 27. How do you create a custom pipe?

**Answer:**

Create a class implementing `PipeTransform` and decorate with `@Pipe`.

```typescript
@Pipe({ name: 'myPipe' })
export class MyPipe implements PipeTransform {
  transform(value: any): any {
    return transformedValue;
  }
}
```

---

## 28. What are Angular lifecycle hooks?

**Answer:**

Hooks called at specific times:

- `ngOnInit()` after component init
- `ngOnChanges()` on input changes
- `ngDoCheck()` custom change detection
- `ngOnDestroy()` before destruction

---

## 29. What is the purpose of `ng-content`?

**Answer:**

Allows projection of external content into component templates (content projection).

---

## 30. What is the difference between `constructor` and `ngOnInit()`?

**Answer:**

`constructor` is a TypeScript class constructor, runs first.

`ngOnInit()` is called by Angular after data-bound properties are initialized.

---

## 31. How to implement route parameters?

**Answer:**

Use `ActivatedRoute` to access route parameters:

```typescript
 this.route.params.subscribe(params => {
  this.id = params['id'];
});
```

---

## 32. How does Angular implement change detection?

**Answer:**

Angular runs a change detection cycle triggered by async events to update DOM. Default strategy checks all components; `OnPush` strategy checks only on input changes or events.

---

## 33. How do you optimize Angular app performance?

**Answer:**

Techniques include: lazy loading, AOT compilation, OnPush change detection, trackBy in `*ngFor`, avoiding unnecessary watchers.

---

## 34. What is the Angular Renderer2?

**Answer:**

An abstraction to manipulate the DOM safely in a platform-independent way.

---

## 35. How do you handle errors in HTTP requests?

**Answer:**

Use RxJS operators like `catchError` in the pipe.

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

---

## 36. What is an Angular Injector?

**Answer:**

It's responsible for instantiating dependencies and resolving providers.

---

## 37. How does Angular handle dependency injection hierarchy?

**Answer:**

Services provided in root are singleton; providers in components create separate instances in that subtree.

---

## 38. How do you create a custom validator?

**Answer:**

Create a function returning `ValidationErrors` or null:

```typescript
function myValidator(control: AbstractControl): ValidationErrors | null {
  return control.value === 'test' ? null : { invalid: true };
}
```

---

## 39. What are Angular Elements?

**Answer:**

Angular components packaged as web components for use outside Angular apps.

## 40. How do you use Angular Material?

**Answer:**
Import Material modules and use ready-made UI components like buttons, dialogs, etc.

---

## 41. What is the difference between `ngIf` and `hidden` attribute?

**Answer:**
`ngIf` adds/removes DOM elements; `hidden` toggles CSS visibility.

---

## 42. How do you use `ViewChild` and `ContentChild`?

**Answer:**
`ViewChild` accesses component/directive from the template.
`ContentChild` accesses projected content inside `ng-content`.

---

## 43. What is a promise and how does it differ from Observable?

**Answer:**
A Promise resolves once, Observables emit multiple values over time and support operators.

---

## 44. How do you perform unit testing in Angular?

**Answer:**
Use Jasmine and Karma test runner. Write specs with `describe` and `it` blocks, test components and services.

---

## 45. How do you perform end-to-end testing?

**Answer:**
Use Protractor or Cypress to automate UI interaction and verify app behavior.

---

## 46. What is the purpose of `entryComponents`?

**Answer:**

Specify components loaded dynamically (deprecated in Angular 9+ due to Ivy).

---

## 47. How do you share data between sibling components?

**Answer:**

Use a shared service with Observables to communicate.

---

## 48. What is NgZone?

**Answer:**

An Angular service to control change detection execution context.

---

## 49. How do you intercept HTTP requests?

**Answer:**

Implement an `HttpInterceptor` class and provide it in the module.

---

## 50. How can you prevent memory leaks in Angular?

**Answer:**

Unsubscribe from Observables in `ngOnDestroy()`, use `async` pipe, avoid global event listeners.

---

## 51. What is the difference between `TemplateRef` and `ViewContainerRef`?

**Answer:**

`TemplateRef` represents an embedded template.

`ViewContainerRef` is a container to insert views dynamically.

---

## 52. How do you create dynamic components?

**Answer:**

Use `ComponentFactoryResolver` or Angular 13+ `ViewContainerRef.createComponent()`.

---

## 53. How do you handle environment-specific configurations?

**Answer:**

Use Angular CLI `environment.ts` files and build configurations.

---

## 54. What is a `BehaviorSubject`?

**Answer:**

An RxJS subject that stores the current value and emits it to new subscribers immediately.

---

## 55. Explain the Angular `ChangeDetectionStrategy`.

**Answer:**

Two strategies:

- `Default`: checks all components every change detection run
- `OnPush`: checks only when inputs change or events occur

---

## 56. What is the difference between `forRoot()` and `forChild()` in routing?

**Answer:**

`forRoot()` registers providers and configures the app-level routes.
`forChild()` configures routes in feature modules without providers.

---

## 57. What is Angular Ivy?

**Answer:**

Angular’s next-generation compiler and rendering engine introduced in Angular 9 improving bundle size and speed.

---

## 58. How does Angular support SEO?

**Answer:**

Using Angular Universal for server-side rendering.

---

## 59. How do you implement internationalization (i18n)?

**Answer:**

Use Angular’s i18n tools, translation files, and runtime locale selection.

---

## 60. How to apply animations in Angular?

**Answer:**

Use Angular’s `@angular/animations` module with trigger, state, transition, and animate functions.

---

## 61. What is `ng-template`?

**Answer:**

A container for template content that is not rendered until used (e.g., with structural directives).

---

## 62. How do you debug Angular applications?

**Answer:**

Use browser dev tools, Augury extension, Angular CLI logging, and source maps.

---

## 63. What is a resolver in Angular routing?

**Answer:**

It pre-fetches data before activating a route.

---

## 64. How do you implement guards?

**Answer:**

Implement interfaces like `CanActivate`, `CanDeactivate`, and provide logic to allow or block navigation.

---

## 65. What is `ngZone.run()`?

**Answer:**

A method to explicitly run code inside Angular’s zone to trigger change detection.

---

## 66. How to share data between parent and child components?

**Answer:**

Use `@Input()` for parent to child and `@Output()` for child to parent communication.

---

## 67. What are pure and impure pipes?

**Answer:**

Pure pipes run only when input references change. Impure pipes run on every change detection cycle.

---

## 68. How do you enable production mode in Angular?

**Answer:**

Call `enableProdMode()` from `@angular/core` and build using the `--prod` flag.

---

## 69. What is differential loading?

**Answer:**

Angular builds separate bundles for modern and legacy browsers to optimize loading.

---

## 70. How do you configure lazy loading?

**Answer:**

Use `loadChildren` property in route configuration.

---

## 71. How do you inject a service into another service?

**Answer:**

Use constructor injection similarly to components.

---

## 72. What is the `APP_INITIALIZER` token?

**Answer:**

Allows executing code during app initialization (e.g., config loading).

---

## 73. How do you implement HTTP interceptors?

**Answer:**

Implement `HttpInterceptor` interface and register with `HTTP_INTERCEPTORS` multi-provider.

---

## 74. What is the difference between `ngOnChanges()` and `ngDoCheck()`?

**Answer:**

`ngOnChanges()` is called when input properties change.
`ngDoCheck()` is a custom change detection hook called every change detection cycle.

---

## 75. How do you configure routing fallback or 404 pages?

**Answer:**

Add a wildcard route with `path: '**'` mapping to a 404 component.

---

## 76. How do you implement reactive form validation?

**Answer:**

Use built-in validators or custom ones on form controls.

---

## 77. What is the purpose of `trackBy` in `*ngFor`?

**Answer:**

Improves performance by tracking list items by unique IDs during DOM updates.

---

## 78. How do you access route query parameters?

**Answer:**

Use `ActivatedRoute.queryParams` Observable.

---

## 79. How can you handle nested routes?

**Answer:**

Define child routes in the route configuration under `children` property.

---

## 80. What is a singleton service?

**Answer:**

A service provided once in root and shared throughout the app.

---

## 81. How do you detect if the app is running on a server or browser?

**Answer:**

Use Angular’s `isPlatformBrowser()` and `isPlatformServer()`.

---

## 82. What are zones in Angular?

**Answer:**

Execution contexts that Angular uses to detect async activity for change detection.

---

## 83. How do you unsubscribe from Observables?

**Answer:**

Use `unsubscribe()` in `ngOnDestroy()`, or use `takeUntil` or `async` pipe.

---

## 84. What is the difference between Angular Elements and Angular Components?

**Answer:**

Angular Elements are Angular components packaged as Web Components.

---

## 85. How do you internationalize an Angular app?

**Answer:**

Mark translatable text with i18n attribute, extract, translate, and build different locales.

---

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

**Answer:**

Compiling Angular templates during build, not runtime.

---

## 87. What is the difference between AOT and JIT?

**Answer:**

AOT compiles at build time, JIT compiles in the browser.

---

## 88. How do you implement server-side rendering?

**Answer:**

Use Angular Universal.

---

## 89. What is a pure function?

**Answer:**

A function with no side effects that returns the same output for same inputs.

---

## 90. How do you implement authentication in Angular?

**Answer:**

Use route guards, services with tokens, HTTP interceptors for headers.

---

## 91. What is a pipe transform function?

**Answer:**

The `transform()` method inside a pipe class that converts input to output.

---

## 92. How do you style Angular components?

**Answer:**

Using component-scoped CSS or global styles.

---

## 93. What is encapsulation in Angular?

**Answer:**

Controls how styles are scoped: Emulated, None, Shadow DOM.

---

## 94. What is the default view encapsulation?

**Answer:**

`ViewEncapsulation.Emulated` - emulates Shadow DOM behavior using scoped CSS.

---

## 95. How to use template reference variables?

**Answer:**

Assign a variable in template to access DOM elements or components.

```html
```

```
<input #myInput />
<button (click)="doSomething(myInput.value)"></button>
```

---

## 96. How do you handle child routes in Angular?

**Answer:**

Define nested routes in the `children` array of a route.

---

## 97. What is the purpose of the `Renderer2`?

**Answer:**

For manipulating DOM elements in a platform-independent way.

---

## 98. How do you create and use a shared module?

**Answer:**

Create a module exporting common components/directives/pipes for reuse.

---

## 99. What are Angular animations?

**Answer:**

Use the `@angular/animations` API to define and trigger CSS or JS animations.

---

## 100. How does Angular CLI help development?

**Answer:**

Automates tasks like scaffolding, building, testing, and deploying Angular apps.

---

## 101. What is `NgZone` and why is it important?

**Answer:**

`NgZone` helps Angular detect async operations and run change detection.

---

## 102. How do you optimize Angular bundle size?

**Answer:**

Use lazy loading, AOT, tree shaking, and differential loading.

## 103. What are the different types of directives?

**Answer:**

Component directives, structural directives, attribute directives.

## 104. How to pass data between routes?

**Answer:**

Use route parameters or query parameters.

## 105. How do you use `ngSwitch`?

**Answer:**

Switch between views based on an expression.

```html
<div [ngSwitch]="value">
  <div *ngSwitchCase="'A'">A</div>
  <div *ngSwitchDefault>Default</div>
</div>
```

## 106. What is lazy loading preload strategy?

**Answer:**

Controls how lazy modules are preloaded, e.g., `PreloadAllModules`.

## 107. How do you test Angular services?

**Answer:**

Use Jasmine to inject service and test methods.

---

## 108. How to use ViewEncapsulation.None?

**Answer:**

Disables style encapsulation; styles are global.

---

## 109. What is the purpose of Angular Guards?

**Answer:**

Control access to routes.

---

## 110. How to upgrade AngularJS app to Angular?

**Answer:**

Use `ngUpgrade` module for hybrid apps.

---

If you'd like, I can keep going with more questions or dive deeper into specific topics like forms, RxJS, routing, performance, testing, or Angular Material. Just let me know!

keep going with more questions

Absolutely! Here are **questions 111–160** with detailed answers to deepen your Angular mastery:

---

## 111. What is the purpose of the `RendererFactory2` service?

**Answer:**

`RendererFactory2` creates instances of `Renderer2`. It's used internally by Angular for DOM manipulation in a platform-agnostic way, helping with server-side rendering or web workers.

---

## 112. Explain Angular's `NgForOf` directive.

**Answer:**

`NgForOf` is the underlying directive for `*ngFor`. It renders a template for each item in a list and tracks changes for efficient DOM updates.

---

## 113. How can you conditionally apply CSS classes in Angular?

**Answer:**

Using `[ngClass]` directive with an object, array, or string.

```html
<div [ngClass]="{'active': isActive, 'disabled': isDisabled}"></div>
```

---

## 114. How do you detect and handle user input in Angular forms?

**Answer:**

Use reactive forms or template-driven forms with event binding or form control value changes to handle inputs and validations.

---

## 115. What is the purpose of `HttpParams`?

**Answer:**

`HttpParams` helps construct URL parameters for HTTP requests in a type-safe and immutable way.

---

## 116. How do you configure global error handling?

**Answer:**

Create a class implementing `ErrorHandler` and provide it in the module to catch unhandled errors.

---

## 117. What are Angular Zones and how do they work?

**Answer:**

Zones are execution contexts that intercept async operations (like promises, setTimeout) to trigger Angular’s change detection automatically.

---

## 118. What is the difference between `Subject`, `BehaviorSubject`, `ReplaySubject`, and `AsyncSubject`?

**Answer:**

- `Subject`: Emits to subscribers after subscription.
- `BehaviorSubject`: Emits the latest value immediately upon subscription.
- `ReplaySubject`: Emits a specified number of previous values to new subscribers.
- `AsyncSubject`: Emits the last value on completion.

---

## 119. How do you implement route reuse strategy?

**Answer:**
Implement `RouteReuseStrategy` to control when routes/components are destroyed or reused, useful for caching views.

---

## 120. How does Angular’s Dependency Injection hierarchy work?

**Answer:**
Injectors are hierarchical: root injector provides singleton services; child injectors can override with scoped instances.

---

## 121. How to handle dynamic forms?

**Answer:**
Build forms programmatically using `FormGroup` and `FormArray` to add controls dynamically.

---

## 122. What is a structural directive?

**Answer:**
A directive that changes DOM layout by adding/removing elements, e.g., `*ngIf`, `*ngFor`.

---

## 123. How do you implement guards for child routes?

**Answer:**

Use `canActivateChild` guard interface to control access to child routes.

---

## 124. How do you secure an Angular app?

**Answer:**

Implement authentication, authorization, use HTTPS, sanitize inputs, avoid eval, and use Angular’s security best practices.

---

## 125. What is the `Renderer2` class and why is it important?

**Answer:**

It provides an abstraction over direct DOM manipulation for platform-independent rendering.

---

## 126. How do you detect changes manually?

**Answer:**

Inject `ChangeDetectorRef` and call `detectChanges()` or `markForCheck()` to trigger change detection manually.

---

## 127. What is the difference between `markForCheck` and `detectChanges`?

**Answer:**

`markForCheck()` marks component for checking in next cycle (used with `OnPush`). `detectChanges()` immediately runs change detection on the component and its children.

---

## 128. What is the purpose of the `HttpClientModule`?

**Answer:**

Provides the `HttpClient` service to make HTTP requests.

---

## 129. How do you cancel HTTP requests?

**Answer:**

Use RxJS `takeUntil` or `unsubscribe()` on the Observable returned by `HttpClient`.

---

## 130. What is the difference between `@ViewChild` and `@ContentChild`?

**Answer:**

`@ViewChild` accesses template DOM elements or child components.

`@ContentChild` accesses projected content inside `ng-content`.

---

## 131. How do you pass data between parent and child components?

**Answer:**

Using `@Input()` and `@Output()` with event emitters.

---

## 132. What are observables in Angular?

**Answer:**

Streams of asynchronous data that Angular uses extensively for events and HTTP.

---

## 133. What is a snapshot in Angular routing?

**Answer:**

A snapshot provides the state of the route at a specific moment, used to access route parameters synchronously.

---

## 134. What are Angular lifecycle hooks? Name a few.

**Answer:**

Methods called at specific points of a component's lifecycle: `ngOnInit()`, `ngOnChanges()`, `ngOnDestroy()`, `ngDoCheck()`, etc.

---

## 135. How do you unsubscribe from an observable?

**Answer:**

Call `.unsubscribe()` in `ngOnDestroy()` or use operators like `takeUntil` or the `async` pipe.

---

## 136. What is Angular Universal?

**Answer:**

A technology for server-side rendering Angular apps for better SEO and faster load times.

---

## 137. How to set up routing in Angular?

**Answer:**

Configure routes in `RouterModule.forRoot()` with path-component mappings and use `<router-outlet>` in templates.

---

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

**Answer:**

Compiles templates and components during build time to improve performance.

---

## 139. What are Angular modules (`NgModule`)?

**Answer:**

Containers for a cohesive block of code — components, services, directives, pipes, and imports.

---

## 140. What are entry components?

**Answer:**

Components loaded dynamically (in Ivy, this is mostly obsolete).

---

## 141. How to optimize Angular app performance?

**Answer:**

Use lazy loading, AOT, OnPush change detection, trackBy in `*ngFor`, tree shaking, and preload strategies.

---

## 142. What is `ng-content`?

**Answer:**

Angular's content projection mechanism for inserting external content inside components.

---

## 143. What are the different forms of data binding?

**Answer:**

Interpolation, property binding, event binding, and two-way binding.

---

## 144. How do you handle HTTP errors?

**Answer:**

Use RxJS `catchError` operator in HTTP Observables.

---

## 145. What is the difference between JIT and AOT compilation?

**Answer:**

JIT compiles templates in the browser at runtime, AOT compiles at build time.

---

## 146. How does Angular's change detection work?

**Answer:**

Detects changes in data-bound properties and updates the DOM accordingly. It uses zones to know when to run.

---

## 147. What is the purpose of `NgZone`?

**Answer:**

Manages async operations and triggers change detection automatically.

---

## 148. How do you style Angular components?

**Answer:**

With scoped component styles or global stylesheets.

---

## 149. What are Angular animations?

**Answer:**

Angular’s built-in support for CSS and JS-based animations using the `@angular/animations` package.

---

## 150. What is the difference between `@HostListener` and `@HostBinding`?

**Answer:**

`@HostListener` listens to events on the host element.
`@HostBinding` binds a property to a host element attribute or class.

---

## 151. What are `@Pipe` and `PipeTransform`?

**Answer:**

`@Pipe` decorator defines a pipe class that implements `PipeTransform` interface for data transformation in templates.

---

## 152. How do you pass data to routes?

**Answer:**

Using route parameters and query parameters.

---

## 153. What are feature modules?

**Answer:**

Modules dedicated to specific features, lazy-loaded or eagerly-loaded for modularization.

---

## 154. What is the difference between `async` and `defer` in script loading?

**Answer:**

`async` loads and executes scripts asynchronously immediately.
`defer` loads scripts asynchronously but executes after the document is parsed.

---

## 155. What is dependency injection (DI) in Angular?

**Answer:**

A design pattern where a class receives dependencies from external sources rather than creating them.

---

## 156. How do you create a reusable component?

**Answer:**

Make the component generic, use `@Input` and `@Output`, and put it in a shared module.

---

## 157. What is the purpose of `Renderer2`?

**Answer:**

Allows safe DOM manipulation in a platform-agnostic way.

---

## 158. How do you handle third-party JavaScript libraries?

**Answer:**

Install via npm, declare typings if needed, and import/use them in Angular.

---

## 159. How do you debug Angular applications?

**Answer:**

Use browser devtools, Angular Augury extension, and console logs.

---

## 160. What is the purpose of Angular CLI?

**Answer:**

To automate creation, building, testing, and deployment of Angular apps.

---

Want me to continue with 50+ more questions? Or focus on any particular domain like Forms, RxJS, Routing, Performance, or Testing?

continue with 50+ more questions

Absolutely! Here are **questions 161–210** with detailed answers:

---

## 161. What is the difference between `forRoot()` and `forChild()` in Angular modules?

**Answer:**

`forRoot()` is used to configure and provide services that should be singleton and available application-wide. It is called once in the root module.
`forChild()` is used in feature modules to register additional routes without re-providing services.

---

## 162. What is the purpose of `NgModule` `exports` array?

**Answer:**

It specifies the components, directives, and pipes that can be used in templates of other modules that import this module.

---

## 163. How do you implement dynamic component loading?

**Answer:**

Use `ViewContainerRef.createComponent()` to load a component dynamically at runtime.

---

## 164. How does Angular's `async` pipe work internally?

**Answer:**

It subscribes to an Observable or Promise and automatically updates the template with the emitted value, unsubscribing when the component is destroyed.

---

## 165. How can you improve Angular app startup time?

**Answer:**

Use lazy loading, AOT compilation, tree shaking, minimizing polyfills, and differential loading.

---

## 166. What is the difference between `ngOnInit()` and constructor in Angular?

**Answer:**

Constructor initializes class members;
`ngOnInit()` is called after Angular initializes the component’s input properties.

---

## 167. What is a resolver in Angular routing?

**Answer:**

A resolver pre-fetches data before a route is activated, ensuring the component has data immediately.

---

## 168. How do you implement pagination in Angular?

**Answer:**

Use component logic with slice operations or use third-party libraries like ngx-pagination.

---

## 169. How do you pass data between unrelated components?

**Answer:**

Use a shared service with Subjects or BehaviorSubjects for communication.

---

## 170. What are Angular decorators?

**Answer:**

Functions that add metadata to classes, methods, or properties to tell Angular how to process them (e.g., `@Component`, `@NgModule`).

---

## 171. How do you create a custom structural directive?

**Answer:**

Use `@Directive()`, inject `TemplateRef` and `ViewContainerRef`, then conditionally add or remove views.

---

## 172. What is `ChangeDetectionStrategy.OnPush`?

**Answer:**

It optimizes change detection by checking only when inputs change or an event originates from the component.

---

## 173. How does Angular handle forms validation?

**Answer:**

Using built-in validators or custom validators attached to form controls, template-driven or reactive.

---

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

**Answer:**

Mark text with `i18n` attribute, extract translations, and build localized versions.

---

## 175. What is tree shaking?

**Answer:**

A build optimization removing unused code from the final bundle.

---

## 176. How do you implement authentication in Angular?

**Answer:**

Using route guards, token storage, HTTP interceptors to add tokens to requests, and services for login/logout.

---

## 177. What is the purpose of Angular's `HttpInterceptor`?

**Answer:**

To intercept and modify HTTP requests and responses globally (e.g., for adding headers or error handling).

---

## 178. How do you handle animations in Angular?

**Answer:**

Use the `@angular/animations` package with triggers, states, transitions, and animation functions.

---

## 179. How does Angular support Progressive Web Apps (PWA)?

**Answer:**

With Angular Service Worker to cache resources and provide offline support.

---

## 180. What is dependency injection hierarchy?

**Answer:**

Angular maintains a tree of injectors, allowing hierarchical service resolution and scoped services.

---

## 181. How do you share data between modules?

**Answer:**

Through services provided at the root or shared module imports.

---

## 182. What are the main lifecycle hooks in Angular?

**Answer:**

`ngOnChanges`, `ngOnInit`, `ngDoCheck`, `ngAfterContentInit`, `ngAfterViewInit`, `ngOnDestroy`.

---

## 183. What is lazy loading and why is it useful?

**Answer:**

Loading modules only when needed to reduce initial bundle size and improve startup performance.

---

## 184. How can you use Angular Material?

**Answer:**

Install Angular Material package, import desired modules, and use ready-to-use UI components.

---

## 185. How do you create a service with providedIn property?

**Answer:**

Use `@Injectable({ providedIn: 'root' })` to provide the service globally without adding it to providers.

---

## 186. How to test components in Angular?

**Answer:**

Use TestBed to configure testing module, create component fixture, and write assertions.

---

## 187. What is the purpose of `NgZone`?

**Answer:**

To manage async operations and control when Angular should run change detection.

---

## 188. How can you improve change detection performance?

**Answer:**

Use `OnPush` strategy, immutable data, and reduce bindings.

---

## 189. What is Angular Ivy?

**Answer:**

Angular's new rendering and compilation engine improving build size and speed.

---

## 190. How do you handle multiple async calls?

**Answer:**

Use RxJS operators like `forkJoin`, `combineLatest`, or `zip`.

---

## 191. How do you handle nested routes?

**Answer:**

Define child routes inside a route's `children` array.

---

## 192. What are pipes and how do you create custom pipes?

**Answer:**

Pipes transform output in templates. Create by implementing `PipeTransform` and decorating with `@Pipe`.

---

## 193. How does Angular's router reuse strategy work?

**Answer:**

Allows controlling if components should be reused or recreated on route changes.

---

## 194. How to unsubscribe from observables automatically?

**Answer:**

Use `async` pipe or operators like `takeUntil`.

---

## 195. What is the use of `HostListener`?

**Answer:**

Listens to events on the host element inside a directive or component.

---

## 196. How can you apply dynamic styles in Angular?

**Answer:**

Using `[ngStyle]` or `[style.property]` bindings.

---

## 197. How do you communicate between parent and child components?

**Answer:**

Via `@Input()` and `@Output()` decorators.

---

## 198. How to add global styles?

**Answer:**

Add CSS in `styles.css` or angular.json under styles array.

---

## 199. What is the difference between `ActivatedRoute` and `Router`?

**Answer:**

`ActivatedRoute` gives info about current route; `Router` controls navigation.

---

## 200. What is an Angular Pipe's `pure` property?

**Answer:**

Indicates if pipe is pure (runs only on input change) or impure (runs every change detection cycle).

---

## 201. How to implement debounce for form inputs?

**Answer:**

Use RxJS operators like `debounceTime` on form value changes Observable.

---

## 202. How do you configure environment variables in Angular?

**Answer:**

Use `environment.ts` files with Angular CLI build configurations.

---

## 203. What is the purpose of `ng-template`?

**Answer:**

Defines a chunk of HTML not rendered until explicitly used.

---

## 204. How to test services?

**Answer:**

Inject service in TestBed and test methods using Jasmine.

---

## 205. How to detect changes manually?

**Answer:**

Use `ChangeDetectorRef.detectChanges()`.

---

## 206. How do you access DOM elements?

**Answer:**

Using `@ViewChild` decorator.

---

## 207. What is Angular’s change detection strategy?

**Answer:**

Controls how often Angular checks for data changes, mainly `Default` and `OnPush`.

---

## 208. How do you implement internationalization?

**Answer:**

Use Angular i18n tools and translation files.

---

## 209. How to upgrade AngularJS apps?

**Answer:**

Use `ngUpgrade` for hybrid apps.

---

## 210. What is `ElementRef` and why should you avoid direct DOM manipulation?

**Answer:**

`ElementRef` wraps native elements. Direct DOM access is discouraged for security and platform independence.

---

Let me know if you want to continue with even more or focus on any specific domain!

continue with even more

Sure! Here are **questions 211–260** with detailed answers:

---

### 211. What is the purpose of `NgModule` `declarations` array?

**Answer:**

It declares the components, directives, and pipes that belong to that module and are visible within it.

---

### 212. How do you implement lazy loading with Angular routing?

**Answer:**

Configure routes with `loadChildren` property pointing to a feature module.

```typescript
{ path: 'feature', loadChildren: () => import('./feature/feature.module').then(m =>
m.FeatureModule) }
```

---

### 213. What is the difference between template-driven and reactive forms?

**Answer:**

Template-driven forms are simple, use directives in templates and are asynchronous. Reactive forms are more robust, defined programmatically, synchronous, and provide better scalability.

---

### 214. How does Angular support Observables?

**Answer:**

Angular uses RxJS library to handle async data streams with operators and Observables.

---

### 215. How can you debounce a button click in Angular?

**Answer:**

Use RxJS `debounceTime` or disable the button temporarily.

---

## 216. What is a forwardRef in Angular?

**Answer:**

Used to refer to references which are not yet defined, helping with circular dependency issues.

---

## 217. How do you detect changes on @Input properties?

**Answer:**

Implement `ngOnChanges(changes: SimpleChanges)` lifecycle hook.

---

## 218. What is the use of `Renderer2` over direct DOM manipulation?

**Answer:**

Provides a safe, platform-independent way to manipulate the DOM without breaking server-side rendering.

---

## 219. What is the purpose of `RouterModule.forRoot()`?

**Answer:**

Initializes the router with root routes and providers.

---

## 220. How do you handle multiple HTTP requests?

**Answer:**

Use RxJS operators like `forkJoin`, `zip`, `combineLatest`.

---

## 221. What is the difference between `Router` and `Location` services?

**Answer:**

`Router` manages navigation and route state; `Location` interacts with the browser URL.

---

## 222. How do you pass data while navigating?

**Answer:**

Using route parameters or state object in `navigate()`.

---

## 223. What are Angular guards and their types?

**Answer:**

Interfaces like `CanActivate`, `CanDeactivate`, `CanLoad` to control route access.

---

## 224. How do you implement route lazy loading with preloading?

**Answer:**

Configure `PreloadAllModules` strategy in the router config.

---

## 225. How do you add HTTP headers globally?

**Answer:**

Implement `HttpInterceptor` and add headers in the intercept method.

---

## 226. What is the difference between `switchMap` and `mergeMap`?

**Answer:**

`switchMap` cancels previous inner Observables; `mergeMap` subscribes to all concurrently.

---

## 227. What is `trackBy` function in `*ngFor`?

**Answer:**

Improves performance by tracking items by a unique identifier to avoid unnecessary

DOM updates.

---

## 228. How do you handle events bubbling in Angular?

**Answer:**

Use `$event.stopPropagation()` or `@HostListener` to prevent bubbling.

---

## 229. What is the difference between `ContentChild` and `ViewChild`?

**Answer:**

`ContentChild` gets projected content inside `<ng-content>`, `ViewChild` accesses component’s own template.

---

## 230. How do you create custom pipes with parameters?

**Answer:**

Add parameters to the `transform()` method after the value.

```typescript
transform(value: string, arg1: any): string {
  // logic
}
```

---

## 231. What are the advantages of reactive forms?

**Answer:**

Immutable data model, synchronous updates, easy validation, and scalable.

---

## 232. How do you perform validation in reactive forms?

**Answer:**

Use built-in validators or custom validator functions passed to form controls.

---

## 233. What is `forwardRef` usage scenario?

**Answer:**

Used in DI when referencing classes not yet defined, preventing circular dependency errors.

---

## 234. How do you test Angular pipes?

**Answer:**

Create pipe instance and call transform method with test data.

---

## 235. How does Angular handle asynchronous validators?

**Answer:**

Validators that return Observables or Promises to perform async checks.

---

## 236. What is a Subject in RxJS?

**Answer:**

An Observable and Observer that can multicast values to multiple subscribers.

---

## 237. How do you implement HTTP caching?

**Answer:**

Use interceptors to cache responses or leverage browser caching.

---

## 238. How do you apply styles conditionally?

**Answer:**

Use `[ngClass]` or `[ngStyle]` with expressions.

---

## 239. What is the use of `ElementRef`?

**Answer:**

Wraps native DOM element; direct manipulation discouraged.

---

## 240. How do you run code before Angular app starts?

**Answer:**

Use `APP_INITIALIZER` provider with a factory function.

---

## 241. What is the difference between `ActivatedRoute` and `RouterState`?

**Answer:**

`ActivatedRoute` is info about the current route; `RouterState` is the full router state tree.

---

## 242. How do you manage state in Angular?

**Answer:**

Use services, BehaviorSubject, or state management libraries like NgRx.

---

## 243. What is the difference between `NgIf` and CSS visibility?

**Answer:**

`NgIf` adds/removes elements; CSS visibility only hides without removing.

---

## 244. How do you handle nested forms?

**Answer:**

Use nested `FormGroup` and `FormArray` inside reactive forms.

---

## 245. How do you use `ContentProjection`?

**Answer:**

Insert external content into component using `<ng-content>`.

---

## 246. How do you improve Angular app SEO?

**Answer:**

Use Angular Universal for server-side rendering.

---

## 247. What is a BehaviorSubject?

**Answer:**

A Subject that stores the last emitted value and emits it immediately to new subscribers.

---

## 248. How do you implement multiple language support?

**Answer:**

Use Angular i18n tools or libraries like ngx-translate.

---

## 249. What is the purpose of `ChangeDetectorRef`?

**Answer:**

Manually trigger or control change detection.

---

## 250. How do you handle errors globally?

**Answer:**

Implement `ErrorHandler` interface.

---

## 251. What is the difference between `templateUrl` and `template`?

**Answer:**

`templateUrl` loads external HTML file, `template` defines inline HTML.

---

## 252. How does Angular detect changes when using `OnPush`?

**Answer:**

Only when input references change or events originate inside the component.

---

## 253. How to access route parameters synchronously?

**Answer:**

Use `ActivatedRoute.snapshot.params`.

---

## 254. How to navigate programmatically?

**Answer:**

Use `Router.navigate()` or `Router.navigateByUrl()`.

---

## 255. How do you implement child routes?

**Answer:**

Define routes inside `children` property of a route.

---

## 256. How do you prevent memory leaks?

**Answer:**

Unsubscribe from Observables, detach event listeners, use `async` pipe.

---

## 257. How do you mock HTTP requests in tests?

**Answer:**

Use `HttpTestingController` with `HttpClientTestingModule`.

---

## 258. What is the purpose of `entryComponents`?

**Answer:**

Components loaded dynamically (mostly obsolete with Ivy).

---

## 259. How to communicate between distant components?

**Answer:**

Use shared service with RxJS subjects.

---

## 260. How do you implement animations?

**Answer:**
Define animations in `@Component` metadata and trigger with Angular animation API.

---

Would you like me to continue further or focus on specific Angular topics next?

continue further

Here you go — **questions 261 to 310** with detailed answers:

---

## 261. What is `NgZone` and how is it useful?

**Answer:**
`NgZone` allows Angular to detect asynchronous tasks and trigger change detection. It helps to optimize performance by running code outside Angular's zone when you don't want change detection to run.

---

## 262. How do you implement route guards to protect routes?

**Answer:**
Create a service implementing guard interfaces (`CanActivate`, `CanDeactivate`, etc.), return `true` or `false` or an Observable/Promise, and configure it in routing.

---

## 263. What are the advantages of Angular CLI?

**Answer:**
Automates project setup, development server, building, testing, and deployment with standard configurations.

---

## 264. How do you create and use custom validators?

**Answer:**
Create a function returning `ValidationErrors | null` and assign it to a form

control.

---

## 265. What is a `FormArray` in reactive forms?

**Answer:**

An array of form controls or groups, useful for dynamic forms with variable controls.

---

## 266. How does Angular handle security?

**Answer:**

Sanitizes untrusted content, uses Angular’s DOM abstraction, prevents XSS by default, encourages safe coding practices.

---

## 267. How to lazy load modules with route preloading?

**Answer:**

Use `PreloadAllModules` strategy in RouterModule options.

---

## 268. What is the purpose of `ViewEncapsulation`?

**Answer:**

Controls how styles are scoped to components (Emulated, None, ShadowDom).

---

## 269. How do you handle server-side rendering?

**Answer:**

Use Angular Universal to render app on server for SEO and faster loads.

---

## 270. What is the `async` pipe and why use it?

**Answer:**

Automatically subscribes/unsubscribes to Observables or Promises, simplifying async data handling in templates.

---

## 271. How do you optimize Angular change detection?

**Answer:**

Use `OnPush`, immutable data, manual change detection, and `trackBy` in lists.

---

## 272. What is a `Subject` and how does it differ from an `Observable`?

**Answer:**

`Subject` is both Observable and Observer; can multicast values to multiple subscribers. `Observable` is unicast by default.

---

## 273. How to perform unit testing on services?

**Answer:**

Use Angular TestBed to inject services and test methods with Jasmine/Karma.

---

## 274. How do you use `ng-content` with select attribute?

**Answer:**

Allows multiple slots for projecting specific content with CSS selectors.

---

## 275. What is the purpose of `APP_INITIALIZER`?

**Answer:**

Runs functions before app initialization, useful for loading config data.

---

## 276. How to handle nested reactive forms?

**Answer:**

Use nested `FormGroup` and `FormArray` structures.

---

## 277. What is the difference between pure and impure pipes?

**Answer:**

Pure pipes execute only when input changes; impure execute every change detection cycle.

---

## 278. How does Angular’s router handle lazy loading?

**Answer:**

Loads modules only when navigating to their route via `loadChildren`.

---

## 279. What is the difference between `BehaviorSubject` and `ReplaySubject`?

**Answer:**

`BehaviorSubject` emits last value immediately to new subscribers.
`ReplaySubject` emits a specified number of past values.

---

## 280. How to create animations with keyframes?

**Answer:**

Use Angular animation keyframes to define intermediate animation states.

---

## 281. What is the purpose of the `providers` array in `NgModule`?

**Answer:**

Registers services at the module level for Dependency Injection.

---

## 282. How do you create a singleton service?

**Answer:**

Use `@Injectable({ providedIn: 'root' })` or provide service in root module.

---

## 283. How to use Angular Material components?

**Answer:**

Import Material modules and use pre-built UI components with Material styles.

---

## 284. What is the role of `ChangeDetectorRef`?

**Answer:**

Allows manual control over when change detection runs.

---

## 285. How to handle form value changes?

**Answer:**

Subscribe to `valueChanges` Observable on form controls or groups.

---

## 286. What is a directive in Angular?

**Answer:**

Class that modifies behavior or appearance of DOM elements.

---

## 287. How to use lifecycle hook `ngAfterViewInit()`?

**Answer:**

Called after component’s view (and child views) initialized, good for DOM-dependent logic.

---

## 288. How to implement route parameters?

**Answer:**

Define in route path and access via `ActivatedRoute.params`.

---

## 289. How to create reusable components?

**Answer:**

Make configurable via `@Input()`, expose events via `@Output()`, and avoid tightly coupling.

---

## 290. What is the difference between `Observable` and `Promise`?

**Answer:**

`Observable` supports multiple values over time; `Promise` resolves once.

---

## 291. How to use pipes to transform data in templates?

**Answer:**

Apply pipes using `|` syntax, e.g., `{{ dateValue | date }}`.

---

## 292. How do you implement route animations?

**Answer:**

Define animations tied to router outlet’s state changes.

---

## 293. What is the purpose of `async`/`await` in Angular?

**Answer:**

Simplifies asynchronous code, works well with Promises.

---

## 294. How to use HTTP interceptors?

**Answer:**

Implement `HttpInterceptor` interface to modify requests/responses globally.

---

## 295. What are the benefits of Angular Ivy?

**Answer:**

Smaller bundles, faster builds, better debugging.

---

## 296. How to implement error handling in HTTP requests?

**Answer:**

Use RxJS `catchError` and display user-friendly messages.

## 297. How to use the `ng-template` tag?

**Answer:**

Defines template fragments not rendered until used.

## 298. What is the purpose of the `entryComponents` array?

**Answer:**

Specifies components to compile for dynamic loading (mostly obsolete now).

## 299. How to test asynchronous code?

**Answer:**

Use `fakeAsync`, `async`, and `tick()` utilities in tests.

## 300. What is the difference between structural and attribute directives?

**Answer:**

Structural changes DOM structure (`*ngIf`), attribute changes appearance/behavior (`[ngClass]`).

## 301. How to use lifecycle hook `ngOnDestroy()`?

**Answer:**

Cleanup resources, unsubscribe from Observables.

## 302. How to implement guards with Observables?

**Answer:**

Return an Observable<boolean> from guard methods.

## 303. How to share data between sibling components?

**Answer:**

Use a shared service with Subjects.

---

## 304. What is Angular’s event binding syntax?

**Answer:**

Use parentheses, e.g., `(click)="handleClick()"`.

---

## 305. How to bind to an input property?

**Answer:**

Use square brackets, e.g., `[property]="value"`.

---

## 306. How to implement two-way binding?

**Answer:**

Use `[(ngModel)]` or `@Input()` + `@Output()` pattern.

---

## 307. What are Angular animations’ triggers?

**Answer:**

Named animations that listen for state changes to run animations.

---

## 308. How to debounce user input?

**Answer:**

Use RxJS operators like `debounceTime` on form control value changes.

---

## 309. How to implement dynamic routes?

**Answer:**

Use parameters in route paths and access via `ActivatedRoute`.

---

### 310. What is Angular Universal?

**Answer:**

A server-side rendering technology to improve SEO and performance.

---

If you want, I can keep going, or focus on testing, RxJS, performance optimization, or Angular forms next!

dive deeper into specific topics like forms, RxJS, routing, performance, testing, or Angular Material. Just let me know!

Great! Let's dive deeper into one of those. Which topic would you like to explore in detail next?

- **Forms** (template-driven, reactive, validation, dynamic forms)
- **RxJS** (observables, operators, subjects, multicasting)
- **Routing** (guards, lazy loading, nested routes, route params)
- **Performance** (change detection, lazy loading, preloading strategies, optimization)
- **Testing** (unit testing, integration testing, mocking HTTP, async tests)
- **Angular Material** (components, theming, responsiveness, accessibility)

Just pick one or more, and I'll provide a detailed, topic-focused set of questions & answers or explanations!

Forms (template-driven, reactive, validation, dynamic forms)

Awesome! Let's dive deep into **Angular Forms** covering template-driven, reactive, validation, and dynamic forms with detailed questions and answers:

---

## Angular Forms Deep Dive

### 1. What are the two types of Angular forms?

**Answer:**

Angular provides two form handling techniques:

- **Template-driven forms**: Use directives in the template to create and manage forms. Suitable for simple forms.
- **Reactive forms**: Created programmatically in the component class using `FormControl`, `FormGroup`, and `FormArray`. Better for complex forms and scalability.

---

## 2. How do template-driven forms work in Angular?

**Answer:**

They rely on Angular directives like `ngModel`, `ngForm`, and `ngSubmit` in the HTML template. Angular automatically creates `FormControl` instances under the hood. Validation and tracking are mostly done declaratively in the template.

---

## 3. How to create a simple template-driven form?

```html
<form #form="ngForm" (ngSubmit)="submitForm(form)">
  <input name="email" ngModel required email>
  <button type="submit" [disabled]="form.invalid">Submit</button>
</form>
```

In component:

```typescript
submitForm(form: NgForm) {
  console.log(form.value);
}
```

---

## 4. What are the main directives used in template-driven forms?

**Answer:**

- `ngForm`: Tracks the form as a whole.

- `ngModel`: Binds form controls.
- `ngModelGroup`: Groups controls.
- Validation directives like `required`, `minlength`, `pattern`.

---

### 5. How do reactive forms differ from template-driven forms?

**Answer:**
Reactive forms are more explicit, built and managed in the component class. They offer synchronous access to the form data model, support dynamic changes, and provide more predictable validations.

---

### 6. How to create a basic reactive form?

```typescript
form = new FormGroup({
  name: new FormControl('', [Validators.required]),
  email: new FormControl('', [Validators.required, Validators.email])
});
```

In template:

```html
<form [formGroup]="form" (ngSubmit)="submit()">
  <input formControlName="name">
  <input formControlName="email">
  <button type="submit" [disabled]="form.invalid">Submit</button>
</form>
```

---

### 7. How do you add validators in reactive forms?

**Answer:**
Validators can be added to form controls when creating them, either built-in like `Validators.required` or custom validators.

```typescript
```

```
new FormControl('', [Validators.required, Validators.minLength(3)])
```

---

## 8. How do you create custom validators?

```typescript
function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): ValidationErrors | null => {
    const forbidden = nameRe.test(control.value);
    return forbidden ? { forbiddenName: { value: control.value } } : null;
  };
}
```

Usage:

```typescript
new FormControl('', [forbiddenNameValidator(/bob/i)])
```

---

## 9. How do you check form control status and errors?

**Answer:**

You can use properties like `.valid`, `.invalid`, `.touched`, `.dirty`, and `.errors` on form controls.

```typescript
if (this.form.controls['email'].invalid && this.form.controls['email'].touched) {
  // show error
}
```

---

## 10. What is a `FormArray` and when do you use it?

**Answer:**

`FormArray` is a dynamic collection of form controls, useful for forms where the number of controls changes at runtime (e.g., adding/removing phone numbers).

---

## 11. How to dynamically add form controls in reactive forms?

```typescript
this.formArray.push(new FormControl(''));
```

Template:

```html
<div formArrayName="phones">
  <div *ngFor="let phone of formArray.controls; let i=index">
    <input [formControlName]="i">
  </div>
</div>
<button (click)="addPhone()">Add Phone</button>
```

---

## 12. How to use `setValue` and `patchValue`?

**Answer:**

- `setValue` requires all controls to be specified.
- `patchValue` updates only the specified controls.

---

## 13. How to listen to form value changes?

```typescript
this.form.valueChanges.subscribe(value => {
  console.log(value);
});
```

---

## 14. How do you reset a form?

**Answer:**

Use `.reset()` method to clear the form and reset status.

```typescript
```

```
this.form.reset();
```

---

## 15. How to disable/enable controls programmatically?

```typescript
 this.form.controls['email'].disable();
this.form.controls['email'].enable();
```

---

## 16. How do you handle validation messages?

**Answer:**

Create reusable methods or pipes to display messages based on `.errors` object of controls.

---

## 17. What is the role of `NgForm` in template-driven forms?

**Answer:**

It represents the form and tracks overall form state like validity and submission status.

---

## 18. How do you submit a form and access values?

**Answer:**

In template-driven forms, use the form reference; in reactive forms, access `.value` on the `FormGroup`.

---

## 19. How can you disable a form submit button when invalid?

```html
<button [disabled]="form.invalid">Submit</button>
```

---

## 20. How to implement cross-field validation?

**Answer:**

Use custom validator on a `FormGroup` that checks multiple controls.

```typescript
function passwordMatchValidator(group: FormGroup) {
  const pass = group.get('password').value;
  const confirm = group.get('confirmPassword').value;
  return pass === confirm ? null : { mismatch: true };
}
```

---

## 21. How to set async validators?

**Answer:**

Pass async validators as third argument in `FormControl` constructor.

```typescript
new FormControl('', null, asyncValidatorFn)
```

---

## 22. How to disable a control conditionally based on other control's value?

**Answer:**

Subscribe to valueChanges of one control and call `.disable()` or `.enable()` on another.

---

## 23. How to prepopulate a form with data?

```typescript
this.form.setValue({ name: 'John', email: 'john@example.com' });
```

---

## 24. What is the difference between `NgModel` and `FormControl`?

**Answer:**

`NgModel` is for template-driven forms, `FormControl` for reactive forms.

---

## 25. How do you track touched and dirty states?

**Answer:**

Angular automatically tracks these; you can check with `.touched` and `.dirty` properties.

---

## 26. How to build complex nested forms?

**Answer:**

Use nested `FormGroup`s and `FormArray`s to represent hierarchical data.

---

## 27. How do you implement dynamic form fields with validation?

**Answer:**

Add controls dynamically with validators inside a `FormArray`.

---

## 28. How to integrate third-party UI libraries with Angular forms?

**Answer:**

Use `ControlValueAccessor` to create form-compatible custom controls.

---

## 29. How to reset form with specific values?

```typescript
this.form.reset({ name: 'Default', email: '' });
```

---

## 30. How to implement conditional validation?

**Answer:**

Use reactive form valueChanges to add/remove validators dynamically.

---

Would you like me to generate more questions on Angular forms or provide code examples/tutorials on specific areas like dynamic forms, validation, or integration?

RxJS (observables, operators, subjects, multicasting)

Perfect! Let's dive deep into **RxJS** in Angular, covering Observables, Operators, Subjects, and Multicasting with detailed questions and answers:

---

## RxJS Deep Dive for Angular

### 1. What is an Observable in RxJS?

**Answer:**
An Observable represents a stream of data or events that can be observed asynchronously over time. It can emit zero or more values, then completes or errors.

---

### 2. How do you create an Observable?

**Answer:**
Using the `new Observable()` constructor or creation functions like `of()`, `from()`, `interval()`, and `fromEvent()`.

```typescript
const obs = new Observable(observer => {
  observer.next(1);
  observer.complete();
});
```

---

### 3. How do you subscribe to an Observable?

```typescript
obs.subscribe({
  next: val => console.log(val),
  error: err => console.error(err),
  complete: () => console.log('Completed')
});
```

---

### 4. What is an Observer?

**Answer:**

An object with callbacks (`next`, `error`, `complete`) that receive notifications from an Observable.

---

### 5. What is the difference between Cold and Hot Observables?

**Answer:**

- **Cold**: Each subscriber receives independent execution of Observable (e.g., HTTP requests).
- **Hot**: Shared execution with data emitted regardless of subscribers (e.g., mouse events).

---

### 6. What are Operators in RxJS?

**Answer:**

Functions that enable transformation, filtering, combination, and error handling of Observable streams.

---

### 7. What is the difference between `map` and `switchMap`?

**Answer:**

- `map`: Transforms emitted values.
- `switchMap`: Maps to a new Observable and unsubscribes from previous, useful for cancelling outdated requests.

---

### 8. What does `mergeMap` do?

**Answer:**

Projects each source value to an Observable and merges all resulting Observables concurrently.

---

### 9. How to use `filter` operator?

```typescript
obs.pipe(
  filter(val => val % 2 === 0)
).subscribe(console.log);
```

Filters emitted values based on a predicate.

---

### 10. What is a Subject in RxJS?

**Answer:**
A Subject is both an Observable and Observer. It can multicast to multiple subscribers.

---

### 11. What are different types of Subjects?

**Answer:**

- `Subject`: No initial value, multicast.
- `BehaviorSubject`: Has an initial value, emits last value on new subscriptions.
- `ReplaySubject`: Replays specified number of past values to new subscribers.
- `AsyncSubject`: Emits only last value on completion.

---

### 12. How is a BehaviorSubject used?

**Answer:**
To hold and emit the current value, often used for state management.

```typescript
const subject = new BehaviorSubject(0);
subject.next(1);
subject.subscribe(val => console.log(val)); // Emits 1 immediately
```

---

### 13. What is multicasting?

**Answer:**

Sharing a single Observable execution among multiple subscribers using Subjects.

---

### 14. How do you convert a Cold Observable to Hot?

**Answer:**

By using `publish()` or `share()` operators with a Subject.

---

### 15. What is `debounceTime` operator?

**Answer:**

Delays values emitted by source Observable until a specified time has passed without another emission.

Useful for limiting rapid user input events.

---

### 16. What does the `take` operator do?

**Answer:**

Emits only the first N values from the source Observable and then completes.

---

### 17. How do you handle errors in RxJS?

**Answer:**

Using `catchError` operator to catch and recover from errors.

---

### 18. What is `combineLatest`?

**Answer:**

Combines the latest values from multiple Observables into one Observable.

---

### 19. What does the `forkJoin` operator do?

**Answer:**

Waits for all Observables to complete, then emits the last values as an array.

---

### 20. How do you unsubscribe from an Observable?

**Answer:**

Call `.unsubscribe()` on the Subscription object or use `takeUntil` operator.

---

### 21. How does Angular's `async` pipe work with Observables?

**Answer:**

It subscribes automatically to an Observable and unsubscribes on component destruction.

---

### 22. How do you use the `switchMap` operator for HTTP requests?

**Answer:**

It cancels previous HTTP calls if a new one is made, avoiding race conditions.

---

### 23. What is the purpose of `Subject.next()`?

**Answer:**

To emit a new value to all subscribers.

---

### 24. How can you create a custom operator?

**Answer:**

By writing a function returning an Observable that uses `pipe` and other operators internally.

---

### 25. What is the difference between `first` and `take(1)`?

**Answer:**

`first` emits the first value that meets a condition, throws if none; `take(1)` emits the first value regardless.

---

### 26. How to implement debounce on form input?

**Answer:**

Use `valueChanges.pipe(debounceTime(300))` to delay processing.

---

### 27. How to share an Observable among multiple subscribers?

**Answer:**

Use `shareReplay()` or `share()` to multicast and replay emitted values.

---

### 28. What is the use of `withLatestFrom`?

**Answer:**

Combines source Observable with latest value from another Observable on emission.

---

### 29. What is `distinctUntilChanged`?

**Answer:**

Prevents emitting duplicate consecutive values.

---

### 30. How do you cancel an ongoing HTTP request?

**Answer:**

Use `switchMap` or unsubscribe manually from the Observable.

---

Want me to keep going with even more RxJS questions or deep-dive into coding examples and use cases?

Routing (guards, lazy loading, nested routes, route params)

Awesome! Let’s dive deep into **Angular Routing** — covering guards, lazy loading, nested routes, and route parameters with detailed questions and answers:

---

## Angular Routing Deep Dive

### 1. What is Angular Router?

**Answer:**
Angular Router enables navigation from one view to another in a single-page application. It interprets URL paths and loads corresponding components.

---

### 2. How do you define routes in Angular?

**Answer:**
Routes are defined as an array of objects with `path` and `component` (or `loadChildren`) properties, typically inside `RouterModule.forRoot()` or `forChild()`.

typescript

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

---

### 3. What is a route guard?

**Answer:**
Route guards are services that control access to routes. They implement interfaces like `CanActivate`, `CanDeactivate`, `CanLoad`, `Resolve`, or `CanActivateChild`.

---

### 4. What is `CanActivate` guard?

**Answer:**
Determines if a route can be activated. Returns `true` / `false` or an

Observable/Promise resolving to boolean.

---

## 5. How do you implement a `CanActivate` guard?

```typescript
@Injectable({ providedIn: 'root' })
export class AuthGuard implements CanActivate {
  constructor(private authService: AuthService, private router: Router) {}

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): boolean {
    if (this.authService.isLoggedIn()) return true;
    this.router.navigate(['/login']);
    return false;
  }
}
```

---

## 6. What is a `CanDeactivate` guard used for?

**Answer:**

Prevents navigation away from a route if there are unsaved changes, asking the user for confirmation.

---

## 7. How do you protect routes with guards?

**Answer:**

Add guard service(s) to the `canActivate` or other guard properties in route definitions.

```typescript
{ path: 'admin', component: AdminComponent, canActivate: [AuthGuard] }
```

---

## 8. What is lazy loading in Angular routing?

**Answer:**

Loading feature modules only when the user navigates to their route, reducing initial bundle size.

---

## 9. How do you configure lazy loading?

```typescript
const routes: Routes = [
  { path: 'admin', loadChildren: () => import('./admin/admin.module').then(m =>
m.AdminModule) }
];
```

---

## 10. What is the difference between eager and lazy loading?

**Answer:**

Eager loading loads all modules at app startup; lazy loading loads on demand.

---

## 11. How do you implement nested routes?

**Answer:**

Use `children` property in route definitions.

```typescript
{
  path: 'products',
  component: ProductsComponent,
  children: [
    { path: 'details/:id', component: ProductDetailsComponent }
  ]
}
```

---

## 12. How do you access route parameters?

**Answer:**

Inject `ActivatedRoute` and access `paramMap` or `params`.

```typescript
this.route.paramMap.subscribe(params => {
  const id = params.get('id');
});
```

---

### 13. What are query parameters?

**Answer:**

Optional key-value pairs in URL after `?` used for additional data.

---

### 14. How do you access query parameters?

```typescript
this.route.queryParamMap.subscribe(params => {
  const filter = params.get('filter');
});
```

---

### 15. How to navigate programmatically?

```typescript
this.router.navigate(['/products', id], { queryParams: { filter: 'new' } });
```

---

### 16. What is a `Resolve` guard?

**Answer:**

Pre-fetches data before activating a route so the component has the data on load.

---

### 17. How do you implement a `Resolve` guard?

typescript

```typescript
@Injectable({ providedIn: 'root' })
export class ProductResolver implements Resolve<Product> {
  constructor(private service: ProductService) {}

  resolve(route: ActivatedRouteSnapshot): Observable<Product> {
    return this.service.getProduct(route.paramMap.get('id'));
  }
}
```

---

## 18. What is `CanLoad` guard?

**Answer:**

Prevents the loading of a lazy-loaded module if the user is unauthorized.

---

## 19. What is `canActivateChild`?

**Answer:**

Guard that controls access to child routes of a route.

---

## 20. How to handle 404 (Not Found) routes?

typescript

```typescript
{ path: '**', component: PageNotFoundComponent }
```

---

## 21. How do you use relative navigation?

typescript

```typescript
this.router.navigate(['../edit'], { relativeTo: this.route });
```

---

## 22. What is `runGuardsAndResolvers` option?

**Answer:**

Controls when guards and resolvers run on re-navigation to the same route.

---

### 23. How do you preload lazy-loaded modules?

**Answer:**

Use `PreloadAllModules` or custom strategies in router config.

```typescript
RouterModule.forRoot(routes, { preloadingStrategy: PreloadAllModules })
```

---

### 24. How to reset navigation history?

**Answer:**

Use `replaceUrl: true` in navigation extras.

---

### 25. What is `routerLinkActive`?

**Answer:**

Directive to add CSS classes when the link's route is active.

---

### 26. How to implement breadcrumbs with routing?

**Answer:**

Use route data and ActivatedRoute snapshot to build breadcrumbs dynamically.

---

### 27. How do you lazy load components without modules?

**Answer:**

Angular 14+ supports standalone components with `loadComponent`.

---

### 28. What is the difference between `ActivatedRoute` and `Router` services?

**Answer:**

`ActivatedRoute` provides info about the current route; `Router` controls navigation.

---

### 29. How to pass data between routes?

**Answer:**

Use route parameters, query parameters, or route `data` property.

---

### 30. How do you configure route reuse strategy?

**Answer:**

Customize Angular's `RouteReuseStrategy` to cache components or routes.

---

Want me to keep going with more routing questions or provide code samples/tutorials?

Performance (change detection, lazy loading, preloading strategies, optimization)

Great choice! Here's a deep dive into **Angular Performance** focusing on change detection, lazy loading, preloading strategies, and general optimization techniques, with detailed questions and answers:

---

## Angular Performance Deep Dive

### 1. What is Angular's change detection mechanism?

**Answer:**

Angular's change detection tracks data changes and updates the DOM accordingly. It runs after events, async tasks, and promises to sync the UI with the data model.

---

### 2. How does the default change detection strategy work?

**Answer:**
Angular uses a tree of components and checks every component every cycle to detect changes. This can be inefficient for large apps.

---

### 3. What is the `OnPush` change detection strategy?

**Answer:**
`OnPush` tells Angular to check a component only when its inputs change by reference or when an event originates inside it. It improves performance by reducing checks.

---

### 4. How do you apply `OnPush` strategy?

```typescript
@Component({
  selector: 'app-my',
  changeDetection: ChangeDetectionStrategy.OnPush,
  templateUrl: './my.component.html'
})
export class MyComponent {}
```

---

### 5. What is immutable data, and why is it important for performance?

**Answer:**
Immutable data means creating new objects instead of mutating existing ones. Angular can detect changes easily by comparing references with immutable data, speeding up `OnPush`.

---

### 6. What is `trackBy` and why use it in `*ngFor`?

**Answer:**
`trackBy` function helps Angular identify items uniquely in lists, preventing unnecessary DOM re-rendering.

html

```
<div *ngFor="let item of items; trackBy: trackById">{{item.name}}</div>
```

---

## 7. How does lazy loading improve app performance?

**Answer:**
Lazy loading loads feature modules only when needed, reducing the initial bundle size and speeding up app startup.

---

## 8. How do you implement lazy loading?

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

---

## 9. What is preloading in Angular routing?

**Answer:**
Preloading loads lazy-loaded modules in the background after app startup to improve subsequent navigation speed.

---

## 10. What preloading strategies does Angular provide?

**Answer:**

- `NoPreloading` (default, no preloading)
- `PreloadAllModules` (preloads all modules)
- Custom preloading strategies implementing `PreloadingStrategy`.

---

## 11. How can you write a custom preloading strategy?

```typescript
```

```
@Injectable()
export class SelectivePreloadingStrategy implements PreloadingStrategy {
 preload(route: Route, load: () => Observable<any>): Observable<any> {
  return route.data && route.data['preload'] ? load() : of(null);
 }
}
```

### 12. What is the purpose of `detach` and `reattach` in `ChangeDetectorRef`?

**Answer:**

They allow manual control to stop and restart change detection on a component to improve performance.

### 13. How to optimize large lists in Angular?

**Answer:**

Use `trackBy`, virtual scrolling (e.g., Angular CDK's `cdk-virtual-scroll-viewport`), and pagination.

### 14. How to minimize bundle size?

**Answer:**

- Use lazy loading
- Avoid importing unnecessary modules
- Use Angular CLI build optimizations (`--prod`)
- Tree shaking
- Use `providedIn: 'root'` for services

### 15. What is Angular CLI's differential loading?

**Answer:**

It creates separate bundles for modern browsers and legacy browsers, improving load speed.

---

### 16. How can you optimize change detection in forms?

**Answer:**
Use reactive forms with `OnPush`, avoid unnecessary subscriptions, and unsubscribe properly.

---

### 17. How to reduce initial load time?

**Answer:**

- Lazy load routes
- Use server-side rendering (Angular Universal)
- Preload modules strategically
- Optimize assets (images, fonts)

---

### 18. What is `ngZone.runOutsideAngular()`?

**Answer:**
Runs code outside Angular’s zone to prevent triggering change detection unnecessarily, useful for performance-intensive tasks.

---

### 19. How to profile Angular app performance?

**Answer:**
Use Chrome DevTools, Angular Profiler, or Lighthouse to identify bottlenecks.

---

### 20. How do you optimize image loading?

**Answer:**
Use lazy loading (`loading="lazy"`), optimize image size, use modern formats like WebP.

---

### 21. How to handle memory leaks?

**Answer:**
Unsubscribe from Observables, remove event listeners, and avoid detached DOM elements.

---

### 22. How does `async` pipe help performance?

**Answer:**
Automatically manages subscription lifecycle, preventing memory leaks and reducing manual code.

---

### 23. How do you cache HTTP requests?

**Answer:**
Implement caching logic in services or HTTP interceptors to prevent unnecessary server calls.

---

### 24. What is change detection tree?

**Answer:**
Angular checks components starting from the root component down the tree.

---

### 25. What are pure and impure pipes in terms of performance?

**Answer:**
Pure pipes execute only on input changes, impure pipes run every change detection cycle and can degrade performance.

---

Want to continue with more questions or practical tips on optimizing Angular apps?

Testing (unit testing, integration testing, mocking HTTP, async tests)

Great! Let's explore **Angular Testing** in detail — covering unit testing, integration testing, HTTP mocking, and async tests with detailed questions and answers:

---

## Angular Testing Deep Dive

### 1. What testing frameworks are commonly used with Angular?

**Answer:**
Angular uses **Jasmine** for writing tests and **Karma** as the test runner by default.

---

### 2. What is a unit test in Angular?

**Answer:**
A unit test verifies the smallest parts of an application, like a component, service, or pipe, in isolation.

---

### 3. How to create a basic unit test for a component?

```typescript
describe('MyComponent', () => {
  let component: MyComponent;
  let fixture: ComponentFixture<MyComponent>;

  beforeEach(() => {
    TestBed.configureTestingModule({
      declarations: [MyComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(MyComponent);
    component = fixture.componentInstance;
  });

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

---

## 4. What is `TestBed` in Angular testing?

**Answer:**

`TestBed` configures and initializes the environment for unit testing components and services.

---

## 5. How do you test component templates?

**Answer:**

Access DOM elements via `fixture.nativeElement` or use `fixture.debugElement` to query elements and verify rendered output.

---

## 6. How do you test service methods?

**Answer:**

Inject the service using `TestBed.inject(ServiceName)` and test methods with various inputs and expected outputs.

---

## 7. What is dependency injection mocking?

**Answer:**

Replacing real dependencies with mock or spy objects in tests to isolate the unit under test.

---

## 8. How do you mock a service?

```typescript
const mockService = {
  getData: jasmine.createSpy('getData').and.returnValue(of('mockData'))
};

TestBed.configureTestingModule({
  providers: [{ provide: RealService, useValue: mockService }]
});
```

---

## 9. What is a spy in Jasmine?

**Answer:**

A spy is a function that tracks calls and arguments, useful for verifying interaction with dependencies.

---

## 10. How to test asynchronous code?

**Answer:**

Use `async()` or `fakeAsync()` utilities from Angular testing.

---

## 11. What is the difference between `async()` and `fakeAsync()`?

**Answer:**

- `async()` wraps a test and waits for asynchronous operations to complete (like Promises).
- `fakeAsync()` uses simulated time with `tick()` to control async flows synchronously.

---

## 12. How to test HTTP calls?

**Answer:**

Use Angular's `HttpClientTestingModule` with `HttpTestingController` to mock HTTP requests.

---

## 13. How to mock HTTP requests in tests?

```typescript
import { HttpTestingController, HttpClientTestingModule } from
'@angular/common/http/testing';

beforeEach(() => {
  TestBed.configureTestingModule({
    imports: [HttpClientTestingModule]
  });
  httpMock = TestBed.inject(HttpTestingController);
});

it('should test HTTP GET', () => {
  service.getData().subscribe(data => {
    expect(data).toEqual(mockData);
  });

  const req = httpMock.expectOne('api/data');
  req.flush(mockData);
});
```

---

## 14. What is integration testing in Angular?

**Answer:**

Testing multiple parts (e.g., components + services) together to verify their interactions.

---

## 15. How to trigger events in tests?

typescript

```typescript
const button = fixture.nativeElement.querySelector('button');
button.click();
fixture.detectChanges();
```

---

## 16. What is `fixture.detectChanges()`?

**Answer:**

Runs change detection to update the component and its template after changes.

---

## 17. How do you test form controls?

**Answer:**

Set values on form controls and check validity and template changes.

---

### 18. How to test routing?

**Answer:**

Use `RouterTestingModule` and `Location` service to simulate navigation and verify routes.

---

### 19. What is a shallow test?

**Answer:**

Testing a component without rendering its child components, using mocks/stubs instead.

---

### 20. How to test lifecycle hooks?

**Answer:**

Call lifecycle methods manually or trigger actions that cause Angular to call them.

---

### 21. How to test output events?

```typescript
 component.someEvent.emit(value);
expect(spy).toHaveBeenCalledWith(value);
```

---

### 22. What is code coverage?

**Answer:**

A metric showing the percentage of code executed during tests, used to assess test quality.

---

### 23. How to test error handling?

**Answer:**

Simulate errors in dependencies or HTTP requests and verify error handling logic.

---

### 24. What are some best practices for Angular testing?

**Answer:**

- Keep tests isolated
- Use descriptive test names
- Mock dependencies
- Avoid testing private methods
- Use test data factories

---

### 25. How do you run tests with coverage report?

```bash
ng test --code-coverage
```

---

Want me to continue with more Angular testing questions, or provide examples of testing async HTTP calls or forms?

give one complete test.spec.ts and explain throughly

Sure! Here's a **complete example** of an Angular component unit test (`.spec.ts`) file with detailed explanations step-by-step.

---

### Example: Testing a Simple Counter Component

**Component:** `counter.component.ts`

```typescript
import { Component } from '@angular/core';

@Component({
  selector: 'app-counter',
  template: `
```

```
    <p>Count: {{ count }}</p>
    <button (click)="increment()">Increment</button>
  `
})
export class CounterComponent {
  count = 0;

  increment() {
    this.count++;
  }
}
```

---

### Test File: `counter.component.spec.ts`

```typescript
import { ComponentFixture, TestBed } from '@angular/core/testing';
import { CounterComponent } from './counter.component';

describe('CounterComponent', () => {
  let component: CounterComponent;
  let fixture: ComponentFixture<CounterComponent>;
  let compiled: HTMLElement;

  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [CounterComponent]
    }).compileComponents();

    fixture = TestBed.createComponent(CounterComponent);
    component = fixture.componentInstance;
    compiled = fixture.nativeElement; // The DOM element of the component
    fixture.detectChanges(); // Trigger initial data binding
  });

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

  it('should display the initial count', () => {
    const p = compiled.querySelector('p');
    expect(p?.textContent).toContain('Count: 0');
  });

  it('should increment count on button click', () => {
    const button = compiled.querySelector('button');
    button?.click(); // Simulate click
    fixture.detectChanges(); // Update DOM after the click

    const p = compiled.querySelector('p');
    expect(component.count).toBe(1); // Check component state
    expect(p?.textContent).toContain('Count: 1'); // Check DOM update
  });

  it('should increment count multiple times', () => {
    component.increment();
    component.increment();
    fixture.detectChanges();

    const p = compiled.querySelector('p');
    expect(component.count).toBe(2);
    expect(p?.textContent).toContain('Count: 2');
  });
});
```

---

## Explanation:

1. **Imports:**
    - `ComponentFixture` helps manage component testing instance and DOM.
    - `TestBed` configures the testing module and environment.
    - `CounterComponent` is the component we want to test.
2. **`describe()` block:**
    - Groups all tests related to `CounterComponent`.
3. **Variables:**
    - `component`: Holds the component instance.
    - `fixture`: Test wrapper around the component and its template.
    - `compiled`: Represents the native DOM element for querying.
4. **`beforeEach(async () => {...})`:**
    - Runs before each test.

    - `TestBed.configureTestingModule()` sets up testing module with declarations.
    - `compileComponents()` compiles template and CSS asynchronously.
    - Creates component instance and gets reference to DOM element.
    - Calls `fixture.detectChanges()` to perform data binding and initialize the template with component data.
5. **Tests:**
    - **Test 1:** Check if the component is created successfully.
    - **Test 2:** Verify initial rendered text shows `Count: 0`.
        - Use `querySelector` to get `<p>` tag.
        - Check text content.
    - **Test 3:** Simulate user interaction by clicking the button.
        - Find the button.
        - Trigger click event.
        - Call `fixture.detectChanges()` so Angular updates the view.
        - Verify component's `count` property incremented.
        - Verify updated count reflected in the DOM.
    - **Test 4:** Call increment method directly multiple times.
        - Check component internal state.
        - Verify DOM reflects updated count.

---

### Why use `fixture.detectChanges()`?

Angular runs change detection to update the DOM after component data changes. When you manually update component properties or simulate events, calling `detectChanges()` applies those changes to the view so the DOM and component state stay in sync during tests.

---

### How to run this test?

1. Run `ng test` in your Angular project.
2. Karma test runner will execute the test suite and show results.
3. You should see all tests passing if the component behaves as expected.

---

If you want, I can provide an example for testing a service, HTTP calls, or forms next!