deep-dive guide for **Intermediate and Advanced Angular Interview Questions with Answers**, covering not just Angular core concepts, but also **RxJS, NgRx, performance optimization, DI, lifecycle hooks, and architecture**

🟦 Intermediate Angular Questions

1. What is the difference between a component and a directive?

- A **component** is a directive with a template that controls a view.
- A **directive** adds behavior or modifies the DOM without its own template (e.g., *ngIf, ngClass).
  In short, all components are directives, but not all directives are components.

---

2. How does Angular handle change detection?

Angular's **change detection mechanism** runs after every asynchronous event (like a click or API call) and checks for data changes in the component tree.
The **Zone.js** library helps detect async tasks and trigger re-rendering.
For performance, developers can use **ChangeDetectionStrategy.OnPush** to update only when input references change.

---

3. Explain the role of Dependency Injection (DI) in Angular.

Dependency Injection is the design pattern Angular uses to supply a component or service with its required dependencies.
It helps in **testability**, **maintainability**, and **loose coupling**.
Angular creates an injector tree that resolves dependencies hierarchically.

---

4. What are Angular lifecycle hooks and why are they important?

Lifecycle hooks let you tap into key moments in a component's lifecycle, such as initialization, change detection, and destruction.
Common ones: ngOnInit, ngOnChanges, ngAfterViewInit, and ngOnDestroy.
They help with **setup, cleanup, and reacting to input changes**.

---

5. How does Angular handle routing and lazy loading?

Angular's **RouterModule** defines routes and can split the app into **feature modules** loaded only when needed — known as **lazy loading**.
This improves initial load performance and keeps the main bundle small.

---

6. What is the difference between Subject, BehaviorSubject, and ReplaySubject in RxJS?

- **Subject**: Emits to all subscribers after subscription.
- **BehaviorSubject**: Stores and emits the latest value immediately to new subscribers.
- **ReplaySubject**: Replays a specified number of previous emissions to new subscribers.

---

7. What is the role of Angular Services?

Services hold reusable logic or data that can be shared across components (like API calls or state management).
They're typically injected via DI and defined as **singleton instances** at module or root level.

---

8. How do Pipes work in Angular?

Pipes transform data in templates — e.g., formatting text, dates, or numbers.
You can create **pure** (deterministic) or **impure** (re-evaluated on every change) pipes depending on your need.

---

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

- **Template-driven forms**: Simpler, suitable for small forms; logic in templates.
- **Reactive forms**: More robust, testable, and scalable; logic defined in TypeScript, ideal for complex or dynamic forms.

---

10. What are Guards in Angular Routing?

Guards are used to control navigation — examples include:

- CanActivate (to enter a route),
- CanDeactivate (to leave a route),
- Resolve (to fetch data before route activation).
  They help with **security** and **route management**.

🟥 Advanced Angular Questions

1. How does Angular's Ivy renderer improve performance?

Ivy, Angular’s rendering engine, compiles components into efficient, tree-shakable instructions.
It reduces bundle size, speeds up compilation, and enables features like **partial compilation** and **faster change detection**.

---

2. Explain the role of RxJS in Angular.

RxJS provides a reactive programming model for managing **asynchronous data streams** like user input, HTTP calls, and WebSockets.
Angular’s HttpClient, Forms, and event systems are heavily based on RxJS observables.

---

3. What is NgRx and when should you use it?

NgRx is a state management library based on the **Redux pattern**, providing a predictable, single source of truth for state.
It’s most useful for **large, complex apps** where multiple components need to share or mutate common state in a controlled way.

---

4. Explain the concept of selectors and effects in NgRx.

- **Selectors**: Functions that extract and derive specific parts of the store state efficiently.
- **Effects**: Handle side effects such as API calls or async operations triggered by dispatched actions.

---

5. What is the difference between TemplateRef and ViewContainerRef?

- **TemplateRef** represents an embedded template (like the content of ng-template).
- **ViewContainerRef** represents a container that can programmatically create, insert, or remove views from that template.
  Together, they enable **dynamic view rendering**.

---

6. How does Angular optimize rendering with OnPush strategy?

When using **OnPush**, Angular skips change detection for a component unless:

- An input reference changes,
- An event originates from inside the component, or

- An observable emits a new value.
  This greatly improves performance in large component trees.

---

7. How do you handle memory leaks in Angular?

Common strategies:

- Unsubscribe from observables in ngOnDestroy, or use takeUntil operators.
- Avoid retaining DOM references or event listeners unnecessarily.
- Use tools like Chrome's **Performance tab** or **RxJS Marbles** for debugging.

---

8. What are Zones in Angular?

Zone.js patches async operations (like setTimeout or Promises) to know when to trigger change detection.
While essential, advanced apps can use **NgZone.runOutsideAngular()** to skip unnecessary re-renders for performance-heavy tasks.

---

9. How does Ahead-of-Time (AOT) compilation work?

AOT compiles templates and components at build time, rather than runtime.
This reduces runtime errors, improves startup time, and provides smaller, more secure bundles.

---

10. What are common Angular performance optimization techniques?

- Use **ChangeDetectionStrategy.OnPush**
- Use **trackBy** in *ngFor loops
- Implement **lazy loading** and **preloading strategies**
- Avoid unnecessary DOM manipulations
- Use **pure pipes** and **memoized selectors**
- Minimize RxJS subscription chains using operators like switchMap and shareReplay