Angular Interview Concepts: Detailed Explanation for 2+ Years Experience

---

1. Angular Core Concepts

Angular is a TypeScript-based front-end web application framework developed by Google. Core concepts include: - **Modules**: Containers to group components, directives, pipes, and services. - **Components**: Building blocks with HTML templates, logic (TypeScript), and styles. - **Templates**: Define the UI layout using Angular syntax (e.g., `*ngIf`, `{{ }}`). - **Metadata & Decorators**: `@Component`, `@NgModule`, etc., tell Angular how to process the class. - **Dependency Injection (DI)**: Built-in DI framework to provide services.

---

2. Component Communication & Data Binding

Angular supports multiple types of binding and communication: - **@Input()**: Pass data from parent to child. - **@Output()**: Emit events from child to parent. - **EventEmitter**: Used with `@Output()` to emit custom events. - **ViewChild / ContentChild**: Access child component or DOM element. - **Two-way binding**: Using `[(ngModel)]` for syncing data. - **Service-based communication**: Sharing data between unrelated components using a shared service with RxJS `Subject` or `BehaviorSubject`.

---

3. Reactive & Template-driven Forms

• **Template-driven Forms**:
• Simpler.
• Declared in the template using `ngModel`.
• Suitable for basic forms.

• Uses `FormsModule`.

• **Reactive Forms**:

• Defined in TypeScript using `FormGroup`, `FormControl`, `FormBuilder`.
• Better scalability, unit testing, dynamic control.
• Uses `ReactiveFormsModule`.

Validation can be template-based or programmatic.

---

4. Directives & Pipes

• **Directives**:
• **Structural** (`*ngIf`, `*ngFor`): Modify layout.
• **Attribute**: Change appearance or behavior (e.g., `ngClass`, `ngStyle`).

• **Custom Directives**: Implemented using `@Directive()`.

- **Pipes**:

- Transform data in templates (e.g., `date`, `uppercase`, `currency`).
- **Custom Pipes**: Created using `@Pipe()` with `transform()` method.

---

5. RxJS & Observables

- **Observables**: Asynchronous data streams.
- **Operators**: `map`, `filter`, `switchMap`, `mergeMap`, `take`, `tap`.
- **Subjects & BehaviorSubjects**: Multicast streams, useful in service-based communication.
- **Subscription Handling**: Always unsubscribe to avoid memory leaks (e.g., `takeUntil`, `async pipe`).

---

6. Routing, Guards, Lazy Loading

- **RouterModule**: Configures application routes.
- **Lazy Loading**: Load modules only when needed via `loadChildren`.
- **Guards**:
- `CanActivate`: Protect routes.
- `CanDeactivate`: Prevent leaving unsaved changes.
- `Resolve`: Preload data.

---

7. HTTPClient, Interceptors, and Services

- **HttpClientModule**: Handles HTTP operations.
- **CRUD operations**: `get`, `post`, `put`, `delete`.
- **Interceptors**: Modify requests or responses globally (e.g., add auth tokens).
- **Error Handling**: Use `catchError` from RxJS.

---

8. Standalone Components & Signals (Angular 17+)

- **Standalone Components**: Do not require a module, declared using `standalone: true`.
- **Signals**:
- A reactive primitive for tracking state.
- Alternative to `BehaviorSubject` or `ngRx` for simple state tracking.
- Introduced in Angular 16+, more ergonomic in Angular 17.

---

9. State Management (NgRx basics)

- **NgRx Store**: Redux-style global state management.
- **Actions, Reducers, Selectors, Effects**: Core elements.

- **Why NgRx?**: For large-scale apps, manage predictable shared state.

---

10. Testing (Jasmine/Karma)

- **Jasmine**: Testing framework for unit tests.
- **Karma**: Test runner.
- **TestBed**: Configures and initializes environment for unit tests.
- **Spies and Mocks**: Replace dependencies.
- **Integration Testing**: Simulate component interactions.

---

11. Angular CLI & Build Optimization

- **Angular CLI**: Tooling for scaffolding, serving, testing, and building.
- **Commands**: `ng generate`, `ng build --prod`, `ng test`, `ng lint`.
- **Optimization**: AOT, Tree-shaking, Lazy Loading, differential loading.

---

12. Real-World Project Scenarios

- **Role-based authentication**
- **Form validation with dynamic fields**
- **Reusable modal and dropdown components**
- **HTTP error interceptor with retry strategy**
- **Multi-step forms with progress tracking**
- **Reusable chart and dashboard components**

---

13. Integration with ASP.NET Core Web API

- **CORS Configuration** in Startup.cs
- **JWT Authentication**: Token-based flow from Angular.
- **HTTP calls**: Angular `HttpClient` calling `.NET` endpoints.
- **Model Binding**: Ensure DTOs match on both sides.
- **File Upload & Download**: Using `FormData` and stream responses.
- **Error Handling**: Map HTTP status codes to Angular UI.

This forms a solid base for Angular interview preparation, especially for developers working in ASP.NET + Angular ecosystems.