Angular Interview Q&A; Cheat Sheet

Short answers • bullet points • code snippets

BASICS

1) What is Angular?

- Angular is a TypeScript-based front-end framework for building SPAs.
- Component-based architecture, RxJS for reactivity, Angular CLI for tooling.
- Includes built-in DI, forms, HTTP, routing.

2) Angular Architecture

- Modules → group functionality.
- Components → control UI & logic.
- Templates → HTML with Angular syntax.
- Services & Dependency Injection.
- Directives, Pipes, RxJS Observables.

3) Components

- Smallest UI building block.
- Defined with @Component decorator.

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

@Component({
  selector: 'app-hello',
  template: `<h1>Hello {{name}}</h1>`
})
export class HelloComponent {
  name = 'Angular';
}
```

4) Data Binding Types

- Interpolation: {{ value }}
- Property binding: [property] = 'value'
- Event binding: (event) = 'handler()'
- Two-way binding: [(ngModel)] = 'value'

```
<input [value]="name" (input)="name=$event.target.value">
<p>Hello {{name}}</p>
```

INTERMEDIATE

5) Directives

- Structural: *ngIf, *ngFor
- Attribute: [ngClass], [ngStyle], custom directives

```
<div *ngIf="isVisible">Visible!</div>
<li *ngFor="let item of items">{{item}}</li>
```

6) Services & Dependency Injection

- Services hold business logic, reusable code.
- Injected into components using constructor.
- Angular uses hierarchical DI system.

```
@Injectable({ providedIn: 'root' })
export class UserService {
  getUsers() { return ['A', 'B', 'C']; }
}

@Component({...})
export class AppComponent {
  constructor(private userService: UserService) {}
}
```

7) Lifecycle Hooks

- ngOnInit – run once after component initialized.
- ngOnChanges – runs when @Input() changes.
- ngOnDestroy – cleanup tasks (unsubscribe).

```
ngOnInit() { console.log('Component initialized'); }
ngOnDestroy() { console.log('Cleanup'); }
```

8) Angular Forms

- Template-driven forms (ngModel).
- Reactive forms (FormControl, FormGroup).
- Reactive forms give better scalability.

```
// Reactive form
form = new FormGroup({
  name: new FormControl('')
});

onSubmit() { console.log(this.form.value); }
```

ADVANCED

9) RxJS & Observables

- Observables provide async streams of data.
- Operators like map, filter, switchMap transform streams.
- Subscribe or use async pipe.

```
this.http.get<User[]>('/api/users')
  .pipe(map(users => users.filter(u => u.active)))
  .subscribe(console.log);
```

10) Change Detection

- Angular runs change detection to update DOM when state changes.
- Strategies: Default (checks all), OnPush (checks only when inputs change).

```
@Component({
  selector: 'app-child',
  changeDetection: ChangeDetectionStrategy.OnPush
})
export class ChildComponent {}
```

11) Routing

- RouterModule defines routes.
- Supports lazy loading, route guards, parameters.

```
const routes: Routes = [
  { path: 'home', component: HomeComponent },
  { path: 'user/:id', component: UserComponent }
];
```

12) Angular Modules

- Root module (AppModule).
- Feature modules (lazy loaded).
- Shared module for reusable components.

```
@NgModule({
  declarations: [AppComponent],
  imports: [BrowserModule, FormsModule],
  bootstrap: [AppComponent]
})
export class AppModule {}
```

EXPERT / PRACTICAL

13) Angular Universal (SSR)

- Server-side rendering improves SEO & first paint.
- Angular Universal pre-renders HTML on server.

14) Ahead-of-Time (AOT) Compilation

- Compiles Angular HTML & TypeScript into JS before browser runs it.
- Faster rendering, better security, smaller bundles.

15) State Management

- NgRx (Redux-like state management).
- Services + BehaviorSubject for simpler apps.
- Akita, NGXS are alternatives.

16) Performance Optimization

- Use OnPush change detection.
- Lazy load modules.
- TrackBy in ngFor.
- Unsubscribe from Observables.
- Preload modules strategically.

17) Testing in Angular

- Unit tests with Jasmine + Karma.
- E2E tests with Protractor (deprecated) or Cypress/Playwright.
- TestBed for component testing.

```
it('should render title', () => {
  const fixture = TestBed.createComponent(AppComponent);
  fixture.detectChanges();
  const compiled = fixture.nativeElement as HTMLElement;
  expect(compiled.querySelector('h1')?.textContent).toContain('Hello Angular');
});
```