Angular Cheat Sheet — modern standalone Angular, verified against angular.dev

NG CLI

```
npm create @angular@latest my-app
ng serve            # dev server :4200
ng build            # prod bundle -> dist/
ng test             # unit tests
ng generate component user    # ng g c
ng generate service api       # ng g s
ng add @angular/material
ng update @angular/core @angular/cli
```

Also `ng g directive|pipe|guard|interceptor|resolver`.

BOOTSTRAP + APPLICATIONCONFIG

```
// main.ts
bootstrapApplication(App, appConfig);

// app.config.ts
export const appConfig: ApplicationConfig = {
  providers: [
    provideBrowserGlobalErrorListeners(),
    provideZonelessChangeDetection(),
    provideRouter(routes),
    provideHttpClient(withFetch()),
  ],
};
```

No root `NgModule` — compose with `provide*` fns.

STANDALONE COMPONENT

```
@Component({
  selector: 'app-user',
  imports: [RouterLink, UpperCasePipe],
  template: `<h1>{{ name() }}</h1>`,
  styleUrl: './user.css',
  changeDetection: ChangeDetectionStrategy.OnPush,
})
export class User {
  name = input.required<string>();
}
```

`standalone: true` is the default; drop it.

TEMPLATE BINDING

`{{ expr }}`	text interpolation
`[src]="u"`	property binding
`[attr.role]="r"`	attribute binding
`[class.on]="a"`	toggle one class
`[style.width.px]="w"`	style + unit
`(click)="save($event)"`	event binding
`[(value)]="name"`	two-way (model)
`#ref`	template ref var
`@let n = a + b;`	template variable

CONTROL FLOW + @DEFER

```
@if (user(); as u) { {{ u.name }} }
@else if (loading()) { … } @else { Guest }

@for (t of todos(); track t.id) {
  {{ $index }}: {{ t.text }}
} @empty { nothing yet }

@switch (status()) {
  @case ('ok') { OK } @default { … }
}

@defer (on viewport; prefetch on idle) {
  <app-chart />
} @placeholder { … } @loading (after 100ms) { … }
  @error { failed }
```

`track` is required on `@for`. Triggers: `idle`, `viewport`, `interaction`, `hover`, `timer`, `immediate`, `when`.

INPUT() / OUTPUT() / MODEL()

```
id  = input.required<number>();
tag = input('draft', { alias: 'label' });
size = input(0, { transform: numberAttribute });

saved = output<User>();
this.saved.emit(user);

// two-way: parent writes [(checked)]="flag"
checked = model(false);
this.checked.set(true);
```

Signal APIs replace `@Input()` / `@Output()`.

SIGNAL / COMPUTED / EFFECT

```
count = signal(0);
count.set(1);
count.update(n => n + 1);

double = computed(() => this.count() * 2);

effect(() => console.log(this.count()));
effect((onCleanup) => {
  const id = setInterval(fn, 1000);
  onCleanup(() => clearInterval(id));
});
untracked(() => this.count());   // read, no dep
```

`computed` is lazy + memoised. `effect` runs after render.

DEPENDENCY INJECTION

```
@Injectable({ providedIn: 'root' })
export class Api {
  private http = inject(HttpClient);
}

// consume anywhere in an injection context
private api = inject(Api);

export const CFG = new InjectionToken<Cfg>('cfg');
providers: [
  { provide: CFG, useValue: cfg },
  { provide: Api, useClass: FakeApi },
]
```

Strategies: `useClass` / `useValue` / `useFactory` / `useExisting` / `multi`. Resolves up: element → route → environment injector.

RXJS + ASYNC + TOSIGNAL

```
term$ = new BehaviorSubject('');

results$ = this.term$.pipe(
  debounceTime(300),
  distinctUntilChanged(),
  switchMap(q => this.api.search(q)),
  catchError(() => of([])),
);

results = toSignal(this.results$, { initialValue: [] });
count$  = toObservable(this.count);
```

Unsubscribe via `async` pipe or `takeUntilDestroyed()`. Template: `@for (r of results$ | async; track r.id)`.

HTTPCLIENT + INTERCEPTOR

```
provideHttpClient(
  withFetch(),
  withInterceptors([authInterceptor]),
)

users$ = this.http.get<User[]>('/api/users', {
  params: { page: 1 },
});
this.http.post<User>('/api/users', body).subscribe();

export const authInterceptor: HttpInterceptorFn =
  (req, next) => next(req.clone({
    setHeaders: { Authorization: inject(Auth).token() },
  }));
```

Errors: `HttpErrorResponse` in `catchError`; test with `HttpTestingController`.

ROUTER + LINK + GUARD

```
export const routes: Routes = [
  { path: '', component: Home, title: 'Home' },
  { path: 'user/:id',
    loadComponent: () => import('./user')
      .then(m => m.User),
    canActivate: [authGuard] },
  { path: '**', redirectTo: '' },
];

export const authGuard: CanActivateFn = () =>
  inject(Auth).isLoggedIn()
    || inject(Router).createUrlTree(['/login']);
```

```
<a routerLink="/user/1" routerLinkActive="active">·
<router-outlet />
```

TYPED REACTIVE FORM

```
private fb = inject(FormBuilder);

form = this.fb.group({
  email: ['', [Validators.required,
               Validators.email]],
  age:   this.fb.control(0, { nonNullable: true }),
  addr:  this.fb.group({ city: [''] }),
});

this.form.value;                  // typed partial
this.form.controls.email.valueChanges;
this.form.patchValue({ age: 30 });
this.form.controls.email.invalid;
```

`<form [formGroup]="form" (ngSubmit)="save()">` with `formControlName="email"`.

LIFECYCLE HOOK ORDER

1. `constructor()` — inject deps
2. `ngOnChanges()` — on input change
3. `ngOnInit()` — once, after 1st change
4. `ngDoCheck()`
5. `ngAfterContentInit()` / `…Checked()`
6. `ngAfterViewInit()` / `…Checked()`
7. `ngOnDestroy()`

```
afterNextRender(fn) / afterRender(fn)  // DOM
inject(DestroyRef).onDestroy(fn)
source$.pipe(takeUntilDestroyed())
```

`OnPush`: check only on input change, event, or signal read.

TESTBED SKELETON

```
beforeEach(async () => {
  await TestBed.configureTestingModule({
    imports: [User],
    providers: [
      provideHttpClientTesting(),
      { provide: Api, useValue: fake },
    ],
  }).compileComponents();
});

it('renders the name', () => {
  const f = TestBed.createComponent(User);
  f.componentRef.setInput('name', 'Ada');
  f.detectChanges();
  expect(f.nativeElement.textContent).toContain('Ada');
});
```

Query: `f.debugElement.query(By.css('h1'))`. Async: `fakeAsync` + `tick()`.