

Guida completa ad Angular

Stefanoo94

Contents

Guida completa ad Angular — Da zero a produzione	2
Panoramica	2
Prerequisiti	2
Ambiente di sviluppo e CLI	3
Installazione	3
Comandi essenziali	3
Struttura di un progetto Angular	3
TypeScript per Angular (breve ripasso)	3
Componenti	4
Creare un componente	4
Lifecycle hooks	4
Template e Data Binding	5
Interpolation, Property/EVENT Binding, Two-way Binding	5
Directives: structural e attribute	5
Comunicazione tra componenti	5
@Input, @Output, EventEmitter	5
ViewChild, ContentChild	6
Pipes	6
Servizi e Dependency Injection	6
HttpClient e gestione API	7
Routing e Lazy Loading	7
Guards e Resolvers	8
Forms: Template-driven e Reactive Forms	8
RxJS e gestione asincrona	8
State management (introduzione)	9
Testing (unit e e2e)	9
Ottimizzazione e performance	9

Build e Deploy	9
Argomenti avanzati	10
Template progetto starter (MyShop - mini e-commerce)	10
Struttura file proposta	10
File di esempio principali (codice)	10
Esercizi guidati e mini-progetti	13
Checklist best practices	13
Comandi CLI riassunto (blocco)	13
Riferimenti e risorse consigliate	14
Appendice: comandi utili per conversione Markdown → PDF	14

Guida completa ad Angular — Da zero a produzione

Autore: Stefanoo94 (puoi sostituire il nome) Data: 2025-11-14

Indice - Panoramica - Prerequisiti - Ambiente di sviluppo e CLI - Installazione - Comandi essenziali - Struttura di un progetto Angular - TypeScript per Angular (breve ripasso) - Componenti - Creare un componente - Lifecycle hooks - Template e Data Binding - Interpolation, Property/EVENT Binding, Two-way Binding - Directives: structural e attribute - Comunicazione tra componenti - @Input, @Output, EventEmitter - ViewChild, ContentChild - Pipes - Servizi e Dependency Injection - HttpClient e gestione API - Routing e Lazy Loading - Guards e Resolvers - Forms: Template-driven e Reactive Forms - RxJS e gestione asincrona - State management (introduzione) - Testing (unit e e2e) - Ottimizzazione e performance - Build e Deploy - Argomenti avanzati - Template progetto starter (MyShop - mini e-commerce) - Struttura file proposta - File di esempio principali (codice) - Esercizi guidati e mini-progetti - Checklist best practices - Comandi CLI riassunto (blocco) - Riferimenti e risorse consigliate - Appendice: comandi utili per conversione Markdown → PDF

Panoramica

Obiettivo: portarti da zero a un livello in cui puoi progettare, sviluppare, testare e distribuire applicazioni Angular complesse, produttive e manutenibili.

Durata suggerita: 8–12 settimane (con percorso intensivo) o 3–6 mesi per approfondimenti e argomenti avanzati.

Metodo: teoria + esercizi pratici + un progetto reale completo (MyShop esempio).

Prerequisiti

- HTML5 e CSS3 (flexbox, grid, responsive)
- JavaScript moderno (ES6+): arrow functions, promises, async/await, modules, destructuring
- TypeScript di base: tipi, interfacce, classi, generics
- Nozioni base di Node.js e npm/yarn
- Familiarità con Git (raccomandata)

Ambiente di sviluppo e CLI

Installazione

1. Installa Node.js (versione LTS consigliata) — <https://nodejs.org>

2. Verifica:

```
node -v  
npm -v
```

3. Installa Angular CLI globalmente:

```
npm install -g @angular/cli
```

Comandi essenziali

- Creare nuovo progetto:

```
ng new my-app  
# opzioni utili:  
# --routing (aggiunge router), --style=scss (usa SCSS)
```

- Servire in sviluppo:

```
cd my-app  
ng serve --open
```

- Generare artefatti:

```
ng generate component nome-componente  
ng generate service nome-servizio  
ng generate module nome-modulo  
# abbreviazioni:  
ng g c nome  
ng g s nome  
ng g m nome
```

- Build di produzione:

```
ng build --configuration production
```

- Test:

```
ng test           # unit tests (Karma + Jasmine)  
ng e2e           # end-to-end (può essere Cypress)  
ng lint
```

Struttura di un progetto Angular

(Principali cartelle/file in src/) - src/ - app/ - app.module.ts - app.component.ts/html/scss - feature-modules/ - assets/ - environments/ - environment.ts - environment.prod.ts - index.html - main.ts - styles.scss - angular.json - package.json - tsconfig.json

TypeScript per Angular (breve ripasso)

- Tipi base: string, number, boolean, any, unknown, void, never
- Interfacce e type alias:

```
export interface Product {
  id: number;
  name: string;
  price: number;
  description?: string;
}
```

- Classi e access modifiers:

```
export class CartService {
  private items: Product[] = [];
  add(product: Product) { this.items.push(product); }
}
```

Componenti

Creare un componente

ng g c shared/header

Esempio base: app.component.ts

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

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
```

```
export class AppComponent {
  title = 'MyShop';
}
```

app.component.html

```
<header>
  <h1>{{ title }}</h1>
</header>
<router-outlet></router-outlet>
```

Lifecycle hooks

Ordine tipico: - ngOnChanges - ngOnInit - ngDoCheck - ngAfterContentInit - ngAfterContentChecked - ngAfterViewInit - ngAfterViewChecked - ngOnDestroy

Esempio:

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

```
@Component({...})
export class ExampleComponent implements OnInit, OnDestroy {
  ngOnInit() { console.log('init'); }
  ngOnDestroy() { console.log('destroy'); }
}
```

Template e Data Binding

Interpolation, Property/EVENT Binding, Two-way Binding

- Interpolation:

```
<p>{{ product.name }} - {{ product.price | currency }}</p>
```

- Property binding:

```
<img [src]="product.image" [alt]="product.name">
```

- Event binding:

```
<button (click)="addToCart(product)">Aggiungi</button>
```

- Two-way binding (FormsModule richiesto):

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

Directives: structural e attribute

- Structural: *ngIf*, *ngFor*

```
<ul>
  <li *ngFor="let p of products">{{ p.name }}</li>
</ul>
```

- Attribute: *[ngClass]*, *[ngStyle]*, e direttive personalizzate

Creare direttiva personalizzata semplice:

ng g directive shared/highlight

Esempio:

```
import { Directive, ElementRef, Renderer2, HostListener } from '@angular/core';

@Directive({selector: '[appHighlight]'})
export class HighlightDirective {
  constructor(private el: ElementRef, private rd: Renderer2) {}
  @HostListener('mouseenter') onEnter() {
    this.rd.setStyle(this.el.nativeElement, 'backgroundColor', 'yellow');
  }
  @HostListener('mouseleave') onLeave() {
    this.rd.removeStyle(this.el.nativeElement, 'backgroundColor');
  }
}
```

Comunicazione tra componenti

@Input, @Output, EventEmitter

Parent → Child:

```
// child.component.ts
@Input() product!: Product;
```

Child → Parent:

```
// child.component.ts
@Output() added = new EventEmitter<Product>();
add() { this.added.emit(this.product); }
```

Nel parent:

```
<app-child [product]="p" (added)="onAdded($event)"></app-child>
```

ViewChild, ContentChild

- ViewChild per accesso diretto a template child:

```
@ViewChild('myInput') input: ElementRef;
ngAfterViewInit() {
  this.input.nativeElement.focus();
}
```

- ContentChild per componenti proiettati via ng-content

Pipes

- Built-in: DatePipe, UpperCasePipe, LowerCasePipe, CurrencyPipe, AsyncPipe
- Creare un pipe:

```
ng g pipe shared/truncate
```

Esempio:

```
import { Pipe, PipeTransform } from '@angular/core';

@Pipe({name: 'truncate'})
export class TruncatePipe implements PipeTransform {
  transform(value: string, limit = 50) {
    return value.length > limit ? value.slice(0, limit) + '...' : value;
  }
}
```

Servizi e Dependency Injection

Creare un servizio:

```
ng g s services/cart
```

Esempio di servizio:

```
import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { Product } from '../models/product.model';

@Injectable({ providedIn: 'root' })
export class CartService {
  private itemsSubject = new BehaviorSubject<Product[]>([]);
  items$ = this.itemsSubject.asObservable();

  add(product: Product) {
    const items = [...this.itemsSubject.getValue(), product];
    this.itemsSubject.next(items);
  }
}
```

```
}  
}
```

HttpClient e gestione API

- Importa HttpClientModule in app.module.ts:

```
import { HttpClientModule } from '@angular/common/http';  
  
@NgModule({  
  imports: [HttpClientModule]  
})  
export class AppModule {}
```

- Esempio di service per chiamate:

```
import { HttpClient } from '@angular/common/http';  
import { Injectable } from '@angular/core';  
import { Observable } from 'rxjs';  
import { Product } from '../models/product.model';  
  
@Injectable({ providedIn: 'root' })  
export class ProductService {  
  private API = 'https://fakestoreapi.com/products';  
  constructor(private http: HttpClient) {}  
  getAll(): Observable<Product[]> {  
    return this.http.get<Product[]>(this.API);  
  }  
  getById(id: number) {  
    return this.http.get<Product>(`${this.API}/${id}`);  
  }  
}
```

- Gestione errori con catchError (RxJS)

```
import { catchError } from 'rxjs/operators';  
import { throwError } from 'rxjs';  
  
this.http.get(...).pipe(  
  catchError(err => {  
    // log, map error  
    return throwError(() => new Error('Errore API'));  
  })  
);
```

Routing e Lazy Loading

Configurare il router:

```
// app-routing.module.ts  
const routes: Routes = [  
  { path: '', component: HomeComponent },  
  { path: 'product/:id', component: ProductDetailComponent },  
  { path: 'admin', loadChildren: () => import('./admin/admin.module').then(m => m.AdminModule) }  
];
```

- Lazy loading per moduli pesanti:

```
{ path: 'shop', loadChildren: () => import('./shop/shop.module').then(m => m.ShopModule) }
```

Guards e Resolvers

- CanActivate per proteggere rotte:

```
ng g guard auth/auth
```

Esempio:

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

- Resolver per pre-caricare dati prima della navigazione:

```
resolve(route: ActivatedRouteSnapshot) {  
  const id = route.paramMap.get('id');  
  return this.productService.getById(+id);  
}
```

Forms: Template-driven e Reactive Forms

Template-driven:

```
<form #f="ngForm" (ngSubmit)="onSubmit(f.value)">  
  <input name="email" ngModel required email />  
</form>
```

Reactive Forms:

```
import { FormBuilder, Validators } from '@angular/forms';  
  
this.form = this.fb.group({  
  name: ['', [Validators.required]],  
  email: ['', [Validators.required, Validators.email]],  
  passwords: this.fb.group({  
    password: ['', [Validators.required]],  
    confirm: ['', [Validators.required]]  
  }, { validators: this.passwordMatch })  
});
```

Async validator (es. controllo email già registrata) — restituisce Observable<ValidationErrors|null>

RxJS e gestione asincrona

Concetti principali: - Observable, Observer, Subscription - Subjects, BehaviorSubject, ReplaySubject - Operators: map, filter, switchMap, mergeMap, concatMap, debounceTime, distinctUntilChanged, catchError

Esempio: ricerca live con debounce e cancellazione chiamate precedenti

```
this.searchInput.valueChanges.pipe(  
  debounceTime(300),  
  distinctUntilChanged(),  
  switchMap(term => this.productService.search(term))  
)<subscribesubscribe>(results => this.results = results);
```

State management (introduzione)

- Soluzioni: NgRx (Redux-like), Akita, NGXS oppure pattern basati su services + BehaviorSubject (facade)
- Raccomandazione: per app medio/piccole usa services con BehaviorSubject; per app grandi valuta NgRx.

Testing (unit e e2e)

Unit test (Jasmine + Karma): - Esempio test per componente:

```
describe('AppComponent', () => {
  let fixture: ComponentFixture<AppComponent>;
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      declarations: [AppComponent],
      imports: [HttpClientTestingModule]
    }).compileComponents();
    fixture = TestBed.createComponent(AppComponent);
  });

  it('should create', () => {
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });
});
```

E2E testing: - Preferire Cypress (più moderno) oppure Playwright - Esempio semplice con Cypress:

```
describe('Home page', () => {
  it('loads', () => {
    cy.visit('/');
    cy.contains('MyShop');
  });
});
```

Ottimizzazione e performance

- ChangeDetectionStrategy.OnPush per componenti immutabili
- Lazy loading di moduli
- Evitare binding costosi in template
- Usare trackBy in *ngFor
- Analizzare bundle con source-map-explorer o webpack-bundle-analyzer
- Abilitare production build con AOT e minification:

```
ng build --configuration production
```

Build e Deploy

- Build prod:

```
ng build --configuration production
```

- Deploy su Firebase Hosting:

```
npm install -g firebase-tools
firebase login
firebase init hosting
firebase deploy
```

- Deploy su Netlify: build static site e carica la cartella dist/
- GitHub Pages (progetto statico)

Argomenti avanzati

- Server-side Rendering (Angular Universal)
- Progressive Web App (ng add @angular/pwa)
- Internationalization (i18n)
- Accessibilità (a11y)
- Micro frontends con Module Federation
- Performance avanzate e profiling

Template progetto starter (MyShop - mini e-commerce)

Obiettivo: fornire un template completo che includa struttura modulare, routing, servizi, store minimo (BehaviorSubject), e alcuni file di esempio.

Struttura file proposta

```
myshop/
  src/
    app/
      core/
        services/
          auth.service.ts
          product.service.ts
          cart.service.ts
        guards/
      features/
        shop/
          shop.module.ts
          product-list/
          product-detail/
        shared/
          components/
          pipes/
          app-routing.module.ts
          app.module.ts
      assets/
      environments/
      index.html
    angular.json
    package.json
```

File di esempio principali (codice)

- 1) app.module.ts

```

import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { HttpClientModule } from '@angular/common/http';
import { AppRoutingModuleModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { SharedModule } from './shared/shared.module';

@NgModule({
  declarations: [AppComponent],
  imports: [
    BrowserModule,
    HttpClientModule,
    AppRoutingModuleModule,
    SharedModule
  ],
  bootstrap: [AppComponent]
})
export class AppModule {}

2) app-routing.module.ts

import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './features/home/home.component';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'shop', loadChildren: () => import('./features/shop/shop.module').then(m => m.ShopModule) },
  { path: '**', redirectTo: '' }
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModuleModule {}

3) product.service.ts (core/services)

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Product } from '../models/product.model';

@Injectable({ providedIn: 'root' })
export class ProductService {
  private API = 'https://fakestoreapi.com/products';
  constructor(private http: HttpClient) {}
  getAll(): Observable<Product[]> {
    return this.http.get<Product[]>(this.API);
  }
  getById(id: number) {
    return this.http.get<Product>(`${this.API}/${id}`);
  }
  search(q: string) {
    return this.http.get<Product[]>(`${this.API}?search=${encodeURIComponent(q)}`);
  }
}

```

```

    }
  }
  4) cart.service.ts (core/services)

import { Injectable } from '@angular/core';
import { BehaviorSubject } from 'rxjs';
import { Product } from '../models/product.model';

@Injectable({ providedIn: 'root' })
export class CartService {
  private items = new BehaviorSubject<Product[]>([]);
  items$ = this.items.asObservable();

  add(product: Product) {
    const updated = [...this.items.getValue(), product];
    this.items.next(updated);
    localStorage.setItem('cart', JSON.stringify(updated));
  }

  remove(productId: number) {
    const updated = this.items.getValue().filter(p => p.id !== productId);
    this.items.next(updated);
    localStorage.setItem('cart', JSON.stringify(updated));
  }

  loadFromLocalStorage() {
    const raw = localStorage.getItem('cart');
    if (raw) this.items.next(JSON.parse(raw));
  }
}

```

5) Esempio product-list component (features/shop/product-list)

```

// product-list.component.ts
import { Component, OnInit } from '@angular/core';
import { ProductService } from 'src/app/core/services/product.service';
import { CartService } from 'src/app/core/services/cart.service';

@Component({
  selector: 'app-product-list',
  templateUrl: './product-list.component.html'
})
export class ProductListComponent implements OnInit {
  products = [];
  loading = false;
  constructor(private ps: ProductService, private cart: CartService) {}
  ngOnInit() {
    this.loading = true;
    this.ps.getAll().subscribe(p => { this.products = p; this.loading = false; });
  }
  addToCart(p) { this.cart.add(p); }
}

```

product-list.component.html

```
<div *ngIf="loading">Caricamento...</div>
```

```

<ul>
  <li *ngFor="let p of products">
    <h3>{{ p.title }}</h3>
    <p>{{ p.price | currency:'EUR' }}</p>
    <button (click)="addToCart(p)">Aggiungi</button>
  </li>
</ul>

```

Esercizi guidati e mini-progetti

Consiglio: implementa i seguenti progetti in ordine crescente di complessità: 1. Todo app (localStorage) 2. Blog reader (API + routing) 3. MyShop (mini-e-commerce — template qui sopra) 4. Dashboard con grafici (ngx-charts) 5. App con autenticazione mock e admin area protetta 6. PWA con caching offline

Per ogni progetto: scrivi test unitari, integra CI (GitHub Actions), configura deploy automatico su Firebase/Netlify.

Checklist best practices

- Componenti: piccoli e single responsibility
- Business logic: nei servizi
- Usare feature modules e barrel files con parsimonia
- Evitare memory leaks: unsubscribe (takeUntil o async pipe)
- Usare OnPush quando possibile
- Centralizzare gestione stato sensibile (auth, cart)
- Testare componenti critici e servizi

Comandi CLI riassunto (blocco)

```

# install cli
npm install -g @angular/cli

# create project
ng new myshop --routing --style=scss

# serve
cd myshop
ng serve --open

# generate
ng g c features/shop/product-list
ng g s core/services/product
ng g m features/shop --route shop --module app.module

# build prod
ng build --configuration production

# test
ng test
ng e2e

```

Riferimenti e risorse consigliate

- Documentazione ufficiale Angular — <https://angular.io> (Tour of Heroes)
- Angular CLI docs — <https://angular.io/cli>
- RxJS docs — <https://rxjs.dev>
- TypeScript handbook — <https://www.typescriptlang.org/docs/>
- Libri e corsi: ng-book, Angular University, corsi aggiornati su Udemy (verifica recensioni)
- Tooling: Angular DevTools (Chrome extension), Augury, ESLint + Prettier

Appendice: comandi utili per conversione Markdown → PDF

Se vuoi convertire il file Markdown in PDF sul tuo computer, ecco alcuni metodi:

1) Pandoc (versatile)

- Installare pandoc + wkhtmltopdf o usare LaTeX (per output più curato)

semplice conversione

```
pandoc guida-angular.md -o guida-angular.pdf
```

con stile e TOC

```
pandoc --pdf-engine=xelatex --toc --toc-depth=3 -V geometry:margin=1in guida-angular.md -o guida-angular.pdf
```

2) Visual Studio Code

- Apri guida-angular.md in VSCode, estensione “Markdown PDF” oppure “Print to PDF” da anteprima (CTRL+P nella preview)

3) GitHub → Stampa / Stampa su PDF

- Carica il .md in un repo, apri la preview su GitHub, stampa (stampa su PDF)

Fine del documento. Se vuoi la versione PDF pronta, dimmi come preferisci riceverla (caricamento qui nella chat, creazione di un repo pubblico su GitHub e push del file convertito, o altro) e la preparo e te la fornisco.