

Complete Angular Guide - Step by Step Tutorial

1. Angular CLI Setup & Basic Commands

Angular CLI - Command line interface for Angular development.

```
bash

# Install Angular CLI globally
npm i -g @angular/cli

# Check version
ng v

# Create new project
ng new your-app-name --inline-style --inline-template

# Start development server
ng serve
```

Generate Commands:

```
bash

ng g c componentName # Generate component
ng g s services/todo # Generate service
ng g p pipes/todo # Generate pipe
ng g guard guards/auth # Generate guard
```

2. Angular Most Used Tools & Packages

Essential packages for Angular development:

```
bash

ng add @angular/material # UI components
npm install @angular/cdk # Component dev kit
ng add @ngrx/store # State management
npm install rxjs # Reactive programming
ng add @angular/pwa # Progressive Web App
npm install -D tailwindcss # CSS framework
npm install --save-dev cypress # E2E testing
```

3. Folder Structure with Redux Constants Utils and Types

Organized folder structure for scalable Angular apps:

```
src/app/
├── components/  # Reusable UI components
├── pages/       # Route components
├── services/    # Business logic & API calls
├── guards/     # Route protection
├── pipes/      # Data transformation
├── models/     # TypeScript interfaces
├── types/      # Type definitions
├── constants/  # App constants
├── utils/      # Helper functions
├── store/      # NgRx state management
└── environments/ # Environment configs
```

Example files:

```
typescript

// types/user.types.ts
export interface User {
  id: number;
  name: string;
  email: string;
}

// constants/api.constants.ts
export const API_ENDPOINTS = {
  USERS: '/api/users',
  PRODUCTS: '/api/products'
} as const;

// utils/helpers.ts
export const formatDate = (date: Date): string => {
  return date.toLocaleDateString();
};
```

4. Angular Lifecycle Hooks

Lifecycle hooks - Methods called at different stages of component lifecycle:

typescript

```
@Component({
  selector: 'app-lifecycle',
  template: `<p>Lifecycle Demo</p>`
})
export class LifecycleComponent implements OnInit, OnDestroy {

  constructor() {} // Dependency injection

  ngOnInit(): void {
    // Component initialization
  }

  ngOnDestroy(): void {
    // Cleanup before destruction
  }
}
```

Hook execution order:

1. `constructor()` - Dependency injection
2. `ngOnInit()` - Component initialization
3. `ngOnDestroy()` - Cleanup

5. Tailwind CSS Configuration

Tailwind - Utility-first CSS framework for Angular:

bash

```
npm install -D tailwindcss postcss autoprefixer
npx tailwindcss init
```

tailwind.config.js:

javascript

```
module.exports = {  
  content: ["/src/**/*.{html,ts}"],  
  theme: {  
    extend: {  
      colors: {  
        primary: '#3B82F6'  
      }  
    },  
  },  
  plugins: [],  
}
```

styles.css:

```
CSS  
  
@tailwind base;  
@tailwind components;  
@tailwind utilities;
```

6. Component Decorator and Class Structure

@Component decorator defines Angular component metadata:

```
typescript
```

```

@Component({
  selector: 'app-product',
  standalone: true,
  template: `
    <div>
      <img [src]="product().image" [class]="imageClass">
      <h3>{{ product().name }}</h3>
      <button (click)="onAddToCart()">Add to Cart</button>
    </div>
  `,
  styles: [`div { padding: 16px; }`],
  imports: [CommonModule]
})
export class ProductComponent {
  @Input() product = signal({ name: 'Product', image: '/assets/img.jpg' });
  @Output() addToCart = new EventEmitter<number>();

  imageClass = 'w-full h-48 object-cover';

  onAddToCart() {
    this.addToCart.emit(this.product().id);
  }
}

```

7. Data Binding and Signals

Signals - Angular's reactive primitive for state management:

typescript

```

@Component({
  template: `
    <div>
      <p>Count: {{ count() }}</p>
      <p>Double: {{ doubleCount() }}</p>
      <button (click)="increment()"> + </button>
    </div>
  `,
})
export class CounterComponent {
  // Signal for reactive state
  count = signal(0);

  // Computed signal - auto updates
  doubleCount = computed(() => this.count() * 2);

  increment() {
    this.count.set(this.count() + 1);
  }
}

```

Data binding types:

```

typescript

// Interpolation
{{ title }}

// Property binding
[src]="imageUrl"
[disabled]="isDisabled"

// Class & style binding
[class.active]="isActive"
[style.color]="textColor"

```

8. Event Binding and Two-way Binding

Event binding - Handle user interactions:

```

typescript

```

```

@Component({
  template: `
    <!-- Event Binding -->
    <button (click)="onClick()">Click me</button>
    <input (keyup)="onKeyUp($event)" (focus)="onFocus()">

    <!-- Two-way Binding -->
    <input [(ngModel)]="name" placeholder="Enter name">
    <p>Hello, {{ name }}!</p>

    <!-- Signal with two-way binding -->
    <input [ngModel]="userName()" (ngModelChange)="userName.set($event)">
  `
})
export class BindingComponent {
  name = '';
  userName = signal('');

  onClick() {
    console.log("Button clicked!");
  }

  onKeyUp(event: KeyboardEvent) {
    console.log((event.target as HTMLInputElement).value);
  }
}

```

9. Conditional Rendering and Loops

New @if/@for syntax (Angular 17+):

```

typescript

```

```

@Component({
  template: `
    <!-- Conditional rendering -->
    @if (isLoggedIn) {
      <p>Welcome {{ user.name }}!</p>
    } @else {
      <p>Please log in</p>
    }

    <!-- Loop with tracking -->
    <ul>
      @for (todo of todos(); track todo.id) {
        <li>{{ todo.title }}</li>
      } @empty {
        <li>No todos</li>
      }
    </ul>
  `
})
export class ControlFlowComponent {
  isLoggedIn = true;
  user = { name: 'John' };

  todos = signal([
    { id: 1, title: 'Learn Angular', completed: false }
  ]);
}

```

Legacy NgIf/NgFor:

```

html
<div *ngIf="showContent">Content</div>
<li *ngFor="let item of items; trackBy: trackByFn">{{ item.name }}</li>

```

10. Component Communication

Parent to Child - Using @Input and @Output:

```

typescript

```



```
// Child Component
@Component({
  selector: 'app-child',
  template: `
    <div>
      <h3>{{ title }}</h3>
      <button (click)="notify()">Notify Parent</button>
    </div>
  `,
})
export class ChildComponent {
  @Input() title: string = "";
  @Output() childEvent = new EventEmitter<string>();

  notify() {
    this.childEvent.emit('Hello from child!');
  }
}
```

```
// Parent Component
@Component({
  template: `
    <app-child
      [title]="parentTitle"
      (childEvent)="onChildEvent($event)">
    </app-child>
  `,
})
export class ParentComponent {
  parentTitle = 'Child Component';

  onChildEvent(data: string) {
    console.log('Received:', data);
  }
}
```

Service Communication - For grandparent to child:

typescript

```

@Injectable({ providedIn: 'root' })
export class CommunicationService {
  private messageSubject = new BehaviorSubject<string>("");
  message$ = this.messageSubject.asObservable();

  sendMessage(message: string) {
    this.messageSubject.next(message);
  }
}

```

11. Angular Router

Router setup - Navigation between components:

typescript

```

// app-routing.module.ts
export const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', loadComponent: () => import('./pages/home/home.component').then(m => m.HomeComponent) },
  { path: 'products/:id', loadComponent: () => import('./pages/product/product.component').then(m => m.ProductComponent) },
  { path: '**', loadComponent: () => import('./pages/not-found/not-found.component').then(m => m.NotFoundComponent) },
];

// Navigation template
@Component({
  template: `
    <nav>
      <a routerLink="/home" routerLinkActive="active">Home</a>
      <a [routerLink]="['/products', productId]>Product</a>
    </nav>
    <router-outlet> </router-outlet>
  `
})
export class NavigationComponent {
  productId = 123;
}

```

Route parameters:

typescript

```
@Component({
  template: `<p>Product ID: {{ productId }}</p>`
})
export class ProductComponent implements OnInit {
  productId: string = "";

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.route.params.subscribe(params => {
      this.productId = params['id'];
    });
  }
}
```

12. Angular Services and Dependency Injection

Services - Singleton classes for business logic:

```
typescript
```

```

@Injectable({
  providedIn: 'root' // Singleton service
})
export class DataService {
  private apiUrl = 'https://api.example.com';

  constructor(private http: HttpClient) {}

  getData(): Observable<any[]> {
    return this.http.get<any[]>(`${this.apiUrl}/data`);
  }
}

// Component using service
@Component({
  template: `
    <ul>
      <li *ngFor="let item of data">{{ item.name }}</li>
    </ul>
  `
})
export class DataComponent implements OnInit {
  data: any[] = [];

  // Modern inject function
  private dataService = inject(DataService);

  ngOnInit() {
    this.dataService.getData().subscribe(data => {
      this.data = data;
    });
  }
}

```

13. NgRx State Management

NgRx - Redux pattern for Angular state management:

```

bash

ng add @ngrx/store
ng add @ngrx/effects

```

typescript

```
// actions/todo.actions.ts
export const loadTodos = createAction('[Todo] Load Todos');
export const addTodo = createAction('[Todo] Add Todo', props<{ todo: Todo }>());

// reducers/todo.reducer.ts
const initialState: TodoState = {
  todos: [],
  loading: false
};

export const todoReducer = createReducer(
  initialState,
  on(loadTodos, state => ({ ...state, loading: true })),
  on(addTodo, (state, { todo }) => ({
    ...state,
    todos: [...state.todos, todo]
  }))
);

// Component using NgRx
@Component({
  template: `
    <button (click)="loadTodos()">Load Todos</button>
    <ul>
      <li *ngFor="let todo of todos$ | async">{{ todo.title }}</li>
    </ul>
  `
})
export class TodoComponent {
  todos$ = this.store.select(selectAllTodos);

  constructor(private store: Store) {}

  loadTodos() {
    this.store.dispatch(loadTodos());
  }
}
```

14. HTTP Client and Async Operations

HTTP Client - Making API calls:

typescript

```
@Injectable({ providedIn: 'root' })
export class ApiService {
  private baseUrl = 'https://api.example.com';

  constructor(private http: HttpClient) {}

  getUsers(): Observable<User[]> {
    return this.http.get<User[]>(`${this.baseUrl}/users`);
  }

  createUser(user: User): Observable<User> {
    return this.http.post<User>(`${this.baseUrl}/users`, user)
      .pipe(
        timeout(5000),
        retry(2),
        catchError(this.handleError)
      );
  }

  // Async/Await approach
  async getUsersAsync(): Promise<User[]> {
    try {
      const users = await this.http.get<User[]>(`${this.baseUrl}/users`).toPromise();
      return users || [];
    } catch (error) {
      console.error('Error:', error);
      throw error;
    }
  }

  private handleError(error: any): Observable<never> {
    console.error('API Error:', error);
    throw error;
  }
}
```

15. Pipes and Async Pipe

Pipes - Transform data in templates:

typescript

```

// Custom Pipe
@Pipe({ name: 'capitalize', standalone: true })
export class CapitalizePipe implements PipeTransform {
  transform(value: string): string {
    return value.charAt(0).toUpperCase() + value.slice(1);
  }
}

// Component using pipes
@Component({
  imports: [AsyncPipe, DatePipe, CapitalizePipe],
  template: `
    <!-- Built-in pipes -->
    <p>{{ today | date:'fullDate' }}</p>
    <p>{{ price | currency:'USD' }}</p>

    <!-- Custom pipe -->
    <p>{{ 'hello' | capitalize }}</p>

    <!-- Async pipe -->
    <ul>
      <li *ngFor="let user of users$ | async">{{ user.name }}</li>
    </ul>
  `
})
export class PipeComponent {
  today = new Date();
  price = 1234.56;
  users$: Observable<User[]>;

  constructor(private apiService: ApiService) {
    this.users$ = this.apiService.getUsers();
  }
}

```

16. Forms (Template-driven and Reactive)

Template-driven Forms:

typescript

```
@Component({
  imports: [FormsModule],
  template: `
    <form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">
      <input name="name" [(ngModel)]="user.name" required>
      <button type="submit" [disabled]="userForm.invalid">Submit</button>
    </form>
  `
})
export class TemplateFormComponent {
  user = { name: '', email: '' };

  onSubmit(form: NgForm) {
    if (form.valid) {
      console.log('Form submitted:', this.user);
    }
  }
}
```

Reactive Forms:

typescript


```

@Component({
  imports: [ReactiveFormsModule],
  template: `
    <form [formGroup]="userForm" (ngSubmit)="onSubmit()">
      <input formControlName="name">
      <input formControlName="email">
      <button type="submit" [disabled]="userForm.invalid">Submit</button>
    </form>
  `
})
export class ReactiveFormComponent {
  userForm: FormGroup;

  constructor(private fb: FormBuilder) {
    this.userForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(2)]],
      email: ['', [Validators.required, Validators.email]]
    });
  }

  onSubmit() {
    if (this.userForm.valid) {
      console.log('Form:', this.userForm.value);
    }
  }
}

```

17. Angular Directives

Custom Directive - Modify DOM behavior:

```

typescript

```

```

@Directive({
  selector: '[appHighlight]',
  standalone: true
})
export class HighlightDirective {
  @Input() appHighlight = "";

  @HostListener('mouseenter', ['$event'])
  onMouseEnter() {
    this.highlight(this.appHighlight || 'yellow');
  }

  @HostListener('mouseleave', ['$event'])
  onMouseLeave() {
    this.highlight("");
  }

  private highlight(color: string) {
    this.el.nativeElement.style.backgroundColor = color;
  }

  constructor(private el: ElementRef) {}
}

// Usage in template
@Component({
  template: `
    <p appHighlight="lightblue">Hover me! </p>
    <p [appHighlight]="color">Dynamic color </p>
  `
})
export class DirectiveComponent {
  color = 'lightgreen';
}

```

18. Effects and DOM Manipulation

Effects - React to signal changes:

typescript

```

@Component({
  template: `
    <div #myDiv>
      <p>Count: {{ count() }}</p>
      <button (click)="increment()">+ </button>
    </div>
  `,
})
export class EffectComponent {
  @ViewChild('myDiv') myDiv!: ElementRef;

  count = signal(0);

  constructor() {
    // Effect runs when count changes
    effect(() => {
      console.log('Count is now:', this.count());

      // Side effects
      if (this.count() > 5) {
        document.title = `High count: ${this.count()}`;
      }
    });
  }

  // DOM manipulation after view init
  ngAfterViewInit() {
    this.myDiv.nativeElement.style.border = '2px solid blue';
  }

  increment() {
    this.count.update(c => c + 1);
  }
}

```

19. Angular CDK

CDK - Component Development Kit utilities:

```
bash
```

```
npm install @angular/cdk
```

typescript

// Drag & Drop

```
@Component({
  imports: [DragDropModule],
  template: `
    <div cdkDropList (cdkDropListDropped)="drop($event)">
      <div *ngFor="let item of items" cdkDrag>
        {{ item }}
      </div>
    </div>
  `,
})
export class DragDropComponent {
  items = ['Item 1', 'Item 2', 'Item 3'];

  drop(event: CdkDragDrop<string[]>) {
    moveItemInArray(this.items, event.previousIndex, event.currentIndex);
  }
}
```

// Virtual Scrolling for large lists

```
@Component({
  imports: [ScrollingModule],
  template: `
    <cdk-virtual-scroll-viewport itemSize="50" class="viewport">
      <div *cdkVirtualFor="let item of items">{{ item }}</div>
    </cdk-virtual-scroll-viewport>
  `,
  styles: ['.viewport { height: 200px; }']
})
export class VirtualScrollComponent {
  items = Array.from({length: 10000}, (_, i) => `Item ${i + 1}`);
}
```

20. E2E Testing with Cypress

Cypress - End-to-end testing framework:

bash

`npm install --save-dev cypress`

`npx cypress open`

typescript

```
// cypress/e2e/app.cy.ts
describe('App E2E Tests', () => {
  beforeEach(() => {
    cy.visit('/');
  });

  it('should display welcome message', () => {
    cy.contains('Welcome');
  });

  it('should navigate to about page', () => {
    cy.get('[data-cy="about-link"]').click();
    cy.url().should('include', '/about');
    cy.contains('About Us');
  });

  it('should submit form', () => {
    cy.get('[data-cy="name-input"]').type('John Doe');
    cy.get('[data-cy="email-input"]').type('john@example.com');
    cy.get('[data-cy="submit-btn"]').click();
    cy.contains('Form submitted successfully');
  });

  it('should load user list', () => {
    cy.intercept('GET', '/api/users', { fixture: 'users.json' });
    cy.get('[data-cy="load-users"]').click();
    cy.get('[data-cy="user-list"]').should('contain', 'John Doe');
  });
});

// cypress/fixtures/users.json
[
  { "id": 1, "name": "John Doe", "email": "john@example.com" },
  { "id": 2, "name": "Jane Smith", "email": "jane@example.com" }
]
```

21. Progressive Web App (PWA)

PWA - Make your Angular app installable:

bash

ng add @angular/pwa

This adds:

- Service Worker for caching
- Web App Manifest
- App icons
- Offline functionality

typescript

// Check for updates

```
@Component({
  template: `
    <button *ngIf="updateAvailable" (click)="updateApp()">
      Update Available
    </button>
  `,
})
export class AppComponent {
  updateAvailable = false;

  constructor(private swUpdate: SwUpdate) {
    if (swUpdate.isEnabled) {
      swUpdate.versionUpdates.subscribe(evt => {
        if (evt.type === 'VERSION_READY') {
          this.updateAvailable = true;
        }
      });
    }
  }

  updateApp() {
    window.location.reload();
  }
}
```

22. Lazy Loading

Lazy Loading - Load modules on demand:

typescript

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

// Route guards for protection
@Injectable()
export class AuthGuard implements CanActivate {
  constructor(private auth: AuthService, private router: Router) {}

  canActivate(): boolean {
    if (this.auth.isAuthenticated()) {
      return true;
    }
    this.router.navigate(['/login']);
    return false;
  }
}
```

23. Server-Side Rendering (SSR)

SSR - Render Angular on the server:

```
bash

ng add @nguniversal/express-engine
npm run build:ssr
npm run serve:ssr
```

```
typescript
```

```

// app.server.module.ts
@NgModule({
  imports: [AppModule, ServerModule],
  bootstrap: [AppComponent]
})
export class AppServerModule {}

// Check platform in components
@Component({
  template: `<div>{{ message }}</div>`
})
export class SSRComponent implements OnInit {
  message = "";

  constructor(@Inject(PLATFORM_ID) private platformId: Object) {}

  ngOnInit() {
    if (isPlatformBrowser(this.platformId)) {
      // Browser-specific code
      this.message = 'Running in browser';
    } else {
      // Server-specific code
      this.message = 'Running on server';
    }
  }
}

```

24. Testing

Unit Testing with Jasmine and Karma:

```

typescript

```



```
// component.spec.ts
```

```
describe('CounterComponent', () => {  
  let component: CounterComponent;  
  let fixture: ComponentFixture<CounterComponent>;  
  
  beforeEach(() => {  
    TestBed.configureTestingModule({  
      imports: [CounterComponent]  
    });  
    fixture = TestBed.createComponent(CounterComponent);  
    component = fixture.componentInstance;  
  });  
  
  it('should create', () => {  
    expect(component).toBeTruthy();  
  });  
  
  it('should increment count', () => {  
    component.increment();  
    expect(component.count()).toBe(1);  
  });  
  
  it('should display count in template', () => {  
    component.count.set(5);  
    fixture.detectChanges();  
    const compiled = fixture.nativeElement;  
    expect(compiled.querySelector('p').textContent).toContain('Count: 5');  
  });  
});
```

```
// Service testing
```

```
describe('DataService', () => {  
  let service: DataService;  
  let httpMock: HttpTestingController;  
  
  beforeEach(() => {  
    TestBed.configureTestingModule({  
      imports: [HttpClientTestingModule],  
      providers: [DataService]  
    });  
    service = TestBed.inject(DataService);  
    httpMock = TestBed.inject(HttpTestingController);  
  });
```

```
it('should fetch users', () => {  
  const mockUsers = [{ id: 1, name: 'John' }];  
  
  service.getUsers().subscribe(users => {  
    expect(users).toEqual(mockUsers);  
  });  
  
  const req = httpMock.expectOne('/api/users');  
  expect(req.request.method).toBe('GET');  
  req.flush(mockUsers);  
});  
});
```

Quick Reference Commands

```
bash  
  
# Project setup  
npm i -g @angular/cli  
ng new my-app  
cd my-app && ng serve  
  
# Generate components  
ng g c components/header  
ng g s services/api  
ng g p pipes/filter  
  
# Build and deploy  
ng build --prod  
ng test  
ng e2e  
  
# Add packages  
ng add @angular/material  
ng add @angular/pwa  
ng add @ngrx/store
```

This guide covers all major Angular concepts with minimal explanations and practical code examples. Each section provides the essential information needed to implement Angular features effectively.