Complete Angular Tutorial Guide

Table of Contents

---

1. Architectural Flow & File Structure

Angular Project Structure

```
my-angular-app/
├── src/
│   ├── app/
│   │   ├── components/
│   │   │   ├── header/
│   │   │   │   ├── header.component.ts
│   │   │   │   ├── header.component.html
│   │   │   │   └── header.component.css
│   │   ├── services/
│   │   │   └── data.service.ts
│   │   ├── models/
│   │   │   └── user.model.ts
│   │   ├── app.component.ts
│   │   ├── app.component.html
│   │   ├── app.module.ts
│   │   └── app-routing.module.ts
│   ├── assets/
│   ├── environments/
│   ├── index.html
│   └── main.ts
├── angular.json
├── package.json
└── tsconfig.json
```

Architecture Flow

1. **main.ts** → Bootstrap the application
2. **app.module.ts** → Root module configuration
3. **app.component.ts** → Root component
4. **Components** → UI building blocks
5. **Services** → Business logic & data
6. **Models** → Data structures

---

2. Constructor & Initialization

Basic Constructor

```typescript
export class UserComponent {
  name: string;
  age: number;

  constructor() {
    this.name = 'John Doe';
    this.age = 25;
    console.log('Component initialized');
  }
}
```

Constructor with Dependencies

typescript

```
import { Component } from '@angular/core';
import { UserService } from './user.service';

@Component({
  selector: 'app-user',
  templateUrl: './user.component.html'
})
export class UserComponent {
  users: any[] = [];

  constructor(private userService: UserService) {
    this.loadUsers();
  }

  private loadUsers() {
    this.users = this.userService.getUsers();
  }
}
```

---

3. Constructor Array & Object Declaration

Array Declaration

typescript

```
export class DataComponent {
  // Different ways to declare arrays
  numbers: number[] = [];
  names: string[] = [];
  users: any[] = [];

  constructor() {
    // Initialize arrays using new
    this.numbers = new Array(1, 2, 3, 4, 5);
    this.names = new Array('Alice', 'Bob', 'Charlie');

    // Initialize with empty arrays
    this.users = new Array();
  }
}
```

Object Declaration

```
typescript
```

```
interface User {
  id: number;
  name: string;
  email: string;
}

export class UserComponent {
  user: User = {} as User;
  userList: User[] = [];

  constructor() {
    // Initialize object using new
    this.user = new Object({
      id: 1,
      name: 'John',
      email: 'john@example.com'
    }) as User;

    // Initialize with object literal
    this.user = {
      id: 1,
      name: 'John',
      email: 'john@example.com'
    };

    // Initialize array with new
    this.userList = new Array<User>();
  }
}
```

---

4. Try-Catch Error Handling

Basic Try-Catch

```
typescript
```

```
export class ErrorHandlingComponent {
  data: any;

  constructor() {
    this.loadData();
  }

  loadData() {
    try {
      // Risky operation
      const result = JSON.parse('{"name": "John"}');
      this.data = result;
      console.log('Data loaded successfully');
    } catch (error) {
      console.error('Error loading data:', error);
      this.data = null;
    }
  }
}
```

Async Try-Catch

```
typescript
```

```
import { HttpClient } from '@angular/common/http';

export class ApiComponent {
  constructor(private http: HttpClient) {}

  async fetchData() {
    try {
      const response = await this.http.get<any>('api/users').toPromise();
      console.log('Data fetched:', response);
      return response;
    } catch (error) {
      console.error('API Error:', error);
      throw new Error('Failed to fetch data');
    }
  }

  // With finally block
  processData() {
    try {
      // Processing logic
      console.log('Processing...');
    } catch (error) {
      console.error('Processing failed:', error);
    } finally {
      console.log('Cleanup completed');
    }
  }
}
```

---

5. Adding TypeScript & Declarations

Interface Declaration

```
typescript
```

```
// user.interface.ts
export interface User {
  id: number;
  name: string;
  email: string;
  isActive?: boolean; // Optional property
}
```

Type Declaration

typescript

```
// types.ts
export type Status = 'active' | 'inactive' | 'pending';
export type UserRole = 'admin' | 'user' | 'moderator';

// Generic type
export type ApiResponse<T> = {
  success: boolean;
  data: T;
  message: string;
};
```

Class with TypeScript

typescript

```
export class UserManager {
  private users: User[] = [];
  readonly maxUsers: number = 100;

  constructor(private apiUrl: string) {}

  addUser(user: User): void {
    if (this.users.length < this.maxUsers) {
      this.users.push(user);
    }
  }

  getUserById(id: number): User | undefined {
    return this.users.find(user => user.id === id);
  }

  // Generic method
  processData<T>(data: T): T {
    console.log('Processing:', data);
    return data;
  }
}
```

---

6. Input Decorator (@Input)

Basic Input Usage

typescript

```
// child.component.ts
import { Component, Input } from '@angular/core';

@Component({
  selector: 'app-child',
  template: `
    <div>
      <h3>{{ name }}</h3>
      <p>Age: {{ age }}</p>
      <p>Active: {{ isActive ? 'Yes' : 'No' }}</p>
    </div>
  `
})
export class ChildComponent {
  @Input() name: string = '';
  @Input() age: number = 0;
  @Input() isActive: boolean = false;
  @Input() user?: User; // Optional input
}
```

Parent Component Usage

```
typescript
```

```
// parent.component.ts
@Component({
  selector: 'app-parent',
  template: `
    <app-child
      [name]="userName"
      [age]="userAge"
      [isActive]="true"
      [user]="currentUser">
    </app-child>
  `
})
export class ParentComponent {
  userName = 'Alice';
  userAge = 30;
  currentUser: User = {
    id: 1,
    name: 'Alice',
    email: 'alice@example.com'
  };
}
```

Input with Validation

typescript

```typescript
import { Component, Input, OnChanges, SimpleChanges } from '@angular/core';

@Component({
  selector: 'app-validated-input',
  template: `<div>Valid: {{ isValid }}</div>`
})
export class ValidatedInputComponent implements OnChanges {
  @Input() email: string = '';
  isValid: boolean = false;

  ngOnChanges(changes: SimpleChanges) {
    if (changes['email']) {
      this.validateEmail();
    }
  }

  private validateEmail() {
    const emailRegex = /^[^\s@]+@[^\s@]+\.[^\s@]+$/;
    this.isValid = emailRegex.test(this.email);
  }
}
```

---

7. Component Decorator (@Component)

Basic Component

typescript

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

@Component({
  selector: 'app-basic',
  templateUrl: './basic.component.html',
  styleUrls: ['./basic.component.css']
})
export class BasicComponent {
  title = 'Basic Component';
}
```

Inline Template Component

typescript

```
@Component({
  selector: 'app-inline',
  template: `
    <div class="container">
      <h2>{{ title }}</h2>
      <button (click)="onClick()">Click Me</button>
      <p *ngIf="showMessage">{{ message }}</p>
    </div>
  `,
  styles: [`
    .container {
      padding: 20px;
      background-color: #f0f0f0;
    }
    h2 {
      color: #333;
    }
  `]
})
export class InlineComponent {
  title = 'Inline Component';
  message = 'Hello World!';
  showMessage = false;

  onClick() {
    this.showMessage = !this.showMessage;
  }
}
```

Component with Providers

typescript

```
@Component({
  selector: 'app-service',
  template: `<div>{{ data }}</div>`,
  providers: [DataService] // Component-level service
})
export class ServiceComponent {
  data: string;

  constructor(private dataService: DataService) {
    this.data = this.dataService.getData();
  }
}
```

---

8. Template Literals

Basic Template Literals

typescript

```
@Component({
  selector: 'app-template-literal',
  template: `
    <div>
      <h2>Welcome, {{ userName }}!</h2>
      <p>You have {{ messageCount }} new messages</p>
      <div [innerHTML]="getFormattedMessage()"></div>
    </div>
  `
})
export class TemplateLiteralComponent {
  userName = 'John';
  messageCount = 5;

  getFormattedMessage(): string {
    return `
      <div>
        <strong>Hello ${this.userName}!</strong>
        <br>
        <em>You have ${this.messageCount} notifications</em>
      </div>
    `;
  }

  // Multi-line template literal
  generateReport(): string {
    return `
      Report for: ${this.userName}
      Date: ${new Date().toLocaleDateString()}
      Messages: ${this.messageCount}
      Status: Active
    `;
  }
}
```

Dynamic Template Literals

typescript

```
export class DynamicTemplateComponent {
  users = ['Alice', 'Bob', 'Charlie'];

  generateUserList(): string {
    return `
      <ul>
        ${this.users.map(user => `<li>${user}</li>`).join('')}
      </ul>
    `;
  }

  createEmailTemplate(name: string, subject: string): string {
    return `
      Dear ${name},

      Subject: ${subject}

      Thank you for your interest.

      Best regards,
      The Team
    `;
  }
}
```

9. Data Attributes

Basic Data Attributes

```
typescript
```

```
@Component({
  selector: 'app-data-attributes',
  template: `
    <div>
      <!-- Static data attributes -->
      <button
        data-action="save"
        data-id="123"
        data-category="primary"
        (click)="handleClick($event)">
        Save Data
      </button>

      <!-- Dynamic data attributes -->
      <div
        [attr.data-user-id]="currentUserId"
        [attr.data-status]="userStatus"
        [attr.data-role]="userRole">
        User Info
      </div>

      <!-- Loop with data attributes -->
      <div *ngFor="let item of items">
        <span
          [attr.data-item-id]="item.id"
          [attr.data-item-type]="item.type">
          {{ item.name }}
        </span>
      </div>
    </div>
  `
})
export class DataAttributesComponent {
  currentUserId = 'user_123';
  userStatus = 'active';
  userRole = 'admin';

  items = [
    { id: 1, name: 'Item 1', type: 'product' },
    { id: 2, name: 'Item 2', type: 'service' }
  ];

  handleClick(event: Event) {
    const button = event.target as HTMLButtonElement;
    const action = button.dataset['action'];
    const id = button.dataset['id'];
    const category = button.dataset['category'];

    console.log('Action:', action);
    console.log('ID:', id);
    console.log('Category:', category);
  }
}
```

Custom Data Attribute Directive

```typescript
import { Directive, ElementRef, Input } from '@angular/core';

@Directive({
  selector: '[appDataTracker]'
})
export class DataTrackerDirective {
  @Input() set appDataTracker(value: string) {
    this.el.nativeElement.setAttribute('data-tracker', value);
  }

  constructor(private el: ElementRef) {}
}

// Usage in component
@Component({
  template: `
    <button appDataTracker="button-click-analytics">
      Track This Button
    </button>
  `
})
export class TrackingComponent {}
```

---

10. Form Group & NgSubmit

Reactive Forms Setup

typescript

```
import { Component } from '@angular/core';
import { FormGroup, FormBuilder, Validators } from '@angular/forms';

@Component({
  selector: 'app-form',
  template: `
    <form [formGroup]="userForm" (ngSubmit)="onSubmit()">
      <div>
        <label for="name">Name:</label>
        <input
          id="name"
          formControlName="name"
          type="text"
          [class.error]="isFieldInvalid('name')">
        <div *ngIf="isFieldInvalid('name')" class="error-message">
          Name is required
        </div>
      </div>

      <div>
        <label for="email">Email:</label>
        <input
          id="email"
          formControlName="email"
          type="email"
          [class.error]="isFieldInvalid('email')">
        <div *ngIf="isFieldInvalid('email')" class="error-message">
          Valid email is required
        </div>
      </div>

      <div>
        <label for="age">Age:</label>
        <input
          id="age"
          formControlName="age"
          type="number"
          [class.error]="isFieldInvalid('age')">
      </div>

      <button
        type="submit"
        [disabled]="userForm.invalid">
      Submit
    </button>
  </form>

  <div *ngIf="submitted">
    <h3>Form Data:</h3>
    <pre>{{ formData | json }}</pre>
  </div>
 `
})
export class FormComponent {
  userForm: FormGroup;
  submitted = false;
  formData: any;

  constructor(private fb: FormBuilder) {
    this.userForm = this.fb.group({
      name: ['', [Validators.required, Validators.minLength(2)]],
      email: ['', [Validators.required, Validators.email]],
      age: ['', [Validators.required, Validators.min(18)]]
    });
  }

  onSubmit() {
    if (this.userForm.valid) {
      this.formData = this.userForm.value;
      this.submitted = true;
      console.log('Form submitted:', this.formData);
    } else {
      this.markAllFieldsAsTouched();
    }
  }

  isFieldInvalid(fieldName: string): boolean {
    const field = this.userForm.get(fieldName);
    return !!(field && field.invalid && (field.dirty || field.touched));
  }

  private markAllFieldsAsTouched() {
    Object.keys(this.userForm.controls).forEach(key => {
      this.userForm.get(key)?.markAsTouched();
    });
  }
}
```

Template Driven Forms

```
typescript
```

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

@Component({
  selector: 'app-template-form',
  template: `
    <form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">
      <div>
        <label for="username">Username:</label>
        <input
          id="username"
          name="username"
          [(ngModel)]="user.username"
          #username="ngModel"
          required
          minlength="3">
        <div *ngIf="username.invalid && username.touched">
          Username is required (min 3 characters)
        </div>
      </div>

      <div>
        <label for="password">Password:</label>
        <input
          id="password"
          name="password"
          type="password"
          [(ngModel)]="user.password"
          #password="ngModel"
          required>
      </div>

      <button type="submit" [disabled]="userForm.invalid">
        Login
      </button>
    </form>
  `
})
export class TemplateFormComponent {
  user = {
    username: '',
    password: ''
  };
  onSubmit(form: any) {
    if (form.valid) {
      console.log('Login attempt:', this.user);
    }
  }
}
```

---

11. Callbacks & Promises

Basic Callbacks

```
typescript
```

```
export class CallbackComponent {
  data: any[] = [];

  constructor() {
    this.loadData(this.onDataLoaded.bind(this));
  }

  // Method that accepts callback
  loadData(callback: (data: any[]) => void) {
    setTimeout(() => {
      const mockData = [
        { id: 1, name: 'Item 1' },
        { id: 2, name: 'Item 2' }
      ];
      callback(mockData);
    }, 1000);
  }

  // Callback function
  onDataLoaded(data: any[]) {
    this.data = data;
    console.log('Data loaded:', data);
  }

  // Error handling callback
  performOperation(
    successCallback: (result: string) => void,
    errorCallback: (error: string) => void
  ) {
    const success = Math.random() > 0.5;

    if (success) {
      successCallback('Operation completed successfully');
    } else {
      errorCallback('Operation failed');
    }
  }
}
```

Promises

```
typescript
```

```
export class PromiseComponent {
  data: any;
  loading = false;
  error: string | null = null;

  constructor() {
    this.loadDataWithPromise();
  }

  // Basic Promise
  loadDataWithPromise() {
    this.loading = true;
    this.error = null;

    const promise = new Promise<any>((resolve, reject) => {
      setTimeout(() => {
        const success = Math.random() > 0.3;
        if (success) {
          resolve({ message: 'Data loaded successfully' });
        } else {
          reject('Failed to load data');
        }
      }, 2000);
    });

    promise
      .then(data => {
        this.data = data;
        console.log('Promise resolved:', data);
      })
      .catch(error => {
        this.error = error;
        console.error('Promise rejected:', error);
      })
      .finally(() => {
        this.loading = false;
      });
  }

  // Async/Await with Promise
  async fetchUserData(userId: number): Promise<any> {
    try {
      const userData = await this.getUserPromise(userId);
      console.log('User data fetched:', userData);
      return userData;
    } catch (error) {
      console.error('Error fetching user:', error);
      throw error;
    }
  }

  private getUserPromise(userId: number): Promise<any> {
    return new Promise((resolve, reject) => {
      setTimeout(() => {
        if (userId > 0) {
          resolve({
            id: userId,
            name: `User ${userId}`,
            email: `user${userId}@example.com`
          });
        } else {
          reject('Invalid user ID');
        }
      }, 1000);
    });
  }

  // Promise chaining
  processDataChain() {
    this.fetchData()
      .then(data => this.transformData(data))
      .then(transformedData => this.saveData(transformedData))
      .then(result => console.log('Process complete:', result))
      .catch(error => console.error('Process failed:', error));
  }

  private fetchData(): Promise<any> {
    return Promise.resolve({ raw: 'data' });
  }

  private transformData(data: any): Promise<any> {
    return Promise.resolve({ ...data, transformed: true });
  }

  private saveData(data: any): Promise<string> {
    return Promise.resolve('Data saved successfully');
  }
}
```

---

12. Nested Routes

App Routing Module

```typescript
// app-routing.module.ts
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { UserComponent } from './user/user.component';
import { UserListComponent } from './user/user-list/user-list.component';
import { UserDetailComponent } from './user/user-detail/user-detail.component';
import { UserEditComponent } from './user/user-edit/user-edit.component';

const routes: Routes = [
  { path: '', redirectTo: '/home', pathMatch: 'full' },
  { path: 'home', component: HomeComponent },
  {
    path: 'users',
    component: UserComponent,
    children: [
      { path: '', component: UserListComponent },
      { path: 'new', component: UserEditComponent },
      { path: ':id', component: UserDetailComponent },
      { path: ':id/edit', component: UserEditComponent }
    ]
  },
  { path: '**', redirectTo: '/home' } // Wildcard route
];

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

Parent Component with Router Outlet

```
typescript
```

```
// user.component.ts
import { Component } from '@angular/core';

@Component({
  selector: 'app-user',
  template: `
    <div class="user-container">
      <nav>
        <a routerLink="/users" routerLinkActive="active" [routerLinkActiveOptions]="{exact: true}">Users</a>
        <a routerLink="/users/new" routerLinkActive="active">New User</a>
      </nav>

      <div class="content">
        <router-outlet></router-outlet>
      </div>
    </div>
  `,
  styles: [`
    .user-container {
      display: flex;
      flex-direction: column;
    }
    nav {
      padding: 10px;
      background-color: #f5f5f5;
    }
    nav a {
      margin-right: 10px;
      padding: 5px 10px;
      text-decoration: none;
    }
    nav a.active {
      background-color: #007bff;
      color: white;
    }
    .content {
      flex: 1;
      padding: 20px;
    }
  `]
})
export class UserComponent {}
```

Child Components with Navigation

typescript

```typescript
// user-list.component.ts
import { Component } from '@angular/core';
import { Router } from '@angular/router';

@Component({
  selector: 'app-user-list',
  template: `
    <div>
      <h2>User List</h2>
      <button (click)="createUser()">Create New User</button>

      <div *ngFor="let user of users" class="user-item">
        <span>{{ user.name }}</span>
        <button (click)="viewUser(user.id)">View</button>
        <button (click)="editUser(user.id)">Edit</button>
      </div>
    </div>
  `
})
export class UserListComponent {
  users = [
    { id: 1, name: 'John Doe' },
    { id: 2, name: 'Jane Smith' }
  ];

  constructor(private router: Router) {}

  createUser() {
    this.router.navigate(['/users/new']);
  }

  viewUser(id: number) {
    this.router.navigate(['/users', id]);
  }

  editUser(id: number) {
    this.router.navigate(['/users', id, 'edit']);
  }
}
```

Route Parameters & Query Parameters

typescript

```
// user-detail.component.ts
import { Component, OnInit } from '@angular/core';
import { ActivatedRoute, Router } from '@angular/router';

@Component({
  selector: 'app-user-detail',
  template: `
    <div>
      <h2>User Details</h2>
      <div *ngIf="user">
        <p><strong>ID:</strong> {{ user.id }}</p>
        <p><strong>Name:</strong> {{ user.name }}</p>
        <p><strong>Email:</strong> {{ user.email }}</p>
      </div>

      <button (click)="editUser()">Edit User</button>
      <button (click)="goBack()">Back to List</button>
    </div>
  `
})
export class UserDetailComponent implements OnInit {
  user: any;
  userId: number = 0;

  constructor(
    private route: ActivatedRoute,
    private router: Router
  ) {}

  ngOnInit() {
    // Get route parameter
    this.route.params.subscribe(params => {
      this.userId = +params['id'];
      this.loadUser();
    });

    // Get query parameters
    this.route.queryParams.subscribe(queryParams => {
      console.log('Query params:', queryParams);
    });
  }

  loadUser() {
    // Mock user data
    this.user = {
      id: this.userId,
      name: `User ${this.userId}`,
      email: `user${this.userId}@example.com`
    };
  }

  editUser() {
    this.router.navigate(['/users', this.userId, 'edit']);
  }

  goBack() {
    this.router.navigate(['/users']);
  }
}
```

---

13. Debugging

Component Debugging Setup

```
typescript
```

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

@Component({
  selector: 'app-debug',
  template: `
    <div>
      <h2>Debug Component</h2>
      <p>Counter: {{ counter }}</p>
      <button (click)="increment()">Increment</button>
      <button (click)="triggerError()">Trigger Error</button>

      <div *ngFor="let item of items; trackBy: trackByFn">
        {{ item.name }}
      </div>
    </div>
  `
})
export class DebugComponent implements OnInit, OnDestroy {
  counter = 0;
  items = [
    { id: 1, name: 'Item 1' },
    { id: 2, name: 'Item 2' }
  ];

  ngOnInit() {
    console.log('DebugComponent initialized');
    console.log('Initial state:', {
      counter: this.counter,
      items: this.items
    });
  }

  ngOnDestroy() {
    console.log('DebugComponent destroyed');
  }

  increment() {
    console.log('Before increment:', this.counter);
    this.counter++;
    console.log('After increment:', this.counter);

    // Debug performance
    console.time('increment-operation');
    // Some operation
    console.timeEnd('increment-operation');
  }

  triggerError() {
    try {
      throw new Error('Intentional error for debugging');
    } catch (error) {
      console.error('Caught error:', error);
      console.trace('Stack trace:');
    }
  }

  trackByFn(index: number, item: any) {
    console.log('TrackBy called for:', item);
    return item.id;
  }

  // Debug method for inspecting objects
  debugObject(obj: any, label: string = 'Object') {
    console.group(label);
    console.log('Type:', typeof obj);
    console.log('Value:', obj);
    console.log('JSON:', JSON.stringify(obj, null, 2));
    console.groupEnd();
  }
}
```

Service Debugging

```
typescript
```

```
import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { tap, catchError } from 'rxjs/operators';
import { throwError } from 'rxjs';

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

  constructor(private http: HttpClient) {
    console.log('DebugService created');
  }

  fetchData() {
    console.log('Fetching data from:', this.apiUrl);

    return this.http.get(`${this.apiUrl}/data`).pipe(
      tap(response => {
        console.log('API Response received:', response);
        this.debugObject(response, 'API Response');
      }),
      catchError(error => {
        console.error('API Error:', error);
        console.log('Error details:', {
          status: error.status,
          message: error.message,
          url: error.url
        });
        return throwError(error);
      })
    );
  }

  private debugObject(obj: any, label: string) {
    console.group(`🔍 ${label}`);
    console.table(obj);
    console.groupEnd();
  }
}
```

14. Console.log()

Basic Console Logging

```typescript
export
```