

# Contents

<b>Angular</b>	<b>1</b>
Project Structure . . . . .	1
Root . . . . .	1
/src . . . . .	2
/app . . . . .	2
Modules . . . . .	2
/app Code . . . . .	2
Component Code . . . . .	4
HTML Syntax . . . . .	5
Router . . . . .	5
Flow Control . . . . .	5
Expressions . . . . .	5
Styling . . . . .	6
TypeScript Syntax . . . . .	6
Basics . . . . .	6
Models . . . . .	7
Modules . . . . .	8
Services . . . . .	9
General Definition . . . . .	9
Usage . . . . .	10
Auth . . . . .	10
Debounce . . . . .	11
Forms . . . . .	11
Template Driven Forms . . . . .	11
Reactive Forms . . . . .	13
Subcomponents . . . . .	14
Component Lifecycle . . . . .	14

## Angular

Angular is a web framework for developing fast and reliable web applications based on TypeScript.

### Project Structure

#### Root

path	features
./	Konfigurationsdateien / ENV
./public	static file serving
./src	source

/src

path	features
./src/styles.css	global CSS
./src/main.ts	bootstrapper
./src/index.html	HTML wrapper without body
./src/app	app code

/app

path	features
./app.component.ts	main component
./app.component.html	main component html
./app.config.ts	app configuration
./app.routes.ts	router config
./app/components/componentName	component folder
./app/services/serviceName.ts	service
./app/models/modelName.ts	model

Component folder contains .ts, .html, .spec.ts & .css

## Modules

./app/module/feature/\* contains a feature module with component.ts, service.ts, module.ts & feature-routing.ts

/app Code

main.ts

```
import { bootstrapApplication } from '@angular/platform-browser';
import { appConfig } from './app/app.config';
import { AppComponent } from './app/app.component';
```

```
bootstrapApplication(AppComponent, appConfig)
  .catch((err) => console.error(err));
```

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>Demos</title>
```

```

    <base href="/">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link rel="icon" type="image/x-icon" href="favicon.ico">
    <link rel="preconnect" href="https://fonts.googleapis.com">
    <link rel="preconnect" href="https://fonts.gstatic.com/" crossorigin>
    <link href="https://fonts.googleapis.com/..." rel="stylesheet">
  </head>
  <body>
    <app-root></app-root>
  </body>
</html>

```

#### app/app.component.ts

```

import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
import { TestComponent } from '../components/test/test.component';

@Component({
  selector: 'app-root',
  standalone: true,
  imports: [
    RouterOutlet, TestComponent
  ],
  /*template:
    <router-outlet></router-outlet> <- Alternative to html file
    <app-test></app-test>
  */
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})

export class AppComponent {
  title = 'demos';
}

```

#### app/app.component.html

```

<div class="container"> global styling
  <!--enable navigation between components-->
  <router-outlet />

  <!--auto redirect to component1 in router config-->

</div>

```

```

<!--optional component binding, if not using router + redirect:-->
<app-test></app-test>

```

#### app/app.config.ts

```

import { ApplicationConfig, provideZoneChangeDetection } from '@angular/core';
import { provideRouter } from '@angular/router';
import { provideHttpClient } from '@angular/common/http';
import { routes } from './app.routes';

export const appConfig: ApplicationConfig = {
  providers: [
    provideZoneChangeDetection({ eventCoalescing: true }),
    provideRouter(routes),
    provideHttpClient()
  ]
};

```

#### app/app.routes.ts

```

import { Routes } from '@angular/router';
import { TestComponent } from './components/test/test.component';
import { AuthGuard } from './services/auth.guard';

const routeConfig: Routes = [
  { path: '', redirectTo: '/component1', pathMatch: 'full' },
  { path: 'component1', component: TestComponent, canActivate: [AuthGuard],
    children: [
      { path: 'subcomponent1', component: TestComponent },
      { path: 'subcomponent2/:token', component: TestComponent }
    ]
  },
  { path: '**', redirectTo: '' } // catch any unfound routes and redirect to home page
];

export default routeConfig;

```

#### Component Code

##### component.ts

```

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

@Component({
  selector: 'app-test',
  standalone: true,
  imports: [],

```

```

    templateUrl: './test.component.html',
    styleUrls: ['./test.component.css']
  })
  export class TestComponent {

  }

```

## HTML Syntax

### Router

Router outlet not need if already included in `app.component.html`

```

<router-outlet></router-outlet>
<a [routerLink]="['/component1']">
  <div></div>
</a>

```

### Flow Control

```

@if (var == null) {
  <div></div>
} @else if (var == 1) {
  <div></div>
} @else {
  <div></div>
}

```

```

<!-- track helps Angular identify unique items in a collection-->
@for (item of items; track item;) {} // track without custom id
@for (item of items; track item.id; let i = $index) {
  <div>
    i gives the index of the current iteration
    {{ item }}
    <div (click)="doSmt(item)"></div> // passed proper item reference to function
  </div>
}
@for (item of items; track item.id; let i = $index,
      first = $first, last = $last, even = $even, odd = $odd) {}

```

### Expressions

```

{{ 1+1 }}
{{ number }}
{{ service.doSmt() }}

```

## Styling

```
<div [ngStyle]="{'background-color': farbe}"></div>
<div [ngStyle]="{'background-color': 'red'}"></div>
<div [ngClass]="boolean ? 'class1' : 'class2'"></div>
```

## TypeScript Syntax

### Basics

```
import { Component, OnInit, inject } from '@angular/core';
import { Router } from '@angular/router';
import { ngStyle, ngClass } from '@angular/common';

import { Location } from '@angular/common';
import { someService } from "../../services/someService.service"

import { TestModuleComponent } from '../test-module.component'; // used in html

// meta info
@Component({
  selector: 'app-angular', // app-"componentname"
  standalone: true, // if true it does not need to be declared in NgModule
  imports: [
    // directive have to be imported here explicitly if used besides for typing
    TestModuleComponent, // for custom components
    ngStyle, // for inline styling
    ngClass, // for inline class
  ],
  templateUrl: './angular.component.html',
  styleUrls: ['./angular.component.css']
})
export class AngularComponent implements OnInit {
  public number: number = 0;
  public numbers: Example = new Example(3);
  public farbe: string = "red";

  public array1: number[];
  public array2: Array<number>;

  public notNull!: string; // not null assertion
  public firstNull: string | null = null;
  private number2: number = 0; // not accessible to html

  // alternative to constructor injection
  private serviceAlt: someService = inject(someService);
```

```

public constructor(
    private router: Router,
    // Location provides access to the browser's URL & navigation history
    private location: Location,
) {
    // constructor generally used for service objects (e.g. location, router)
    // can also be used for var inits
    // services are defined by dependency injection
    // services only exists once and follow singleton pattern
}

// OnInit is a lifecycle hook/method
public ngOnInit(): void {
    // advanced inits
    // component relevante inits
    // var inits
    // load data
    this.number = 1;
}

// function
public function(number: number): void {
    this.number = number;
}

// routing
public back(): void {
    this.location.back(); // return last path/window
}
public navigate() {
    this.router.navigate(["/route"]);
}

// interval
public interval() {
    const intervalID = setInterval(() => {}, 100); // in milliseconds
    clearInterval(intervalID);
}
}

```

## Models

Data definition for components / forms - Interfaces - Custom types - Classes

## Definition

```

export class Example {
  public num: number;

  public constructor(num: number){
    this.num = num;
  }

  public dosmth() {
    return;
  }
}

```

### Usage

```

import { Example } from "../../models/Example";

public numbers: Example = new Example(3);

```

### Modules

Container that organizes related code. - You can define your own modules  
 - Groups components, services and elements into a *cohesive unit* - Modular architecture enables **lazy loading** - Modules are reusable

### Definition

```

import { NgModule } from '@angular/core';

@NgModule ({
})
export class TestModule {
  public sayHello(): string {
    return 'Hello World!';
  }
}

```

### Usage

```

@Component({
  imports: [
    TestModule
  ],
});
export class ComponentLoadsModule() {
  public ngOnInit() {
    this.testModule.sayHello()
  }
}

```



```

    }
  }
}

```

## Services

The component uses a service to retrieve photo data from a server - A service is an object that only exists once (singleton pattern) - To define a service, the decorator “Injectable” is used - To use a service, typically the constructor of the using class defines a property of the service type

### General Definition

```

import { Injectable } from '@angular/core';
import { HttpClient } from '@angular/common/http';
import { Observable } from 'rxjs';

@Injectable({
  providedIn: 'root'
}) // make injectable/mark as injectable service
export class ApiService {
  private url = '';

  public constructor(
    private httpClient: HttpClient // get, delete, patch, post, put
  ) { }

  // simple example
  public get(): Observable<number[]> {
    return this.httpClient.get(this.url) as Observable<number[]>;
  }

  // transform data and pass observable to caller
  public getComplex(): Observable<boolean> {
    const url = "";
    const body = "";
    const observable = new Observable<boolean>(subscriber => {
      const serverCall = this.httpClient.post(url + "/", body); // this.httpClient.ge
      serverCall.subscribe({
        next: res => {
          console.log(res);
          subscriber.next(true); // yield result to caller of get()
        },
        error: err => {
          console.log(err);
          subscriber.next(false); // yield result to caller of get()
        }
      })
    })
  }
}

```

```

        });
    });

    return observable;
}
}

```

## Usage

```

import { apiService } from '../../../services/api.service.ts'

export class Component {
    public constructor(private service: apiService);

    public useService() {
        this.service.get()
            .subscribe({
                next: (result) => { /* success */ },
                error: (err) => { /* fail */ }
            });

        // alternative:
        this.service.get()
            .subscribe((res: boolean) => {
                if(res) { /* success */ }
                else { /* fail */ }
            });
    }
}

```

## Auth

```

import { Injectable } from '@angular/core';
import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, Router } from '@angular/router';
import { Observable } from 'rxjs';
import { AuthService } from '../auth.service';

@Injectable({
    providedIn: 'root'
})
export class AuthGuard implements CanActivate {
    constructor(private authService: AuthService, private router: Router) { }

    canActivate(next: ActivatedRouteSnapshot, state: RouterStateSnapshot)
        : Observable<boolean> | Promise<boolean> | boolean {
        if (this.authService.isLoggedIn()) {

```

```

        return true;
    }
    else {
        this.router.navigate(['/login']);
        return false;
    }
}
}

```

## Debounce

## Forms

### Template Driven Forms

- simple to set up and use
- suitable for smaller forms
- angular handles most logic automatically

### HTML:

- Forms require names for every input
- Property binding: if value changes in DOM, then in the Attribute to -> works **bidirectional**

```

<form role="form" #loginForm="ngForm">
  <input
    type="email"
    name="inputEmail"
    [(ngModel)]="email"
    #inputEmail="ngModel"
    required
    email
    (keyup)="onInputChange($event)">
</input>

```

<- !!! Important for errors  
 <- !!! for email validation

```

use like this: [(ngModel)]="var_name_in_component"
- used for bidirectional data binding
- ngModel requires a name attribute

```

```

#inputEmail="ngModel"
- #inputEmail is value of name attribute
- creates a reference to ngModel directive instance named inputEmail
- allows access to properties like pristine, valid, dirty
  (opposite of pristine, has been modified),
  touched (input has been focused, not about modified), errors

```

```

    ngModel directive only work if ngModel binding has been used

onedirectional binding:
<input type="" name="" [ngModel]="number" readonly></input>
    - useful for readonly inputs

event:
    - (keyup)="variable=$event"
    - (keyup)="function()"

error box:
@if (!inputEmail.pristine || inputEmail.valid)) {
    <div>
        pristine ist used if the box hasn't been touched
        (since empty inputs are considered invalid)
    </div>
    @if (inputEmail.errors?.['required']) {
        <div>ngModule directives auto. generate validators & error objects</div>
    }
    @if (inputEmail.errors?.['email']) {
        <div>Invalid email format!</div>
    }
}
alternative:
<div [hidden]="username.pristine || username.valid">err msg</div>

click event
<button (click)="formFunction(loginForm)" [disabled]="!loginForm.valid">
    loginForm is the name of the reference to the form with ngModule directive
    instance
</button>
</form>

<!--Form control information:-->
<div>form status: {{ loginForm.status }}</div>
@for (key of keys(loginForm); track key; let nr = $index) {
    <div>{{ nr }}</div>
    <div>{{ key }}</div> <- key entspricht name attribute in Komponente
    <div>{{ loginForm.controls[key].status }}</div>
    <div>{{ loginForm.controls[key].pristine }}</div>
}

<!--Conditional disabled attribute for buttons-->
<div [disabled]="smt <= 5"></div>

```

TS:

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

import { FormsModule } from '@angular/forms'; // template driven forms
import { NgForm } from '@angular/forms'; // template driven + form directive for type

@Component({
  selector: 'app-test',
  standalone: true,
  imports: [
    FormsModule, // for ngModel binding/directives
  ],
  templateUrl: './test.component.html',
  styleUrls: ['./test.component.css']
})

export class TestComponent {
  public email: string = "";

  public onChange(event: any) {
    // both works
    console.log(event.target.value);
    console.log(this.email);
  }

  // template driven form
  public formFunction(form: NgForm) {
    // ngModel obj, cant get form values directly by name attribute
    console.log('Form Submitted!', form.value.inputEmail);
    form.reset();
  }

  public keys(form: any): string[] {
    return Object.keys(form.controls); // returns object names as iterable
  }
}
```

## Reactive Forms

- offer more control
- for complex and dynamic forms
- better scalability and testability
- form login is implemented in component class

HTML:

```

<form [formGroup]="reactiveForm" (ngSubmit)="onSubmit()">
  <input formControlName="amount" type="number" required>
  @if (amount?.touched && amount?.invalid) {
    <p>
      @if (amount?.errors?.['required']) {
        <span>Amount is required. </span>
      }
      @if (amount?.errors?.['minZero']) {
        <span>Please enter a number > 0.</span>
      }
    </p>
  }
</form>

```

## Subcomponents

```

<app-test-module name="parameter"></app-test-module>

```

## Component Lifecycle

1. Component creation: `ngOnChanges()` -> `ngOnInit()`
2. Content projection: `ngAfterContentInit()` -> `ngAfterContentChecked()`
3. View Initialization: `ngAfterViewInit()` -> `ngAfterViewChecked()`
4. Change detection runs repeatedly: `ngDoCheck()` -> `ngAfterContentChecked()`  
-> `ngAfterViewChecked`
5. Component destruction: `ngOnDestroy()`