**Angular Notes**

* **Angular Command Line**

**Angular install**

npm install -g @angular/cli

ng new <project name>

cd <project name>

**run the files**

ng serve

**Add bootstrap files**

npm install bootstrap@4.6.0

**Add model class**

ng g class yourclass –type=model -–skip-tests

**Add ng lint**

ng add @angular-eslint/schematics

**Add ng pwa**

ng add @angular/pwa@12.0.0

**Add ngrx**

ng add @ngrx/store

**Add ngrx effects**

ng add @ngrx/effects

**Add devtools**

npm install --save-dev @ngrx/store-devtools

**Generate files**

ng g c <Component name>

ng g c -is <Component name> -–skip-tests(inline style)

ng g m <Module name> --routing

ng g cl <Class name>

ng g s <service name>

ng g p <pipe name>

ng g d <directive name>

ng g i <interface name>

**Rxjs update**

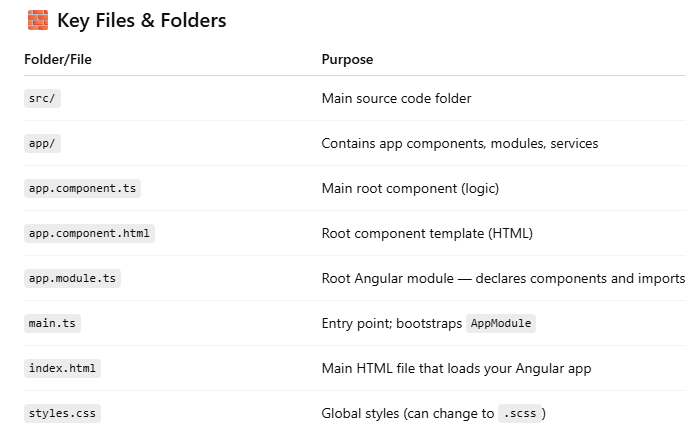
npm i rxjs@latest

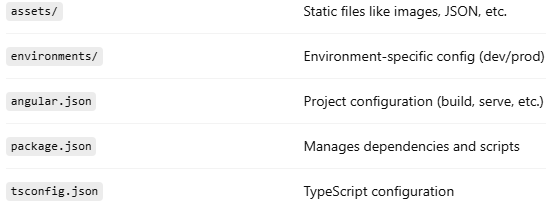
**guard**

ng g guard <guard name>

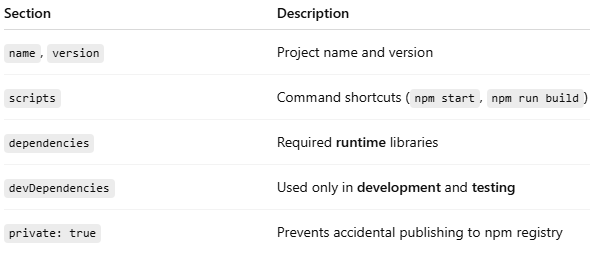
* **Angular Structure**



****

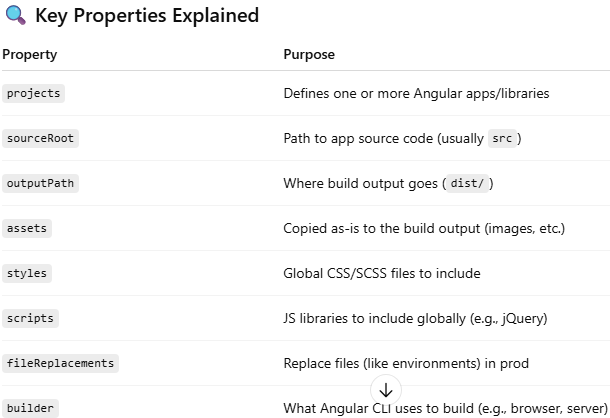
****

* **Anguar Package JSON**

****

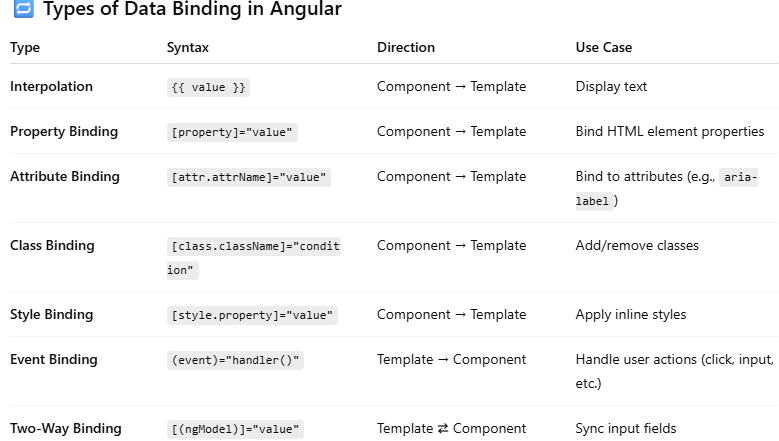
****

* **Angular Angular Json**

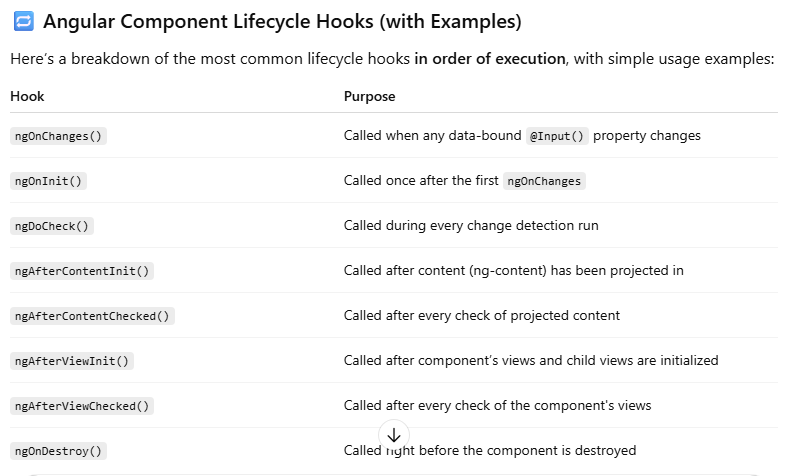
****

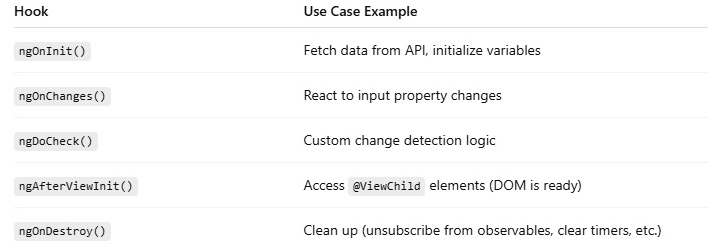
**** ****

* **Angular Databinding**

****

* **Angular Lifecycle**

****



* **Angular Component**

@Component({

selector: 'app-hello', // Used as use Element/Attribute/Class

templateUrl: './hello.component.html', // HTML template

template: `<h1>Hello, {{ name }}!</h1>` //Define html tag using backticks

styleUrls: ['./hello.component.css'] // CSS styles

styles: [` //Define all css using backticks

h1 {

color: green;

font-size: 24px;

}

`]

})



* **App Module**

@NgModule({

declarations: [

AppComponent, // declare all components, pipes, directives here

HelloComponent

],

// import necessary Angular modules like HttpClientModule, any module not using in lazyload etc

imports: [

BrowserModule,

FormsModule,

AppRoutingModule,

FormsModule,

ReactiveFormsModule,

HttpClientModule,

//StoreModule.forRoot(appReducer),

//StoreDevtoolsModule.instrument({ logOnly: environment.production })

//StoreModule.forRoot({counter: counterReducer}),

],

providers: [ // services (DI)

{

provide: HTTP\_INTERCEPTORS,

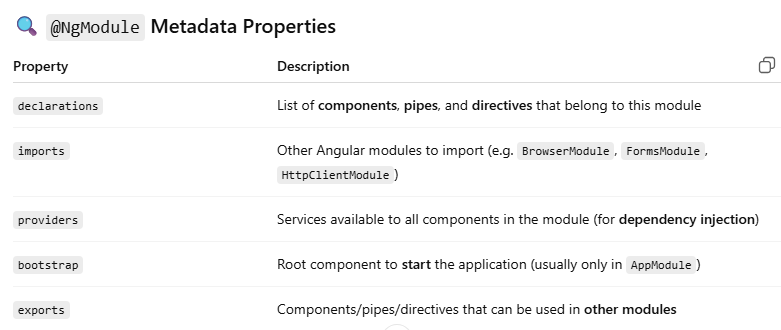
useClass: AuthInterceptor,

multi: true // important for supporting multiple interceptors

}

],

bootstrap: [AppComponent] // root component to bootstrap

})

* **Angular Service**

In **Angular**, a **service** is a class that contains **reusable business logic**, **data access**, or **shared state** that can be injected into components or other services using **dependency injection (DI)**.

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

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

import { Observable } from 'rxjs';

@Injectable({ providedIn: 'root' })

//All logic Inside Class

export class DataService {

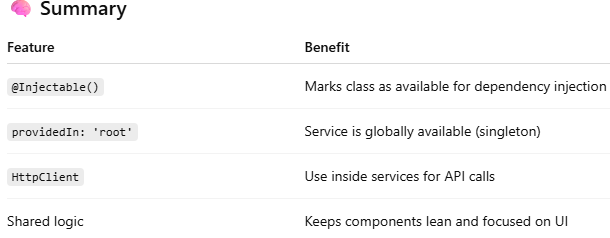
constructor(private http: HttpClient) {}

getUsers(): Observable<any> {

return this.http.get('https://jsonplaceholder.typicode.com/users');

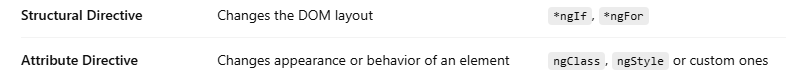
}

}

****

* **Angular Directive**

In **Angular**, a **directive** is a class that adds behavior or modifies the structure or appearance of elements in the DOM.

****

**Custom Directive**

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

@Directive({

selector: '[appHighlight]' // use as attribute selector

})

export class HighlightDirective {

constructor(private el: ElementRef, private renderer: Renderer2) {}

@HostListener('mouseenter') onMouseEnter() {

this.renderer.setStyle(this.el.nativeElement, 'background-color', 'yellow');

}

@HostListener('mouseleave') onMouseLeave() {

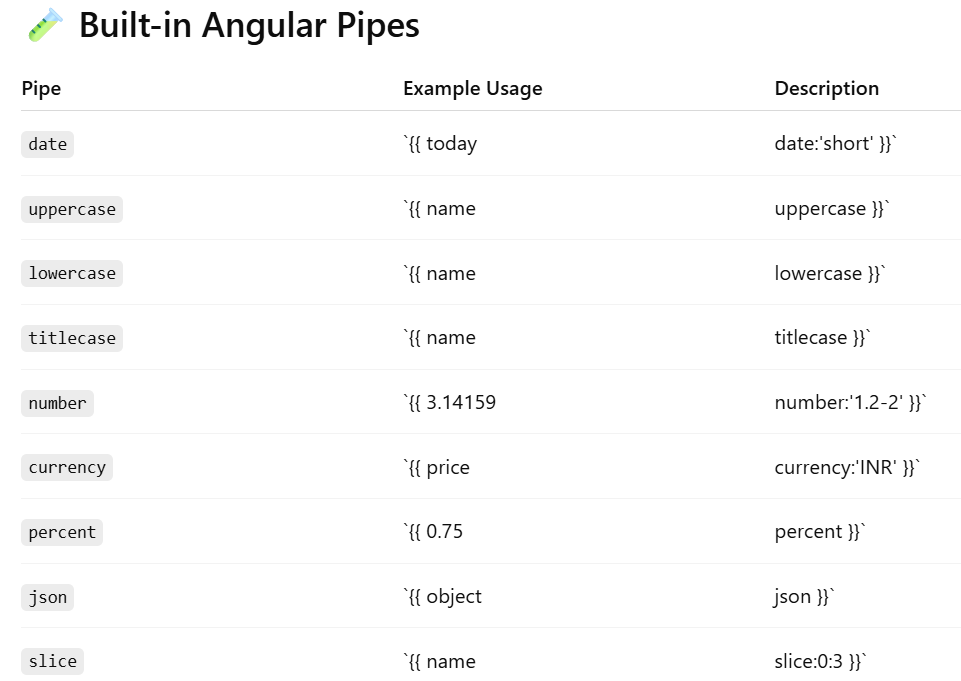
this.renderer.removeStyle(this.el.nativeElement, 'background-color');

}

}

* **Angular Pipe**

In Angular, a **Pipe** is a way to **transform data** in your templates — like formatting dates, numbers, text, or even custom transformations.



**Custom Pipe**

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

@Pipe({

name: 'shorten'

})

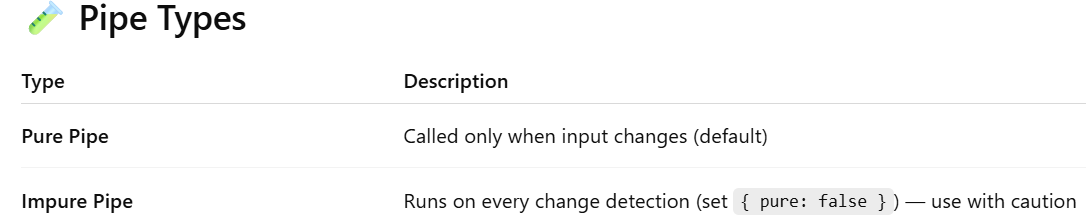
export class ShortenPipe implements PipeTransform {

transform(value: string, limit: number = 10): string {

return value.length > limit ? value.slice(0, limit) + '...' : value;

}

}



* **Components Communicate with each other**

1. **Parent to Child: Sharing Data via Input()**

****

1. **Child to Parent: Sharing Data via ViewChild()**



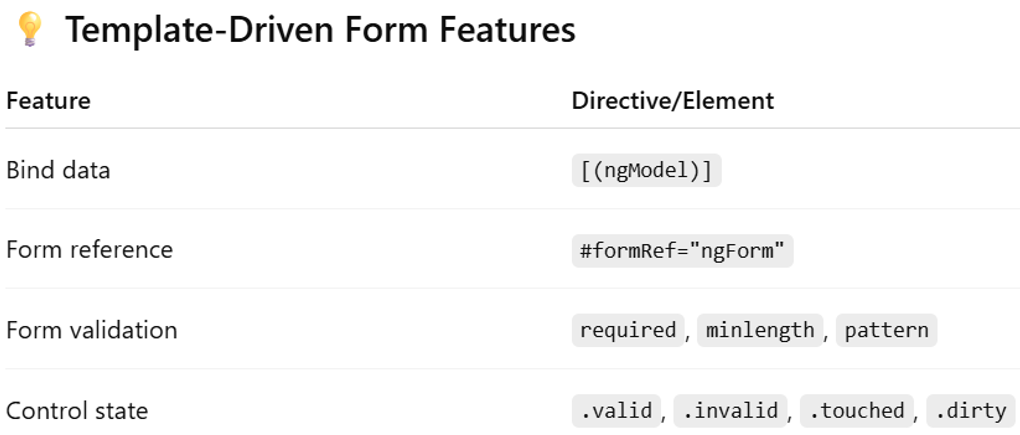


1. **Child to Parent**: Sharing Data via Output() and EventEmitter()





* **Template Driven Form**



**HOW USE**

1. **Import FormsModule in AppModule**

import { FormsModule } from '@angular/forms';

@NgModule({

imports: [FormsModule],

})

1. **Create a Form in the Template**

<form **#userForm="ngForm"** **(ngSubmit)="onSubmit(userForm)">**

<label>Name:</label>

<input type="text" **name="name" [(ngModel)]="user.name" required** />

<br />

<label>Email:</label>

<input type="email" name="email" [(ngModel)]="user.email" required />

<br />

<label>Gender:</label>

<select name="gender" [(ngModel)]="user.gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select>

<br />

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

</form>

<pre>{{ userForm.value | json }}</pre>

1. **Component TS**

export class AppComponent {

@ViewChild('userForm') userForm!: NgForm;

user = {

name: '',

email: '',

gender: 'male'

};

onSubmit(form: any) {

console.log('Form Submitted:', form.value);

}

patchForm() {

this.userForm.form.setValue({

name: 'Neha Jain',

gender: 'female'

});

}

setForm() {

this.userForm.form.setValue({

name: 'Neha Jain',

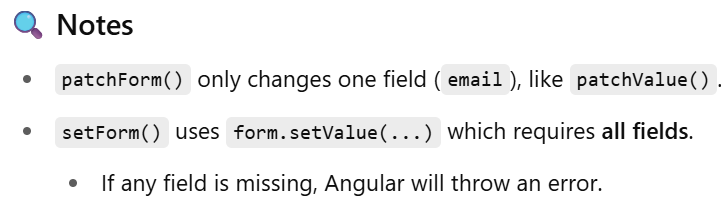
email: 'neha@example.com',

gender: 'female'

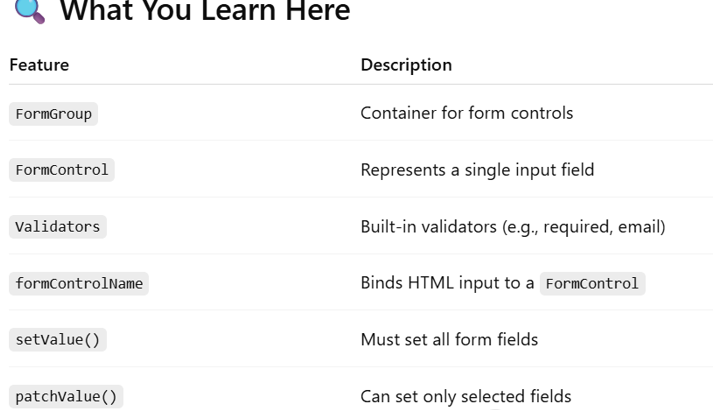
});

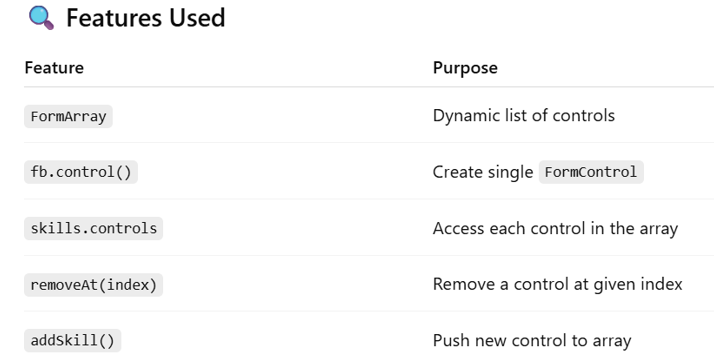
}

}



* **Reactive Form**





**FORMARRAY WITHOUT FORMABUILDER**

1. **Import ReactiveModule**

@NgModule({

imports: [ReactiveFormsModule]

})

1. **Component Ts**

import { FormGroup, FormControl, FormBuilder, Validators, FormArray } from '@angular/forms';

export class AppComponent implements OnInit {

userForm!: FormGroup;

constructor(private fb: FormBuilder) {}

ngOnInit() {

this.userForm = this.fb.group({

name: ['', Validators.required],

email: ['', [Validators.required, Validators.email]],

skills: this.fb.array([this.fb.control('', Validators.required)])

});

}

get skills() {

return this.userForm.get('skills') as FormArray;

}

addSkill() {

this.skills.push(this.fb.control('', Validators.required));

}

removeSkill(index: number) {

this.skills.removeAt(index);

}

onSubmit() {

console.log('Submitted:', this.userForm.value);

}

}

1. **Template file**

<form **[formGroup]="userForm" (ngSubmit)="onSubmit()"**>

<label>Name:</label>

<input type="text" **formControlName="name"** />

<div \*ngIf="userForm.get('name')?.invalid && userForm.get('name')?.touched">

Name is required

</div>

<br />

<label>Email:</label>

<input type="email" formControlName="email" />

<div \*ngIf="userForm.get('email')?.invalid && userForm.get('email')?.touched">

Valid email is required

</div>

<br />

<label>Skills:</label>

<div **formArrayName="skills"**>

<div \***ngFor="let skill of skills.controls; let i = index"**>

<input **[formControlName]="i"** />

<button type="button" (click)="removeSkill(i)" \*ngIf="skills.length > 1">Remove</button>

</div>

<button type="button" (click)="addSkill()">Add Skill</button>

</div>

<br />

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

</form>

<pre>{{ userForm.value | json }}</pre>

**Without FormBuilder**

1. **Compontent Ts**

import { FormGroup, FormControl, Validators } from '@angular/forms';

export class AppComponent implements OnInit {

userForm!: FormGroup;

ngOnInit() {

this.userForm = new FormGroup({

name: new FormControl('', Validators.required),

email: new FormControl('', [Validators.required, Validators.email]),

gender: new FormControl('male')

});

}

onSubmit() {

console.log('Form Submitted:', this.userForm.value);

}

patchForm() {

this.userForm.patchValue({

email: 'patch@example.com'

});

}

setForm() {

this.userForm.setValue({

name: 'Neha Jain',

email: 'neha@example.com',

gender: 'female'

});

}

}

1. **Template File**

<form **[formGroup]="userForm" (ngSubmit)="onSubmit()"**>

<label>Name:</label>

<input type="text" **formControlName="name"** />

<div \*ngIf="userForm.get('name')?.invalid && userForm.get('name')?.touched">

Name is required

</div>

<br />

<label>Email:</label>

<input type="email" formControlName="email" />

<div \*ngIf="userForm.get('email')?.invalid && userForm.get('email')?.touched">

Valid email is required

</div>

<br />

<label>Gender:</label>

<select formControlName="gender">

<option value="male">Male</option>

<option value="female">Female</option>

</select>

<br />

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

</form>

<br />

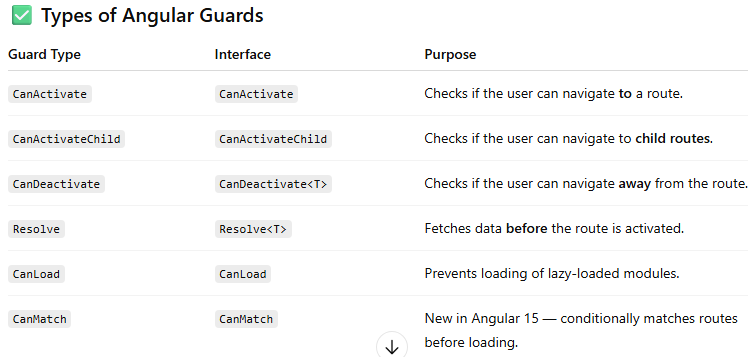
<button (click)="patchForm()">Patch Value</button>

<button (click)="setForm()">Set Value</button>

<pre>{{ userForm.value | json }}</pre>

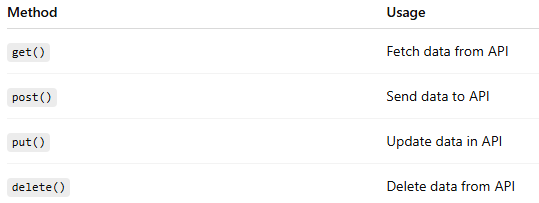
* **Guard**

In **Angular**, **Guards** are used to control navigation — whether a user can access or leave a particular route. Guards implement specific interfaces and are typically used for **authentication**, **authorization**, **data fetching**, or **preventing unsaved changes from being lost**.

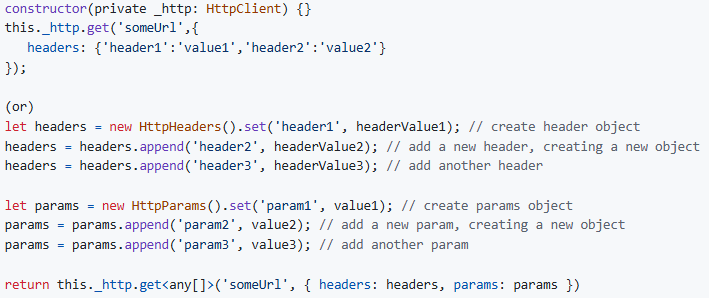
****

* **HttpClient Module**

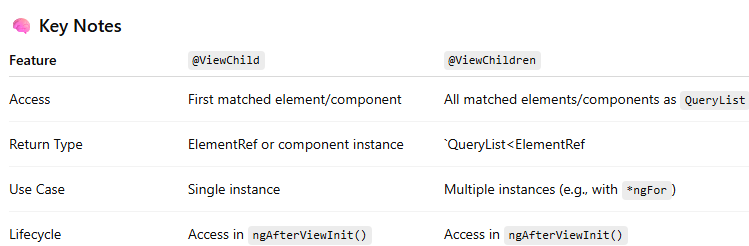
In Angular, the HttpClientModule is used to make HTTP requests to a backend API (e.g., GET, POST, PUT, DELETE). It is part of the @angular/common/http package.

****

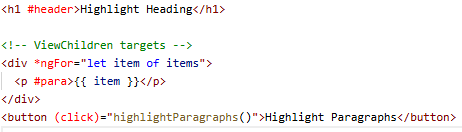
**Both Headers and Params**

****

* **ViewChild and ViewChildren**

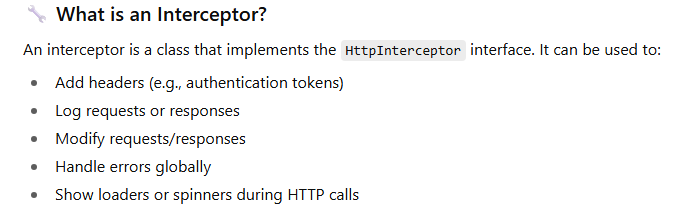
****

****

****

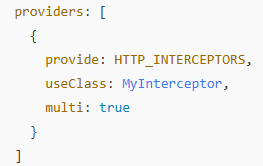
* **Interceptor**

In Angular, **interceptors** are a powerful feature used with the **HttpClient** module to **intercept and modify HTTP requests and responses globally**.

****

****

**Registering the Interceptor**

****

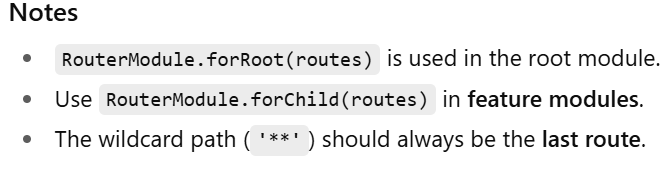
* **Routing Module**

**With WildCard**

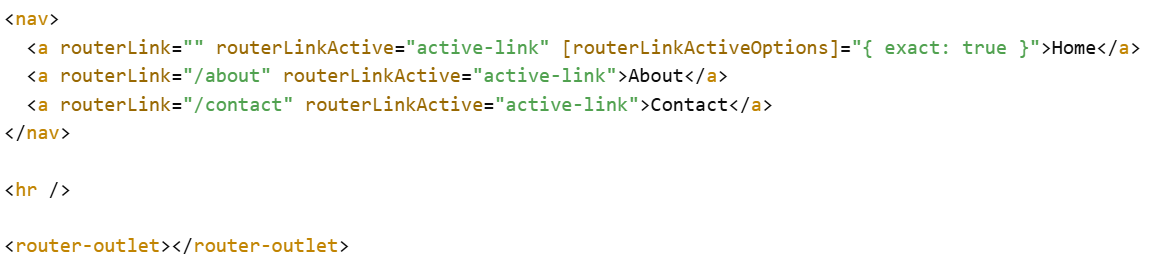
****

**With LazyLoad**

****

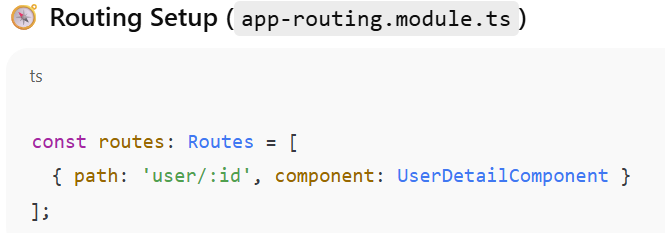
****

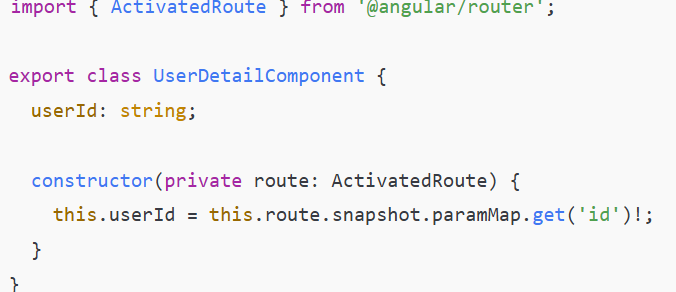
* **Router Outlet**

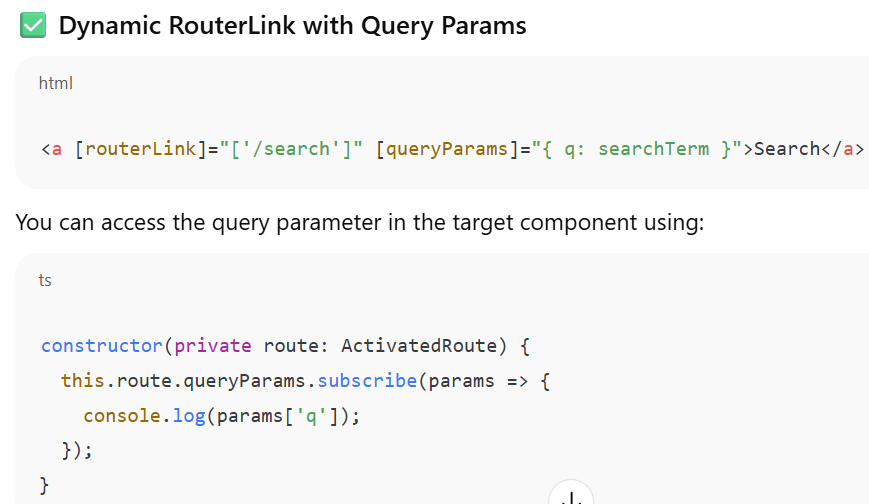
****

**Dynamic routerlink**

****

****

****

****

* **Router Navigate**

organisation/wl/project-detail?id=74412095&caf=64721797&projectId=64721766&clearence=3

**How implements in route navigate**

constructor(private router: Router) {}

this.router.navigate(

['organisation', 'wl', 'project-detail'],

{

queryParams: {

id: 74412095,

caf: 64721797,

projectId: 64721766,

clearence: 3

}

}

);

**OR**

this.router.navigate(

['organisation/wl/project-detail'],

{

queryParams: {

id: 74412095,

caf: 64721797,

projectId: 64721766,

clearence: 3

}

}

);

**what use state**

In Angular's Router.navigate, the state property is used to **pass data to the target route without using query parameters or route parameters**.

this.router.navigate(['/wlc-raise-eds', id], {

state: { name: 'Test', role: 'developer' }

});

**Target Component**

import { Router } from '@angular/router';

constructor(private router: Router) {

const data = this.router.getCurrentNavigation()?.extras.state;

console.log('Data received:', data);

}

**Below URL Implements**

wlc-agenda-summary/WL%2FAGENDA%2FNBWL%2F103588%2F2025/edit

this.router.navigate(['/wlc-agenda-summary', agendaId, 'edit']);

* **ActivatedRoute**

import { ActivatedRoute } from '@angular/router';

constructor(private route: ActivatedRoute) {

        this.route.params.subscribe((params) => {

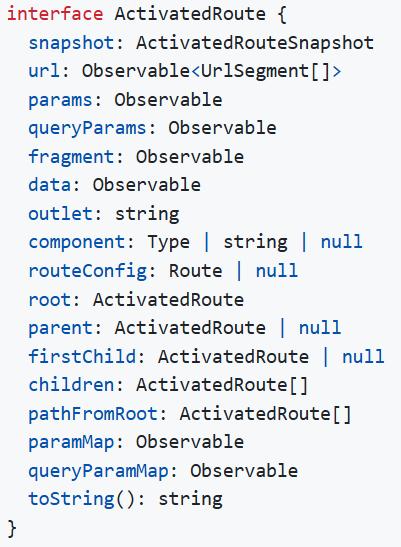
            this.page = params['page'];

            this.agendaMomId = params['id'];

        });

}

**ActivatedRoute Interface looks like:**





const routes: Routes = [

{

path: 'dashboard/:userId',

component: DashboardComponent,

data: { title: 'User Dashboard' },

children: [

{

path: 'reports',

component: ReportsComponent,

outlet: 'sidebar',

data: { title: 'User Reports' }

}

]

}

];

**Example URL:**

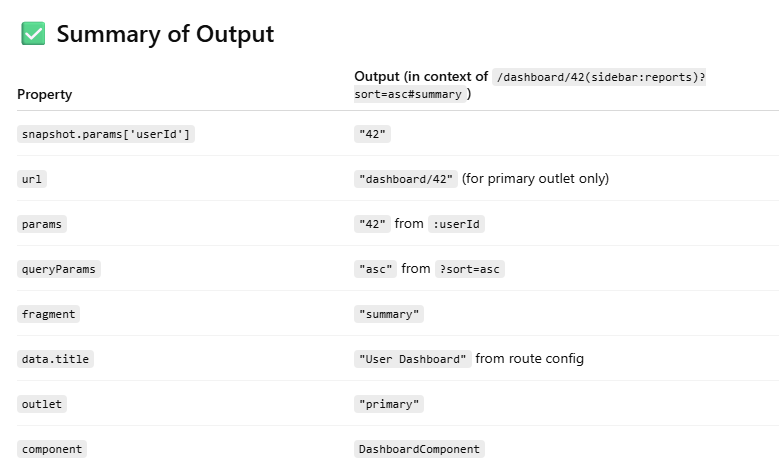
/dashboard/42(sidebar:reports)?sort=asc#summary





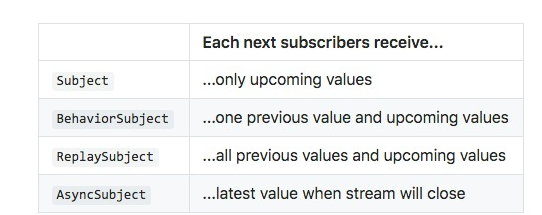


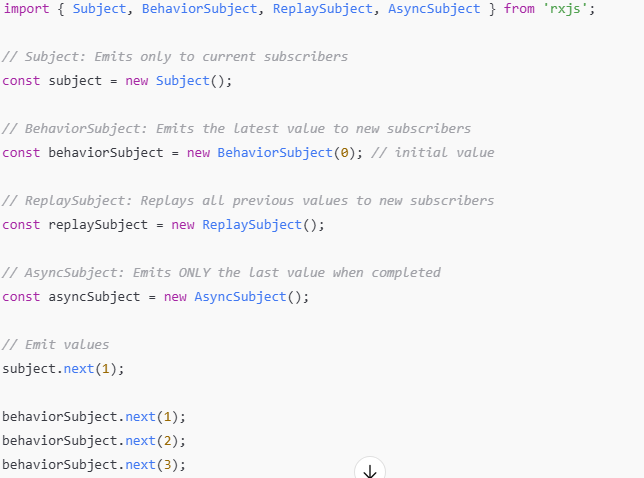


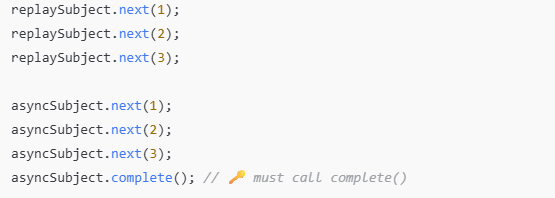




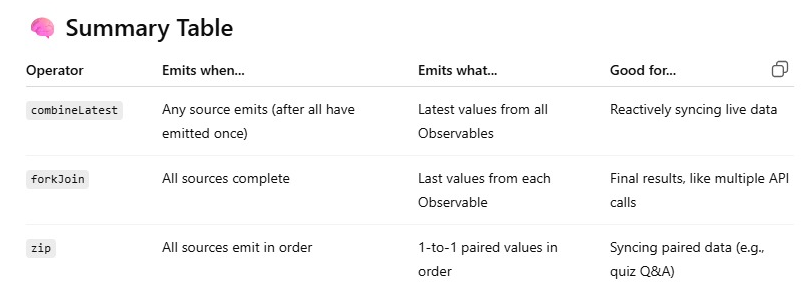
**RXJS Notes**

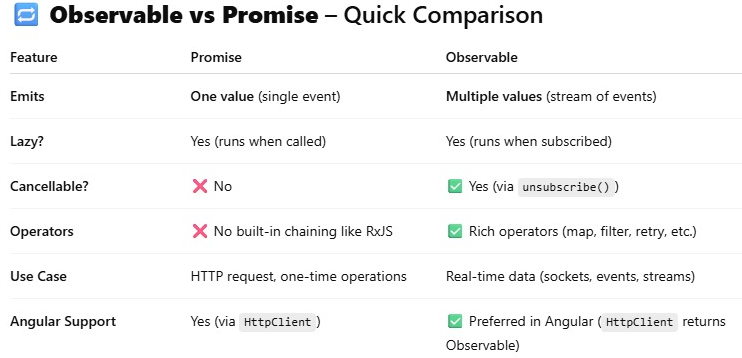
****

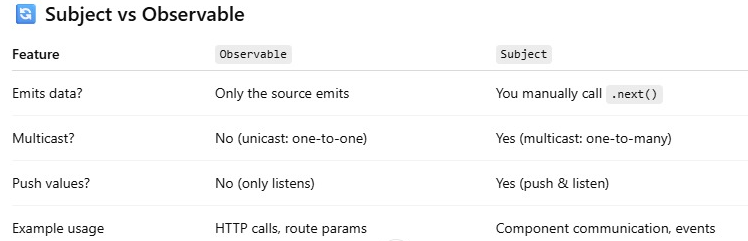




****

****

****

****

