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

@Directive({

selector:"[dirname]"

})

class CardHoverDirective {

constructor(private el: ElementRef, private renderer: Renderer) {

this.el.nativeElement.style.backgroundColor = color; //by ElementRef

renderer.setElementStyle(el.nativeElement, 'backgroundColor', 'gray');

//by renderer

@[HostListener](https://angular.io/api/core/HostListener)('mouseenter') onMouseEnter() { //by 'mouseenter' dynamic

this.highlight('yellow');

}

}

private highlight(color: string) {

this.el.nativeElement.style.backgroundColor = color;

}

}

<p dirname >Show me my color</p>

Custom directive is class along with @directive decorator.

Import Directive, ElementRef ,Renderer from angular core.

Directive This will provide the @Directive decorator and decorator has metadata selector.

selector value is dirname is name of directive, Angular will look for this attribute on the html element, and applies the directive logic to that element.

And inject instance of ElementRef ,Renderer in constructor.

Here ElementRef is class which can get reference of DOM element.

Renderer is a class which has set of methods for setting properties on element

Ex: setElementStyle which is used for set css colors.

**Passing value to Custom Directive (dynamic):**

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

@Directive({

selector: '[appColor]'

})

export class ColorDirective implements OnInit {

@Input() appColor: string;

constructor(private el: ElementRef) { }

ngOnInit(){

this.el.nativeElement.style.backgroundColor = this.appColor;

}

}

<p [appColor]= "'green'">Show me Green Color</p>

<p [appColor]= "'red'">Show me Red Color</p>

**Parent to child:**

Parent:

messageToSendP: string = '';

<app-child [receivedParentMessage]="messageToSendP"></app-child>

Child:

@Input() receivedParentMessage: string;

**Child to parent:**

Parent:

<app-child (messageToEmit)="getMessage($event)"></app-child>

receivedChildMessage: string;

getMessage(message: string) {

this.receivedChildMessage = message;

}

Child:

<button (click)="sendMessageToParent(messageToSendC)">Send to Parent</button>

@Output() messageToEmit = new EventEmitter<string>();

|  |
| --- |
|  |
|  |
|  |  |
|  |  |

sendMessageToParent(message: string)

{ this.messageToEmit.emit(message)

}

**Custom pipes:**

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

@Pipe ({

  name : 'sqrt'

})

export class SqrtPipe implements PipeTransform {

  transform(val : number) : number {

    return Math.sqrt(val);

  }

}

Pipe is a class along with @pipe decorator with metadata name

Here class will implement the PipeTransform interface .and transform method defined in the class.

<https://www.techiediaries.com/angular-tutorial-reactive-forms/>

Routes:

routing module.

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

import { Routes, RouterModule } from '@angular/router';

import { LoginComponent } from './login/login.component';

import { AdminComponent } from './admin/admin.component';

const routes: Routes = [

{ path: '', pathMatch: 'full', redirectTo: 'login'},

{ path: 'login', component: LoginComponent },

{ path: 'admin', component: AdminComponent }

{ path: 'admin', component: AdminComponent, canActivate: [AuthGuard] }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule { }

Router guard:

Used for provide security for components. For ex: admin component

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

import { CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot } from '@angular/router';

import { Observable } from 'rxjs';

import { AuthService } from './auth.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService){}

canActivate(

next: ActivatedRouteSnapshot,

state: RouterStateSnapshot): Observable<boolean> | Promise<boolean> | boolean {

return this.authService.isLoggedIn();

}

}

Reactive forms:

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

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

import { User } from '../user';

import { AuthService } from '../auth.service';

export class LoginComponent implements OnInit {

loginForm: FormGroup;

isSubmitted = false;

constructor(private authService: AuthService, private router: Router, private formBuilder: FormBuilder ) { }

ngOnInit() {

this.loginForm = this.formBuilder.group({

email: ['', Validators.required],

password: ['', Validators.required]

});

}

get formControls() { return this.loginForm.controls; }

login(){

console.log(this.loginForm.value);

this.isSubmitted = true;

if(this.loginForm.invalid){

return;

}

this.authService.login(this.loginForm.value);

this.router.navigateByUrl('/admin');

}

}

<form [formGroup]="loginForm" class="login-container" (ngSubmit)="login()">

<p [ngClass]="{ 'has-error': isSubmitted && formControls.email.errors }">

<input type="email" placeholder="Email" formControlName="email">

</p>

<div \*ngIf="isSubmitted && formControls.email.errors" class="help-block">

<div \*ngIf="formControls.email.errors.required">Email is required</div>

</div>

<p [ngClass]="{ 'has-error': isSubmitted && formControls.password.errors }">

<input type="password" placeholder="Password" formControlName="password">

</p>

<div \*ngIf="isSubmitted && formControls.password.errors" class="help-block">

<div \*ngIf="formControls.password.errors.required">Password is required</div>

</div>

<p>

<input type="submit" value="Log in">

</p>

</form>

Import FormBuilder, FormGroup, Validators  clasess from angular core

Import FormBuilder class to create a reactive form. Also, import FormGroup class to create a group of form controls and Validators for validation

Line 11: Create a property registerForm of type FormGroup

Line 13: Inject a formBuilder instance using constructor

Line 16: formBuilder.group() method creates a FormGroup. It takes an object whose keys are FormControl names and values are their definitions

onSubmit() {

this.submitted = true;

    // stop the process here if form is invalid

    if (this.registerForm.invalid) {

        return;

    }

Formcontrol. Formgroup, formbuilder:

<input type="text" formControlName="username"></ion-input>

this.username = new FormControl('agustin', Validators.required);

this.user\_data = new FormGroup({

username: new FormControl('agustin', Validators.required),

city: new FormControl('Montevideo', Validators.required)

});

this.user\_data = new FormArray([

new FormControl('agustin', Validators.required),

new FormControl('Montevideo', Validators.required)

]);

Template driven forms:

<div class="container">

<div class="row">

<div class="form\_bg">

<form #form="ngForm" (ngSubmit)="registerUser(form)">

<h2 class="text-center">Registration page</h2>

<br/>

<div class="form-group">

<input type="text" class="form-control" placeholder="First Name" name="firstname" required ngModel>

</div>

<div class="form-group">

<input type="text" class="form-control" placeholder="Last Name" name="lastname" required ngModel>

</div>

<div class="form-group">

<input type="email" class="form-control" id="email" placeholder="Email" name="email" email required ngModel #email="ngModel">

<span class="help-bpx" \*ngIf="email.touched && !email.valid ">Please enter the Email Value</span>

</div>

<div class="form-group">

</div>

<br/>

<div class="align-center">

<button type="submit" class="btn btn-default" id="register" [disabled]="!form.valid" >Register</button>

</div>

</form>

</div>

</div>

</div>

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

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

@Component({

selector: 'app-formdemobasics',

templateUrl: './formdemobasics.component.html',

styleUrls: ['./formdemobasics.component.css']

})

export class FormdemobasicsComponent implements OnInit {

constructor() { }

ngOnInit() {

}

Register(regForm:NgForm){

console.log(regForm);

}

}

Services:

<https://www.c-sharpcorner.com/article/crud-operation-in-angular-7-using-web-api/>

1. **import** { Injectable } from '@angular/core';
2. **import** { HttpClient } from '@angular/common/http';
3. **import** { HttpHeaders } from '@angular/common/http';
4. **import** { Observable } from 'rxjs';
5. **import** { Employee } from './employee';
7. After that we write all methods related to consume web **in** employee.service.ts
8. @Injectable({
9. providedIn: 'root'
10. })
12. **export** **class** EmployeeService {
13. url = 'http://localhost:65389/Api/Employee';
14. constructor(**private** http: HttpClient) { }
15. getAllEmployee(): Observable<Employee[]> {
16. **return** **this**.http.get<Employee[]>(**this**.url + '/AllEmployeeDetails');
17. }
18. getEmployeeById(employeeId: string): Observable<Employee> {
19. **return** **this**.http.get<Employee>(**this**.url + '/GetEmployeeDetailsById/' + employeeId);
20. }
21. createEmployee(employee: Employee): Observable<Employee> {
22. **const** httpOptions = { headers: **new** HttpHeaders({ 'Content-Type': 'application/json'}) };
23. **return** **this**.http.post<Employee>(**this**.url + '/InsertEmployeeDetails/',
24. employee, httpOptions);
25. }
26. updateEmployee(employee: Employee): Observable<Employee> {
27. **const** httpOptions = { headers: **new** HttpHeaders({ 'Content-Type': 'application/json'}) };
28. **return** **this**.http.put<Employee>(**this**.url + '/UpdateEmployeeDetails/',
29. employee, httpOptions);
30. }
31. deleteEmployeeById(employeeid: string): Observable<number> {
32. **const** httpOptions = { headers: **new** HttpHeaders({ 'Content-Type': 'application/json'}) };
33. **return** **this**.http.**delete**<number>(**this**.url + '/DeleteEmployeeDetails?id=' +employeeid,httpOptions);
34. }
35. }

Crud:

1. **import** { Component, OnInit } from '@angular/core';
2. **import** { FormBuilder, Validators } from '@angular/forms';
3. **import** { Observable } from 'rxjs';
4. **import** { EmployeeService } from '../employee.service';
5. **import** { Employee } from '../employee';
7. @Component({
8. selector: 'app-employee',
9. templateUrl: './employee.component.html',
10. styleUrls: ['./employee.component.css']
11. })
12. **export** **class** EmployeeComponent **implements** OnInit {
13. dataSaved = **false**;
14. employeeForm: any;
15. allEmployees: Observable<Employee[]>;
16. employeeIdUpdate = **null**;
17. massage = **null**;
19. constructor(**private** formbulider: FormBuilder, **private** employeeService:EmployeeService) { }
21. ngOnInit() {
22. **this**.employeeForm = **this**.formbulider.group({
23. EmpName: ['', [Validators.required]],
24. DateOfBirth: ['', [Validators.required]],
25. EmailId: ['', [Validators.required]],
26. Gender: ['', [Validators.required]],
27. Address: ['', [Validators.required]],
28. PinCode: ['', [Validators.required]],
29. });
30. **this**.loadAllEmployees();
31. }
32. loadAllEmployees() {
33. **this**.allEmployees = **this**.employeeService.getAllEmployee();
34. }
35. onFormSubmit() {
36. **this**.dataSaved = **false**;
37. **const** employee = **this**.employeeForm.value;
38. **this**.CreateEmployee(employee);
39. **this**.employeeForm.reset();
40. }
41. loadEmployeeToEdit(employeeId: string) {
42. **this**.employeeService.getEmployeeById(employeeId).subscribe(employee=> {
43. **this**.massage = **null**;
44. **this**.dataSaved = **false**;
45. **this**.employeeIdUpdate = employee.EmpId;
46. **this**.employeeForm.controls['EmpName'].setValue(employee.EmpName);
47. **this**.employeeForm.controls['DateOfBirth'].setValue(employee.DateOfBirth);
48. **this**.employeeForm.controls['EmailId'].setValue(employee.EmailId);
49. **this**.employeeForm.controls['Gender'].setValue(employee.Gender);
50. **this**.employeeForm.controls['Address'].setValue(employee.Address);
51. **this**.employeeForm.controls['PinCode'].setValue(employee.PinCode);
52. });
54. }
55. CreateEmployee(employee: Employee) {
56. **if** (**this**.employeeIdUpdate == **null**) {
57. **this**.employeeService.createEmployee(employee).subscribe(
58. () => {
59. **this**.dataSaved = **true**;
60. **this**.massage = 'Record saved Successfully';
61. **this**.loadAllEmployees();
62. **this**.employeeIdUpdate = **null**;
63. **this**.employeeForm.reset();
64. }
65. );
66. } **else** {
67. employee.EmpId = **this**.employeeIdUpdate;
68. **this**.employeeService.updateEmployee(employee).subscribe(() => {
69. **this**.dataSaved = **true**;
70. **this**.massage = 'Record Updated Successfully';
71. **this**.loadAllEmployees();
72. **this**.employeeIdUpdate = **null**;
73. **this**.employeeForm.reset();
74. });
75. }
76. }
77. deleteEmployee(employeeId: string) {
78. **if** (confirm("Are you sure you want to delete this ?")) {
79. **this**.employeeService.deleteEmployeeById(employeeId).subscribe(() => {
80. **this**.dataSaved = **true**;
81. **this**.massage = 'Record Deleted Succefully';
82. **this**.loadAllEmployees();
83. **this**.employeeIdUpdate = **null**;
84. **this**.employeeForm.reset();
86. });
87. }
88. }
89. resetForm() {
90. **this**.employeeForm.reset();
91. **this**.massage = **null**;
92. **this**.dataSaved = **false**;
93. }
94. }

Formgroup vs form array:

orderForm: FormGroup;

items: FormArray;

constructor(private formBuilder: FormBuilder) {}

ngOnInit() {

this.orderForm = this.formBuilder.group({

customerName: '',

email: '',

items: this.formBuilder.array([ this.createItem() ])

});

}

createItem(): FormGroup {

return this.formBuilder.group({

name: '',

description: '',

price: ''

});

}

## Adding to the FormArray Dynamically

addItem(): void {

this.items = this.orderForm.get('items') as FormArray;

this.items.push(this.createItem());

}

<div formArrayName="items"

\*ngFor="let item of orderForm.get('items').controls; let i = index;">

<div [formGroupName]="i">

<input formControlName="name" placeholder="Item name">

<input formControlName="description" placeholder="Item description">

<input formControlName="price" placeholder="Item price">

</div>

Chosen name: {{ orderForm.controls.items.controls[i].controls.name.value }}

</div>

Other way:

exampleForm = new FormGroup ({

firstName: new FormControl(),

lastName: new FormControl(),

alias: new FormArray([addNewAlias()])

});

addNewAlias() {

return new FormGroup ({

firstName: new FormControl(),

});

}

get aliases() {

const alias= this.exampleForm.get("alias") as FormArray;

alias.push(addNewAlias());

}

<form [formGroup]="exampleForm">

<div class="form-group">

<label>First Name:</label>

<input class="form-control" formControlName="firstName">

<label>Last Name:</label>

<input class="form-control" formControlName="lastName">

<div formArrayName="alias">

<h3>Add alias</h3>

<button (click)="addNewAlias();" >Add another alias </button>

<div \*ngFor="let address of aliases.controls; let i=index">

<input type="text" [formControlName] = "i" >

</div>

</div>

</div>

</form>