**LISTADOS DE CODIGO**

**CURSO ANGULAR 19**

**9. ANGULAR PROJECTS AND TOOLS**

9.1 🡪**datasource.model.ts**

import { Product } from "./product.model";

export class SimpleDataSource {

private data: Product[];

constructor() {

this.data = new Array<Product>(

new Product(1, "Kayak", "Watersports", 275),

new Product(2, "Lifejacket", "Watersports", 48.95),

new Product(3, "Soccer Ball", "Soccer", 19.50),

new Product(4, "Corner Flags", "Soccer", 34.95),

new Product(5, "Thinking Cap", "Chess", 16));

}

getData(): Product[] {

return this.data;

}

}

9.2 🡪 **repository.model.ts**

import { Product } from "./product.model";

import { SimpleDataSource } from "./datasource.model";

export class Model {

private dataSource: SimpleDataSource;

private products: Product[];

private locator = (p: Product, id: number) => p.id == id;

constructor() {

this.dataSource = new SimpleDataSource();

this.products = new Array<Product>();

this.dataSource.getData().forEach(p => this.products.push(p));

}

getProducts(): Product[] {

return this.products;

}

getProduct(id: number): Product {

return this.products.find(p => this.locator(p, id));

}

saveProduct(product: Product) {

if (product.id == 0 || product.id == null) {

product.id = this.generateID();

this.products.push(product);

} else {

let index = this.products

.findIndex(p => this.locator(p, product.id));

this.products.splice(index, 1, product);

}

}

deleteProduct(id: number) {

let index = this.products.findIndex(p => this.locator(p, id));

if (index > -1) {

this.products.splice(index, 1);

}

}

private generateID(): number {

let candidate = 100;

while (this.getProduct(candidate) != null) {

candidate++;

}

return candidate;

}

}

9.3 🡪 **product.component.ts**

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

import { Model } from "./repository.model";

@Component({

selector: "app-product",

templateUrl: "./product.component.html",

styleUrl: "./product.component.css",

})

export class ProductComponent {

model: Model = new Model();

}

**9.4 🡪 index.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>Ejemplo</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

**<**app-product**></**app-product **>**

</body>

</html>

**9.5 🡪 main.server.ts**

import { bootstrapApplication } from '@angular/platform-browser';

import { config } from './app/app.config.server';

import { TableComponent } from './app/core/table/table.component';

const bootstrap = () => bootstrapApplication(TableComponent, config);

export default bootstrap;

**9.6 🡪 main.ts**

import { bootstrapApplication } from '@angular/platform-browser';

import { appConfig } from './app/app.config';

import { ProductComponent } from './app/product/product.component';

bootstrapApplication(ProductComponent, appConfig)

.catch((err) => console.error(err));

**10. DATA BINDINGS**

**10.1 🡪 product.component.ts**

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

import { Model } from "./repository.model";

@Component({

selector: "app",

templateUrl: "template.html"

})

export class ProductComponent {

model: Model = new Model();

getClasses(key: number): string {

let product = this.model.getProduct(key);

return "p-2 " + (product?.price && product.price < 50 ? "bg-info" : "bg-warning");

}

}

**10.2 🡪 product.component.html**

<div class="text-white m-2">

**<div class="p-2" [ngClass]="getClassMap(1)">**

**The first product is {{model.getProduct(1)?.name}}**

**</div>**

**<div class="p-2" [ngClass]="getClassMap(2)">**

**The second product is {{model.getProduct(2)?.name}}**

**</div>**

**<div class="p-2"**

**[ngClass]="{'bg-success':model.getProduct(3)?.price < 50,'bg-info':model.getProduct(3)?.price >= 50}">**

**The third product is {{model.getProduct(3)?.name}}**

**</div>**

</div>

**11. DIRECTIVAS BUILT-IN**

**11.1 🡪 product**.**component.ts**

import { ApplicationRef, Component } from "@angular/core";

import { Model } from "./repository.model";

import { CommonModule } from '@angular/common';

import { Product } from "./product.model";

@Component({

selector: "app",

imports: [CommonModule], // Importa CommonModule aquí

templateUrl: "template.html"

})

export class ProductComponent {

model: Model = new Model();

constructor(ref: ApplicationRef) {

if (typeof window !== 'undefined') {

(<any>window).appRef = ref;

(<any>window).model = this.model;

}

}

getProductByPosition(position: number): Product {

return this.model.getProducts()[position];

}

getProduct(key: number): Product {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

getProductCount(): number {

console.log("getProductCount invoked");

return this.getProducts().length;

}

targetName: string = "Kayak";

}

**11.2 🡪 template.html**

<div class="text-white m-2">

<div class="bg-info p-2">

There are **{{**getProductCount()**}}** products.

</div>

**<div \*ngIf="getProductCount() > 4" class="bg-info p-2">**

**There are more than 4 products in the model**

**</div>**

**<div \*ngIf="getProductByPosition(0).name != 'Kayak'" class="bg-info p-2">**

**The first product isn't a Kayak**

**</div>**

</div>

**11.3 🡪 template.html**

<div class="text-white m-2">

<div class="bg-info p-2">

There are **{{**getProductCount()**}}** products.

</div>

**<div class="bg-info p-2 mt-1" [ngSwitch]="getProductCount()">**

**<span \*ngSwitchCase="2">There are two products</span>**

**<span \*ngSwitchCase="5">There are five products</span>**

**<span \*ngSwitchDefault>This is the default</span>**

**</div>**

</div>

**11.4 🡪 prouduct.component.html**

<ng-template #titleTemplate>

<h4 class="p-2 bg-success text-white">Repeat Content</h4>

</ng-template>

<ng-template [ngTemplateOutlet]="titleTemplate"></ng-template>

<div class="bg-info p-2">

There are **{{**getProductCount()**}}** products.

</div>

<ng-template [ngTemplateOutlet]="titleTemplate"></ng-template>

**11.5 🡪 prouduct.component.html**

**<ng-template #titleTemplate let-text="title">**

**<h4 class="p-2 bg-success text-white">{{text}}</h4>**

**</ng-template>**

**<ng-template [ngTemplateOutlet]="titleTemplate"**

**[ngTemplateOutletContext]="{title:'Header'}"></ng-template>**

<div class="bg-info p-2">

There are **{{**getProductCount()**}}** products.

</div>

**<ng-template [ngTemplateOutlet]="titleTemplate"**

**[ngTemplateOutletContext]="{title:'Footer'}"></ng-template>**

**12. EVENTS AND FORMS**

**12.1 🡪 prouduct.component.ts**

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

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

import { Model } from "./repository.model";

import { CommonModule } from '@angular/common';

import { Product } from './product.model';

@Component({

selector: 'app-product',

standalone: true,

imports: [RouterOutlet, CommonModule],

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

styleUrl: './product.component.css'

})

export class ProductComponent {

title = 'example';

model: Model = new Model();

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

selectedProduct: string | undefined;

}

**12.2 🡪 prouduct.component.html**

<div class="text-white m-2">

<table class="table table-sm table-bordered mt-1 text-dark">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts();

let i=index">

<td>**{{**i+1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>$**{{**item.price**}}**</td>

</tr>

</tbody>

</table>

</div>

**12.3 🡪 prouduct.component.html**

<div class="text-white m-2">

<div class="bg-info text-white p-2">

Selected Product: **{{**selectedProduct ?? '(None)'**}}**

</div>

<table class="table table-sm table-bordered mt-1 text-dark">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts();

let i=index">

<td (mouseover)="selectedProduct=item.name"

[class.bg-info]="getSelected(item)">

**{{**i+1**}}**

</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>$**{{**item.price**}}**</td>

</tr>

</tbody>

</table>

<div class="form-group">

<label>Product Name</label>

<input class="form-control" (input)="selectedProduct=$any($event.target).value"

[value]="selectedProduct ?? ''" />

</div>

<div class="form-group">

<label>Product Name</label>

<input class="form-control" (input)="selectedProduct=$any($event.target).value"

[value]="selectedProduct ?? ''" />

</div>

</div>

**12.4 🡪 product.component.html**

<div class="text-dark m-2">

<div class="bg-info text-white mb-2 p-2">Model Data : **{{**jsonProduct ?? '(None)'**}}**</div>

<div class="form-group">

<label>Product Name</label>

<input class="form-control" [(ngModel)]="newProduct.name" />

</div>

<div class="form-group">

<label>Category</label>

<input class="form-control" [(ngModel)]="newProduct.category" />

</div>

<div class="form-group">

<label>Price</label>

<input class="form-control" [(ngModel)]="newProduct.price" />

</div>

<button class="btn btn-primary" (click)="addProduct(newProduct)">Create</button>

</div>

**12.5 🡪 product.component.html**

<div class="m-2">

<div class="bg-info text-white mb-2 p-2">Model Data : **{{**jsonProduct **}}**</div>

<form novalidate (ngSubmit)="addProduct(newProduct)">

<div class="form-group">

<label>Product Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

required

minlength="5"

pattern="^[A-Za-z ]+$"/>

</div>

<button class="btn btn-primary" (click)="addProduct(newProduct)">Create</button>

</form>

</div>

**12.6 🡪 styles.css**

html, body { height: 100%; }

body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }

**input.ng-dirty.ng-invalid {**

**border: 2px solid #ff0000**

**}**

**input.ng.dirty.ng-valid {**

**border: 2px solid #6bc502**

**}**

**12.7 🡪 product.component.html**

<div class="m-2">

<div class="bg-info text-white mb-2 p-2">Model Data : **{{**jsonProduct**}}**</div>

<form novalidate (ngSubmit)="addProduct(newProduct)">

<div class="form-group">

<label>Product Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

**#name="ngModel"**

required

minlength="5"

pattern="^[A-Za-z ]+$" />

**<ul class="text-danger list-unstyled mt-1" \*ngIf=" name.dirty && name.invalid">**

**<li \*ngIf="name.errors?.['required']">**

**You must enter a product name**

**</li>**

**<li \*ngIf="name.errors?.['pattern']">**

**Product names can only contain letters and spaces**

**</li>**

**<li \*ngIf="name.errors?.['minlength']">**

**Product names must be al least**

**{{ name.errors?.['minlength'].requiredLength }} characters**

**</li>**

**</ul>**

**</div>**

<button class="btn btn-primary mt-2" type="submit">Create</button>

</form>

</div>

**12.8 🡪 product.component.ts**

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

import { Model } from "./repository.model";

import { Product } from "./product.model";

**import { NgModel, ValidationErrors } from "@angular/forms";**

@Component({

selector: "app",

templateUrl: "template.html"

})

export class ProductComponent {

model: Model = new Model();

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

newProduct: Product = new Product();

get jsonProduct() {

return JSON.stringify(this.newProduct);

}

addProduct(p: Product) {

console.log("New Product: " + this.jsonProduct);

}

**getMessages(errs: ValidationErrors | null, name: string): string[] {**

**let messages: string[] = [];**

**for (let errorName in errs) {**

**switch (errorName) {**

**case "required":**

**messages.push(`You must enter a ${name}`);**

**break;**

**case "minlength":**

**messages.push(`A ${name} must be at least**

**${errs['minlength'].requiredLength}**

**characters`);**

**break;**

**case "pattern":**

**messages.push(`The ${name} contains**

**illegal characters`);**

**break;**

**}**

**}**

**return messages;**

**}**

**getValidationMessages(state: NgModel, thingName?: string) {**

**let thing: string = state.path?.[0] ?? thingName;**

**return this.getMessages(state.errors, thing)**

**}**

}

**12.9 🡪 product.component.html**

<div class="p-2">

**<form novalidate #form="ngForm" (ngSubmit)="submitForm(form)">**

**<div class="bg-danger text-white p-2 mb-2"**

**\*ngIf="formSubmitted && form.invalid">**

**There are problems with the form**

**</div>**

<div class="form-group">

<label>Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

#name="ngModel"

required

minlength="5"

pattern="^[A-Za-z ]+$" />

**<ul class="text-danger list-unstyled mt-1" \*ngIf="(formSubmitted || name.dirty) && name.invalid">**

<li \*ngFor="let error of getValidationMessages(name)">

**{{**error**}}**

</li>

</ul>

</div>

<button class="btn btn-primary mt-2" type="submit">

Create

</button>

</form>

</div>

**12.10 🡪 product.component.ts**

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

import { Model } from "./repository.model";

import { Product } from "./product.model";

import { NgModel, ValidationErrors, NgForm } from "@angular/forms";

@Component({

selector: "app",

templateUrl: "template.html"

})

export class ProductComponent {

model: Model = new Model();

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

newProduct: Product = new Product();

get jsonProduct() {

return JSON.stringify(this.newProduct);

}

addProduct(p: Product) {

console.log("New Product: " + this.jsonProduct);

}

getMessages(errs : ValidationErrors | null, name: string) : string[] {

let messages: string[] = [];

for (let errorName in errs) {

switch (errorName) {

case "required":

messages.push(`You must enter a ${name}`);

break;

case "minlength":

messages.push(`A ${name} must be at least

${errs['minlength'].requiredLength}

characters`);

break;

case "pattern":

messages.push(`The ${name} contains

illegal characters`);

break;

}

}

return messages;

}

getValidationMessages(state: NgModel, thingName?: string) {

let thing: string = state.path?.[0] ?? thingName;

return this.getMessages(state.errors, thing)

}

formSubmitted: boolean = false;

submitForm(form: NgForm) {

this.formSubmitted = true;

if (form.valid) {

this.addProduct(this.newProduct);

this.newProduct = new Product();

form.resetForm();

this.formSubmitted = false;

}

}

**getFormValidationMessages(form: NgForm): string[] {**

**let messages: string[] = [];**

**Object.keys(form.controls).forEach(k => {**

**this.getMessages(form.controls[k].errors, k)**

**.forEach(m => messages.push(m));**

**});**

**return messages;**

**}**

}

**12.11 🡪 product.component.html (FORMULARIO COMPLETO)**

<div class="p-2">

<form novalidate #form="ngForm" (ngSubmit)="submitForm(form)">

<div class="bg-danger text-white p-2 mb-2"

\*ngIf="formSubmitted && form.invalid">

There are problems with the form

<ul>

<li \*ngFor="let error of getFormValidationMessages(form)">

**{{**error**}}**

</li>

</ul>

</div>

<div class="form-group">

<label>Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

#name="ngModel"

required

minlength="5"

pattern="^[A-Za-z ]+$" />

</div>

**<div class="form-group">**

**<label>Category</label>**

**<input class="form-control" name="category"**

**[(ngModel)]="newProduct.category" required />**

**</div>**

**<div class="form-group">**

**<label>Price</label>**

**<input class="form-control" name="price"**

**[(ngModel)]="newProduct.price" required type="number"/>**

**</div>**

<button class="btn btn-primary mt-2" type="submit"

[disabled]="formSubmitted && form.invalid"

[class.btn-secondary]="formSubmitted && form.invalid">

Create

</button>

</form>

</div>

**13. DIRECTIVAS ATTRIBUTE**

**13.1 🡪 product.component.html**

<div class="container-fluid">

<div class="row p-2">

<div class="col-6">

<form novalidate #form="ngForm" (ngSubmit)="submitForm(form)">

<div class="bg-danger text-white p-2 mb-2"

\*ngIf="formSubmitted && form.invalid">

There are problems with the form

<ul>

<li \*ngFor="let error of getFormValidationMessages(form)">

**{{**error**}}**

</li>

</ul>

</div>

<div class="form-group">

<label>Product Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

#name="ngModel"

required

minlength="5"

pattern="^[A-Za-z ]+$" />

<ul class="text-danger list-unstyled mt-1" \*ngIf="(formSubmitted || name.dirty) && name.invalid">

<li \*ngFor="let error of getValidationMessages(name)">**{{**error**}}**</li>

</ul>

</div>

<div class="form-group">

<label>Category</label>

<input class="form-control" name="category"

[(ngModel)]="newProduct.category" required />

</div>

<div class="form-group">

<label>Price</label>

<input class="form-control" name="price"

[(ngModel)]="newProduct.price" required type="number" />

</div>

<button class="btn btn-primary mt-2" type="submit"

[disabled]="formSubmitted && form.invalid"

[class.btn-secondary]="formSubmitted && form.invalid">

Create

</button>

</form>

</div>

<div class="col">

<table class="table table-bordered table-striped">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts(); let i = index">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>$**{{**item.price**}}**</td>

</tr>

</tbody>

</table>

</div>

</div>

</div>

**13.2 🡪 product.component.ts**

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

import { Model } from "./repository.model";

import { Product } from "./product.model";

@Component({

selector: "app",

templateUrl: "template.html"

})

export class ProductComponent {

model: Model = new Model();

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

newProduct: Product = new Product();

**addProduct(p: Product) {**

**this.model.saveProduct(p);**

**}**

submitForm() {

this.addProduct(this.newProduct);

}

}

**13.3 🡪 attr.directive.ts**

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

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

constructor(element: ElementRef) {

element.nativeElement.classList.add("bg-success", "fw-bold");

}

}

**13.4 🡪 attr.directive.ts**

**import { Directive, ElementRef, Attribute } from "@angular/core";**

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

constructor(element: ElementRef**, @Attribute("pa-attr-class") bgClass: string**) {

**element.nativeElement.classList.add(bgClass || "table-success", "fw-bold");**

}

}

**13.5 🡪 attr.directive.ts**

import { Directive, ElementRef, Attribute } from "@angular/core";

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

**constructor(element: ElementRef, @Attribute("pa-attr") bgClass: string) {**

element.nativeElement.classList.add(bgClass || “table-success", "fw-bold");

}

}

**13.6 🡪 attr.directive.ts**

**import { Directive, ElementRef, Input } from "@angular/core";**

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

**constructor(private element: ElementRef) {}**

**@Input("pa-attr")**

**bgClass: string | null = " ";**

**ngOnInit() {**

**this.element.nativeElement.classList.add(this.bgClass || "table-success", "fw-bold");**

**}**

}

**13.7 🡪 attr.directive.ts**

**import { Directive, ElementRef, Input, SimpleChanges } from "@angular/core";**

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

constructor(private element: ElementRef) {}

@Input("pa-attr")

bgClass: any;

**ngOnChanges(changes: SimpleChanges ) {**

**let change = changes["bgClass"];**

**let classList = this.element.nativeElement.classList;**

**if (!change.isFirstChange() && classList.contains(change.previousValue)) {**

**classList.remove(change.previousValue);**

**}**

**if (!classList.contains(change.currentValue)) {**

**classList.add(change.currentValue);**

**}**

**}**

}

**13.8 🡪 attr.directive.ts**

**import { Directive, ElementRef,**

**Input, SimpleChanges, Output, EventEmitter**

**} from "@angular/core";**

**import { Product } from "./product.model";**

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

**constructor(private element: ElementRef) {**

**this.element.nativeElement.addEventListener("click", () => {**

**if (this.product != null) {**

**this.click.emit(this.product.category);**

**}**

**});**

**}**

@Input("pa-attr")

bgClass: string | null = **""**;

**@Input("pa-product")**

**product: Product = new Product();**

**@Output("pa-category")**

**click = new EventEmitter<string>();**

ngOnChanges(changes: { [property: string]: SimpleChange }) {

let change = changes["bgClass"];

let classList = this.element.nativeElement.classList;

if (!change.isFirstChange() && classList.contains(change.previousValue)) {

classList.remove(change.previousValue);

}

if (!classList.contains(change.currentValue)) {

classList.add(change.currentValue);

}

}

}

**13.9 🡪 attr.directive.ts**

**import { Directive, ElementRef,**

**Input, SimpleChange, Output, EventEmitter, HostListener, HostBinding**

} from "@angular/core";

import { Product } from "./product.model";

@Directive({

selector: "[pa-attr]"

})

export class PaAttrDirective {

@Input("pa-attr")

**@HostBinding("class")**

bgClass: any;

@Input("pa-product")

product: Product = new Product();

@Output("pa-category")

click = new EventEmitter<string>();

**@HostListener("click")**

**triggerCustomEvent() {**

**if (this.product != null) {**

**this.click.emit(this.product.category);**

**}**

**}**

}

**13.10 🡪 twoway.directive.ts**

import { Input, Output, EventEmitter, Directive,

HostBinding, HostListener, SimpleChange } from "@angular/core";

@Directive({

selector: "input[paModel]",

**exportAs: "paModel"**

})

export class PaModel {

**direction: string = "None"**

@Input("paModel")

modelProperty: string | undefined = "";

@HostBinding("value")

fieldValue: string = "";

ngOnChanges(changes: { [property: string]: SimpleChange }) {

let change = changes["modelProperty"];

if (change.currentValue != this.fieldValue) {

this.fieldValue = changes["modelProperty"].currentValue || "";

**this.direction = "Model"**

}

}

@Output("paModelChange")

update = new EventEmitter<string>();

@HostListener("input", ["$event.target.value"])

updateValue(newValue: string) {

this.fieldValue = newValue;

this.update.emit(newValue);

**this.direction = "Element";**

}

}

**14. DIRECTIVAS STRUCTURAL**

**14.1 🡪 template.html**

<div class="p-2">

<table class="table table-bordered table-striped">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts(); let i = index"

[pa-attr]="getProducts().length < 6 ? 'table-success' : 'table-warning'"

[pa-product]="item" (pa-category)="newProduct.category = $event">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td [pa-attr]="item.category == 'Soccer' ? 'table-info' : null">

**{{**item.category**}}**

</td>

<td [pa-attr]="'table-info’”>$**{{**item.price**}}**</td>

</tr>

</tbody>

</table>

</div>

**14.2 🡪 structure.directive.ts**

import {

Directive, SimpleChanges, ViewContainerRef, TemplateRef, Input

} from "@angular/core";

@Directive({

selector: "[paIf]"

})

export class PaStructureDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) { }

@Input("paIf")

expressionResult: boolean | undefined;

ngOnChanges(changes: SimpleChanges) {

let change = changes["expressionResult"];

if (!change.isFirstChange() && !change.currentValue) {

this.container.clear();

} else if (change.currentValue) {

this.container.createEmbeddedView(this.template);

}

}

}

**14.3 🡪 iterator.directive.ts**

import { Directive, ViewContainerRef, TemplateRef,

Input} from "@angular/core";

@Directive({

selector: "[paForOf]"

})

export class PaIteratorDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) {}

@Input("paForOf")

dataSource: any;

ngOnInit() {

this.container.clear();

for (let i = 0; i < this.dataSource.length; i++) {

this.container.createEmbeddedView(this.template,

new PaIteratorContext(this.dataSource[i]));

}

}

}

class PaIteratorContext {

constructor(public $implicit: any) {}

}

**14.4 🡪 iterator.directive.ts**

import { Directive, ViewContainerRef, TemplateRef,

Input} from "@angular/core";

@Directive({

selector: "[paForOf]"

})

export class PaIteratorDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) {}

@Input("paForOf")

dataSource: any;

ngOnInit() {

this.container.clear();

for (let i = 0; i < this.dataSource.length; i++) {

this.container.createEmbeddedView(this.template,

**new PaIteratorContext(this.dataSource[i],i,this.dataSource.length));**

}

}

}

class PaIteratorContext {

**odd: boolean; even: boolean;**

**first: boolean; last: boolean;**

**constructor(public $implicit: any,**

**public index: number, total: number) {**

**this.odd = index % 2 == 1;**

**this.even = !this.odd;**

**this.first = index == 0;**

**this.last = index == total - 1;**

**}**

}

**14.5 🡪 product.component.html**

<div class="p-2">

<div class="form-check m-2">

<input type="checkbox" class="form-check-input" [(ngModel)]="showTable" />

<label class="form-check-label">Show Table</label>

</div>

<table \*paIf="showTable" class="table table-bordered table-striped">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

**<ng-template [paForOf]="getProducts()" let-item let-i="index" let-odd="odd" let-even="even">**

**<tr>**

**<td>{{i+1}}</td>**

**<td [class.bg-info]="odd" [class.bg-warning]="even">{{item.name}}</td>**

**<td>{{item.category}}</td>**

**<td>{{item.price}}</td>**

**</tr>**

**</ng-template>**

</tbody>

</table>

</div>

**14.6 🡪 iterator.directive.ts**

import {

Directive, ViewContainerRef, TemplateRef,

Input, SimpleChange

} from "@angular/core";

@Directive({

selector: "[paForOf]"

})

export class PaIteratorDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) { }

@Input("paForOf")

dataSource: any;

**ngOnInit() {**

**this.updateContent();**

**}**

**ngDoCheck() {**

**console.log("ngDoCheck Called");**

**this.updateContent();**

**}**

**private updateContent() {**

**this.container.clear();**

**for (let i = 0; i < this.dataSource.length; i++) {**

**this.container.createEmbeddedView(this.template,**

**new PaIteratorContext(this.dataSource[i], i, this.dataSource.length));**

**}**

**}**

}

class PaIteratorContext {

odd: boolean; even: boolean;

first: boolean; last: boolean;

constructor(public $implicit: any,

public index: number, total: number) {

this.odd = index % 2 == 1;

this.even = !this.odd;

this.first = index == 0;

this.last = index == total - 1;

**//setInterval(() => {**

**// this.odd = !this.odd; this.even = !this.even;**

**// this.$implicit.price++;**

**//}, 2000);**

}

}

**14.7 🡪 product.component.html**

**<div class="container-fluid">**

**<div class="row p-2">**

**<div class="col-4">**

**<form class="m-2" (ngSubmit)="submitForm()">**

**<div class="form-group">**

**<label>Name</label>**

**<input class="form-control" name="name"**

**[(ngModel)]="newProduct.name" />**

**</div>**

**<div class="form-group">**

**<label>Category</label>**

**<input class="form-control" name="category"**

**[(ngModel)]="newProduct.category" />**

**</div>**

**<div class="form-group">**

**<label>Price</label>**

**<input class="form-control" name="price"**

**[(ngModel)]="newProduct.price" />**

**</div>**

**<button class="btn btn-primary mt-2" type="submit">Create</button>**

**</form>**

**</div>**

<div class="col">

**<table class="table table-sm table-bordered table-striped">**

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th></tr>

</thead>

<tbody>

<tr \*paFor="let item of getProducts(); let i = index; let odd = odd;

let even = even" [class.table-info]="odd"

[class.table-warning]="even" class="align-middle">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>**{{**item.price**}}**</td>

</tr>

</tbody>

</table>

</div>

</div>

</div>

**14.8 🡪 iterator.directive.ts**

**import {**

**Directive, ViewContainerRef, TemplateRef, Input,**

**IterableDiffer, IterableDiffers, IterableChangeRecord,**

**ViewRef**

**} from "@angular/core";**

@Directive({

selector: "[paForOf]"

})

export class PaIteratorDirective {

**private differ: IterableDiffer<any> | undefined;**

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>,

**private differs: IterableDiffers**) { }

@Input("paForOf")

dataSource: any;

ngOnInit() {

**this.differ =**

**<IterableDiffer<any>>this.differs.find(this.dataSource).create();**

}

ngDoCheck() {

**let changes = this.differ?.diff(this.dataSource);**

**if (changes != null) {**

**console.log("ngDoCheck called, changes detected");**

**let arr: IterableChangeRecord<any>[] = [];**

**changes.forEachAddedItem(addition => arr.push(addition));**

**arr.forEach(addition => {**

**if (addition.currentIndex != null) {**

**this.container.createEmbeddedView(this.template,**

**new PaIteratorContext(addition.item, addition.currentIndex,**

**arr.length));**

**}**

**});**

**}**

}

**//private updateContent() {**

**// this.container.clear();**

**// for (let i = 0; i < this.dataSource.length; i++) {**

**// this.container.createEmbeddedView(this.template,**

**// new PaIteratorContext(this.dataSource[i], i, this.dataSource.length));**

**// }**

**//}**

}

class PaIteratorContext {

odd: boolean = false; even: boolean = false;

first: boolean = false; last: boolean = false;

constructor(public $implicit: any,

public index: number, total: number) {

this.odd = index % 2 == 1;

this.even = !this.odd;

this.first = index == 0;

this.last = index == total - 1;

}

}

**14.9 🡪 iterator.directive.ts**

**import {**

**Directive, ViewContainerRef, TemplateRef, Input,**

**IterableDiffer, IterableDiffers, ChangeDetectorRef, IterableChangeRecord,**

**ViewRef**

**} from "@angular/core";**

@Directive({

selector: "[paForOf]"

})

export class PaIteratorDirective {

private differ: IterableDiffer<any> | undefined;

**private views: Map<any, PaIteratorContext> = new Map<any, PaIteratorContext>();**

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>,

private differs: IterableDiffers,

private changeDetector: ChangeDetectorRef) { }

@Input("paForOf")

dataSource: any;

ngOnInit() {

this.differ =

<IterableDiffer<any>>this.differs.find(this.dataSource).create();

}

ngDoCheck() {

let changes = this.differ?.diff(this.dataSource);

if (changes != null) {

let arr: IterableChangeRecord<any>[] = [];

changes.forEachAddedItem(addition => arr.push(addition));

arr.forEach(addition => {

if (addition.currentIndex != null) {

**let context = new PaIteratorContext(addition.item,**

**addition.currentIndex, arr.length);**

**context.view = this.container.createEmbeddedView(this.template,**

**context);**

**this.views.set(addition.trackById, context);**

}

});

**let removals = false;**

**changes.forEachRemovedItem(removal => {**

**removals = true;**

**let context = this.views.get(removal.trackById);**

**if (context != null && context.view != null) {**

**this.container.remove(this.container.indexOf(context.view));**

**this.views.delete(removal.trackById);**

**}**

**});**

**if (removals) {**

**let index = 0;**

**this.views.forEach(context =>**

**context.setData(index++, this.views.size));**

**}**

}

}

}

class PaIteratorContext {

**index: number = 0;**

**odd: boolean = false; even: boolean = false;**

**first: boolean = false; last: boolean = false;**

**view: ViewRef | undefined;**

constructor(public $implicit: any,

public position: number, total: number) {

**this.setData(position, total);**

}

**setData(index: number, total: number) {**

**this.index = index;**

**this.odd = index % 2 == 1;**

**this.even = !this.odd;**

**this.first = index == 0;**

**this.last = index == total - 1;**

**}**

}

**14.10 🡪 cellColor.directive.ts**

import { Directive, HostBinding } from "@angular/core";

@Directive({

selector: "td"

})

export class PaCellColor {

@HostBinding("class")

bgClass: string = "";

setColor(dark: Boolean) {

this.bgClass = dark ? "table-dark" : "";

}

}

**14.11 🡪 cellColorSwitcher.directive.ts**

import { Directive, Input, SimpleChanges, ContentChild }

from "@angular/core";

import { PaCellColor } from "./cellColor.directive";

@Directive({

selector: "table"

})

export class PaCellColorSwitcher {

@Input("paCellDarkColor")

modelProperty: Boolean | undefined;

@ContentChild(PaCellColor)

contentChild: PaCellColor | undefined;

ngOnChanges(changes: SimpleChanges) {

if (this.contentChild != null) {

this.contentChild.setColor(changes["modelProperty"].currentValue);

}

}

}

**14.12 🡪 cellColorSwitcher.directive.ts**

**import { Directive, Input, SimpleChanges, ContentChildren, QueryList }**

**from "@angular/core";**

import { PaCellColor } from "./cellColor.directive";

@Directive({

selector: "table"

})

export class PaCellColorSwitcher {

@Input("paCellDarkColor")

modelProperty: Boolean | undefined;

**@ContentChildren(PaCellColor, { descendants: true })**

**contentChildren: QueryList<PaCellColor> | undefined;**

**ngOnChanges(changes: SimpleChanges) {**

**this.updateContentChildren(changes["modelProperty"].currentValue);**

**}**

**private updateContentChildren(dark: Boolean) {**

**if (this.contentChildren != null && dark != undefined) {**

**this.contentChildren.forEach((child, index) => {**

**child.setColor(index % 2 ? dark : !dark);**

**});**

**}**

**}**

}

**15. COMPONENTS**

**15.1 🡪 product-table.component.ts**

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

@Component({

selector: 'app-product-table',

imports: [],

template: "<div>This is the table component</div>",

})

export class ProductTableComponent {}

**15.2 🡪 product-form.component.ts**

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

@Component({

selector: 'app-product-form',

imports: [],

template: '<div>This is the form component</div>',

})

export class ProductFormComponent {

}

**15.3 🡪 product-table.component.ts**

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

import { Model } from '../repository.model';

import { Product } from '../product.model';

import { PaIteratorDirective } from '../../directives/iterator.directive';

@Component({

selector: 'app-product-table',

imports: [PaIteratorDirective],

templateUrl: 'product-table.component.html'

})

export class ProductTableComponent {

**@Input("model")**

**dataModel: Model | undefined;**

**getProduct(key: number): Product | undefined {**

**return this.dataModel?.getProduct(key);**

**}**

**getProducts(): Product[] | undefined {**

**return this.dataModel?.getProducts();**

**}**

**deleteProduct(key: number) {**

**this.dataModel?.deleteProduct(key);**

**}**

**showTable: boolean = true;**

}

**15.4 🡪 product-table.component.html**

<table class="table table-bordered table-striped">

<thead>

<tr><th></th><th>Name</th><th>Category</th><th>Price</th><th></th></tr>

</thead>

<tbody>

<tr \*paFor="let item of getProducts(); let i=index; let odd=odd; let even=even">

<td>**{{**i+1**}}**</td>

<td [class.bg-info]="odd" [class.bg-warning]="even">**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>$**{{**item.price**}}**</td>

<td class="text-center">

<button class="btn btn-danger btn-sm"

(click)="deleteProduct(item.id)">

Delete

</button>

</td>

</tr>

</tbody>

</table>

**15.5 🡪 product-form.component.ts**

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

import { Product } from '../product.model';

import { CommonModule } from '@angular/common';

import { FormsModule, NgForm, NgModel, ValidationErrors } from '@angular/forms';

import { Model } from '../repository.model';

@Component({

selector: 'app-product-form',

imports: [FormsModule, CommonModule],

templateUrl: 'product-form.component.html',

})

export class ProductFormComponent {

newProduct: Product = new Product();

model: Model = new Model();

formSubmitted: boolean = false;

@Output("paNewProduct")

newProductEvent = new EventEmitter<Product>();

getMessages(errs: ValidationErrors | null, name: string): string[] {

let messages: string[] = [];

for (let errorName in errs) {

switch (errorName) {

case "required":

messages.push(`You must enter a ${name}`);

break;

case "minlength":

messages.push(`A ${name} must be at least

${errs['minlength'].requiredLength}

characters`);

break;

case "pattern":

messages.push(`The ${name} contains

illegal characters`);

break;

}

}

return messages;

}

getValidationMessages(state: NgModel, thingName?: string) {

let thing: string = state.path?.[0] ?? thingName;

return this.getMessages(state.errors, thing)

}

getFormValidationMessages(form: NgForm): string[] {

let messages: string[] = [];

Object.keys(form.controls).forEach(k => {

this.getMessages(form.controls[k].errors, k)

.forEach(m => messages.push(m));

});

return messages;

}

submitForm(form: NgForm) {

this.formSubmitted = true;

if (form.valid) {

this.newProductEvent.emit(this.newProduct);

this.newProduct = new Product();

form.resetForm();

this.formSubmitted = false;

}

}

}

**15.6 🡪 product-form.component.html**

<form novalidate #form="ngForm" (ngSubmit)="submitForm(form)">

<div class="bg-danger text-white p-2 mb-2"

\*ngIf="formSubmitted && form.invalid">

There are problems with the form

<ul>

<li \*ngFor="let error of getFormValidationMessages(form)">

**{{**error**}}**

</li>

</ul>

</div>

<div class="form-group">

<label>Product Name</label>

<input class="form-control"

name="name"

[(ngModel)]="newProduct.name"

#name="ngModel"

required

minlength="5"

pattern="^[A-Za-z ]+$" />

<ul class="text-danger list-unstyled mt-1" \*ngIf="(formSubmitted || name.dirty) && name.invalid">

<li \*ngFor="let error of getValidationMessages(name)">**{{**error**}}**</li>

</ul>

</div>

<div class="form-group">

<label>Category</label>

<input class="form-control" name="category"

[(ngModel)]="newProduct.category" required />

</div>

<div class="form-group">

<label>Price</label>

<input class="form-control" name="price"

[(ngModel)]="newProduct.price" required type="number" />

</div>

<button class="btn btn-primary mt-2" type="submit"

[disabled]="formSubmitted && form.invalid"

[class.btn-secondary]="formSubmitted && form.invalid">

Create

</button>

</form>

**15.7 🡪 toggle-view.component.html**

import { CommonModule } from '@angular/common';

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

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

@Component({

selector: 'app-toggle-view',

standalone: true,

imports: [FormsModule,CommonModule],

templateUrl: './toggle-view.component.html',

styleUrl: './toggle-view.component.css'

})

export class ToggleViewComponent {

showContent: boolean = true;

}

**15.8 🡪 toggle-view.component.html**

<div class="form-check">

<label class="form-check-label">

Show Content

<input class="form-check-input" type="checkbox" [(ngModel)]="showContent" />

</label>

</div>

<ng-content \*ngIf="showContent"></ng-content>

**16. PIPES**

**16.1 🡪 addTax.pipe.ts**

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

@Pipe({

name: 'addTax'

})

export class AddTaxPipe implements PipeTransform {

defaultRate: number = 10;

transform(value: any, rate?: any): number {

let valueNumber = Number.parseFloat(value);

let rateNumber = rate == undefined ?

this.defaultRate : Number.parseInt(rate);

let returnValue = valueNumber + (valueNumber \* (rateNumber / 100));

return parseFloat(returnValue.toFixed(2));

}

}

**16.2 🡪 product-table.component.html**

**<div class= "my-2">**

**<label>Tax Rate:</label>**

**<select class= "form-select" [value]="taxRate || 0" (change)="taxRate=$any($event).target.value">**

**<option value="0">None</option>**

**<option value="10">10%</option>**

**<option value="20">20%</option>**

**<option value="50">50%</option>**

**</select>**

**</div>**

<table class="table table-sm table-bordered table-striped">

<thead class="table-light">

<tr><th></th><th>Name</th><th>Category</th><th>Price</th><th></th></tr>

</thead>

<tbody>

<tr \*paFor="let item of getProducts(); let i = index; let odd = odd;

let even = even" [class.table-info]="odd" [class.table-warning]="even"

class="align-middle">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

**<td>{{item.price | addTax:(taxRate || 0)}}</td>**

<td class="text-center">

<button class="btn btn-danger btn-sm"

(click)="deleteProduct(item.id)">

Delete

</button>

</td>

</tr>

</tbody>

</table>

**16.3 🡪 category-filter.pipe.ts**

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

import { Product } from '../product/product.model';

@Pipe({

name: 'filter',

pure:true

})

export class CategoryFilterPipe implements PipeTransform {

transform(products: Product[] | undefined, category: string | undefined): Product[] {

if (!products) {

return [];

}

// Si la categoría es 'None', mostramos todos los productos

if (category === 'None' || category === undefined) {

return products;

}

// Filtramos los productos según la categoría seleccionada

return products.filter(p => p.category === category);

}

}

**16.4 🡪 productTable.component.html**

<div class="my-2">

<label>Tax Rate:</label>

<select class="form-select" [value]="taxRate || 0" (change)="taxRate=$any($event).target.value">

<option value="0">None</option>

<option value="10">10%</option>

<option value="20">20%</option>

<option value="50">50%</option>

</select>

</div>

<div class="my-2">

<label>Category Filter:</label>

<select class="form-select" [(ngModel)]="categoryFilter">

<option>None</option>

<option>Watersports</option>

<option>Soccer</option>

<option>Chess</option>

</select>

</div>

<table class="table table-sm table-bordered table-striped">

<thead class="table-light">

<tr><th></th><th>Name</th><th>Category</th><th>Price</th><th></th></tr>

</thead>

<tbody>

<tr \*paFor="let item of getProducts() | filter:categoryFilter;

let i = index; let odd = odd;

let even = even" [class.table-info]="odd" [class.table-warning]="even"

class="align-middle">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>**{{**item.price | addTax:(taxRate || 0) | currency:"USD":"symbol"**}}**</td>

<td class="text-center">

<button class="btn btn-danger btn-sm"

(click)="deleteProduct(item.id)">

Delete

</button>

</td>

</tr>

</tbody>

</table>

**16.5 🡪 productTable.component.html**

<div class="my-2">

<label>Tax Rate:</label>

<select class="form-select" [value]="taxRate || 0" (change)="taxRate=$any($event).target.value">

<option value="0">None</option>

<option value="10">**{{** 0.1 | percent**}}**</option>

<option value="20">**{{** 0.2 | percent**}}**</option>

<option value="50">**{{** 0.5 | percent**}}**</option>

<option value="150">**{{** 1.5 | percent**}}**</option>

</select>

table class="table table-sm table-bordered table-striped">

<thead class="table-light">

<tr><th></th><th>Name</th><th>Category</th><th>Price</th><th></th></tr>

</thead>

<tbody>

<tr \*paFor="let item of getProducts()

let i = index; let odd = odd;

let even = even" [class.table-info]="odd" [class.table-warning]="even"

class="align-middle">

<td>**{{**i + 1**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

**<td>**

**{{**item.price | addTax:(taxRate ?? 0) | currency:"EUR":"symbol":"1.0-2"**}} </td>**

<td class="text-center">

<button class="btn btn-danger btn-sm"

(click)="deleteProduct(item.id)">

Delete

</button>

</td>

</tr>

</tbody>

</table>

</div>

**17. SERVICES**

**17.1 🡪 discount.service.ts**

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

@Injectable({

providedIn: 'root'// Esto asegura que el servicio es singleton y global,lo puede usar cualquier componente.

})

export class DiscountService {

private discountValue: number = 10;

public get discount(): number {

return this.discountValue;

}

public set discount(newValue: number) {

this.discountValue = newValue ?? 0;

}

public applyDiscount(price: number) {

return Math.max(price - this.discountValue, 5);

}

constructor() { }

}

**17.2 🡪 discount-display.component.ts**

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

import { DiscountService } from '../../services/discount.service';

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

@Component({

selector: 'app-discount-display',

imports: [FormsModule],

template: `<div class="bg-info text-white p-2 my-2">

The discount is {{discounter?.discount}}

</div>`

})

export class DiscountDisplayComponent {

@Input("discounter")

discounter?: DiscountService;

}

**17.3 🡪 discount-editor.component.ts**

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

import { DiscountService } from '../../services/discount.service';

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

@Component({

selector: 'app-discount-editor',

imports: [FormsModule],

template: `<div class="form-group">

<label>Discount</label>

<input [(ngModel)]="discounter!.discount"

class="form-control"

type="number" />

</div>`

})

export class DiscountEditorComponent {

@Input("discounter")

discounter?: DiscountService;

}

**17.4 🡪 discount.pipe.ts**

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

import { DiscountService } from '../services/discount.service';

@Pipe({

name: 'discount',

pure: false

})

export class DiscountPipe implements PipeTransform {

constructor(private discount: DiscountService) { }

transform(price: number): number {

return this.discount.applyDiscount(price);

}

}

**17.5 🡪 discount-amount.directive.ts**

import { Directive, Input, KeyValueDiffer, KeyValueDiffers, SimpleChange } from '@angular/core';

import { DiscountService } from '../services/discount.service';

@Directive({

selector: "td[pa-price]",

exportAs: "discount"

})

export class DiscountAmountDirective {

private differ?: KeyValueDiffer<any, any>;

constructor(private keyValueDiffers: KeyValueDiffers,

private discount: DiscountService) { }

@Input("pa-price")

originalPrice?: number;

discountAmount?: number;

ngOnInit() {

this.differ =

this.keyValueDiffers.find(this.discount).create();

}

ngOnChanges(changes: { [property: string]: SimpleChange }) {

if (changes["originalPrice"] != null) {

this.updateValue();

}

}

ngDoCheck() {

if (this.differ?.diff(this.discount) != null) {

this.updateValue();

}

}

updateValue() {

const price = this.originalPrice ?? 0; // Si `originalPrice` es `undefined`, usar 0

this.discountAmount = price - this.discount.applyDiscount(price);

}

}

**20. CARACTERISTICAS AVANZADAS**

**20.1 🡪 product.model.ts**

export class Product {

constructor(public id?: number,

public name?: string,

public category?: string,

public price?: number) {}

}

**20.2 🡪 static.datasource.ts**

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

import { Product } from "./product.model";

@Injectable()

export class StaticDataSource {

private data: Product[];

constructor() {

this.data = new Array<Product>(

new Product(1, "Kayak", "Watersports", 275),

new Product(2, "Lifejacket", "Watersports", 48.95),

new Product(3, "Soccer Ball", "Soccer", 19.50),

new Product(4, "Corner Flags", "Soccer", 34.95),

new Product(5, "Thinking Cap", "Chess", 16));

}

getData(): Product[] {

return this.data;

}

}

**20.3 🡪 repository.model.ts**

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

import { Product } from "./product.model";

import { StaticDataSource } from "./static.datasource";

@Injectable()

export class Model {

private products: Product[];

private locator = (p: Product, id: number) => p.id == id;

constructor(private dataSource: StaticDataSource) {

this.products = new Array<Product>();

this.dataSource.getData().forEach(p => this.products.push(p));

}

getProducts(): Product[] {

return this.products;

}

getProduct(id: number): Product | undefined {

return this.products.find(p => this.locator(p, id));

}

saveProduct(product: Product) {

if (product.id == 0 || product.id == null) {

product.id = this.generateID();

this.products.push(product);

} else {

let index = this.products

.findIndex(p => this.locator(p, product.id));

this.products.splice(index, 1, product);

}

}

deleteProduct(id: number) {

let index = this.products.findIndex(p => this.locator(p, id));

if (index > -1) {

this.products.splice(index, 1);

}

}

private generateID(): number {

let candidate = 100;

while (this.getProduct(candidate) != null) {

candidate++;

}

return candidate;

}

}

**20.4 🡪 shared.state.service.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService } from '../shared-state.service';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-table',

imports: [CommonModule],

standalone: true,

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

styleUrl: './table.component.css',

providers: []

}export class TableComponent {

constructor(private model: Model, private state: SharedStateService) { }

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

deleteProduct(key?: number) {

if (key != undefined) {

this.model.deleteProduct(key);

}

}

editProduct(key?: number) {

this.state.update(MODES.EDIT, key)

}

createProduct() {

this.state.update(MODES.CREATE);

}

}

)

**20.5 🡪 table.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService } from '../shared-state.service';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-table',

imports: [CommonModule],

standalone: true,

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

styleUrl: './table.component.css',

providers: []

})

export class TableComponent {

constructor(private model: Model, private state: SharedStateService) { }

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

deleteProduct(key?: number) {

if (key != undefined) {

this.model.deleteProduct(key);

}

}

editProduct(key?: number) {

this.state.update(MODES.EDIT, key)

}

createProduct() {

this.state.update(MODES.CREATE);

}

}

**20.6 🡪 table.component.html**

<table class="table table-sm table-bordered table-striped">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Category</th>

<th>Price</th>

<th></th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts()">

<td>**{{**item.id**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>**{{**item.price | currency:"USD" **}}**</td>

<td class="text-center">

<button class="btn btn-danger btn-sm m-1"

(click)="deleteProduct(item.id)">

Delete

</button>

<button class="btn btn-warning btn-sm" (click)="editProduct(item.id)">

Edit

</button>

</td>

</tr>

</tbody>

</table>

<button class="btn btn-primary mt-1" (click)="createProduct()">

Create New Product

</button>

**20.7 🡪 message.model.ts**

export class Message {

constructor(public text: string,

public error: boolean = false) { }

}

**20.8 🡪 message.service.ts**

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

import { Observable, ReplaySubject, Subject } from 'rxjs';

import { Message } from './message.model';

@Injectable({

providedIn: 'root'

})

export class MessageService {

messages: Observable<Message> = new ReplaySubject<Message>(1);

reportMessage(msg: Message) {

(this.messages as Subject<Message>).next(msg);

}

}

**20.9 🡪 message.component.ts**

import { MessageService } from '../message.service';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-message',

imports: [CommonModule],

standalone: true,

providers: [],

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

styleUrl: './message.component.css'

})

export class MessageComponent {

lastMessage?: Message;

constructor(messageService: MessageService) {

messageService.messages.subscribe(msg => this.lastMessage = msg);

}

}

**20.10 🡪 message.component.html**

<div \*ngIf="lastMessage"

class="bg-info text-white p-2 text-center"

[class.bg-danger]="lastMessage.error">

<h4>**{{**lastMessage.text**}}**</h4>

</div>

**20.11 🡪 form.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService, StateUpdate } from '../shared-state.service';

import { MessageService } from '../../messages/message.service';

import { Message } from '../../messages/message.model';

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

@Component({

selector: 'app-form',

imports: [FormsModule],

standalone: true,

providers: [],

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

styleUrl: './form.component.css'

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

constructor(private model: Model, private state: SharedStateService,

private messageService: MessageService) {

this.state.changes.subscribe((upd) => this.handleStateChange(upd))

this.messageService.reportMessage(new Message("Creating New Product"));

}

handleStateChange(newState: StateUpdate) {

this.editing = newState.mode == MODES.EDIT;

if (this.editing && newState.id) {

Object.assign(this.product, this.model.getProduct(newState.id)

?? new Product());

this.messageService.reportMessage(

new Message(`Editing ${this.product.name}`));

} else {

this.product = new Product();

this.messageService.reportMessage(new Message("Creating New Product"));

}

}

submitForm(form: NgForm) {

if (form.valid) {

//console.log("Formulario enviado", this.product)

this.model.saveProduct(this.product);

this.product = new Product();

form.resetForm();

}

else {

console.log("Formulario no valido");

}

}

}

**20.12 🡪 form.component.html**

<form #form="ngForm" (ngSubmit)="submitForm(form)" (reset)="form.resetForm()">

<div class="form-group">

<label>Name</label>

<input class="form-control" name="name"

[(ngModel)]="product.name" required />

</div>

<div class="form-group">

<label>Category</label>

<input class="form-control" name="category"

[(ngModel)]="product.category" required />

</div>

<div class="form-group">

<label>Price</label>

<input class="form-control" name="price"

[(ngModel)]="product.price"

required pattern="^[0-9\.]+$" />

</div>

<div class="mt-2">

<button type="submit" class="btn btn-primary"

[class.btn-warning]="editing" [disabled]="form.invalid">

**{{**editing ? "Save" : "Create"**}}**

</button>

<button type="reset" class="btn btn-secondary m-1">Cancel</button>

</div>

</form>

**20.13 🡪 form.component.css**

input.ng-dirty.ng-invalid { border: 2px solid #ff0000 }

input.ng-dirty.ng-valid { border: 2px solid #6bc502 }

**20.14 🡪 index.html**

<!doctype html>

<html lang="en">

<head>

<meta charset="utf-8">

<title>ExampleApp</title>

<base href="/">

<meta name="viewport" content="width=device-width, initial-scale=1">

<link rel="icon" type="image/x-icon" href="favicon.ico">

</head>

<body>

<app-root></app-root>

</body>

</html>

**21. FORMS API PART I**

**21.1 🡪 form.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService, StateUpdate } from '../shared-state.service';

import { MessageService } from '../../messages/message.service';

import { Message } from '../../messages/message.model';

import { FormControl, FormsModule, NgForm, ReactiveFormsModule, Validators } from '@angular/forms';

@Component({

selector: 'app-form',

imports: [FormsModule, ReactiveFormsModule],

standalone: true,

providers: [],

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

styleUrl: './form.component.css’

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

**nameField: FormControl = new FormControl("", {**

**validators: [**

**Validators.required,**

**Validators.minLength(3),**

**Validators.pattern("^[A-Za-z ]+$")**

**],**

**updateOn: "change"**

});

.../

**21.2 🡪 validation-helper.pipe.ts**

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

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

@Pipe({

name: 'validationFormat'

})

export class ValidationHelperPipe implements PipeTransform {

transform(source: any, name: any): string[] {

if (source instanceof FormControl) {

return this.formatMessages((source as FormControl).errors, name)

}

return this.formatMessages(source as ValidationErrors, name)

}

formatMessages(errors: ValidationErrors | null, name: string): string[] {

let messages: string[] = [];

for (let errorName in errors) {

switch (errorName) {

case "required":

messages.push(`You must enter a ${name}`);

break;

case "minlength":

messages.push(`A ${name} must be at least

${errors['minlength'].requiredLength}

characters`);

break;

case "pattern":

messages.push(`The ${name} contains

illegal characters`);

break;

}

}

return messages;

}

}

**21.3 🡪 form.component.html**

<div class="form-group">

<label>Name</label>

**<input class="form-control" name="name" [formControl]="nameField" #name="ngForm" />**

**<ul class="text-danger list-unstyled mt-1" \*ngIf="name.dirty && name.invalid">**

**<li \*ngFor="let err of name.errors | validationFormat:'name'">**

**{{ err }}**

**</li>**

**</ul>**

</div>

**21.4 🡪 form.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService, StateUpdate } from '../shared-state.service';

import { MessageService } from '../../messages/message.service';

import { Message } from '../../messages/message.model';

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

import { ValidationHelperPipe } from '../validation-helper.pipe';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-form',

imports: [FormsModule, ReactiveFormsModule, ValidationHelperPipe, CommonModule],

standalone: true,

providers: [],

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

styleUrl: './form.component.css'

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

nameField: FormControl = new FormControl("", {

validators: [

Validators.required,

Validators.minLength(3),

Validators.pattern("^[A-Za-z ]+$")

],

updateOn: "change"

});

categoryField: FormControl = new FormControl();

productForm: FormGroup = new FormGroup({

name: this.nameField, category: this.categoryField

});

constructor(private model: Model, private state: SharedStateService,

private messageService: MessageService) {

this.state.changes.subscribe((upd) => this.handleStateChange(upd))

this.messageService.reportMessage(new Message("Creating New Product"));

}

ngOnInit() {

this.productForm.statusChanges.subscribe(newStatus => {

if (newStatus == "INVALID") {

let invalidControls: string[] = [];

for (let controlName in this.productForm.controls) {

if (this.productForm.controls[controlName].invalid) {

invalidControls.push(controlName)

}

}

this.messageService.reportMessage(new Message(`INVALID: ${invalidControls.join(", ")}`))

} else {

this.messageService.reportMessage(new Message(newStatus));

}

})

}

handleStateChange(newState: StateUpdate) {

this.editing = newState.mode == MODES.EDIT;

if (this.editing && newState.id) {

Object.assign(this.product, this.model.getProduct(newState.id)

?? new Product());

this.messageService.reportMessage(

new Message(`Editing ${this.product.name}`));

//this.nameField.setValue(this.product.name);

//this.categoryField.setValue(this.product.category);

} else {

this.product = new Product();

this.messageService.reportMessage(new Message("Creating New Product"));

//this.nameField.setValue("");

//this.categoryField.setValue("");

}

}

}

**21.5 🡪 form.component.ts**

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

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

import { Product } from "../model/product.model";

import { Model } from "../model/repository.model"

import { Message } from "../messages/message.model"

import { MessageService } from "../messages/message.service";

import { MODES, SharedState, StateUpdate } from "./sharedState.service";

@Component({

selector: "paForm",

templateUrl: "form.component.html",

styleUrls: ["form.component.css"]

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

**//nameField: FormControl = new FormControl("", {**

**// validators: [**

**// Validators.required,**

**// Validators.minLength(3),**

**// Validators.pattern("^[A-Za-z ]+$")**

**// ],**

**// updateOn: "change"**

**//});**

**//categoryField: FormControl = new FormControl();**

**productForm: FormGroup = new FormGroup({**

**name: new FormControl("", {**

**validators: [**

**Validators.required,**

**Validators.minLength(3),**

**Validators.pattern("^[A-Za-z ]+$")**

**],**

**updateOn:"change"**

**}),**

**category: new FormControl()**

**});**

…/

**21.6 🡪 form.component.html**

**<form [formGroup]="productForm" #form="ngForm"**

**(ngSubmit)="submitForm()" (reset)="resetForm()">**

<div class="form-group">

<label>Name</label>

<input class="form-control" formControlName="name" />

</div>

**<div class="form-group">**

**<label>Category</label>**

**<input class="form-control" formControlName="category" />**

**</div>**

**<div class="form-group">**

**<label>Price</label>**

**<input class="form-control" formControlName="price" />**

**</div>**

**<div class="mt-2">**

**<button type="submit" class="btn btn-primary"**

**[class.btn-warning]="editing"**

**[disabled]="form.invalid">**

**{{editing ? "Save" : "Create"}}**

**</button>**

**<button type="reset" class="btn btn-secondary m-1">Cancel</button>**

**</div>**

</form>

**21.7 🡪 form.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService, StateUpdate } from '../shared-state.service';

import { MessageService } from '../../messages/message.service';

import { Message } from '../../messages/message.model';

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

import { ValidationHelperPipe } from '../validation-helper.pipe';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-form',

imports: [FormsModule, ReactiveFormsModule, ValidationHelperPipe, CommonModule],

standalone: true,

providers: [],

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

styleUrl: './form.component.css'

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

productForm: FormGroup = new FormGroup({

name: new FormControl("", {

validators: [

Validators.required,

Validators.minLength(3),

Validators.pattern("^[A-Za-z ]+$")

],

updateOn:"change"

}),

**category: new FormControl("", { validators: Validators.required }),**

**price: new FormControl("", {**

**validators: [Validators.required, Validators.pattern("^[0-9\.]+$")]**

**})**

});

constructor(private model: Model, private state: SharedState,

private messageService: MessageService) {

this.state.changes.subscribe((upd) => this.handleStateChange(upd))

this.messageService.reportMessage(new Message("Creating New Product"));

}

**//ngOnInit() {**

**// this.productForm.statusChanges.subscribe(newStatus => {**

**// if (newStatus == "INVALID") {**

**// let invalidControls: string[] = [];**

**// for (let controlName in this.productForm.controls) {**

**// if (this.productForm.controls[controlName].invalid) {**

**// invalidControls.push(controlName)**

**// }**

**// }**

**// this.messageService.reportMessage(new Message(`INVALID: ${invalidControls.join(", ")}`))**

**// } else {**

**// this.messageService.reportMessage(new Message(newStatus));**

**// }**

**// })**

**//}**

handleStateChange(newState: StateUpdate) {

this.editing = newState.mode == MODES.EDIT;

if (this.editing && newState.id) {

Object.assign(this.product, this.model.getProduct(newState.id)

?? new Product());

this.messageService.reportMessage(

new Message(`Editing ${this.product.name}`));

} else {

this.product = new Product();

this.messageService.reportMessage(new Message("Creating New Product"));

}

**this.productForm.reset(this.product);**

}

**submitForm() {**

**if (this.productForm.valid) {**

**Object.assign(this.product, this.productForm.value);**

**this.model.saveProduct(this.product);**

**this.product = new Product();**

**this.productForm.reset();**

**}**

**}**

**resetForm() {**

**this.editing = true;**

**this.product = new Product();**

**this.productForm.reset();**

**}**

}

**21.8 🡪 validation-errors.directive.ts**

import { Directive, Input, TemplateRef, ViewContainerRef } from '@angular/core';

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

import { ValidationHelperPipe } from './validation-helper.pipe';

@Directive({

selector: '[validationErrors]'

})

export class ValidationErrorsDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) { }

@Input("validationErrorsControl")

name: string = ""

@Input("validationErrorsLabel")

label?: string;

@Input("validationErrors")

formGroup?: FormGroup;

ngOnInit() {

let formatter = new ValidationHelperPipe();

if (this.formGroup && this.name) {

let control = this.formGroup?.get(this.name);

if (control) {

control.statusChanges.subscribe(() => {

if (this.container.length > 0) {

this.container.clear();

}

if (control && control.dirty && control.invalid

&& control.errors) {

formatter.formatMessages(control.errors,

this.label ?? this.name).forEach(err => {

this.container.createEmbeddedView(this.template,

{ $implicit: err });

})

}

})

}

}

}

}

**21.9 🡪 table.component.html**

<table class="table table-sm table-bordered table-striped">

<thead>

<tr>

**<th>ID</th>**

**<th>Name</th>**

**<th>Category</th>**

**<th>Price</th>**

**<th>Details</th>**

**<th></th>**

</tr>

</thead>

<tbody>

**<tr \*ngFor="let item of getProducts()">**

**<td>{{item.id}}</td>**

**<td>{{item.name}}</td>**

**<td>{{item.category}}</td>**

**<td>{{item.price | currency:"USD" }}</td>**

**<td>**

**<ng-container \*ngIf="item.details else empty">**

**{{ item.details?.supplier }}, {{ item.details?.keywords}}**

**</ng-container>**

**<ng-template #empty>(None)</ng-template>**

**</td>**

<td class="text-center">

<button class="btn btn-danger btn-sm m-1"

(click)="deleteProduct(item.id)">

Delete

</button>

<button class="btn btn-warning btn-sm" (click)="editProduct(item.id)">

Edit

</button>

</td>

</tr>

</tbody>

</table>

<button class="btn btn-primary mt-1" (click)="createProduct()">

Create New Product

</button>

**21.10 🡪 form.component.html**

<form [formGroup]="productForm" #form="ngForm"

(ngSubmit)="submitForm()" (reset)="resetForm()">

<div class="form-group">

<label>Name</label>

<input class="form-control" formControlName="name" />

<ul class="text-danger list-unstyled mt-1">

<li \*validationErrors="productForm; control:'name'; let err">

**{{** err **}}**

</li>

</ul>

</div>

<div class="form-group">

<label>Category</label>

<input class="form-control" formControlName="category" />

<ul class="text-danger list-unstyled mt-1">

<li \*validationErrors="productForm; control:'category'; let err">

**{{** err **}}**

</li>

</ul>

<div class="form-group">

<label>Price</label>

<input class="form-control" formControlName="price" />

<ul class="text-danger list-unstyled mt-1">

<li \*validationErrors="productForm; control:'price'; let err">

**{{** err **}}**

</li>

</ul>

</div>

**<ng-container formGroupName="details">**

**<div class="form-group">**

**<label>Supplier</label>**

**<input class="form-control" formControlName="supplier" />**

**<ul class="text-danger list-unstyled mt-1">**

**<li \*validationErrors="productForm; control:'details.supplier';**

**label: 'supplier'; let err">**

**{{ err }}**

**</li>**

**</ul>**

**</div>**

**<div class="form-group">**

**<label>Keywords</label>**

**<input class="form-control" formControlName="keywords" />**

**<ul class="text-danger list-unstyled mt-1">**

**<li \*validationErrors="productForm; control:'details.keywords';**

**label: 'keyword'; let err">**

**{{ err }}**

**</li>**

**</ul>**

**</div>**

**</ng-container>**

<div class="mt-2">

<button type="submit" class="btn btn-primary"

[class.btn-warning]="editing"

[disabled]="form.invalid">

**{{**editing ? "Save" : "Create"**}}**

</button>

<button type="reset" class="btn btn-secondary m-1">Cancel</button>

</div>

</form>

**22. FORMS API PART II**

**22.1 🡪 form.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { MODES, SharedStateService, StateUpdate } from '../shared-state.service';

import { MessageService } from '../../messages/message.service';

import { Message } from '../../messages/message.model';

import { FormArray, FormControl, FormGroup, FormsModule, NgForm, ReactiveFormsModule, Validators } from '@angular/forms';

import { ValidationHelperPipe } from '../validation-helper.pipe';

import { CommonModule } from '@angular/common';

import { ValidationErrorsDirective } from '../validation-errors.directive';

@Component({

selector: 'app-form',

imports: [FormsModule, ReactiveFormsModule,

ValidationHelperPipe, CommonModule, ValidationErrorsDirective],

standalone: true,

providers: [],

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

styleUrl: './form.component.css'

})

export class FormComponent {

product: Product = new Product();

editing: boolean = false;

**keywordGroup = new FormArray([**

**this.createKeywordFormControl(),**

**])**

**productForm: FormGroup = new FormGroup({**

**name: new FormControl("", {**

**validators: [**

**Validators.required,**

**Validators.minLength(3),**

**Validators.pattern("^[A-Za-z ]+$")**

**],**

**updateOn: "change"**

**}),**

**category: new FormControl("", { validators: Validators.required }),**

**price: new FormControl("", {**

**validators: [Validators.required, Validators.pattern("^[0-9\.]+$")]**

**}),**

**details: new FormGroup({**

**supplier: new FormControl("", { validators: Validators.required }),**

**keywords: this.keywordGroup**

**})**

**});**

**constructor(private model: Model, private state: SharedState,**

**private messageService: MessageService) {**

**this.state.changes.subscribe((upd) => this.handleStateChange(upd))**

**this.messageService.reportMessage(new Message("Creating New Product"));**

**}**

**// ngOnInit() {**

**// this.productForm.get("details")?.statusChanges.subscribe(newStatus => {**

**// this.messageService.reportMessage(new Message(`Details ${newStatus}`));**

**// })**

**// }**

handleStateChange(newState: StateUpdate) {

this.editing = newState.mode == MODES.EDIT;

**this.keywordGroup.clear();**

if (this.editing && newState.id) {

Object.assign(this.product, this.model.getProduct(newState.id)

?? new Product());

this.messageService.reportMessage(

new Message(`Editing ${this.product.name}`));

**this.product.details?.keywords?.forEach(val => {**

**this.keywordGroup.push(this.createKeywordFormControl());**

**})**

} else {

this.product = new Product();

this.messageService.reportMessage(new Message("Creating New Product"));

}

**if (this.keywordGroup.length == 0) {**

**this.keywordGroup.push(this.createKeywordFormControl());**

**}**

this.productForm.reset(this.product);

}

submitForm() {

if (this.productForm.valid) {

Object.assign(this.product, this.productForm.value);

this.model.saveProduct(this.product);

this.product = new Product();

**this.keywordGroup.clear();**

**this.keywordGroup.push(this.createKeywordFormControl());**

this.productForm.reset();

}

}

resetForm() {

**this.keywordGroup.clear();**

**this.keywordGroup.push(this.createKeywordFormControl());**

this.editing = true;

this.product = new Product();

this.productForm.reset();

}

**createKeywordFormControl(): FormControl {**

**return new FormControl();**

**}**

}

**22.2 🡪 limit.ts**

import { AbstractControl, ValidationErrors, ValidatorFn } from "@angular/forms";

export class LimitValidator {

static Limit(limit: number): ValidatorFn {

return (control: AbstractControl): ValidationErrors | null => {

let val = parseFloat(control.value);

if (isNaN(val) || val > limit) {

return { "limit": { "limit": limit, "actualValue": val } };

}

return null;

}

}

}

**22.3 🡪 hillow.directive.ts**

import { Directive, Input, SimpleChanges } from "@angular/core";

import { AbstractControl, NG\_VALIDATORS, ValidationErrors,

Validator, ValidatorFn } from "@angular/forms";

export class HiLowValidator {

static HiLow(high:number, low: number) : ValidatorFn {

return (control: AbstractControl) : ValidationErrors | null => {

let val = parseFloat(control.value);

if (isNaN(val) || val > high || val < low) {

return {"hilow": {"high": high, "low": low, "actualValue": val}};

}

return null;

}

}

}

@Directive({

selector: 'input[high][low]',

providers: [{provide: NG\_VALIDATORS, useExisting: HiLowValidatorDirective,

multi: true}]

})

export class HiLowValidatorDirective implements Validator {

@Input()

high: number | string | undefined

@Input()

low: number | string | undefined

validator?: (control: AbstractControl) => ValidationErrors | null;

ngOnChanges(changes: SimpleChanges): void {

if ("high" in changes || "low" in changes) {

let hival = typeof(this.high) == "string"

? parseInt(this.high) : this.high;

let loval = typeof(this.low) == "string"

? parseInt(this.low) : this.low;

this.validator = HiLowValidator.HiLow(hival ?? Number.MAX\_VALUE,

loval ?? 0);

}

}

validate(control: AbstractControl): ValidationErrors | null {

return this.validator?.(control) ?? null;

}

}

**22.4 🡪 unique.ts**

import { AbstractControl, FormArray, ValidationErrors, ValidatorFn }

from "@angular/forms";

export class UniqueValidator {

static unique(): ValidatorFn {

return (control: AbstractControl): ValidationErrors | null => {

if (control instanceof FormArray) {

let badElems = control.controls.filter((child, index) => {

return control.controls.filter((c, i2) => i2 != index).

some(target => target.value != ""

&& target.value == child.value);

});

if (badElems.length > 0) {

return { "unique": {} };

}

}

return null;

}

}

}

**22.5 🡪 validation.helper.ts**

import { Pipe } from "@angular/core";

import { FormControl, ValidationErrors } from "@angular/forms";

@Pipe({

name: "validationFormat"

})

export class ValidationHelper implements PipeTransform {

transform(source: any, name: any): string[] {

if (source instanceof FormControl) {

return this.formatMessages((source as FormControl).errors, name)

}

return this.formatMessages(source as ValidationErrors, name)

}

formatMessages(errors: ValidationErrors | null, name: string): string[] {

let messages: string[] = [];

for (let errorName in errors) {

switch (errorName) {

case "required":

messages.push(`You must enter a ${name}`);

break;

case "minlength":

messages.push(`A ${name} must be at least

${errors['minlength'].requiredLength}

characters`);

break;

case "pattern":

messages.push(`The ${name} contains

illegal characters`);

break;

case "limit":

messages.push(`The ${name} must be less than

${errors['limit'].limit}`);

break;

case "hilow":

messages.push(`The ${name} must be between

${errors['hilow'].low} and ${errors['hilow'].high}`);

break;

case "unique":

messages.push(`The ${name} must be unique`);

break;

}

}

return messages;

}

}

**22.6 🡪 unique.ts**

import { AbstractControl, FormArray, ValidationErrors, ValidatorFn }

from "@angular/forms";

export class UniqueValidator {

**static uniquechild(control: AbstractControl) : ValidationErrors | null {**

**return control.parent?.hasError("unique") ? {"unique-child": {}} : null;**

**}**

static unique() : ValidatorFn {

return (control: AbstractControl) : ValidationErrors | null => {

**let badElems: AbstractControl[] = [];**

**let goodElems: AbstractControl[] = [];**

if (control instanceof FormArray) {

control.controls.forEach((child, index) => {

**if (control.controls.filter((c, i2) => i2 != index)**

**.some(target => target.value != ""**

**&& target.value == child.value)) {**

**badElems.push(child);**

**} else {**

**goodElems.push(child);**

**}**

})

**setTimeout(() => {**

**badElems.forEach(c => {**

**if (!c.hasValidator(this.uniquechild)) {**

**c.markAsDirty();**

**c.addValidators(this.uniquechild)**

**c.updateValueAndValidity({onlySelf: true,**

**emitEvent: false});**

**}**

**})**

**goodElems.forEach(c => {**

**if (c.hasValidator(this.uniquechild)) {**

**c.removeValidators(this.uniquechild);**

**}**

**c.updateValueAndValidity({ onlySelf: true,**

**emitEvent: false})**

**})**

**}, 0);**

}

**return badElems.length > 0 ? {"unique": {}} : null;**

}

}

}

**22.7 🡪 filteredFormArray.ts**

**import { AbstractControl, FormArray } from "@angular/forms";**

export type ValueFilter = (value: any) => boolean;

export class FilteredFormArray extends FormArray {

filter: ValueFilter | undefined = (val) => val == "" || val == null;

\_updateValue() {

(this as { value: any }).value =

this.controls.filter((control) =>

(control.enabled || this.disabled) && !this.filter?.(control.value)

).map((control) => control.value);

}

**override push(control: AbstractControl,**

**options?: { emitEvent?: boolean | undefined; }): void {**

**super.push(control, options);**

**this.controls.forEach(c => c.updateValueAndValidity());**

**}**

**override removeAt(index: number,**

**options?: { emitEvent?: boolean | undefined; }): void {**

**super.removeAt(index, options);**

**this.controls.forEach(c => c.updateValueAndValidity());**

**}**

}

**22.8 🡪 prohibited.ts**

import { AbstractControl, AsyncValidatorFn, ValidationErrors } from "@angular/forms";

import { Observable, Subject } from "rxjs";

export class ProhibitedValidator {

static prohibitedTerms: string[] = ["ski", "swim"]

static prohibited(): AsyncValidatorFn {

return (control: AbstractControl): Promise<ValidationErrors | null>

| Observable<ValidationErrors | null> => {

let subject = new Subject<ValidationErrors | null>();

setTimeout(() => {

let match = false;

this.prohibitedTerms.forEach(word => {

if ((control.value as string).toLowerCase().indexOf(word) > -1) {

subject.next({ "prohibited": { prohibited: word } })

match = true;

}

});

if (!match) {

subject.next(null);

}

subject.complete();

}, 1000);

return subject;

}

}

}

**23. HTTP REQUEST**

**23.1 🡪 restData.js**

module.exports = function () {

var data = {

products: [

{ id: 1, name: "Kayak", category: "Watersports", price: 275,

details: { supplier: "Acme", keywords: ["boat", "small"]} },

{ id: 2, name: "Lifejacket", category: "Watersports", price: 48.95,

details: { supplier: "Smoot Co", keywords: ["safety"]} },

{ id: 3, name: "Soccer Ball", category: "Soccer", price: 19.50 },

{ id: 4, name: "Corner Flags", category: "Soccer", price: 34.95 },

{ id: 5, name: "Stadium", category: "Soccer", price: 79500 },

{ id: 6, name: "Thinking Cap", category: "Chess", price: 16 },

{ id: 7, name: "Unsteady Chair", category: "Chess", price: 29.95 },

{ id: 8, name: "Human Chess Board", category: "Chess", price: 75 },

{ id: 9, name: "Bling Bling King", category: "Chess", price: 1200 }

]

}

return data

}

**23.2 🡪 rest.datasource.ts**

import { Injectable, Inject, InjectionToken } from "@angular/core";

import { HttpClient, HttpHeaders } from "@angular/common/http";

import { Observable, throwError } from "rxjs";

import { Product } from "./product.model";

export const REST\_URL = new InjectionToken("rest\_url");

@Injectable()

export class RestDataSource {

constructor(private http: HttpClient,

@Inject(REST\_URL) private url: string) { }

getData(): Observable<Product[]> {

return this.http.get<Product[]>(this.url);

}

}

**23.3 🡪 rest.datasource.ts**

import { Injectable, Inject, InjectionToken } from "@angular/core";

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

import { Observable } from "rxjs";

import { Product } from "./product.model";

export const REST\_URL = new InjectionToken("rest\_url");

@Injectable()

export class RestDataSource {

constructor(private http: HttpClient,

@Inject(REST\_URL) private url: string) { }

getData(): Observable<Product[]> {

return this.http.get<Product[]>(this.url);

}

**saveProduct(product: Product): Observable<Product> {**

**return this.http.post<Product>(this.url, product);**

**}**

**updateProduct(product: Product): Observable<Product> {**

**return this.http.put<Product>(`${this.url}/${product.id}`, product);**

**}**

**deleteProduct(id: number): Observable<Product> {**

**return this.http.delete<Product>(`${this.url}/${id}`);**

**}**

}

**23.4 🡪 repository.model.ts**

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

import { Product } from "./product.model";

import { Observable } from "rxjs";

import { RestDataSource } from "./rest.datasource";

@Injectable()

export class Model {

private products: Product[] = new Array<Product>();

private locator = (p: Product, id: number) => p.id == id;

constructor(private dataSource: RestDataSource) {

this.dataSource.getData().subscribe(data => this.products = data);

}

getProducts(): Product[] {

return this.products;

}

getProduct(id: number): Product {

return this.products.find(p => this.locator(p, id));

}

saveProduct(product: Product) {

**if (product.id == 0 || product.id == null) {**

**this.dataSource.saveProduct(product)**

**.subscribe(p => this.products.push(p));**

**} else {**

**this.dataSource.updateProduct(product).subscribe(p => {**

**let index = this.products**

**.findIndex(item => this.locator(item, p.id));**

**this.products.splice(index, 1, p);**

**});**

**}**

}

deleteProduct(id: number) {

**this.dataSource.deleteProduct(id).subscribe(() => {**

**let index = this.products.findIndex(p => this.locator(p, id));**

**if (index > -1) {**

**this.products.splice(index, 1);**

**}**

**})**

}

}

**23.5 🡪 rest.datasource.ts**

import { Injectable, Inject, InjectionToken } from "@angular/core";

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

import { Observable } from "rxjs";

import { Product } from "./product.model";

export const REST\_URL = new InjectionToken("rest\_url");

@Injectable()

export class RestDataSource {

constructor(private http: HttpClient,

@Inject(REST\_URL) private url: string) { }

getData(): Observable<Product[]> {

**return this.sendRequest<Product[]>("GET",this.url);**

}

saveProduct(product: Product): Observable<Product> {

**return this.sendRequest<Product>("POST", this.url,product);**

}

updateProduct(product: Product): Observable<Product> {

**return this.sendRequest<Product>("PUT",`${this.url}/${product.id}`, product);**

}

deleteProduct(id: number): Observable<Product> {

**return this.sendRequest<Product>("DELETE",`${this.url}/${id}`);**

}

**private sendRequest<T>(verb: string, url: string, body?: Product): Observable<T> {**

**return this.http.request<T>(verb, url,{**

**body: body**

**});**

**}**

}

**23.6 🡪 errorHandler.ts**

import { ErrorHandler, Injectable, NgZone } from "@angular/core";

import { MessageService } from "./message.service";

import { Message } from "./message.model";

@Injectable()

export class MessageErrorHandler implements ErrorHandler {

constructor(private messageService: MessageService, private ngZone: NgZone) {

}

handleError(error:any) {

let msg = error instanceof Error ? error.message : error.toString();

this.ngZone.run(() => this.messageService

.reportMessage(new Message(msg, true)), 0);

}

}

**24. ROUTING I**

**24.1 🡪 repository.model.ts**

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

import { Product } from "./product.model";

import { StaticDataSource } from "./static.datasource";

**import { Observable, ReplaySubject } from "rxjs";**

import { RestDataSource } from "./rest.datasource";

@Injectable()

export class Model {

private products: Product[];

private locator = (p: Product, id?: number) => p.id == id;

**private replaySubject: ReplaySubject<Product[]>;**

constructor(private dataSource: RestDataSource) {

this.products = new Array<Product>();

**this.replaySubject = new ReplaySubject<Product[]>(1);**

**this.dataSource.getData().subscribe(data => {**

**this.products = data**

**this.replaySubject.next(data);**

**this.replaySubject.complete();**

**});**

}

getProducts(): Product[] {

return this.products;

}

getProduct(id: number): Product | undefined {

return this.products.find(p => this.locator(p, id));

}

**getProductObservable(id: number): Observable<Product | undefined> {**

**let subject = new ReplaySubject<Product | undefined>(1);**

**this.replaySubject.subscribe(products => {**

**subject.next(products.find(p => this.locator(p, id)));**

**subject.complete();**

**});**

**return subject;**

**}**

saveProduct(product: Product) {

if (product.id == 0 || product.id == null) {

this.dataSource.saveProduct(product)

.subscribe(p => this.products.push(p));

} else {

this.dataSource.updateProduct(product).subscribe(p => {

let index = this.products

.findIndex(item => this.locator(item, p.id));

this.products.splice(index, 1, p);

});

}

}

deleteProduct(id: number) {

this.dataSource.deleteProduct(id).subscribe(() => {

let index = this.products.findIndex(p => this.locator(p, id));

if (index > -1) {

this.products.splice(index, 1);

}

});

}

}

**24.2 🡪 form.component.ts**

constructor(private model: Model, activeRoute: ActivatedRoute) {

this.editing = activeRoute.snapshot.params["mode"] == "edit";

let id = activeRoute.snapshot.params["id"];

if (id != null) {

model.getProductObservable(id).subscribe(p => {

Object.assign(this.product, p || new Product());

**this.product.name = activeRoute.snapshot.params["name"] ?? this.product.name;**

**this.product.category = activeRoute.snapshot.params["category"] ?? this.product.category;**

**let price = activeRoute.snapshot.params["price"];**

**if (price != null) {**

**this.product.price == Number.parseFloat(price);**

**}**

this.productForm.patchValue(this.product);

});

}

}

**24.3 🡪 message.component.ts**

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

import { Message } from '../message.model';

import { MessageService } from '../message.service';

import { CommonModule } from '@angular/common';

import { NavigationCancel, NavigationEnd, Router } from '@angular/router';

@Component({

selector: 'app-message',

imports: [CommonModule],

standalone: true,

providers: [],

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

styleUrl: './message.component.css'

})

export class MessageComponent {

lastMessage?: Message;

constructor(messageService: MessageService, router:Router) {

messageService.messages.subscribe(msg => this.lastMessage = msg);

router.events

.subscribe(e => {

if (e instanceof NavigationEnd || e instanceof NavigationCancel){

this.lastMessage = undefined;

}

})

}

}

**24.4 🡪 table.component.html**

<table class="table table-sm table-bordered table-striped">

<tr>

<th>ID</th><th>Name</th><th>Category</th><th>Price</th><th></th>

</tr>

<tr \*ngFor="let item of getProducts()">

<td style="vertical-align:middle">**{{**item.id**}}**</td>

<td style="vertical-align:middle">**{{**item.name**}}**</td>

<td style="vertical-align:middle">**{{**item.category**}}**</td>

<td style="vertical-align:middle">

**{{**item.price | currency:"USD" **}}**

</td>

<td class="text-center">

<button class="btn btn-danger btn-sm" (click)="deleteProduct(item.id)">

Delete

</button>

**<button class="btn btn-warning btn-sm"**

**[routerLink]="['/form', 'edit',item.id]">**

**Edit**

**</button>**

</td>

</tr>

</table>

**<button class="btn btn-primary" routerLink="/form/create">**

**Create New Product**

**</button>**

<button class="btn btn-danger" (click)="deleteProduct(-1)">

Generate HTTP Error

</button>

**24.5 🡪 table.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { CommonModule } from '@angular/common';

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

@Component({

selector: 'app-table',

imports: [CommonModule, RouterModule],

standalone: true,

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

styleUrl: './table.component.css',

providers: []

})

export class TableComponent {

constructor(private model: Model) { }

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts();

}

deleteProduct(key?: number) {

if (key != undefined) {

this.model.deleteProduct(key);

}

}

}

**25. ROUTING II**

**25.1 🡪 repository.model.ts**

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

import { Product } from "./product.model";

import { StaticDataSource } from "./static.datasource";

import { Observable, ReplaySubject } from "rxjs";

import { RestDataSource } from "./rest.datasource";

@Injectable()

export class Model {

private products: Product[];

private locator = (p: Product, id?: number) => p.id == id;

private replaySubject: ReplaySubject<Product[]>;

constructor(private dataSource: RestDataSource) {

this.products = new Array<Product>();

this.replaySubject = new ReplaySubject<Product[]>(1);

this.dataSource.getData().subscribe(data => {

this.products = data

this.replaySubject.next(data);

this.replaySubject.complete();

});

}

getProducts(): Product[] {

return this.products;

}

getProduct(id: number): Product | undefined {

return this.products.find(p => this.locator(p, id));

}

getProductObservable(id: number): Observable<Product | undefined> {

let subject = new ReplaySubject<Product | undefined>(1);

this.replaySubject.subscribe(products => {

subject.next(products.find(p => this.locator(p, id)));

subject.complete();

});

return subject;

}

**getNextProductId(id?: number): Observable<number> {**

**let subject = new ReplaySubject<number>(1);**

**this.replaySubject.subscribe(products => {**

**let nextId = 0;**

**let index = products.findIndex(p => this.locator(p, id));**

**if (index > -1) {**

**nextId = products[products.length > index + 1**

**? index + 1 : 0].id ?? 0;**

**} else {**

**nextId = id || 0;**

**}**

**subject.next(nextId);**

**subject.complete();**

**});**

**return subject;**

**}**

**getPreviousProductid(id?: number): Observable<number> {**

**let subject = new ReplaySubject<number>(1);**

**this.replaySubject.subscribe(products => {**

**let nextId = 0;**

**let index = products.findIndex(p => this.locator(p, id));**

**if (index > -1) {**

**nextId = products[index > 0**

**? index - 1 : products.length - 1].id ?? 0;**

**} else {**

**nextId = id || 0;**

**}**

**subject.next(nextId);**

**subject.complete();**

**});**

**return subject;**

**}**

saveProduct(product: Product) {

if (product.id == 0 || product.id == null) {

this.dataSource.saveProduct(product)

.subscribe(p => this.products.push(p));

} else {

this.dataSource.updateProduct(product).subscribe(p => {

let index = this.products

.findIndex(item => this.locator(item, p.id));

this.products.splice(index, 1, p);

});

}

}

deleteProduct(id: number) {

this.dataSource.deleteProduct(id).subscribe(() => {

let index = this.products.findIndex(p => this.locator(p, id));

if (index > -1) {

this.products.splice(index, 1);

}

});

}

}

**25.2 🡪 product-count.component.ts**

import { ChangeDetectorRef, Component, KeyValueDiffer, KeyValueDiffers } from '@angular/core';

import { Model } from '../../model/repository.model';

@Component({

selector: 'app-product-count',

imports: [],

template: `<div class="bg-info text-white p-2">There are

{{count}} products

</div>`,

styleUrl: './product-count.component.css'

})

export class ProductCountComponent {

private differ?: KeyValueDiffer<any, any>;

count: number = 0;

constructor(private model: Model,

private keyValueDiffers: KeyValueDiffers,

private changeDetector: ChangeDetectorRef) { }

ngOnInit() {

this.differ = this.keyValueDiffers

.find(this.model.getProducts())

.create();

}

ngDoCheck() {

if (this.differ?.diff(this.model.getProducts()) != null) {

this.updateCount();

}

}

private updateCount() {

this.count = this.model.getProducts().length;

}

}

**25.3 🡪 category-count.component.ts**

import { ChangeDetectorRef, Component, KeyValueDiffer, KeyValueDiffers } from '@angular/core';

import { Model } from '../../model/repository.model';

@Component({

selector: 'app-category-count',

imports: [],

template: `<div class="bg-primary p-2 text-white">

There are {{count}} categories

</div>`,

styleUrl: './category-count.component.css'

})

export class CategoryCountComponent {

private differ?: KeyValueDiffer<any, any>;

count: number = 0;

constructor(private model: Model,

private keyValueDiffers: KeyValueDiffers,

private changeDetector: ChangeDetectorRef) { }

ngOnInit() {

this.differ = this.keyValueDiffers

.find(this.model.getProducts())

.create();

}

ngDoCheck() {

if (this.differ?.diff(this.model.getProducts()) != null) {

this.count = this.model.getProducts()

.map(p => p.category)

.filter((category, index, array) => array.indexOf(category) == index)

.length;

}

}

}

**25.4 🡪 not-found.component.ts**

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

@Component({

selector: 'app-not-found',

imports: [],

template: `<h3 class="bg-danger text-white p-2">Sorry, something went wrong</h3>

<button class="btn btn-primary" routerLink="/">Start Over</button>`,

styleUrl: './not-found.component.css'

})

export class NotFoundComponent {

}

**25.5 🡪 table.component.ts**

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

import { Product } from '../../model/product.model';

import { Model } from '../../model/repository.model';

import { CommonModule } from '@angular/common';

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

@Component({

selector: 'app-table',

imports: [CommonModule, RouterModule],

standalone: true,

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

styleUrl: './table.component.css',

providers: []

})

export class TableComponent {

category: string | null = null;

constructor(public model: Model, activeRoute: ActivatedRoute) {

activeRoute.params.subscribe(params => {

this.category = params["category"] || null;

})

}

getProduct(key: number): Product | undefined {

return this.model.getProduct(key);

}

getProducts(): Product[] {

return this.model.getProducts()

.filter(p => this.category == null || p.category == this.category);

}

get categories(): string[] {

return (this.model.getProducts()

.map(p => p.category)

.filter((c, index, array) => c != undefined && array.indexOf(c) == index)) as string[];

}

deleteProduct(key?: number) {

if (key != undefined) {

this.model.deleteProduct(key);

}

}

}

**25.6 🡪 table.component.html**

**<div class="container-fluid">**

**<div class="row">**

**<div class="col-auto">**

**<div class="d-grid gap-2">**

**<button class="btn btn-secondary"**

**routerLink="/" routerLinkActive="bg-primary">**

**All**

**</button>**

**<button \*ngFor="let category of categories"**

**class="btn btn-secondary"**

**[routerLink]="['/table', category]"**

**routerLinkActive="bg-primary">**

**{{category}}**

**</button>**

**</div>**

**</div>**

**<div class="col">**

<table class="table table-sm table-bordered table-striped">

<thead>

<tr>

<th>ID</th>

<th>Name</th>

<th>Category</th>

<th>Price</th>

<th>Details</th>

<th></th>

</tr>

</thead>

<tbody>

<tr \*ngFor="let item of getProducts()">

<td>**{{**item.id**}}**</td>

<td>**{{**item.name**}}**</td>

<td>**{{**item.category**}}**</td>

<td>**{{**item.price | currency:"USD" **}}**</td>

<td>

<ng-container \*ngIf="item.details else empty">

**{{** item.details?.supplier **}}**, **{{** item.details?.keywords**}}**

</ng-container>

<ng-template #empty>(None)</ng-template>

</td>

<td class="text-center">

<button class="btn btn-danger btn-sm m-1"

(click)="deleteProduct(item.id)">

Delete

</button>

<button class="btn btn-warning btn-sm"

[routerLink]="['/form', 'edit', item.id]">

Edit

</button>

</td>

</tr>

</tbody>

</table>

**</div>**

**</div>**

**</div>**

**<div class="gap-2 text-center">**

<button class="btn btn-primary m-1" routerLink="/form/create">

Create New Product

</button>

<button class="btn btn-danger m-1" (click)="deleteProduct(-1)">

Generate HTTP Error

</button>

<button class="btn btn-danger m-1" routerLink="/does/not/exist">

Generate Routing Error

</button>

**</div>**

**25.7 🡪 productCount.component.ts**

import {

Component, KeyValueDiffer, KeyValueDiffers, ChangeDetectorRef

} from "@angular/core";

import { Model } from "../model/repository.model";

**import { ActivatedRoute } from "@angular/router";**

@Component({

selector: "paProductCount",

template: `<div class="bg-info text-white p-2">There are

{{count}} products

</div>`

})

export class ProductCountComponent {

private differ?: KeyValueDiffer<any, any>;

count: number = 0;

**private category?: string;**

constructor(private model: Model,

private keyValueDiffers: KeyValueDiffers,

private changeDetector: ChangeDetectorRef,

**activeRoute: ActivatedRoute) {**

**activeRoute.pathFromRoot.forEach(route => route.params.subscribe(params => {**

**if (params["category"] != null) {**

**this.category = params["category"];**

**this.updateCount();**

**}**

**}))**

}

ngOnInit() {

this.differ = this.keyValueDiffers

.find(this.model.getProducts())

.create();

}

ngDoCheck() {

if (this.differ?.diff(this.model.getProducts()) != null) {

this.updateCount();

}

}

private updateCount() {

**this.count = this.model.getProducts()**

**.filter(p => this.category == null || p.category == this.category)**

**.length;**

}

}

**2. PROYECTO ANGULAR**

**2.1 🡪 app.component.ts**

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

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

import { TodoItem } from './todoItem';

import { TodoList } from './todoList';

@Component({

selector: 'app-root',

imports: [RouterOutlet],

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

styleUrl: './app.component.css'

})

export class AppComponent {

private list = new TodoList("Bob", [

new TodoItem("Go for run", true),

new TodoItem("Get flowers"),

new TodoItem("Collect tickets"),

]);

get username(): string {

return this.list.user;

}

get itemCount(): number {

return this.list.items.filter(item => !item.complete).length;

}

}

**2.2 🡪 app.component.html**

<mat-toolbar color="primary" class="mat-elevation-z3">

<span class="spacer flexible-spacer"></span>

<span>**{{** username **}}**'s To Do List</span>

<span class="spacer flexible-spacer"></span>

<mat-icon matBadge="**{{** itemCount **}}**" matBadgeColor="accent">checklist</mat-icon>

</mat-toolbar>

**<div class="tableContainer">**

**<table mat-table [dataSource]="items" class="mat-elevation-z3 fullWidth">**

**<ng-container matColumnDef="id">**

**<th mat-header-cell \*matHeaderCellDef>#</th>**

**<td mat-cell \*matCellDef="let i = index"> {{ i + 1 }} </td>**

**</ng-container>**

**<ng-container matColumnDef="task">**

**<th mat-header-cell \*matHeaderCellDef>Task</th>**

**<td mat-cell \*matCellDef="let item"> {{ item.task }} </td>**

**</ng-container>**

**<ng-container matColumnDef="done">**

**<th mat-header-cell \*matHeaderCellDef>Done</th>**

**<td mat-cell \*matCellDef="let item">**

**{{ item.complete }}**

**</td>**

**</ng-container>**

**<tr mat-header-row \*matHeaderRowDef="['id', 'task', 'done']"></tr>**

**<tr mat-row \*matRowDef="let row; columns: ['id', 'task', 'done'];"></tr>**

**</table>**

**</div>**

**5. REAL APPLICATION: SPORTSTORE**

**5.1 🡪 data.json**

{

"products": [

{

"id": 1,

"name": "Kayak",

"category": "Watersports",

"description": "A boat for one person",

"price": 275

},

{

"id": 2,

"name": "Lifejacket",

"category": "Watersports",

"description": "Protective and fashionable",

"price": 48.95

},

{

"id": 3,

"name": "Soccer Ball",

"category": "Soccer",

"description": "FIFA-approved size and weight",

"price": 19.50

},

{

"id": 4,

"name": "Corner Flags",

"category": "Soccer",

"description": "Give your playing field a professional touch",

"price": 34.95

},

{

"id": 5,

"name": "Stadium",

"category": "Soccer",

"description": "Flat-packed 35,000-seat stadium",

"price": 79500

},

{

"id": 6,

"name": "Thinking Cap",

"category": "Chess",

"description": "Improve brain efficiency by 75%",

"price": 16

},

{

"id": 7,

"name": "Unsteady Chair",

"category": "Chess",

"description": "Secretly give your opponent a disadvantage",

"price": 29.95

},

{

"id": 8,

"name": "Human Chess Board",

"category": "Chess",

"description": "A fun game for the family",

"price": 75

},

{

"id": 9,

"name": "Bling King",

"category": "Chess",

"description": "Gold-plated, diamond-studded King",

"price": 1200

}

],

"orders": []

}

**5.2 🡪 *authMiddleware.js***

const jwt = require("jsonwebtoken");

const APP\_SECRET = "myappsecret";

const USERNAME = "admin";

const PASSWORD = "secret";

const mappings = {

get: ["/api/orders", "/orders"],

post: ["/api/products", "/products", "/api/categories", "/categories"]

}

function requiresAuth(method, url) {

return (mappings[method.toLowerCase()] || [])

.find(p => url.startsWith(p)) !== undefined;

}

module.exports = function (req, res, next) {

if (req.url.endsWith("/login") && req.method == "POST") {

if (req.body && req.body.name == USERNAME && req.body.password == PASSWORD) {

let token = jwt.sign({ data: USERNAME, expiresIn: "1h" }, APP\_SECRET);

res.json({ success: true, token: token });

} else {

res.json({ success: false });

}

res.end();

return;

} else if (requiresAuth(req.method, req.url)) {

let token = req.headers["authorization"] || "";

if (token.startsWith("Bearer<")) {

token = token.substring(7, token.length - 1);

try {

jwt.verify(token, APP\_SECRET);

next();

return;

} catch (err) { }

}

res.statusCode = 401;

res.end();

return;

}

next();

}

**5.3 🡪 *static.datasource.ts***

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

import { Product } from "./product.model";

import { Observable, from } from "rxjs";

@Injectable()

export class StaticDataSource {

private products: Product[] = [

new Product(1, "Product 1", "Category 1", "Product 1 (Category 1)", 100),

new Product(2, "Product 2", "Category 1", "Product 2 (Category 1)", 100),

new Product(3, "Product 3", "Category 1", "Product 3 (Category 1)", 100),

new Product(4, "Product 4", "Category 1", "Product 4 (Category 1)", 100),

new Product(5, "Product 5", "Category 1", "Product 5 (Category 1)", 100),

new Product(6, "Product 6", "Category 2", "Product 6 (Category 2)", 100),

new Product(7, "Product 7", "Category 2", "Product 7 (Category 2)", 100),

new Product(8, "Product 8", "Category 2", "Product 8 (Category 2)", 100),

new Product(9, "Product 9", "Category 2", "Product 9 (Category 2)", 100),

new Product(10, "Product 10", "Category 2", "Product 10 (Category 2)", 100),

new Product(11, "Product 11", "Category 3", "Product 11 (Category 3)", 100),

new Product(12, "Product 12", "Category 3", "Product 12 (Category 3)", 100),

new Product(13, "Product 13", "Category 3", "Product 13 (Category 3)", 100),

new Product(14, "Product 14", "Category 3", "Product 14 (Category 3)", 100),

new Product(15, "Product 15", "Category 3", "Product 15 (Category 3)", 100),

];

getProducts(): Observable<Product[]> {

return from([this.products]);

}

}

**5.4 🡪 *product.repository.ts***

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

import { Product } from "./product.model";

import { StaticDataSource } from "./static.datasource";

@Injectable({

providedIn: 'root', // Esto hace que esté disponible en toda la aplicación

})

export class ProductRepository {

private products: Product[] = [];

private categories: string[] = [];

constructor(private dataSource: StaticDataSource) {

dataSource.getProducts().subscribe(data => {

this.products = data;

this.categories = data.map(p => p.category ?? "(None)")

.filter((c, index, array) => array.indexOf(c) == index).sort();

});

}

getProducts(category?: string): Product[] {

return this.products

.filter(p => category == undefined || category == p.category);

}

getProduct(id: number): Product | undefined {

return this.products.find(p => p.id == id);

}

getCategories(): string[] {

return this.categories;

}

}

**5.5 🡪 *store.component.ts***

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

import { ProductRepository } from '../model/product.repository';

import { Product } from '../model/product.model';

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

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-store',

standalone:true,

imports: [CommonModule, FormsModule],

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

styleUrl: './store.component.css'

})

export class StoreComponent {

constructor(private repository: ProductRepository) { }

get products(): Product[] {

return this.repository.getProducts();

}

get categories(): string[] {

return this.repository.getCategories();

}

}

**5.6 🡪 *store.component.html***

<div class="container-fluid">

<div class="row">

<div class="bg-dark text-center text-white p-2">

<span class="navbar-brand ml-2">SPORTS STORE</span>

</div>

</div>

<div class="row text-white">

<div class="col-3 bg-info p-2">

**{{**categories.length**}}** Categories

</div>

<div class="col-9 bg-success p-2">

**{{**products.length**}}** Products

</div>

</div>

</div>

**5.7 🡪 *counter.directive.ts***

import {

Directive, ViewContainerRef, TemplateRef, Input, SimpleChanges

} from "@angular/core";

@Directive({

selector: "[counterOf]"

})

export class CounterDirective {

constructor(private container: ViewContainerRef,

private template: TemplateRef<Object>) {

}

@Input("counterOf")

counter: number = 0;

ngOnChanges(changes: SimpleChanges) {

this.container.clear();

for (let i = 0; i < this.counter; i++) {

this.container.createEmbeddedView(this.template,

new CounterDirectiveContext(i + 1));

}

}

}

class CounterDirectiveContext {

constructor(public $implicit: any) { }

}

**6. REAL APPLICATION: SPORTSTORE ORDERS AND CHECKOUT**

**6.1 🡪 *cart.model.ts***

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

import { Product } from "./product.model";

@Injectable({

providedIn: 'root', // Esto hace que esté disponible en toda la aplicación

})

export class Cart {

public lines: CartLine[] = [];

public itemCount: number = 0;

public cartPrice: number = 0;

addLine(product: Product, quantity: number = 1) {

let line = this.lines.find(line => line.product.id == product.id);

if (line != undefined) {

line.quantity += quantity;

} else {

this.lines.push(new CartLine(product, quantity));

}

this.recalculate();

}

updateQuantity(product: Product, quantity: number) {

let line = this.lines.find(line => line.product.id == product.id);

if (line != undefined) {

line.quantity = Number(quantity);

}

this.recalculate();

}

removeLine(id: number) {

let index = this.lines.findIndex(line => line.product.id == id);

this.lines.splice(index, 1);

this.recalculate();

}

clear() {

this.lines = [];

this.itemCount = 0;

this.cartPrice = 0;

}

private recalculate() {

this.itemCount = 0;

this.cartPrice = 0;

this.lines.forEach(l => {

this.itemCount += l.quantity;

this.cartPrice += l.lineTotal;

})

}

}

export class CartLine {

constructor(public product: Product,

public quantity: number) {}

get lineTotal() {

return this.quantity \* (this.product.price ?? 0);

}

}

**6.2 🡪 *cart-summary.component.ts***

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

import { Cart } from '../../model/cart.model';

import { CommonModule } from '@angular/common';

@Component({

selector: 'app-cart-summary',

standalone:true,

imports: [CommonModule],

templateUrl: './cart-summary.component.html',

styleUrl: './cart-summary.component.css'

})

export class CartSummaryComponent {

constructor(public cart: Cart) { }

}

**6.3 🡪 *cart-summary.component.html***

<div class="float-end">

<small class="fs-6">

Your cart:

<span \*ngIf="cart.itemCount > 0">

**{{** cart.itemCount **}}** item(s)

**{{** cart.cartPrice | currency:"USD":"symbol":"2.2-2" **}}**

</span>

<span \*ngIf="cart.itemCount == 0">

(empty)

</span>

</small>

<button class="btn btn-sm bg-dark text-white"

[disabled]="cart.itemCount == 0" routerLink="/cart">

<i class="fa fa-shopping-cart"></i>

</button>

</div>

**6.4 🡪 *storeFirst.guard.ts***

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

import {

ActivatedRouteSnapshot, RouterStateSnapshot,

Router

} from "@angular/router";

import { StoreComponent } from "./store/store.component";

@Injectable({

providedIn: 'root', // Esto hace que esté disponible en toda la aplicación

})

export class StoreFirstGuard {

private firstNavigation = true;

constructor(private router: Router) { }

canActivate(route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): boolean {

if (this.firstNavigation) {

this.firstNavigation = false;

if (route.component != StoreComponent) {

this.router.navigateByUrl("/");

return false;

}

}

return true;

}

}

**6.5 🡪 *cart-detail.component.html***

<div class="container-fluid">

<div class="row">

<div class="bg-dark text-white p-2">

<span class="navbar-brand ml-2">SPORTS STORE</span>

</div>

</div>

<div class="row">

<div class="col mt-2">

<h2 class="text-center">Your Cart</h2>

<table class="table table-bordered table-striped p-2">

<thead>

<tr>

<th>Quantity</th>

<th>Product</th>

<th class="text-end">Price</th>

<th class="text-end">Subtotal</th>

</tr>

</thead>

<tbody>

<tr \*ngIf="cart.lines.length == 0">

<td colspan="4" class="text-center">

Your cart is empty

</td>

</tr>

<tr \*ngFor="let line of cart.lines">

<td>

<input type="number" class="form-control-sm" style="width:5em" [value]="line.quantity"

(change)="cart.updateQuantity(line.product,

$any($event).target.value)" />

</td>

<td>**{{**line.product.name**}}**</td>

<td class="text-end">

**{{**line.product.price | currency:"USD":"symbol":"2.2-2"**}}**

</td>

<td class="text-end">

**{{**(line.lineTotal) | currency:"USD":"symbol":"2.2-2" **}}**

</td>

<td class="text-center">

<button class="btn btn-sm btn-danger" (click)="cart.removeLine(line.product.id ?? 0)">

Remove

</button>

</td>

</tr>

</tbody>

<tfoot>

<tr>

<td colspan="3" class="text-end">Total:</td>

<td class="text-end">

**{{**cart.cartPrice | currency:"USD":"symbol":"2.2-2"**}}**

</td>

</tr>

</tfoot>

</table>

</div>

</div>

<div class="row">

<div class="col">

<div class="text-center">

<button class="btn btn-primary m-1" routerLink="/store">

Continue Shopping

</button>

<button class="btn btn-secondary m-1" routerLink="/checkout" [disabled]="cart.lines.length == 0">

Checkout

</button>

</div>

</div>

</div>

</div>

**6.6 🡪 *order.model.ts***

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

import { Cart } from "./cart.model";

@Injectable({

providedIn: 'root', // Esto hace que esté disponible en toda la aplicación

})

export class Order {

public id?: number;

public name?: string;

public address?: string;

public city?: string;

public state?: string;

public zip?: string;

public country?: string;

public shipped: boolean = false;

constructor(public cart: Cart) { }

clear() {

this.id = undefined;

this.name = this.address = this.city = undefined;

this.state = this.zip = this.country = undefined;

this.shipped = false;

this.cart.clear();

}

}

**6.7 🡪 *order.repository.ts***

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

import { Observable } from "rxjs";

import { Order } from "./order.model";

import { StaticDataSource } from "./static.datasource";

@Injectable({

providedIn: 'root', // Esto lo hace disponible globalmente

})

export class OrderRepository {

private orders: Order[] = [];

constructor(private dataSource: StaticDataSource) {}

getOrders(): Order[] {

return this.orders;

}

saveOrder(order: Order): Observable<Order> {

return this.dataSource.saveOrder(order);

}

}

**6.8 🡪 *check-out.component.html***

<div class="container-fluid">

<div class="row">

<div class="bg-dark text-white p-2">

<span class="navbar-brand ml-2">SPORTS STORE</span>

</div>

</div>

</div>

<div \*ngIf="orderSent" class="m-2 text-center">

<h2>Thanks!</h2>

<p>Thanks for placing your order.</p>

<p>We'll ship your goods as soon as possible.</p>

<button class="btn btn-primary" routerLink="/store">Return to Store</button>

</div>

<form \*ngIf="!orderSent" #form="ngForm" novalidate (ngSubmit)="submitOrder(form)" class="m-2">

<div class="form-group">

<label>Name</label>

<input class="form-control" #name="ngModel" name="name" [(ngModel)]="order.name" required />

<span \*ngIf="submitted && name.invalid" class="text-danger">

Please enter your name

</span>

</div>

<div class="form-group">

<label>Address</label>

<input class="form-control" #address="ngModel" name="address" [(ngModel)]="order.address" required />

<span \*ngIf="submitted && address.invalid" class="text-danger">

Please enter your address

</span>

</div>

<div class="form-group">

<label>City</label>

<input class="form-control" #city="ngModel" name="city" [(ngModel)]="order.city" required />

<span \*ngIf="submitted && city.invalid" class="text-danger">

Please enter your city

</span>

</div>

<div class="form-group">

<label>State</label>

<input class="form-control" #state="ngModel" name="state" [(ngModel)]="order.state" required />

<span \*ngIf="submitted && state.invalid" class="text-danger">

Please enter your state

</span>

</div>

<div class="form-group">

<label>Zip/Postal Code</label>

<input class="form-control" #zip="ngModel" name="zip" [(ngModel)]="order.zip" required />

<span \*ngIf="submitted && zip.invalid" class="text-danger">

Please enter your zip/postal code

</span>

</div>

<div class="form-group">

<label>Country</label>

<input class="form-control" #country="ngModel" name="country" [(ngModel)]="order.country" required />

<span \*ngIf="submitted && country.invalid" class="text-danger">

Please enter your country

</span>

</div>

<div class="text-center">

<button class="btn btn-secondary m-1" routerLink="/cart">Back</button>

<button class="btn btn-primary m-1" type="submit">Complete Order</button>

</div>

</form>

**6.9 🡪 *rest.datasource.ts***

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

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

import { Observable } from "rxjs";

import { Product } from "./product.model";

import { Order } from "./order.model";

const PROTOCOL = "http";

const PORT = 3500;

@Injectable({

providedIn: 'root', // Esto lo hace disponible globalmente

})

export class RestDataSource {

baseUrl: string;

constructor(private http: HttpClient) {

this.baseUrl = `${PROTOCOL}://${location.hostname}:${PORT}/`;

}

getProducts(): Observable<Product[]> {

return this.http.get<Product[]>(this.baseUrl + "products");

}

saveOrder(order: Order): Observable<Order> {

return this.http.post<Order>(this.baseUrl + "orders", order);

}

}

**7. REAL APPLICATION: SPORTSTORE ADMINISTRATION**

**7.1 🡪 *auth.component.ts***

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

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

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

@Component({

templateUrl: "auth.component.html"

})

export class AuthComponent {

username?: string;

password?: string;

errorMessage?: string;

constructor(private router: Router) { }

authenticate(form: NgForm) {

if (form.valid) {

//perform authentication

this.router.navigateByUrl("/admin/main");

} else {

this.errorMessage = "Form Data Invalid";

}

}

}

**7.2 🡪 *auth.component.html***

<div class="bg-info p-2 text-center text-white">

<h3>SportsStore Admin</h3>

</div>

<div class="bg-danger mt-2 p-2 text-center text-white" \*ngIf="errorMessage != null">

**{{**errorMessage**}}**

</div>

<div class="p-2">

<form novalidate #form="ngForm" (ngSubmit)="authenticate(form)">

<div class="form-group">

<label>Name</label>

<input class="form-control" name="username"

[(ngModel)]="username" required />

</div>

<div class="form-group">

<label>Password</label>

<input class="form-control" type="password" name="password"

[(ngModel)]="password" required />

</div>

<div class="text-center p-2">

<button class="btn btn-secondary m-1" routerLink="/">Go back</button>

<button class="btn btn-primary m-1" type="submit">Log In</button>

</div>

</form>

</div>

**7.3 🡪 *admin.module.ts***

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

import { CommonModule } from "@angular/common";

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

import { RouterModule } from "@angular/router";

import { AuthComponent } from "./auth.component";

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

let routing = RouterModule.forChild([

{ path: "auth", component: AuthComponent },

{ path: "main", component: AdminComponent },

{ path: "\*\*", redirectTo: "auth" }

]);

@NgModule({

imports: [CommonModule, FormsModule, routing],

declarations: [AuthComponent, AdminComponent],

})

export class AdminModule { }

**7.4 🡪 *auth.service.ts***

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

import { Observable } from "rxjs";

import { RestDataSource } from "./rest.datasource";

@Injectable()

export class AuthService {

constructor(private datasource: RestDataSource) {}

authenticate(username: string, password: string): Observable<boolean> {

return this.datasource.authenticate(username, password);

}

get authenticated(): boolean {

return this.datasource.auth\_token != null;

}

clear() {

this.datasource.auth\_token = undefined;

}

}

**7.5 🡪 *auth.component.ts***

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

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

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

import { AuthService } from "../model/auth.service";

@Component({

templateUrl: "auth.component.html"

})

export class AuthComponent {

username?: string;

password?: string;

errorMessage?: string;

constructor(private router: Router,

private auth: AuthService) { }

authenticate(form: NgForm) {

if (form.valid) {

this.auth.authenticate(this.username ?? "", this.password ?? "")

.subscribe(response => {

if (response) {

this.router.navigateByUrl("/admin/main");

}

this.errorMessage = "Authentication Failed";

})

} else {

this.errorMessage = "Form Data Invalid";

}

}

}

**7.6 🡪 *auth.guard.ts***

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

import { ActivatedRouteSnapshot, RouterStateSnapshot,

Router } from "@angular/router";

import { AuthService } from "../model/auth.service";

@Injectable()

export class AuthGuard {

constructor(private router: Router,

private auth: AuthService) { }

canActivate(route: ActivatedRouteSnapshot,

state: RouterStateSnapshot): boolean {

if (!this.auth.authenticated) {

this.router.navigateByUrl("/admin/auth");

return false;

}

return true;

}

}

**7.7 🡪 *material.module.ts***

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

const features: any[] = [];

@NgModule({

imports: [features],

exports: [features]

})

export class MaterialFeatures {}

**7.8 🡪 *productTable.component.ts***

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

@Component({

template: `

<h3 style="padding-top: 10px">

Product Table Placeholder

</h3>

`

})

export class ProductTableComponent {}

***7.9 🡪 productEditor.component.ts***

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

@Component({

template: `

<h3 style="padding-top: 10px">

Product Table Placeholder

</h3>

`

})

export class ProductEditorComponent { }

***7.10 🡪 orderTable.component.ts***

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

@Component({

template: `

<h3 style="padding-top: 10px">

Order Table Placeholder

</h3>

`

})

export class OrderTableComponent {}

***7.11 🡪 admin.component.html***

**<mat-toolbar color="primary">**

**<button mat-icon-button \*ngIf="sidenav.mode === 'over'"**

**(click)="sidenav.toggle()">**

**<mat-icon \*ngIf="!sidenav.opened">menu</mat-icon>**

**<mat-icon \*ngIf="sidenav.opened">close</mat-icon>**

**</button>**

**<span></span>**

**SportsStore Administration**

**<span></span>**

**</mat-toolbar>**

**<mat-sidenav-container>**

**<mat-sidenav #sidenav="matSidenav" class="mat-elevation-z8">**

**<button mat-button class="menu-button"**

**routerLink="/admin/main/products"**

**routerLinkActive="mat-accent"**

**(click)="sidenav.close()">**

**<mat-icon>shopping\_cart</mat-icon>**

**<span>Products</span>**

**</button>**

**<button mat-button class="menu-button"**

**routerLink="/admin/main/orders"**

**routerLinkActive="mat-accent"**

**(click)="sidenav.close()">**

**<mat-icon>local\_shipping</mat-icon>**

**<span>Orders</span>**

**</button>**

**<mat-divider></mat-divider>**

**<button mat-button class="menu-button logout" (click)="logout()">**

**<mat-icon>logout</mat-icon>**

**<span>Logout</span>**

**</button>**

**</mat-sidenav>**

**<mat-sidenav-content>**

**<div class="content">**

**<router-outlet></router-outlet>**

**</div>**

**</mat-sidenav-content>**

**</mat-sidenav-container>**

***7.12 🡪 styles.css***

html, body { height: 100%; }

body { margin: 0; font-family: Roboto, "Helvetica Neue", sans-serif; }

mat-toolbar span {

flex: 1 1 auto;

}

.menu-button {

width: 100%;

font-size: 1rem;

}

.menu-button .mat-icon {

margin-right: 10px;

}

.menu-button span {

flex: 1 1 auto;

}

mat-sidenav {

margin: 16px;

width: 175px;

border-right: none;

border-radius: 4px;

padding: 4px;

}

mat-sidenav .mat-divider {

margin-top: 20px;

margin-bottom: 5px;

}

mat-sidenav-container {

height: calc(100vh - 60px);

}

mat-sidenav .mat-button-wrapper {

display: flex;

width: 100%;

justify-content: baseline;

align-content: center;

}

mat-sidenav .mat-button-wrapper mat-icon {

margin-top: 5px;

}

mat-sidenav .mat-button-wrapper span {

text-align: start;

}