npx -y @angular/cli new frontend --routing --style=scss --skip-git

cd frontend

1. Genera SharedModule y componentes header/footer:

npx ng g module modules/shared

npx ng g component modules/shared/components/header --skip-tests --inline-style

npx ng g component modules/shared/components/footer --skip-tests --inline-style

1. Declara y exporta header/footer en SharedModule:  
   Abre src/app/modules/shared/shared.module.ts y asegúrate de que contenga:

typescript

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

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

**import** { HeaderComponent } **from** './components/header/header.component';

**import** { FooterComponent } **from** './components/footer/footer.component';

@NgModule({

declarations: [HeaderComponent, FooterComponent],

imports: [CommonModule],

exports: [HeaderComponent, FooterComponent]

})

**export** **class** SharedModule {}

1. Importa SharedModule en AppModule:  
   Abre src/app/app.module.ts y añade:

typescript

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

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

**import** { AppRoutingModule } **from** './app-routing.module';

**import** { AppComponent } **from** './app.component';

**import** { SharedModule } **from** './modules/shared/shared.module';

@NgModule({

declarations: [AppComponent],

imports: [BrowserModule, AppRoutingModule, SharedModule],

providers: [],

bootstrap: [AppComponent]

})

**export** **class** AppModule {}

1. Añade header y footer al layout raíz:  
   Pon esto en src/app/app.component.html:

xml

<app-header></app-header>

<router-outlet></router-outlet>

<app-footer></app-footer>

1. Arranca el proyecto y disfruta de cero errores:

bash

ng serve -o

Sigue estos pasos para crear el componente “logo” y mostrar el SVG que tienes en assets:

**1. Genera el componente en el sitio correcto**

Ejecuta en la raíz de frontend:

bash

npx ng generate component modules/shared/components/logo --skip-tests --inline-style

**2. Usa la imagen SVG en el template**

Pon en src/app/modules/shared/components/logo/logo.component.html (borra el contenido anterior):

xml

<img src="assets/images/Ibercaja-logo-2025.svg" alt="Ibercaja Logo" style="max-width:120px; height:auto;">

* La ruta assets/images/Ibercaja-logo-2025.svg funciona porque cualquier archivo/imagen puesto en src/assets/ es servido por Angular CLI en /assets/....

**3. Exporta LogoComponent en SharedModule**

En src/app/modules/shared/shared.module.ts añade el import y el export:

typescript

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

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

**import** { HeaderComponent } **from** './components/header/header.component';

**import** { FooterComponent } **from** './components/footer/footer.component';

**import** { LogoComponent } **from** './components/logo/logo.component'; *// importa aquí*

@NgModule({

declarations: [HeaderComponent, FooterComponent, LogoComponent],

imports: [CommonModule],

exports: [HeaderComponent, FooterComponent, LogoComponent] *// expórtalo aquí*

})

**export** **class** SharedModule {}

**4. Úsalo donde quieras**

Por ejemplo, en tu header:

xml

*<!-- En src/app/modules/shared/components/header/header.component.html -->*

<app-logo></app-logo>

**5. Reinicia/guarda y abre la app**

Tu logo SVG ahora se mostrará y estará listo para recibir más contenido dinámico más adelante.

Aquí tienes el componente “logo” modificado para aceptar la ruta del SVG como un parámetro @Input(), aunque sigue mostrando un valor por defecto si no se le pasa:

**1. LogoComponent (logo.component.ts)**

typescript

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

@Component({

selector: 'app-logo',

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

styleUrls: []

})

**export** **class** LogoComponent {

@Input() src = 'assets/images/Ibercaja-logo-2025.svg';

@Input() alt = 'Ibercaja Logo';

@Input() width: string | number = 120; *// puedes cambiar el valor por defecto*

}

**2. Template (logo.component.html)**

xml

<img [src]="src" [alt]="alt" [style.max-width.px]="width" style="height:auto;">

**3. Uso flexible**

* Por defecto (no le pasas nada):

xml

<app-logo></app-logo>

* Con ruta y descripción personalizada:

xml

<app-logo src="assets/images/otro-logo.svg" alt="Otro Logo" [width]="80"></app-logo>

Para crear el componente “copyright” que reciba el texto por @Input, sigue estos pasos:

**1. Generar el componente**

bash

npx ng generate component modules/shared/components/copyright --skip-tests --inline-style

**2. copyright.component.ts**

typescript

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

@Component({

selector: 'app-copyright',

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

styleUrls: []

})

**export** **class** CopyrightComponent {

@Input() text: string = '(c) Alejandro Abadía';

}

**3. copyright.component.html**

xml

<footer style="font-size:0.9em; color:#888; text-align:center; padding:8px 0;">

{{ text }}

</footer>

**4. Exporta el componente en SharedModule**

Abre src/app/modules/shared/shared.module.ts y añade CopyrightComponent:

typescript

**import** { CopyrightComponent } **from** './components/copyright/copyright.component';

@NgModule({

declarations: [

HeaderComponent,

FooterComponent,

LogoComponent,

CopyrightComponent

],

imports: [CommonModule],

exports: [

HeaderComponent,

FooterComponent,

LogoComponent,

CopyrightComponent

]

})

**export** **class** SharedModule {}

**5. Usar el componente**

Por defecto:

xml

<app-copyright></app-copyright>

Con texto personalizado:

xml

<app-copyright text="(c) Tu Empresa, 2025"></app-copyright>

Con esto puedes mostrar el texto por defecto o pasar el que quieras por @Input en cualquier parte del layout o del footer.

Aquí tienes el plan actualizado para crear tu botón reutilizable adaptado al nuevo sistema moderno, BEM y con funcionalidades maximizadas desde el original.

**1. Comando de consola**

Desde la raíz del proyecto:

bash

npx ng generate component modules/shared/components/ui-button --skip-tests --inline-style=false --inline-template=false

Esto te creará los archivos:

* ui-button.component.ts
* ui-button.component.html
* ui-button.component.scss
* ui-button.component.spec.ts

**2. ui-button.component.ts**

typescript

**import** { Component, Input, Output, EventEmitter, OnInit, OnChanges, SimpleChanges } **from** '@angular/core';

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

**import** { DomSanitizer, SafeHtml } **from** '@angular/platform-browser';

@Component({

selector: 'app-ui-button',

templateUrl: './ui-button.component.html',

styleUrls: ['./ui-button.component.scss']

})

**export** **class** UiButtonComponent **implements** OnInit, OnChanges {

@Input() type: 'button' | 'submit' = 'button';

@Input() disabled = false;

@Input() active = false;

@Input() fullWidth = false;

@Input() color: 'primary' | 'secondary' | 'accent' | 'danger' | 'surface' | 'success' | 'warning' | 'info' | 'icon' = 'primary';

*// Iconografía*

@Input() icon?: string;

@Input() iconPosition: 'left' | 'right' | 'top' | 'only' = 'left';

@Input() iconSize: 'xs' | 's' | 'm' | 'l' | 'xl' | number = 'm';

*// Texto del botón*

@Input() textSize: 'xs' | 's' | 'm' | 'l' | 'xl' | number = 'm';

*// Accesibilidad*

@Input() ariaLabel?: string;

*// Loading state*

@Input() loading: boolean = false;

@Output() onClick = **new** EventEmitter<MouseEvent>();

svgContent: SafeHtml | **null** = **null**;

**private** **readonly** iconSizes = { xs: 16, s: 20, m: 24, l: 32, xl: 40 };

**private** **readonly** textSizes = { xs: '0.75rem', s: '0.875rem', m: '1rem', l: '1.25rem', xl: '1.5rem' };

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer) {}

ngOnInit() {

**this**.loadSvg();

}

ngOnChanges(changes: SimpleChanges) {

**if** (changes['icon'] && !changes['icon'].isFirstChange()) {

**this**.loadSvg();

}

}

**private** loadSvg() {

**if** (**this**.icon) {

**this**.http.get(`assets/icons/${**this**.icon}.svg`, { responseType: 'text' }).subscribe({

next: svg => **this**.svgContent = **this**.sanitizer.bypassSecurityTrustHtml(svg),

error: () => **this**.svgContent = **null**

});

} **else** {

**this**.svgContent = **null**;

}

}

getIconStyle() {

**const** size = **typeof** **this**.iconSize === 'number'

? `${**this**.iconSize}px`

: `${**this**.iconSizes[**this**.iconSize] ?? **this**.iconSizes.m}px`;

**return** {

width: size,

height: size,

minWidth: size,

minHeight: size,

verticalAlign: **this**.iconPosition === 'top' ? 'middle' : 'text-bottom'

};

}

getTextStyle() {

**const** size = **typeof** **this**.textSize === 'number'

? `${**this**.textSize}px`

: **this**.textSizes[**this**.textSize] ?? **this**.textSizes.m;

**return** { fontSize: size };

}

getButtonClasses(): string {

**return** [

'ui-button',

`ui-button--${**this**.color}`,

**this**.fullWidth ? 'ui-button--full-width' : '',

**this**.active ? 'ui-button--active' : '',

**this**.icon ? 'ui-button--has-icon' : '',

**this**.icon ? `ui-button--icon-${**this**.iconPosition}` : '',

**this**.iconPosition === 'only' ? 'ui-button--icon-only' : ''

].filter(Boolean).join(' ');

}

handleClick(event: MouseEvent) {

**if** (!**this**.disabled && !**this**.loading) {

**this**.onClick.emit(event);

}

}

}

**3. ui-button.component.html**

xml

<button

[attr.type]="type"

[class]="getButtonClasses()"

[disabled]="disabled || loading"

[attr.aria-label]="ariaLabel"

(click)="handleClick($event)"

>

<span \*ngIf="loading" class="ui-button\_\_spinner" aria-hidden="true"></span>

<ng-container \*ngIf="icon && (iconPosition === 'left' || iconPosition === 'top' || iconPosition === 'only')">

<span

class="ui-button\_\_icon"

[ngStyle]="getIconStyle()"

[innerHTML]="svgContent"

\*ngIf="svgContent"

aria-hidden="true"

></span>

</ng-container>

<span

class="ui-button\_\_text"

[ngStyle]="getTextStyle()"

\*ngIf="iconPosition !== 'only' && !loading"

>

<ng-content></ng-content>

</span>

<ng-container \*ngIf="icon && iconPosition === 'right'">

<span

class="ui-button\_\_icon"

[ngStyle]="getIconStyle()"

[innerHTML]="svgContent"

\*ngIf="svgContent"

aria-hidden="true"

></span>

</ng-container>

</button>

**4. ui-button.component.scss**

text

@import '~src/styles/variables';

@import '~src/styles/mixins';

.ui-button {

display: inline-flex;

align-items: center;

justify-content: center;

min-width: 2rem;

min-height: 2rem;

font-weight: 500;

line-height: 1.2;

border-radius: var(--radius-md);

transition: all var(--transition-base);

cursor: pointer;

outline: none;

border: none;

position: relative;

padding: var(--space-xs) var(--space-m);

background: var(--color-primary);

color: var(--color-primary-contrast);

&--primary { background: var(--color-primary); color: var(--color-primary-contrast);}

&--secondary { background: var(--color-secondary); color: var(--color-primary-contrast);}

&--accent { background: var(--color-accent); color: var(--color-primary-contrast);}

&--danger { background: var(--color-error); color: var(--color-primary-contrast);}

&--surface { background: var(--color-surface); color: var(--color-text);}

&--success { background: var(--color-success); color: var(--color-primary-contrast);}

&--warning { background: var(--color-warning); color: var(--color-primary-contrast);}

&--info { background: var(--color-info); color: var(--color-primary-contrast);}

&--icon { background: none; color: var(--color-primary); }

&--active { box-shadow: 0 0 0 2px var(--color-secondary-aa); }

&--full-width { width: 100%; }

&--has-icon { }

&--icon-left .ui-button\_\_icon { margin-right: var(--space-xs);}

&--icon-right .ui-button\_\_icon { margin-left: var(--space-xs);}

&--icon-top .ui-button\_\_icon { margin-bottom: var(--space-xs);}

&--icon-only { padding: var(--space-xs); }

&[disabled] {

opacity: 0.6;

cursor: not-allowed;

pointer-events: none;

}

.ui-button\_\_icon {

display: inline-flex;

vertical-align: middle;

transition: opacity 0.2s;

}

.ui-button\_\_text {

display: inline;

align-items: center;

vertical-align: middle;

font-weight: inherit;

}

.ui-button\_\_spinner {

width: 1em; height: 1em;

border: 2px solid var(--color-primary-contrast);

border-top-color: var(--color-primary);

border-radius: 50%;

display: inline-block;

animation: spin 0.65s linear infinite;

vertical-align: middle;

margin-right: var(--space-xs);

}

}

@keyframes spin {

100% { transform: rotate(360deg); }

}

* Todo BEM, variables y mixins modernos.
* Incluye color, variantes, iconografía SVG por parámetro, loading y accesibilidad.
* Listo para Angular y para crecer con tus tokens/scss globales.

Aquí tienes el nuevo componente adaptado para el menú de administración, con estructura moderna, reutilizando tus estilos y lógica BEM, y preparado para expandirse con submenús o distintos iconos.

**1. Comando consola Angular**

bash

npx ng generate component modules/shared/components/admin-menu --skip-tests --inline-style=false --inline-template=false

**2. admin-menu.component.ts**

typescript

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

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

**export** **interface** AdminMenuItem {

label: string;

icon?: string;

route?: any[] | string;

active?: boolean;

disabled?: boolean;

external?: boolean;

submenu?: AdminMenuItem[];

}

@Component({

selector: 'app-admin-menu',

templateUrl: './admin-menu.component.html',

styleUrls: ['./admin-menu.component.scss']

})

**export** **class** AdminMenuComponent {

@Input() items: AdminMenuItem[] = [

{ label: 'Dashboard', icon: 'dashboard', route: '/admin/dashboard' },

{ label: 'Países', icon: 'flag', route: '/admin/countries' },

{ label: 'Áreas', icon: 'map', route: '/admin/areas' },

{ label: 'Idiomas', icon: 'language', route: '/admin/languages' },

{ label: 'Usuarios', icon: 'user', route: '/admin/users' }

];

@Input() footer?: string;

constructor(**public** router: Router) {}

isActive(item: AdminMenuItem): boolean {

**return** item.active ?? (item.route && **this**.router.url.startsWith(**typeof** item.route === 'string' ? item.route : item.route[0]));

}

}

**3. admin-menu.component.html**

xml

<nav class="admin-menu" aria-label="Menú administración">

<ul class="admin-menu\_\_list" role="menubar">

<li

\*ngFor="let item of items"

class="admin-menu\_\_item"

[class.admin-menu\_\_item--active]="isActive(item)"

[class.admin-menu\_\_item--disabled]="item.disabled"

role="none"

>

<ng-container \*ngIf="!item.external; else external">

<a

\*ngIf="item.route"

class="admin-menu\_\_link"

[routerLink]="item.route"

[attr.aria-current]="isActive(item) ? 'page' : null"

[attr.tabindex]="item.disabled ? -1 : 0"

[class.admin-menu\_\_link--active]="isActive(item)"

[class.admin-menu\_\_link--disabled]="item.disabled"

[routerLinkActive]="'admin-menu\_\_link--active'"

[routerLinkActiveOptions]="{ exact: true }"

>

<span \*ngIf="item.icon" class="admin-menu\_\_icon" aria-hidden="true">

<app-ui-button

[icon]="item.icon"

color="icon"

size="sm"

variant="icon"

[disabled]="item.disabled"

aria-hidden="true"

></app-ui-button>

</span>

<span class="admin-menu\_\_label">{{ item.label }}</span>

</a>

</ng-container>

<ng-template #external>

<a

[href]="item.route"

class="admin-menu\_\_link"

target="\_blank"

rel="noopener"

[tabindex]="item.disabled ? -1 : 0"

[class.admin-menu\_\_link--disabled]="item.disabled"

>

<span \*ngIf="item.icon" class="admin-menu\_\_icon" aria-hidden="true">

<app-ui-button

[icon]="item.icon"

color="icon"

size="sm"

variant="icon"

[disabled]="item.disabled"

aria-hidden="true"

></app-ui-button>

</span>

<span class="admin-menu\_\_label">{{ item.label }}</span>

</a>

</ng-template>

</li>

</ul>

<div class="admin-menu\_\_footer" \*ngIf="footer">{{ footer }}</div>

</nav>

**4. admin-menu.component.scss**

text

@import '~src/styles/variables';

@import '~src/styles/mixins';

.admin-menu {

background: var(--color-background);

border-bottom: 1px solid var(--color-surface);

width: 100%;

min-height: 54px;

display: flex;

align-items: flex-end;

flex-direction: column;

&\_\_list {

display: flex;

flex-direction: row;

width: 100%;

list-style: none;

padding: 0;

margin: 0;

}

&\_\_item {

flex: 1 1 0;

display: flex;

justify-content: center;

align-items: flex-end;

padding: 0;

margin: 0;

background: transparent;

border: none;

&--active .admin-menu\_\_link,

&--active .admin-menu\_\_label,

.admin-menu\_\_link--active {

font-weight: 900;

color: var(--color-primary-dark, #073864);

background: var(--color-background, #fff);

}

&--disabled .admin-menu\_\_link,

.admin-menu\_\_link--disabled {

opacity: 0.5;

pointer-events: none;

cursor: not-allowed;

}

}

&\_\_link {

width: 100%;

min-width: 0;

border: none;

outline: none;

background: none;

border-radius: .4em .4em 0 0;

height: 48px;

padding: 0 2em;

font-size: 1.07rem;

font-family: var(--font-main), sans-serif;

color: var(--color-primary, #253889);

font-weight: 600;

letter-spacing: .01em;

box-shadow: none;

position: relative;

display: flex;

align-items: center;

justify-content: center;

transition: background 0.12s, color 0.16s, box-shadow 0.12s;

white-space: nowrap;

&::after {

content: '';

position: absolute;

left: 20%;

right: 20%;

bottom: 0;

height: 3px;

background: var(--color-primary-aa, #0b7ad0);

border-radius: 2px 2px 0 0;

opacity: 0;

transition: all 0.18s;

}

&.admin-menu\_\_link--active::after {

left: 8%;

right: 8%;

opacity: 1;

background: var(--color-primary-dark, #073864);

}

&:hover,

&:focus-visible {

background: var(--color-primary-h8, #eef6fc);

color: var(--color-primary-dark, #073864);

}

&:active {

background: var(--color-background, #fff);

color: var(--color-primary-dark, #073864);

}

&:disabled {

opacity: 0.5;

cursor: not-allowed;

}

}

&\_\_icon {

margin-right: var(--space-xs);

display: inline-flex;

align-items: center;

justify-content: center;

}

&\_\_label {

display: block;

font-size: inherit;

font-family: inherit;

line-height: 1.1;

font-weight: 600;

}

&\_\_footer {

width: 100%;

margin-left: auto;

padding: 1rem;

font-size: 14px;

text-align: center;

color: var(--color-accent);

}

}

@media (max-width: 700px) {

.admin-menu\_\_link { height: 39px; font-size: .97rem; padding: 0 1em; }

}

El menú soporta:

* Navegación con rutas internas y externas.
* Icono SVG vía app-ui-button.
* BEM puro.
* Adaptable a cambios de items vía @Input.
* Footer opcional y todos los estilos normalizados, sin acoplamiento con el antiguo.

Puedes añadir submenús fácilmente extendiendo la estructura de AdminMenuItem. Si necesitas esa funcionalidad, dímelo y te muestro cómo anidarla.

Tus errores y cómo resolverlos:

**1. Error al importar variables/mixins SCSS**

No uses la tilde ~ en Angular 13+ (Webpack 5) al importar archivos SCSS. Usa rutas relativas desde el archivo .scss:

Cambia en admin-menu.component.scss:

text

// Antes (ERRÓNEO)

@import '~src/styles/variables';

@import '~src/styles/mixins';

// Después (CORRECTO, si estás en src/app/modules/shared/components/admin-menu/)

@import '../../../../../styles/variables';

@import '../../../../../styles/mixins';

(Si tus styles están en src/styles/ y tu component está 5 niveles por debajo, ajusta los ../ a tu estructura.)

**2. Errores [routerLink], [routerLinkActive], [routerLinkActiveOptions]**

Solución: Importa RouterModule en el módulo donde declaras AdminMenuComponent.  
Por ejemplo, si lo declaras dentro de SharedModule, añade:

typescript

**import** { RouterModule } **from** '@angular/router';

@NgModule({

imports: [

*// otros imports...*

RouterModule

],

*// ...*

})

**export** **class** SharedModule {}

**3. Errores de tipo TypeScript**

Para asegurar tipado correcto y evitar "boolean | undefined", reemplaza:

typescript

[disabled]="item.disabled"

por

typescript

[disabled]="!!item.disabled"

(repite para todos los usos similares)

Y dentro de TS:

typescript

isActive(item: AdminMenuItem): boolean {

**return** !!(item.active ?? (item.route && **this**.router.url.startsWith(**typeof** item.route === 'string' ? item.route : item.route[0])));

}

**Resumen de cambios**

* Todas las rutas de SCSS: rutas relativas, sin ~.
* Importa RouterModule en el módulo donde está el componente.
* Añade !! en [disabled] y en isActive para forzar tipo boolean.

Con estos cambios, tu componente de menú será funcional, escalable y 100% compatible con Angular moderno y tu sistema de estilos actual. ¿Quieres el código concreto del módulo actualizado?

Para crear una zona justo debajo del header pensada para poner menús de selección (como el menú de administración más adelante), lo ideal es usar un componente contenedor tipo "toolbar" o "menubar".

**1. Crea el componente contenedor para los menús**

Comando:

bash

npx ng generate component modules/shared/components/menu-bar --skip-tests --inline-style=false --inline-template=false

Esto generará la carpeta y archivos necesarios. En este componente podrás insertar el menú de administración o cualquier otro menú.

**2. Contenido de los archivos**

**menu-bar.component.ts**

typescript

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

@Component({

selector: 'app-menu-bar',

templateUrl: './menu-bar.component.html',

styleUrls: ['./menu-bar.component.scss']

})

**export** **class** MenuBarComponent { }

**menu-bar.component.html**

Deja un ng-content para máxima flexibilidad (podrá contener menús, filtros, tabs...):

xml

<nav class="menu-bar" aria-label="Zona de menús de navegación">

<ng-content></ng-content>

</nav>

**menu-bar.component.scss**

text

@import 'styles/variables';

@import 'styles/mixins';

.menu-bar {

width: 100%;

min-height: 48px;

background: var(--color-surface, #f7fafd);

border-bottom: 1px solid var(--color-border, #e3eefa);

display: flex;

align-items: center;

padding: 0 var(--space-m);

box-shadow: 0 2px 4px rgba(37,56,137,0.02);

z-index: 5;

position: relative;

}

**3. Cómo usar el componente**

En tu template principal (app.component.html o en el layout principal de cada módulo):

xml

<app-header></app-header>

<app-menu-bar>

*<!-- Aquí van tus menús, por ejemplo: -->*

*<!-- <app-admin-menu \*ngIf="esSeccionAdmin"></app-admin-menu> -->*

</app-menu-bar>

<router-outlet></router-outlet>

<app-footer></app-footer>

* El componente app-menu-bar sirve de zona para cualquier menú o barra de selección.
* En el futuro, simplemente añade <app-admin-menu> en esa área cuando estés en la parte de administración, o cualquier otro menú contextual.

¿Quieres que te incluya lógica para solo mostrar app-admin-menu en rutas /admin usando Angular? Pídelo si lo necesitas.Aquí tienes una solución limpia y escalable para tu zona de menús:

**1. Comando Angular**

bash

npx ng generate component modules/shared/components/menu-bar --skip-tests --inline-style=false --inline-template=false

**2. menu-bar.component.ts**

typescript

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

@Component({

selector: 'app-menu-bar',

templateUrl: './menu-bar.component.html',

styleUrls: ['./menu-bar.component.scss']

})

**export** **class** MenuBarComponent { }

**3. menu-bar.component.html**

xml

<nav class="menu-bar" aria-label="Selector de menús">

<ng-content></ng-content>

</nav>

**4. menu-bar.component.scss**

text

@import 'styles/variables';

@import 'styles/mixins';

.menu-bar {

width: 100%;

min-height: 48px;

background: var(--color-surface, #f7fafd);

border-bottom: 1px solid var(--color-border, #e3eefa);

display: flex;

align-items: center;

padding: 0 var(--space-m);

box-shadow: 0 1px 4px rgba(20,40,80,0.04);

z-index: 10;

position: relative;

}

**5. Uso en tu layout principal (por ejemplo en app.component.html):**

xml

<app-header></app-header>

<app-menu-bar>

*<!-- Aquí van tus menús, por ejemplo el de administración -->*

*<!-- <app-admin-menu \*ngIf="esSeccionAdmin"></app-admin-menu> -->*

</app-menu-bar>

<router-outlet></router-outlet>

<app-footer></app-footer>

Ahora puedes renderizar cualquier menú en esa zona simplemente insertándolo como hijo de <app-menu-bar>. Es ideal para switches entre menú público, administración, filtros, tabs, etc. y mantiene tu estructura independiente y BEM.

El error “is not a known element” para <app-menu-bar> significa que el componente existe, pero Angular no lo reconoce porque no está declarado y exportado correctamente en los módulos.

**Para solucionarlo:**

1. Asegúrate de que MenuBarComponent está en tu SharedModule:

En src/app/modules/shared/shared.module.ts:

typescript

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

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

**import** { MenuBarComponent } **from** './components/menu-bar/menu-bar.component';

*// ...otros imports...*

@NgModule({

declarations: [

*// ...otros componentes...*

MenuBarComponent

],

imports: [

CommonModule

*// ...otros módulos...*

],

exports: [

*// ...otros componentes...*

MenuBarComponent

]

})

**export** **class** SharedModule { }

1. Asegúrate de que SharedModule se importa en tu AppModule:

En src/app/app.module.ts:

typescript

**import** { SharedModule } **from** './modules/shared/shared.module';

@NgModule({

imports: [

*// ...otros módulos...*

SharedModule

],

*// ...*

})

**export** **class** AppModule { }

Con esto, <app-menu-bar> será reconocido en cualquier template perteneciente a un módulo que importe SharedModule (como app.component.html).

Para mostrar únicamente los dos botones (“Público” y “Administración”) en la zona pública usando tu componente admin-menu, utiliza el input items de admin-menu y pásale el array de botones correspondientes.

**1. En tu layout público (por ejemplo en app.component.html o donde uses <app-menu-bar>):**

xml

<app-header></app-header>

<app-menu-bar>

<app-admin-menu

[items]="[

{ label: 'Público', icon: 'public', route: '/public' },

{ label: 'Administración', icon: 'lock', route: '/admin' }

]">

</app-admin-menu>

</app-menu-bar>

<router-outlet></router-outlet>

<app-footer></app-footer>

* Cambia "public" y "lock" por los nombres reales de los SVG que tengas en assets/icons/.

**2. ¿Dónde definir los ítems de menú?**

Si necesitas más lógica, puedes declararlos como variable de clase en, por ejemplo, app.component.ts:

typescript

**public** mainMenu = [

{ label: 'Público', icon: 'public', route: '/public' },

{ label: 'Administración', icon: 'lock', route: '/admin' }

];

Y luego:

xml

<app-menu-bar>

<app-admin-menu [items]="mainMenu"></app-admin-menu>

</app-menu-bar>

Pero puedes poner el array inline si es estático, como antes.

El componente admin-menu generará los botones navegables reutilizando los mismos estilos y lógica BEM. Así tienes la zona pública lista y, en cualquier momento, puedes inyectar el mismo componente en la zona de administración con todos los ítems CRUD, simplemente cambiando el array.Esta es la forma recomendada para tener sólo los botones ("Público" y "Administración") usando tu componente admin-menu en el menú-bar público:

**app.component.html (o el layout público que uses)**

xml

<app-header></app-header>

<app-menu-bar>

<app-admin-menu

[items]="[

{ label: 'Público', icon: 'public', route: '/public' },

{ label: 'Administración', icon: 'lock', route: '/admin' }

]"

></app-admin-menu>

</app-menu-bar>

<router-outlet></router-outlet>

<app-footer></app-footer>

* Cambia "public" y "lock" por el nombre del icono SVG que tengas en assets/icons/ si lo deseas, o elimina el campo "icon" si no quieres icono.
* No necesitas más lógica: con ese array, el menú te muestra los dos botones y navega correctamente.

De ese modo, tu mismo componente de menú es 100% reutilizable para la parte pública y la parte de administración. Solamente le cambias la lista de items por @Input.

**Solución: Importa HttpClientModule en tu AppModule**

Debes importar el módulo de HTTP de Angular en tu app.module.ts:

typescript

**import** { HttpClientModule } **from** '@angular/common/http';

@NgModule({

imports: [

*// otros módulos...*

HttpClientModule *// <-- Añade esto aquí, preferentemente el primero o segundo*

],

*// ...*

})

**export** **class** AppModule { }

* Guarda los cambios y vuelve a ejecutar tu aplicación (puede requerir parar y volver a lanzar ng serve).
* Solo necesitas importar HttpClientModule una vez, normalmente solo en AppModule.
* Si tienes CoreModule, ponlo allí, pero en la gran mayoría de estructuras basta con AppModule.

Tu idea y la estructura son correctas desde el punto de vista de patrones Angular:

* Es estándar ubicar la página de bienvenida pública en src/app/modules/public/ y usar allí los componentes y rutas propias del módulo público.

**1. Comando para crear componente landing-page**

Desde la raíz del proyecto ejecuta:

bash

npx ng generate component modules/public/pages/landing-page --skip-tests --inline-style=false --inline-template=false

**2. Añade la ruta en el public-routing.module.ts**

Abre (o crea si no existe) src/app/modules/public/public-routing.module.ts y añade la ruta:

typescript

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

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

**import** { LandingPageComponent } **from** './pages/landing-page/landing-page.component';

**const** routes: Routes = [

{ path: '', component: LandingPageComponent },

*// ... otras rutas públicas futuras*

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

**export** **class** PublicRoutingModule {}

(Y asegúrate de importar este módulo en el módulo PublicModule.)

**3. landing-page.component.html**

Pega solo el título, más adelante lo irás ampliando:

xml

<h1>Normativas FATCA/CRS</h1>

Puedes añadir estilos mínimos en el SCSS generado si lo prefieres.

**4. Haz que el botón “Público” apunte a esta landing page**

En tu menú usa:

typescript

{ label: 'Público', icon: 'icon-user', route: '/public' }

o según como esté tu routing principal.

De este modo la landing pública es modular, canónica (Angular best-practice), y puede crecer de forma ordenada en tu código.  
Si luego quieres estructura más SEO o i18n, se puede modularizar aún más, pero esto es totalmente estándar.

El error y tu último mensaje confirman la causa:  
El módulo PublicModule no existe, por eso la import dinámica falla y Angular no puede cargar ni encontrar el código ni las rutas.

**Solución: Crea el archivo public.module.ts**

Crea este archivo en la ruta:

text

frontend/src/app/modules/public/public.module.ts

Y pon dentro este contenido mínimo canónico:

typescript

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

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

**import** { PublicRoutingModule } **from** './public-routing.module';

**import** { LandingPageComponent } **from** './pages/landing-page/landing-page.component';

@NgModule({

declarations: [

LandingPageComponent

*// ...aquí otros componentes públicos si tienes más*

],

imports: [

CommonModule,

PublicRoutingModule

]

})

**export** **class** PublicModule { }

* Si tienes más componentes públicos, inclúyelos en el array declarations.
* Si tienes un SharedModule propio, agrégalo en imports también.

**Resumen y comprobaciones**

1. Ubicación correcta:  
   frontend/src/app/modules/public/public.module.ts
2. Declaraciones:  
   El componente (como LandingPageComponent) debe estar en declarations.
3. Routing:  
   Tu PublicRoutingModule queda importado, y es el que establecerá las rutas hijas.
4. Lazy-loading:  
   Tu Angular detectará ahora el módulo para el path /public en producción y desarrollo.

Después de crear el archivo y guardar, reinicia el servidor (ng serve).  
El error “Module not found” desaparecerá y la navegación funcionará correctamente.Te falta crear el archivo public.module.ts en la carpeta frontend/src/app/modules/public/, por eso Angular no puede encontrar ni importar el módulo.

Copia este bloque y pégalo en un archivo llamado public.module.ts en esa ruta:

typescript

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

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

**import** { PublicRoutingModule } **from** './public-routing.module';

**import** { LandingPageComponent } **from** './pages/landing-page/landing-page.component';

@NgModule({

declarations: [

LandingPageComponent

*// otros componentes públicos cuando los crees*

],

imports: [

CommonModule,

PublicRoutingModule

]

})

**export** **class** PublicModule { }

* Si tienes otros componentes públicos, añádelos en declarations.
* Si usas SharedModule, añádelo a imports.

Luego simplemente vuelve a lanzar tu servidor (mejor para el proceso y ejecuta de nuevo ng serve), y todo funcionará.  
¡Con esto ya tendrás lazy-loading funcional y tu página pública activada!

**Solución paso a paso**

**1. Deja tu LandingPageComponent**solo**en el módulo al que pertenece (PublicModule):**

* En src/app/modules/public/public.module.ts  
  (Esto está bien, déjalo así):

typescript

declarations: [

LandingPageComponent

*// otros componentes públicos cuando los crees*

],

* En src/app/app.module.ts  
  Elimina esto si tienes algo así:

typescript

declarations: [

AppComponent,

LandingPageComponent, *// <---- BORRAR ESTA LINEA*

*// ...*

],

Debe quedar:

typescript

declarations: [

AppComponent

*// ...otros componentes raíz, pero NO LandingPageComponent*

],

* No debes importar LandingPageComponent manualmente en ningún sitio más. Angular lo encontrará por el módulo.

**2. Guarda los cambios, para y vuelve a lanzar el servidor**

Haz Ctrl+C en la terminal y ejecuta de nuevo ng serve.

**3. Estado final**

* LandingPageComponent está solo en las declaraciones de PublicModule.
* Ninguna otra declaración (ni en AppModule, ni en otro lugar).

Con esto desaparece el error NG6007 y tu ruta /public funcionará perfectamente. Si tienes más dudas sobre otros módulos o componentes avísalo, pero esto arregla tu problema de routing y compilación.

**Checklist Resumido (¡todo bien si ya ves la landing!):**

* LandingPageComponent solo está declarado en PublicModule, no en AppModule.
* Todo el código de la zona pública está bajo src/app/modules/public/.
* El routing principal usa loadChildren para /public.
* Los estilos, iconos y todos los assets cargan correctamente.
* El menú lleva correctamente a la página pública y la navegación es fluida.

Cualquier ampliación sobre nuevas páginas públicas, login, guards, internacionalización, diseño o lo que desees añadir al landing, ya parte de una base sólida y estándar.

La forma más eficiente y canónica en Angular es reutilizar el mismo componente admin-menu como menú secundario también en la parte pública, pero con items y estilos distintos según contexto. No hace falta otro componente si sólo cambia el contenido y el menú es genérico.

**Qué debes hacer**

**1. Crea el nuevo componente de página pública "Countries"**

Ubicación:

text

src/app/modules/public/pages/countries/countries.component.ts

Comando CLI recomendado:

bash

npx ng generate component modules/public/pages/countries --skip-tests --inline-style=false --inline-template=false

En el HTML pon:

xml

<h1>Información por país</h1>

**2. Añade la ruta en tu public-routing.module.ts:**

typescript

**const** routes: Routes = [

{ path: '', component: LandingPageComponent },

{ path: 'countries', component: CountriesComponent }

];

**3. Añade el menú secundario solo en la parte pública**

En el template que hace de layout de la zona pública (public.component.html o directamente en landing-page.component.html si solo tienes una), añade:

xml

<app-admin-menu

[items]="[

{ label: 'Información general', icon: 'icon-info', route: '/public' },

{ label: 'Información por países', icon: 'icon-world-globe', route: '/public/countries' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

* Los iconos son opcionales (icon-info y icon-world-globe deben estar en tu assets si quieres iconos visuales).

Para que sólo salga ahí, simplemente pon este bloque encima del <router-outlet> en tu plantilla raíz de zona pública (no en el global).

**4. Si el menú se repite en varias páginas públicas, crea un componente "layout" público donde lo incluyas y usas <router-outlet>, así es más escalable.**

Resumen:

* Reutiliza el componente actual admin-menu.
* Crea la nueva página y la nueva ruta pública.
* Añade el bloque <app-admin-menu> *solo* en las páginas públicas, encima del <router-outlet> (o en un layout público si tienes).

Así tendrás dos menús:

* El global principal arriba (como ahora).
* El secundario, sólo en el área pública, con las dos opciones nuevas y funcionando perfectamente.

¿Necesitas plantilla ejemplo del nuevo public.component.html completo, o de la estructura de archivos final? Avísame.Lo más recomendable y DRY (Don’t Repeat Yourself) es reutilizar el mismo componente admin-menu para este segundo menú en la vista pública, cambiando los items por el nuevo conjunto de opciones. Esto es canónico Angular siempre que el menú sea parametrizable y esté bien desacoplado (como es tu caso).

**Pasos para implementar lo que deseas**

**1. Crear la página "countries" pública**

* Haz:

bash

ng generate component modules/public/pages/countries --skip-tests

* Abre countries.component.html y pon:

xml

<h1>Información por país</h1>

**2. Añadir la ruta en public-routing.module.ts**

typescript

**const** routes: Routes = [

{ path: '', component: LandingPageComponent },

{ path: 'countries', component: CountriesComponent }

];

**3. Colocar el menú secundario solo en la parte pública**

En el layout de la zona pública (ejemplo, tu landing-page.component.html o mejor, un public.component.html que actúe como wrapper), pon esto justo debajo del menú global:

xml

<app-admin-menu

[items]="[

{ label: 'Información general', icon: 'icon-info', route: '/public' },

{ label: 'Información por países', icon: 'icon-world-globe', route: '/public/countries' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

(Usa los iconos que prefieras, asegúrate que icon-info y icon-world-globe están en assets/icons/).

**4. ¿Dónde poner este menú?**

* Si tienes un componente layout público (public.component.html): ahí.
* Si solo tienes la landing, ponlo en landing-page.component.html hasta que estructures más páginas públicas.
* Mejor: crea un componente contenedor public.component.ts y pon ahí el menú + <router-outlet>, así sólo aparece en la zona pública.

**Resumen**

* Sí: reutiliza admin-menu para el menú público con items diferentes.
* Añade la nueva ruta, crea el nuevo componente y actualiza el menú con las rutas deseadas.
* Sitúalo en una plantilla pública, no en el global.

Cualquier detalle de implementación o estructura de layout, pídelo y te lo escribo ajustado a tu código real.

Si no tienes un public.component.html (ni el componente), puedes crearlo ahora para actuar como layout contenedor de toda la zona pública. Esto es recomendable para que el submenú se muestre siempre encima del contenido público, sin tener que duplicarlo en cada página.

**Pasos recomendados**

**1. Crea el componente de layout público**

(Ejecuta:)

bash

npx ng generate component modules/public/public --skip-tests

Esto genera:

* public.component.ts
* public.component.html
* public.component.scss

**2. Pon el menú secundario en public.component.html:**

xml

<app-admin-menu

[items]="[

{ label: 'Información general', icon: 'icon-info', route: '/public' },

{ label: 'Información por países', icon: 'icon-world-globe', route: '/public/countries' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

**3. Actualiza tu public-routing.module.ts:**

Haz que todas las páginas públicas sean hijas de este layout:

typescript

**const** routes: Routes = [

{

path: '',

component: PublicComponent,

children: [

{ path: '', component: LandingPageComponent },

{ path: 'countries', component: CountriesComponent }

*// ...futuras rutas aquí*

]

}

];

Esto hace que el menú secundario siempre aparezca, y sólo el contenido dentro de <router-outlet> cambie según la subruta.

**4. Añade CountriesComponent si no lo tienes:**

bash

npx ng generate component modules/public/pages/countries --skip-tests

Y en su HTML pon:

xml

<h1>Información por país</h1>

Así, el menú secundario sólo aparece en la zona pública, es reutilizable/parametrizable, y tu código sigue la arquitectura más estándar y mantenible de Angular modular.

¿Necesitas el código exacto del nuevo public.component.ts o ayuda con algún import? Avísalo.Lo ideal en este caso es crear precisamente ese componente: un public.component que actúe como layout para la zona pública, y poner ahí (en su HTML) el menú secundario justo arriba del <router-outlet>.

Así, tu menú de contexto solo aparece en la parte pública y nunca en la privada/admin, y no tienes que repetir el menú en cada página pública.

**Pasos claros para hacerlo:**

1. Genera el contenedor público:

bash

ng generate component modules/public/public --skip-tests

Esto te creará public.component.ts, public.component.html, etc.

1. Pon esto en public.component.html:

xml

<app-admin-menu

[items]="[

{ label: 'Información general', icon: 'icon-info', route: '/public' },

{ label: 'Información por países', icon: 'icon-world-globe', route: '/public/countries' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

1. Actualiza tu public-routing.module.ts así:

typescript

**const** routes: Routes = [

{

path: '',

component: PublicComponent,

children: [

{ path: '', component: LandingPageComponent },

{ path: 'countries', component: CountriesComponent }

]

}

];

De este modo, todas las rutas públicas cuelgan lógicamente de tu layout público, y el menú secundario aparece siempre arriba.

1. No repitas el menú en los .html de cada página pública, solo en el layout.

¿No tienes aún el componente CountriesComponent? Si no lo tienes todavía, créalo antes con:

bash

ng generate component modules/public/pages/countries --skip-tests

Así tu estructura será canónica, el código DRY y el menú secundario solo aparecerá en el área pública automáticamente. Si necesitas los TS completos de algún archivo, pídelo detalladamente.

Tus errores actuales se deben a lo siguiente:

**1.**PublicComponent y CountriesComponent no están importados ni declarados

Solución:

* Asegúrate de tener los archivos:
  + src/app/modules/public/public/public.component.ts
  + src/app/modules/public/pages/countries/countries.component.ts
* Importa y declara ambos en public.module.ts:

typescript

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

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

**import** { PublicRoutingModule } **from** './public-routing.module';

**import** { PublicComponent } **from** './public/public.component';

**import** { LandingPageComponent } **from** './pages/landing-page/landing-page.component';

**import** { CountriesComponent } **from** './pages/countries/countries.component';

@NgModule({

declarations: [

PublicComponent,

LandingPageComponent,

CountriesComponent

*// ...otros componentes públicos futuros*

],

imports: [

CommonModule,

PublicRoutingModule

]

})

**export** **class** PublicModule { }

**2.**Error "app-admin-menu is not a known element"

Esto pasa porque AdminMenuComponent (tu menú) no está disponible en el contexto del módulo público.

Solución:

* Exporta el componente de menú desde SharedModule (normalmente ya está) y
* Asegúrate de importar el SharedModule en PublicModule:

typescript

**import** { SharedModule } **from** '../shared/shared.module';

@NgModule({

*// ...*

imports: [

CommonModule,

SharedModule, *// <- Añade esto aquí*

PublicRoutingModule

]

})

**3.**En public-routing.module.ts

Tus rutas deben usar los componentes correctos importados:

typescript

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

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

**import** { PublicComponent } **from** './public/public.component';

**import** { LandingPageComponent } **from** './pages/landing-page/landing-page.component';

**import** { CountriesComponent } **from** './pages/countries/countries.component';

**const** routes: Routes = [

{

path: '',

component: PublicComponent,

children: [

{ path: '', component: LandingPageComponent },

{ path: 'countries', component: CountriesComponent }

]

}

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

**export** **class** PublicRoutingModule {}

**Resumen de comprobaciones**

* Ambos componentes están creados/importados/declarados (PublicComponent, CountriesComponent).
* SharedModule está importado en PublicModule.
* Las rutas usan los componentes correctos y están correctamente importados.
* Reinicia el servidor tras los cambios.

La estructura óptima en Angular para la parte de administración es idéntica a lo ya realizado en "public", pero en un nuevo directorio admin dentro de src/app/modules/. Así tendrás la gestión separada, modular y profesional para crecer sin riesgos en el futuro.

**Pasos para crear el módulo "admin" con menú dinámico**

**1. Estructura recomendada de carpetas**

text

src/app/modules/admin/

admin.module.ts

admin-routing.module.ts

admin/

admin.component.ts/html/scss

pages/

dashboard/

countries/

continents/

areas/

dependencies/

languages/

translations/

**2. Comandos para generar los componentes**

bash

*# Crea componente de layout para admin*

ng generate component modules/admin/admin --skip-tests

*# Crea cada página:*

ng generate component modules/admin/pages/dashboard --skip-tests

ng generate component modules/admin/pages/countries --skip-tests

ng generate component modules/admin/pages/continents --skip-tests

ng generate component modules/admin/pages/areas --skip-tests

ng generate component modules/admin/pages/dependencies --skip-tests

ng generate component modules/admin/pages/languages --skip-tests

ng generate component modules/admin/pages/translations --skip-tests

**3. admin-routing.module.ts**

typescript

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

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

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

**import** { DashboardComponent } **from** './pages/dashboard/dashboard.component';

**import** { CountriesComponent } **from** './pages/countries/countries.component';

**import** { ContinentsComponent } **from** './pages/continents/continents.component';

**import** { AreasComponent } **from** './pages/areas/areas.component';

**import** { DependenciesComponent } **from** './pages/dependencies/dependencies.component';

**import** { LanguagesComponent } **from** './pages/languages/languages.component';

**import** { TranslationsComponent } **from** './pages/translations/translations.component';

**const** routes: Routes = [

{

path: '',

component: AdminComponent,

children: [

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

{ path: 'dashboard', component: DashboardComponent },

{ path: 'countries', component: CountriesComponent },

{ path: 'continents', component: ContinentsComponent },

{ path: 'areas', component: AreasComponent },

{ path: 'dependencies', component: DependenciesComponent },

{ path: 'languages', component: LanguagesComponent },

{ path: 'translations', component: TranslationsComponent }

]

}

];

@NgModule({

imports: [RouterModule.forChild(routes)],

exports: [RouterModule]

})

**export** **class** AdminRoutingModule {}

**4. admin.module.ts**

typescript

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

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

**import** { AdminRoutingModule } **from** './admin-routing.module';

**import** { SharedModule } **from** '../shared/shared.module';

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

**import** { DashboardComponent } **from** './pages/dashboard/dashboard.component';

**import** { CountriesComponent } **from** './pages/countries/countries.component';

**import** { ContinentsComponent } **from** './pages/continents/continents.component';

**import** { AreasComponent } **from** './pages/areas/areas.component';

**import** { DependenciesComponent } **from** './pages/dependencies/dependencies.component';

**import** { LanguagesComponent } **from** './pages/languages/languages.component';

**import** { TranslationsComponent } **from** './pages/translations/translations.component';

@NgModule({

declarations: [

AdminComponent,

DashboardComponent,

CountriesComponent,

ContinentsComponent,

AreasComponent,

DependenciesComponent,

LanguagesComponent,

TranslationsComponent

],

imports: [

CommonModule,

SharedModule,

AdminRoutingModule

]

})

**export** **class** AdminModule { }

**5. admin.component.html**

Pon tu menú de botones usando el mismo <app-admin-menu> (o el menú parametrizable) ya existente, cambiando los items:

xml

<app-admin-menu

[items]="[

{ label: 'Dashboard', icon: 'icon-dashboard', route: '/admin/dashboard' },

{ label: 'Países', icon: 'icon-country', route: '/admin/countries' },

{ label: 'Continentes', icon: 'icon-continents', route: '/admin/continents' },

{ label: 'Áreas', icon: 'icon-areas', route: '/admin/areas' },

{ label: 'Dependencias', icon: 'icon-dependencies', route: '/admin/dependencies' },

{ label: 'Lenguajes', icon: 'icon-languages', route: '/admin/languages' },

{ label: 'Traducciones', icon: 'icon-translations', route: '/admin/translations' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

(Si no tienes todos esos iconos aún, usa los existentes y luego los creas en assets.)

**6. Añade módulo "admin" al app-routing.module.ts**

Asegúrate de enlazarlo por lazy loading:

typescript

**const** routes: Routes = [

{ path: 'public', loadChildren: () => **import**('./modules/public/public.module').then(m => m.PublicModule) },

{ path: 'admin', loadChildren: () => **import**('./modules/admin/admin.module').then(m => m.AdminModule) },

*// ...*

];

Con esto, la estructura de administración será idéntica y tan escalable como la pública.  
Cuando accedas a /admin, verás el menú y navegará entre las páginas según el botón seleccionado.

¿Quieres el CSS para el reparto de botones, o algún archivo concreto listo para copiar? Solo dilo.La solución más profesional y mantenible pasa por replicar la estructura modular que tienes en "public", pero en un directorio admin dentro de src/app/modules/. Así, cada área tiene su menú, layout y routing desacoplado y escalable.

**Pasos recomendados (OJO que colisiona countries, sería mejor llamarlos adminDashboardm¡, adminCountries….)**

**1. Crea el módulo y los componentes de administración**

* Desde la raíz del proyecto ejecuta estos comandos:

text

ng generate module modules/admin --route admin --module app-routing

ng generate component modules/admin/admin --skip-tests

ng generate component modules/admin/pages/dashboard --skip-tests

ng generate component modules/admin/pages/countries --skip-tests

ng generate component modules/admin/pages/continents --skip-tests

ng generate component modules/admin/pages/areas --skip-tests

ng generate component modules/admin/pages/dependencies --skip-tests

ng generate component modules/admin/pages/languages --skip-tests

ng generate component modules/admin/pages/translations --skip-tests

**2. Routing: admin-routing.module.ts**

Configura un layout anidado igual que en "public":

typescript

**const** routes: Routes = [

{

path: '',

component: AdminComponent,

children: [

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

{ path: 'dashboard', component: DashboardComponent },

{ path: 'countries', component: CountriesComponent },

{ path: 'continents', component: ContinentsComponent },

{ path: 'areas', component: AreasComponent },

{ path: 'dependencies', component: DependenciesComponent },

{ path: 'languages', component: LanguagesComponent },

{ path: 'translations', component: TranslationsComponent }

]

}

];

**3. Menú en admin.component.html**

Pon el menú como hiciste en public:

xml

<app-admin-menu

[items]="[

{ label: 'Dashboard', route: '/admin/dashboard', icon: 'icon-dashboard' },

{ label: 'Países', route: '/admin/countries', icon: 'icon-country' },

{ label: 'Continentes', route: '/admin/continents', icon: 'icon-continents' },

{ label: 'Áreas', route: '/admin/areas', icon: 'icon-areas' },

{ label: 'Dependencias', route: '/admin/dependencies', icon: 'icon-dependencies' },

{ label: 'Lenguajes', route: '/admin/languages', icon: 'icon-languages' },

{ label: 'Traducciones', route: '/admin/translations', icon: 'icon-translations' }

]"

></app-admin-menu>

<router-outlet></router-outlet>

(Usa los iconos que tengas, el resto puedes crearlos como SVGs luego.)

**4. Por ahora, en cada HTML de página (dashboard.component.html, etc.) pon solo el título:**

xml

<h1>Dashboard</h1> *<!-- O el título correspondiente -->*

**5. Asegúrate de importar SharedModule y cualquier módulo común en tu admin.module.ts.**

Con esto:

* El menú de admin será tan dinámico como el de public.
* Al añadir más botones se repartirá automáticamente el espacio.
* Navegar a /admin abrirá el "dashboard" por defecto, y el resto de rutas según botón.
* El diseño es escalable y DRY (no se repite código ni styling).

Cuando quieras hacer CRUDs de verdad, solo tendrás que ampliar cada página correspondiente. Si necesitas el código exacto de alguno de los archivos (TS, HTML, rutas o menú), indícalo.

El error se debe a que tienes todos los componentes de administración (AdminComponent, DashboardComponent, CountriesComponent, etc.) declarados en dos módulos distintos:

* En AdminModule (correcto)
* Y en AppModule (incorrecto)

**Cómo solucionarlo**

1. Ve a tu archivo src/app/app.module.ts  
   Busca la sección de declaraciones (declarations) y elimina estas líneas:

typescript

AdminComponent,

DashboardComponent,

CountriesComponent,

ContinentsComponent,

AreasComponent,

DependenciesComponent,

LanguagesComponent,

TranslationsComponent,

Tu AppModule debe declarar únicamente el AppComponent y quizás otros componentes básicos de nivel raíz, pero nunca los pertenecientes a módulos hijos/feature (admin, public, etc.).

1. Mantén esas declaraciones solo en admin.module.ts.

**Ejemplo correcto de declarations en AppModule:**

typescript

@NgModule({

declarations: [

AppComponent

*// ...otros componentes globales si los necesitas, pero NO los de Admin.*

],

*// ... imports, etc.*

})

**export** **class** AppModule { }

**Ejemplo correcto de declarations en admin.module.ts:**

typescript

@NgModule({

declarations: [

AdminComponent,

DashboardComponent,

CountriesComponent,

ContinentsComponent,

AreasComponent,

DependenciesComponent,

LanguagesComponent,

TranslationsComponent

],

*// ... imports, etc.*

})

**export** **class** AdminModule { }

**Después de limpiar las declaraciones**

* Guarda los archivos.
* Vuelve a ejecutar ng serve.
* El error desaparecerá y el lazy loading funcionará perfectamente.

Si quieres, pásame el bloque actual de declarations de tu app.module.ts y te indico exactamente cómo dejarlo.Este error ocurre porque tienes los componentes de administración (como AdminComponent, DashboardComponent, CountriesComponent, etc.) declarados tanto en AppModule como en AdminModule, y Angular no permite que un componente esté en dos módulos al mismo tiempo.

**Solución**

* Debes eliminar de declarations de tu AppModule todas las referencias a:
  + AdminComponent
  + DashboardComponent
  + CountriesComponent
  + ContinentsComponent
  + AreasComponent
  + DependenciesComponent
  + LanguagesComponent
  + TranslationsComponent

Solo deben estar declaradas en AdminModule.

Tu array de declarations en AppModule debe quedar así (y nada más de admin):

typescript

declarations: [

AppComponent

*// ... otros componentes globales, pero NO los de admin*

],

En AdminModule, ahí SÍ deben estar:

typescript

declarations: [

AdminComponent,

DashboardComponent,

CountriesComponent,

ContinentsComponent,

AreasComponent,

DependenciesComponent,

LanguagesComponent,

TranslationsComponent

],

Corrige esto, guarda y vuelve a lanzar ng serve y todo funcionará perfecto.  
¡Nunca declares ningún componente de un módulo feature (como admin/public) directamente en tu AppModule! Así evitas estos errores y aprovechas el lazy loading real.Tu error se debe a que has declarado los componentes de administración en dos módulos: tanto en AppModule como en AdminModule. En Angular, cada componente debe declararse solo en un módulo.

**Solución**

* Entra en tu src/app/app.module.ts.
* Busca el array declarations y elimina todo lo relacionado con administración:
  + AdminComponent
  + DashboardComponent
  + CountriesComponent
  + ContinentsComponent
  + AreasComponent
  + DependenciesComponent
  + LanguagesComponent
  + TranslationsComponent

La sección debe quedar así:

typescript

declarations: [

AppComponent *// (otros como el layout global, pero NO los admin)*

],

(NO debe estar ninguno de los componentes de admin.)

Mantén esas declaraciones solamente en tu admin.module.ts y en ningún otro módulo.

Guarda y relanza Angular (ng serve). El error desaparecerá y tu arquitectura será rígida, estable y canónica.  
¿Quieres el ejemplo completo de cómo deben quedar ambos archivos? Avísame.El error se debe a que has declarado los componentes de administración (AdminComponent, DashboardComponent, CountriesComponent, etc.) en ambos módulos: tanto en el AppModule como en el AdminModule. Cada componente solo debe ser declarado en su módulo específico.

**Solución:**

1. Abre tu archivo src/app/app.module.ts
2. En el array declarations, elimina las líneas:
   * AdminComponent
   * DashboardComponent
   * CountriesComponent
   * ContinentsComponent
   * AreasComponent
   * DependenciesComponent
   * LanguagesComponent
   * TranslationsComponent
3. Deja solo AppComponent (y cualquier otro que sea común para toda la app y no pertenezca a un módulo feature).

Por ejemplo:

typescript

@NgModule({

declarations: [

AppComponent

*// ...otros componentes globales, pero ninguno de administración*

],

...

})

**export** **class** AppModule { }

En admin.module.ts deben estar todas las declaraciones de admin SIN repetir en el módulo raíz.

Una vez hecho esto, guarda y vuelve a compilar. Tus rutas de administración y el lazy-loading funcionarán perfecto.

 Voy a crear un componente ui-heading (o ui-title) similar, que soporte iconos con todas las posiciones que solicitas. Te proporciono los archivos:

ui-heading.component.ts

typescript

**import** { Component, Input, OnInit, OnChanges, SimpleChanges } **from** '@angular/core';

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

**import** { DomSanitizer, SafeHtml } **from** '@angular/platform-browser';

@Component({

selector: 'app-ui-heading',

templateUrl: './ui-heading.component.html',

styleUrls: ['./ui-heading.component.scss']

})

**export** **class** UiHeadingComponent **implements** OnInit, OnChanges {

*// Tipo de heading (h1, h2, h3, etc.)*

@Input() level: 1 | 2 | 3 | 4 | 5 | 6 = 1;

*// Texto del heading*

@Input() text: string = '';

*// Alineación del texto*

@Input() textAlign: 'left' | 'center' | 'right' = 'left';

*// Color del heading*

@Input() color: 'primary' | 'secondary' | 'accent' | 'text' | 'muted' = 'text';

*// Iconografía*

@Input() icon?: string;

@Input() iconPosition: 'left' | 'right' | 'top' | 'bottom' = 'left';

@Input() iconSize: 'xs' | 's' | 'm' | 'l' | 'xl' | number = 'm';

*// Tamaño del texto (sobreescribe el tamaño por defecto del nivel)*

@Input() textSize?: 'xs' | 's' | 'm' | 'l' | 'xl' | 'xxl' | number;

*// Accesibilidad*

@Input() ariaLabel?: string;

*// Para contenido HTML si es necesario*

@Input() allowHtml: boolean = false;

svgContent: SafeHtml | **null** = **null**;

**private** **readonly** iconSizes = { xs: 16, s: 20, m: 24, l: 32, xl: 40 };

**private** **readonly** textSizes = {

xs: '0.75rem',

s: '0.875rem',

m: '1rem',

l: '1.25rem',

xl: '1.5rem',

xxl: '2rem'

};

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer) {}

ngOnInit() {

**this**.loadSvg();

}

ngOnChanges(changes: SimpleChanges) {

**if** (changes['icon'] && !changes['icon'].isFirstChange()) {

**this**.loadSvg();

}

}

**private** loadSvg() {

**if** (**this**.icon) {

**this**.http.get(`/assets/icons/${**this**.icon}.svg`, { responseType: 'text' }).subscribe({

next: svg => **this**.svgContent = **this**.sanitizer.bypassSecurityTrustHtml(svg),

error: () => **this**.svgContent = **null**

});

} **else** {

**this**.svgContent = **null**;

}

}

getIconStyle() {

**const** size = **typeof** **this**.iconSize === 'number'

? `${**this**.iconSize}px`

: `${**this**.iconSizes[**this**.iconSize] ?? **this**.iconSizes.m}px`;

**return** {

width: size,

height: size,

minWidth: size,

minHeight: size,

verticalAlign: **this**.iconPosition === 'top' || **this**.iconPosition === 'bottom' ? 'middle' : 'text-bottom'

};

}

getTextStyle() {

**if** (**this**.textSize) {

**const** size = **typeof** **this**.textSize === 'number'

? `${**this**.textSize}px`

: **this**.textSizes[**this**.textSize] ?? **this**.textSizes.m;

**return** { fontSize: size };

}

**return** {};

}

getHeadingClasses(): string {

**return** [

'ui-heading',

`ui-heading--level-${**this**.level}`,

`ui-heading--color-${**this**.color}`,

`ui-heading--align-${**this**.textAlign}`,

**this**.icon ? 'ui-heading--has-icon' : '',

**this**.icon ? `ui-heading--icon-${**this**.iconPosition}` : ''

].filter(Boolean).join(' ');

}

getHeadingTag(): string {

**return** `h${**this**.level}`;

}

}

ui-heading.component.html

xml

<ng-container [ngSwitch]="level">

*<!-- H1 -->*

<h1 \*ngSwitchCase="1"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h1>

*<!-- H2 -->*

<h2 \*ngSwitchCase="2"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h2>

*<!-- H3 -->*

<h3 \*ngSwitchCase="3"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h3>

*<!-- H4, H5, H6 siguen el mismo patrón... -->*

<h4 \*ngSwitchCase="4"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

*<!-- Mismo contenido que H3 -->*

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h4>

<h5 \*ngSwitchCase="5"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h5>

<h6 \*ngSwitchCase="6"

[class]="getHeadingClasses()"

[style]="getTextStyle()"

[attr.aria-label]="ariaLabel">

<span \*ngIf="icon && iconPosition === 'top'"

class="ui-heading\_\_icon ui-heading\_\_icon--top"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_content">

<span \*ngIf="icon && iconPosition === 'left'"

class="ui-heading\_\_icon ui-heading\_\_icon--left"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

<span class="ui-heading\_\_text" [innerHTML]="allowHtml ? text : null">

{{ allowHtml ? '' : text }}

</span>

<span \*ngIf="icon && iconPosition === 'right'"

class="ui-heading\_\_icon ui-heading\_\_icon--right"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</span>

<span \*ngIf="icon && iconPosition === 'bottom'"

class="ui-heading\_\_icon ui-heading\_\_icon--bottom"

[innerHTML]="svgContent"

[ngStyle]="getIconStyle()"></span>

</h6>

</ng-container>

ui-heading.component.scss

text

@import '../../../../../styles/variables';

@import '../../../../../styles/mixins';

.ui-heading {

font-weight: 600;

line-height: 1.3;

margin: 0;

// Alineación de texto

&--align-left { text-align: left; }

&--align-center { text-align: center; }

&--align-right { text-align: right; }

// Colores

&--color-text { color: var(--color-text); }

&--color-primary { color: var(--color-primary); }

&--color-secondary { color: var(--color-secondary); }

&--color-accent { color: var(--color-accent); }

&--color-muted { color: var(--color-text-muted); }

// Tamaños por defecto según nivel

&--level-1 {

font-size: 2.5rem;

font-weight: 700;

margin-bottom: var(--space-l);

}

&--level-2 {

font-size: 2rem;

font-weight: 600;

margin-bottom: var(--space-m);

}

&--level-3 {

font-size: 1.75rem;

font-weight: 600;

margin-bottom: var(--space-m);

}

&--level-4 {

font-size: 1.5rem;

font-weight: 500;

margin-bottom: var(--space-s);

}

&--level-5 {

font-size: 1.25rem;

font-weight: 500;

margin-bottom: var(--space-s);

}

&--level-6 {

font-size: 1.125rem;

font-weight: 500;

margin-bottom: var(--space-xs);

}

// Layout con iconos

&--has-icon {

display: flex;

align-items: center;

gap: var(--space-xs);

}

&--icon-top {

flex-direction: column;

align-items: center;

.ui-heading\_\_content {

display: flex;

align-items: center;

gap: var(--space-xs);

}

}

&--icon-bottom {

flex-direction: column;

align-items: center;

.ui-heading\_\_content {

display: flex;

align-items: center;

gap: var(--space-xs);

}

}

&--icon-left {

.ui-heading\_\_content {

display: flex;

align-items: center;

gap: var(--space-xs);

}

}

&--icon-right {

.ui-heading\_\_content {

display: flex;

align-items: center;

gap: var(--space-xs);

}

}

.ui-heading\_\_icon {

display: inline-flex;

align-items: center;

justify-content: center;

color: currentColor; // ← ESTO HEREDA EL COLOR DEL TEXTO

transition: opacity 0.2s;

// Posiciones específicas

&--top {

margin-bottom: var(--space-xs);

}

&--bottom {

margin-top: var(--space-xs);

}

&--left {

margin-right: var(--space-xs);

}

&--right {

margin-left: var(--space-xs);

}

// Asegurarse de que el SVG herede el color

:deep(svg) {

fill: currentColor;

stroke: currentColor;

color: currentColor;

}

}

.ui-heading\_\_content {

display: flex;

align-items: center;

width: 100%;

}

.ui-heading\_\_text {

flex: 1;

display: inline;

}

}

// Responsive design

@media (max-width: 768px) {

.ui-heading {

&--level-1 { font-size: 2rem; }

&--level-2 { font-size: 1.75rem; }

&--level-3 { font-size: 1.5rem; }

&--level-4 { font-size: 1.25rem; }

&--level-5 { font-size: 1.125rem; }

&--level-6 { font-size: 1rem; }

}

}

Ejemplos de uso:

xml

*<!-- Título H1 simple -->*

<app-ui-heading

level="1"

text="Mi Título Principal">

</app-ui-heading>

*<!-- Título con icono a la izquierda -->*

<app-ui-heading

level="2"

text="Dashboard"

icon="dashboard"

iconPosition="left"

color="primary">

</app-ui-heading>

*<!-- Título centrado con icono arriba -->*

<app-ui-heading

level="1"

text="Bienvenido"

icon="welcome"

iconPosition="top"

textAlign="center"

color="accent">

</app-ui-heading>

*<!-- Título con icono personalizado y tamaño -->*

<app-ui-heading

level="3"

text="Configuración"

icon="settings"

iconPosition="right"

iconSize="l"

textSize="xl">

</app-ui-heading>

Características principales:

✅ Soporte para todos los niveles de heading (H1-H6)  
✅ Iconos en 4 posiciones: izquierda, derecha, arriba, abajo  
✅ Alineación de texto: izquierda, centro, derecha  
✅ Colores temáticos que soportan modo oscuro  
✅ Los iconos heredan el color del texto (color: currentColor)  
✅ Tamaños responsivos  
✅ Accesibilidad con aria-label  
✅ Misma estructura que tu ui-button para consistencia

¿Te gusta esta implementación o quieres que ajuste alguna parte específica?

ng generate component modules/shared/components/ui-heading --skip-tests

Elimina las declaraciones duplicadas

Solo debe declararse UiHeadingComponent en un único módulo:

* Ve a app.module.ts y elimina cualquier línea como:

typescript

**import** { UiHeadingComponent } **from** './shared/components/ui-heading/ui-heading.component';

y elimínalo también de cualquier array declarations y exports en AppModule.

* UiHeadingComponent solo debe estar en SharedModule (o el módulo donde tienes resto de UIs reutilizables).

**2.**Asegúrate de que el componente está bien declarado y exportado en shared.module.ts

Ejemplo para src/app/modules/shared/shared.module.ts:

typescript

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

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

**import** { UiHeadingComponent } **from** './components/ui-heading/ui-heading.component';

*// ... otros imports de UI*

@NgModule({

declarations: [

UiHeadingComponent,

*// otros componentes...*

],

imports: [

CommonModule

],

exports: [

UiHeadingComponent,

*// otros componentes...*

]

})

**export** **class** SharedModule {}

**3.**Importa SharedModule en AppModule

En app.module.ts:

typescript

**import** { SharedModule } **from** './modules/shared/shared.module';

@NgModule({

declarations: [

AppComponent,

*// (No pongas UiHeadingComponent aquí)*

*// otros componentes principales...*

],

imports: [

SharedModule,

*// otros módulos...*

]

})

**export** **class** AppModule {}

**4.**Revisa ruta del import en SharedModule y AppModule

Tu ruta correcta para UiHeadingComponent será:

typescript

**import** { UiHeadingComponent } **from** './components/ui-heading/ui-heading.component';

Y para SharedModule en AppModule:

typescript

**import** { SharedModule } **from** './modules/shared/shared.module';

**5.**Verifica componentes de header, footer y menu

Todos los componentes globales (app-header, app-footer, app-menu-bar, etc.) deben estar solo en un módulo que esté importado a AppModule. Si son componentes del área compartida, también van en SharedModule.

**6.**Compila de nuevo

text

ng serve

**Resumen rápido:**

* Declara UiHeadingComponent solo en SharedModule
* Exporta UiHeadingComponent en SharedModule
* Importa SharedModule en AppModule
* No declares ni importes UiHeadingComponent directamente en AppModule
* Haz lo mismo con otros componentes compartidos (header, footer, menu).

¿Quieres ayuda para ajustar aquí tus imports (compárteme los arrays reales si necesitas)?

Aquí tienes tres ejemplos de uso del componente ui-heading para tu Dashboard, mostrando diferentes posibles configuraciones:

**1.**Título simple, H1, alineado a la izquierda:

xml

<app-ui-heading

[Level]="1"

text="Panel de Administración"

color="primary"

textAlign="left">

</app-ui-heading>

**2.**Título con icono a la izquierda, H2, centrado:

xml

<app-ui-heading

[level]="2"

text="Dashboard"

icon="dashboard"

iconPosition="left"

color="accent"

textAlign="center"

ariaLabel="Entrar al dashboard">

</app-ui-heading>

**3.**Título con icono arriba, H1, color personalizado y tamaño grande:

xml

<app-ui-heading

[level]="1"

text="Resumen Diario"

icon="calendar"

iconPosition="top"

color="secondary"

textAlign="center"

textSize="xl">

</app-ui-heading>

Solo copia una de estas líneas en el HTML de tu página/dashboard, ajustando los textos, iconos o propiedades que quieras mostrar!  
El icono debe existir como SVG en tu carpeta /assets/icons con el nombre que indiques (por ejemplo, dashboard.svg, calendar.svg).

Aquí tienes el planteamiento y los pasos para crear tu componente BEM de Tarjeta Stats reutilizable con icono:

**1.**Comando Angular para el componente

bash

ng generate component shared/components/ui-stat-card --skip-tests

**2.**HTML (ui-stat-card.component.html):

xml

<div class="ui-stat-card">

<span \*ngIf="icon" class="ui-stat-card\_\_icon">

<svg \*ngIf="svgContent" [innerHTML]="svgContent" [ngStyle]="{ width: iconSize, height: iconSize }"></svg>

</span>

<span class="ui-stat-card\_\_label">{{ label }}</span>

<span class="ui-stat-card\_\_value">{{ value }}</span>

</div>

**3.**TypeScript (ui-stat-card.component.ts):

typescript

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

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

**import** { DomSanitizer, SafeHtml } **from** '@angular/platform-browser';

@Component({

selector: 'app-ui-stat-card',

templateUrl: './ui-stat-card.component.html',

styleUrls: ['./ui-stat-card.component.scss']

})

**export** **class** UiStatCardComponent **implements** OnInit {

@Input() label!: string;

@Input() value!: string | number;

@Input() icon?: string; *// nombre svg sin extensión*

@Input() iconSize: string = '40px'; *// XL por defecto*

svgContent: SafeHtml | **null** = **null**;

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer) {}

ngOnInit() {

**if** (**this**.icon) {

**this**.http.get(`/assets/icons/${**this**.icon}.svg`, { responseType: 'text' }).subscribe({

next: svg => **this**.svgContent = **this**.sanitizer.bypassSecurityTrustHtml(svg),

error: () => **this**.svgContent = **null**

});

}

}

}

**4.**SCSS (ui-stat-card.component.scss, 100% BEM y adaptable):

text

.ui-stat-card {

background: var(--color-surface, #fff);

border-radius: var(--radius-md, 9px);

box-shadow: 0 3px 10px rgba(0,0,0,0.07);

padding: var(--space-m, 1em) var(--space-xl, 2em);

min-width: 160px;

text-align: center;

display: flex;

flex-direction: column;

align-items: center;

justify-content: flex-start;

transition: box-shadow 0.3s;

&\_\_icon {

display: flex;

justify-content: center;

align-items: center;

margin-bottom: var(--space-s, 0.5em);

svg {

fill: var(--color-primary, #10416a);

color: var(--color-primary, #10416a);

width: 40px; // XL por defecto

height: 40px;

transition: fill 0.2s;

}

}

&\_\_label {

margin: 0.5em 0 0.5em 0;

color: var(--color-tertiary, #727272);

font-weight: 600;

font-size: 1em;

}

&\_\_value {

font-size: 2.2em;

font-weight: 700;

color: var(--color-text, #222);

background: none;

letter-spacing: 0.03em;

}

}

**5.**Ejemplo de uso en tu dashboard

En tu HTML principal:

xml

<div class="dashboard-admin\_\_cards">

<app-ui-stat-card

label="Usuarios activos"

[value]="452"

icon="user"

iconSize="56px">

</app-ui-stat-card>

<app-ui-stat-card

label="Países registrados"

[value]="193"

icon="flag"

iconSize="56px">

</app-ui-stat-card>

*<!-- ...más tarjetas -->*

</div>

**6.**Estilos generales y responsive

Mantén tu breakpoints y variables, solo cambia la clase .dashboard-admin\_\_card por .ui-stat-card en tus SCSS globales o importa el CSS del componente.

\*\*¿Quieres que te prepare algún icono SVG, lógica de valores desde API, o alguna variante visual (colores por tipo) en la tarjeta?\*\*Para crear tu módulo de tarjeta estadística reutilizable adaptado a BEM y incluir el icono XL arriba, tienes que:

1. Crear el componente:

bash

1. ng generate component modules/shared/components/ui-stat-card --skip-tests
2. Copiar estilos del dashboard al nuevo componente, adaptando la clase raíz:

text

.ui-stat-card {

background: var(--color-surface, #fff);

border-radius: var(--radius-md, 9px);

box-shadow: 0 3px 10px rgba(0,0,0,0.07);

padding: var(--space-m, 1em) var(--space-xl, 2em);

min-width: 160px;

text-align: center;

display: flex;

flex-direction: column;

align-items: center;

&\_\_icon {

display: flex;

justify-content: center;

align-items: center;

margin-bottom: var(--space-s, 0.5em);

svg {

width: 40px; height: 40px;

fill: var(--color-primary, #10416a);

color: var(--color-primary, #10416a);

}

}

&\_\_label {

margin: 0.5em 0;

color: var(--color-tertiary, #727272);

font-weight: 600;

font-size: 1em;

}

&\_\_value {

font-size: 2.2em;

font-weight: 700;

color: var(--color-text, #222);

background: none;

letter-spacing: 0.03em;

}

}

1. HTML del componente (ui-stat-card.component.html):

xml

<div class="ui-stat-card">

<span \*ngIf="icon" class="ui-stat-card\_\_icon">

<svg \*ngIf="svgContent" [innerHTML]="svgContent"></svg>

</span>

<span class="ui-stat-card\_\_label">{{ label }}</span>

<span class="ui-stat-card\_\_value">{{ value }}</span>

</div>

1. TypeScript (ui-stat-card.component.ts):

typescript

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

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

**import** { DomSanitizer, SafeHtml } **from** '@angular/platform-browser';

@Component({

selector: 'app-ui-stat-card',

templateUrl: './ui-stat-card.component.html',

styleUrls: ['./ui-stat-card.component.scss']

})

**export** **class** UiStatCardComponent **implements** OnInit {

@Input() label!: string;

@Input() value!: string | number;

@Input() icon?: string; *// nombre svg sin extensión*

svgContent: SafeHtml | **null** = **null**;

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer) {}

ngOnInit() {

**if** (**this**.icon) {

**this**.http.get(`/assets/icons/${**this**.icon}.svg`, { responseType: 'text' })

.subscribe({

next: svg => **this**.svgContent = **this**.sanitizer.bypassSecurityTrustHtml(svg),

error: () => **this**.svgContent = **null**

});

}

}

}

1. Uso en dashboard (sustituyendo las tarjetas actuales):

xml

<div class="dashboard-admin\_\_cards">

<app-ui-stat-card label="Usuarios activos" value="452" icon="user"></app-ui-stat-card>

<app-ui-stat-card label="Países registrados" value="193" icon="flag"></app-ui-stat-card>

*<!-- ... -->*

</div>

* El icono SVG XL se muestra arriba del título, tal y como pides.
* Todo el CSS lo llevas y adaptas al nuevo componente.
* Lo puedes extender para color, fondo, etc., según tu sistema.

¿Quieres que te haga una variante para fondo oscuro, o necesitas un input extra para personalizar el color del icono?Para crear el módulo de tarjetas reutilizables y BEM, realiza estos pasos:

1. Crea el componente:

bash

ng generate component shared/components/ui-stat-card --skip-tests

1. Copia los estilos y renómbralos a .ui-stat-card, siguiendo BEM:

text

.ui-stat-card {

background: var(--color-surface, #fff);

border-radius: var(--radius-md, 9px);

box-shadow: 0 3px 10px rgba(0,0,0,0.07);

padding: var(--space-m, 1em) var(--space-xl, 2em);

min-width: 160px;

display: flex;

flex-direction: column;

align-items: center;

text-align: center;

&\_\_icon {

margin-bottom: var(--space-s, 0.5em);

svg {

width: 40px; height: 40px;

fill: var(--color-primary, #10416a);

color: var(--color-primary, #10416a);

}

}

&\_\_label {

color: var(--color-tertiary, #727272);

margin-bottom: 0.25em;

font-size: 1em; font-weight: 600;

}

&\_\_value {

font-size: 2.2em; font-weight: 700;

color: var(--color-text, #222);

letter-spacing: 0.03em;

}

}

1. HTML del componente (ui-stat-card.component.html):

xml

<div class="ui-stat-card">

<span \*ngIf="icon" class="ui-stat-card\_\_icon">

<svg \*ngIf="svgContent" [innerHTML]="svgContent"></svg>

</span>

<span class="ui-stat-card\_\_label">{{ label }}</span>

<span class="ui-stat-card\_\_value">{{ value }}</span>

</div>

1. TypeScript (ui-stat-card.component.ts):

typescript

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

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

**import** { DomSanitizer, SafeHtml } **from** '@angular/platform-browser';

@Component({

selector: 'app-ui-stat-card',

templateUrl: './ui-stat-card.component.html',

styleUrls: ['./ui-stat-card.component.scss']

})

**export** **class** UiStatCardComponent **implements** OnInit {

@Input() label!: string;

@Input() value!: string | number;

@Input() icon?: string; *// nombre svg*

svgContent: SafeHtml | **null** = **null**;

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer) {}

ngOnInit() {

**if** (**this**.icon) {

**this**.http.get(`/assets/icons/${**this**.icon}.svg`, { responseType: 'text' }).subscribe({

next: svg => **this**.svgContent = **this**.sanitizer.bypassSecurityTrustHtml(svg),

error: () => **this**.svgContent = **null**

});

}

}

}

1. Usa el nuevo componente:

xml

<div class="dashboard-admin\_\_cards">

<app-ui-stat-card label="Usuarios activos" value="452" icon="user"></app-ui-stat-card>

<app-ui-stat-card label="Países registrados" value="193" icon="flag"></app-ui-stat-card>

</div>

Así tendrás tarjetas BEM, con icono XL arriba, estilos limpios y reutilizables. ¿Necesitas variante de color o inputs extra?

1. Asegúrate de declarar y exportar el componente en tu shared.module.ts (en la misma carpeta):

typescript

**import** { UiStatCardComponent } **from** './components/ui-stat-card/ui-stat-card.component';

@NgModule({

declarations: [

UiStatCardComponent,

*// otros...*

],

exports: [

UiStatCardComponent,

*// otros...*

]

})

1. Ya puedes usar <app-ui-stat-card> en cualquier HTML del proyecto.

¡Perfecto! Si quieres, pégame “los campos de la tabla” pero para el conteo total de registros en una tabla (como countries), NO es necesario conocer los campos individuales, solo el nombre de la tabla.

A continuación te detallo todo el flujo completo y comentado (Node.js/Express + SQLite + Angular) para mostrar el total de countries en el Dashboard:

A. Backend (Node.js/Express + SQLite)

1. Ruta API REST sencilla (total países):

javascript

*// api/countries/count.js*

**const** express = require('express');

**const** router = express.Router();

**const** sqlite3 = require('sqlite3').verbose();

**const** db = **new** sqlite3.Database('./path/to/yourdb.sqlite'); *// Pon tu ruta*

router.get('/count', (req, res) => {

db.get('SELECT COUNT(\*) as total FROM countries', [], (err, row) => {

**if** (err) **return** res.status(500).json({ error: err.message });

res.json({ total: row.total });

});

});

module.exports = router;

*// En tu app.js principal:*

*/\**

*const countriesCountRoute = require('./api/countries/count');*

*app.use('/api/countries', countriesCountRoute);*

*\*/*

B. Frontend Angular

1. Servicio Angular para consultar el conteo

typescript

*// src/app/services/countries.service.ts*

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

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

**import** { Observable } **from** 'rxjs';

@Injectable({ providedIn: 'root' })

**export** **class** CountriesService {

constructor(**private** http: HttpClient) {}

getCountriesCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('/api/countries/count');

}

}

2. Componente Dashboard: usar el servicio

typescript

*// src/app/modules/admin/components/admin-dashboard/admin-dashboard.component.ts*

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

**import** { CountriesService } **from** 'src/app/services/countries.service';

@Component({

selector: 'app-admin-dashboard',

templateUrl: './admin-dashboard.component.html',

styleUrls: ['./admin-dashboard.component.scss']

})

**export** **class** AdminDashboardComponent **implements** OnInit {

countriesCount: number | **null** = **null**;

constructor(**private** countriesService: CountriesService) {}

ngOnInit() {

**this**.countriesService.getCountriesCount().subscribe({

next: result => **this**.countriesCount = result.total,

error: err => **this**.countriesCount = **null**

});

}

}

3. Template Dashboard (HTML):

xml

*<!-- src/app/modules/admin/components/admin-dashboard/admin-dashboard.component.html -->*

<div class="dashboard-stat">

<span>Total de países:</span>

<span class="stat-number">

{{ countriesCount !== null ? countriesCount : '...' }}

</span>

</div>

*(Puedes estilizar como quieras)*

4. Recomendación Extra

Cuando esto funcione, puedes hacer lo mismo para las demás entidades, usando rutas /api/<tabla>/count o generalizando la API.

¿Quieres que extienda el código para *varias tablas* en la misma consulta/servicio? ¿O prefieres primero probar solo countries?  
Si necesitas rutas de Express más avanzadas (por ejemplo, /api/stats?table=countries para usar el mismo endpoint para varios), te lo adapto también.

Perfecto! Aquí tienes cómo integrar el dato dinámico (total países) en tu dashboard, manteniendo tu estructura y componentes UI:

**Ejemplo aplicado a tu código:**

xml

<div class="dashboard-admin">

<app-ui-heading

[level]="1"

text="Panel de Administración"

icon="icon-dashboard"

iconPosition="left"

color="primary"

textAlign="left"

ariaLabel="Entrar al dashboard">

</app-ui-heading>

<div class="dashboard-admin\_\_cards">

<app-ui-stat-card label="Usuarios activos" value="452" icon="icon-user"></app-ui-stat-card>

*<!-- Este es el stat dinámico de países -->*

<app-ui-stat-card label="Países registrados" [value]="countriesCount" icon="icon-country"></app-ui-stat-card>

<app-ui-stat-card label="Continentes" value="452" icon="icon-continents"></app-ui-stat-card>

<app-ui-stat-card label="Áreas" value="193" icon="icon-area"></app-ui-stat-card>

*<!-- Más tarjetas según lo necesario -->*

</div>

</div>

Cambia value="193" por [value]="countriesCount".  
Así lo tomará del componente TS.

**En el TS del componente (admin-dashboard.component.ts):**

typescript

**export** **class** AdminDashboardComponent **implements** OnInit {

countriesCount: number | **null** = **null**;

constructor(**private** countriesService: CountriesService) {}

ngOnInit() {

**this**.countriesService.getCountriesCount().subscribe({

next: result => **this**.countriesCount = result.total,

error: err => **this**.countriesCount = **null**

});

}

}

Así tendrás el valor del backend en tu dashboard, 100% dinámico y escalable. Si quieres agregar más estadísticas, repite este patrón para usuarios, continentes, áreas, etc.

¿Quieres que lo extienda para todos los <app-ui-stat-card> y te agregue el servicio multiplataforma?Para integrarlo solo debes cambiar la tarjeta de países así:

xml

<app-ui-stat-card label="Países registrados" [value]="countriesCount" icon="icon-country"></app-ui-stat-card>

Y en tu componente TS, cargando el valor con el servicio:

typescript

countriesCount: number|**null** = **null**;

ngOnInit() {

**this**.countriesService.getCountriesCount().subscribe(resp => **this**.countriesCount = resp.total);

}

¡Así quedará vinculada y será dinámica! Si quieres que otros stats sean dinámicos, haz lo mismo para cada servicio y reemplaza los valores hardcodeados por los nombres de variable en [value].

Solución correcta (Angular):

En el HTML, haz esto:

xml

<app-ui-stat-card

label="Países registrados"

[value]="countriesCount ?? ''"

icon="icon-country">

</app-ui-stat-card>

* ?? '' fuerza que si countriesCount es null, se pase *string vacío*, que sí es válido.

Alternativas equivalentes:

* [value]="countriesCount || ''"
* [value]="countriesCount === null ? '' : countriesCount"

Resumen

* Mantén el tipo en el TS: countriesCount: number|null = null;
* El HTML:  
  <app-ui-stat-card ... [value]="countriesCount ?? ''" ... ></app-ui-stat-card>

Compila de nuevo y el error desaparece. Así puedes seguir trabajando con datos dinámicos aunque el valor aún no haya llegado.

npm install sqlite3 dento de backend

El error persiste porque la ruta que usas en el constructor de SQLite:

js

**const** db = **new** sqlite3.Database(path.join(\_\_dirname, './db/countries.db'));

realmente intenta acceder a una ruta relativa donde el subdirectorio db está dentro de la carpeta api/countries/; es decir, busca la base de datos en:

text

D:\desarrollos\countries2\backend\api\countries\db\countries.db

pero tu archivo de base de datos está en:

text

D:\desarrollos\countries2\backend\db\countries.db

Por eso tienes el error SQLITE\_CANTOPEN — el archivo y/o la carpeta no existen en la ubicación buscada.

**Solución**

Deberías cambiar la línea en count.js por esta, para que la ruta apunte a la carpeta db en el backend:

js

**const** db = **new** sqlite3.Database(path.join(\_\_dirname, '..', '..', 'db', 'countries.db'));

Esto funcionará porque:

* \_\_dirname apunta a D:\desarrollos\countries2\backend\api\countries
* .., .. retrocede dos niveles hasta D:\desarrollos\countries2\backend
* Luego añade db\countries.db

Guarda el archivo, verifica que el archivo countries.db realmente existe en la carpeta indicada, y vuelve a lanzar:

text

npm run dev

Con eso el backend debería poder abrir la base de datos correctamente y dejar de caerse. Ahora sí podrás acceder con tu comando curl.El problema es que la ruta actual en tu código:

js

**const** db = **new** sqlite3.Database(path.join(\_\_dirname, './db/countries.db'));

está buscando el archivo de base de datos en una ruta relativa a /api/countries/, por lo que realmente intenta abrir:

text

D:\desarrollos\countries2\backend\api\countries\db\countries.db

Sin embargo, tu archivo real está en:

text

D:\desarrollos\countries2\backend\db\countries.db

**Solución**

Cambia la línea por:

js

**const** db = **new** sqlite3.Database(path.join(\_\_dirname, '..', '..', 'db', 'countries.db'));

Así el programa buscará la base de datos donde realmente está, en la ruta esperada. Guarda el archivo, asegúrate de que el archivo countries.db existe efectivamente en la carpeta db, y vuelve a ejecutar:

text

npm run dev

Con esto, el backend debería poder abrir correctamente el archivo SQLite y el error desaparecerá. Luego podrás acceder con curl sin problemas.

**Solución**

Debes indicarle a Angular que las peticiones API sean redirigidas al puerto y host del backend, normalmente configurando un proxy en desarrollo. Haz lo siguiente:

1. Crea (o edita) el archivo proxy.conf.json en la raíz de tu proyecto Angular (frontend), con este contenido:

json

{

"/api": {

"target": "http://localhost:3000",

"secure": false

}

}

1. Modifica el comando de arranque del frontend en el package.json de Angular para incluir el proxy, por ejemplo:

text

ng serve --proxy-config proxy.conf.json

1. Lanza el frontend con ese comando.

Esto hará que todas las llamadas a rutas que comiencen por /api desde el frontend sean redirigidas al backend Express en el puerto 3000, y la ruta /api/countries/count funcionará correctamente.

Si prefieres no usar proxy, deberás cambiar manualmente en el código Angular la URL del endpoint para que apunte a http://localhost:3000/api/countries/count, pero la solución recomendada para desarrollo es el proxy.El error 404 (Not Found) en el log de Angular indica que el frontend está intentando acceder al endpoint /api/countries/count en el propio servidor de desarrollo (localhost:4200), pero esa ruta no existe allí. El backend Express, que expone el endpoint real, está corriendo en localhost:3000, así que el frontend debe redirigir sus peticiones a ese puerto para que funcionen correctamente.

**Solución recomendada: Proxy de Angular**

1. Crea un archivo proxy.conf.json en la raíz de tu proyecto Angular (en la carpeta frontend):

json

{

"/api": {

"target": "http://localhost:3000",

"secure": false

}

}

1. Lanza el servidor de Angular con el parámetro de proxy:

text

ng serve --proxy-config proxy.conf.json

Esto hará que las llamadas a /api/countries/count se redirijan automáticamente al backend Express en el puerto 3000, evitando el error 404 en desarrollo.

**Alternativa sin proxy**

En tu código Angular, cambia la URL completa en la petición HTTP para que apunte a http://localhost:3000/api/countries/count en vez de /api/countries/count. Sin embargo, la configuración del proxy es la forma estándar y más cómoda de integrar Angular con Node.js en desarrollo.

Aquí tienes los pasos para consumir y mostrar el número de registros para las tablas areas, continents, dependencies, languages y multilingualnames, siguiendo la estructura que ya tienes para countries y evitando errores pasados:

Backend (API):  
Debes crear un endpoint similar al de countries/count para cada tabla en tu archivo de rutas. Ejemplo para áreas:

javascript

*// areasCount.js*

**const** express = require('express');

**const** router = express.Router();

**const** sqlite3 = require('sqlite3').verbose();

**const** path = require('path');

**const** db = **new** sqlite3.Database(path.join(\_\_dirname, '..', '..', 'db', 'countries.db'));

router.get('/count', (req, res) => {

db.get('SELECT COUNT(\*) as total FROM areas', [], (err, row) => {

**if** (err) **return** res.status(500).json({ error: err.message });

res.json({ total: row.total });

});

});

module.exports = router;

Repite esto para las otras tablas y monta los endpoints en tu servidor Express, por ejemplo:

* /api/areas/count
* /api/continents/count
* /api/dependencies/count
* /api/languages/count
* /api/multilingualnames/count

Frontend (servicio Angular):  
En tu servicio, crea una función para cada endpoint usando la misma estructura que ya tienes:

typescript

*// countries.service.ts*

getAreasCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('http://localhost:3000/api/areas/count');

}

getContinentsCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('http://localhost:3000/api/continents/count');

}

getDependenciesCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('http://localhost:3000/api/dependencies/count');

}

getLanguagesCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('http://localhost:3000/api/languages/count');

}

getTranslationsCount(): Observable<{ total: number }> {

**return** **this**.http.get<{ total: number }>('http://localhost:3000/api/multilingualnames/count');

}

Así evitas inconsistencias y mantienes una arquitectura uniforme.

Dashboard (componente Angular):  
En tu componente, declara las variables y consume los servicios en el ngOnInit:

typescript

*// admin-dashboard.component.ts*

areasCount: number | **null** = **null**;

continentsCount: number | **null** = **null**;

dependenciesCount: number | **null** = **null**;

languagesCount: number | **null** = **null**;

translationsCount: number | **null** = **null**;

ngOnInit(): **void** {

**this**.countriesService.getCountriesCount().subscribe({

next: (result) => **this**.countriesCount = result.total,

error: (err) => **this**.countriesCount = **null**

});

**this**.countriesService.getAreasCount().subscribe({

next: (result) => **this**.areasCount = result.total,

error: (err) => **this**.areasCount = **null**

});

*// Repite para el resto...*

}

Dashboard HTML:  
Usa los valores en la vista:

xml

<app-ui-stat-card label="Áreas" [value]="areasCount ?? 0" icon="icon-area"></app-ui-stat-card>

<app-ui-stat-card label="Continentes" [value]="continentsCount ?? 0" icon="icon-continents"></app-ui-stat-card>

<app-ui-stat-card label="Dependencias" [value]="dependenciesCount ?? 0" icon="icon-dependencies"></app-ui-stat-card>

<app-ui-stat-card label="Idiomas" [value]="languagesCount ?? 0" icon="icon-languages"></app-ui-stat-card>

<app-ui-stat-card label="Traducciones" [value]="translationsCount ?? 0" icon="icon-translations"></app-ui-stat-card>

Checklist para evitar errores:

* Los endpoints de API deben devolver res.json({ total: row.total })
* Las funciones del servicio Angular deben coincidir exactamente con la ruta del endpoint de API.
* Maneja los errores en el componente asignando null y mostrando 0 en la interfaz.
* Repite siempre la arquitectura que ya tienes funcionando (countries).

Solo asegúrate de montar bien cada router en el servidor principal (app.js/serve.js):

js

app.use('/api/areas', require('./routes/areasCount'));

app.use('/api/continents', require('./routes/continentsCount'));

app.use('/api/dependencies', require('./routes/dependenciesCount'));

app.use('/api/languages', require('./routes/languagesCount'));

app.use('/api/multilingualnames', require('./routes/translationsCount'));

Revisión rápida:

* En tu app.js tienes líneas como:

js

app.use('/api/areas', require('./routes/areasCount'));

app.use('/api/continents', require('./routes/continentsCount'));

app.use('/api/dependencies', require('./routes/dependenciesCount'));

app.use('/api/languages', require('./routes/languagesCount'));

app.use('/api/multilingualnames', require('./routes/translationsCount'));

* Esto implica que los archivos deben estar realmente en la carpeta:  
  D:\desarrollos\countries2\backend\routes\

Solución:

1. Verifica que todos los archivos (areasCount.js, continentsCount.js, etc) están en la ruta correcta:
   * La carpeta debe ser backend/routes/
   * NO en la raíz ni en otra subcarpeta.
2. Si por ejemplo tienes areasCount.js en backend/ en vez de backend/routes/, simplemente muévelo:
   * De: D:\desarrollos\countries2\backend\areasCount.js
   * A: D:\desarrollos\countries2\backend\routes\areasCount.js
3. Haz esto para todos los archivos de conteo y asegúrate de que estén en la carpeta correcta.

¿Lo hiciste así?  
Porque la referencia require('./routes/areasCount') es relativa al archivo principal app.js y busca dentro de la carpeta routes.

Resumen rápido:

* Todos los archivos JS tipo <tabla>Count.js deben ir en backend/routes/
* Si están bien y sigue el error, verifica typo o mayúsculas/minúsculas.
* Si mueves los archivos, reinicia nodemon.

Sí, puedes colocar todos tus archivos <tabla>Count.js en el directorio api en vez de routes para unificar la lógica de endpoints tipo conteo.

Solamente tienes que ajustar las rutas de los require en tu app.js:

* Usa:

js

app.use('/api/areas', require('./api/areas/count'));

app.use('/api/continents', require('./api/continents/count'));

app.use('/api/dependencies', require('./api/dependencies/count'));

app.use('/api/languages', require('./api/languages/count'));

app.use('/api/multilingualnames', require('./api/multilingualnames/count'));

* Renombra si quieres los archivos como count.js dentro de cada carpeta (areas/count.js, etc), igual que ya tienes para countries.

Ventajas:

* Mantienes todos los endpoints REST organizados dentro de api/<tabla>/count.js.
* El import en app.js se vuelve más legible y consistente.

Ejemplo final del app.js:

js

app.use('/api/countries', require('./api/countries/count'));

app.use('/api/areas', require('./api/areas/count'));

app.use('/api/continents', require('./api/continents/count'));

app.use('/api/dependencies', require('./api/dependencies/count'));

app.use('/api/languages', require('./api/languages/count'));

app.use('/api/multilingualnames', require('./api/multilingualnames/count'));

Conclusión:  
¡Esto es perfectamente válido y recomendable! Renombra y mueve los archivos, actualiza tus require, y funcionará correctamente.

**1.**ui-stat-card.component.html

Reemplaza el contenido por lo siguiente para convertir la card en un botón accesible y navegable:

xml

<button

class="ui-stat-card"

[ngClass]="{ 'active': isActive }"

(click)="onClick()"

type="button"

tabindex="0"

[attr.aria-label]="label"

>

<span class="ui-stat-card\_\_icon" \*ngIf="svgContent" [innerHTML]="svgContent"></span>

<span class="ui-stat-card\_\_label">{{ label }}</span>

<span class="ui-stat-card\_\_value">{{ value }}</span>

</button>

* Elimina cualquier div, usa solo button.
* Usa las clases BEM que ya tienes.
* tabindex="0" y aria-label mejoran la accesibilidad.

**2.**ui-stat-card.component.ts

Agrega un Input adicional para la ruta y un método de click que navegue al detalle:

typescript

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

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

@Component({

selector: 'app-ui-stat-card',

templateUrl: './ui-stat-card.component.html',

styleUrls: ['./ui-stat-card.component.scss']

})

**export** **class** UiStatCardComponent **implements** OnInit {

@Input() label!: string;

@Input() value!: string | number;

@Input() icon?: string;

@Input() detailRoute?: string; *// NUEVO*

svgContent: SafeHtml | **null** = **null**;

isActive = false;

constructor(**private** http: HttpClient, **private** sanitizer: DomSanitizer, **private** router: Router) {}

ngOnInit() { */\* tu lógica svg \*/* }

onClick(): **void** {

**if** (**this**.detailRoute) {

**this**.router.navigate([**this**.detailRoute]);

}

}

}

* Usa [detailRoute] para la navegación fácilmente desde el dashboard.

**3.**ui-stat-card.component.scss

Acomoda los estilos para el comportamiento visual del botón, reutilizando variables:

text

.ui-stat-card {

// ... tus estilos actuales ...

cursor: pointer;

border: none;

outline: none;

transition: box-shadow 0.2s, background 0.2s;

&:hover,

&:focus {

box-shadow: 0 6px 20px rgba(0,0,0,0.15);

background: var(--color-surface-hover, #e5f1fb);

}

&:active,

&.active {

background: var(--color-primary-light, #e0eeff);

box-shadow: 0 2px 6px rgba(0,0,0,0.13);

outline: 2px solid var(--color-primary, #10416a);

}

}

* Reutiliza tus variables y paleta corporativa.
* Añade hover, focus, active.

**4.**En el dashboard (ejemplo en admin-dashboard.component.html):

xml

    <app-ui-stat-card

      label="Países"

      [value]="countriesCount ?? ''"

      icon="icon-country"

      detailRoute="admin/countries"

    ></app-ui-stat-card>

    <app-ui-stat-card

      label="Continentes"

      [value]="continentsCount ?? ''"

      icon="icon-continents"

      detailRoute="admin/continents"

    ></app-ui-stat-card>

    <app-ui-stat-card

      label="Áreas"

      [value]="areasCount ?? ''"

      icon="icon-area"

      detailRoute="admin/areas"

    ></app-ui-stat-card>

    <app-ui-stat-card

      label="Dependencias"

      [value]="dependenciesCount ?? ''"

      icon="icon-dependencies"

      detailRoute="admin/dependencies"

    ></app-ui-stat-card>

    <app-ui-stat-card

      label="Lenguajes"

      [value]="languagesCount ?? ''"

      icon="icon-languages"

      detailRoute="admin/languages"

    ></app-ui-stat-card>

    <app-ui-stat-card

      label="Traducciones"

      [value]="translationsCount ?? ''"

      icon="icon-translations"

      detailRoute="admin/translations"

    ></app-ui-stat-card>

Con esto:

* Cada ficha es un botón, navega al detalle al hacer click.
* Estilos uniformes y reutilizables.
* Accesibilidad garantizada.
* Reutilizas tu SCSS y estructura BEM actual.

¿Quieres el snippet completo de dashboard para copiar-pegar? ¿O necesitas adaptar la parte de rutas?

**Primero:**Diseño General de la Estructura y Componentización

1. Estructura Recomendada de Componentes (Angular):

* Tabla paginada y ordenable:
  + Componente reutilizable para renderizar listas con paginación, orden y selección múltiple.
  + Inputs: columnas, datos, configuraciones de paginado, orden, filtros.
* Buscador/filtro:
  + Componente propio para filtrar por texto y/o columna específica.
* Modal para edición/alta de registro:
  + Componente modal reutilizable con inputs dinámicos según los campos recibidos.
* Zona común de botones:
  + Componente shared-buttons para "Nuevo X", "Editar", "Borrar". Integra ui-button y activa/desactiva según selección.
* Componente de bandera:
  + Componente flag-icon. Recibe código ISO2 y renderiza la imagen desde assets.
* Componente de checkboxes múltiples:
  + Para selección por fila y selección global.
* Service compartido:
  + Encapsula llamadas HTTP CRUD.
* Interfaces/types:
  + Para tipar cada tipo de registro (Country, Area, etc.)

Directorio recomendado:  
/src/app/modules/shared/components/  
Allí irán los componentes genéricos para máxima reutilización.

2. Primer paso concreto:  
Vamos a crear un esquema base del componente de la tabla paginada, y del flag-icon para países (serán los dos componentes "core" que luego se expandirán).

Paso 1: Componente tabla reutilizable

* Input:
  + columns: TableColumn[] (definición metadatos columna: clave, etiqueta, tipo, ordenable...)
  + rows: T[]
  + pagination: PaginationOptions
  + search: string (opcional)
  + selection: Set<T> para gestionar selección múltiple

Paso 2: Componente flag-icon

* Input:
  + iso2: string
* Output:
  + Renderiza <img src="assets/icons/flags/{iso2}.svg"/>

¡Perfecto! Ahora sí, te propongo el DISEÑO Y ARQUITECTURA BASE para tu módulo de administración de países (*countries*) que será totalmente reutilizable para áreas, continentes, lenguajes, etc.

**1.**Estructura y carpetas recomendada (Angular + BEM + Modular):

text

src/app/modules/

|-- shared/

| |-- components/

| |-- flag-icon/

| |-- search-box/

| |-- dropdown/

| |-- accessible-form/

| |-- confirm-dialog/

| |-- paginated-table/

| |-- toolbar-buttons/

|-- admin/

| |-- pages/

| |-- countries/

| |-- admin-countries.component.ts

| |-- admin-countries.component.html

| |-- admin-countries.component.scss

| |-- (tests, models, etc.)

* Todo componente que pueda usarse con áreas, lenguajes o continentes va a shared/components
* Las páginas como countries, areas, etc., van a admin/pages/

**2.**Arquitectura de *admin-countries* (base para cualquier entidad):

La página tendrá:

* Campo de búsqueda (un filtro simple y rápido)
* Toolbar de botones (Nuevo, Editar, Borrar; usable y activable por selección)
* Tabla paginada, ordenable y seleccionable por checkbox
* Icono decorativo (bandera reutilizada como componente)
* Modal para alta/edición (reusable con cualquier modelo/campos)
* Dialog de confirmación para borrar
* Dropdowns para filtros avanzados (si se requiere)

Ejemplo esquemático de integración HTML

xml

<div class="countries-admin">

*<!-- Filtro, Toolbar, Tabla, Modal, Diálogo -->*

<app-search-box (search)="onFilter($event)"></app-search-box>

<app-toolbar-buttons

[selected]="selectedItems"

(new)="onNew()"

(edit)="onEdit()"

(delete)="onDelete()"

entity="país">

</app-toolbar-buttons>

<app-paginated-table

[items]="filteredCountries"

[columns]="countryColumns"

[pageSize]="10"

(selectRows)="onSelectRows($event)"

(sort)="onSort($event)">

*<!-- Renderizado custom por columna -->*

<ng-template #countryTemplate let-country>

<app-flag-icon [iso2]="country.alpha2may"></app-flag-icon>

{{country.defaultname}}

</ng-template>

</app-paginated-table>

<app-accessible-form

\*ngIf="showModal"

[form]="countryForm"

[fields]="countryFields"

(submitForm)="onFormSubmit($event)">

</app-accessible-form>

<app-confirm-dialog

[visible]="showConfirm"

[message]="confirmMsg"

(confirm)="onConfirmDelete()"

(cancel)="showConfirm = false">

</app-confirm-dialog>

</div>

Ejemplo typescript base

typescript

*// admin-countries.component.ts*

selectedItems = [];

filteredCountries = [];

countryColumns = [

{ key: 'select', label: '', type: 'checkbox' },

{ key: 'flag', label: '', type: 'custom', template: countryTemplate },

{ key: 'defaultname', label: 'Nombre', sortable: true },

{ key: 'alpha2may', label: 'Código ISO2', sortable: true },

*// ...otros campos*

];

countryForm: FormGroup;

countryFields: FieldConfig[]; *// Define los campos para el formulario*

onFilter(term: string) { */\* filtra países \*/* }

onSelectRows(rows: any[]) { **this**.selectedItems = rows; }

onNew() { */\* abre modal en modo alta \*/* }

onEdit() { */\* abre modal y carga país seleccionado \*/* }

onDelete() { */\* muestra confirm-dialog \*/* }

onFormSubmit(data: any) { */\* alta/edición real \*/* }

onConfirmDelete() { */\* elimina país/es seleccionados \*/* }

Puntos clave y buenas prácticas

* Todos los componentes son inputs/outputs y templates customizables (lo que facilita la reutilización)
* SCSS BEM: usa bloques como .countries-admin\_\_toolbar, .countries-admin\_\_table, etc. para la página y componentes
* La integración de banderas e íconos se hace con inputs dinámicos
* Las acciones CRUD solo se activan si hay registros seleccionados (botones en zona común, nunca en la fila)

**3.**Filosofía para área, continente, idioma, etc.

* Replica exactamente el mismo esquema, cambiando solo el modelo de datos y los campos/columnas/renderizado de cada tabla y form.
* Todos los componentes *shared* ya estarán listos solo para inyectar props dinámicos.

**1.**Estructura recomendada

toolbar-buttons.component.ts

typescript

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

@Component({

selector: 'app-toolbar-buttons',

templateUrl: './toolbar-buttons.component.html',

styleUrls: ['./toolbar-buttons.component.scss']

})

**export** **class** ToolbarButtonsComponent {

@Input() selected: any[] = [];

@Input() entity: string = 'elemento';

@Output() **new** = **new** EventEmitter<**void**>();

@Output() edit = **new** EventEmitter<**void**>();

@Output() **delete** = **new** EventEmitter<**void**>();

}

toolbar-buttons.component.html

xml

<div class="toolbar-buttons">

<button

type="button"

class="ui-button ui-button--main"

(click)="new.emit()">

Nuevo {{ entity }}

</button>

<button

type="button"

class="ui-button"

[disabled]="!selected?.length || selected.length !== 1"

(click)="edit.emit()">

Editar

</button>

<button

type="button"

class="ui-button ui-button--danger"

[disabled]="!selected?.length"

(click)="delete.emit()">

Borrar

</button>

</div>

toolbar-buttons.component.scss (ejemplo BEM)

text

.toolbar-buttons {

display: flex;

gap: 1rem;

margin-bottom: 1.5rem;

.ui-button {

min-width: 7rem;

&--danger {

background: #d32f2f;

color: white;

}

&--main {

background: #1976d2;

color: white;

font-weight: bold;

}

}

}

**2.**Uso e integración en páginas administrativas

xml

<app-toolbar-buttons

[selected]="selectedItems"

[entity]="'país'"

(new)="onNew()"

(edit)="onEdit()"

(delete)="onDeleteSelected()">

</app-toolbar-buttons>

* Botón "Nuevo": Siempre activo
* Botón "Editar": Solo activo si hay 1 elemento seleccionado
* Botón "Borrar": Activo si hay 1 o más seleccionados
* Prop entity: Texto dinámico según el CRUD (país, área, lenguaje, continente...)

**3.**Ventajas de esta arquitectura

* Reutilizable para cualquier entidad (solo cambiando entity)
* Extensible: puedes añadir más acciones (copiar, exportar, importar...) fácilmente
* Compatible con cualquier sistema de selección (solo tienes que pasar el array de seleccionados)
* Claramente separada del resto del layout
* Estilos claros y compatibles con tu sistema UI (BEM + variantes)

¿Quieres el fragmento base para la integración con el resto de la página y flujo CRUD de countries, o quieres refinar algún aspecto de este componente antes de seguir?

**1.**Comando Angular CLI

Abre terminal en la ruta raíz del proyecto (donde está el angular.json):

bash

ng generate component modules/shared/components/toolbar-buttons

o, equivalente:

bash

ng g c modules/shared/components/toolbar-buttons

Esto crea:

* toolbar-buttons.component.ts
* toolbar-buttons.component.html
* toolbar-buttons.component.scss
* toolbar-buttons.component.spec.ts  
  en la carpeta indicada.

**2.**Importación y exportación

* El componente debe estar declarado y exportado en el módulo compartido (SharedModule).
* Ubica/ruta tu SharedModule, normalmente en:

text

src/app/modules/shared/shared.module.ts

En ese archivo, verifica que esté así:

typescript

**import** { ToolbarButtonsComponent } **from** './components/toolbar-buttons/toolbar-buttons.component';

@NgModule({

declarations: [

ToolbarButtonsComponent,

*// otros componentes compartidos...*

],

exports: [

ToolbarButtonsComponent,

*// otros componentes compartidos...*

],

imports: [

CommonModule,

*// otros módulos necesarios...*

]

})

**export** **class** SharedModule { }

**3.**Uso en cualquier página

Importa el SharedModule en el módulo de cada página que necesite la toolbar. Ejemplo:  
En el módulo de admin countries:

typescript

**import** { SharedModule } **from** '../../shared/shared.module';

@NgModule({

imports: [

SharedModule,

*// ...otros imports*

],

*// ...*

})

**export** **class** AdminCountriesModule { }

Ya podrás usar <app-toolbar-buttons> directamente en el HTML de tu página.

¿Quieres el código base completo para los archivos .ts/.html/.scss? Si lo deseas, te lo detallo para pegar directamente tras el comando.

Actualmente, tu archivo admin-countries.component.ts solo tiene la clase básica vacía.

Para integrar la toolbar-buttons y preparar el CRUD, debes añadir:

**1.**Variables de selección

typescript

selectedItems: any[] = [];

**2.**Handlers para los botones

typescript

onNew() {

*// Abre modal de alta país*

}

onEdit() {

*// Solo si selectedItems.length === 1, abre modal de edición país*

}

onDeleteSelected() {

*// Solo si selectedItems.length > 0, muestra confirm-dialog*

}

**3.**Ejemplo de clase final

typescript

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

@Component({

selector: 'app-admin-countries',

templateUrl: './admin-countries.component.html',

styleUrls: ['./admin-countries.component.scss']

})

**export** **class** AdminCountriesComponent {

selectedItems: any[] = [];

onNew() {

*// Abre modal alta*

*// this.showModal = true;*

*// this.editMode = false;*

}

onEdit() {

**if** (**this**.selectedItems.length === 1) {

*// Abre modal edición*

*// Cargar datos de país seleccionado*

}

}

onDeleteSelected() {

**if** (**this**.selectedItems.length) {

*// Muestra dialog confirmación*

*// this.showConfirm = true;*

}

}

}

**4.**Recuerda importar el ToolbarButtonsComponent

El ToolbarButtonsComponent estará disponible si tu módulo importa el SharedModule (ver respuesta anterior).

**5.**En el HTML del componente, añade algo así:

xml

<app-toolbar-buttons

[selected]="selectedItems"

[entity]="'país'"

(new)="onNew()"

(edit)="onEdit()"

(delete)="onDeleteSelected()">

</app-toolbar-buttons>

En resumen:

* Añade la propiedad selectedItems
* Añade los métodos onNew, onEdit, onDeleteSelected
* Asegúrate de importar el SharedModule en el módulo de la página

¿Quieres el fragmento HTML completo de la integración, o prefieres seguir con el código TypeScript/SCSS en este archivo?

Aquí tienes el código completo para el componente toolbar-buttons listo para Angular, siguiendo la arquitectura modular propuesta:

**1. toolbar-buttons.component.ts**

typescript

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

@Component({

selector: 'app-toolbar-buttons',

templateUrl: './toolbar-buttons.component.html',

styleUrls: ['./toolbar-buttons.component.scss']

})

**export** **class** ToolbarButtonsComponent {

@Input() selected: any[] = [];

@Input() entity: string = 'elemento';

@Output() **new** = **new** EventEmitter<**void**>();

@Output() edit = **new** EventEmitter<**void**>();

@Output() **delete** = **new** EventEmitter<**void**>();

}

**2. toolbar-buttons.component.html**

xml

<div class="toolbar-buttons">

<button

type="button"

class="ui-button ui-button--main"

(click)="new.emit()">

Nuevo {{ entity }}

</button>

<button

type="button"

class="ui-button"

[disabled]="!selected?.length || selected.length !== 1"

(click)="edit.emit()">

Editar

</button>

<button

type="button"

class="ui-button ui-button--danger"

[disabled]="!selected?.length"

(click)="delete.emit()">

Borrar

</button>

</div>

**3. toolbar-buttons.component.scss**

text

.toolbar-buttons {

display: flex;

gap: 1rem;

margin-bottom: 1.5rem;

.ui-button {

min-width: 7rem;

padding: 0.5rem 1.2rem;

font-size: 1rem;

border-radius: 6px;

border: none;

cursor: pointer;

transition: background 0.2s;

&--danger {

background: #d32f2f;

color: white;

font-weight: bold;

}

&--main {

background: #1976d2;

color: white;

font-weight: bold;

}

&:disabled {

background: #e0e0e0;

color: #aaa;

cursor: not-allowed;

}

}

}

**4. Importar y exportar en SharedModule:**

Asegúrate que en src/app/modules/shared/shared.module.ts esté así:

typescript

**import** { ToolbarButtonsComponent } **from** './components/toolbar-buttons/toolbar-buttons.component';

@NgModule({

declarations: [

ToolbarButtonsComponent,

*// otros...*

],

exports: [

ToolbarButtonsComponent,

*// otros...*

],

imports: [

CommonModule,

*// otros...*

]

})

**export** **class** SharedModule { }

**5. Importar SharedModule en tu módulo de página:**

Por ejemplo, en src/app/modules/admin/pages/countries/admin-countries.module.ts:

typescript

**import** { SharedModule } **from** '../../../shared/shared.module';

@NgModule({

imports: [

SharedModule,

*// otros...*

],

*// ...*

})

**export** **class** AdminCountriesModule { }

**6. Usar el componente en tu HTML:**

xml

<app-toolbar-buttons

[selected]="selectedItems"

[entity]="'país'"

(new)="onNew()"

(edit)="onEdit()"

(delete)="onDeleteSelected()">

</app-toolbar-buttons>

Listo para copiar, pegar y adaptar!  
¿Quieres que te pase el boilerplate para el resto de la página admin-countries o integración con la tabla y selección?

Aquí está el código completo del módulo para el componente toolbar-buttons en Angular:

**1.**toolbar-buttons.component.ts

typescript

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

@Component({

selector: 'app-toolbar-buttons',

templateUrl: './toolbar-buttons.component.html',

styleUrls: ['./toolbar-buttons.component.scss']

})

**export** **class** ToolbarButtonsComponent {

@Input() selected: any[] = [];

@Input() entity: string = 'elemento';

@Output() **new** = **new** EventEmitter<**void**>();

@Output() edit = **new** EventEmitter<**void**>();

@Output() **delete** = **new** EventEmitter<**void**>();

}

**2.**toolbar-buttons.component.html

xml

<div class="toolbar-buttons">

<button

type="button"

class="ui-button ui-button--main"

(click)="new.emit()">

Nuevo {{ entity }}

</button>

<button

type="button"

class="ui-button"

[disabled]="!selected.length || selected.length !== 1"

(click)="edit.emit()">

Editar

</button>

<button

type="button"

class="ui-button ui-button--danger"

[disabled]="!selected.length"

(click)="delete.emit()">

Borrar

</button>

</div>

**3.**toolbar-buttons.component.scss

text

.toolbar-buttons {

display: flex;

gap: 1rem;

margin-bottom: 1.5rem;

.ui-button {

min-width: 7rem;

padding: 0.5rem 1.2rem;

font-size: 1rem;

border-radius: 6px;

border: none;

cursor: pointer;

transition: background 0.2s;

&--danger {

background: #d32f2f;

color: white;

font-weight: bold;

}

&--main {

background: #1976d2;

color: white;

font-weight: bold;

}

&:disabled {

background: #e0e0e0;

color: #aaa;

cursor: not-allowed;

}

}

}

**4.**toolbar-buttons.component.spec.ts**(opcional test de Angular CLI)**

typescript

**import** { ComponentFixture, TestBed } **from** '@angular/core/testing';

**import** { ToolbarButtonsComponent } **from** './toolbar-buttons.component';

describe('ToolbarButtonsComponent', () => {

**let** component: ToolbarButtonsComponent;

**let** fixture: ComponentFixture<ToolbarButtonsComponent>;

beforeEach(**async** () => {

**await** TestBed.configureTestingModule({

declarations: [ToolbarButtonsComponent]

}).compileComponents();

fixture = TestBed.createComponent(ToolbarButtonsComponent);

component = fixture.componentInstance;

fixture.detectChanges();

});

it('should create', () => {

expect(component).toBeTruthy();

});

});

**5.**Declarar y exportar en SharedModule

Asegúrate que el componente esté en el archivo shared.module.ts:

typescript

**import** { ToolbarButtonsComponent } **from** './components/toolbar-buttons/toolbar-buttons.component';

@NgModule({

declarations: [

ToolbarButtonsComponent,

*// demás compartidos aquí*

],

exports: [

ToolbarButtonsComponent,

*// demás compartidos aquí*

],

imports: [

CommonModule,

*// demás módulos aquí*

]

})

**export** **class** SharedModule { }

**6.**Importa el SharedModule en el módulo de tu página

En admin-countries.module.ts o el módulo de tu página, añade:

typescript

**import** { SharedModule } **from** '../../shared/shared.module';

@NgModule({

imports: [

SharedModule,

*// demás*

],

})

**export** **class** AdminCountriesModule { }

Con esto, tu componente toolbar-buttons estará totalmente disponible y reutilizable en toda la app, escalable para cualquier entidad administrable.

¿Quieres la integración directa con la tabla y selección de registros, o necesitas otro fragmento del módulo?

Aquí tienes el componente bandera (flag) flexible, reutilizable y completamente preparado para tamaños y escala de grises:

**1.**flag-icon.component.ts

typescript

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

@Component({

selector: 'app-flag-icon',

templateUrl: './flag-icon.component.html',

styleUrls: ['./flag-icon.component.scss']

})

**export** **class** FlagIconComponent {

@Input() iso2: string; *// Ej: 'ES', 'FR'*

@Input() size: 'xs' | 's' | 'm' | 'l' | 'xl' | string = 'm'; *// string permite px/rem custom*

@Input() grayscale = false;

**get** sizeClass(): string {

**const** validSizes = ['xs','s','m','l','xl'];

**if** (validSizes.includes(**this**.size)) **return** `flag--${**this**.size}`;

**return** '';

}

**get** customSize(): string | **null** {

*// Si el size no es preset, se asume personalizado CSS.*

**const** validSizes = ['xs','s','m','l','xl'];

**return** validSizes.includes(**this**.size) ? **null** : **this**.size;

}

}

**2.**flag-icon.component.html

xml

*<!-- Usa png/svg/local sprite según tu sistema de flags. Ejemplo: https://hatscripts.github.io/circle-flags/flags/{iso2}.svg -->*

<span

class="flag"

[ngClass]="[sizeClass, grayscale ? 'flag--grayscale' : '']"

[ngStyle]="customSize ? {'width': customSize, 'height': customSize} : null"

[title]="iso2"

>

<img

\*ngIf="iso2"

[src]="'assets/flags/' + iso2.toLowerCase() + '.svg'"

[alt]="iso2"

[width]="customSize || null"

[height]="customSize || null"

loading="lazy"

/>

</span>

**3.**flag-icon.component.scss

text

.flag {

display: inline-block;

vertical-align: middle;

&--xs { width: 16px; height: 16px; }

&--s { width: 20px; height: 20px; }

&--m { width: 28px; height: 28px; }

&--l { width: 36px; height: 36px; }

&--xl { width: 48px; height: 48px; }

img {

display: block;

width: 100%;

height: 100%;

border-radius: 4px;

user-select: none;

pointer-events: none;

object-fit: cover;

filter: none;

transition: filter 0.12s;

}

&--grayscale img {

filter: grayscale(100%);

opacity: 0.7;

}

}

Uso en Angular:

xml

<app-flag-icon iso2="ES" size="m"></app-flag-icon>

<app-flag-icon iso2="FR" size="xl"></app-flag-icon>

<app-flag-icon iso2="DE" size="40px" grayscale="true"></app-flag-icon>

* Si pasas grayscale="true", la bandera aparece en escala de grises.
* El tamaño puede ser uno de los tokens corporativos (xs, s, m, l, xl) o un valor CSS (32px, 2rem, etc).

Adaptación a tu sistema:

* Cambia la ruta del src en el <img> si usas otro sistema/biblioteca de banderas.
* Si quieres el SVG inline, avísame y modelo la plantilla para inline SVG.

¿Necesitas tip tokens/tamaños UI concretos o descripción SCSS si tienes una guideline CSS corporativa específica?

**1.**Crea el componente con Angular CLI

Abre la terminal en la raíz del proyecto Angular y ejecuta:

bash

ng generate component modules/shared/components/flag-icon

O de forma abreviada:

bash

ng g c modules/shared/components/flag-icon

Esto crea la carpeta flag-icon con los archivos base.

**2.**Importar y exportar en el SharedModule

En src/app/modules/shared/shared.module.ts:

1. Importa el componente:

typescript

**import** { FlagIconComponent } **from** './components/flag-icon/flag-icon.component';

1. Agrégalo a declaraciones y a exports:

typescript

@NgModule({

declarations: [

FlagIconComponent,

*// ...otros*

],

exports: [

FlagIconComponent,

*// ...otros*

],

imports: [

CommonModule,

*// ...otros necesarios*

]

})

**export** **class** SharedModule { }

**3.**Importa SharedModule en todos los módulos principales

Haz esto en el módulo de la parte pública y del admin, por ejemplo:

**Para público:**

typescript

**import** { SharedModule } **from** 'src/app/modules/shared/shared.module';

@NgModule({

imports: [SharedModule, ...],

*// ...*

})

**export** **class** PublicModule { }

**Para privado (admin):**

typescript

**import** { SharedModule } **from** 'src/app/modules/shared/shared.module';

@NgModule({

imports: [SharedModule, ...],

*// ...*

})

**export** **class** AdminModule { }

**4.**Ahora puedes usar el componente bandera en cualquier template de la aplicación:

xml

<app-flag-icon iso2="ES" size="m"></app-flag-icon>

<app-flag-icon iso2="US" size="xl" [grayscale]="true"></app-flag-icon>

¿Quieres el código listo para los cuatro archivos generados (ts, html, scss, spec.ts), o solo el core del componente? Si tienes una guideline/scss de tamaño corporativo específica, pása el fragmento y adapto los tokens.

Actualiza el componente así:

flag-icon.component.html

xml

<span

class="flag"

[ngClass]="[sizeClass, grayscale ? 'flag--grayscale' : '']"

[ngStyle]="customSize ? {'width': customSize, 'height': customSize} : null"

[title]="iso2"

>

<img

\*ngIf="iso2"

[src]="'assets/icons/flags/flag-' + iso2.toLowerCase() + '.svg'"

[alt]="iso2"

[width]="customSize || null"

[height]="customSize || null"

loading="lazy"

/>

</span>

Comando para crearlo

En tu terminal (PowerShell) en la raíz del proyecto, ejecuta:

bash

ng generate component modules/shared/components/flag-icon

Para tenerlo disponible en toda la aplicación

En tu src/app/modules/shared/shared.module.ts, importa/declara/exporta:

typescript

**import** { FlagIconComponent } **from** './components/flag-icon/flag-icon.component';

@NgModule({

declarations: [

FlagIconComponent,

*// otros...*

],

exports: [

FlagIconComponent,

*// otros...*

],

imports: [

CommonModule,

*// otros...*

]

})

**export** **class** SharedModule { }

Luego en todos los módulos donde quieras usarlo (público y privado), importa tu SharedModule:

typescript

**import** { SharedModule } **from** 'src/app/modules/shared/shared.module';

@NgModule({

imports: [ SharedModule */\*, ... otros \*/* ]

})

**export** **class** AdminModule {} *// o PublicModule, CountriesModule, etc*

Ya puedes usarlo así en cualquier template:

xml

<app-flag-icon iso2="es" size="xl"></app-flag-icon>

<app-flag-icon iso2="fr" size="m" [grayscale]="true"></app-flag-icon>

<app-flag-icon iso2="de" size="32px"></app-flag-icon>

¿Quieres que te pase el bloque completo de flag-icon TS/HTML/SCSS para copiar-pegar? ¿Hay estilos corporativos/tokens de tamaño más concretos que quieras adaptar?

Resumen de tu flag-icon.component.ts

typescript

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

@Component({

selector: 'app-flag-icon',

templateUrl: './flag-icon.component.html',

styleUrls: ['./flag-icon.component.scss']

})

**export** **class** FlagIconComponent {

@Input() iso2: string = ''; *// Corregido!*

@Input() size: 'xs' | 's' | 'm' | 'l' | 'xl' | string = 'm';

@Input() grayscale = false;

**get** sizeClass(): string {

**const** validSizes = ['xs','s','m','l','xl'];

**if** (validSizes.includes(**this**.size)) **return** `flag--${**this**.size}`;

**return** '';

}

**get** customSize(): string | **null** {

**const** validSizes = ['xs','s','m','l','xl'];

**return** validSizes.includes(**this**.size) ? **null** : **this**.size;

}

}