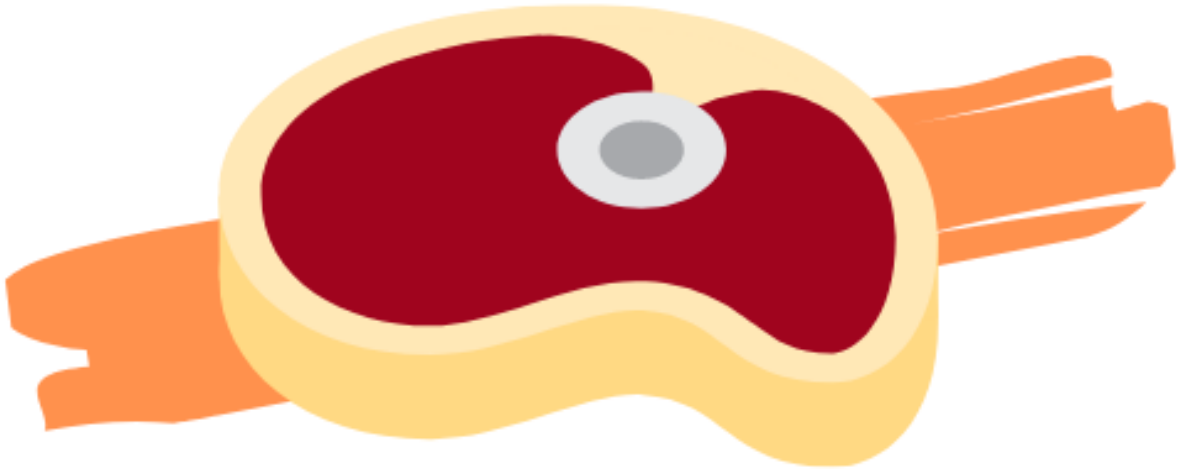

ANGULAR

Tabla de contenido

BOOTSTRAP 4

Opción 1: Bootstrap + Ng-bootstrap

1. **Dependencia Bootstrap:**
npm install bootstrap --save

2. **Importar el css de BS en styles.css:**
Añadir: @import "~bootstrap/dist/css/bootstrap.css";

3. **Instalación ng-bootstrap:**
npm install --save @ng-bootstrap/ng-bootstrap

4. **Añadir @angular/localize:**
ng add @angular/localize

5. **Añadir import de NgbModule en src/app/app.module.ts:**

```
import {NgbModule} from '@ng-bootstrap/ng-bootstrap';

@NgModule({
  // [...]
  imports: [NgbModule, /* [...] */],
  // [...]
})
export class AppModule {
}
```

Opción 2: Sólo Bootstrap CSS

1. **Dependencia Bootstrap:**
npm install bootstrap --save

2. **Importar el css de BS en styles.css:**
Añadir: @import "~bootstrap/dist/css/bootstrap.css";

FONT-AWESOME

1. Instalación
npm install font-awesome --save

2. Añadir a angular.json
"styles": [
"src/styles.css",
"node_modules/font-awesome/css/font-awesome.css"],

SINTAXIS PLANTILLAS

Template syntax	
`<input [value]="firstName">`	Binds property `value` to the result of expression `firstName`.
`<div [attr.role]="myAriaRole">`	Binds attribute `role` to the result of expression `myAriaRole`.
`<div [class.extra-sparkle]="isDelightful">`	Binds the presence of the CSS class `extra-sparkle` on the element to the truthiness of the expression `isDelightful`.
`<div [style.width.px]="mySize">`	Binds style property `width` to the result of expression `mySize` in pixels. Units are optional.

`<button (click)="readRainbow($event)">`	Calls method `readRainbow` when a click event is triggered on this button element (or its children) and passes in the event object.
`<div title="Hello {{ponyName}}">`	Binds a property to an interpolated string, for example, "Hello Seabiscuit". Equivalent to: `<div [title]="'Hello ' + ponyName">`
`<p>Hello {{ponyName}}</p>`	Binds text content to an interpolated string, for example, "Hello Seabiscuit".
`<my-cmp [(title)]="name">`	Sets up two-way data binding. Equivalent to: `<my-cmp [title]="name" (titleChange)="name=$event">`

`<video #movieplayer ...>`
`<button (click)="movieplayer.play()">`
`</video>`
	Creates a local variable `movieplayer` that provides access to the `video` element instance in data-binding and event-binding expressions in the current template.

`<p *myUnless="myExpression">...</p>`	The `*` symbol turns the current element into an embedded template. Equivalent to: `<ng-template [myUnless]="myExpression"><p>...</p></ng-template>`
`<p>Card No.: {{cardNumber | myCardNumberFormatter}}</p>`	Transforms the current value of expression `cardNumber` via the pipe called `myCardNumberFormatter`.
`<p>Employer: {{employer?.companyName}}</p>`	The safe navigation operator (`?`) means that the `employer` field is optional and if `undefined`, the rest of the expression should be ignored.
`<svg:rect x="0" y="0" width="100" height="100"/>`	An SVG snippet template needs an `svg:` prefix on its root element to disambiguate the SVG element from an HTML component.

`<svg>`
`<rect x="0" y="0" width="100" height="100"/>`
`</svg>`
	An `<svg>` root element is detected as an SVG element automatically, without the prefix.

NG-MODULES

NgModules	`import { NgModule } from '@angular/core';`
`@NgModule({ declarations: ..., imports: ..., exports: ..., providers: ..., bootstrap: ...}) class MyModule {}`	Defines a module that contains components, directives, pipes, and providers.
declarations: `[MyRedComponent, MyBlueComponent, MyDatePipe]`	List of components, directives, and pipes that belong to this module.
imports: `[BrowserModule, SomeOtherModule]`	List of modules to import into this module. Everything from the imported modules is available to `declarations` of this module.
exports: `[MyRedComponent, MyDatePipe]`	List of components, directives, and pipes visible to modules that import this module.
providers: `[MyService, { provide: ... }]`	List of dependency injection providers visible both to the contents of this module and to importers of this module.
entryComponents: `[SomeComponent, OtherComponent]`	List of components not referenced in any reachable template, for example dynamically created from code.
bootstrap: `[MyAppComponent]`	List of components to bootstrap when this module is bootstrapped.

COMPONENTES

Component configuration	`@Component` extends `@Directive`, so the `@Directive` configuration applies to components as well
moduleId: `module.id`	If set, the `templateUrl` and `styleUrl` are resolved relative to the component.
viewProviders: `[MyService, { provide: ... }]`	List of dependency injection providers scoped to this component's view.
template: `'Hello {{name}}'` **templateUrl:** `'my-component.html'`	Inline template or external template URL of the component's view.
styles: `['.primary {color: red}']` **styleUrls:** `['my-component.css']`	List of inline CSS styles or external stylesheet URLs for styling the component's view.

CICLO DE VIDA

Directive and component change detection and lifecycle hooks	(implemented as class methods)
`constructor(myService: MyService, ...) { ... }`	Called before any other lifecycle hook. Use it to inject dependencies, but avoid any serious work here.
`ngOnChanges(changeRecord) { ... }`	Called after every change to input properties and before processing content or child views.
`ngOnInit() { ... }`	Called after the constructor, initializing input properties, and the first call to `ngOnChanges`.
`ngDoCheck() { ... }`	Called every time that the input properties of a component or a directive are checked. Use it to extend change detection by performing a custom check.
`ngAfterContentInit() { ... }`	Called after `ngOnInit` when the component's or directive's content has been initialized.
`ngAfterContentChecked() { ... }`	Called after every check of the component's or directive's content.
`ngAfterViewInit() { ... }`	Called after `ngAfterContentInit` when the component's views and child views / the view that a directive is in has been initialized.
`ngAfterViewChecked() { ... }`	Called after every check of the component's views and child views / the view that a directive is in.
`ngOnDestroy() { ... }`	Called once, before the instance is destroyed.

DECORADORES DE CLASE

Class decorators	`import { Directive, ... } from '@angular/core';`
`@Component({...})` `class MyComponent() {}`	Declares that a class is a component and provides metadata about the component.
`@Directive({...})` `class MyDirective() {}`	Declares that a class is a directive and provides metadata about the directive.
`@Pipe({...})` `class MyPipe() {}`	Declares that a class is a pipe and provides metadata about the pipe.
`@Injectable()` `class MyService() {}`	Declares that a class can be provided and injected by other classes. Without this decorator, the compiler won't generate enough metadata to allow the class to be created properly when it's injected somewhere.

DECORADORES

Class field decorators for directives and components	`import { Input, ... } from '@angular/core';`
`@Input()` `myProperty;`	Declares an input property that you can update via property binding (example: `<my-cmp [myProperty]="someExpression">`).
`@Output()` `myEvent = new EventEmitter();`	Declares an output property that fires events that you can subscribe to with an event binding (example: `<my-cmp (myEvent)="doSomething()">`).

DIRECTIVAS

Built-in directives	`import { CommonModule } from '@angular/common';`
`<section *ngIf="showSection">`	Removes or recreates a portion of the DOM tree based on the `showSection` expression.
`<li *ngFor="let item of list">`	Turns the li element and its contents into a template, and uses that to instantiate a view for each item in list.

`<div [ngSwitch]="conditionExpression">`
`<ng-template [ngSwitchCase]="case1Exp">...</ng-template>`
`<ng-template ngSwitchCase="case2LiteralString">...</ng-template>`
`<ng-template ngSwitchDefault>...</ng-template>`
`</div>`
	Conditionally swaps the contents of the div by selecting one of the embedded templates based on the current value of `conditionExpression`.

`<div [ngClass]="{'active': isActive, 'disabled': isDisabled}">`	Binds the presence of CSS classes on the element to the truthiness of the associated map values. The right-hand expression should return {class-name: true/false} map.
`<div [ngStyle]="{'property': 'value'}">` `<div [ngStyle]="dynamicStyles()">`	Allows you to assign styles to an HTML element using CSS. You can use CSS directly, as in the first example, or you can call a method from the component.
Forms	`import { FormsModule } from '@angular/forms';`
`<input [(ngModel)]="userName">`	Provides two-way data-binding, parsing, and validation for form controls.

<ng-container> to the rescue

The Angular `<ng-container>` is a grouping element that doesn't interfere with styles or layout because Angular *doesn't put it in the DOM.*

Here's the conditional paragraph again, this time using `<ng-container>`.

src/app/app.component.html (ngif-ngcontainer)

```
<p>
  I turned the corner
  <ng-container *ngIf="hero">
    and saw {{hero.name}}. I waved
  </ng-container>
  and continued on my way.
</p>
```

Class binding with Class

There are another shorthand way to bind CSS Class to HTML element.

```
<div [class.<className>]="condition"></div>
```

Where

className is name of the class, which you want to bind to.

condition must return true or false. A return value of true adds the class and a false removes the class.

In the following example, the class `red` and `size20` is added to the `div` element.

```
<div [class.red]="true" [class.size20]="true">Test</div>
```

ROUTING & NAVEGACIÓN

Configuración

1. **Generar módulo AppRoutingmodule:**

ng generate module app-routing –flat

2. **Se puede borrar lo relativo al declarations de @NgModule y lo relativo al CommonModule.**
3. **Añadir el import de RouterModule y Routes:**

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

4. **Importar los componentes que queramos usar.**
5. **Añadir las rutas.**
6. **Inicializar el router:**

```
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
```

7. **Añadir <router-oulet> donde queramos que renderice.**

src > app > TS app-routing.module.ts > ...

```typescript
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';

// Componentes
import {HomeComponent} from './home/home.component';
import {AboutComponent} from './about/about.component';
import {ContactComponent} from './contact/contact.component';

// Rutas
const routes: Routes = [
  {path: '', redirectTo: 'home', pathMatch:'full'},
  {path: 'home', component: HomeComponent},
  {path: 'about', component: AboutComponent},
  {path: 'contact', component: ContactComponent},
  {path: 'contactus', redirectTo: 'Contact'},
  {path: '**', component: HomeComponent}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Sistema de navegación

1. **Unificar en carpeta UI.**
2. **Generar componente (navbar = 1 componente)**

ng generate module app-routing –flat

3. **Cambiar href por routerLink**
4. **Añadir routerLinkActive="active" a cada routerLink.**
5. **Añadir navbar a la plantilla.**

```
<nav class="navbar navbar-toggleable-md navbar-expand-lg navbar-light bg-faded navbar-inverse bg-primary">
    <button class="navbar-toggler navbar-toggler-right"
            type="button" data-toggle="collapse" data-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" routerLink="/home">Ejemplo de Routing</a>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
        <div class="navbar-nav">
            <a class="nav-item nav-link active" routerLink="/home" routerLinkActive="active">Home
                <span class="sr-only">(current)</span>
            </a>
            <a class="nav-item nav-link" routerLink="/about" routerLinkActive="active">Acerca de nosotros</a>
            <a class="nav-item nav-link" routerLink="/contact" routerLinkActive="active">Contacto</a>
        </div>
    </div>
</nav>
```

6. **Para que la hamburguesa funcione.**
   - Opción A: Con ng-bootstrap:

src > app > ui > navbar > TS navbar.component.ts > NavbarCompor

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

@Component({
  selector: 'app-navbar',
  templateUrl: './navbar.component.html',
  styleUrls: ['./navbar.component.css']
})
export class NavbarComponent implements OnInit {
  isCollapsed: boolean;

  constructor() {
    this.isCollapsed = true;
  }

  ngOnInit(): void {
  }

}
```

navbar.component.html > nav.navbar.navbar-toggleable-md.navbar-light.bg-faded.navbar-inverse.bg-primary > div#navbarNavAltMarku

```html
<nav class="navbar navbar-toggleable-md navbar-light bg-faded navbar-inverse bg-primary">
    <button class="navbar-toggler navbar-toggler-right"
            type="button" data-toggle="collapse" data-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation"
            (click)="isCollapsed = !isCollapsed">
        <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" routerLink="/home">Ejemplo de Routing</a>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup" [ngbCollapse]="isCollapsed">
        <div class="navbar-nav">
            <a class="nav-item nav-link active" routerLink="/home" routerLinkActive="active">Home
                <span class="sr-only">(current)</span>
            </a>
            <a class="nav-item nav-link" routerLink="/about" routerLinkActive="active">Acerca de nosotros</a>
            <a class="nav-item nav-link" routerLink="/contact" routerLinkActive="active">Contacto</a>
        </div>
    </div>
</nav>
```

- Opción B: Mediante TS:

```
c > app > ui > navbar > TS navbar.component.ts > NavbarCompone
1   import { Component, OnInit } from '@angular/core';
2
3   @Component({
4     selector: 'app-navbar',
5     templateUrl: './navbar.component.html',
6     styleUrls: ['./navbar.component.css']
7   })
8   export class NavbarComponent implements OnInit {
9     show:boolean = false;
10
11    constructor() {}
12
13    ngOnInit(): void {
14    }
15
16    toggleCollapse() {
17      this.show = !this.show;
18    }
19  }
```

```
<nav class="navbar navbar-toggleable-md navbar-light bg-faded navbar-inverse bg-primary">
    <button class="navbar-toggler navbar-toggler-right"
            type="button" data-toggle="collapse" data-target="#navbarNav"
            aria-controls="navbarNav" aria-expanded="false"
            aria-label="Toggle navigation"
            (click)="toggleCollapse()">
        <span class="navbar-toggler-icon"></span>
    </button>
    <a class="navbar-brand" routerLink="/home">Ejemplo de Routing</a>
    <div class="collapse navbar-collapse" id="navbarNavAltMarkup" [class.show]="show">
        <div class="navbar-nav">
            <a class="nav-item nav-link active" routerLink="/home" routerLinkActive="active">Home
                <span class="sr-only">(current)</span>
            </a>
            <a class="nav-item nav-link" routerLink="/about" routerLinkActive="active">Acerca de nosotros</a>
            <a class="nav-item nav-link" routerLink="/contact" routerLinkActive="active">Contacto</a>
        </div>
    </div>
</nav>
```

Routing con código

Con el servicio Router:

```ts
import { Component, OnInit } from '@angular/core';
import { Router } from '@angular/router';


@Component({
  selector: 'app-contact',
  templateUrl: './contact.component.html',
  styleUrls: ['./contact.component.css']
})
export class ContactComponent implements OnInit {

  constructor( private router: Router) {}

  ngOnInit(): void {
  }

  volverAInicio() {
    this.router.navigateByUrl('home');
  }

}

```

```html
<button (click)="volverAInicio()">Volver a inicio</button>
```

Routing con parámetros

/producto/12 → me mostraría el producto con id 12

/producto/cod_producto → me mostrará el producto con código "cod_producto".

```
{ path: 'product/:id', component: ProductComponent },
```

1. **Recibir parámetros:**

Inyectando el servicio ActivatedRoute.

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

```
export class ProductComponent {
    public id:string;

    constructor (private _route:ActivatedRoute) {
        this.id=_route.snapshot.paramMap.get('id');
```

Inyección del servicio

Este nombre debe coincidir con el del parámetro que hemos puesto en el array de rutas

Ir a la pantalla anterior

Con el servicio Location:

```ts
import { Location } from '@angular/common';
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-about',
  templateUrl: './about.component.html',
  styleUrls: ['./about.component.css']
})
export class AboutComponent implements OnInit {

  constructor(private location: Location) { }

  ngOnInit(): void {
  }

  volverAtras() {
    this.location.back();
  }
}
```

```html
<button (click)="volverAtras()">Volver atrás</button>
```

Redirigir con programación

1. **import { Router } from '@angular/router';**
2. **Inyectar el servicio Router:**
   constructor (private _router: Router) {}
3. **Llamar al método navigate() pasando array con la dirección y subdirección**

Ejemplos de llamadas a navigate:

```
this._router.navigate(['/empleado', idEmpleado]);

this._router.navigate(['/empleado','123','departamento', num]);
```

si num vale 3, me lleva a /empleado/123/departamento/3

```
this._router.navigate(['/home');
```

—

Routing and navigation	`import { Routes, RouterModule, ... } from '@angular/router';`

`const routes: Routes = [`
`{ path: '', component: HomeComponent },`
`{ path: 'path/:routeParam', component: MyComponent },`
`{ path: 'staticPath', component: ... },`
`{ path: '**', component: ... },`
`{ path: 'oldPath', redirectTo: '/staticPath' },`
`{ path: ..., component: ..., data: { message: 'Custom' } }`
`]);`
 	
`const routing = RouterModule.forRoot(routes);`
	Configures routes for the application. Supports static, parameterized, redirect, and wildcard routes. Also supports custom route data and resolve.

`<router-outlet></router-outlet>` `<router-outlet name="aux"></router-outlet>`	Marks the location to load the component of the active route.

`<a routerLink="/path">`
`<a [routerLink]="[ '/path', routeParam ]">`
`<a [routerLink]="[ '/path', { matrixParam: 'value' } ]">`
`<a [routerLink]="[ '/path' ]" [queryParams]="{ page: 1 }">`
`<a [routerLink]="[ '/path' ]" fragment="anchor">`
	Creates a link to a different view based on a route instruction consisting of a route path, required and optional parameters, query parameters, and a fragment. To navigate to a root route, use the `/` prefix; for a child route, use the `./`prefix; for a sibling or parent, use the `../` prefix.

`<a [routerLink]="[ '/path' ]" routerLinkActive="active">`	The provided classes are added to the element when the `routerLink` becomes the current active route.

SERVICIOS

Configuración

- Servicio = lógica de negocio && acceso a datos.
- Es una clase con el decorador @Injectable.
- Una única instancia compartida(Singleton).
- Cada clase que lo quiera usar lo inyecta en su constructor.
- Se pueden inyectar varios servicios.
- Un servicio puede ser inyectado dentro de otro.

1. Crear un servicio

ng generate service nombreServicio

2. Registrar el proveedor en app.module:

```
  ],
  providers: [UserServiceService],
  bootstrap: [AppComponent]
})
```

3. Inyectarlo donde lo queramos usar:

```
constructor(private userService: UserServiceService) {

}

ngOnInit(): void {
  this.usuarios = this.userService.getAllUsers();
}

borrarUsuario(user: User) {
  this.userService.deleteUser(user.id);
}
```

Dependency injection configuration	
`{ provide: MyService, useClass: MyMockService }`	Sets or overrides the provider for `MyService` to the `MyMockService` class.
`{ provide: MyService, useFactory: myFactory }`	Sets or overrides the provider for `MyService` to the `myFactory` factory function.
`{ provide: MyValue, useValue: 41 }`	Sets or overrides the provider for `MyValue` to the value `41`.