

Curso de Angular 14

Enrique de la Calle Santa Ana

14 Marzo 2,024

Índice

1. Introducción a Angular
2. Angular Cli
3. Componentes
4. Directivas
5. Formularios Reactivos
6. Servicios
7. Peticiones HTTP

Índice

- 8. Routing
- 9. Pipes
- 10. Estado de la Aplicación
- 11. Expecificidad CSS , BEM , Animaciones y preprocesador SASS
- 12. Testing
- 13. Builds y Despliegue
- 14. Proyecto Final

01

Introducción a Angular

Introduccion Angular

- **Introducción:**
- <https://docs.angular.lat/>
- Ventajas y desventajas
- Novedades sobre A14
- Creación de entorno trabajo
- <https://angular.io/guide/setup-local>
- Creamos nuestra primera App
- Analizamos los ficheros

Revisión de puntos

- ☐ ¿Por qué Angular?
- ☐ Ventajas y desventajas principales
- ☐ Características de Angular 14.2
- ☐ Diferencias más destacables frente a las versiones actuales de Angular
- ☐ ¿Qué es el renderizador de Angular?
- ☐ Instalación de Angular CLI a través de NPM
- ☐ Diferenciando entre instalaciones locales y globales de NPM
- ☐ ¿Qué es NPX y por qué usarlo?
- ☐ Creación de tu primer proyecto Angular ☐

Revisión de puntos

- ☐ Análisis de la estructura del proyecto y sus archivos de configuración
- ☐ Desplegando nuestra aplicación localmente con Angular CLI

02

Comandos Ng-cli

Revisión de puntos

- ☐ Introducción Comandos CLI esenciales
- ☐ ng new
- ☐ ng serve
- ☐ ng generate
- ☐ ng add
- ☐ ng build
- ☐ ng update
- ☐ Otros comandos existentes

03

Componentes / Material

Links Interés

- <https://codingpotions.com/angular-componentes>
- <https://ngchallenges.gitbook.io/project/componentes>
- <https://codingpotions.com/angular-comunicacion-componentes/>
- <https://platzi.com/clases/2486-angular-componentes/41180-ciclo-de-vida-de-componentes/#:~:text=Ciclo%20de%20vida%20en%20Angular,los%20inputs%20en%20todo%20momento>
- <https://material.angular.io/guide/getting-started>

Componentes

- Componente es un decorador Typescript (personaliza visual)
- Componente como unidad visual mínima en Angular
- Respeta MVC (template = vista = html, css) , (Modelo = prop clase) , (Controlador = Typescript o clase)
- Metadatos en un componente (selector , templateUrl , styleUrls)
- Recomendación:
- las Clases CamelCase y terminadas en Component (Si es componente)
- El selector en Minuscula y separado por guiones (html no dife may)

Componentes

- Creación Componente
- Se puede generar a mano creando todos los ficheros o
- Ng generate component components/contador
- Revisamos los ficheros generados
- Enlace a datos
- {{cliente.nombre}} interpolación solo en un sentido
- [property] = “value” Propiedad o atributo
- (event) = “handler” Evento o controlador
- [(ng-model)] = “property” propiedad (two binding)

Practica Componentes

- Creamos un proyecto nuevo Componentes / No Router / CSS
- C:\Ang14\npx ng new 03_Componentes
- C:\Ang14\npx generate component components/contador
- Revisamos la estructura y vemos como app.module.ts se ajusta
- Vemos la jerarquía de estilos css en cada parte de la app
- Incorporo en la pagina app.component.html el contador (varios fijos)
- Le paso el atributo mediante [] en varios contadores
- `@Input() titulo:string = '';`

Practica Componentes

- En app.component.ts incorporamos un array de contadores

```
interface ContadorInterface {  
  id:number,  
  title:string,  
  inicio:number,  
  valor:number  
}  
  
aContadores:ContadorInterface[] = [  
  {id:1, title:"Contador 1", inicio: 0 },  
  {id:2, title:"Contador 2", inicio: 5 },  
  {id:3, title:"Contador 3", inicio: 4 },  
]
```

Practica Componentes

- Quiero ver el array de contadores en la app (introduzco el concepto pipe | json)
- `<p>Prueba {{ aContadores | json }}</p>`
- Adapto la estructura con un bucle json
- `<table border="1">`
- `<thead>`
- `<tr>`
- `<th *ngFor="let contador of aContadores" >{{contador.title}}</th>`
- `</tr>`
- `</thead>`
- `<tbody>`
- `<tr>`
- `<td *ngFor="let contador of aContadores" ><app-contador></app-contador></td>`
- `</tr>`
- `</tbody>`
- `</table>`

Practica Componentes

- He introducido el concepto directiva de estructura (*ngFor="let contador of aContadores ")
- Directiva de Atributo (modifican las características de un componente como title ,etc [])
- Directivas de estructura (modifican el dom *ngFor ,etc)
- Directivas de plantilla que son los componentes en si
- Le pasamos como directivas de atributo , ID , title , contador y presentamos en el contador su valor
- Incorporamos un timer: (podríamos hacer un timerinterval de JS pero aprovechamos y ponemos)
- `import { interval, Observable, Subscription } from 'rxjs';`
- `const unseg:Observable<number> = interval(1000);`
- Adelantamos la comunicación Bidireccional con ngModel
- `app.module: import { FormsModule } from '@angular/forms'`
- `App.component: <input [ngModel]="currentItem.name" (ngModelChange)="currentItem.name=$event" id="example-change">`

Codigo contador.component.ts

- `import { Component, Input, Output, OnChanges, OnInit, OnDestroy, SimpleChanges, EventEmitter } from '@angular/core';`
- `import { interval, Observable, Subscription } from 'rxjs';`
- `const unseg:Observable<number> = interval(1000);`
- `@Component({ selector: 'app-contador', templateUrl: './contador.component.html', styleUrls: ['./contador.component.css']})`
- `export class ContadorComponent implements OnInit, OnDestroy, OnChanges {`
- `@Input() id:number = 0; @Input() valor:number = 0; @Input() parentMessage:string = "";`
- `@Output() messageEvent = new EventEmitter<string>();`
- `obs?: Subscription;`
- `constructor() {`
- `this.obs = unseg.subscribe(x=>{`
- `console.log("temporizador un seg", x)`
- `this.valor++;`
- `this.messageEvent.emit(`Papa: soy ${this.id} con el valor ${this.valor} `);`
- `})`
- `}`

Codigo contador.component.ts

```
○   ngOnInit(): void {  
○       console.log("init elemento")  
○   }  
○   ngOnDestroy() {  
○       console.log("destruyo elemento")  
○       this.obs?.unsubscribe();  
○   }  
○   ngOnChanges(changes: SimpleChanges) {  
○       console.log("cammbios ", changes)  
○   }  
○   ngDoCheck(){  
○       //console.log("do check")  
○   }  
○   }
```

Ciclo de Vida del componente

<https://platzi.com/clases/2486-angular-componentes/41180-ciclo-de-vida-de-componentes/#:~:text=Ciclo%20de%20vida%20en%20Angular,los%20inputs%20en%20todo%20momento>

- constructor
- ngOnChanges
- ngOnInit
- ngDoCheck
- ngAfterContentInit
- ngAfterContentChecked
- ngAfterViewInit
-
- ngOnDestroy

Revisión de puntos

- **3. COMPONENTES**

- ☐ Metadatos de componentes
- ☐ Creación de un componente
- ☐ Instanciando componentes en archivos HTML
- ☐ Introducción al ngModel
- ☐ Data binding
- ☐ Operador de coalescencia nula
- ☐ Anidado de componentes
- ☐ Pasando datos al componente a través de @Inputs
- ☐ Respondiendo a eventos con @Outputs
- ☐ Ciclo de Vida de los componentes
- ☐ ¿Cuándo usar el ciclo de vida de los componentes en aplicaciones reales?
- ☐ Aplicando estilos a los componentes desde una hoja CSS o SCSS

03 (Bis)

Material

Angular Material

Librería Grafica (Google) para incorporar a nuestros Proyectos

Alternativas: BootStrap , PrimeNg , etc

- Mkdir c:\Ang14\demo_material
- Cd , copio proto0 y npm i
- Npx ng add @angular/material@14 (Trampa Aviso)
- Revisamos app.module

```
• import { MatSlideToggleModule } from '@angular/material/slide-toggle';  
•  
•     @NgModule ({  
•         imports: [  
•             MatSlideToggleModule,  
•         ]  
•     })  
•  
•     class AppModule {}
```

Angular Material (Cont. Y Schematics)

- `App.component.html`
- `<mat-slide-toggle>Toggle me!</mat-slide-toggle>`
- Angular Material tiene distintos templates y macros
- Un ejemplo seria un marco de navegación
- `ng generate @angular/material:navigation navega`
- (Si error versión)
- `"npx ng update @angular/material@14"`

Revisión de puntos

- **3. COMPONENTES**

- ☐ Introducción a Angular Material como framework de componentes
- ☐ Breve introducción a los componentes más destacables de Angular Material
- ☐ Instalación y configuración de Angular Material en un proyecto Angular
- ☐ ¿Qué son los schematics?
- ☐ Comentarios acerca de los schematics de Angular Material
- ☐ Empleando componentes de Angular Material en un proyecto Angular

04

Directivas/Modulos

Links

- <https://medium.com/notasdeangular/directivas-en-angular-efb8a8cf78e0> (Directivas)
- <https://codigoencasa.com/que-son-las-directivas-en-angular/> (Directivas)
- <https://www.freecodecamp.org/espanol/news/como-usar-y-crear-directivas-personalizadas-en-angular/#:~:text=%C2%BFQu%C3%A9%20es%20una%20directiva%20angular,adjuntar%20comportamientos%20personalizados%20al%20DOM>
- <https://www.tutorialesprogramacionya.com/angularya/detalleconcepto.php?punto=14&codigo=14&inicio=0> (Módulos)
- <https://www.youtube.com/watch?v=KzTAvLxN60U> (Módulos)
- <https://angular.io/guide/file-structure> (Estructura carpetas y multiproyectos)
- <https://material.angular.io/guide/getting-started> (Instalación Angular Material)
- <https://ng-bootstrap.github.io/#/getting-started> (Instalacion Bootstrap)

Directivas

- Directivas son funciones que el renderizador ejecutara cuando las encuentre
- Las directivas pueden ser de angular o definidas por nosotros
- Directivas de atributo
- Manipulan la apariencia y el comportamiento (p.e. `ngClass` , `ngStyle` , `ngModel`)
- Directivas estructurales

Cambian la estructura del DOM , bucles , condicionales , etc

Comienzan siempre por un asterisco (`*NgIf` , `* NgFor` , etc)

- Directivas de componente (Componentes)

Es una directiva que proporciona el factor visual de nuestra clase/objeto

```
@Component({selector: 'app-contador', templateUrl:
'./contador.component.html', styleUrls: ['./contador.component.css']
})
export class ContadorComponent implements OnInit, OnDestroy, OnChanges {}
```

Directivas

- *ngIf
- *ngFor
- ngSwitch / * ngSwitchCase
- *ngPlural
- ngTemplate
- ngComponentOutlet

Directivas

- Creación de una directiva
- ng generate directive

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

@Directive({ selector: '[Mifondo]' })
export class MifondoDirective {
  constructor(private elementRef: ElementRef) {
    elementRef.nativeElement.style.background = 'red';
  }
}
```

- Pasar Parámetros a Directiva
- Añade @ Input() en la clase directiva con el mismo nombre que la directiva (@Input() highlight;) y pasar el valor así <p highlight="blue">Highlight Directive</p>
- Añade @ Input() en la clase directiva con cualquier nombre de variable (@Input() colorName;) y pasar el valor así <p highlight="blue" colorName="green">Highlight Directive</p>

Revisión de puntos

- **4. DIRECTIVAS**

- ☐ Directivas de atributo
- ☐ Buenas prácticas en el uso de directivas de atributo
- ☐ Directivas estructurales (condicionales y bucles)
- ☐ Creación de directivas personalizadas
- ☐ Buenas prácticas en el uso de directivas estructurales

Modulos

- <https://www.tutorialesprogramacionya.com/angularya/detalleconcepto.php?punto=14&codigo=14&inicio=0>
- Modularizar , librerías , etc. Lo veremos mas en profundidad en ejerc.
- Agrupamiento de recursos en un modulo
- `npx ng generate module mimodulo`
- Genera una carpeta mimodulo con los ficheros correspondientes
- Para generar un componente en ese modulo
- `npx ng generate component mimodulo/contador`

Revisión de puntos

- **3. (BIS) MODULOS**
 - ☐ ¿Qué son los módulos?
 - ☐ La organización de un proyecto mediante módulos
 - ☐ Creación de módulos en un proyecto Angular
 - ☐ Inyección de dependencias en Angular
 - ☐ Creación de una plantilla HTML inicial

05

Formularios Reactivos

05 Forms y Reactive Forms

- Definición
 - Un formulario es un control o conjunto de controles que esperan la interacción del usuario
 - Los formularios en Angular están contruidos sobre el formulario HTML estándar, para ayudarte a crear controladores personalizados y simplificar la experiencia de validación de campos
- Pueden ser de dos tipos
 - Basado en plantillas
 - Son los basados principalmente en los controles del HTML
 - Usamos la directiva `ngModel` para enlazar bidireccionalmente
 - TIPS:
 - Si lo encapsulamos en un componente standard form hay que detallar “name”.
 - No olvidar `import { FormsModule } from '@angular/forms'` en `app.module`
 - Ejercicio 1
 - Incorporamos un simple input con la directiva `ngModel` , vemos como el dato se transmite en ambas direcciones

05 Forms y Reactive Forms

- Reactivos
 - Mas robusto, mas escalable , incorporamos funcionalidades de validación
 - Incorpora componentes nuevos como FormControl , FormGroup , Validación , etc que nos ayuda a tener un mayor control sobre el formulario
 - TIPS:
 - No olvidar `import { ReactiveFormsModule } from '@angular/forms'` en `app.module`
 - Ejercicio 2
 - Igual que el ejercicio 1 pero demostrando que tiene mas funcionalidad al usar el componente `FormControl`
- Con reactive Forms podemos agrupar controles
 - Un solo control es fácilmente gestionable con `FormControl`, pero quizás nos interese tener un agrupamiento de controles
 - Los agrupamos y tenemos un objeto `values` con toda la info
 - Ejercicio 3
 - Grupo componentes e incluyo `select` , `check` , `radio`

05 Forms y Reactive Forms

- Validaciones
 - Reactive Forms tiene componentes de validación llamado “Validators”
 - Se incluyen en el FormControl para verificar su contenido
 - Validación simple como un require o un email en Ejercicio 4
 - Se pueden incluir más de una condición en un array de Validators . p.e. required y mayor que 10 , etc
 - Hay múltiples validadores
 - Valor Requerido
 - Numéricos , Max , Min,
 - Texto: formato email , min long , max long , pattern(Expresión regular)
 - Se pueden agrupar en un Validator.compose
 - Ejercicio 5
 - Validaciones múltiples
 - Aprovecho el ciclo de vida para que si cambia un dato , rechequee la validación
 - Utilizo un compose con pattern para solo números y varias mas

Revisión de puntos

- **5. FORMULARIOS REACTIVOS**

- ☐ Introducción a los Formularios Reactivos
- ☐ Creación de Formularios Reactivos
- ☐ Introducción a la validación de campos en formularios reactivos
- ☐ Introducción a FormControl y FormGroup
- ☐ Validando campos obligatorios
- ☐ Validando campos numéricos
- ☐ Validando campos booleanos
- ☐ Introducción a FormArray
- ☐ Validando campos de tipo lista

Revisión de puntos

- **5. FORMULARIOS REACTIVOS**

- ☐ Validando expresiones regulares
- ☐ Accediendo a los errores
- ☐ Mostrar mensajes de error de validación
- ☐ Anidación de validaciones
- ☐ Validando que dos campos sean iguales
- ☐ Comprobando el estado del formulario
- ☐ Accediendo al contenido del formulario
- ☐ Creación de un formulario de Login reactivo
- ☐ Creación de un formulario de Registro reactivo
- ☐ ¿Qué son los formularios estrictamente tipados?
- ☐ Haciendo uso de los formularios tipados

06

Servicios

Links Servicios

- <https://imaginaformacion.com/tutoriales/servicios-llamadas-api-angular>
- <https://desarrolloweb.com/articulos/servicios-angular.html>
- <https://codigoencasa.com/angular-promesas-vs-observables-elige-tu-destino/>
- <https://codingpotions.com/angular-servicios-llamadas-http/>

Servicios

- Definición
 - Un servicio es un **Decorador** de Angular que mantiene la lógica de acceso a los datos , los servicios serán consumidos por los componentes , y estos delegaran en ellos la responsabilidad de acceder a la información y a la realización de operaciones con los datos
 - Dicho de otra manera , un servicio es el responsable de acceder y gestionar los datos , encapsulando su proceder
 - Un servicio se crea en Angular 14 con el siguiente comando
 - `ng generate service path/servicio`
 - Ejemplos de servicios
 - Servicio de acceso a datos de un cliente
 - Servicio de Api /REST con datos remotos
 - Servicio de Autenticacion , etc
- Un servicio puede hacer llamadas a terceros y estos provocar cierta latencia , por ello los servicios se pueden consumir con diferentes tipos de llamadas
 - Servicios simples (solo accedo a datos de memoria de mi app)
 - Servicios HTTP (peticiones cliente http)

Creamos un servicio simple

- Objetivo: Tenemos un array con los alumnos , queremos consumir un elemento lista con esos datos
 - Primero creamos el componente lista
 - `npx ng generate component components/lista`
 - Después creamos el servicio curso
 - `npx ng generate service services/curso`
- Vemos en app.modules.ts los cambios
 - Importa la lista y el servicio
 - `import { ListaComponent } from './components/lista/lista.component';`
 - `import { CursoService } from './services/curso.service';`
 - En la marca providers inyecta el servicio
 - `providers: [CursoService],`
- Analizamos curso.service.ts
 - `import { Injectable } from '@angular/core';`
 - `@Injectable({ providedIn: 'root' })`
 - El resto como si fuera una clase cualquiera con sus métodos



07

Peticiones HTTP

Peticiones HTTP

- Antes de explicar que es una petición necesitamos una herramienta local
 - Su nombre es json-server y sirve para poner un servidor de datos json
 - Se instala con `npm install -D json-server`
 - Creamos una carpeta server y disponemos de un fichero json
 - Lo hacemos correr con
 - `npx json-server server/db.json -p 3000`
 - Eso abre un servidor web en el Puerto determinado
 - Lo confirmamos con <http://localhost:3000>
 - Incorporamos el fichero db.json demo

Peticiones HTTP

- Una petición http es una llamada a un recurso mediante el protocolo http
- Puesto que el objetivo principal es el transito de datos , también se llama servicios http
- Requisitos necesarios para trabajar con servicios http
 - Inclusion en app.module
 - `import { HttpClientModule } from "@angular/common/http";`
 - Inclusion en el servicio
 - `import { HttpClient } from "@angular/common/http";`
 - E inyectar en el constructor del servicio
 - `constructor(private http: HttpClient) {}`

Peticiones HTTP

- La mejor manera de explicarlo es remplazar el servicio simple de loteria por un servicio api REST
- Copiamos el proyecto y renombramos la carpeta
- 06_Servicios_Practica1 => 06_Servicios_http
- Cambios en app.module
 - import { HttpClientModule } from "@angular/common/http";
 - y ponerlo en la seccion imports
- Cambios en loterias.service
 - Creamos una constante para la url (no es necesario)
 - const URL_DB = "http://localhost:3000";
 - vaciamos participantes a [] , porque haremos una function loadData
 - loadData(){
 - this.http.get(URL_DB+'/participantes').subscribe((data) => {
 - this.participantes = data ;
 - });
 - }

Peticiones HTTP

- Incluimos ese load en el constructor
- constructor(private http: HttpClient) {
- this.loadData();
- }
- El método addParticipante cambia un poco ya que mandamos un método post y en su finalización , provocamos el refresco
- addParticipante(name:string){
- let rec = {
- id:this.getMaxId().toString(),
- nombre:name,
- ganados:0
- }
- this.http.post(URL_DB+'/participantes', rec).subscribe((data)=>{
- this.loadData();
- })
-
- }

08

Routing

Routing

- Links
 - <https://www.techiediaries.com/routing-angular-router/>
- Definición
 - El enrutado permite la navegacion y el movimiento entre los distintos componentes de nuestra app
- Para iniciarlo podemos decir que si en el wizard del new
 - `npx ng new 08_routing`
 - ? Would you like to add Angular routing? (y/N)
 - CSS ok
- Revisamos nuestra app y vemos que nos ha creado un fichero `app.routing.module.ts`
 - En su contenido pondremos las rutas de nuestra app
- En nuestro core (`app.module.ts`) , ha importado ese fichero

Routing

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';

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

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Routing

- También observamos que en la parte visual (app.component.html) , incorpora un componente
 - `<router-outlet></router-outlet>`
- Con esos cambios ya podemos incorporar nuestros componentes creados con `npx ng generate component` a nuestra app
- Un ejemplo de como quedaria el fichero de enrutado, (se parte de que los componentes ya han sido creados)
 - ```
const routes: Routes = [
 { path: '', component: AboutComponent },
 { path: 'about', component: AboutComponent },
 { path: 'contact', component: ContactComponent },
 { path: 'home', component: HomeComponent },
 { path: 'news', component: NewsComponent },
];
```
- Paso de parametros:
  - Si incorporamos los dos puntos en el atributo de routes entendera que es un parametro
  - `{path:'lista2/:id', component:Lista2Component},`
  - Para consumir el uso de parametros , necesitamos inyectar el servicio de routing al componente
    - `import { ActivatedRoute } from "@angular/router";`
    - Y en el constructor:  

```
constructor(private route: ActivatedRoute) {}
```
    - De esta manera ya tenemos acceso a la variable route

# Routing

- Si quisieramos almacenar en una variable nuestra

```
export class Lista2Component {
 myParams:any;
 constructor(private route: ActivatedRoute) {
 this.route.params.subscribe(params => this.myParams=params);
 }
}
```

- Lo comprueba en nuestro html
- `<p>Los parametros son {{ myParams | json }}</p>`

# Routing

- Los Guards son “interceptores” o Guardianes que escuchan los cambios de url para verificar si se tiene acceso o no a ellos
  - Se generan con `npx ng generate guard xxxxx`
  - En el código de guard se programa si el usuario puede acceder a esa url
- Los Resolvers en el enrutado son parecidos a los guards
  - Cuando navegas a la ruta , el resolve se ejecuta y devuelve los datos que estan en la clase

```
const routes: Routes = [
 { path: 'noticias', component: NoticiasComponent, resolve: { data: NewsResolver } },
];
```

Cuando nos subscribimos al servicio de enrutado podemos ver los cambios que ha hecho el resolve ( como los parametros de antes )

```
this.route.data.subscribe((data: { data: string }) => {
 console.log(data.data); // ¡Datos resueltos!
});
```

# Routing

## EJERCICIO

- Proyecto 14
- Material Design
- Con el schematics de Navegacion
- Tres componentes
  - Main
  - Login
  - Register
  -

`npx ng add @angular/material@14`

`npx ng generate @angular/material:navigation <component-name>`  
de `app.component.html` quitamos todo menos el router y ponemos `<app-navega>`

# 09

## Pipes



# Pipes

- Definición
  - Un pipe o “tubería”, son transformadores que se utilizan para devolver un valor manipulado
    - Estos pipes pueden ser para conversión de fechas , números , etc
- Podemos definir nuestros propios pipes
- `ng generate pipe pipes/elevado`

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

@Pipe({
 name: 'elevado'
})
export class ElevadoPipe implements PipeTransform {

 transform(value: unknown, ...args: unknown[]): unknown {
 return Math.pow (value as number , args[0] as number)
 }
}
```

**NOTA: Revisar el `app.module.ts` para su importación**

# 10

## Estado de la Aplicación

# Estado de la Aplicación

- Links:
- <https://alejandrora.medium.com/aprender-angular-ngrx-c33a99fcbc4a>
- <https://digital55.com/blog/estructura-basica-store-ngrx/>
- <https://codigoencasa.com/ngrx-buenas-practicas/>
- El Estado de aplicación centraliza toda la información global de la app , evitando acceder y actualizar las variables directamente
- Se incorpora el concepto de store o almacén donde se unifica toda la info de la app
- Gracias a esa centralización , de puede “custodiar” el estado de una app en un determinado momento para hacer restauraciones por ejemplo
- También esta indicado para hacer seguimientos de cambio de estados o ficheros logs o seguridad al acceso al dato
- Estado Inicial del store
  - Son los valores iniciales del store , pueden ser constantes o llamar a rutinas de carga de estos via fichero o remotos
- Acciones
  - Son las peticiones que hacemos a los datos , para su actualizacion , creacion o borrado del estado de la app
- Reductores
  - Controlan las mutaciones de estos datos y generan los nuevos estados
- Vemos el ejemplo en 14\_proyecto\_v2
- NOTA: No Olvidar : `npm install @ngrx/store — save`

# 11

## Especificidad CSS

# CSS

- Ya hemos visto la piramide CSS
  - Styles.css en el root
  - App.component.css en el main
  - Components/elem.css en el componente
- BEM ( Bloque – Elemento – Modificador )
  - Es una metodologia de nomenclatura del CSS
  - Aunque Angular permite encapsular por app / componente, BEM es util para aplicar estilos especificos
  - Supongamos que tenemos un componente product-card

# CSS

```
<!-- product-card.component.html -->
<div class="product-card">
 <img [src]="product.imageUrl"
 class="product-card__image"
 >
 <h3 class="product-card__title">{{ product.name }}</h3>
 <p class="product-card__price">
 {{ product.price | currency }}
 </p>
</div>
```

# CSS

- Y su Hoja de estilo

```
/* product-card.component.scss */
```

```
.product-card {
```

```
 border: 1px solid #ddd;
```

```
 padding: 16px;
```

```
 background-color: #f9f9f9;
```

```
 &__image {
```

```
 max-width: 100%;
```

```
 height: auto;
```

```
 display: block;
```

```
 margin-bottom: 12px;
```

```
 }
```

# CSS

```
&__title {
 font-size: 18px;
 font-weight: bold;
 margin-bottom: 8px;
}

&__active {color: brown;}

&__price {
 font-size: 16px;
 color: #007bff;
}
}
```

- El modificador iría concatenado a la clase

product-card\_\_title\_\_active



# CSS / Animaciones

- Links
- <https://docs.angular.lat/guide/animations>
- Rotate:
- <https://stackoverflow.com/questions/44330983/angular-animation-rotation-180-click-image>
- SCSS
- <https://www.geeksforgeeks.org/how-do-you-create-application-to-use-scss/#:~:text=Method%20%3A%20Use%20or%20configure,as%20the%20default%20styling%20language.>
- Las animaciones se ubican en la definición de la clase en la propiedad animatio
- Ver ejemplo 11\_scss ( Nuevo proyecto con hoja de estilo scss )

# 12

## Testing

# Documentación

- **Documentación**

- La propiedad de dejar documento y bien claro nuestro código
- Los beneficios que aporta son claridad, disponibilidad para otros programadores, etc
- Hay varias herramientas para ello
  - **JSDoc** (<https://jsdoc.app/>)
  - **ESDoc** (<https://esdoc.org/>)
  - **documentJs** (<https://documentjs.com/>)
  - **Compodoc** (<https://compodoc.app/>)

- **Prácticas con Compodoc**

- En cualquier proyecto, podemos incorporar la documentación
- `npm i --save-dev @compodoc/compodoc`
- Se puede usar el fichero de configuración standard ( `tsconfig.app.json` ) pero preferimos montar el nuestro

# Documentación

```
config.doc.json
{
 "extends": "./tsconfig.json",
 "compilerOptions": {
 "outDir": "./out-tsc/app",
 "types": []
 },
 "files": [
 "src/main.ts",
 "src/polyfills.ts"
],
 "include": [
 "src/**/*.d.ts",
 "src/**/*.ts"
]
}
```

# Documentacion

```
config.doc.json
{
 "extends": "./tsconfig.json",
 "compilerOptions": {
 "outDir": "./out-tsc/app",
 "types": []
 },
 "files": [
 "src/main.ts",
 "src/polyfills.ts"
],
 "include": [
 "src/**/*.d.ts",
 "src/**/*.ts"
]
}
```

# Testing

- Nos fijamos ahora en los spec.ts
- Mediante comandos it podemos hacer preguntas a nuestra app y esperar resultado ( expect )
- Loteria.service.spec.ts

```
import { TestBed } from '@angular/core/testing';
import { LoteriaService } from './loteria.service';
describe('LoteriaService', () => {
 let service: LoteriaService;
 beforeEach(() => {
 TestBed.configureTestingModule({});
 service = TestBed.inject(LoteriaService);
 });
 it('should be created', () => {
 expect(service).toBeTruthy();
 });
 it('Los participantes deben ser 4', () => {
 expect(service.participantes.length).toEqual(4);
 });
});
```

# 13

## Builds y Despliegue

# 14

## Proyecto Final





