Angular

Angular CLI

Comandos

Crear nuevo proyecto.

```
ng new PROYECT-NAME
```

Crear componente / servicio / modulo / guard / pipe.

```
ng generate service NAME
```

Consultar documentacion de palabra.

```
ng doc WORD
```

Crear carpeta de distribucion para despliegue.

```
ng build
```

Consultar version de Angular.

```
ng version
```

Ejecutar aplicacion.

```
ng serve
```

Actualizar paquetes del proyecto.

```
ng update
```

Realizar pruebas e2e.

```
ng e2e
```

Realizar pruebas unitarias en el proyecto.

```
ng test PROYECT
```

Pipes

Para mostrar informacion en HTML con formato.

{{{ nombre | uppercase }}	-Mayuscula
{{ nombre | lowercase }}	-Minusculas
{{ moneda | currency }}	-Formato moneda
{{ today | date }}	-Formato fecha
{{ today | date:'yyyy-dd-MM}}	-Fecha personalizada
{{ titulo | titlecase }}	-Primera mayuscula
{{ obj | json }}	-Muestra un JSON legible
{{ obj_expression | async }}	-Subscribe a promesa o observable y muestra emision.

NgClass y NgStyle

Recibe un objeto donde la llave es la propiedad a añadir y el valor lo que se añadira

```
<img
[ngStyle]="{ 'display': isLoaded ? '': 'none' }"
[ngClass]="{ 'text-center': true }"
```

Directivas Common

For

Angular tradicional

```
<img *ngFor="let i of gifService.gifs">
```

Angular 17 (track = identificador de item)

```
@for( i of gifService.gifs; track i.id ){
  <img [src]="i.url" alt="">
}
```

If - else

Angular tradicional

```
<img *ngif="5==5; else eleBlock">
<ng-template #elseBlock>
  <p>else</p>
</ng-template>
```

Angular 17

```
@if ( 5 == 5 ) {
  <img alt="">
}@else {
  <p></p>
}
```

Switch

Angular tradicional

```
<div [ngSwitch]="numero">
  <div *ngSwitchCase="5"> ... </div>
  <div *ngSwitchDefault> ... </div>
</div>
```

Angular 17

```
@switch (numero){
  @case (5) {
    <p>Hola</p>
  }
  @default {
    <p>Hola defect</p>
  }
}
```

VARIABLES DE PLANTILLA

Con un # puedes crear una referencia al objeto html y mandarla por un metodo a el componente ts y manipular el elemento tal como un vanilla javascript.

```
<input #variable (click)="atraparVariable( variable )">
```

```
atraparVariable( input: HTMLInputElement ){  }
```

Angular

Decoradores

@ViewChild()
Tal como un selector para modificar en JavaScript, viewChild acepta como parametro el selector para el elemento del HTML,

```
@ViewChild(".div") elementHTML!: ElementRef<HTMLElement>;
```

@Input()
Para recibir informacion desde un componente padre.

```
@Input() image!: string;
```

```
<app-gif [image]=" i.images.original.url " ></app-gif>
```

@Output()
Para emitir un evento al padre, lo que significa poder compartir informacion. En el padre con $event recibe la informacion en un metodo.

```
@Output() public eventEmiter: EventEmitter<Character> = new EventEmitter();
emitirPersonaje(){
    this.eventEmiter.emit( {...this.personaje} );
}
```

```
<app-form-dbz (eventEmiter)="pushCharacter($event)" ></app-form-dbz>
```

@Component()
Define una clase como componente, con selector html, plantilla html y estilos CS, tambien puede definir animaciones, etc.

```
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrl: './app.component.scss'
})
export class AppComponent {  }
```

@NgModule()
Define una clase como modulo, puede importar otros modulos, declarar los componentes que conforman su modulo, exportar componentes para que se usen fuera de su modulo si el modulo se usa en otro y declarar funciones o servicios proveedores

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

@HostListener()
Escucha un evento de el componente en general, en click, si se da click en el componente, no importa que.

```
@HostListener('click')
public onClick() {
    console.log("Click");
}
```

Routing

RouterModule
El router module te permite cargar componente o importar un modulo de forma perezosa dependiendo la ruta.
Es importante saber que si se usa un componente en la ruta, cuyo componente NO pertenece al modulo donde se esta importando en la ruta, las dependencias de el modulo que ese componente ajeno necesita no se cargaran en el modulo de la ruta, lo que puede llevar a problemas.

```
import { Route, RouterModule } from '@angular/router';
import { AboutPageComponent } from './shared/pages/about-pag
import { HomePageComponent } from './shared/pages/home-page/
import { SharedModule } from './shared/shared.module';

const routes: Route[]= [
  {
      path: 'home',
      component: HomePageComponent
  },
  {
      path: "**",
      redirectTo: "home"
  }
]

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    SharedModule,
    RouterModule.forRoot( routes )
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

El contenido de la ruta se cargara en la etiqueta.

```
<hr>
<router-outlet></router-outlet>
```

Angular

RouterModule.forRoot( )
La importacion de las rutas raiz siempre se importaran en el modulo principal “App.Module”, seran las rutas raiz y principales del proyecto.
RouterModule.forChild( )
La importacion de las rutas hijas se hara en modulos hijos de forRoot o de otro forFeature, seran rutas que dependeral de la ruta raiz
Route[]:
El arreglo de objetos de tipo ruta define las rutas posibles, contiene atributos como:

- **path:** string que define el path de la ruta.
- **component**: componente a cargar en la ruta
- **redirectTo**: redirecciona a otra ruta.
- **pathMatch**: define presicion de la ruta (“full”)
- **loadChildren**: importa modulos perezosa.
- **children**: otras rutas hijas de la ruta

LazyLoad

loadChildren( )
La importacion de modulos hijos se puede realizar hasta que visiten la ruta de ese modulo, por lo que al ser este evento asincrono se tratara de una funcion con una promesa

```
{
  path: "perros",  // PATH DE EL MODULO
  loadChildren: () => {
    return import("./perros/perros.module")
      .then( m => m.PerrosModule ) // LAZY LOAD PARA CARGAR EL MODULO
  },
  component: PerrosComponent // COMPONENTE PRINCIPAL A CARGAR DEL MODULO HIJO
},
```

Es buena practica declarar tambien el componente principal del modulo perezoso, ya que facilitara el uso de rutas hijas en el modulo perezoso, debido a que el componente principal podra tener el <router-outlet> para mostrar el contenido SPA.

Parametros por ruta

En el modulo de rutas:

```
{
  path: "puddle/:id",
  component: PuddleComponent,
},
```

En el componente donde se atrapara.

```
export class PuddleComponent implements OnInit {
  constructor( private activatedRoute: ActivatedRoute ){}

  ngOnInit(): void {
    this.activatedRoute.params.subscribe( param => param["id"] )
  }
}
```

Custom Pipes

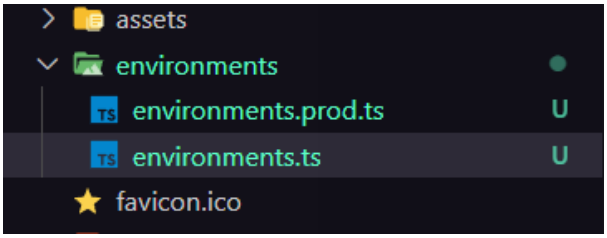
Al igual que un componente, tiene que ser declarado en el modulo y si quiere ser compartido tiene que exportarse en el mismo.
@Pipe(): El decorador Pipe de la clase, declarara el nombre con el que se utilizara,
PipeTransform: La clase debera importar la interfaz PipeTransform, que nos otorgara el metodo transform que se ejecutara al usar el pipe.
Transform(): Como primer parametro sera el valor a transformar, mientras que los consecuentes pueden ser usados para afectar el comportamiento, se retornada el contenido a mostrar.

```
@Pipe({
  name: 'toggle'  // Nombre del pipe
})
export class TogglePipe implements PipeTransform {
  transform(value: string, toUpper: boolean = false ): string {
    return toUpper ?
      value.toUpperCase() :
      value.toLocaleLowerCase();
  }
}
```

```
<h1> {{ 'HoLa' | toggle: true }} </h1> <!— HOLA —>
<h1> {{ 'HoLa' | toggle: false }} </h1> <!— hola —>
```

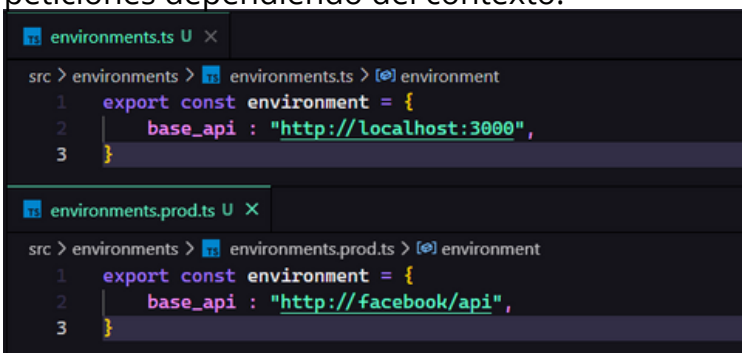
Environments

Las variables de entorno on variables globales que cambiaran su valor dependiendo del contexto de la aplicacion.
Para ello se crean generalmente a la altura de src



Angular

Por ejemplo, para cambiar la url base de las peticiones dependiendo del contexto.



Para declarar que se cambiara el uso de un archivo con otro a la hora de hacer el build se especificara en el angular.json.en production.

Se puede correr una simulacion de prod con **ng serve --configuration production**

Guards

Los guards de Angular permiten controlar la entrada y salida de las rutas antes de que se ejecute si quiera la carga de un modulo perezoso, es util para modulos de autenticacion por ejemplo, o si podra salir de un formulario.
Un guard no es mas que un servicio que implementa una interfaz de un guard.

CanActive

Determina si un usuario puede entrar a una ruta de la aplicacion, retornando un bool en el metodo de la interfaz.

Angular 17

```ts
import { CanActivateFn } from '@angular/router';

export const accesoGuard: CanActivateFn = (route, state) => {
  //* VALIDACION O PROCESO PARA DEJAR SI PUEDE ACCEDER A NO
  return true; // TRUE para acceder, FALSE para denegar
};
```

En un servicio

```ts
@Injectable({ providedIn: "root" })
export class accesoGuard implements CanActivate{

  canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot) {
    //* VALIDACION O PROCESO PARA DEJAR SI PUEDE ACCEDER A NO
    return true; // TRUE para acceder, FALSE para denegar
  }
```

Implementacion en la ruta

```ts
{
  path: "info",
  loadChildren: () => import("./info/info.module").then( m => m.Inf
  component: InfoComponent,
  canActivate: [ accesoGuard ]
},
```

CanDeactivate

Determina si un usuario puede salir de un componente, util si el usuario va a salir de la ruta sin guardar cambios, a diferencia de canActive, este no controlara si se afecta la ruta directamente, solo si la aplicacion redirecciona internamente.

```ts
export const salidGuard: CanDeactivateFn<unknown> =
(component, currentRoute, currentState, nextState) => {
  console.log("Pinche guard no me dejes salir!!!!");
  return false; //* True si puede salir de la ruta, false si no
};
```

CanMatch

Similar a canActive, solo dejara acceder a la ruta si devuelve verdadero, pero al devolver false, buscara otra ruta con el mismo nombre ya se redireccionara ahi automaticamente.

```ts
export const matchGuard: CanMatchFn = (route, segments) => {
  //? LOGICA PARA DEJAR ACCEDER O NO
  return true;
};
```

Inyectar en una función

Si se necesita usar un servicio, pero no se esta usando en una clase si no solo es funcion, usa Inject

```ts
export const accesoGuard: CanActivateFn = (route, state) => {
  const enrutador: Router = Inject( Router );
  enrutador.navigateByUrl("/login");
  return true;
};
```

Services

Un servicio no es mas que una clase singleton, donde la informacion que guarde sera consistente en todos los componentes, como una clase estatica se podria ver, se puede declarar que solo se usara en un modulo o en toda la aplicacion.

```ts
@Injectable({
  providedIn: 'root' // Servicio en root
  //? En caso de no ser root, se debe especificar en el modulo
  //? Declarado en el apartado de providers
})
export class ServicioService {
  // EL dato sera el mismo en cualquier lugar
  public nombre = "Luis";
  constructor() { }
}
```

Angular

Resolver

Los resolvers tambiense declaran en las rutas, se encargan de resolver informacion antes de acceder a la ruta para afirmar que la informacion este lista una vez se acceda. Es un servicio que implementa Resolve o usarlo en funcion con ResolveFn

Crear Resolve Angular 17

```
export const fetchUserData: ResolveFn<User> =
(route: ActivatedRouteSnapshot, state: RouterStateSnapshot)
: Observable<User> => {
    return of({ name: "John Doe", age: 30 }).pipe(delay(1000));
}
```

Con servicio

```
@Injectable({
    providedIn: 'root'
})
export class UserResolver implements Resolve<User>{
    constructor(private http: HttpClient) { }
    resolve(route: ActivatedRouteSnapshot, state: RouterStateSnapshot): Observable<User> {
        return this.http.get<User>('api/user');
    }
}
```

El parametro generico que se asigna, es el modelado de los datos que resolvera. El metodo resolve debe regresar un observable que emita un modelo que sea el mismo que el generico

Implementar en ruta

```
{
  path: "login",
  loadChildren: () => import("./login/login.module").then( m => m.Lo
  component: LoginComponent,
  resolve: {
    userData : fetchUserData
  }
},
```

Acceder a los datos resueltos.

Se accede a los datos resueltos con el activated route snapshot data

```
export class LoginComponent {
  userData ?: User;

  constructor(private route: ActivatedRoute) {}

  ngOnInit() {
    this.userData = this.route.snapshot.data['userData'];
  }
}
```

Standalone

Los standalone puede ser *componentes, pipes y cosas que dependan de un modulo* para usarse, los standalone son como un modulo que solo tiene un componente, esos componentes importan sus propias dependencias y se importan como modulos en demas partes.

Definicion

```
@Component({
  selector: 'app-componente-solo',
  templateUrl: './componente-solo.component.html',
  standalone: true,
  imports: [ CommonModule ]
})
export class ComponenteSoloComponent {
  mostrar = false;

}
```

Uso

Se importa tal cual fuera un modulo, en el modulo que se va a utilizar dicho componente

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

Y en HTML se invocara con su selector.

```
hola mundo
<app-componente-solo></app-componente-solo>
```

Formularios Reactivos

El uso de formularios reactivos facilita las validaciones y tratamiento de datos en formularios de una forma mas fácil y legible en Angular.

Importaciones en modulo.

```
import { ReactiveFormsModule } from '@angular/forms';

@NgModule({
  declarations: [ /* Components */ ],
  imports: [
    /* Other imports */
    ReactiveFormsModule,
  ]
})
```

Angular

Creación de un campo del formulario (control)

HTML

```
<input type="text"
       class="form-control"
       [formControl]="producto"
       placeholder="Nombre del producto">
```

TypeScript

```
export class BasicPageComponent {
  //? FormControl( ValorPrincipal, Validadores, Validadores asincronos )
  public producto = new FormControl("Default Value", [], []);

}
```

Creación de formulario (grupo de controls)

HTML

```
  <form [formGroup]="form" (ngSubmit)="subir()" autocomplete="off">
      <input type="text"
      class="form-control"
      formControlName="producto"
      placeholder="Nombre del producto">
```

TypeScript

```
public form!: FormGroup;
constructor(
  private fb: FormBuilder
){}

ngOnInit(): void {
  this.form = new FormGroup({
    "producto": new FormControl("some",[],[])
  });
}
```

Creación de formulario con FormBuilder

TypeScript

```
public form!: FormGroup;
constructor(
  private fb: FormBuilder
){}

ngOnInit(): void {
  this.form = this.fb.group({
    "producto" : ["valor", [],[]]
  });
}
```

FormArray