UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Programador Web Avanzado

Clase N° 9: Angular – Componentes, Servicios, validación de formularios

Profesor: Ing. Leandro Rodolfo Gil Carrano
Email: leangilutn@gmail.com

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Componentes

Árbol de componentes

Tendrás un árbol de componentes que forman tu aplicación y cada persona lo podrá organizar de su manera preferida.
Siempre existirá un componente padre y a partir de ahí podrán colgar todas las ramas que sean necesarias para crear tu aplicación.
En nuestro árbol, como posible organización, podemos tener en un primer nivel los bloques principales de la pantalla de nuestra aplicación.

- Una barra de herramientas, con interfaces para acciones principales (lo que podría ser una barra de navegación, menús, botonera, etc.).
- Una parte principal, donde se desplegarán las diferentes "pantallas" de la aplicación.
- Un área de logueo de usuarios.

Árbol de componentes

Luego, cada uno de los componentes principales se podrá subdividir, si se desea, en nuevos árboles de componentes.

- En la barra de herramientas principal podríamos tener un componente por cada herramienta.
- En el área principal podríamos tener un componente para cada "pantalla" de la aplicación o "vista".
- A su vez, dentro de cada "vista" o "pantalla" podíamos tener otra serie de componentes que implementen diversas funcionalidades.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Componentes vs Directivas

Los componentes son **piezas de negocio**, mientras que las directivas se suelen usar para presentación y **problemas estructurales.**

Podemos pensar en un componente como un contenedor donde solucionas una necesidad de tu aplicación.

Tipos de directivas

Existen tres tipos de directivas:

- **Componentes**: Un componente es una directiva con un template. Habrá muchas en tu aplicación y resuelven necesidades del negocio.
- **Directivas de atributos**: Cambian la apariencia o comportamiento de un elemento. Por ejemplo tenemos **ngClass**, que nos permite colocar una o más clases de CSS (atributo class) en un elemento.
- **Directivas estructurales**: Son las que realizan cambios en el DOM del documento, añadiendo, manipulando o quitando elementos. Por ejemplo ngFor, que nos sirve para hacer una repetición (similar al ngRepeat de Angular 1.x), o ngIf que añade o remueve elementos del DOM con respecto a una expresión condicional.

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Decorador

¿Qué es un decorador?

Básicamente es una implementación de un patrón de diseño de software que en sí sirve para extender una función mediante otra función, pero sin tocar aquella original, que se está extendiendo. El decorador recibe una función como argumento (aquella que se quiere decorar) y devuelve esa función con alguna funcionalidad adicional.

Las funciones decoradoras comienzan por una "@" y a continuación tienen un nombre. Ese nombre es el de aquello que queramos decorar, que ya tiene que existir previamente. Podríamos decorar una función, una propiedad de una clase, una clase, etc.

¿Qué información se agrega con el decorador?

```
@Component({
    moduleId: module.id,
    selector: 'test-angular2-app',
    templateUrl: 'test-angular2.component.html',
    styleUrls: ['test-angular2.component.css']
})
```

¿Qué información se agrega con el decorador?

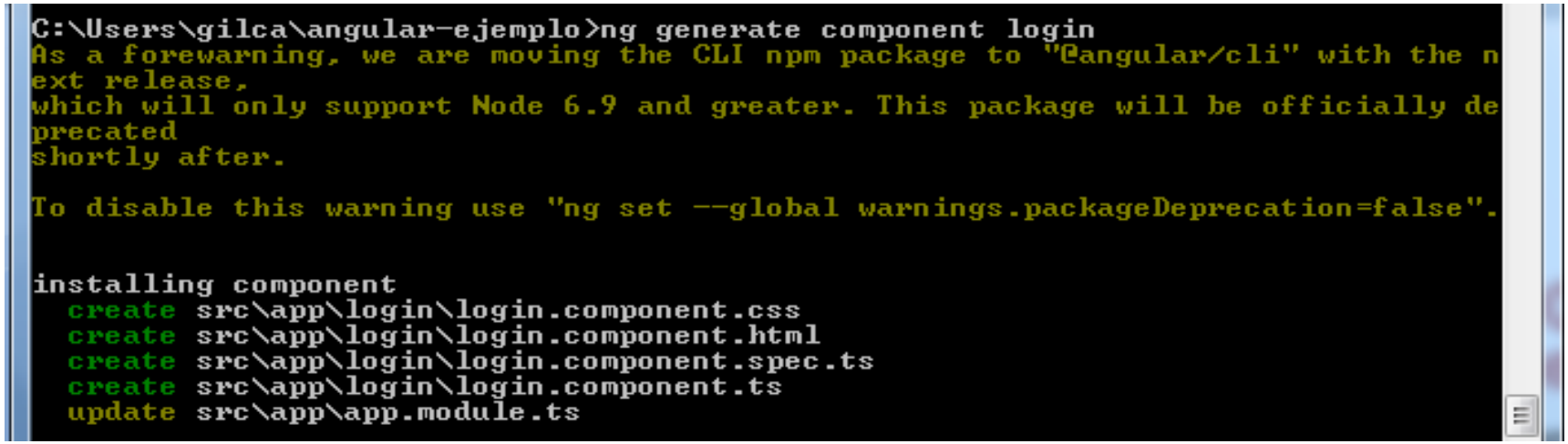
- **moduleId**: No vamos a ver todavía esta propiedad, es algo que tiene que ver con CommonJS y sirve para poder resolver Urls relativas.
- **selector**: este es el nombre de la etiqueta nueva que crearemos cuando se procese el componente. Es la etiqueta que usarás cuando quieras colocar el componente en cualquier lugar del HTML.
- **templateUrl**: es el nombre del archivo .html con el contenido del componente, en otras palabras, el que tiene el código de la vista.
- **styleUrls**: es un array con todas las hojas de estilos CSS que deben procesarse como estilo local para este componente. Como ves, podríamos tener una única declaración de estilos, o varias si lo consideramos necesario.

Crear un nuevo Componente

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ng generate

Para crear un nuevo componente debemos ejecutar:

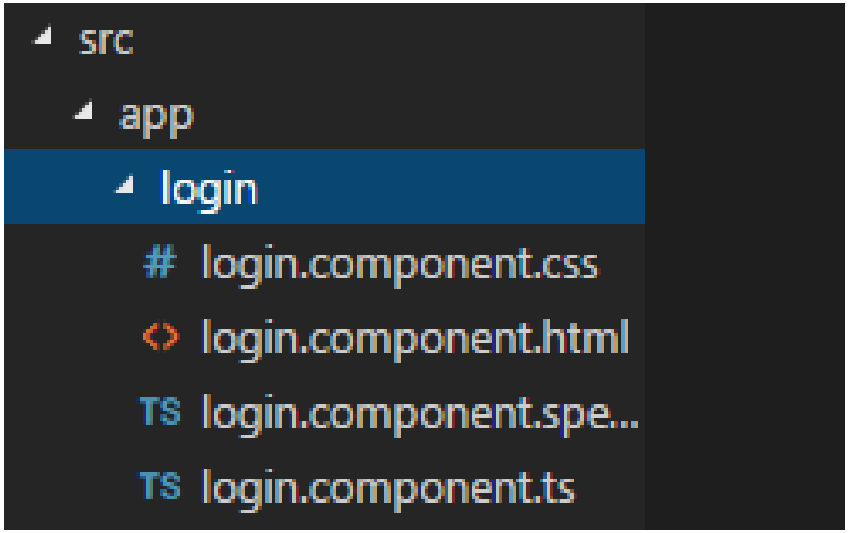


ng generate component nombre_componente

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ng generate

Si observas ahora la carpeta "src/app" encontrarás que se ha creado un directorio nuevo con el mismo nombre del componente que acabamos de crear.



Utilizar componente creado

En el lugar de la aplicación donde lo vayas a usar el componente, tienes que escribir el HTML necesario para que se muestre. El HTML no es más que la etiqueta del componente, que se ha definido en la función decoradora, atributo "selector"

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

  constructor() { }
```

Utilizar componente creado

Si queremos utilizar dicho componente debemos agregar en el HTML que queramos la etiqueta **<app-login></app-login>**

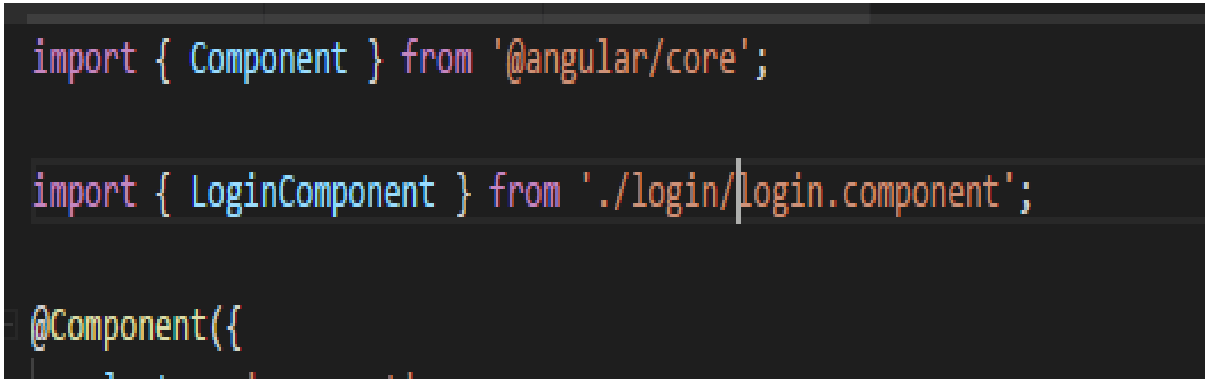
```
<h1>
  Hola {{nombre_persona}}
</h1>
<input [(ngModel)]="nombre_persona"/>

<app-login></app-login>
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Utilizar componente creado

En el componente desde donde pretendas usar ese otro componente necesitas importar su código.

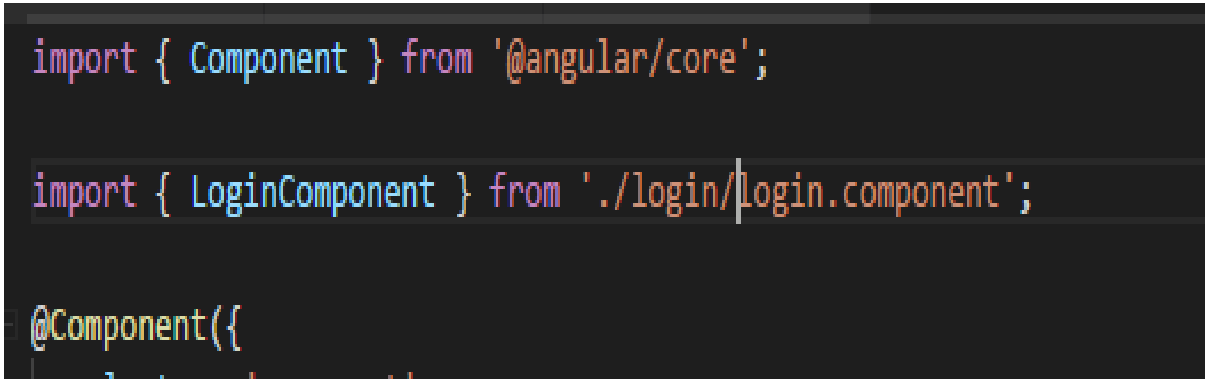


Como veras debes indicar la carpeta en la cual se encuentra tu componente (login) y el nombre del componente (login.component)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Utilizar componente creado

En el componente desde donde pretendas usar ese otro componente necesitas importar su código.



Como veras debes indicar la carpeta en la cual se encuentra tu componente (login) y el nombre del componente (login.component)

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ejercicio 1

Crear el componente **<app-home></app-home>**

Este componente deberá tener la home del sitio que queramos desarrollar, por ejemplo un ecommerce.

Nota: No incluir en el componente home el head, header y footer. Estos los vamos a incluir directamente en el archivo index.html

ngClass

Directivas

Nos permite alterar las clases CSS que tienen los elementos de la página.
Las directivas se usarán para solucionar necesidades específicas de manipulación del DOM y otros temas estructurales.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

[class]

Lo primero es decir que la directiva ngClass no es necesaria en todos los casos. Las cosas más simples ni siquiera la necesitan. El atributo "class" de las etiquetas si lo pones entre corchetes funciona como propiedad a la que le puedes asignar algo que tengas en tu modelo

```
export class AppComponent {
  title = 'app works!';

  clase="css-class";
  mostrar(){

    this.clase="prueba";
  }
}
```

```
<style>
  .css-class{
    background: red;
  }
  .prueba{
    background: green;
  }
</style>
<h1>
  {{title}}
</h1>

<button (click)="mostrar()">Mostrar contenido oculto</button>

<div [class]="clase">Una clase marcada por el modelo</div>
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ngClass

La directiva ngClass es necesaria para, de una manera cómoda asignar cualquier clase CSS entre un grupo de posibilidades.

A esta directiva le indicamos como valor:

1. Un array con la lista de clases a aplicar. Ese array lo podemos especificar de manera literal en el HTML.

```
<!-- Ejemplo ngClass con array -->
<p [ngClass]="['negativo', 'off']">Pueden aplicarse varias clases</p>
```

Definición del array con javascript:

Vista (html)

```
<p [ngClass]="ngClass_array">Pueden aplicarse varias clases</p>
```

Controlador (ts)

```
ngClass_array=['positivo', 'si'];
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ngClass

1. Un objeto con propiedades y valores (lo que sería un literal de objeto Javascript). Cada nombre de propiedad es una posible clase CSS que se podría asignar al elemento y cada valor es una expresión que se evaluará condicionalmente para aplicar o no esa clase.

Vista (html)

```
<!-- Ejemplo ngClass con objetos de javascript -->
<p [ngClass]="{positivo: cantidad > 0, negativo: cantidad < 0, off: desactivado, on: !desactivado }">Línea</p>
```

Controlador (ts)

```
export class AppComponent {
  title = 'app works!';

  clase="css-class";
  ngClass_array=['positivo', 'si'];
  cantidad=1;
  desactivado=false;
```

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ngFor

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ngFor

La directiva ngFor, capaz de hacer una repetición de elementos dentro de la página. Esta repetición nos permite recorrer una estructura de array y para cada uno de sus elementos replicar una cantidad de elementos en el DOM.

Vista (html)

```
<p *ngFor="let pregunta of preguntas">
{{pregunta}}
</p>
```

Controlador (ts)

```
preguntas: string[] = [
"¿Que día es hoy?",
"¿Cmo estuvo tu día hoy?",
"¿Estás aprendiendo Angular 2?"
];
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

ngFor con objetos

Vista (html)

```
<h2>Ejemplo de aplicacion de ngFor con objetos</h2>
<p *ngFor="let objPregunta of preguntasObj">
{{objPregunta.pregunta}}:
<br>
<span class="si">Si {{objPregunta.si}}</span> /
<span class="no">No {{objPregunta.no}}</span>
</p>
```

Controlador (ts)

```
preguntasObj = [
{
pregunta: "¿Que día es hoy?",
si: 22,
no: 95
},
{
pregunta: "¿Cmo estuvo tu día hoy?",
si: 262,
no: 3
},
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ejercicio 2

Continuando con el ejercicio 1 utilizar la directiva **ngFor** para mostrar los productos cargados en un mock up de datos.

Utilizar el resto de las directivas vistas para armar una home atractiva del ecommerce.

ngIf

ngIf

Si la condición se cumple, su elemento se inserta en el DOM, en caso contrario, se elimina del DOM.

Vista (html)

```
<h2>Ejemplo de aplicacion de ngIf</h2>
<p *ngIf="ejemplo_ngif">Mostra mensaje con ngIf</p>
```

Controlador (ts)

```
ejemplo_ngif=false;
mostrar(){

  this.clase="prueba";
    this.ejemplo_ngif=true;
}
```

UTN.BA

UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Routing

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

App rounting

Tanto para la raíz como para las ramas funcionales se creará un fichero. En la raíz será llamado **app-routing.module.ts** con un contenido como este:

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

const routes: Routes = [];

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

App rounting

Este módulo de un único fichero sirve para definir las rutas de otro módulo padre asociado, **app.module.ts**, el cual quedará más o menos así:

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

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

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

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Generar rutas

En el archivo **app-routing.module.ts** debemos especificar las rutas de nuestra aplicación. Lo hacemos en la constante **routes**

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import { HomeComponent } from './home/home.component';
import { LoginComponent } from './login/login.component';
import { ContactoComponent } from './contacto/contacto.component';

const routes: Routes = [
  {path:"", component:HomeComponent},
  {path:"login",component:LoginComponent},
  {path:"contacto",component:ContactoComponent}

];

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

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Generar rutas

En el archivo **app.module.ts** agregamos los componentes en declarations

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

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Links

Para enlazar las paginas debemos utilizar la directiva **routerLink**

```
<nav>
  <ul>
    <li routerLink="/">Home</li>
    <li routerLink="/login">Login</li>
  </ul>
</nav>
```

```
<!--menú de navegación, sin href-->
<nav>
  <a [routerLink]="['/']">Inicio</a>
  <a [routerLink]="['/contacto',42]">Contacto</a>
</nav>
<!--Este componente nativo hace que el enrutador cargue una página dinámicamente-->
<router-outlet></router-outlet>
```

En el primer caso redirigimos a / y /login

En el segundo a /contacto/42, siendo 42 un parámetro dinámico

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Rutas hijas

Se define como ruta hija a aquellas subrutas, por ejemplo tenemos el componente o ruta principal **contacto** y las subrutas **nuevo, lista**

Quedando **/contacto/nuevo** y **/contacto/lista**

```
const routes: Routes = [
  {
    path: 'contacto',
    component: ContactoComponent,
    children: [ // rutas hijas, se verán dentro del componente padre
      {
        path: 'nuevo', // la ruta real es movimientos/nuevo
        component: ContactoComponent
      },
      {
        path: 'lista',
        component: ContactoComponent
      }
    ]
  },
  {
    path: 'contacto/:id', // parámetro variable id
    component: ContactoComponent
  }
];
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Parámetros - Rounting

Para recibir parametros en el routing debemos agregar a la ruta **:nombre_parametro** por ejemplo:

```
const routes: Routes = [
  {
    path: 'contacto',
    component: ContactoComponent,
    children: [ // rutas hijas, se verán dentro del componente padre
      {
        path: 'nuevo', // la ruta real es movimientos/nuevo
        component: ContactoComponent
      },
      {
        path: 'lista',
        component: ContactoComponent
      }
    ]
  },
  {
    path: 'contacto/:id', // parámetro variable id
    component: ContactoComponent
  }
];
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Parámetros - Enlace

En el enlace debemos definir lo siguiente

```
<!--menú de navegación, sin href-->
<nav>
  <a [routerLink]="['/']">Inicio</a>
  <a [routerLink]="['/contacto',42]">Contacto</a>
</nav>
<!--Este componente nativo hace que el enrutador cargue una página dinámicamente-->
<router-outlet></router-outlet>
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Parámetros - Componente

En el componente podemos obtener el parámetro de la siguiente manera

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

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

  constructor(private route: ActivatedRoute) { }

  ngOnInit() {
      this.route.params
      .subscribe(params => {
        const _id = params['id'].toString();
        console.log("Id",_id);

      });
  }

}
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Comandos

```
ng generate component home
```

```
ng generate service usuarios
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Formularios

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Validación de formularios

Para validar formularios utilizaremos **formBuilder**

Primero debemos importar el componente **ReactiveFormsModule** en **app.module.ts** e incluirlo en **imports:**

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

```
imports: [
  BrowserModule,
  FormsModule,
  HttpModule,
  ReactiveFormsModule
],
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Validación de formularios

Luego debemos incluir **FormBuilder**, **Validators** dentro del componente en el cual queramos realizar la validación de formularios:

```
import { FormBuilder,  Validators} from '@angular/forms';
```

Vamos a declarar la variable **loginForm,** la cual contendrá la lógica de validación del formulario. Por otro lado en el constructor debemos recibir el **FormBuilder** como parámetro.

```
export class LoginComponent implements OnInit {
  public loginForm
  constructor(public fb: FormBuilder) {
    this.loginForm = this.fb.group({
      email: ["", Validators.required],
      password: ["", Validators.required]
    });
  }

  ngOnInit() {
  }

}
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Validación de formularios

En la vista debemos declarar **[formGroup]** en el form que queramos validar:

```
<form [formGroup]="loginForm" (ngSubmit)="doLogin($event)">
    <input formControlName="email" type="email" placeholder="Emai
```

En cada elemento del formulario a validar, en lugar de utilizar un ngModel, debemos utilizar **formControlName.** Le debemos asignar el mismo nombre que le asignamos en el controlador.

```
<input formControlName="email" type="email" placeholder="Email">
```

Clase validators

Nos provee las siguientes validaciones:

- **Validators.required** = Comprueba que el campo sea llenado.
- **Validators.minLength** = Comprueba que el campo cumpla con un mínimo de caracteres.
- **Validators.maxLength** = Comprueba que el campo cumpla con un máximo de caracteres.
- **Validators.pattern** = Comprueba que el campo cumpla con un patrón usando una expresión regular.
- **Validators.email** = Comprueba que el campo cumpla con un patrón de correo válido.

Clase validators

- **loginForm.get('nombre').errors**: Si el campo nombre del formulario myForm tiene algún error esta condición será true.
- **loginForm.get('nombre').dirty**: Devuelve true si el usuario ha interactuado con el campo en cuestión.
- **loginForm.get('nombre').hasError('required')**: Devuelve true si el campo nombre arroja un error de tipo required sobre el campo nombre.
- **loginForm.invalid**: Devuelve true si el campo nombre arroja un error de tipo required sobre el campo nombre.

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Ejercicio 4

Al final de la pagina home desarrollara agregar un formulario de registros de usuarios que tenga los siguientes campos:

- Nombre (*)
- Apellido (*)
- Teléfono
- Email (*)
- Password (*): Debe tener entre 6 y 10 caracteres

(*) Campos obligatorios

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Servicios

Definición

Los Componentes son grandes consumidores de servicios. No recuperan datos del servidor, ni validan inputs de usuario, ni logean nada directamente en consola. Delegan todo este tipo de tareas a los Servicios.

Los servicios deberían contener/hacer algo muy específico. Por ejemplo, serían susceptibles de encapsular en un servicio:

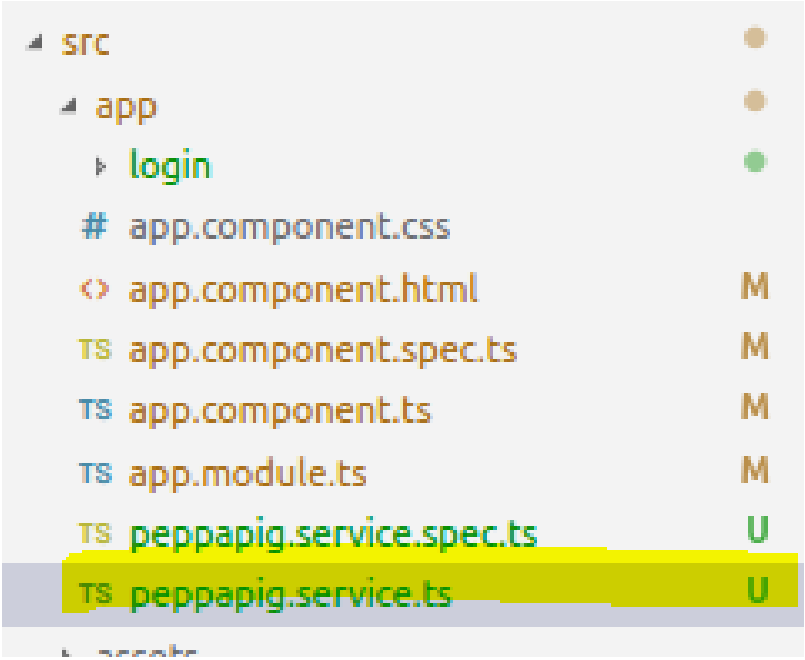
- Servicio de logging
- Servicio de datos
- Bus de mensajes
- Cálculo de Impuestos
- Configuración de la app

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Crear un servicio

Ejecutar **ng generate service nombre-servicio**

Por ejemplo ejecutar **ng generate service pepapig**. Esto nos generar un archivo peppapig.service.ts en el directorio app



UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Crear un servicio

En el contenido del archivo incluimos lo siguiente:

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

@Injectable({
  providedIn: 'root'
})
export class PeppapigService {

  constructor() { }
  getPeppaFriends(){
    return ["pedro poni","madamme gazelle", "dani dog"];
  }
}
```

UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Utilizar un servicio desde un componente

Luego incluimos el servicio en el componente en el cual queramos utilizarlo. Por ejemplo en el **app.component.ts**

```
import { PeppapigService } from './peppapig.service';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css'],
  providers: [PeppapigService]
})
export class AppComponent {
  title = 'app works!';
  visible = false;
  clase="css-class";

  constructor(public peppaservice:PeppapigService) {
```

Utilizar un servicio desde un componente

Luego llamamos al método definido del service, por ejemplo:

```
}
obtenerServicePeppaPig(){
  console.log(this.peppaservice.getPeppaFriends());

}
```