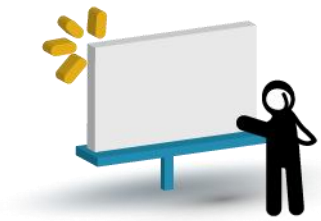
UTN.BA
UNIVERSIDAD TECNOLÓGICA NACIONAL
FACULTAD REGIONAL BUENOS AIRES

Centro de e-Learning

Desarrollo con Angular

Centro de e-Learning SCEU UTN - BA.
Medrano 951 2do piso (1179) // Tel. +54 11 4867 7589 / Fax +54 11 4032 0148
www.sceu.frba.utn.edu.ar/e-learning

Unidad 2: Validación de formularios. Creación de componentes avanzados



Presentación:

En la presente unidad concluimos con el tema comenzado en la unidad anterior, Angular 2. Aquí aprendemos diferentes conceptos y elementos propuestos por el framework, para lograr conectarnos con nuestro servidor de datos y desarrollar, así, una aplicación funcional.

Objetivos:

Que los participantes:

- Sepan crear y utilizar componentes.
- Aprendan a utilizar las directivas.
- Comprendan para qué y cómo utilizar el decorator.
- Sepan validar formularios.

Bloques temáticos:

- Directivas.
- Formularios.

Consignas para el aprendizaje colaborativo

En esta Unidad los participantes se encontrarán con diferentes tipos de actividades que, en el marco de los fundamentos del MEC*, los referenciarán a tres comunidades de aprendizaje, que pondremos en funcionamiento en esta instancia de formación, a los efectos de aprovecharlas pedagógicamente:

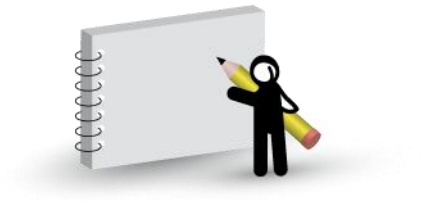
- Los foros proactivos asociados a cada una de las unidades.
- La Web 2.0.
- Los contextos de desempeño de los participantes.

Es importante que todos los participantes realicen algunas de las actividades sugeridas y compartan en los foros los resultados obtenidos.

Además, también se propondrán reflexiones, notas especiales y vinculaciones a bibliografía y sitios web.

El carácter constructivista y colaborativo del MEC nos exige que todas las actividades realizadas por los participantes sean compartidas en los foros.

**** El MEC es el modelo de E-learning colaborativo de nuestro Centro.***



Tomen nota:

Las actividades son opcionales y pueden realizarse en forma individual, pero siempre es deseable que se las realice en equipo, con la finalidad de estimular y favorecer el trabajo colaborativo y el aprendizaje entre pares. Tenga en cuenta que, si bien las actividades son opcionales, su realización es de vital importancia para el logro de los objetivos de aprendizaje de esta instancia de formación. Si su tiempo no le permite realizar todas las actividades, por lo menos realice alguna, es fundamental que lo haga. Si cada uno de los participantes realiza alguna, el foro, que es una instancia clave en este tipo de cursos, tendrá una actividad muy enriquecedora.

Asimismo, también tengan en cuenta cuando trabajen en la Web, que en ella hay de todo, cosas excelentes, muy buenas, buenas, regulares, malas y muy malas. Por eso, es necesario aplicar filtros críticos para que las investigaciones y búsquedas se encaminen a la excelencia. Si tienen dudas con alguno de los datos recolectados, no dejen de consultar al profesor-tutor. También aprovechen en el foro proactivo las opiniones de sus compañeros de curso y colegas.

Directivas

¿Que son las directivas de atributo?

Las Directivas nos permiten 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.

[class]

Lo primero es decir que la directiva class 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>
```

[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'];
```

2. 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>
```

Aplica la clase css "positivo" cuando la variable "cantidad" es mayor a 0 o negativo cuando la misma es menor a 0

Controlador (ts)

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

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

[ngStyle]

El **ngStyle** me sirve para aplicar estilos directamente a elementos html. La documentación de Angular podemos encontrarla aquí: https://angular.io/api/common/NgStyle

Su sintaxis es muy sencilla simplemente la colocamos entre corchetes **[ngStyle]**

```
<p [ngStyle] = "{'font-size':'18px'}">
  Esta es mi primer etiqueta <p>
</p>
<!-- Agregamos una variable y la utilizamos para definir el font-size -->
<p [ngStyle] = "{'font-size':tamano + 'px'}">
  Esta es mi primer etiqueta <p>
</p>
<!-- Lo mismo de arriba lo podemos escribir de esta forma, los atributos se reemplaza
  el - por la letra en mayuscula de la palabra que sigue  -->
<p [style.fontSize] = "'40px'">
  Esta es mi primer etiqueta <p>
</p>
<p [style.fontSize.px] = "tamano">
  Esta es mi primer etiqueta <p>
</p>
```

“tamaño” es una variable de clase definida en el controlador:

```
export class NgStyleComponent implements OnInit {

  tamano:number = 30;

  constructor() { }

  ngOnInit() {
  }

}
```

Crear directiva propia

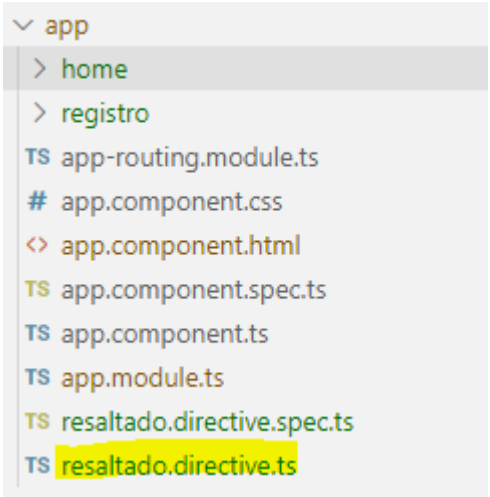
Para crear una directiva propia debemos realizar los siguientes pasos:

1. Ejecutar el siguiente comando por consola
   ng g directive *nombre_directiva*
   Ejemplo

```
PS C:\sites\angular\ejs2020> ng g directive resaltado
```

2. Las modificaciones realizadas el ejecutar dicho comando son las siguientes

   a. Se crea el archivo ***nombre_directiva*.directive.ts**



   b. En el **app.module.ts** se producen las siguientes modificaciones

```
import { ResaltadoDirective } from './resaltado.directive'; //Se agrega al crear la directiva

@NgModule({
  declarations: [
    AppComponent,
    HomeComponent,
    RegistroComponent,
    ResaltadoDirective //Se agrega al crear la directiva
  ],
```

3. En el archivo .directive generado aplicamos la lógica que queramos tenga nuestra directiva, ejemplo

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

@Directive({
  selector: '[appResaltado]'
})
export class ResaltadoDirective {
  constructor( private el:ElementRef ) {
    //console.log("Directiva llamada");
    //el.nativeElement.style.backgroundColor = "yellow";
  }
  @Input("appResaltado") nuevoColor:string;

  @HostListener('mouseenter') mouseEntro(){

    this.resaltar( this.nuevoColor || 'yellow' );
    //si eliminamos la funcion resaltar comentamos su llamado y descomentamos esta linea
    //de abajo solo reslata en amarillo. La funcion resaltar me permite recibir el color por parametro
    //y resaltar en ese color en lugar de siempre en amarillo como lo hariamos en la linea de abajo,
    //en este caso tendriamos que sacar en el html el pasaje del parametro
    //this.el.nativeElement.style.backgroundColor = "yellow";
  }
  @HostListener('mouseleave') mouseSalio(){
    this.resaltar( null );
    //si eliminamos la funcion resaltar comentamos su llamado y descomentamos esta linea
    //de abajo solo reslata en amarillo
    //this.el.nativeElement.style.backgroundColor = null;
  }
  private resaltar( color:string ){
    this.el.nativeElement.style.backgroundColor = color;
  }
}
```

4. Para invocar nuestra directiva desde un componente lo hacemos con [] y el nombre de la misma **“[appResaltado]”** (definido en el decorator @Directive)

```
<p [appResaltado]="'orange'">Resaltado</p>
```

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.

Controlador (ts)

```
productos=[]
constructor() {
  this.productos=[
    "producto 1",
    "producto 2"
  ]
}
```

Vista (html)

```
<div *ngFor="let producto of productos">
  <p>{{producto}}</p>
</div>
```

Ejemplo de ngFor utilizando objetos

Controlador (ts)

```
productos=[]
constructor() {
  this.productos=[
    {
      "id":1,
      "nombre":"prod1",
      "precio":100
    },
    {
      "id":2,
      "nombre":"prod2",
      "precio":200
    }
  ]
```

Vista (html)

```
<div *ngFor="let producto of productos">
  <p>{{producto.nombre}}</p>
  <p>{{producto.precio}}</p>
</div>
```

ngFor solo recorre arrays, no podes recorrer un json utilizando ngFor.

ngIf

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

Vista (html)

```
<div *ngIf="!gano">
  <div *ngFor="let producto of productos">
    <p>{{producto.nombre}}</p>
    <p>{{producto.precio}}</p>
  </div>
</div>
```

Controlador (ts)

```
export class HomeComponent implements OnInit {
  gano=false;

  constructor() {
```

Gano es una variable de clase, cuando se modifique su valor a “true” el div se renderizara. En caso de ser false el mismo no se renderiza (no es que se oculta, directamente no se renderiza)

ngSwitch

Vista (html)

```
<div [ngSwitch]="alerta">
  <div *ngSwitchCase="'success'">success</div>
  <div *ngSwitchCase="'info'">info</div>
  <div *ngSwitchCase="'warning'">warning</div>
  <div *ngSwitchDefault>default</div>
</div>
```

Controlador (ts)

```
export class HomeComponent implements OnInit {


  alerta = "info"

  constructor() {
```

En este caso se evalúa el valor de “alerta” y se renderiza el case correspondiente en la vista.

Formularios

Validación de formularios

Para validar formularios utilizaremos **formBuilder**

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

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

```
@NgModule({
  imports: [
    BrowserModule,
    AppRoutingModule,
    ReactiveFormsModule,//Incluir
    FormsModule,//Incluir
  ],
```

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

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

Vamos a declarar la variable **myForm**, la cual contendrá la lógica de validación del formulario. Por otro lado en el constructor debemos inyectar **FormBuilder**.

```
myForm:FormGroup

constructor(public fb:FormBuilder) { }
```

En el ngOnInit o en el constructor asignamos a la variable **myForm** el resultado del método group de **FormBuilder**

```
ngOnInit() {
  this.myForm = this.fb.group({
    nombre:['',[Validators.required,Validators.minLength(4)]],
    apellido:['',[Validators.required]],
    email:['',[Validators.required]],
    password:['',[Validators.required,Validators.minLength(6),,Validators.maxLength(8)]],
  })
}
```

Este método recibe como parámetro un objeto con el siguiente formato:

- El índice de cada tupla será el nombre del campo del formulario. Ejemplo **nombre** hace referencia al input nombre de la vista
- El valor de cada tupla recibe un array
    - Primer elemento será el valor por default que muestre el campo al cargar la vista.
    - Segundo elemento es un subarray con validaciones (ver validators), en el ejemplo el campo nombre esta validando que el mismo este completo y un min length de 4 caracteres.

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

```
<form [formGroup]="myForm" (ngSubmit)="registro()">
```

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 type="text" formControlName="nombre" >
```

*Hace referencia al índice de cada tupla del objeto definido en **group***

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.

Mostrar errores en la vista

• **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.

Utilizando los métodos anteriores podemos mostrar los errores en la vista de diversas maneras. Por ejemplo utilizando el *ngIf

```
<div>
  <label for="">Nombre</label>
  <input type="text" formControlName="nombre" >
  <div *ngIf="myForm.get('nombre').errors && myForm.get('nombre').dirty">
    <p *ngIf="myForm.get('nombre').hasError('required')">El campo es obligatorio</p>
    <p *ngIf="myForm.get('nombre').hasError('minlength')">Debe introducir al menos 4 caracteres</p>
  </div>
</div>
```

En el primer *ngIf consultamos si el campo nombre (utilizando el método **get**) tiene algún error y con el dirty validamos si el usuario ya interactuó con dicho campo

En los *ngIf internos consultamos si el error que tiene el campo es **required** o **minlength** para mostrar un error diferente en pantalla según el caso.

Otra manera que tenemos de mostrar los errores es utilizando **[ngClass]**

```
<input type="text" formControlName="nombre"
[ngClass]="{error:myForm.get('nombre').errors && myForm.get('nombre').dirty}">
```

En este caso asignamos la clase css **error** si el campo nombre no cumple con alguna validación y el usuario ya interactuó con el

Bibliografía utilizada y sugerida

https://cli.angular.io/

https://www.typescriptlang.org/

http://victorroblesweb.es/tag/manual-de-angular-2-en-espanol/

https://app.desarrolloweb.com/manuales/manual-angular2

https://www.youtube.com/watch?v=H9Dtqy3Fd40

http://www.angulartypescript.com/angular-2-formbuilder-example/

Lo que vimos:

En esta unidad hemos aprendido directivas, validación de formularios y creación de componentes con Angular.

Lo que viene:

En la próxima unidad veremos qué son y cómo utilizar servicios para conectarnos a nuestro api externo.