

## Material de estudio OBLIGATORIO EJE 2 Angular - Formularios y Validación de Formularios

Sitio: [Instituto Superior Politécnico Córdoba](#)

Curso: Programador Web - TSDWAD - 2022

Libro: Material de estudio OBLIGATORIO EJE 2 Angular - Formularios y Validación de Formularios

Imprimido por: Ezequiel Maximiliano GIAMPAOLI

Día: martes, 23 mayo 2023, 6:35 PM

## Tabla de contenidos

1. Formularios en Angular
2. Formularios Reactivos
3. Validación de Formularios
4. Validar previo enviar el formulario

## Formularios en Angular

La utilización de formularios en aplicaciones es muy común, permiten al usuario ingresar datos a la base de datos realizar el login, registrarse, entre otras acciones muy necesarias.

Angular nos provee dos tipos de formularios:

- **Basados en plantilla (Template driven):** proporcionan un enfoque basados en directivas. Los mismos, hacen foco en escenarios simples y no son tan reusables.
- **Reactivos (Model Driven):** proporcionan un enfoque basado en modelos por lo que son más robustos, escalables, reusables y testeables.

	Reactivos	Basados en plantillas
Configuración	Explícita. Se crea en la clase del componente.	Implícita. Se crea a través de directivas.
Modelo de Datos	Estructurado e inmutable	No estructurado y mutable
Flujo de datos (entre vista y el modelo)	Síncrono	Asíncrono
Validación de Formularios	Por medio de funciones	Por medio de directivas

Tabla: Comparación formularios reactivos y formularios de plantilla

Fuente: <https://docs.angular.lat/guide/forms-overview>

Es importante agregar, que en este documento se abordarán los formularios reactivos dado que son mucho más robustos, escalables, mantenibles y testeables.

## Formularios reactivos

Proveen un approach explícito e inmutable para administrar el estado del formulario cuyos valores cambian con el tiempo. Cada cambio de estado en el formulario retorna un nuevo valor permitiendo mantener la integridad con el modelo.

Además, cada elemento de la vista está directamente enlazado al modelo mediante una instancia de FormControl. Las actualizaciones de la vista al modelo y del modelo a la vista son síncronas y no dependen de la representación en la Interfaz de Usuario.

Dichos formularios están basados en flujos de datos del tipo Observable, donde cada entrada/valor puede ser accedido de manera asíncrona.

**Nota:** el concepto de Observable se publicó en el libro anterior. (ver documentación oficial: <https://docs.angular.lat/guide/observables-in-angular>).

Para que Angular interprete nuestros formularios como reactivos, debemos importar el módulo de **Formularios Reactivos** en el archivo app.module.ts o en los módulos dónde necesites trabajar dichos formularios.

```
import { NgModule } from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

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

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

**Nota:** Tanto los formularios reactivos como los basados en plantillas realizan un seguimiento de los cambios de valor entre los elementos de entrada del formulario con los que interactúan los usuarios y los datos del formulario en su modelo de componente. Los dos enfoques comparten bloques de construcción subyacentes, pero difieren en cómo se crean y administran las instancias comunes de control de formularios (<https://docs.angular.lat/guide/forms-overview>).

## FormControl

Es la unidad más pequeña de un formulario reactivo.

Ejemplo: un cuadro de texto, un calendario, una lista desplegable, etc.

Para configurar un FormControl reactivo, ejecutar los siguientes pasos:

1. Importar la clase FormControl: `import {FormControl} from '@angular/forms';`

2. Declarar el nuevo form control en el archivo .ts:

```
@Component({
  selector: 'app-iniciar-sesion',
  templateUrl: './iniciar-sesion.component.html',
  styleUrls: ['./iniciar-sesion.component.css']
})
export class IniciarSesionComponent implements OnInit {
  mail= new FormControl('', [], {});
  constructor() { }
  ngOnInit(): void {
  }
}
```

3. En el archivo .html, enlazar el form control al template utilizando el property binding [formControl]:

Sintaxis:

[formControl]="variable Form Control"

```
<form class="m-2">
  <div class="form-group">
    <label for="email">Email address:</label>
    <input type="email" [formControl]="mail"
class="form-control" placeholder="Enter email"
id="email">
  </div>
  ...
</form>
```

De esta manera, el control de formulario "mail" y el input type se comunican entre sí. La vista refleja los cambios en el modelo y el modelo refleja los cambios en la vista.

## Form Builder

Dado que crear instancias de FormControl manualmente puede llegar a ser repetitivo, Angular nos provee el servicio FormBuilder.

Para usar este servicio, ejecutar los siguientes pasos:

1. Importar el servicio FormBuilder
2. Inyectar el servicio FormBuilder
3. Generar el contenido del formulario, creando el form group con sus respectivos form controls y enlazando al template.

Ejemplo:

Continuando con nuestro ejemplo, vamos a crear el formulario de inicio de sesión.

Para ello, ejecutar los siguientes pasos:

1. En el archivo iniciar-sesion.component.ts importar la clase:

```
import { FormBuilder, FormGroup } from '@angular/forms';
```
2. Inyectar en el constructor el formBuilder:

```
constructor(private formBuilder: FormBuilder){ }
```

3. En el constructor crear el grupo de controles para el formulario:

```
this.form = this.formBuilder.group({
  password: ['', []],
  mail: ['', []]
});
```

4. En el archivo iniciar-sesion.component.html enlazar el form builder utilizando el property binding [formGroup] y los atributos

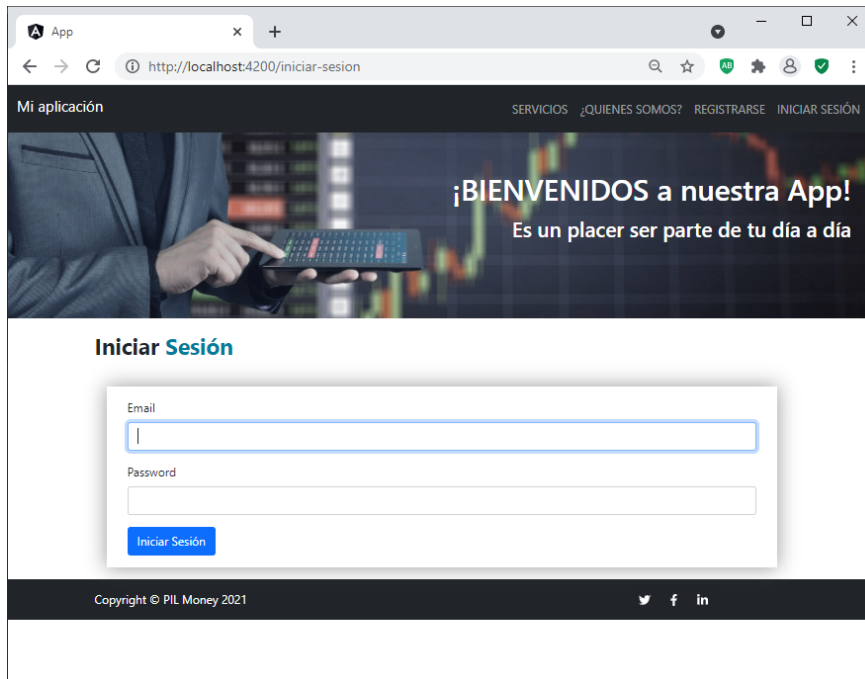
formControlName con sus respectivos form control:

Enlace al servicio FormBuilder

```
<form [formGroup]="form" >
  <div class="m-3">
    <label for="exampleInputEmail1" class="form-label">Email</label>
    <input type="email" class="form-control" formControlName="mail">
  </div>
  <div class="m-3">
    <label for="exampleInputPassword1" class="form-label">Password</label>
    <input type="password" class="form-control" formControlName="password">
  </div>
  <div class="m-3">
    <button type="submit" class="btn btn-primary">Iniciar Sesión</button>
  </div>
</form>
```

Enlace al formControl

5. Ejecutar el comando ng serve (si no está corriendo el servidor), y luego ir a <http://localhost:4200/iniciar-sesion/> para ver el formulario.



The screenshot shows a web browser window with the address bar displaying `http://localhost:4200/iniciar-sesion`. The page has a dark header with the text "Mi aplicación" on the left and navigation links "SERVICIOS", "¿QUIENES SOMOS?", "REGISTRARSE", and "INICIAR SESIÓN" on the right. Below the header is a large banner image of a person in a suit holding a tablet, with the text "¡BIENVENIDOS a nuestra App!" and "Es un placer ser parte de tu día a día". The main content area is titled "Iniciar Sesión" and contains a login form with two input fields: "Email" and "Password". Below the fields is a blue button labeled "Iniciar Sesión". The footer is dark and contains the text "Copyright © PIL Money 2021" and social media icons for Twitter, Facebook, and LinkedIn.

App

http://localhost:4200/iniciar-sesion

Mi aplicación

SERVICIOS ¿QUIENES SOMOS? REGISTRARSE INICIAR SESIÓN

¡BIENVENIDOS a nuestra App!  
Es un placer ser parte de tu día a día

Iniciar Sesión

Email

Password

Iniciar Sesión

Copyright © PIL Money 2021

Twitter Facebook LinkedIn

## Validaciones de Formularios

Angular nos provee la clase `Validators`, la cual contiene una serie de métodos estáticos que nos permiten validar las entradas de datos comunes tales como el formato del email, valores numéricos, máximos y mínimos, cantidad mínima y máxima de caracteres, entre otros.

Métodos estáticos que que provee angular:

- **min()** : realiza el control de que un número sea mayor o igual al número ingresado.
- **max()**: realiza el control de que un número sea menor o igual al número ingresado.
- **required()**: valor requerido.
- **required True()**, valor requerido como verdadero. Utilizado para la casilla de verificación.
- **email()**: valor requerido pase la verificación de valor de mail.
- **minlength()**: requiere que el valor ingresado sea de una longitud mínima según la cantidad de caracteres requeridos.
- **maxlength()**: requiere que el valor ingresado sea de una longitud máxima según la cantidad de caracteres requeridos.
- **pattern()**: requiere que el valor ingresado coincida con un patrón de expresiones regulares.
- **nullValidator()**: no realizará ninguna validación.
- **compose()**: composición de varias validaciones en una sola función.
- **composeAsync()**: composición de varias validaciones asíncronas en una sola función. (<https://docs.angular.lat/api/forms/Validators> )

### Tipo de validaciones

Angular nos provee dos tipos de validaciones:

- **Validadores síncronas**: consisten en funciones síncronas que toman una instancia de control y devuelven inmediatamente un conjunto de errores de validación o un valor nulo.
- **Validadores asíncronas**: consisten en funciones asíncronas que toman una instancia de control y devuelven una Promesa u Observable que

```
22 form: FormGroup = this.formBuilder.group({
23   //pasar los parametros que tengo en el html
24   nombreUser: ['', ],
25   nombreCompleto: ['', ],
26   mail : ['', ],
27   clave: ['', ],
28
29
30 })
```

luego emite un conjunto de errores de validación o nulos.

Validador **síncronos**: validadores que se ejecutan sincrónicamente cuando el usuario carga una letra o apreta un boton

Validador **asíncronos**: se realizarán validaciones a destiempo.

### Validaciones con Form Builder

Siguiendo nuestro ejemplo de inicio de sesión, veamos los pasos para configurar las validaciones de nuestro formulario.

Para ello, ejecutar los siguientes pasos:

1. Importar la clase `Validators`

```
import { Validators } from '@angular/forms';
```
2. En el archivo `iniciar-sesion.component.ts` escribir las validaciones.

En nuestro caso, vamos a trabajar con validaciones síncronas por lo que configuramos entre comas las requeridas para nuestros form controls sabiendo que:

- **mail**, debe ser requerido para el inicio de sesión así como también debe respetar el formato propio del mail.
- **password**, debe ser también requerido para el inicio de sesión y debe tener al menos 8 dígitos.

En el código:

```
this.form= this.formBuilder.group({
  password:['',[Validators.required, Validators.minLength(8)]],
  mail:['', [Validators.required, Validators.email]]
})
```

Luego, crear las propiedades Password y Mail (getter) para acceder luego desde la vista:

```
get Password()
{
  return this.form.get("password");
}

get Mail()
{
  return this.form.get("mail");
}
```

A fin de lograr una mejor experiencia de usuario, en el archivo iniciar-sesion.component.html crear un div inmediatamente después del input type que contendrá los mensajes al usuario. Los mismos, se mostrarán al usuario dependiendo de la validación. Para ello, utilizaremos las directivas \*ngIf.

```
<form [formGroup]="form" >
  <div class="m-3">
    <label for="exampleInputEmail1" class="form-label">Email</label>
    <input type="email" class="form-control" formControlName="mail">
    <div *ngIf="Mail?.errors && Mail?.touched">
      <p *ngIf="Mail?.hasError('required')" class="text-danger">
        El email es requerido.
      </p>
      <p *ngIf="Mail?.hasError('email')" class="text-danger">
        El formato del email debe ser válido.
      </p>
    </div>
  </div>
```

Mail, propiedad (getter) declarada en el componente. Nos permite acceder al formControl.

hasError (). Método que permite evaluar de qué error se trata.

```
<div class="m-3">
  <label for="exampleInputPassword1" class="form-label">Password</label>
  <input type="password" class="form-control" formControlName="password">
  <div *ngIf="Password?.errors && Password?.touched">
    <p *ngIf="Password?.hasError('required')" class="text-danger">
      El password es requerido.
    </p>
    <p *ngIf="Password?.errors?.minlength" class="text-danger">
      El password debe ser de 8 o más caracteres.
    </p>
  </div>
  <div class="m-3">
    <button type="submit" class="btn btn-primary">Iniciar Sesión</button>
  </div>
</form>
```

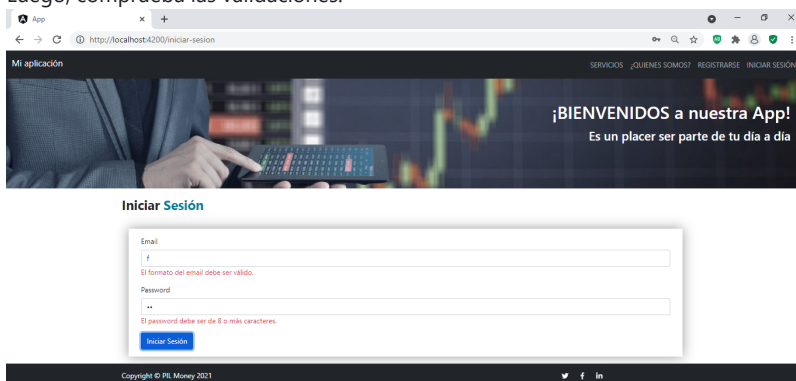
error, propiedad que especifica si el form control tiene errores de validación.

touched, propiedad que especifica si el form control fue tocado.

Observa las propiedades:

- **.touched** Propiedad booleana que especifica si el form control fue tocado por el usuario.
- **.errors** Propiedad booleana que especifica si el formulario tiene errores (falla una o más validaciones).

Ejecutar el comando `ng serve` (si no está corriendo el servidor), y luego ir a <http://localhost:4200/iniciar-sesion/> para ver el formulario. Luego, comprueba las validaciones.



**Nota:** observa que el color rojo de los mensajes está dado porque el párrafo implementa la clase *text-danger* de Bootstrap.



## Validar previo enviar el formulario

Para hacer validaciones previo a enviar los datos al servidor, es buena práctica configurar el evento onSubmit como sigue:

1. En la etiqueta <form> agregar (ngSubmit):

Método que se ejecutará cuando suceda el evento.

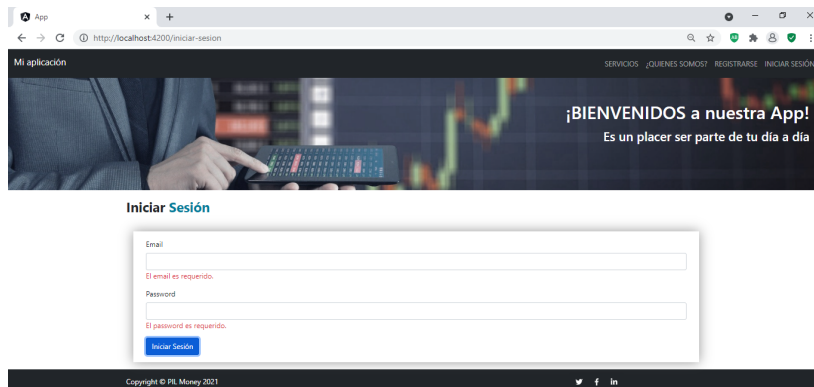
```
<form [formGroup]="form" (ngSubmit)="onEnviar($event)" >
```

2. En el archivo .ts configurar el método onEnviar:

```
onEnviar(event: Event)
{
  // Cancela la funcionalidad por defecto.
  event.preventDefault();

  if (this.form.valid)
  {
    alert ("Enviar al servidor...")
  }
  else
  {
    // Activa todas las validaciones.
    this.form.markAllAsTouched();
  }
}
```

De esta manera, si presionas el botón enviar sin ingresar ningún valor, se ejecutarán todas las validaciones.



## Clases de control de estado

Las clases de estado en Angular son una forma de gestionar el estado de los formularios de manera reactiva.

Angular refleja automáticamente muchas propiedades de control en el FormControl como clases CSS.

Estas clases son:

- **ng-valid:** se aplica cuando el valor del FormControl es válido.
- **ng-invalid:** se aplica cuando el valor del FormControl es inválido.
- **ng-pristine:** se aplica cuando el FormControl no ha sido modificado.
- **ng-dirty:** se aplica cuando el FormControl ha sido modificado.

- **ng-touched:** se aplica cuando el FormControl ha sido tocado.
- **ng-untouched:** se aplica cuando el FormControl no ha sido tocado.

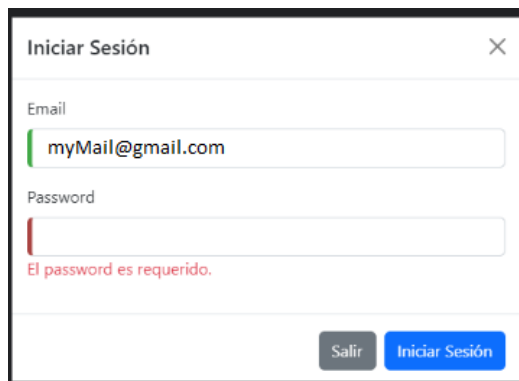
Es decir que, podemos asociar un estilo .css al input type de acuerdo a un estado: válido, inválido, modificado, etc.

Ejemplo: En nuestro formulario de inicio de sesión, podríamos configurar el archivo \*.css del componente o bien, el archivo styles.css el estilo asociado a la validez del dato:

Configura las  
**clases** según el  
estado del  
formulario.

```
.ng-valid:not(form).ng-touched {  
  border-left: 5px solid #42A948; /*  
green */  
}  
  
.ng-invalid:not(form).ng-touched {  
  border-left: 5px solid #a94442; /*  
red */  
}
```

Resultado:



Iniciar Sesión

Email

myMail@gmail.com

Password

El password es requerido.

Salir Iniciar Sesión



**Desafío:** Intenta ahora, utilizando formularios reactivos con sus correspondientes validaciones crear el formulario de registro.