

Angular

10.1. Template driven forms

1. Formularios reactivos vs dirigidos por el modelo

- Angular permite desarrollar formularios utilizando dos paradigmas de programación, las principales características de uno y otro son las siguientes:
- Formularios dirigidos por el modelo (**Template Driven Forms**)
 - Fáciles de usar.
 - Se ajustan adecuadamente en escenarios simples, pero no son útiles para escenarios complejos.
 - Uso de formularios similar al utilizado en AngularJS (predecesor).
 - Manejo automático del formulario y los datos por el propio framework de Angular.
 - La creación de pruebas unitarias en este paradigma es compleja.

1. Formularios reactivos vs dirigidos por el modelo

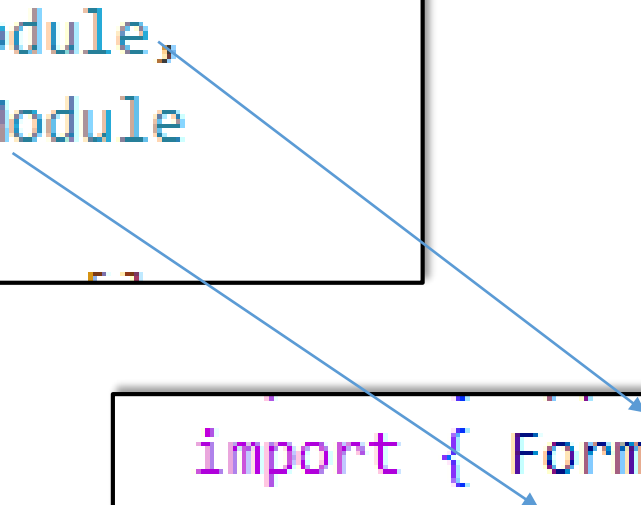
- Formularios reactivos (**Reactive Forms**)
 - Más flexibles.
 - Se ajustan adecuadamente en escenarios complejos.
 - No existe data-binding, sino que se hace uso de un modelo de datos inmutable.
 - Más código en el componente y menos en el lenguaje de marcas HTML.
 - Se pueden realizar transformaciones reactivas, tales como:
 - Manejo de eventos basados en intervalos de tiempo.
 - Manejo de eventos cuando los componentes son distintos hasta que se produzca un cambio.
 - Añadir elementos dinámicamente.
 - Fáciles de testear.

2. Template driven forms

- Para poder utilizar los formularios dirigidos por la plantilla necesitamos agregar en **app.module.ts** un import a **FormsModule** (como cuando utilizábamos two-way data binding) y a **ComoonModule**, ambos con sus import correspondientes.

2. Template driven forms

```
imports: [  
  BrowserModule,  
  AppRoutingModule,  
  FormsModule,  
  CommonModule  
],
```



The diagram consists of two blue arrows. The first arrow originates from the `FormsModule` entry in the `imports` array and points to the `FormsModule` in the `import { FormsModule } from '@angular/forms';` statement. The second arrow originates from the `CommonModule` entry in the `imports` array and points to the `CommonModule` in the `import { CommonModule } from '@angular/common';` statement.

```
import { FormsModule } from '@angular/forms';  
import { CommonModule } from '@angular/common';
```

2. Template driven forms

- Vamos a realizar un formulario de login con un email y un password para ilustrar los template driven forms.
- Generamos una aplicación nueva (**ng new template-driven -standalone=false**)
- Instalamos bootstrap (**npm i --save bootstrap**)
- Creamos el componente login, que es el único que vamos a necesitar, ni modelos ni servicios. (**ng g c Components\login**)
- Modificamos el fichero **app-routing.module.ts** para indicar este hecho.

2. Template driven forms

```
const routes: Routes = [  
  { path: '', component: LoginComponent },  
];
```

2. Template driven forms

- En primer lugar, debemos definir un formulario (form), este formulario tendrá un método asociado (ngSubmit) y un identificador # de tipo ngForm.

```
<form (ngSubmit)="onSubmit(Formulario)" #Formulario="ngForm">
```


2. Template driven forms

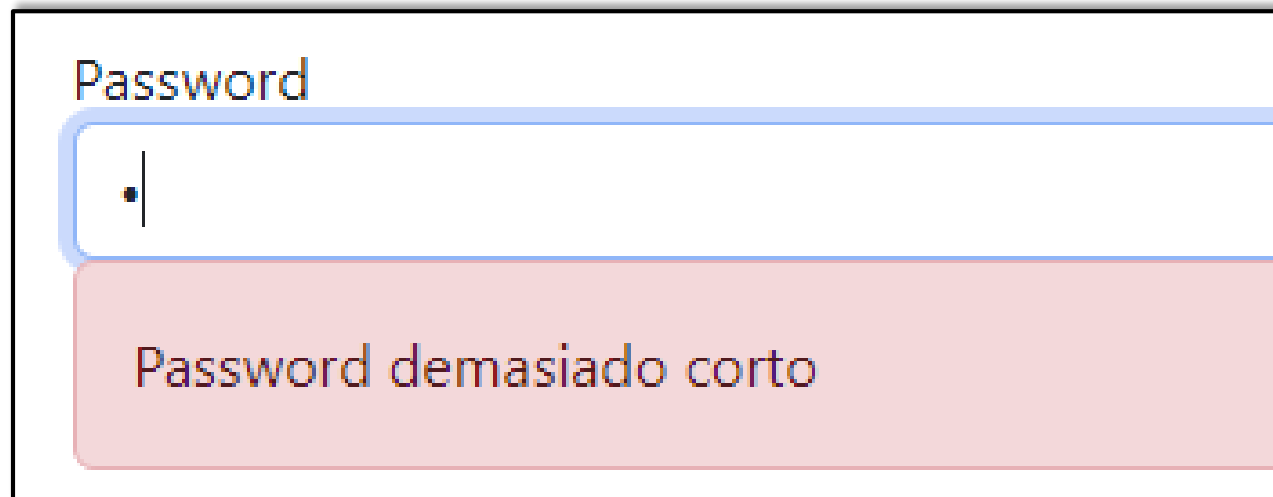
- En estos formularios, el control de los datos se hace desde el template.

```
<label for="name">Password</label>
  <input type="password" class="form-control" id="name"
    required name="password" ngModel
    #password="ngModel" minlength="4">

  <div [hidden]="password.valid || password.pristine"
    class="alert alert-danger">
    Password demasiado corto
  </div>
```

2. Template driven forms

- En caso de que el **password**, controlamos que tenga una longitud **menor** de 4 y **required**.
- Tenemos un input [**hidden**] que se mostrará en caso de que no se cumplan los requisitos.



The image shows a form validation example. It features a label "Password" above a text input field. The input field contains a single character, represented by a small dot and a vertical cursor line. Below the input field, there is a red rectangular box containing the text "Password demasiado corto", which translates to "Password too short".

2. Template driven forms

- El atributo **email.pristine** se activa una vez hacemos cambios en el campo.
- El atributo **email.valid** indica si ha pasado todas las validaciones.

2. Template driven forms

- Para validar emails debemos utilizar el atributo **pattern** de HTML.

```
<label for="email">Email</label>
<input type="email" class="form-control" id="email"
      required name="email" ngModel
      #email="ngModel" pattern="[a-zA-Z0-9_]+(.[a-zA-Z0-9_]+)*@[a-zA-Z0-9_]+(.[a-zA-Z0-9_]+)*.[a-zA-Z]{1,5}">

<div [hidden]="email.valid || email.pristine"
      class="alert alert-danger">
  Email mal formado
</div>
```

2. Template driven forms

Email

Email mal formado

2. Template driven forms

- En este caso el botón de enviar estará deshabilitado hasta que se cumplan todas las condiciones.

```
<button type="submit" class="btn btn-success"  
        [disabled]="!Formulario.form.valid">Submit</button>
```

2. Template driven forms

- En el componente, tendremos un método que se ejecutará cuando el envío del formulario se produzca.

```
//  
export class LoginComponent {  
  
    submitted=false;  
  
    onSubmit(f:NgForm) {  
        this.submitted = true;  
        console.log(f.value.email);  
        console.log(f.value.password);  
    }  
}
```