9 Formularios

Los formularios son los elementos clave de una aplicación, ya que son la forma que tiene el usuario de introducir datos para realizar las funciones más importantes de la misma, como por ejemplo, crear registros o realizar búsquedas.

Angular permite implementar formularios mediante dos modalidades, Template-Driven y Reactive. En esencia, en la primera modalidad, Angular realiza la lógico de captura de datos y validación del lado del template HTML, mientras que en la segunda opción el formulario es gestionado de manera programática en la clase JavaScript del componente.

9.1 Creación de formularios Template-Driven

Vamos a crear un formulario con esta técnica. Para ello en primer lugar, creamos un nuevo componente para añadir nuevos registros de proveedores a nuestra aplicación, completando en la consola:

```
ng g c proveedores/addprovee - - spec false
```

Comenzamos añadiendo en la plantilla del componente, archivo addprovee.component.html el código html de un formulario con los campos que necesitamos:

src/app/proveedores/addprovee/addprovee.component.html

```
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset 2">
    <h2>Añadir nuevo proveedor</h2>
    <form>
      <div class="form-group">
        <label for="nombre">Nombre</label>
        <input type="text"
              class="form-control"
              id="nombre">
      </div>
      <div class="form-group">
        <label for="cif">C.I.F.</label>
        <input type="text"
```

```
        class="form-control"
        id="cif">
</div>
<div class="form-group">
  <label for="direccion">Dirección</label>
  <input type="text"
        class="form-control"
        id="direccion">
</div>
<div class="form-group">
  <label for="cp">Código Postal</label>
  <input type="text"
        class="form-control"
        id="cp">
</div>
<div class="form-group">
  <label for="localidad">Localidad</label>
  <input type="text"
        class="form-control"
        id="localidad" >
</div>
<div class="form-group">
  <label for="provincia">Provincia</label>
  <input type="text"
        class="form-control"
        id="provincia" >
</div>
<div class="form-group">
  <label for="telefono">Teléfono</label>
  <input type="number"
        class="form-control"
        id="telefono" >
</div>
<div class="form-group">
  <label for="email">Correo Electrónico</label>
  <input type="text"
        class="form-control"
        id="email">
</div>
<div class="form-group">
  <label for="contacto">Persona de contacto</label>
```

```
 <input type="text"
              class="form-control"
              id="contacto">
       </div>
     <button type="submit"
                class="btn btn-success">
                Añadir Proveedor</button>
    </form>
   <hr>
  </div>
</div>
```

Ahora vamos añadir el nuevo componente al *routing* en el módulo, añadiendo para ello en el archivo app.module.ts:

src/app/app.module.ts

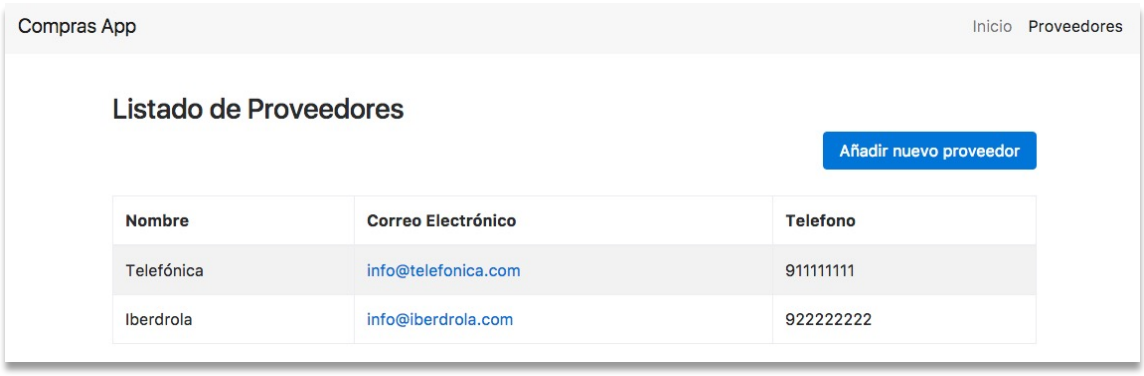
```
…
{ path: 'addprovee', component: AddproveeComponent },
…
```

Y también vamoa a modificar el listado de proveedores, para añadir un botón de acceso al nuevo componente en el que introducir datos, añadiendo en el archivo proveedores.component.html la siguiente línea:

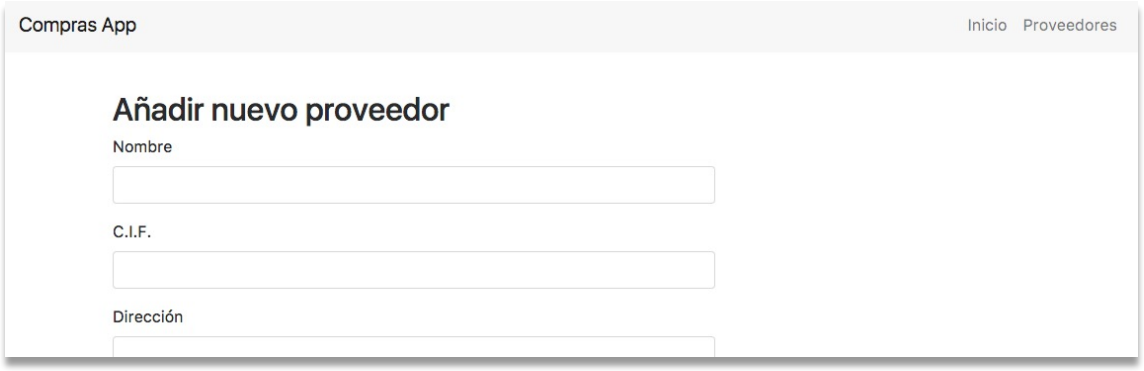
src/app/proveedores/proveedores/proveedores.component.html

```
…
<h1>Listado de Proveedores</h1>
<a class="btn btn-primary float-md-right" routerLink="/addprovee">Añadir nuevo proveedor</a>
<br>
…
```

De esta manera si navegamos al listado de proveedores disponemos de un botón para añadir nuevos proveedores:



Y si lo pulsamos llegamos a la vista del componente addproveedor con el formulario:



De momento el formulario es una simple plantilla HTML que no realiza funcionalidad alguna, por lo que llega el momento de llamar a los paquetes de Angular para poder trabajar con él.

Antes de comenzar a añadir la lógica del formulario, debemos modificar el módulo raíz de la aplicación para añadir la funcionalidad de formularios de Angular.

Por tanto, añadimos al código del archivo app.module.ts:

src/app/app.module.ts

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

y en los import de la clase añadimos:

```
 imports: [
   BrowserModule,
   RouterModule.forRoot(routes),
   FormsModule
 ],
```

El siguiente paso es programar el objeto que recibirá los valores del formulario en nuestro componente, para ello vamos a añadir las siguientes líneas de código en la clase del archivo addprovee.component.ts:

src/app/proveedores/addprovee/addprovee.component.ts

```
…
proveedor: any;

 constructor(){
   this.proveedor = {
     nombre: '',
     cif: '',
     direccion: '',
     cp: '',
     localidad: '',
     provincia: '',
     telefono: null,
     email: '',
```

```
        contacto: ''
    }
  }
…
```

Con esto ya tenemos el objeto proveedor preparado para enlazar con el formulario, así que ahora vamos a incorporar al formulario html el código necesario para enlazar con el componente.

En primer lugar modificamos el inicio de la etiqueta form en el archivo addprovee.component.html:

src/app/proveedores/addprovee/addprovee.component.html

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

En la cual hemos añado una id local llamada formpro para identificar nuestro formulario y lo asociamos a la directiva ngForm de Angular.

Esta directiva hará que los valores del formulario se asignen a un objeto llamado formpro cuando ejecutemos el submit del formulario, que a su vez hará que se ejecute el método onSubmit( ) en el componente.

Además, tenemos que modificar cada campo del formulario, para añadir ngModel y el atributo name con el nombre del campo:

```
…
      <div class="form-group">
        <label for="nombre">Nombre</label>
        <input type="text"
              class="form-control"
              id="nombre"
              ngModel
              name="nombre">
      </div>
…     <! --repetir en todos los componentes -- >
```

A continuación, regresamos al componente para, con los datos de nuestro formulario html, enlazarlos al objeto. De nuevo en addprovee.component.ts modificamos los import:

src/app/proveedores/addprovee/addprovee.component.ts

```
import { Component, OnInit, ViewChild } from '@angular/core';
import { NgForm } from '@angular/forms';
…
```

En la clase, añadimos una vista en la que añadir el objeto del formulario:

```
…
@ViewChild('formpro') formpro: NgForm;
proveedor: any;
…
```

Y posteriormente, después del constructor donde inicializamos el objeto proveedor, añadimos el método onSubmit():

```
…
  onSubmit(){

    this.proveedor.nombre = this.formpro.value.nombre;
    this.proveedor.cif = this.formpro.value.cif;
    this.proveedor.direccion = this.formpro.value.direccion;
    this.proveedor.cp = this.formpro.value.cp;
    this.proveedor.localidad = this.formpro.value.localidad;
    this.proveedor.provincia = this.formpro.value.provincia;
    this.proveedor.telefono = this.formpro.value.telefono;
    this.proveedor.email = this.formpro.value.email;
    this.proveedor.contacto = this.formpro.value.contacto;

    this.formpro.reset();
  }
…
```

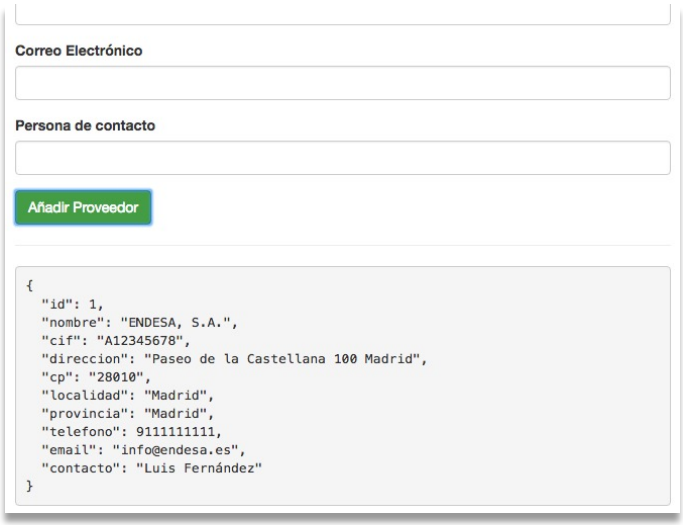
En este método, que se lanza con el botón del formulario, se pasará el valor del objeto formpro y de cada propiedad a su correspondiente en el objeto proveedor del componente.

Finalmente, para comprobar el funcionamiento del formulario, en la vista del archivo addprovee.component.html añadimos después del cierre del formulario, el código para visualizar el objeto que se crea en el formulario, aplicándole el *pipe* json para su presentación "pretty":

src/app/proveedores/addprovee/addprovee.component.html

```
…
    </form>
    <hr>
    <pre>{{ proveedor | json }}</pre>
…
```

Cuando accedemos en el navegador comprobamos que al completar los datos de cada campo, y pulsar en el botón añadir, se muestra el objeto:



9.2 Carga de datos en campos select

Es habitual disponer de campos select en los formularios de una aplicación, así que vamos a comprobar con nuestro ejemplo como implementarlos en Angular.

Supongamos que necesitamos que el campo provincia se complete a través de un campo de tipo select con todas las provincias españolas. Pues gracias a la directiva ngFor podemos crear un array en el componente y recorrerlo en el select para cada opción.

Para llevar a cabo este proceso, en primer lugar añadimos a la clase del componente (antes del constructor) en el archivo addprovee.component.ts el siguiente array:

src/app/proveedores/addprovee/addprovee.component.ts

```
…
  provincias: string[] = [
'Álava','Albacete','Alicante','Almería','Asturias','Ávila','Badajoz','Barcelona',
'Burgos', 'Cáceres', 'Cádiz','Cantabria','Castellón','Ciudad Real','Córdoba',
'La Coruña','Cuenca','Gerona','Granada','Guadalajara',
'Guipúzcoa','Huelva','Huesca','IslasBaleares','Jaén','León','Lérida','Lugo',
'Madrid', 'Málaga','Murcia','Navarra','Orense','Palencia','Las Palmas'
'Pontevedra','La Rioja','Salamanca','Segovia','Sevilla','Soria','Tarragona',
'Santa Cruz de Tenerife', 'Teruel', 'Toledo', 'Valencia', 'Valladolid', 'Vizcaya',
'Zamora','Zaragoza' ]
…
```

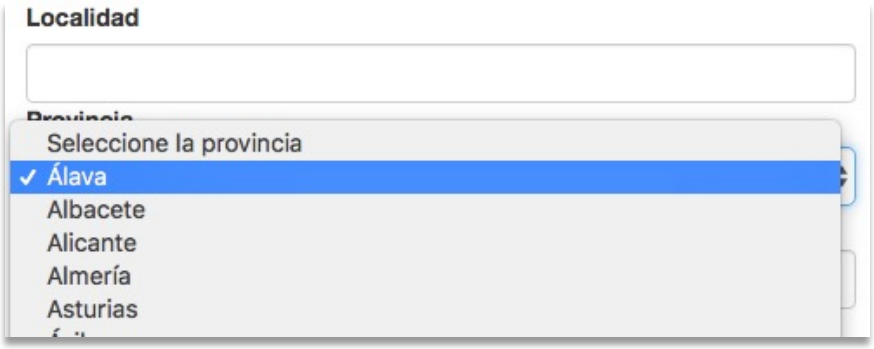
Puedes cambiar este array de provincias o estados por los de tu país, buscándolos en google como “array javascript de…”

Y modificamos el campo de provincias en la vista, addprovee.component.html de la siguiente forma:

src/app/proveedores/addprovee/addprovee.component.html

```
…
<div class="form-group">
        <label for="provincia">Provincia</label>
        <select class="form-control"
                id="provincia"
                ngModel
                name="provincia">
                <option value="">Seleccione la provincia</option>
                <option *ngFor="let provincia of provincias"
                [value]="provincia">{{provincia}} </option>
        </select>
</div>
…
```

Y así, ya tendremos disponible todas las provincias como podemos comprobar en el navegador:



9.3 Validación de campos mediante HTML

Angular permite realizar una validación de campos de formularios que en el caso de emplear la técnica anterior, Template Driven, utiliza la validación nativa de HTML.

Gracias a los estados de cada campo, Angular permite implementar clases CSS y elementos HTML dinámicos de ayuda al usuario, para completar los formularios.

Para aprender esta funcionalidad, vamos a comenzar por añadir al campo email los atributos HTML5 required e email en el archivo addprovee.component.html:

src/app/proveedores/addprovee/addprovee.component.html

```
…
      <div class="form-group">
        <label for="email">Correo Electrónico</label>
        <input type="text"
              class="form-control"
              id="email"
              ngModel
              name="email"
              required
              email>
      </div>
…
```

De esta manera, ahora el campo de Correo Electrónico será obligatorio y también deberá cumplir la sintaxis de dirección de correo electrónico.

Por defecto, Angular elimina la validación HTML nativa por lo que los anteriores atributos no impedirán que empleemos el botón submit y tampoco lanzarán mensaje alguno.

Pero de momento, nos sirven para identificar los estados por los que pasa un campo en un formulario Angular implementados a través de ngForm.

De estos estados, almacenados en el objeto del formulario, nos vamos a fijar en *pristine* (el usuario aún no ha modificado el campo desde su carga), *dirty* (el usuario ha modificado el campo) , *touched* (el usuario ha abandonado el campo) y *valid* e *invalid* (el valor del campo cumple o incumple la regla de validación).

Para ver el estado del campo Email según operemos en el formulario, añadimos a la vista, antes del cierre del form, el siguiente código:

src/app/proveedores/addprovee/addprovee.component.html

```
…
    <hr>
    <pre> Estado Dirty: {{ formpro.controls.email.dirty }}</pre>
    <pre> Estado Pristine: {{ formpro.controls.email.pristine }}</pre>
    <pre> Estado Touched: {{ formpro.controls.email.touched }}</pre>
    <pre> Estado Valid: {{ formpro.controls.email.valid }}</pre>
    <pre> Estado Invalid: {{ formpro.controls.email.invalid }}</pre>
  </form>
…
```

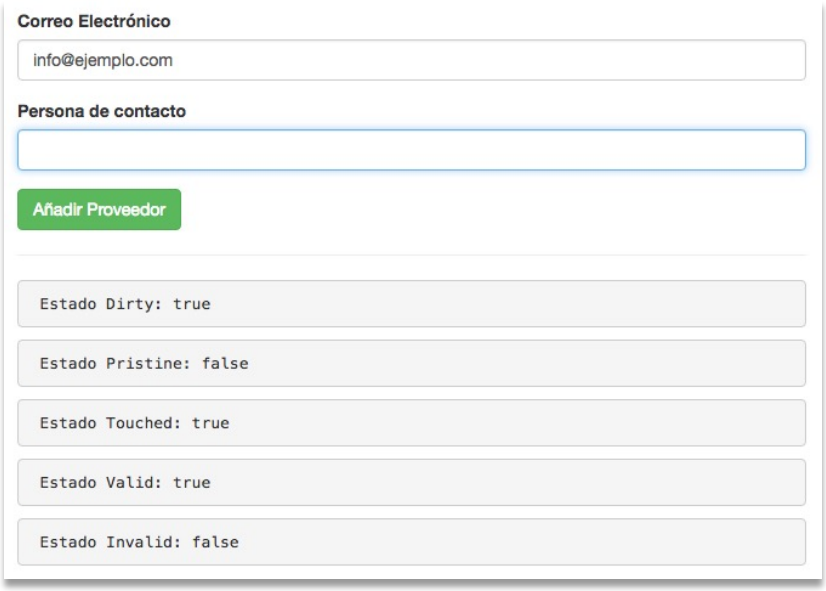
Este código nos muestra el valor de cada estado del campo email. Al iniciar el formulario, el valor del estado *pristine* será *true* (aún no hemos escrito nada) y el de *invalid* y valid, *true* y *false* respectivamente, ya que aún no está validado.

Cuando comencemos a escribir, *dirty* pasará de *false* a *true* y al revés *pristine*.

Por otra parte, al abandonar el foco el estado *touched* pasará a *true*.

Finalmente, si introducimos un valor de correo electrónico correcto, el estado *valid* pasará a *true* y el *invalid* a *false*.

Podemos comprobar en el navegador, los cambios a medida que escribimos:



¿Para qué usar estos estados? Pues por ejemplo, para el uso de clases CSS dinámicas en función del estado que ayuden al usuario a completar el formulario.

Vamos a ponerlo en práctica. En el archivo de estilos, addprovee.component.css, añadimos una serie de clases CSS de Angular (las que comienzan por ng) relacionadas los estados del campo:

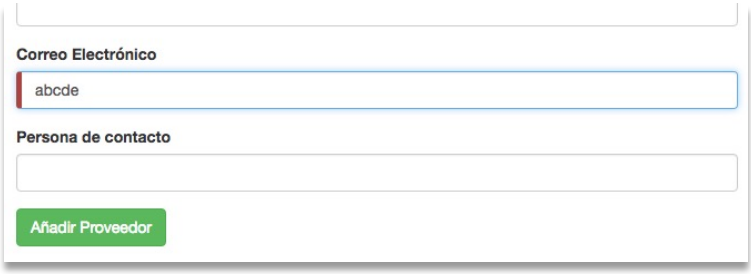
src/app/proveedores/addprovee/addprovee.component.html

```
input.ng-invalid.ng-touched {
    border-left: 5px solid #a94442;
}

input.ng-valid.ng-dirty {
    border-left: 5px solid #42A948;
}
```

Que implicará en primer lugar, que cuando el campo tenga los estados inválido y touched, es decir que no sea ha introducido correctamente el valor y se ha pasado al siguiente, se mostrara un borde rojo de 5 px en la izquierda del campo.

En cambio, si el campo se ha comenzado a completar y se valida, se mostrará un borde verde de 5 px.

Lo podemos comprobar en nuestro navegador:



También podemos añadir más ayudas visuales con un icono junto al label del campo y un texto de advertencia empleando la directiva ngIf y los estados.

Por ejemplo, para el campo mail, en el archivo addprovee.component.html, sustituimos su div por el siguiente:

src/app/proveedores/addprovee/addprovee.component.html

```
…
<div class="form-group">
        <label for="email">Correo Electrónico</label>
         <i class="fa fa-check-circle check"                    ①
            *ngIf="email.valid"></i>
        <i class="fa fa-exclamation-circle uncheck"
            *ngIf="email.invalid && email.touched"></i>
        <input type="text"
              class="form-control"
              id="email"
              ngModel
              name="email"
              required
              email
              #email="ngModel">     ②
        <p class="alert alert-danger" *ngIf="email.invalid && email.touched"> ③
          Por favor introduzca una dirección de correo correcta.
        </p>
      </div>
…
```

Donde hemos ① introducido dos iconos que se mostrarán en función de la expresión de su ngIf, hemos ② creado la id local email asociándola a ngModel y hemos ③ creado un texto de alerta que se mostrará también de acuerdo a la expresión ngIf.

Añadimos las clases CSS de los iconos al archivo addprovee.component.css:

src/app/proveedores/addprovee/addprovee.component.css

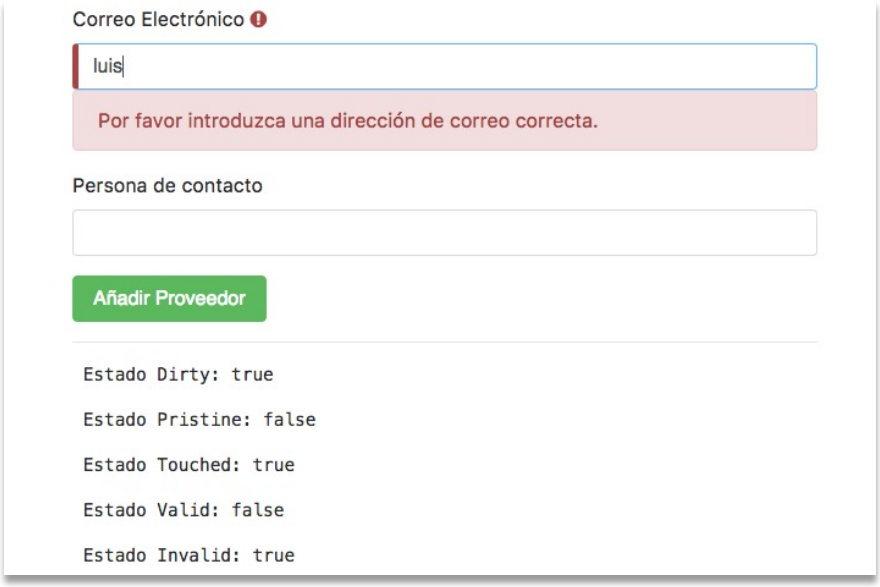
```
…
.check {
    color: #42A948;
}

.uncheck {
    color: #a94442;
}
…
```

Y la refencia al CDN de *Font Awesome Icons* en el head del index.html para obtener los iconos:

src/index.html

```
…
  <script src="https://use.fontawesome.com/bacad173cf.js"></script>
…
```

Finalmente, comprobamos en el navegador:



Para finalizar la validación de campos de un formulario, podemos añadir la lógica necesaria para que el botón de envío solo se active cuando todos los campos estén en estado valid.

Para ello, simplemente modificamos el botón submit en el formulario en el archivo addprovee.component.html de la siguiente forma:

src/app/proveedores/addprovee/addprovee.component.html

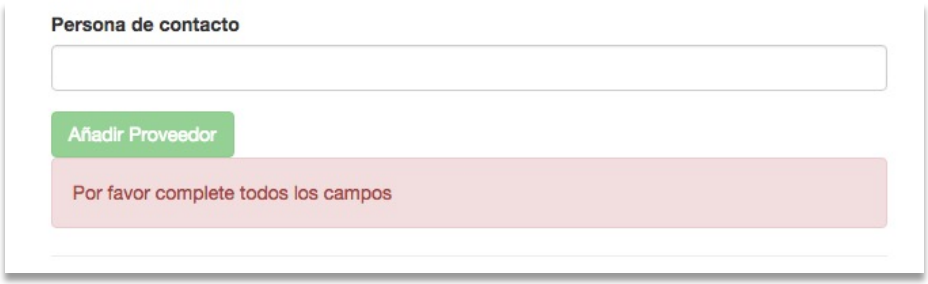
```
…
    <button type="submit"
          class="btn btn-success"
          [disabled]="!formpro.valid" ①
          >Añadir Proveedor</button>
    <p class="alert alert-danger " *ngIf="!formpro.valid">  ②
      Por favor complete todos los campos
    </p>
…
```

En el cual:

① Añadimos el atributo disabled mientras todo el formulario no sea válido.

② Y un texto de advertencia con un ngIf.

También añadiremos los iconos junto al label y el atributo required al resto de campos y podremos comprobar en el navegador el funcionamiento de todo el formulario.



Como tal, nuestro formulario está listo pero los datos quedan almacenados en el objeto proveedor en memoria.

Aprenderemos más adelante, como almacenar de manera persistente los datos mendiante la conexión de la aplicación Angular a un servidor de base de datos.

9.4 Creación de formularios Reactive

Otra de las técnicas para crear formularios es Reactive, que lleva la generación y gestión del mismo del lado del archivo TypeScript del componente.

Para implementar esta técnica en nuestra aplicación, en primer lugar vamos a crear un nuevo componente llamado presupuestos. Una vez más, desde el directorio del proyecto en la consola, completamos:

```
ng g c presupuestos/addpres  - -spec false
```

A continuación incorporamos el nuevo component a nuestro routing y el paquete de Angular para crear formularios reactivos, para lo cual lo añadimos en el archivo app.module.ts de la siguiente forma:

src/app/app.module.ts

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

Y:

```
…
{ path: 'addpres', component: AddpresComponent},
…

…
  imports: [
    BrowserModule,
    RouterModule.forRoot(routes),
    FormsModule,
    ReactiveFormsModule
  ],
…
```

Para poder acceder a este nuevo componente en el navegador, añadimos el link al menú de cabecera en el archivo header.component.html, de la siguiente manera:

src/app/header/header.component.html

```
…
<ul class="nav navbar-nav navbar-right">
    <li class="nav-item" routerLinkActive="active"
                [routerLinkActiveOptions]="{exact:true}">
          <a routerLink="/" >Inicio </a>
    </li>
    <li class="nav-item"routerLinkActive="active">
          <a routerLink="/proveedores">Proveedores</a>
    </li>
    <li class="nav-item" routerLinkActive="active">
          <a routerLink="/addpres">Añadir Presupuesto</a>
    </li>
</ul>
…
```

Dejando listo el componente para ser empleado.

Vamos a comenzar a crear nuestro formulario. En primer lugar añadimos el siguiente formulario HTML a nuestro código en el archivo addpres.component.html:

src/app/presupuestos/addpres/addpres.component.html

```
<div class="row">
  <div class="col-xs-12 col-sm-10 col-md-8 col-sm-offset-1 col-md-offset 2">
    <h2>Añadir nuevo presupuesto</h2>
    <form>
       <div class="form-group">
         <label for="proveedor">Proveedor</label>
         <input type="text"
               class="form-control"
               id="proveedor">
       </div>
```

```
<div class="form-group">
  <label for="fecha">Fecha Presupuesto</label>
  <input type="date"
        class="form-control"
        id="fecha">
</div>
<div class="form-group">
  <label for="concepto">Concepto</label>
  <input type="text"
        class="form-control"
        id="concepto">
</div>
<div class="form-group">
  <label for="base">Base Imponible</label>
  <input type="number"
        class="form-control"
        id="base">
</div>
<div class="form-group">
  <label for="tipo">Tipo de IVA</label>
  <select class="form-control"
         id="tipo">
         <option value="">Seleccione...</option>
         <option value=0> 0 %</option>
         <option value=0.04> 4 %</option>
         <option value=0.10>10 %</option>
         <option value=0.21>21 %</option>
  </select>
</div>
<div class="form-group">
  <label for="iva">Importe IVA</label>
  <input type="number"
        class="form-control"
        id="iva">
</div>
<div class="form-group">
  <label for="total">Total Factura IVA Incluido</label>
  <input type="number"
        class="form-control"
        id="total">
</div>
<button class="btn btn-primary"
```

```
                type="submit">Añadir Presupuesto</button>
    </form>
   </div>
</div>
```

Ahora comenzamos a añadir la lógica a nuestro componente. Para ello, en el archivo addpres.component.ts, comenzamos modificando las importaciones:

src/app/presupuestos/addpres/addpres.component.ts

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

Las cuales, nos permitirán gestionar el formulario desde el componente.

A continuación, dentro de la clase añadimos la propiedad presupuestoForm que será del tipo FormGroup y el objeto presupuesto:

```
  …
  presupuestoForm: FormGroup;
  presupuesto: any;
  …
```

Seguidamente en el constructor, para generar el formulario, creamos un objeto de nombre, por ejemplo, pf de la clase FormBuilder de Angular:

```
…
constructor(private pf: FormBuilder) { }
…
```

Continuamos en la clase, y dentro de ngOnInit igualamos presupuestoForm al objeto pf y inicializamos dentro de este y vaciós, los campos que tendrá el formulario:

```
…
ngOnInit() {
   this.presupuestoForm = this.pf.group({
     proveedor: '',
     fecha: '',
     concepto: '',
     base: '',
     tipo: '',
     iva: '',
     total: ''
   });
 }
…
```

Que serán los mismos establecidos en el formulario HTML del archivo addpres.component.html. Regresamos a este archivo y añadimos en el inicio de la etiqueta form:

src/app/presupuestos/addpres/addpres.component.html

```
…
<form [formGroup]="presupuestoForm">
…
```

Y en cada campo añadimos el atributo formControlName con el nombre del campo, por ejemplo:

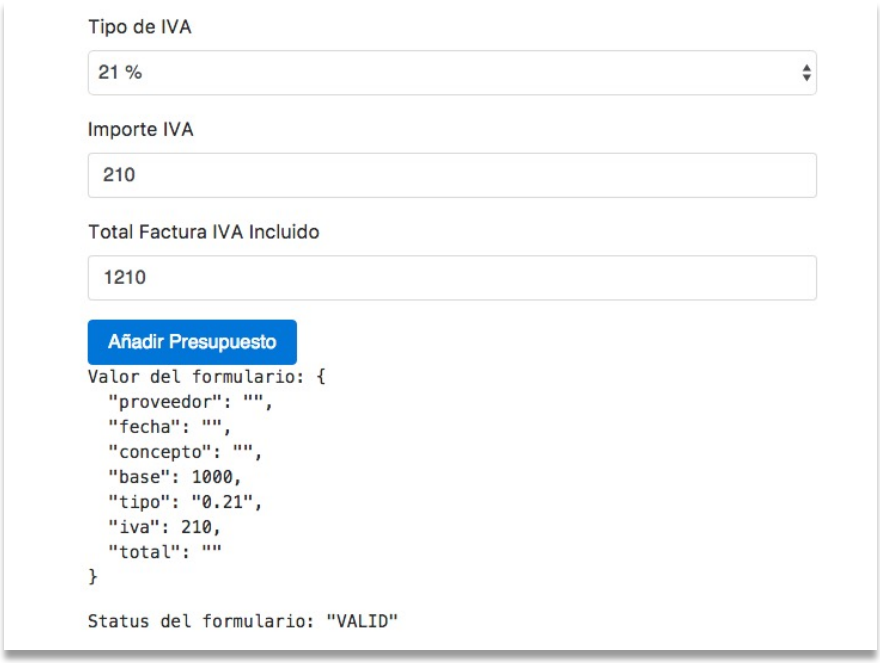
```
…
      <div class="form-group">
        <label for="proveedor">Proveedor</label>
        <input type="text"
              class="form-control"
              id="proveedor"
```

```
            formControlName="proveedor">
        </div>
… <!--repetir para cada campo-->
```

Para visualizar la toma de datos del formulario, añadimos tras el cierre del form las siguientes dos líneas:

```
…
    <pre>Valor del formulario: {{ presupuestoForm.value | json }}</pre>
    <pre>Status del formulario: {{ presupuestoForm.status | json }}</pre>
…
```

Ahora vamos al navegador y comprobamos como al completar campos se actualizan automáticamente los valores del formulario en el objeto presupuestoForm:



A continuación vamos a añadir un método onSubmit, para poder guardar los datos del formulario en el objeto presupuesto del componente.

Para ello, volvemos a modificar el inicio de la etiqueta form en el archivo addpres.component.html de la siguiente forma:

src/app/presupuestos/addpres/addpres.component.html

```
…
   <form [formGroup]="presupuestoForm" (ngSubmit)="onSubmit()">
…
```

Y tras el cierre de la etiqueta form, modificamos:

```
…
   </form>
   <hr>
   <pre>Valor del formulario: {{ presupuestoForm.value | json }}</pre>
   <pre>Status del formulario: {{ presupuestoForm.status | json }}</pre>
   <hr>
   <pre>{{ presupuesto | json }}</pre>
…
```

En el componente, dentro de la clase, en el archivo addpres.component.ts añadimos el siguiente código:

src/app/presupuestos/addpres/addpres.component.ts

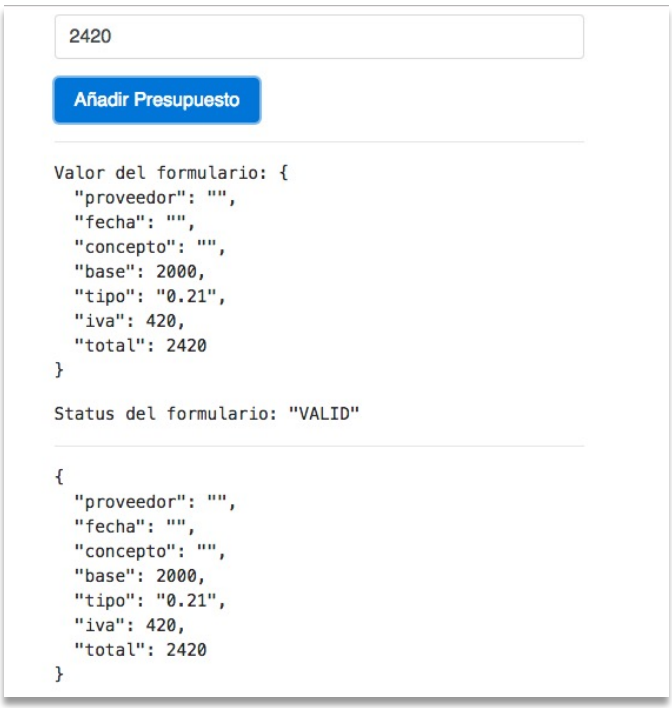
```
…
 onSubmit() {
   this.presupuesto = this.savePresupuesto();
 }
…
```

En el que definimos onSubmit() como el método que iguala el objeto presupuesto del componente con otro método savePresupuesto que definimos a continuación:

```
…
 savePresupuesto() {
   const savePresupuesto = {
     proveedor: this.presupuestoForm.get('proveedor').value,
     fecha: this. presupuestoForm.get('fecha').value,
     concepto: this.presupuestoForm.get('concepto').value,
     base: this.presupuestoForm.get('base').value,
     tipo: this.presupuestoForm.get('tipo').value,
     iva: this.presupuestoForm.get('iva').value,
     total: this.presupuestoForm.get('total').value
   };
   return savePresupuesto;
 }
…
```

Método que iguala cada valor del formulario con su correspondiente propiedad del objeto del componente y lo devuelve con un return.

Ahora en el navegador, podemos comprobar como los datos introducidos en el formulario son guardados en el objeto presupuesto cuando pulsamos en el botón añadir presupuesto:



9.5 Validación de campos programática

Mediante este procedimiento de formularios reactivos, Angular permite añadir validación programática, es decir, independiente de los atributos de validación HTML5.

Vamos a comprobarlo en nuestro ejemplo, para lo cual en el componente en el archivo addpres.component.ts añadimos Validators:

src/app/presupuestos/addpres/addpres.component.ts

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

A continuación la sintaxis para añadir validaciones en formularios reactivos es, dentro del método createForm, crear un array de cada campo con el valor de inicialización, y la propiedad o propiedades de validación mediante la clase de Angular forms Validators.

Por ejemplo, añadimos a los campos que necesitemos que sean obligatorios la clase required:

```
…
ngOnInit() {
  this.presupuestoForm = this.pf.group({
    proveedor: ['', Validators.required ],
    fecha: ['', Validators.required ],
    cif: ['', Validators.required ],
    concepto: ['', Validators.required ],
    base: ['', Validators.required ],
    tipo: ['', Validators.required ],
    iva: ['', Validators.required ],
    total: ['', Validators.required ]
  });
 }
…
```

Ahora en la vista, archivo addpres.component.html, modificamos el código del botón para que solo se muestre cuando el formulario sea válido:

src/app/presupuestos/addpres/addpres.component.html

```
…
      <button type="submit"
          class="btn btn-primary"
          [disabled]="!presupuestoForm.valid"
          >Añadir Presupuesto</button>
      <p class="alert alert-danger" *ngIf="!presupuestoForm.valid">
        Por favor complete todos los campos
      </p>
…
```

Y en el archivo addpres.component.css copiamos las clases CSS para validar, empleadas en el archivo addprovee.component.css del apartado anterior.

Comprobamos el funcionamiento en el navegador.

Además de la obligatoriedad de completar el campo, tenemos otras clases de validación, por ejemplo longitud mínima.

Vamos a comprobarlo forzando a que el campo concepto tenga un mínimo de 10 caracteres. Para ello modificamos en el archivo addpres.component.ts la siguiente línea:

src/app/presupuestos/addpres/addpres.component.ts

```
…
concepto: ['', [ Validators.required, Validators.minLength(10)]],
…
```

Que convierte el campo en obligatorio y con un mínimo de 10 caracteres para que pase a estado válido.

El resto de clases de validación de Angular se pueden encontrar en el siguiente enlace de su documentación:

https://angular.io/api/forms/Validators

Para finalizar nuestro formulario, podemos implementar para cada campos los iconos y textos de validación del apartado anterior. Pero en este caso el objeto utilizado para obtener el estado del campo tiene una sintaxis diferente, siendo el nombre del formulario seguido de controls y el nombre del campo.

Cambiamos en cada campo del archivo addpres.component.html el código por el siguiente:

src/app/presupuestos/addpres/addpres.component.html

```
<div class="form-group">
  <label for="proveedor">Proveedor</label>
  <i class="fa fa-check-circle check"
      *ngIf="presupuestoForm.controls.proveedor.valid"></i>
  <i class="fa fa-exclamation-circle uncheck"
      *ngIf="presupuestoForm.controls.proveedor.invalid &&
      presupuestoForm.controls.proveedor.touched"></i>
  <input type="text"
        class="form-control"
        id="proveedor"
        formControlName="proveedor">
  <p class="alert alert-danger"
        *ngIf="presupuestoForm.controls.proveedor.invalid &&
        presupuestoForm.controls.proveedor.touched">
    El campo Proveedor es obligatorio.
  </p>
</div>
```

9.6 valueChanges

Otra de las enormes ventajas que proporciona la gestión de formularios reactive es la posibilidad de observar los cambios que se produzcan en los campos del formulario, lo que permite realizar formularios totalmente dinámicos.

En el ejemplo anterior, tenemos dos campos, importe del IVA y total que en la vida real se obtienen a partir del producto de la base por el tipo de IVA y la suma de la base más el IVA respectivamente.

Vamos a ver como podemos incorporar las funciones aritméticas a nuestro código TypeScript y, lo más importante, como Angular detecta cambios en el valor de un campo para actualizar los valores de otros de manera automática.

Para ello en primer lugar, vamos a modificar la clase del componente en el archivo addpres.component.ts de la siguiente manera:

src/app/presupuestos/addpres/addpres.component.ts

```
…
export class AddpresComponent implements OnInit {

  presupuestoForm: FormGroup;
  presupuesto: any;
  base: any;
  tipo: any;
  iva: any = 0;
  total: any = 0;
…
```

Donde al comienzo, hemos creado las propiedades base, tipo, iva y total, de las cuales las dos últimas se inicializan a cero.

A continuación, modificamos en ngOnInit ① los campos iva y total, para igualarlos a las propiedades anteriores, tambien creamos ② el método onChanges():

```
…
ngOnInit() {
this.presupuestoForm = this.pf.group({
      proveedor: ['', Validators.required ],
      fecha: ['', Validators.required ],
      concepto: ['', [ Validators.required, Validators.minLength(10)] ],
      base: ['', Validators.required ],
      tipo: ['', Validators.required ],
      iva: this.iva ,                                    ①
      total: this.total
    });
this.onChanges();                                        ②
}
…
```

Nos queda finalmente añadir el método onChanges:

```
…
onChanges(): void {
      this.presupuestoForm.valueChanges.subscribe(valor => {        ①
        this.base = valor.base;          ②
        this.tipo = valor.tipo;
        this.presupuestoForm.value.iva = this.base * this.tipo;                    ③
        this.presupuestoForm.value.total = this.base + (this.base * this.tipo);
      });
  }
…
```

En cual:

① Utilizamos el observable valueChanges y nos sucribimos a el para obtener el objeto valor, que se actualizará cada vez que se produzca un cambio en algún campo del formulario.

② Igualamos la propiedad base y tipo a cada campo correspondiente.

③ Y establecemos el valor de iva y total en el formulario mediante una fórmula aritmética de los dos valores anteriores.

Una vez modificado el componente, nos queda cambiar la vista en archivo addpres.component.ts de los campos iva y total de la siguiente manera:

src/app/presupuestos/addpres/addpres.component.html

```
…
<div class="form-group">
  <label for="iva">Importe IVA</label>
  <input type="number"
        class="form-control"
        id="iva"
        formControlName="iva"
        [(ngModel)]="presupuestoForm.value.iva" ①
        disabled> ②
 </div>
 <div class="form-group">
  <label for="total">Total Factura IVA Incluido</label>
  <input type="number"
        class="form-control"
        id="total"
        formControlName="total"
        [(ngModel)]="presupuestoForm.value.total"
        disabled>
 </div>
…
```

En el cual:

① Añadimos la directiva ngModel asociada al valor en el formulario de cada campo.

② Eliminamos los iconos y textos de advertencia de validación y añadimos el atributo disabled para que el campo no sea accesible.

Con estos pasos nuestro formulario queda finalizado y podemos comprobar en el navegador como los campos iva y total se actualizan automáticamente cada vez que introducimos los datos necesarios para su cálculo.