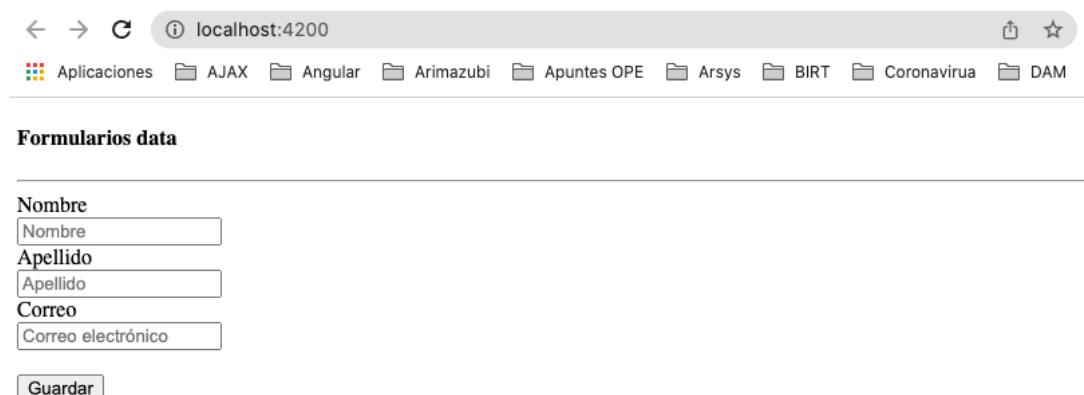


1. **Inicio del proyecto:** nos lo bajamos desde el curso de moodle y lo abrimos. Instalamos las dependencias de node:

```
$ npm install
```

Lanzamos el proyecto:

```
$ ng serve -o
```



Instalamos Bootstrap:

```
jsersan@iMac-de-Jose formReactivo % npm install bootstrap jquery @popperjs/core
added 3 packages, removed 1 package, and audited 916 packages in 4s
105 packages are looking for funding
  run `npm fund` for details
4 moderate severity vulnerabilities

To address all issues, run:
  npm audit fix --force

Run `npm audit` for details.
jsersan@iMac-de-Jose formReactivo %
```

Modificamos el archivo angular.json:

```
101   "styles": [
102     "node_modules/bootstrap/dist/css/bootstrap.min.css",
103     "src/styles.scss"
104   ],
105   "scripts": [
106     "node_modules/jquery/dist/jquery.min.js",
107     "node_modules/@popperjs/core/dist/umd/popper.min.js",
108     "node_modules/bootstrap/dist/js/bootstrap.min.js"
109   ]
```

En index.html incorporamos las librerías de Angular:

```

7   <meta name="viewport" content="width=device-width, initial-scale=1">
8   <link rel="icon" type="image/x-icon" href="favicon.ico">
9   <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/css/bootstrap.min.css" integrity="sha384-PsH8R72JQ3SOaIeqoIgjsP0WZMzjiX4OYqL7QHxU1Z00yQ6JGmOoEoQ+enQWlJ" crossorigin="anonymous">
10  <script src="https://code.jquery.com/jquery-3.2.1.slim.min.js" integrity="sha384-KJ3o2DKtIk" crossorigin="anonymous">
11  <script src="https://cdn.jsdelivr.net/npm/popper.js@1.12.3/dist/umd/popper.min.js" integrity="sha384-eLtdVQ5EOrJS7RDIyVE9c3qF7E4kWW46sd2UxqoA9fVXUJ6U9zZrE" crossorigin="anonymous">
12  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.0.0-beta.2/js/bootstrap.min.js" integrity="sha384-vZ2WRJMwsq5D4qZkEsrdKyn6sVChQoZmcC4N4OvqBjF5Dk4ZK+Hx" crossorigin="anonymous">
-- 

```

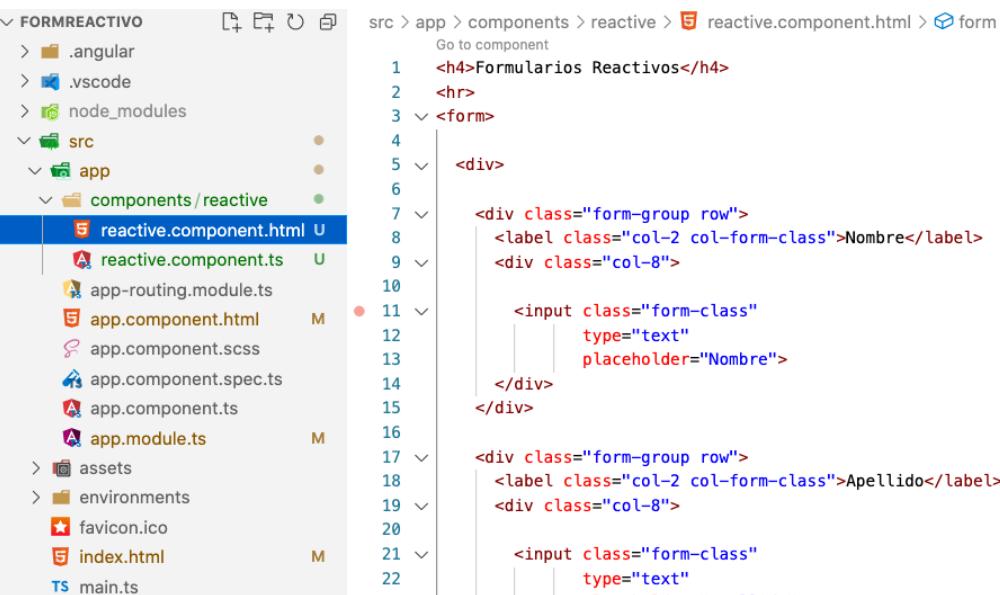
Creamos el componente reactive:

```

jsersan@iMac-de-Jose formReactiva % ng g c components/reactive --skip-tests
CREATE src/app/components/reactive/reactive.component.html (23 bytes)
CREATE src/app/components/reactive/reactive.component.ts (257 bytes)
UPDATE src/app/app.module.ts (603 bytes)
jsersan@iMac-de-Jose formReactiva %

```

De este componente ReactiveComponent el template es el html adjunto:



```

src > app > components > reactive > reactive.component.html > form
Go to component
1 <h4>Formularios Reactivos</h4>
2 <hr>
3 <form>
4   <div>
5     <div class="form-group row">
6       <label class="col-2 col-form-class">Nombre</label>
7       <div class="col-8">
8         <input class="form-class" type="text" placeholder="Nombre">
9       </div>
10      </div>
11    <div class="form-group row">
12      <label class="col-2 col-form-class">Apellido</label>
13      <div class="col-8">
14        <input class="form-class" type="text" placeholder="Apellido">
15      </div>
16    </div>
17  </div>
18
19
20
21
22
-- 

```

En app.module.ts importamos FormsModule y ReactiveFormsModule:

```

src > app > app.module.ts > AppModule
1 import { NgModule } from '@angular/core';
2 import { FormsModule, ReactiveFormsModule } from '@angular/forms';
3 import { BrowserModule } from '@angular/platform-browser';
4
5 import { AppRoutingModule } from './app-routing.module';
6 import { AppComponent } from './app.component';
7 import { ReactiveComponent } from './components/reactive/reactive.component';
8
9 @NgModule({
10   declarations: [
11     AppComponent,
12     ReactiveComponent
13   ],
14   imports: [
15     BrowserModule,
16     AppRoutingModule,
17     FormsModule,
18     ReactiveFormsModule
19   ],
-- 

```

Para tener esta configuración, en el componente reactive creado anteriormente copiamos el contenido del html adjunto en el template:

```

7   <div class="form-group row">
8     <label class="col-2 col-form-class">Nombre</label>
9     <div class="col-8">
10
11       <input class="form-control"
12         type="text"
13         placeholder="Nombre">
14     </div>
15   </div>
16
17   <div class="form-group row">
18     <label class="col-2 col-form-class">Apellido</label>
19     <div class="col-8">
20
21       <input class="form-control"
22         type="text"
23         placeholder="Apellido">
24     </div>
25   </div>
26
27 </div>
28

```

En app.component.html:

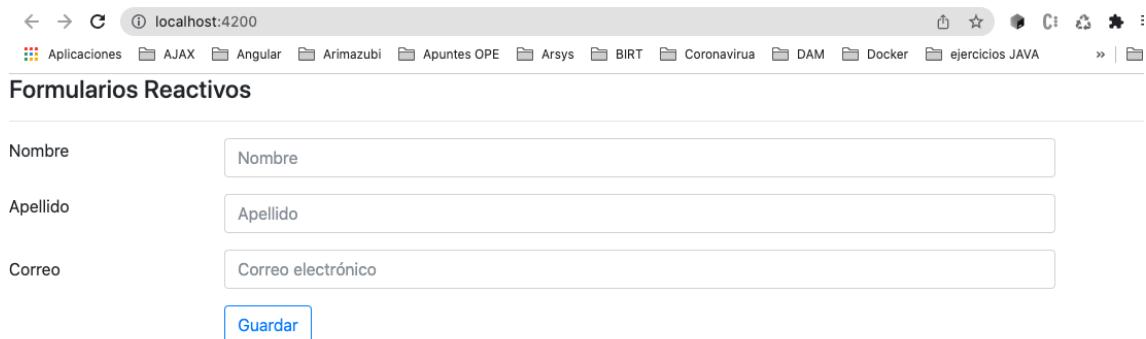


```

src > app > app.component.html > ...
1 <app-reactive></app-reactive>
2
3 <router-outlet></router-outlet>
4

```

Resultado:



A partir de ahora, trabajaremos en el componente. Creamos un objeto forma de tipo FormGroup:

```
2 import { FormGroup } from '@angular/forms';
3
4 @Component({
5   selector: 'app-reactive',
6   templateUrl: './reactive.component.html',
7   styles: [
8   ]
9 })
10 export class ReactiveComponent implements OnInit {
11
12   forma: FormGroup
```

Vamos a llenar de contenido este formulario en el constructor a través de una función denominada crearFormulario:

```
10 export class ReactiveComponent implements OnInit {
11
12   forma: FormGroup
13
14   constructor() {
15     this.crearFormulario();
16   }
17
18   ngOnInit(): void {
19   }
20
21   crearFormulario(){}
--
```

Creamos esta función:

```
12 forma!: FormGroup;
13
14 constructor(private fb: FormBuilder) {
15   this.crearFormulario();
16 }
17
18 ngOnInit(): void {
19 }
20
21 crearFormulario(){
22
23   this.forma = this.fb.group({
24     nombre: ['',],
25     apellido: ['',],
26     correo: ['']
27   });
28
29 }
```

El resultado sigue siendo el mismo:

A screenshot of a web browser window titled "localhost:4200". The page has a header "Formularios Reactivos". Below it is a form with three input fields: "Nombre" (placeholder "Nombre"), "Apellido" (placeholder "Apellido"), and "Correo" (placeholder "Correo electrónico"). Below the inputs is a blue "Guardar" button.

En el template añadimos propiedades al formulario:

```
 1 <h4>Formularios Reactivos</h4>
 2 <hr>
 3 <form autocomplete="off" [formGroup]="forma" (ngSubmit)="guardar()">
 4   <div>
```

Creamos en el componente la función guardar():

```
31   guardar(){
32     console.log(this.forma)
33 }
```

Desaparece el error en el template:

```
 1 <h4>Formularios Reactivos</h4>
 2 <hr>
 3 <form autocomplete="off" [formGroup]="forma" (ngSubmit)="guardar()">
```

Si rellenamos el formulario:

```
21   crearFormulario(){
22
23     this.forma = this.fb.group({
24       nombre: ['Txema'],
25       apellido: ['Serrano'],
26       correo: ['jsersan@gmail.com']
27     });
28
29 }
```

En la consola tenemos estos valores para formGroup :

```
touched: false
▼ value:
  apellido: "Serrano"
  correo: "jsersan@gmail.com"
  nombre: "Txema"
```

Si se fijaron el formulario está vacío por lo que hay que enlazar los campos de texto con este objeto. En el template HTML:

```

7   <div class="form-group row">
8     <label class="col-2 col-form-class">Nombre</label>
9     <div class="col-8">
10
11       <input class="form-control"
12         type="text"
13         placeholder="Nombre"
14         formControlName="nombre">
15     </div>
16   </div>
17
18   <div class="form-group row">
19     <label class="col-2 col-form-class">Apellido</label>
20     <div class="col-8">
21
22       <input class="form-control"
23         type="text"
24         placeholder="Apellido"
25         formControlName="apellido">
26     </div>
27   </div>
28 </div>
29
30 <div class="form-group row">
31   <label class="col-2 col-form-class">Correo</label>
32   <div class="col-8">
33
34     <input class="form-control"
35       type="email"
36       placeholder="Correo electrónico"
37       formControlName="correo"
38     >
39   </div>
40 </div>

```

Vista previa:

The screenshot shows a reactive form with three input fields: Nombre (Txema), Apellido (Serrano), and Correo (jsersan@gmail.com). Below the form is a 'Guardar' button. To the right, the browser's developer tools console displays the state of the FormGroup object. The console output includes:

- Angular is running in development mode. Call enableProdMode() to enable production mode.
- [webpack-dev-server] Live Reloading enabled.
- reactive.component.ts:548
- FormGroup {_pendingDirty: false, _hasOwnPendingAsyncValidator: false, _pendingTouched: false, _parent: null, _onCollectionChange: f, ...}
 ▶ controls: {nombre: FormControl, apellido: FormControl, correo: FormControl}
 ▶ errors: null
 ▶ pristine: true
 ▶ status: "VALID"
 ▶ statusChanges: EventEmitter<{closed: false, observers: Array(0), isTouched: false}>
 ▶ value:
 ▶ apellido: "Serrano"
 ▶ correo: "jsersan@gmail.com"
 ▶ nombre: "Txema"
 ▶ [Object]: Object

1. Validaciones síncronas.

Quitamos la inicialización del formulario:

```

21  crearFormulario(){
22    this.forma = this.fb.group({
23      nombre:  ['', ],
24      apellido: ['', ],
25      correo:  ['', ]
26    });
27 }
```

Le añadimos un segundo parámetro que son los validadores síncronos que no requieren interacción con servicios web externos o se ejecutan en el mismo hilo de tiempo. Angular dispone de una serie de librerías para hacer las validaciones:

```

21  crearFormulario(){
22    this.forma = this.fb.group({
23      nombre:  ['', Validators.required],
24      apellido: ['', ],
25      correo:  ['', ]
26    });
27 }
```

Probamos a enviar el formulario. Recuerden que en el template HTML no hemos puesto la propiedad required:

```

reactive.component.ts:30
▼ FormGroup {_pendingDirty: false, _hasOwnPendingAsyncValidator: false, _pendingTouc
  hed: false, _parent: null, _onCollectionChange: f, ...} ⓘ
  ► controls: {nombre: FormControl, apellido: FormControl, correo: FormControl}
  errors: null
  pristine: true
  status: "INVALID"
```

Para poder ver las propiedades del formulario sin tener que estar yendo a la consola en reactive.component.html:

```

51  </form>
52  <hr>
53  <pre>
54    Estado del formulario: {{ forma.valid }} <br>
55    Status: {{ forma.status }}
56  </pre>
57
58  <pre>
59    {{ forma.value | json }}
60  </pre>
```

Vista previa:

Formularios Reactivos

Nombre	Txema
Apellido	Serrano
Correo	jsersan@gmail.com

Guardar

Estado del formulario: true

Status: VALID

```
{  
  "nombre": "Txema",  
  "apellido": "Serrano",  
  "correo": "jsersan@gmail.com"  
}
```

Recuerden además que el mínimo de caracteres para nombre y apellido eran cinco. Empleamos además una expresión regular para chequear el email:

```
21  crearFormulario(){  
22    this.forma = this.fb.group({  
23      nombre: ['', [Validators.required, Validators.minLength(5)]],  
24      apellido: ['', [Validators.required, Validators.minLength(5)]],  
25      correo: ['', [Validators.required, Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$')]]  
26    });  
27  }
```

Probamos:

Estado del formulario: false

Status: INVALID

```
{  
  "nombre": "Txema",  
  "apellido": "Serrano",  
  "correo": "jsersan"  
}
```

Si ponemos un correo válido:

Estado del formulario: true

Status: VALID

```
{  
  "nombre": "Txema",  
  "apellido": "Serrano",  
  "correo": "jsersan@gmail.com"  
}
```

2. Validaciones y HTML.

Si hay algún error en el formulario lo mostraremos en el formulario colocando el control en rojo. En el componente creamos esta función:

```
21 |   get nombreNoValido(){
22 |     return this.forma.get('nombre')?.invalid && this.forma.get('nombre')?.touched
23 |   }
..
```

En el template:

```
11 |   <input class="form-control"
12 |     type="text"
13 |     placeholder="Nombre"
14 |     formControlName="nombre"
15 |     [class.is-invalid]="nombreNoValido">
16 |   </div>
17 | </div>
18 |
```

Si ahora tocamos el campo nombre y no tecleamos:

Formularios Reactivos

Nombre	<input type="text" value="Nombre"/>
Apellido	<input type="text" value="Apellido"/>
Correo	<input type="text" value="Correo electrónico"/>
<input type="button" value="Guardar"/>	

Vamos a colocar un texto que nos indique el error:

```
11 |   <input class="form-control"
12 |     type="text"
13 |     placeholder="Nombre"
14 |     formControlName="nombre"
15 |     [class.is-invalid]="nombreNoValido">
16 |     <small *ngIf="nombreNoValido" class="text-danger">
17 |       Ingrese 5 caracteres mínimo
18 |     </small>
..
```

Probamos:

Nombre	<input type="text" value="txem"/> <small>Ingrese 5 caracteres mínimo</small>
Apellido	<input type="text" value="Apellido"/>

Hacemos lo mismo para apellido y correo:

```

21  get nombreNoValido(){
22  | return this.forma.get('nombre')?.invalid && this.forma.get('nombre')?.touched
23 }
24
25  get apellidoNoValido(){
26  | return this.forma.get('apellido')?.invalid && this.forma.get('apellido')?.touched
27 }
28
29  get correoNoValido(){
30  | return this.forma.get('correo')?.invalid && this.forma.get('correo')?.touched
31 }
~~

```

En el template:

Formularios Reactivos

Nombre	<input type="text" value="Txema"/>
Apellido	<input type="text" value="Serrano"/>
Correo	<input type="text" value="jsersan@gmail.com"/>
<button>Guardar</button>	

Estado del formulario: true
Status: VALID
{
"nombre": "Txema",
"apellido": "Serrano",
"correo": "jsersan@gmail.com"
}

Lo que tenemos que hacer ahora es sólo guardar si el formulario es válido:

```

42  guardar(){
43  console.log(this.forma)
44
45  if(this.forma.invalid){
46  | return Object.values( this.forma.controls).forEach( control => {
47  | | control.markAsTouched();
48  | });
49
50 }
~~

```

Si en el formulario vacío pulsamos el botón Guardar:

Nombre	<input type="text" value="Nombre"/> Ingrese 5 caracteres mínimo
Apellido	<input type="text" value="Apellido"/> Ingrese el apellido
Correo	<input type="text" value="Correo electrónico"/> Ingrese el correo electrónico

4. Agrupaciones de los objetos.

Podemos tener campos compuestos del tipo dirección: calle, número, piso, letra, etc., por ejemplo. En el componente:

```
34  crearFormulario(){
35    this.forma = this.fb.group({
36      nombre: ['', [Validators.required, Validators.minLength(5)]],
37      apellido: ['', [Validators.required, Validators.minLength(5)]],
38      correo: ['', [Validators.required, Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$')]],
39      direccion: this.fb.group([
40        poblacion: ['', Validators.required],
41        ciudad: ['', Validators.required]
42      ]),
43    });
44 }
```

Probamos:

Status: INVALID

```
{
"nombre": "",
"apellido": "",
"correo": "",
"direccion": {
  "calle": "",
  "ciudad": ""
}
}
```

Creamos estos campos en el template después del correo:

```
51  <div class="form-group row">
52    <label class="col-2 col-form-class">Direccion</label>
53    <div class="form-row col">
54      <div class="col">
55        <input class="form-control"
56          type="text"
57          placeholder="Poblacion">
58      </div>
59
60      <div class="col">
61        <input class="form-control"
62          type="text"
63          placeholder="Ciudad">
64      </div>
65    </div>
66  </div>
```

Vista previa:

Correo	Correo electrónico	
Direccion	Poblacion	Ciudad
<input type="button" value="Guardar"/>		

Para que el botón tome todo el ancho:

```
68 <div class="form-group row">
69   <label class="col-2 col-form-label">&nbsp;</label>
70   <div class="input-group col">
71     <button type="submit" class="btn btn-outline-primary btn-block">
72       Guardar
73     </button>
74   </div>
75 </div>
76 </div>
```

Resultado:

Nombre	<input placeholder="Nombre" type="text"/>
Apellido	<input placeholder="Apellido" type="text"/>
Correo	<input placeholder="Correo electrónico" type="text"/>
Direccion	<input placeholder="Poblacion" type="text"/> <input placeholder="Ciudad" type="text"/>
<input type="button" value="Guardar"/>	

Como la dirección es un campo compuesto, establecemos la propiedad formGroupName en el template:

```
51 <div class="form-group row" formGroupName="direccion">
52   <label class="col-2 col-form-label">Dirección</label>
53
54   <div class="form-row col">
55     <div class="col">
56       <input class="form-control"
57         type="text"
58         placeholder="Poblacion"
59         formControlName="poblacion">
60     </div>
61
62     <div class="col">
63       <input class="form-control"
64         type="text"
65         placeholder="Provincia"
66         formControlName="provincia">
67     </div>
68   </div>
69 </div>
```

Así ahora:

```

Estado del formulario: true
Status: VALID
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  }
}

```

Si enviamos la información en la consola tenemos:

```

pristine: false
status: "VALID"
statusChanges: EventEmitter_ {closed: false, observers: 0}
touched: true
value:
  apellido: "Serrano"
  correo: "jsersan@gmail.com"
  dirección:
    población: "Arrasate"
    provincia: "Gipuzkoa"
  nombre: "Txema"

```

Ahora debemos tratar de forma parecida a los campos no compuestos para el control visual de los errores. En el componente creamos dos funciones para población y provincia:

```

34  get poblacionNoValida(){
35  |  return this.forma.get('direccion.poblacion')?.invalid && this.forma.get('direccion.poblacion')?.touched
36  |
37
38  get provinciaNoValida(){
39  |  return this.forma.get('direccion.provincia')?.invalid && this.forma.get('direccion.provincia')?.touched
40  }

```

En el template para población:

```

54  <div class="form-row col">
55    <div class="col">
56      <input class="form-control"
57        type="text"
58        placeholder="Poblacion"
59        FormControlName="poblacion"
60        [class.is-invalid]="poblacionNoValida">
61    </div>

```

Para provincia:

```

63   <div class="col">
64     <input class="form-control"
65       type="text"
66       placeholder="Provincia"
67       formControlName="provincia"
68       [class.is-invalid]="provinciaNoValida">
69   </div>

```

Probamos. Tocamos provincia y otro campo:

Correo Correo electrónico

Dirección Población Provincia

Estado del formulario: Arrasate Guipúzcoa Guardar

Status: INVALID

En cambio, si le damos a Guardar:

Nombre Nombre
Ingrese 5 caracteres mínimo

Apellido Apellido
Ingrese el apellido

Correo Correo electrónico
Ingrese el correo electrónico

Dirección Población Provincia

Guardar

Status: INVALID

No se activan porque los controles población y provincia son instancias de dirección. Lo arreglamos así:

```

69   guardar() {
70     console.log(this.forma);
71
72     if (this.forma.invalid) {
73       return Object.values(this.forma.controls).forEach((control) => {
74         if (control instanceof FormGroup) {
75           Object.values(control.controls).forEach((control) =>
76             | control.markAsTouched();
77           );
78         } else {
79           | control.markAsTouched();
80         }
81       });
82     }
83   }

```

Probamos:

Formularios Reactivos

The screenshot shows a browser developer tools console with a reactive form. The form has four fields: Nombre, Apellido, Correo, and Dirección. The Nombre field is empty and has a red border, with an error message "Ingrese 5 caracteres mínimo". The Apellido field is empty and has a red border, with an error message "Ingrese el apellido". The Correo field is empty and has a red border, with an error message "Ingrese el correo electrónico". The Dirección field contains two sub-fields: Poblacion and Provincia, both of which are empty and have red borders.

Estado del formulario: false

Status: INVALID

```
{
  "nombre": "",
  "apellido": "",
  "correo": "",
  "direccion": {
    "poblacion": "",
    "provincia": ""
  }
}
```

The console also shows Angular development mode information and a stack trace for a reactive component.

5. Reset y carga de la data inicial.

Nos creamos un nuevo método cargarData():

```
13  constructor(private fb: FormBuilder) {
14    this.crearFormulario();
15    this.cargarDataAlFormulario();
16 }
```

La función es:

```
70  cargarDataAlFormulario(){
71    this.forma.setValue({
72      nombre: 'Txema',
73      apellido: 'Serrano',
74      correo: 'jsersan@gmail.com',
75      direccion: {
76        poblacion: 'Arrasate',
77        provincia: 'Gipuzkoa'
78      }
79    });
80 }
```

Guardamos cambios y recargamos:

Formularios Reactivos

Nombre	Txema
Apellido	Serrano
Correo	jsersan@gmail.com
Dirección	Arrasate Gipuzkoa

Estado del formulario: true
Status: VALID

```
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  }
}
```

Cuando hagamos el posteo de la información haremos un reset del formulario. Así en guardar() del componente:

```
95  }
96
97  // Posteo de la información
98  this.forma.reset();
99
100 }
```

El reset me permite establecer valores individuales de los campos y los que no especifique, simplemente me los limpia. Así en cargarDataAlFormulario:

```

70  cargarDataAlFormulario(){
71    // this.forma.setValue({
72      this.forma.reset({
73        nombre: 'Txema',
74        apellido: 'Serrano',
75        correo: 'jsersan@gmail.com',
76      });
77    }
78

```

Así, al recargar:



The screenshot shows a browser's developer tools with the 'Console' tab selected. The output in the console is as follows:

```

Angular is running in development mode. core.js:24833
Call enableProdMode() to enable production mode.
[webpack-dev-server] Live Reloading index.js:548
enabled.

```

Below the console, there is a screenshot of a reactive form component. The form has four fields: Nombre (Txema), Apellido (Serrano), Correo (jsersan@gmail.com), and Dirección (with sub-fields Poblacion and Provincia). A 'Guardar' button is at the bottom. The status of the form is shown as 'INVALID'.

```

Estado del formulario: false
Status: INVALID
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": null,
    "provincia": null
  }
}

```

El reset no te obliga a llenar todos los campos.

6. Arreglos de FormControl - FormArray.

En la función crearFormulario() del componente creamos una propiedad denominada pasatiempos que es un array:

```

52  crearFormulario() {
53    this.forma = this.fb.group({
54      nombre: ['', [Validators.required, Validators.minLength(5)]],
55      apellido: ['', [Validators.required, Validators.minLength(5)]],
56      correo: [
57        '',
58        [
59          Validators.required,
60          Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+\.[a-z]{2,3}$'),
61        ],
62      ],
63      direccion: this.fb.group({
64        poblacion: ['', Validators.required],
65        provincia: ['', Validators.required],
66      }),
67      pasatiempos: this.fb.array([])
68    });
69  }

```

Probamos y vemos que aparece un array vacío denominado pasatiempos:

The screenshot shows a browser window at localhost:4200 displaying a reactive form titled "Formularios Reactivos". The form has four fields: Nombre (Txema), Apellido (Serrano), Correo (jsersan@gmail.com), and Dirección (Arrasate, Gipuzkoa). Below the form is a button labeled "Guardar". Underneath the form, the state of the form is shown as JSON:

```

Estado del formulario: true
Status: VALID
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  },
  "pasatiempos": []
}

```

Dentro de pasatiempos colocamos tres arrays vacíos en crearFormulario:

```

67   |     direccion: this.fb.group({
68   |       poblacion: ['', Validators.required],
69   |       provincia: ['', Validators.required],
70   |     },
71   |     pasatiempos: this.fb.array([
72   |       [],
73   |       [],
74   |       []
75   |     ])
76   |
77 });

```

Ahora lo enlazamos con el html. Colocamos una tabla antes del botón:

```

73 <div class="row">
74   <div class="col">
75     <table class="table">
76       <thead class="thead-light">
77         <tr>
78           <th>ID</th>
79           <th>Pasatiempo</th>
80           <th>Borrar</th>
81         </tr>
82       </thead>
83     </table>
84   </div>
85 </div>
86
87 <div class="form-group row">

```

Vista previa:

ID	Pasatiempo	Borrar
1	<input type="text"/>	
2	<input type="text"/>	
3	<input type="text"/>	

[Guardar](#)

Estado del formulario: true

Status: VALID

```

{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  },
  "pasatiempos": [
    null,
    null,
    null
  ]
}

```

Después del thead colocamos el tbody con una id, descripción y un control para eliminar el elemento de la tabla. Lo haremos con un ngFor pero en el template no tenemos acceso al array controls.

```
83     <tbody formArrayName="pasatiempos">
84       <tr *ngFor="let control of pasatiempos.controls; let i = index">
85         <td>{{ i+1 }}</td>
86         <td>
```

Nos creamos una función en el componente que nos devuelva los pasatiempos:

```
18   ngOnInit(): void {}
19
20   get pasatiempos(): FormArray {
21     return this.forma.get('pasatiempos') as FormArray;
22   }
--
```

Ahora en el template:

```
83     <tbody formArrayName="pasatiempos">
84       <tr *ngFor="let control of pasatiempos.controls; let i = index">
85         <td>{{ i+1 }}</td>
86         <td>
87           <input type="text"
88             class="form-control"
89             [FormControlName]="i">
90         </td>
91         <td></td>
92       </tr>
93     </tbody>
```

Vista previa:

ID	Pasatiempo	Borrar
1	<input type="text"/>	<button>Borrar</button>
2	<input type="text"/>	
3	<input type="text"/>	

Estado del formulario: true

Status: VALID

```
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  },
  "pasatiempos": [
    null,
    null,
    null
  ]
}
```

En el último td nos creamos el botón:

```

86 |     <td>
87 |       <input type="text"
88 |         class="form-control"
89 |         [FormControlName]="i">
90 |     </td>
91 |     <td>
92 |       <button class="btn btn-danger">
93 |         Borrar
94 |       </button>
95 |     </td>

```

Vista previa:

ID	Pasatiempo	Borrar
1	Pescar	<button>Borrar</button>
2	Surf	<button>Borrar</button>
3	Leer	<button>Borrar</button>

[Guardar](#)

Estado del formulario: true

Status: VALID

```

{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  },
  "pasatiempos": [
    "Pescar",
    "Surf",
    "Leer"
  ]
}

```

Debajo de la tabla creamos un botón con la clase primary:

```

98 |   </table>
99 |   <button class="btn btn-primary mt-3 mb-5 btn-block"
100|     type="button">
101|     + Agregar
102|   </button>

```

Vista previa:

ID	Pasatiempo	Borrar
1	<input type="text"/>	<button>Borrar</button>
2	<input type="text"/>	<button>Borrar</button>
3	<input type="text"/>	<button>Borrar</button>

[+ Agregar](#)

[Guardar](#)

Ahora podemos añadir y borrar FormControl de forma dinámica. En el componente borramos los arrays que había creado manualmente. Dejamos pasatiempos así:

```
71 |  |  pasatiempos: this.fb.array([]) 
72 |  |  ;
```

Desaparecieron los input text:

ID	Pasatiempo	Borrar
----	------------	--------

Creamos la función que nos cree los controles dinámicamente. En el template agregamos el evento:

```
~~
99 |  |  <button class="btn btn-primary mt-3 mb-5 btn-block"
100 |  |  |  type="button"
101 |  |  |  (click)="agregarPasatiempo()"
102 |  |  + Agregar
103 |  |  </button>
```

Creamos el método en el componente:

```
88 | agregarPasatiempo(){
89 |   this.pasatiempos.push(this.fb.control('Nuevo elemento', Validators.required))
90 |
91 }
```

Probamos. Pulsamos un par de veces:

ID	Pasatiempo	Borrar
1	Nuevo elemento	<button>Borrar</button>
2	Nuevo elemento	<button>Borrar</button>
+ Agregar		
Guardar		

Estado del formulario: true

Status: VALID

```
{
  "nombre": "Txema",
  "apellido": "Serrano",
  "correo": "jsersan@gmail.com",
  "direccion": {
    "poblacion": "Arrasate",
    "provincia": "Gipuzkoa"
  },
  "pasatiempos": [
    "Nuevo elemento",
    "Nuevo elemento"
  ]
}
```

Dejamos la función agregarPasatiempo():

```
88  agregarPasatiempo(){
89  |   this.pasatiempos.push(this.fb.control(''))
90  }
```

La forma para borrar un elemento en particular:

```
91  |   |   |   <td>
92  |   |   |   |   <button class="btn btn-danger"
93  |   |   |   |   |   type="button"
94  |   |   |   |   |   (click)="borrarPasatiempo(i)">
95  |   |   |   |   Borrar
96  |   |   |   </button>
97  |   |   |   </td>
```

En el componente:

```
-->
92  |   borrarPasatiempo(i: number){
93  |   |   this.pasatiempos.removeAt(i)
94  |   }
```

Probamos: añade y elimina correctamente.

7. Validaciones personalizadas.

Vamos a crear un servicio para las validaciones personalizadas:

```
jsersan@iMac-de-Jose formReactiva % ng g s services/validadores --skip-tests
CREATE src/app/services/validadores.service.ts (140 bytes)
jsersan@iMac-de-Jose formReactiva %
```

En este servicio me creo una función noSerrano para no permitir ningún apellido Serrano que devuelve un objeto array de string de tipo boolean:

```
11  noSerrano( control: FormControl):{[s:string]: boolean} {
12    return{
13      noSerrano:true
14    }
15 }
```

Esta función es verdadera si en la caja de texto hay un valor “Serrano”. Para esto:

```
11  noSerrano( control: FormControl):{[s:string]: boolean} {
12
13  if (control.value.toLowerCase() === 'serrano'){
14    return{
15      noSerrano:true
16    }
17  }
18
19  return null;
20
21 }
```

Tenemos este error que es debido al tslint de vscode. Inyectamos el servicio en el constructor del componente:

```
13  constructor(private fb: FormBuilder,
14               private validadores: ValidadoresService) {
15    this.crearFormulario();
16    this.cargarDataAlFormulario();
17 }
```

Lo aplicamos en el campo apellido:

```
57  crearFormulario() {
58    this.forma = this.fb.group({
59      nombre: ['', [Validators.required, Validators.minLength(5)]],
60      apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
61    })
62 }
```

Probamos y tenemos este error:

```
Error: src/app/services/validadores.service.ts:19:5 - error TS2322: Type 'null' is
not assignable to type '{ [s: string]: boolean; }'.
```

```
19  return null;
~~~~~
```

Vamos a services/validadores.service.ts. No nos permite retornar null.

```

11  noSerrano( control: FormControl):{ [s:string]: boolean } {
12
13    if( control.value.toLowerCase() === 'serrano'){
14      return {
15        noSerrano: true
16      }
17    }
18
19    return null;
-->
```

Para evitar esto, en el tsconfig.json ponemos la propiedad strict a false:

```

1  /* To learn more about this file see: https://angular.io/config/tsconfig.
2  {
3    "compileOnSave": false,
4    "compilerOptions": {
5      "baseUrl": "./",
6      "outDir": "./dist/out-tsc",
7      "forceConsistentCasingInFileNames": true,
8      "strict": false,
```

Ahora, con esta estructura, ya no me deja introducir un usuario de apellido Serrano.
Formularios Reactivos

Nombre	<input type="text" value="Txema"/>
Apellido	<input type="text" value="Serrand"/> Ingrese el apellido

Si ahora ponemos de apellido Serrano2:

```

✖ > ERROR TypeError: Cannot read properties of null (reading 'toLowerCase')
  at noSerrano (validadores.service.ts:13:23)                         core.mjs:6461
  at forms.mjs:811:40
  at Array.map (<anonymous>)
  at executeValidators (forms.mjs:811:23)
  at FormControl._composedValidatorFn (forms.mjs:842:28)
  at FormControl._runValidator (forms.mjs:2530:38)
  at FormControl.updateValueAndValidity (forms.mjs:2507:32)
  at FormControl.setValue (forms.mjs:2918:14)
  at FormControl.reset (forms.mjs:2954:14)
  at forms.mjs:3370:21
```

Es debido a que estoy tratando de pasar a minúsculas un campo numérico y da error. Lo soluciono en el servicio poniendo ‘?’

```

11  noSerrano( control: FormControl):{ [s:string]: boolean } {
12
13    if( control.value?.toLowerCase() === 'serrano'){
14      return {
15        noSerrano: true
16      }
17    }
18
```

8. Validaciones que el password1 sea igual al password2.

Debajo del correo nos creamos dos campos contraseñas:

```

52 <div class="form-group row">
53   <label class="col-2 col-form-label">Contraseña</label>
54   <div class="col">
55
56     <input class="form-control"
57       type="text"
58       placeholder="Contraseña"
59       formControlName="pass1">
60   </div>
61 </div>
62
63 <div class="form-group row">
64   <label class="col-2 col-form-label">Repita Contraseña</label>
65   <div class="col">
66
67     <input class="form-control"
68       type="text"
69       placeholder="Repita Contraseña"
70       formControlName="pass2">
71   </div>
72 </div>
73

```

Como nos están definidos tendremos un error en la consola:

```

✖ > ERROR Error: Cannot find control with name: 'pass1' core.mjs:6461
    at _throwError (forms.mjs:1794:11)
    at setUpControl (forms.mjs:1574:13)
    at FormGroupDirective.addControl (forms.mjs:5296:9)
    at FormControlName._setUpControl (forms.mjs:5858:43)
    at FormControlName.ngOnChanges (forms.mjs:5803:18)
    at FormControlName.rememberChangeHistoryAndInvokeOnChangesHook (core.mjs:1491:1)
    at callHook (core.mjs:2528:1)
    at callHooks (core.mjs:2487:1)
    at executeInitAndCheckHooks (core.mjs:2438:1)
    at selectIndexInternal (core.mjs:8380:1)

Angular is running in development mode. Call enableProdMode() to enable core.mjs:24833
production mode.

✖ > ERROR Error: Cannot find control with name: 'pass2' core.mjs:6461
    at _throwError (forms.mjs:1794:11)
    at setUpControl (forms.mjs:1574:13)
    at FormGroupDirective.addControl (forms.mjs:5296:9)
    at FormControlName._setUpControl (forms.mjs:5858:43)
    at FormControlName.ngOnChanges (forms.mjs:5803:18)
    at FormControlName.rememberChangeHistoryAndInvokeOnChangesHook (core.mjs:1491:1)
    at callHook (core.mjs:2528:1)
    at callHooks (core.mjs:2487:1)
    at executeInitAndCheckHooks (core.mjs:2438:1)
    at selectIndexInternal (core.mjs:8380:1)

```

Definimos estos campos en CrearFormulario() del componente:

```

68   crearFormulario() {
69     this.forma = this.fb.group({
70       nombre: ['', [Validators.required, Validators.minLength(5)]],
71       apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
72       pass1: ['', Validators.required],
73       pass2: ['', Validators.required],
74       correo: [
75         '',
76         [
77           Validators.required,
78           Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,3}$'),
79         ],
80       ],

```

Vista previa:

Formularios Reactivos

Nombre	<input type="text" value="Txema"/>
Apellido	<input type="text" value="Serrano"/>
Correo	<input type="text" value="jsersan@gmail.com"/>
Contraseña	<input type="password" value="Contraseña"/>
Repita Contraseña	<input type="password" value="Repita Contraseña"/>
Correo	<input type="text" value="jsersan@gmail.com"/>
Dirección	<input type="text" value="Arrasate"/> Gipuzkoa

Nos creamos en el componente una función get que nos permita hacer la comprobación de que las contraseñas son iguales. En el componente implementamos la función:

```

57   get pass1NoValido() {
58     return this.forma.get('pass1').invalid && this.forma.get('pass1').touched
59   }
60 }
61

```

En el template ahora:

```

56   <input class="form-control"
57     type="text"
58     placeholder="Contraseña"
59     formControlName="pass1"
60     [class.is-invalid]="pass1NoValido">
61

```

Guardo los cambios y hago click en contraseña1 sin escribir nada.

Formularios Reactivos

Nombre	<input type="text" value="Txema"/>
Apellido	<input type="text" value="Serrano"/>
Correo	<input type="text" value="jsersan@gmail.com"/>
Contraseña	<input type="password" value="Contraseña"/>

Ahora hacemos la parte del passw2, copiamos el get:

```

62   get pass2NoValido() {
63     const pass1 = this.forma.get('pass1').value;
64     const pass2 = this.forma.get('pass2').value;
65
66     return ( pass1 === pass2 ) ? false : true;
67

```

En el template:

```

64  <div class="form-group row">
65    <label class="col-2 col-form-label">Repita Contraseña</label>
66    <div class="col">
67
68      <input class="form-control"
69        type="text"
70        placeholder="Repita Contraseña"
71        [class.is-invalid]="pass2NoValido"
72        formControlName="pass2">
73    </div>
74  </div>
75

```

Probamos:

Contraseña	<input type="text" value="123"/>
Repita Contraseña	<input type="text" value="Repita Contraseña"/>

Si ponemos la segunda:

Contraseña	<input type="text" value="123"/>
Repita Contraseña	<input type="text" value="123"/>

No queremos que se vean las contraseñas, cambiamos el tipo a password.

```

56    <input class="form-control"
57      type="password"
58      placeholder="Contraseña"
59      formControlName="pass1"
60      [class.is-invalid]="pass1NoValido">

```

Vista previa:

Contraseña	<input type="password" value="..."/>	
Repita Contraseña	<input type="password" value=".."/>	
Correo	<input type="text" value="jsersan@gmail.com"/>	
Dirección	<input type="text" value="Arrasate"/> <input type="text" value="Gipuzkoa"/>	
ID	Pasatiempo	Borrar
+ Agregar		
Guardar		

Estado del formulario: true

Status: VALID

Si se fijan, aunque hay errores el formulario sigue apareciendo como válido. Para ello en CrearFormulario() hacemos una validación especial:

```

69  crearFormulario() {
70    this.forma = this.fb.group({
71      nombre: ['', [Validators.required, Validators.minLength(5)]],
72      apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
73      pass1: ['', Validators.required],
74      pass2: ['', Validators.required],
75      correo: [
76        '',
77        [
78          Validators.required,
79          Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,3}$'),
80        ],
81      ],
82      direccion: this.fb.group({
83        poblacion: ['', Validators.required],
84        provincia: ['', Validators.required],
85      }),
86      pasatiempos: this.fb.array([])
87    },{
88      validators: this.validadores.passwordsIguales('pass1', 'pass2')
89    });
90  }

```

La función passwordsIguales la creamos en el servicio. Como la validación es a nivel de formulario, lo que voy a recibir es un formulario:

```

22  passwordsIguales(pass1Name: string, pass2Name: string){
23
24    return (formGroup: FormGroup) => {
25
26      const pass1Control = formGroup.controls[pass1Name];
27      const pass2Control = formGroup.controls[pass2Name];
28
29      if( pass1Control.value === pass2Control.value){
30        pass2Control.setErrors(null);
31      } else {
32        pass2Control.setErrors({ noEsIgual: true});
33      }
34    }
35  }

```

Probamos y cuando las contraseñas son iguales tenemos:

Estado del formulario: true

Status: VALID

```
{
"nombre": "Txema",
"apellido": "Dominguez",
"pass1": "123",
"pass2": "123",
```

9. Validadores asíncronos.

Creamos un nuevo campo Usuario debajo del correo:

```

52  <div class="form-group row">
53    <label class="col-2 col-form-label">Usuario</label>
54    <div class="col">
55      <input class="form-control"
56        type="email"
57        placeholder="Nombre de usuario"
58        formControlName="usuario"
59        [class.is-invalid]="usuarioNoValido">
60        <small *ngIf="usuarioNoValido" class="text-danger">
61          | Ese nombre de usuario ya ha sido elegido
62        </small>
63      </div>
64    </div>
65  </div>

```

Creamos la función usuarioNoValido en el componente:

```

57  get usuarioNoValido() {
58    return this.forma.get('usuario').invalid && this.forma.get('usuario').touched
59  }
60

```

La función crearFormulario:

```

74  crearFormulario() {
75    this.forma = this.fb.group({
76      nombre: ['', [Validators.required, Validators.minLength(5)]],
77      apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
78      pass1: ['', Validators.required],
79      pass2: ['', Validators.required],
80      correo: [
81        '',
82        [
83          Validators.required,
84          Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,3}$'),
85        ],
86      ],
87      usuario: ['', , this.validadores.existeUsuario],
88      direccion: this.fb.group({
89        poblacion: ['', Validators.required],
90        provincia: ['', Validators.required],
91      }),
92      pasatiempos: this.fb.array([])
93    }, {
94      validators: this.validadores.passwordsIguales('pass1', 'pass2')
95    });
96  }

```

Nos debemos crear el método existeUsuario en validadores.service.ts:

```

12  existeUsuario(): Promise<any> | Observable<any>{
13    return{
14      existe: true
15    }
16  }

```

Para evitar tener que poner any, nos creamos una pequeña interfaz de la siguiente manera en el servicio:

```

16  existeUsuario(control: FormControl): Promise<ErrorValidate> | Observable<ErrorValidate>{
17
18    return new Promise( (resolve, reject) =>{
19
20      setTimeout(() =>{
21        if(control.value === 'strider'){
22          resolve({existe:true})
23        }else{
24          resolve(null)
25        }
26      }, 3500 )
27
28    })
29

```

Recordando que la validación asíncrona se define así (con un espacio como segundo parámetro:

```
usuario: ['', , this.validadores.existeUsuario],
```

La función crearFormulario() queda

```

74  crearFormulario() {
75    this.forma = this.fb.group({
76      nombre: ['', [Validators.required, Validators.minLength(5)]],
77      apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
78      pass1: ['', Validators.required],
79      pass2: ['', Validators.required],
80      correo: [
81        '',
82        [
83          Validators.required,
84          Validators.pattern('^[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,3}$')
85        ],
86      ],
87      usuario: ['', , this.validadores.existeUsuario],
88      direccion: this.fb.group({
89        poblacion: ['', Validators.required],
90        provincia: ['', Validators.required],
91      }),
92      pasatiempos: this.fb.array([])
93    },
94    {
95      validators: this.validadores.passwordsIguales('pass1', 'pass2')
96    });

```

Probamos:

```
Estado del formulario: false
```

```
Status: INVALID
```

En la función cargarDataAlFormulario():

```

98     cargarDataAlFormulario(){
99       // this.forma.setValue({
100      this.forma.reset([
101        nombre: 'Txema',
102        apellido: 'Serrano',
103        correo: 'jsersan@gmail.com',
104        pass1: '123',
105        pass2: '123',
106        direccion: [
107          poblacion: 'Arrasate',
108          provincia: 'Gipuzkoa'
109        ]
110      ]);
111    }

```

Probamos en el formulario. Mientras escribe el nombre de usuario en los tres primeros segundos y medio el estado es “pending”:

```

Estado del formulario: false

Status: PENDING

{
  "nombre": "Txema",
  "apellido": "Serrano",
  "pass1": "123",
  "pass2": "123",
  "correo": "jsersan@gmail.com",
  "usuario": "123",
  "direccion": {

```

Después de este tiempo:

```

Estado del formulario: false

Status: INVALID

{
  "nombre": "Txema",
  "apellido": "Serrano",
  "pass1": "123",
  "pass2": "123",
  "correo": "jsersan@gmail.com",
  "usuario": "123",

```

Como verdaderamente no quiero que se dispare la petición mientras el usuario escribe porque en caso contrario sería una validación síncrona en la parte de los validadores. En el servicio:

```

16   existeUsuario(control: FormControl): Promise<ErrorValidate> | Observable<ErrorValidate>{
17
18   if(!control.value) {
19     return Promise.resolve(null);
20   }

```

Si ahora en el formulario intentáramos crear un usuario “strider”, en tres segundos y medio aparece:

Formularios Reactivos

Nombre	<input type="text" value="Txema"/>
Apellido	<input type="text" value="Serrano"/>
Correo	<input type="text" value="jsersan@gmail.com"/>
Jsuario	<input type="text" value="strider"/> <small>Este nombre de usuario ya ha sido elegido</small>

10. Detectar cambios en los valores, controles o estado del formulario.

Lo que vamos a intentar es disparar un evento cuando algún campo del formulario cambia. Sería interesante crear una promesa u observable para escuchar los cambios en el formulario:

```

74  crearFormulario() {
75    this.forma = this.fb.group({
76      nombre: ['', [Validators.required, Validators.minLength(5)]],
77      apellido: ['', [Validators.required, Validators.minLength(5), this.validadores.noSerrano]],
78      pass1: ['', Validators.required],
79      pass2: ['', Validators.required],
80      correo: [
81        '',
82        [
83          Validators.required,
84          Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+.[a-z]{2,3}$'),
85        ],
86      ],
87      usuario: ['', , this.validadores.existeUsuario],
88      direccion: this.fb.group({
89        poblacion: ['', Validators.required],
90        provincia: ['', Validators.required],
91      }),
92      pasatiempos: this.fb.array([])
93    },
94    {
95      validators: this.validadores.passwordsIguales('pass1', 'pass2')
96    });
  
```

En el constructor invoco un método crearListener():

```

10  export class ReactiveComponent implements OnInit {
11    forma!: FormGroup;
12
13    constructor(private fb: FormBuilder,
14                private validadores: ValidadoresService) {
15      this.crearFormulario();
16      this.cargarDataAlFormulario();
17      this.crearListeners();
18    }
  
```

Creamos este método:

```

99  crearListeners(){
100    this.forma.valueChanges.subscribe( valor =>{
101      console.log(valor);
102    });
103  }
  
```

Probamos y en cuanto escriba en cualquier campo del formulario en la consola tenemos:

```

  react
▼ {nombre: 'Txema S', apellido: 'Serrano', pass1: '123', pass2: '123', coi
  com', ...} ⓘ
  apellido: "Serrano"
  correo: "jsersan@gmail.com"
  dirección: {poblacion: 'Arrasate', provincia: 'Gipuzkoa'}
  nombre: "Txema S"
  pasatiempos: []
  pass1: "123"
  pass2: "123"
  usuario: null
  
```

Puede que también necesitemos el status changes:

```

99  crearListeners() {
100    this.forma.valueChanges.subscribe( valor =>{
101      console.log(valor);
102    });
103
104    this.forma.statusChanges.subscribe(status => console.log({status}))
105  }
  
```

Recargamos y ponemos de nombre de usuario “periko”: el formulario es válido.

Si el usuario es “strider”:

El formulario es inválido porque ese nombre ya está reservado.

Cuestionario relativo a formularios:

Pregunta 1:

En el manejo de los formularios por el Template, ¿Angular crea el ngForm por nosotros en caso de que la necesitemos?

Verdadero

Falso

Pregunta 2:

Para evaluar una caja de texto y que cumpla un patrón, ¿Qué podemos utilizar?

Una variable

Una función

Una expresión regular

Pregunta 3:

Para la aproximación por Template, ¿Es necesario importar ReactiveFormsModuleModule?

Verdadero

Falso

Pregunta 4:

¿Qué es el estado pristine en un input?

Significa que el input no ha sido tocado.

Significa que el input tiene el valor inicial y que no se ha modificado.

Significa que el input no se puede modificar

Pregunta 5:

¿Qué es una validación asíncrona ?

- Es una validación que se ejecuta a destiempo y permite comprobar los datos contra un objeto.
- Es una validación que se ejecuta y espera la respuesta para poder comprobar si pasa la validación o no.**
- Es una validación que se dispara cuando el usuario cambia el valor del input.

Pregunta 6:

¿Qué tipo de objeto puede aceptar un FormControl en la parte de las validaciones?

- Una clase
- Una función**
- Una variable
- Una expresión regular

Pregunta 7:

¿Cómo podemos detectar cambios en el valor de algún input?

- Nos subscribimos al observador "statusChange"
- Agregamos una función al input
- Creamos un método en el formControl que esta relacionado al input
- Nos subscribimos al observador "valueChanges" del control**

Pregunta 8:

¿Cómo podemos hacer un reset del formulario con la aproximación del template?

Creamos la referencia local a la forma (#forma), y luego podemos llamar el reset de la misma.

No se puede

Creamos una referencia a la forma (#forma), y de ahí la tenemos que convertir en una NgForm, para poder ejecutar el reset.

Pregunta 9:

Si utilizamos el ngModel en un input, ¿qué debemos importar en el módulo?

FormsModule

ReactiveFormsModule

NgModelModule

Pregunta 10:

¿Cómo relacionamos en los formularios reactivos, un input con una propiedad del FormGroup?

Agregamos el atributo formControlName al input

Agregamos el atributo formControl al input

Agregamos el atributo formName al input

Agregamos la propiedad "name" y el ngModel