Formularios reactivos

La solución en Angular 8 pasa por desacoplar el modelo y la vista, introduciendo una capa que gestione ese doble enlace. Los servicios y directivas del módulo ReactiveFormsModule que viene en la librería `@angular/forms` permiten programar **formularios reactivos conducidos por el código**.

La directiva `[(ngModel)]="model.property"` con su popular *banana in a box* establece el doble enlace entre el elemento de la vista al que se le aplica y una propiedad del modelo. Los cambios en la vista son trasladados automáticamente al modelo, y al revés; cualquier cambio en el modelo se refleja inmediatamente en la vista.

Se pueden establecer validaciones y configurar los eventos que disparan las actualizaciones; pero todo ello usando más y más atributos y directivas en la plantilla. Son los formularios *template driven* que degeneran en un *html* farragoso y difícil de mantener.

FormControl

El formulario se define como un **grupo de controles**. Cada control tendrá un nombre y una configuración. Esa definición permite establecer un valor inicial al control.

Form view

Mientras tanto en la vista html… Este trabajo previo y extra que tienes que hacer en el controlador se recompensa con una **mayor limpieza en la**

vista. Lo único necesario será asignar por nombre el elemento html con el control *typescript* que lo gestionará.

> Para ello usaremos dos directivas que vienen dentro del módulo *reactivo* son `[formGroup]="objetoFormulario"` para el formulario en su conjunto, y `formControlName="nombreDelControl"` para cada control.

Validación y estados

La validación es una pieza clave de la entrada de datos en cualquier aplicación. Es el primer **frente de defensa ante errores de usuarios**; ya sean involuntarios o deliberados.

Dichas validaciones se solían realizar agregando atributos html tales como el archiconocido `required`. Pero todo eso ahora se traslada a la configuración de cada control, donde podrás establecer una o varias reglas de validación sin mancharte con html.

Validadores predefinidos y personalizados

De nuevo tienes distintas sobrecargas que te permiten resolver limpiamente casos sencillos de una sola validación, o usar baterías de reglas que vienen predefinidas como funciones en el objeto `Validators`

Estados de cambio y validación

Una vez establecidas las reglas, es hora de aplicarlas y avisar al usuario en caso de que se incumplan. Los formularios y controles reactivos están gestionados por **máquinas de estados** que determinan en todo momento la situación de cada control y del formulario en si mismo.

Estados de validación

Al establecer una o más reglas para uno o más controles activamos el sistema de chequeo y control del estado de cada control y del formulario en su conjunto.

La máquina de estados de validación contempla los siguientes estados mutuamente excluyentes:

- **VALID**: el control ha pasado todos los chequeos
- **INVALID**: el control ha fallado al menos en una regla.
- **PENDING**: el control está en medio de un proceso de validación
- **DISABLED**: el control está desactivado y exento de validación

Cuando un control incumple con alguna regla de validación, estas se reflejan en su propiedad `errors` que será un objeto con una propiedad por cada regla insatisfecha y un valor o mensaje de ayuda guardado en dicha propiedad.

Estados de modificación

Los controles, y el formulario, se someten a otra máquina de estados que monitoriza el valor del control y sus cambios.

La máquina de estados de cambio contempla entre otros los siguientes:

- **PRINSTINE**: el valor del control no ha sido cambiado por el usuario
- **DIRTY**: el usuario ha modificado el valor del control.
- **TOUCHED**: el usuario ha tocado el control lanzando un evento `blur` al salir.
- **UNTOUCHED**: el usuario no ha tocado y salido del control lanzando ningún evento `blur`.

Como en el caso de los estados de validación, el formulario también se somete a estos estados en función de cómo estén sus controles.

La validación particular para cada control permite informar al usuario del fallo concreto. Es una **buena práctica de usabilidad** el esperar a que edite un control antes de mostrarle el fallo. Y también es muy habitual usar la misma estrategia para cada control.

Recordar que dentro de la vista puede accederse a cada control específico de la siguiente forma :

form.get(‘control’).status : Accedemos a la propiedad del control
form.get(‘control’).touched : Accedemos a la propiedad touched

Siendo form la instancia del formulario que iniciamos dentro de la etiqueta <form> e instanciamos en el .ts del component