a. ¿Qué son, para qué sirven y cómo se utilizan FormControl, FormGroup y FormBuilder?

Son elementos esenciales de los formularios reactivos.

`FormControl` corresponde a la base de cualquier formulario de este tipo, representando cada elemento individual de la plantilla. Se trata del elemento que permite asignar valores iniciales y formas de validación. Estos, por tanto, encapsulan el estado y comportamiento de los *input fields*, *checkboxes*, *radio buttons* y listas desplegables de selección. Se crean a través de la clase `FormControl` del paquete de `forms` de Angular, y se pueden importar individualmente como un módulo más, o a través del módulo `ReactiveFormsModule`:

```
import { FormControl } from '@angular/forms';
```

`FormGroup` permite agrupar diferentes *form controls* bajo el mismo formulario, ofreciendo la posibilidad de hacer seguimiento a lo introducido en el formulario de forma individual, por ejemplo para obtener todos los resultados de forma simultánea o para comprobar si el formulario, en su globalidad, es válido. Se aplica a través de un atributo que se asocia al elemento del formulario. Realizando esta acción, se puede colocar un atributo `formControlName` en cada *form control* del formulario en lugar de utilizar `formControl`, aplicando el *binding* de esta forma.

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

`FormBuilder` por su parte, permite la construcción de formularios de una forma menos ompulosa. Permite crear elementos `FormGroup` y `FormControl` sin tener que llamar manualmente cada uno de ellos a través de `new`.

b. Busca en la página oficial de Angular (o utiliza un recurso de O'Reilly) en el que se especifiquen todos los validadores que incluye Angular para ser utilizados en los formularios reactivos. Construye una tabla de resumen de estos.

Validador	Descripción	Uso
min()	Requiere que el número control del campo sea mayor o igual al proporcionado	`static min(min: number): ValidatorFn`
max()	Requiere que el número control del campo sea menor o igual al proporcionado	`static max(max :number): ValidatorFn`
required()	Requiere que el campo posea un valor no vacío	`static required(control: AbstractControl<any, any>): ValidationErrores | null`

Validador	Descripción	Uso
requiredTrue()	Requiere que el valor del campo sea `true`, como en el caso de las *checkboxes*	`static requiredTrue(control: AbstractControl<any, any>): ValidationErrors | null`
email()	Requiere que el valor del campo pase un test de validación de email	`static email(control: AbstractControl<any, any>): ValidationErrors | null`
minLength()	Requiere que el valor del campo contenga un número de caracteres mayor o igual al especificado. Puede usarse tanto en `number` como en `string`	`static minLength(minLength: number): ValidatorFn`
maxLength()	Requiere que el valor del campo contenga un número de caracteres menor o igual al especificado. Puede usarse con `number` o `string`	`static minlength(maxLength: number): ValidatorFn`
pattern()	Comprueba si el valor del campo pasa un patrón de expresión regular	`static pattern(pattern: string| RegExp): ValidatorFn`
nullValidator()	No realiza ninguna validación	`static nullValidator(control: AbstractControl<any, any>): ValidationErrors | null`
compose()	Compone varios validadores en una sola función que devuelve la unión del mapeo de errores	`static compose(validators: ValidatorFn[]): ValidatorFn | null`
composeAsync()	Realiza una función similar a la anterior pero de forma asíncrona	`static composeAsync(validators: AsyncValidatorFn[]): AsyncValidatorFn | null`

c. ¿Qué son, cuáles son y para qué sirven los estados en los formularios reactivos?

Los estados de control nos permiten saber cuándo y cómo el usuario ha interactuado con el formulario, y realizar acciones en función a ello. Existen los

siguientes:

- `pristine`: se considera como tal cuando no se ha realizado ningún tipo de interacción por parte del usuario. Todos los campos del formulario se encuentran en este estado al principio, y solo cambian una vez el usuario ha introducido un valor o ha hecho *focus* en el campo.
- `dirty`: Un campo se considera así cuando el usuario interactúa con él, cambiando su valor. Esto puede darse cuando el usuario introduce un valor en un campo en el que se puede añadir texto o cuando elige una opción de un menú desplegable, por ejemplo. En términos amplios, indica, que el campo ha sido modificado.
- `touched`: se considera así cuando el usuario ha realizado `focus` en el campo, y ha pasado a otra cosa. Por ejemplo, si este hace clic en un campo `input` y luego hace clic fuera del mismo, se considera que ha sido `touched`. Se usa a menudo para decidir si colocar mensajes de validación.