

Angular

Formularios

CertiDevs

 ndice de contenidos

1. Formularios en Angular	1
2. Template-driven forms	1
2.1. Ejemplo 1	1
2.1.1. Configuraci�n inicial	1
2.1.2. Paso 1: Crear un componente para el formulario	1
2.1.3. Paso 2: Definir la estructura del formulario en la plantilla del componente	2
2.1.4. Paso 3: Vincular los elementos del formulario a las propiedades del componente	2
2.1.5. Paso 4: Mostrar el componente en la aplicaci�n	3
2.2. Validaci�n de formularios	3
2.3. Ejemplo 2: tareas	4
2.3.1. Paso 1: Crear el componente para el formulario de tarea	4
2.3.2. Paso 2: Definir la estructura del formulario en la plantilla del componente	4
2.3.3. Paso 3: Vincular los elementos del formulario a las propiedades del componente	6
2.3.4. Mostrar el componente en la aplicaci�n	6
3. Reactive Forms	7
3.1. Configuraci�n inicial	7
3.2. Creando un formulario reactivo	7
3.2.1. Paso 1: Crear un componente para el formulario	7
3.2.2. Paso 2: Importar las clases necesarias en el componente	8
3.2.3. Paso 3: Definir la estructura del formulario en el componente	8
3.2.4. Paso 4: Vincular los elementos del formulario a las instancias de las clases en la plantilla del componente	8
3.2.5. Paso 5: Manejo de eventos de env�o del formulario	9
3.2.6. Paso 6: Mostrar el componente en la aplicaci�n	9
3.3. Validaci�n de formularios reactivos	10
3.4. Ejemplo 1	11
3.4.1. Paso 1: Crear un componente para el formulario de tarea	11
3.4.2. Paso 2: Crear una interfaz para la Tarea	11
3.4.3. Paso 3: Definir la estructura del formulario en el componente	12
3.4.4. Paso 4: Vincular los elementos del formulario en la plantilla del componente	12
3.4.5. Paso 5: Mostrar el componente en la aplicaci�n	14
3.5. Ejemplo 2	14
3.5.1. Paso 1: Crear un componente para el formulario de inscripci�n	14
3.5.2. Paso 2: Crear las interfaces para Estudiante y Curso	14
3.5.3. Paso 3: Definir la estructura del formulario utilizando FormGroup, FormControl y FormArray	15
3.5.4. Paso 4: Vincular los elementos del formulario en la plantilla del componente	16
3.5.5. Mostrar el componente en la aplicaci�n	17

1. Formularios en Angular

Angular proporciona dos enfoques para manejar formularios en aplicaciones web:

- ¥ **Template-driven forms:** Este enfoque utiliza la l gica de plantillas para manejar la interacci n con el usuario y la validaci n de datos. Est  m s enfocado en el lado de la vista y es m s adecuado para formularios simples.
- ¥ **Reactive forms:** Este enfoque utiliza la l gica en el componente para manejar la interacci n y validaci n de datos. Est  m s enfocado en el lado del componente y es m s adecuado para formularios complejos y din micos.

En este tutorial, nos centraremos en los template-driven forms.

2. Template-driven forms

2.1. Ejemplo 1

2.1.1. Configuraci n inicial

Para comenzar, necesitas importar el m dulo `FormsModule` en tu m dulo principal.

Abre el archivo `app.module.ts` y realiza los siguientes cambios:

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

@NgModule({
  imports: [
    BrowserModule,
    FormsModule // A adir FormsModule aqu 
  ],
  // ...
})
export class AppModule { }
```

Para crear un formulario b sico, sigue estos pasos:

- ¥ Crea un componente para el formulario.
- ¥ Define la estructura del formulario en la plantilla del componente.
- ¥ Vincula los elementos del formulario a las propiedades del componente.

2.1.2. Paso 1: Crear un componente para el formulario

Ejecuta el siguiente comando para generar un nuevo componente llamado `mi-formulario`:

```
ng generate component mi-formulario
```

2.1.3. Paso 2: Definir la estructura del formulario en la plantilla del componente

Abre el archivo `mi-formulario.component.html` y reemplaza su contenido con el siguiente código:

```
<form (ngSubmit)="onSubmit()">
  <div>
    <label for="nombre">Nombre: </label>
    <input type="text" id="nombre" [(ngModel)]="nombre" name="nombre" required>
  </div>
  <div>
    <label for="email">Email: </label>
    <input type="email" id="email" [(ngModel)]="email" name="email" required>
  </div>

  <button type="submit">Enviar</button>
</form>
```

En este ejemplo, hemos creado un formulario simple con dos campos de entrada: nombre y correo electrónico.

También hemos utilizado la directiva `[(ngModel)]` para vincular cada campo de entrada a una propiedad del componente y hemos añadido la etiqueta `required` para indicar que ambos campos son obligatorios.

2.1.4. Paso 3: Vincular los elementos del formulario a las propiedades del componente

Ahora, abre el archivo `mi-formulario.component.ts` y modifícalo de la siguiente manera:

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-mi-formulario',
  templateUrl: './mi-formulario.component.html',
  styleUrls: ['./mi-formulario.component.css']
})
export class MiFormularioComponent {
  nombre = '';
  email = '';

  onSubmit() {
    console.log('Nombre:', this.nombre);
    console.log('Email:', this.email);
  }
}
```

```
}
```

En este ejemplo, hemos definido dos propiedades en nuestro componente, nombre y email, que están vinculadas a los campos de entrada en nuestra plantilla mediante la directiva `[(ngModel)]`.

También hemos creado una función llamada `onSubmit()` que se ejecuta cuando el formulario se envía.

2.1.5. Paso 4: Mostrar el componente en la aplicación

Para ver el componente `mi-formulario` en la aplicación, abre el archivo `app.component.html` y reemplaza su contenido con lo siguiente:

```
<app-mi-formulario></app-mi-formulario>
```

Ahora, ejecuta la aplicación con el comando `ng serve` y visita <http://localhost:4200> en tu navegador. Deberías ver el formulario que acabamos de crear.

2.2. Validación de formularios

Angular proporciona varias herramientas para validar formularios.

En este ejemplo, vamos a utilizar la validación por defecto del navegador y mostrar mensajes de error personalizados.

Primero, añade el siguiente CSS al archivo `mi-formulario.component.css`:

```
input.ng-invalid.ng-touched {  
  border-color: red;  
}  
.error-message {  
  color: red;  
  font-size: 0.8em;  
}
```

Este CSS resalta los campos no válidos en rojo y aplica un estilo personalizado a los mensajes de error.

Ahora, actualiza el archivo `mi-formulario.component.html` con el siguiente código:

```
<form (ngSubmit)="onSubmit()" #miForm="ngForm">  
  <div>  
    <label for="nombre">Nombre: </label>  
    <input type="text" id="nombre" [(ngModel)]="nombre" name="nombre" required  
      #nombreRef="ngModel">  
    <div *ngIf="nombreRef.invalid && nombreRef.touched" class="error-message">  
      El nombre es obligatorio.
```

```

    </div>
  </div>
  <div>
    <label for="email">Email: </label>
    <input type="email" id="email" [(ngModel)]="email" name="email" required
    #emailRef="ngModel">
    <div *ngIf="emailRef.invalid && emailRef.touched" class="error-message">
      El email es obligatorio y debe ser v lido.
    </div>
  </div>
</div>

<button type="submit" [disabled]="miForm.invalid">Enviar</button>
</form>

```

Hemos realizado los siguientes cambios:

-   Hemos a adido una referencia local al formulario (`#miForm="ngForm"`) para acceder a su estado.
-   Hemos a adido una referencia local a cada campo de entrada (`#nombreRef="ngModel"` y `#emailRef="ngModel"`) para acceder a su estado y validaci n.
-   Hemos a adido mensajes de error personalizados que se muestran cuando los campos son inv lidos y han sido tocados (es decir, el usuario ha interactuado con ellos).
-   Hemos deshabilitado el bot n de env o si el formulario no es v lido (`[disabled]="miForm.invalid"`).

Con estos cambios, nuestro formulario ahora valida los campos y muestra mensajes de error personalizados.

Si todos los campos son v lidos, el formulario se puede enviar y la funci n `onSubmit()` se ejecutar .

2.3. Ejemplo 2: tareas

Vamos a crear un nuevo componente llamado `tarea-form` que permita crear y editar objetos de tipo Tarea.

Una tarea tendr  los siguientes atributos: id, t tulo, descripci n, fecha de vencimiento y estado.

2.3.1. Paso 1: Crear el componente para el formulario de tarea

Ejecuta el siguiente comando para generar un nuevo componente llamado `tarea-form`:

```
ng generate component tarea-form
```

2.3.2. Paso 2: Definir la estructura del formulario en la plantilla del componente

Abre el archivo `tarea-form.component.html` y reemplaza su contenido con el siguiente c digo:

```

<form (ngSubmit)="onSubmit()" #tareaForm="ngForm">
  <input type="hidden" [(ngModel)]="tarea.id" name="id">
  <div>
    <label for="titulo">Título: </label>
    <input type="text" id="titulo" [(ngModel)]="tarea.titulo" name="titulo" required
    #tituloRef="ngModel">
    <div *ngIf="tituloRef.invalid && tituloRef.touched" class="error-message">
      El título es obligatorio.
    </div>
  </div>
  <div>
    <label for="descripcion">Descripción: </label>
    <textarea id="descripcion" [(ngModel)]="tarea.descripcion" name="descripcion"
    required #descripcionRef="ngModel"></textarea>
    <div *ngIf="descripcionRef.invalid && descripcionRef.touched" class="error-
    message">
      La descripción es obligatoria.
    </div>
  </div>
  <div>
    <label for="fechaVencimiento">Fecha de vencimiento: </label>
    <input type="datetime-local" id="fechaVencimiento"
    [(ngModel)]="tarea.fechaVencimiento" name="fechaVencimiento" required
    #fechaVencimientoRef="ngModel">
    <div *ngIf="fechaVencimientoRef.invalid && fechaVencimientoRef.touched"
    class="error-message">
      La fecha de vencimiento es obligatoria.
    </div>
  </div>
  <div>
    <label for="estado">Estado: </label>
    <select id="estado" [(ngModel)]="tarea.estado" name="estado" required #estadoRef
    ="ngModel">
      <option value="pendiente">Pendiente</option>
      <option value="en_progreso">En progreso</option>
      <option value="completada">Completada</option>
    </select>
    <div *ngIf="estadoRef.invalid && estadoRef.touched" class="error-message">
      El estado es obligatorio.
    </div>
  </div>

  <button type="submit" [disabled]="tareaForm.invalid">Guardar</button>
</form>

```

En este ejemplo, hemos creado un formulario para agregar y editar tareas con campos de entrada para título, descripción, fecha de vencimiento y estado.

También hemos utilizado la directiva `[(ngModel)]` para vincular cada campo de entrada a una propiedad del objeto tarea.

2.3.3. Paso 3: Vincular los elementos del formulario a las propiedades del componente

Abre el archivo `tarea-form.component.ts` y modifícalo de la siguiente manera:

```
import { Component, Input } from '@angular/core';

interface Tarea {
  id: number;
  titulo: string;
  descripcion: string;
  fechaVencimiento: string;
  estado: string;
}

@Component({
  selector: 'app-tarea-form',
  templateUrl: './tarea-form.component.html',
  styleUrls: ['./tarea-form.component.css']
})
export class TareaFormComponent {
  @Input() tarea: Tarea = {
    id: 0,
    titulo: '',
    descripcion: '',
    fechaVencimiento: '',
    estado: 'pendiente'
  };
  onSubmit() {
    console.log('Tarea:', this.tarea);
  }
}
```

En este ejemplo, hemos definido una interfaz `Tarea` y una propiedad `tarea` en nuestro componente, que está vinculada a los campos de entrada en nuestra plantilla mediante la directiva `[(ngModel)]`.

También hemos creado una función llamada `onSubmit()` que se ejecuta cuando el formulario se envía.

2.3.4. Mostrar el componente en la aplicación

Para ver el componente `tarea-form` en la aplicación, abre el archivo `app.component.html` y reemplaza su contenido con lo siguiente:

```
<app-tarea-form></app-tarea-form>
```

Ahora, ejecuta la aplicación con el comando `ng serve` y visita <http://localhost:4200> en tu navegador. Deberías ver el formulario de tarea que acabamos de crear.

Con este ejemplo, hemos creado un formulario de tareas utilizando el enfoque de formularios básicos o "template-driven".

3. Reactive Forms

Los formularios reactivos en Angular son una alternativa a los formularios basados en plantillas (template-driven) y ofrecen un enfoque más programático y basado en componentes para trabajar con formularios.

Con los formularios reactivos, se definen formularios y campos de formulario utilizando instancias de clases específicas (`FormGroup`, `FormControl`, `FormArray`), y se maneja la validación y el flujo de datos directamente en el componente.

3.1. Configuración inicial

Para empezar a trabajar con formularios reactivos, debes importar el módulo `ReactiveFormsModule` en tu módulo principal.

Abre el archivo `app.module.ts` y realiza los siguientes cambios:

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

@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule // Añadir ReactiveFormsModule aquí
  ],
  // ...
})
export class AppModule { }
```

3.2. Creando un formulario reactivo

Para crear un formulario reactivo, sigue estos pasos:

- ¥ Crea un componente para el formulario.
 - + Importa las clases necesarias en el componente (`FormGroup`, `FormControl`, etc.).
- ¥ Define la estructura del formulario en el componente utilizando instancias de las clases importadas.
- ¥ Vincula los elementos del formulario a las instancias de las clases en la plantilla del componente.

3.2.1. Paso 1: Crear un componente para el formulario

Ejecuta el siguiente comando para generar un nuevo componente llamado `mi-formulario-reactivo`:

```
ng generate component mi-formulario-reactivo
```

3.2.2. Paso 2: Importar las clases necesarias en el componente

Abre el archivo `mi-formulario-reactivo.component.ts` e importa las clases `FormGroup` y `FormControl` de `@angular/forms`:

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-mi-formulario-reactivo',
  templateUrl: './mi-formulario-reactivo.component.html',
  styleUrls: ['./mi-formulario-reactivo.component.css']
})
export class MiFormularioReactivoComponent {
  // ...
}
```

3.2.3. Paso 3: Definir la estructura del formulario en el componente

A continuaci3n, define la estructura del formulario utilizando instancias de `FormGroup` y `FormControl`.

En este ejemplo, vamos a crear un formulario simple con dos campos: nombre y email.

```
export class MiFormularioReactivoComponent {
  formulario = new FormGroup({
    nombre: new FormControl(''),
    email: new FormControl('')
  });
  // ...
}
```

En este ejemplo, hemos creado un objeto formulario que es una instancia de `FormGroup`.

Dentro de este objeto, hemos definido dos controles de formulario, nombre y email, que son instancias de `FormControl`.

Cada control de formulario se inicializa con un valor vac3o.

3.2.4. Paso 4: Vincular los elementos del formulario a las instancias de las clases en la plantilla del componente

Ahora, vamos a vincular los elementos del formulario en la plantilla html del componente a las instancias de las clases que hemos creado en el componente.

Abre el archivo `mi-formulario-reactivo.component.html` y reemplaza su contenido con el siguiente

código:

```
<form [formGroup]="formulario" (ngSubmit)="onSubmit()">
  <div>
    <label for="nombre">Nombre: </label>
    <input type="text" id="nombre" formControlName="nombre">
  </div>
  <div>
    <label for="email">Email: </label>
    <input type="email" id="email" formControlName="email">
  </div>

  <button type="submit">Enviar</button>
</form>
```

Aquí, hemos realizado los siguientes cambios:

- Hemos vinculado la instancia `FormGroup` en nuestro componente a nuestro formulario utilizando la directiva `[formGroup]="formulario"`.
- Hemos vinculado cada instancia `FormControl` en nuestro componente a los campos de entrada correspondientes utilizando la directiva `formControlName`.

También hemos añadido un evento `ngSubmit` que se activará cuando el formulario se envíe.

A continuación, vamos a implementar la función `onSubmit()` en nuestro componente.

3.2.5. Paso 5: Manejo de eventos de envío del formulario

Para manejar el evento de envío del formulario, implementa la función `onSubmit()` en el componente:

```
export class MiFormularioReactivoComponent {
  // ...
  onSubmit() {
    console.log('Formulario enviado:', this.formulario.value);
  }
}
```

En este ejemplo, simplemente estamos mostrando los valores del formulario en la consola cuando se envía el formulario.

3.2.6. Paso 6: Mostrar el componente en la aplicación

Para ver el componente `mi-formulario-reactivo` en la aplicación, abre el archivo `app.component.html` y reemplaza su contenido con lo siguiente:

```
<app-mi-formulario-reactivo></app-mi-formulario-reactivo>
```

Ahora, ejecuta la aplicación con el comando `ng serve` y visita <http://localhost:4200> en tu navegador. Deberás ver el formulario reactivo que acabamos de crear.

3.3. Validación de formularios reactivos

La validación de formularios reactivos es similar a la validación de formularios basados en plantillas.

Sin embargo, en lugar de utilizar atributos HTML y directivas específicas de Angular, se utilizan funciones validadoras proporcionadas por Angular en el componente.

Aquí hay un ejemplo de cómo agregar validación a nuestro formulario reactivo:

Importa las funciones de validación necesarias en el componente.

En este caso, importaremos `Validators` de `@angular/forms`.

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

Agrega las funciones de validación a los controles de formulario correspondientes en el componente.

```
formulario = new FormGroup({  
  nombre: new FormControl('', Validators.required),  
  email: new FormControl('', [Validators.required, Validators.email])  
});
```

En este ejemplo, hemos agregado la validación `required` al control de formulario nombre y las validaciones `required` y `email` al control de formulario email.

Actualiza la plantilla del componente para mostrar mensajes de error y deshabilitar el botón de envío si el formulario no es válido.

```
<form [formGroup]="formulario" (ngSubmit)="onSubmit()">  
  <!-- ... -->  
  <div *ngIf="formulario.get('nombre').invalid && formulario.get('nombre').touched"  
    class="error-message">  
    El nombre es obligatorio.  
  </div>  
  
  <div *ngIf="formulario.get('email').invalid && formulario.get('email').touched"  
    class="error-message">  
    <p *ngIf="formulario.get('email').errors.required">El email es  
    obligatorio. </p>
```

```

    <p *ngIf="formulario.get('email').errors.email">El email no es v lido.</p>
  </div>

  <button type="submit" [disabled]="formulario.invalid">Enviar</button>
</form>

```

En este ejemplo, hemos utilizado la funci n `get()` para acceder a los controles de formulario individuales y verificar su validez.

Tambi n hemos utilizado la propiedad `errors` para verificar los errores espec ficos y mostrar los mensajes de error apropiados.

Finalmente, hemos deshabilitado el bot n de env o si el formulario no es v lido utilizando la propiedad `invalid`.

3.4. Ejemplo 1

Vamos a crear un ejemplo donde se utiliza un formulario reactivo para gestionar objetos de tipo Tarea con m ltiples atributos, incluyendo uno de tipo fecha. Para ello, seguiremos estos pasos:

-   Crear un componente para el formulario de tarea.
-   Crear una interfaz para la Tarea.
-   Definir la estructura del formulario en el componente utilizando instancias de `FormGroup`, `FormControl`, y `Validators`.
-   Vincular los elementos del formulario en la plantilla del componente a las instancias de las clases en el componente.

3.4.1. Paso 1: Crear un componente para el formulario de tarea

Ejecuta el siguiente comando para generar un nuevo componente llamado `tarea-reactiva-form`:

```
ng generate component tarea-reactiva-form
```

3.4.2. Paso 2: Crear una interfaz para la Tarea

Abre el archivo `tarea-reactiva-form.component.ts` e implementa la interfaz `Tarea`:

```

import { Component } from '@angular/core';

interface Tarea {
  id: number;
  titulo: string;
  descripci n: string;
  // fechaVencimiento: string;
  fechaVencimiento: Date; // Cambiar a tipo Date
  estado: string;
}

```

```

}
@Component({
  selector: 'app-tarea-reactiva-form',
  templateUrl: './tarea-reactiva-form.component.html',
  styleUrls: ['./tarea-reactiva-form.component.css']
})
export class TareaReactivaFormComponent {
  // ...
}

```

3.4.3. Paso 3: Definir la estructura del formulario en el componente

A continuaci3n, importa `FormGroup`, `FormControl`, y `Validators`, y define la estructura del formulario utilizando estas clases:

```

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

export class TareaReactivaFormComponent {
  tareaForm = new FormGroup({
    id: new FormControl(null),
    titulo: new FormControl('', Validators.required),
    descripci3n: new FormControl('', Validators.required),
    // fechaVencimiento: new FormControl('', Validators.required),
    fechaVencimiento: new FormControl(null), // Utilizar un tipo de dato compatible
    estado: new FormControl('pendiente', Validators.required),
  });
  // ...
}

```

3.4.4. Paso 4: Vincular los elementos del formulario en la plantilla del componente

Abre el archivo `tarea-reactiva-form.component.html` y reemplaza su contenido con el siguiente c3digo:

```

<form [formGroup]="tareaForm" (ngSubmit)="onSubmit()">
  <input type="hidden" FormControlName="id">
  <div>
    <label for="titulo">T3tulo: </label>
    <input type="text" id="titulo" FormControlName="titulo">
    <div *ngIf="tareaForm.get('titulo').invalid && tareaForm.get('titulo').touched"
      class="error-message">
      El t3tulo es obligatorio.
    </div>
  </div>
  <div>
    <label for="descripci3n">Descripci3n: </label>
    <textarea id="descripci3n" FormControlName="descripci3n"></textarea>
  </div>

```

```

Ê   <div *ngIf="tareaForm.get('descripcion').invalid &&
tareaForm.get('descripcion').touched" class="error-message">
Ê     La descripción es obligatoria.
Ê   </div>
Ê </div>
Ê <div>
Ê   <label for="fechaVencimiento">Fecha de vencimiento: </label>
Ê   <input type="datetime-local" id="fechaVencimiento"
formControlName="fechaVencimiento">
<!--   <input type="date" id="fechaVencimiento" formControlName="fechaVencimiento">--
>
Ê   <div *ngIf="tareaForm.get('fechaVencimiento').invalid &&
tareaForm.get('fechaVencimiento').touched" class="error-message">
Ê     La fecha de vencimiento es obligatoria.
</div>
Ê </div>
Ê <div>
Ê   <label for="estado">Estado: </label>
Ê   <select id="estado" formControlName="estado">
Ê     <option value="pendiente">Pendiente</option>
Ê     <option value="en_progreso">En progreso</option>
Ê     <option value="completada">Completada</option>
Ê   </select>
Ê   <div *ngIf="tareaForm.get('estado').invalid && tareaForm.get('estado').touched"
class="error-message">
Ê     El estado es obligatorio.
Ê   </div>
Ê </div>

<button type="submit" [disabled]="tareaForm.invalid">Guardar Tarea</button>
</form>

```

En la plantilla, hemos creado un formulario con campos correspondientes a cada atributo de la interfaz Tarea.

También hemos añadido mensajes de error y deshabilitado el botón de envío si el formulario no es válido.

Agrega la función `onSubmit()` en el componente para manejar el envío del formulario:

```

export class TareaReactivaFormComponent {
Ê   // ...
Ê   onSubmit() {
Ê     // const tarea: Tarea = this.tareaForm.value;

Ê     // o capturar uno a uno los atributos:
Ê     const tarea: Tarea = {
Ê       id: this.tareaForm.value.id,
Ê       titulo: this.tareaForm.value.titulo,
Ê       descripcion: this.tareaForm.value.descripcion,

```

```

    fechaVencimiento: new Date(this.tareaForm.value.fechaVencimiento),
  };
  console.log('Tarea guardada:', tarea);
}
}

```

Esta función obtiene el valor del formulario y lo guarda en una variable tarea de tipo Tarea, luego muestra en consola la tarea guardada.

3.4.5. Paso 5: Mostrar el componente en la aplicación

Para ver el componente `tarea-reactiva-form` en la aplicación, abre el archivo `app.component.html` y reemplaza su contenido con lo siguiente:

```
<app-tarea-reactiva-form></app-tarea-reactiva-form>
```

Ahora, ejecuta la aplicación con el comando `ng serve` y visita <http://localhost:4200> en tu navegador. Deberías ver el formulario de tarea reactivo que acabamos de crear.

Este ejemplo demuestra cómo utilizar formularios reactivos en Angular para gestionar objetos de tipo Tarea con múltiples atributos, incluyendo uno de tipo fecha.

Los formularios reactivos ofrecen un enfoque más programático y escalable para trabajar con formularios en Angular, y son especialmente útiles cuando se trata de formularios complejos y dinámicos.

3.5. Ejemplo 2

En este ejemplo, crearemos un formulario reactivo en Angular que representa una asociación Many-to-Many entre dos modelos, `Estudiante` y `Curso`.

Un estudiante puede estar inscrito en varios cursos y un curso puede tener varios estudiantes inscritos.

Utilizaremos `FormGroup` anidado y `FormArray` para manejar esta estructura de datos en el formulario.

3.5.1. Paso 1: Crear un componente para el formulario de inscripción

Ejecuta el siguiente comando para generar un nuevo componente llamado `inscripcion-form`:

```
ng generate component inscripcion-form
```

3.5.2. Paso 2: Crear las interfaces para Estudiante y Curso

Abre el archivo `inscripcion-form.component.ts` e implementa las interfaces `Estudiante` y `Curso`:


```
import { Component } from '@angular/core';

interface Estudiante {
  id: number;
  nombre: string;
  cursos: Curso[];
}

interface Curso {
  id: number;
  nombre: string;
}

@Component({
  selector: 'app-inscripcion-form',
  templateUrl: './inscripcion-form.component.html',
  styleUrls: ['./inscripcion-form.component.css']
})
export class InscripcionFormComponent {
  // ...
}
```

3.5.3. Paso 3: Definir la estructura del formulario utilizando FormGroup, FormControl y FormArray

A continuaci3n, importa `FormGroup`, `FormControl`, `FormArray`, y `Validators`, y define la estructura del formulario utilizando estas clases:

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

export class InscripcionFormComponent {
  cursos: Curso[] = [
    { id: 1, nombre: 'Matemáticas' },
    { id: 2, nombre: 'Física' },
    { id: 3, nombre: 'Química' },
  ];

  inscripcionForm = new FormGroup({
    estudiante: new FormGroup({
      id: new FormControl(null),
      nombre: new FormControl('', Validators.required),
    }),
    cursos: new FormArray([]),
  });
  // ...
}
```

Aquí, hemos creado un `FormGroup` anidado para el estudiante y un `FormArray` para los cursos.

Tambi n hemos a adido algunos cursos de ejemplo en la propiedad `cursos`.

3.5.4. Paso 4: Vincular los elementos del formulario en la plantilla del componente

Abre el archivo `inscripcion-form.component.html` y reemplaza su contenido con el siguiente c digo:

```
<form [formGroup]="inscripcionForm" (ngSubmit)="onSubmit()">
  <div formGroupName="estudiante">
    <input type="hidden" formControlName="id">

    <div>
      <label for="nombre">Nombre del estudiante:</label>
      <input type="text" id="nombre" formControlName="nombre">
      <div *ngIf="inscripcionForm.get('estudiante.nombre').invalid &&
inscripcionForm.get('estudiante.nombre').touched" class="error-message">
        El nombre del estudiante es obligatorio.
      </div>
    </div>
  </div>

  <div formArrayName="cursos">
    <label>Cursos:</label>
    <div *ngFor="let curso of cursos; index as i">
      <input type="checkbox" [value]="curso" (change)="onCursoChange(curso,
$event.target.checked)">
      {{ curso.nombre }}
    </div>
  </div>

  <button type="submit" [disabled]="inscripcionForm.invalid">Inscribir</button>
</form>
```

En la plantilla, hemos creado un formulario con campos correspondientes a cada atributo de las interfaces Estudiante y Curso.

Adem s, hemos utilizado el `FormArray` para representar la asociaci n Many-to-Many entre estudiantes y cursos, mostrando una lista de casillas de verificaci n para cada curso.

Agrega la funci n `onCursoChange()` en el componente para manejar la selecci n y eliminaci n de cursos en el `FormArray`:

```
export class InscripcionFormComponent {
  // ...

  onCursoChange(curso: Curso, isChecked: boolean) {
    const cursosArray = this.inscripcionForm.get('cursos') as FormArray;

    if (isChecked) {
```

```

    cursosArray.push(new FormControl(curso));
  } else {
    const index = cursosArray.controls.findIndex(x => x.value.id === curso.id);
    cursosArray.removeAt(index);
  }
}
}
}

```

Esta funci3n agrega o elimina un curso en el `FormArray` de cursos seg3n si la casilla de verificaci3n correspondiente est3 seleccionada o no.

Agrega la funci3n `onSubmit()` en el componente para manejar el env3o del formulario:

```

export class InscripcionFormComponent {
  // ...

  onSubmit() {
    const inscripcion: Estudiante = this.inscripcionForm.value.estudiante;
    inscripcion.cursos = this.inscripcionForm.value.cursos;

    console.log('Inscripci3n realizada:', inscripcion);
  }
}

```

Esta funci3n obtiene el valor del formulario y lo guarda en una variable `inscripcion` de tipo `Estudiante`, luego muestra en consola la inscripci3n realizada.

3.5.5. Mostrar el componente en la aplicaci3n

Para ver el componente `inscripcion-form` en la aplicaci3n, abre el archivo `app.component.html` y reemplaza su contenido con lo siguiente:

```

<app-inscripcion-form></app-inscripcion-form>

```

Ahora, ejecuta la aplicaci3n con el comando `ng serve` y visita <http://localhost:4200> en tu navegador. Deber3as ver el formulario de inscripci3n que representa la asociaci3n Many-to-Many entre estudiantes y cursos utilizando `FormGroup` anidado y `FormArray`.

Este ejemplo demuestra c3mo utilizar formularios reactivos en Angular para manejar estructuras de datos m3s complejas y asociaciones Many-to-Many entre modelos. La combinaci3n de `FormGroup` anidado y `FormArray` proporciona una soluci3n flexible y escalable para manejar este tipo de casos de uso.