



► 4. Formularios Reactivos

Objetivos

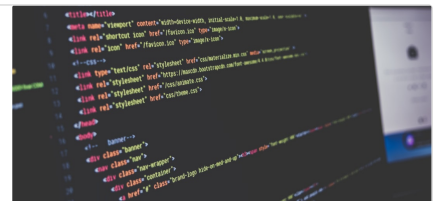
- Crear los formularios por código
- Aplicar validaciones personalizadas
- Dividir grandes formularios en otros menores

Referencia

How to Validate Angular Reactive Forms

IntroductionIn this article, we will learn about validations in reactive forms in Angular. We will create a simple user registration form and implement some inbuilt validations on it. Along with the inbuilt validations, we will also implement some custom validations to

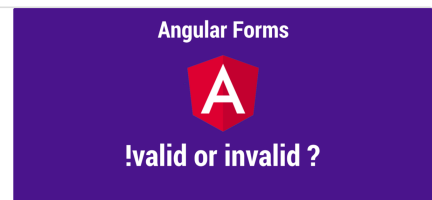
 <https://www.freecodecamp.org/news/how-to-validate-angular-reactive-forms/>



Valid and Invalid in Angular Forms


In Angular one of most common ways to validate forms is to disable the submit button. It's a nice way (alongside with helpful messages) to indicate the user that something is not right in the form. So to disable the button we use something like this So above there

 <https://itnext.io/valid-and-invalid-in-angular-forms-61cfa3f2a0cd>



Implementing reusable and reactive forms in Angular - Angular inDepth


In this article we will learn about two ways to implement reactive and reusable forms that have the capability to be used as sub-forms and also be used as standalone forms. I assume you know what forms are and you have worked with Angular reactive forms.

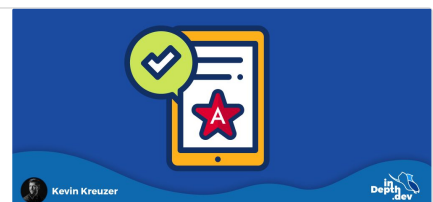
 <https://indepth.dev/posts/1415/implementing-reusable-and-reactive-forms-in-angular-2>



The best way to implement custom validators - Angular inDepth

Forms are an essential part of every Angular application and necessary to get data from the user. But, collected data is only valuable if the quality is right - means if it meets our criteria. In Angular, we can use Validators to validate the user input.

 <https://indepth.dev/posts/1319/the-best-way-to-implement-custom-validators>



Angular: Nested Reactive Forms Using ControlValueAccessors(CVAs) - Angular inDepth

In this post, we will see how to implement nested reactive forms using composite control value accessors(CVAs). Kara Erickson of the Angular Core Team presented this approach at the Angular Connect 2017 .

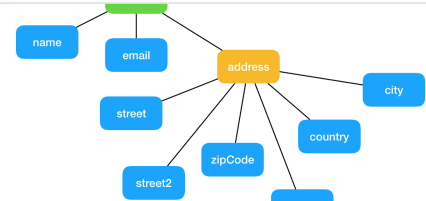
<https://indepth.dev/posts/1245/angular-nested-reactive-forms-using-controlvalueaccessors-cvas>



Nesting Forms in Angular

Greetings! Look at the image above and try to notice that how many form groups we need here? What should be the form output structure? New to Reactive forms? Please visit official docs. Okay! Now you have a structure in your mind. I'm quite confident that

<https://blog.koverhoop.com/nesting-forms-in-angular-1aabf93bd271>



▶ 1 - Construcción de formularios por código

Agregar ResourceNew Page con su controlador y servicio

```
<section>
  <ab-ui-header [header]="header"></ab-ui-header>
  <ab-resource-new *ngIf="categories$ | async as categories"
    [categories]="categories" (send)="onSend($event)">
  </ab-resource-new>
</section>

export class ResourceNewPage implements OnInit {
  categories$: Observable<Category[]>;
  header = {
    heroClass: 'is-warning',
    title: 'Add a new resource',
    subtitle: 'Help us complete the Angular Builders Catalog.',
  };

  constructor(private service: ResourceNewService) {}

  ngOnInit(): void {
    this.categories$ = this.service.getCategories$();
  }
  onSend(newResource: Resource) {
    this.service.postNewResource$(newResource).subscribe();
  }
}

export class ResourceNewService {
  private readonly categoriesUrl = `https://api-angularbuilders.herokuapp.com/v1/categories`;
  private readonly resourcesUrl = `https://api-angularbuilders.herokuapp.com/v1/resources`;

  constructor(private http: HttpClient) {}

  getCategories$() {
    return this.http
      .get<{ data: Category[] }>(this.categoriesUrl)
      .pipe(map((result) => result.data));
  }
  postNewResource$(newResource: Resource) {
    return this.http
      .post<{ data: Resource }>(`${this.resourcesUrl}/`, newResource)
```

```

        .pipe(map((result) => result.data));
    }
}

```

Form Builder

Importar el ReactiveFormsModule

```

@NgModule({
  imports: [
    CommonModule,
    RouterModule.forChild([
      {
        path: '',
        pathMatch: 'full',
        canActivate: [AuthGuard],
        component: ResourceNewPage,
      },
    ]),
    UiModule,
    ReactiveFormsModule,
  ],
  declarations: [ResourceNewPage, ResourceNewForm],
})
export class ResourceNewModule {}

```

Construir el formulario con la dependencia del Form Builder

```

export class ResourceNewForm implements OnInit {
  @Input() categories: Category[] = [];
  @Output() send = new EventEmitter<Resource>();
  form!: FormGroup;

  constructor(private fb: FormBuilder) {}

  ngOnInit(): void {
    this.form = this.fb.group({
      categoryId: new FormControl('', [Validators.required]),
      resourcename: new FormControl('', [
        Validators.required,
        Validators.minLength(2),
      ]),
      description: new FormControl('', [Validators.minLength(3)]),
      url: new FormControl('', []),
    });
  }

  onSubmit() {
    this.send.next(this.form.value);
  }

  hasErrorToShow(formControlName: string) {
    const control = this.form.controls[formControlName];
    return control.touched && control.invalid;
  }

  getErrorMessage(formControlName: string) {
    const control = this.form.controls[formControlName];
    return JSON.stringify(control.errors);
  }
}

```

Asignar el formulario y los controles en la plantilla html

```
<form [formGroup]="form" (submit)="onSubmit()" class="box mt-4 mb-4" autocomplete="off">
  <fieldset class="mb-4">
    <div class="field mt-4 ">
      <label for="categoryId" class="label"> A category </label>
      <div class="select"><select id="categoryId" formControlName="categoryId" class="select"
        #selector>
        <option *ngFor="let category of categories"
          [value]="category.id" #selector>
          {{ category.name }}
        </option>
      </select></div>
    </div>
    <div class="field mt-4 ">
      <label for="resourcename" class="label"> The resource commercial name </label>
      <input formControlName="resourcename"
        name="resourcename"
        id="resourcename"
        placeholder="The ngSuperTool"
        class="input" autocomplete="off" />
      <span *ngIf="hasErrorToShow('resourcename')">
        <em> {{ getErrorMessage('resourcename') }} </em>
      </span>
    </div>
    <div class="field mt-4 ">
      <label for="description" class="label"> A short description </label>
      <input formControlName="description"
        name="description"
        id="description"
        placeholder="Cool things about the resource"
        class="input" autocomplete="off" />
      <span *ngIf="hasErrorToShow('description')">
        <em> {{ getErrorMessage('description') }} </em>
      </span>
    </div>
    <div class="field mt-4 ">
      <label for="url" class="label"> The url of the home page or repository </label>
      <input formControlName="url"
        type="url"
        name="url"
        id="url"
        placeholder="https://cool.dev"
        class="input" autocomplete="off" />
    </div>
  </fieldset>
  <input type="submit" value="Add to the catalog!" [disabled]="form.invalid"
    class="button is-link mt-4" />
</form>
```

2 - Controles para formularios reactivos (CVA)

Crear componentes que puedan actuar como controles de un formulario reactivo

```
<div class="field mt-4">
  <label [for]="formControlName" class="label">
    {{ label }} <span *ngIf="hasError()">*</span>
  </label>
  <input [name]="formControlName">
```

```

        [type]="type"
        [id]="formControlName"
        [value]="value"
        (input)="onInput($event)"
        [placeholder]="placeholder"
        class="input" autocomplete="off" />
<span *ngIf="hasErrorToShow()">
  <em> {{ getErrorMessage() }} </em>
</span>
</div>

```

Implementar ControlValueAccesor y proveer NG_VALUE_ACCESSOR

```

@Component({
  selector: 'ab-form-control',
  templateUrl: './control.component.html',
  styles: [],
  changeDetection: ChangeDetectionStrategy.OnPush,
  providers: [
    {
      provide: NG_VALUE_ACCESSOR,
      useExisting: forwardRef(() => ControlComponent),
      multi: true,
    },
  ],
})
export class ControlComponent implements ControlValueAccessor {
  @Input() form!: FormGroup;
  @Input() formControlName = '';
  @Input() label = '';
  @Input() placeholder = '';
  @Input() type = 'text';

  value: unknown;
  emitChange: any;
  emitTouch: any;
  constructor() {}

  writeValue(obj: any): void {
    this.value = obj;
  }
  registerOnChange(fn: any): void {
    this.emitChange = fn;
  }
  registerOnTouched(fn: any): void {
    this.emitTouch = fn;
  }

  onInput(event: any) {
    this.value = event.target.value;
    this.emitChange(this.value);
    this.emitTouch();
  }

  hasErrorToShow() {
    const control = this.form.controls[this.formControlName];
    return control.touched && control.invalid;
  }
  hasError() {
    const control = this.form.controls[this.formControlName];
    return control.invalid;
  }
  getErrorMessage() {
    const control = this.form.controls[this.formControlName];

```

```

    return JSON.stringify(control.errors);
  }
}

```

El componente principal que da más limpio. Pero lo mejor es tener centralizada la presentación de todos los controles.

```

<ab-form-control
  [form]="form"
  formControlName="resourcenam"
  type="text"
  label="The resource commercial name"
  placeholder="The ngSuperTool">
</ab-form-control>
<ab-form-control
  [form]="form"
  formControlName="description"
  type="text"
  label="A short description"
  placeholder="Cool things about the resource">
</ab-form-control>
<ab-form-control
  [form]="form"
  type="url"
  formControlName="url"
  label="The home page"
  placeholder="https://www.ng-super-tool.dev">
</ab-form-control>

```

▶ 3 - Subformularios

👨‍🔧 Sub Formulario para cursos

```

<form [formGroup]="form" autocomplete="off">
  <ab-form-control
    [form]="form"
    formControlName="date"
    type="date"
    label="The starting day">
  </ab-form-control>
  <ab-form-control
    [form]="form"
    formControlName="teacher"
    type="text"
    label="The instructor name"
    placeholder="Jane Doe">
  </ab-form-control>
  <ab-form-control
    [form]="form"
    formControlName="academy"
    type="text"
    label="The academy or online platform"
    placeholder="The best place to learn Angular">
  </ab-form-control>
</form>
export class CourseForm {
  form!: FormGroup;
}

```

```

constructor(private fb: FormBuilder) {}
buildForm() {
  this.form = this.fb.group({
    date: new FormControl(''),
    teacher: new FormControl(''),
    academy: new FormControl(''),
  });
  return this.form;
}
}

```



La incrustación en el formulario padre se hace por el lado de la vista (lo normal) pero *también* por el controlador.

```

<ab-course-form [hidden]="form.value.categoryId !== 'courses'"></ab-course-form>
@ViewChild(CourseForm, { static: true }) courseSubForm!: CourseForm;
...
ngOnInit(): void {
  this.form = this.fb.group({
    categoryId: new FormControl('', [Validators.required]),
    resourcename: new FormControl('', [Validators.required, Validators.minLength(2)]),
    description: new FormControl('', [Validators.minLength(3)]),
    url: new FormControl('', []),
    course: this.courseSubForm.buildForm(),
  });
}
...
if (newResource.categoryId !== 'courses') {
  delete newResource.course;
}
}

```

4 - Validaciones propias y personalizadas


A cada control se le puede asignar una validación predefinida o crear una personalizada. Para las segundas creamos un servicio propio.

```
ng g s services/item-validators --project=add-item
```

Funciones validadoras.

Devuelven null cuando todo va bien. Y otras cosas... cuando falla.



 Factorías de funciones para para recibir parámetros

```

import { AbstractControl, FormGroup } from '@angular/forms';
type functionValidator = (control: AbstractControl) => any | null;
type functionFormValidator = (formGroup: FormGroup) => any | null;

```

```

export class AbValidators {
  static realisticDate(control: AbstractControl): any | null {
    const value = new Date(control.value);
    const today = new Date();
    const min = new Date(today.setFullYear(today.getFullYear() - 1));
    const max = new Date(today.setFullYear(today.getFullYear() + 1));
    if (value < min || value > max) {
      return {
        'realistic-date': 'One year before or after today',
      };
    }
    return null;
  }

  static includes(expected: string): functionValidator {
    return function (control: AbstractControl): any | null {
      const value = control.value as string;
      if (
        value &&
        value.toLocaleLowerCase().includes(expected.trim().toLocaleLowerCase())
      ) {
        return null;
      } else {
        return { includes: expected + ' not found' };
      }
    };
  }

  static confirmed(
    controlName: string,
    matchingControlName: string
  ): functionFormValidator {
    return function (formGroup: FormGroup): any | null {
      const control = formGroup.controls[controlName];
      const matchingControl = formGroup.controls[matchingControlName];
      if (control.value !== matchingControl.value) {
        return { confirmedValidator: true };
      } else {
        return null;
      }
    };
  }
}

```

Asignamos los validadores a cada control o al propio formulario (si los hubiera)

```

// course form...
buildGroup() {
  this.form = this.fb.group({
    date: new FormControl('', ABValidators.realisticDate),
    teacher: new FormControl(''),
    academy: new FormControl(''),
  });
  return this.form;
}

// main form
ngOnInit(): void {
  this.form = this.fb.group(
    {
      categoryId: new FormControl('aa', [Validators.required]),
      resource: new FormControl('aaa', [
        Validators.required,

```



```

        Validators.minLength(2),
    ]),
    description: new FormControl('', [Validators.minLength(3)]),
    url: new FormControl('https://', [AbValidators.includes('https://')]),
    course: this.courseSubForm.buildForm(),
    price: new FormControl(0, []),
    confirmPrice: new FormControl(0, []),
  },
  {
    validators: [AbValidators.confirmed('price', 'confirmPrice')],
    updateOn: 'blur',
  }
);
}
}
}
---

```



Alberto Basalo