

# Les formulaires dans Angular

Framework Front-end

## Table des matières

1.	La dualité des validations .....	2
2.	FormBuilder.....	3
3.	Dans mon formulaire.....	4
4.	Évènements.....	6
5.	Et maintenant ? .....	6

## 1. La dualité des validations

Avant de se lancer dans la conception d'un formulaire sur Angular, il faut que nous abordions un aspect important de la sécurité : les **vérifications**.

Les vérifications consistent en un nombre plus ou moins important de **barrières** que nous allons mettre en place pour un nombre de raisons :

- **La sécurité**, qui peut paraître comme étant le premier critère évident. En effet, on ne veut pas se retrouver à avoir des millions d'envoi d'e-mails depuis notre formulaire de contact parce qu'un bot est passé par là.
- **L'expérience utilisateur** est également primordiale. En effet, on ne veut pas que l'utilisateur puisse saisir n'importe quoi dans nos formulaires car cela risque de rendre son expérience future problématique. Par exemple, si on le laisse taper n'importe quoi dans le champ « adresse e-mail », il peut ne jamais recevoir l'e-mail de confirmation de son compte.

Ces **barrières** peuvent être classées en deux catégories que je vais détailler ici :

- Les vérifs **back-end**, c'est-à-dire tout ce qui va être effectué côté serveur pour s'assurer de ne pas faire n'importe quoi. Par exemple : Vérifier qu'on a bien des données dans le tableau **\$\_POST** avant de traiter une insertion en base de données, vérifier qu'on a bien une adresse e-mail en base de données avant d'envoyer un e-mail etc...
- Les vérifs **front-end**, c'est-à-dire tout ce qui va être effectué côté client. Par exemple, l'attribut **required** dans une balise **input**, vérifier si le champ est vide ou non avant d'autoriser à cliquer sur le bouton envoyer, etc.

Angular étant un framework front-end, il ne nous permettra pas de gérer les vérifs **back-end**.

*En revanche, il est armé jusqu'aux dents pour gérer les vérifs **front-end***

Reprendons alors le formulaire que nous avons intégré sur le projet **demo-router**, et tâchons de le blinder un peu afin de pouvoir passer la balle côté serveur.

## 2. FormBuilder

FormBuilder est une classe du moteur Angular qui nous permet de **bâtir** (littéralement) un formulaire en tant qu'**objet**, accompagné de toutes ses vérifications. Cette classe va donc devoir être importée avec un certain nombre de classes **annexes**.

### formulaire.ts

```
//Import des classes dont je vais avoir besoin pour gérer mon formulaire
import { FormBuilder, FormGroup, ReactiveFormsModuleModule, Validators } from '@angular/forms';
```

Nous allons en plus de ça avoir besoin d'importer ReactiveFormsModule dans le tableau des imports car nous allons nous en servir dans le template :

### formulaire.ts

```
@Component({
  selector: 'app-formulaire',
  imports: [ReactiveFormsModule],
  templateUrl: './formulaire.html',
  styleUrls: ['./formulaire.scss'],
})
```

Toujours dans notre classe formulaire.ts, nous allons créer une propriété de type **FormGroup** que nous allons appeler **formulaire**. C'est cette propriété qui va nous permettre de contenir notre objet formulaire.

```
export class Formulaire {

  // On contient notre formulaire ici
  formulaire: FormGroup;
```

Enfin, nous allons **injecter** la classe FormBuilder dans une propriété de notre classe, grâce au **constructeur**.

Ce FormBuilder va nous permettre de construire un **FormGroup**, c'est à dire une association de champs avec leur valeur par défaut, puis leurs vérifications à effectuer :

```
// Et on injecte notre FormBuilder
constructor(private fb: FormBuilder){
  this.formulaire = this.fb.group({
    nom: ['', [Validators.required, Validators.minLength(3)]]
  });
}
```

Chaque clé du tableau ici est le nom d'un champ, suivi de respectivement : sa valeur par défaut, et enfin un tableau de **validation à effectuer**, caractérisées par des objets de la classe **Validators**. Voici un tableau de toutes les vérifs possibles avec la classe **Validators** :

Validateur	Utilité	Exemple
required	Champ obligatoire	Validators.required
requiredTrue	Checkbox/radio doit être cochée	Validators.requiredTrue
email	Format email valide	Validators.email
minLength(n)	Minimum n caractères	Validators.minLength(3)
maxLength(n)	Maximum n caractères	Validators.maxLength(50)
pattern(regex)	Correspond à un motif	Validators.pattern(/^\d{5}\$/)
min(n)	Nombre minimum	Validators.min(0)
max(n)	Nombre maximum	Validators.max(100)

### 3. Dans mon formulaire

Grâce à ces vérifications données par la classe **Validators**, je vais pouvoir intégrer des **tests** dans mon template. Par exemple, reprenons les vérifications du champ « nom », nous l'avons mis en **required**, mais également en **minLength(3)**.

Premièrement, nous devons faire correspondre l'input HTML avec son nom dans le **FormGroup**. Pour cela, on lui ajout l'attribut **formControlName**, comme ceci :

```
<input type="text" class="form-control" formControlName="nom"
       placeholder="Votre nom"
       required
/>
```

Grâce à ça, Angular est **déjà** capable de vérifier l'**état de validité** de l'input. Nous pouvons alors gérer des messages d'erreur que nous allons afficher en dessous du formulaire, comme ceci :

```
@if (formulaire.get('nom')?.hasError('required') && formulaire.get('nom')?.touched) {
  <div class="text-danger small mt-1">Le nom est requis</div>
}
@if (formulaire.get('nom')?.hasError('minlength') && formulaire.get('nom')?.touched) {
  <div class="text-danger small mt-1">Le nom doit contenir au moins 3 caractères</div>
}
```

Ici nous avons deux vérifications faites grâce à la méthode **hasError** :

- **hasError('required')** nous permet de vérifier si le champ est bien rempli
- **hasError('minlength')** nous permet de vérifier si le champ répond bien à sa contrainte minLength(3)

Enfin, on vérifie si notre champ a été modifié avec l'attribut **touched** qui sera true ou non, afin de n'afficher le message d'erreur que si l'input a été modifié : On ne veut pas disputer le client dès qu'il arrive sur le site alors qu'il n'a encore rien fait !

**FACULTATIF :** Nous pouvons également ajouter une classe bootstrap **is-invalid** qui entourera l'input en rouge si il est invalide. Pour faire dépendre cette classe de la validité du champ, on peut utiliser la notation suivante :

```
[class.is-invalid]="formulaire.get('nom')?.invalid && formulaire.get('nom')?.touched"
```

Ce qui nous donne un input complet ressemblant à ceci :

```
<div class="col-md-6">
  <label for="nom" class="form-label">Nom</label>
  <input
    type="text"
    class="form-control"
    formControlName="nom"
    [class.is-invalid]="formulaire.get('nom')?.invalid && formulaire.get('nom')?.touched"
    placeholder="Votre nom"
    required
  />
  @if (formulaire.get('nom')?.hasError('required') && formulaire.get('nom')?.touched) {
    <div class="text-danger small mt-1">Le nom est requis</div>
  }
  @if (formulaire.get('nom')?.hasError('minlength') && formulaire.get('nom')?.touched) {
    <div class="text-danger small mt-1">Le nom doit contenir au moins 3 caractères</div>
  }
</div>
```

*Je vous laisse recommencer l'opération avec les inputs email, sujet, message 😊*

## 4. Évènements

Lorsque l'on veut mettre en place des évènements dans un formulaire, nous avons accès aux mêmes évènements Javascript que d'habitude. Par exemple, si nous voulons détecter l'évènement « change » sur un input, nous allons faire de l'**event binding** sur ce même évènement, comme ceci par exemple :

```
(change)="maFonction()"
```

Mais on veut parfois pouvoir transmettre des paramètres lors d'un appel de fonction. Au hasard : la valeur de l'input qui a été modifié !

Pour cela nous avons recours à l'objet **\$event**, qui sera un objet de la classe Event. Et depuis cet objet, nous avons accès à sa propriété **target**. Nous allons alors **caster** cette propriété **target** comme un objet de type **HTMLSelectElement**, ou **HTMLInputElement** qui portera la propriété **value**.

Démonstration dans le HTML :

```
<select id="region" class="form-control"
        (change)="fetchDepartements($event)"
```

Et dans le TypeScript :

```
fetchDepartements(e: Event) {
    const value = (e.target as HTMLSelectElement).value;
```

## 5. Et maintenant ?

Maintenant que les vérifications sont en place sur nos inputs, nous allons pouvoir nous attaquer aux balises restantes dans notre formulaire : le **button** et la balise **form**

Premièrement, on va pouvoir désactiver notre bouton tant que le formulaire n'est pas valide. Pour cela, rien de plus simple, on va faire de l'attribute binding avec l'attribut **disabled** de notre bouton, et faire dépendre cet attribut de la valeur de notre formulaire, comme ceci :

```
<button type="submit" class="btn btn-primary btn-lg" [disabled]="formulaire.invalid">
    Envoyer le message
</button>
```

Ensuite, nous devons associer notre formulaire avec le FormGroup que nous avons créé dans la propriété formulaire, et gérer ce que va faire notre formulaire une fois envoyé. Pour cela, nous allons faire de l'**event binding** avec l'évènement ngSubmit, et surtout de l'attribute binding avec l'attribut **formGroup** de mon formulaire, comme ceci :

```
<form [formGroup]="formulaire" (ngSubmit)="soumettre()">
```

*Pourquoi faire l'event binding sur ngSubmit et pas submit tout court ?*

*Parce que ngSubmit est un évènement spécifique à Angular qui va gérer un certain nombre de choses qui nous arrange bien : bloquer l'envoi du formulaire si il n'est pas à l'état « valide », empêcher le recharge de la page (comportement par défaut du navigateur) et surtout va s'exécuter APRES la validation du formulaire contrairement à submit qui est l'évènement natif du navigateur auquel on devrait ajouter un e.preventDefault etc...*

Il ne nous reste plus maintenant qu'à coder la méthode « soumettre() » que nous venons d'associer à l'évènement **ngSubmit**, comme ceci :

```
soumettre(): void {
  if (this.formulaire.valid) {
    console.log('Formulaire valide !');
    console.log(this.formulaire.value);
  } else {
    console.log('Formulaire invalide');
  }
}
```

En testant, on retrouvera bien en console tous les champs renseignés, ou alors un message d'erreur si il en manque.

Fichiers complets en annexe :

### formulaire.ts

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

//Import des classes dont je vais avoir besoin pour gérer mon formulaire
import { FormBuilder, FormGroup, ReactiveFormsModuleModule, Validators } from
'@angular/forms';
@Component({
  selector: 'app-formulaire',
  imports: [ReactiveFormsModule],
  templateUrl: './formulaire.html',
  styleUrls: ['./formulaire.scss'],
})
export class Formulaire {

  // On contient notre formulaire ici
  formulaire: FormGroup;
  // Et on injecte notre FormBuilder
  constructor(private fb: FormBuilder){
    this.formulaire = this.fb.group({
      nom: ['', [Validators.required, Validators.minLength(3)]],
      email: ['', [Validators.required, Validators.email]],
      sujet: ['', [Validators.required]],
      message: ['', [Validators.required, Validators.minLength(10)]]});
  }

  soumettre(): void {
    if (this.formulaire.valid) {
      console.log('Formulaire valide !');
      console.log(this.formulaire.value);
    } else {
      console.log('Formulaire invalide');
    }
  }
}
```

## formulaire.html

```
<!-- Formulaire de contact -->
<section class="bg-light py-5 rounded-3 mt-5">
  <div class="row justify-content-center">
    <div class="col-lg-8">
      <h2 class="h3 mb-4 text-center">Envoyez-nous un message</h2>
      <form [formGroup]="formulaire" (ngSubmit)="soumettre()">
        <div class="row mb-3">
          <div class="col-md-6">
            <label for="nom" class="form-label">Nom</label>
            <input
              type="text"
              id="nom"
              class="form-control"
              formControlName="nom"
              [class.is-invalid]="formulaire.get('nom')?.invalid &&
formulaire.get('nom')?.touched"
              placeholder="Votre nom"
              required
            />
            @if (formulaire.get('nom')?.hasError('required') && formulaire.get('nom')?.touched) {
              <div class="text-danger small mt-1">Le nom est requis</div>
            }
            @if (formulaire.get('nom')?.hasError('minlength') && formulaire.get('nom')?.touched) {
              <div class="text-danger small mt-1">Le nom doit contenir au moins 3
caractères</div>
            }
          </div>
          <div class="col-md-6">
            <label for="email" class="form-label">Email</label>
            <input
              type="email"
              id="email"
              class="form-control"
              formControlName="email"
              [class.is-invalid]="formulaire.get('email')?.invalid &&
formulaire.get('email')?.touched"
              placeholder="Votre email"
              required
            />
            @if(formulaire.get('email')?.hasError('required') && formulaire.get('email')?.touched)
{
              <div class="text-danger small mt-1">L'email est requis</div>
}
            @if(formulaire.get('email')?.hasError('email') && formulaire.get('email')?.touched) {
              <div class="text-danger small mt-1">L'email est invalide</div>
}
          </div>
        </div>
      </form>
    </div>
  </div>
</section>
```

```
</div>

<div class="mb-3">
  <label for="sujet" class="form-label">Sujet</label>
  <input
    type="text"
    id="sujet"
    class="form-control"
    [class.is-invalid]="formulaire.get('sujet')?.invalid && formulaire.get('sujet')?.touched"
    formControlName="sujet"
    placeholder="Sujet de votre message"
    required
  />
  @if(formulaire.get('sujet')?.hasError('required') && formulaire.get('sujet')?.touched) {
    <div class="text-danger small mt-1">Le sujet est requis</div>
  }
</div>

<div class="mb-3">
  <label for="message" class="form-label">Message</label>
  <textarea
    class="form-control"
    id="message"
    formControlName="message"
    [class.is-invalid]="formulaire.get('message')?.invalid &&
formulaire.get('message')?.touched"
    rows="5"
    placeholder="Votre message..."
    required
  ></textarea>
  @if(formulaire.get('message')?.hasError('required') && formulaire.get('message')?.touched)
  {
    <div class="text-danger small mt-1">Le message est requis</div>
  }
  @if(formulaire.get('message')?.hasError('minlength') && formulaire.get('message')?.touched)
  {
    <div class="text-danger small mt-1">Le message doit faire au moins 10 caractères</div>
  }
</div>

<div class="d-grid">
  <button type="submit" class="btn btn-primary btn-lg" [disabled]="formulaire.invalid">
    Envoyer le message
  </button>
</div>
</form>
</div>
</div>
</section>
```