

Cours Angular : Les Formulaires

1. Introduction

Les **formulaires** sont essentiels dans une application Angular : authentification, inscription, saisie de données, recherche, etc.

Angular propose **deux approches complémentaires** : - **Template-driven forms** (simples) - **Reactive forms** (professionnelles et scalables)

Objectifs : - Comprendre les deux types de formulaires - Savoir quand utiliser chaque approche - Gérer la validation - Manipuler l'état du formulaire - Appliquer les bonnes pratiques en entreprise

2. Template-driven Forms

2.1 Principe

Les formulaires pilotés par le template reposent principalement sur : - HTML - `ngModel`

👉 Adaptés aux **petits formulaires simples**.

2.2 Configuration

Dans `app.module.ts` :

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

@NgModule({
  imports: [FormsModule]
})
export class AppModule {}
```

2.3 Exemple simple

```
<form #loginForm="ngForm" (ngSubmit)="onSubmit(loginForm)">
  <input type="email" name="email" ngModel required>
  <input type="password" name="password" ngModel required minlength="6">

  <button type="submit" [disabled]="loginForm.invalid">Connexion</button>
</form>
```

```
onSubmit(form: any) {
  console.log(form.value);
}
```

2.4 Validation

- required
- minlength, maxlength
- email

```
<input name="email" ngModel required email>
```

3. Reactive Forms (Formulaires réactifs)

3.1 Principe

Les **Reactive Forms** sont basés sur : - TypeScript - Observables - Une structure déclarative

👉 Recommandés pour les **applications professionnelles**.

3.2 Configuration

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

@NgModule({
  imports: [ReactiveFormsModule]
})
export class AppModule {}
```

3.3 Crédit d'un formulaire

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

loginForm = new FormGroup({
  email: new FormControl('', [Validators.required, Validators.email]),
  password: new FormControl('', [Validators.required,
  Validators.minLength(6)])
});
```

Template :

```
<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">
  <input type="email" formControlName="email">
  <input type="password" formControlName="password">

  <button type="submit" [disabled]="loginForm.invalid">Connexion</button>
</form>
```

4. FormBuilder (bonne pratique)

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

constructor(private fb: FormBuilder) {}

loginForm = this.fb.group({
  email: ['', [Validators.required, Validators.email]],
  password: ['', [Validators.required, Validators.minLength(6)]]
});
```

5. Validation des formulaires

5.1 États d'un contrôle

- valid / invalid
 - touched / untouched
 - dirty / pristine

5.2 Messages d'erreur

```
<div *ngIf="loginForm.get('email')?.invalid &&
loginForm.get('email')?.touched">
    Email invalide
</div>
```

6. Validateurs personnalisés

6.1 Validator synchrone

```
import { AbstractControl, ValidationErrors } from '@angular/forms';

export function strongPassword(control: AbstractControl): ValidationErrors |
```

```
null {
  return control.value.length >= 8 ? null : { weakPassword: true };
}
```

Utilisation :

```
password: ['', [strongPassword]]
```

6.2 Validator asynchrone

```
checkEmail(control: AbstractControl) {
  return this.http.get(`/api/check-email/${control.value}`);
}
```

7. FormArray

Permet de gérer des champs dynamiques.

```
skills = this.fb.array([
  this.fb.control('')
]);
```

```
<div formArrayName="skills">
  <div *ngFor="let skill of skills.controls; let i = index">
    <input [formControlName]= "i">
  </div>
</div>
```

8. Suivi des changements

```
this.loginForm.valueChanges.subscribe(value => {
  console.log(value);
});
```

9. Cas pratique complet

Objectif

Formulaire d'inscription avec validation.

```
registerForm = this.fb.group({
  name: ['', Validators.required],
  email: ['', [Validators.required, Validators.email]],
  password: ['', Validators.required],
  confirmPassword: ['', Validators.required]
});
```

10. Template-driven vs Reactive

Template-driven	Reactive Forms
Simple	Structuré
Peu de TS	Beaucoup de TS
Peu scalable	Très scalable
Petits projets	Projets pro

11. Bonnes pratiques

- Privilégier **Reactive Forms**
- Centraliser la validation
- Toujours afficher les erreurs utilisateur
- Ne jamais faire de logique métier dans le template

12. Résumé

- Angular propose 2 types de formulaires
- Reactive Forms = standard professionnel
- Validation puissante et extensible
- Base de toutes les applications Angular réelles

👉 Prochaine étape recommandée : **Formulaires avancés, performance et intégration API**