

TP5: Gestion des Formulaires Angular avec Template-Driven et ReactiveForms

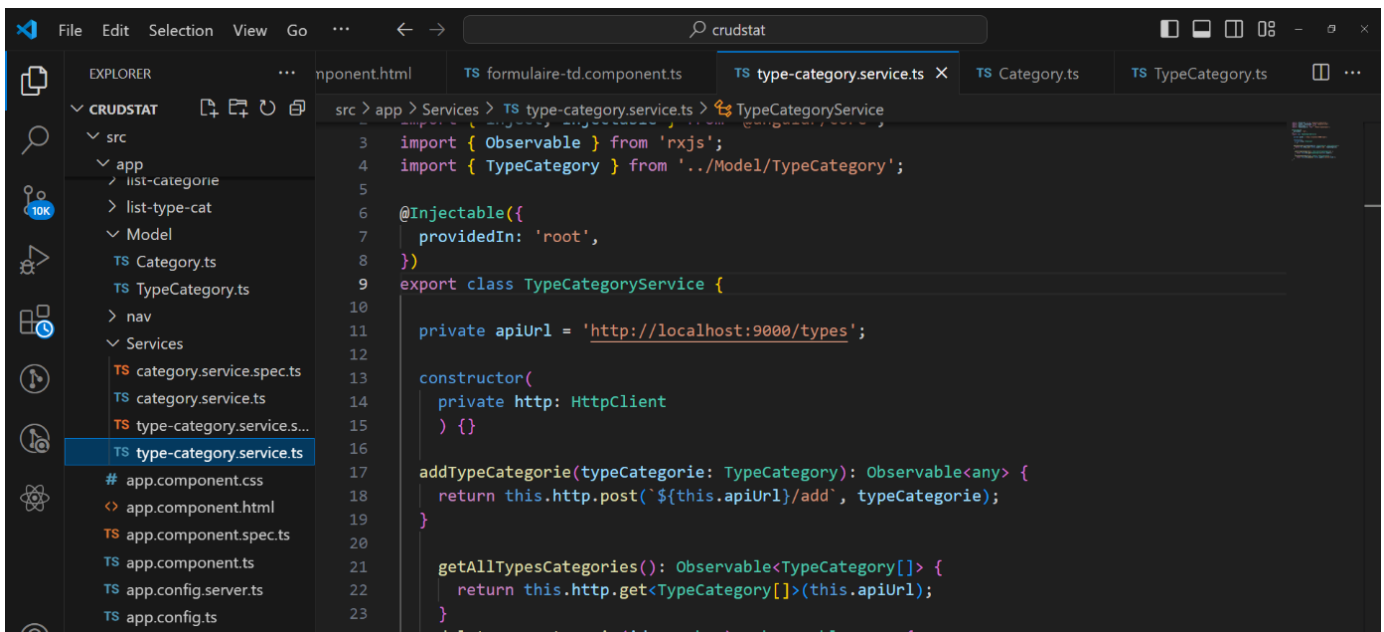
Objectif du TP :

- Apprendre à utiliser Angular Forms (Template-driven et Reactive Forms) pour collecter et envoyer des données via un formulaire.
- Implémenter un formulaire permettant d'ajouter un objet `TypeCategory` en utilisant deux techniques différentes : **Template-driven form** et **Reactive form**.
- Utiliser les services HTTP dans Angular pour envoyer les données au backend (API).

Partie 1 : Configuration du Projet Angular

Étape 1 : Créer un Service `TypeCategoryService`

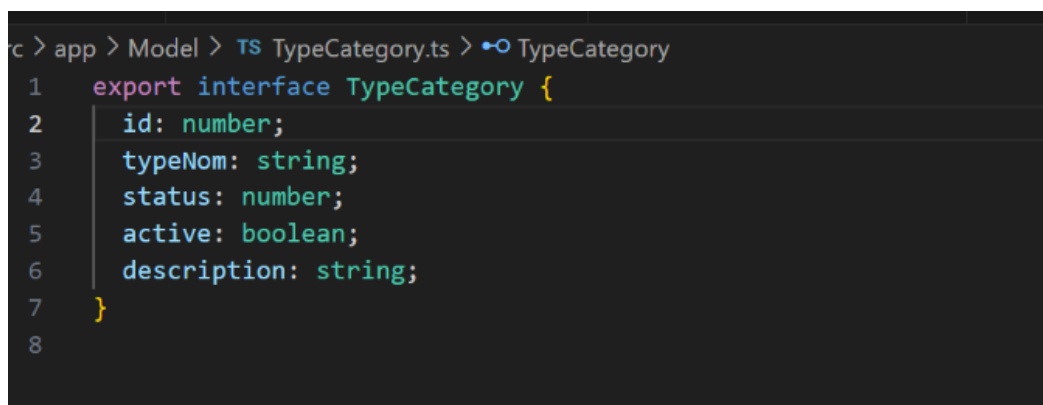
Le service `TypeCategoryService` permet d'interagir avec l'API pour envoyer et récupérer des données.



```
src > app > Services > TS type-category.service.ts > TypeCategoryService
3 import { Observable } from 'rxjs';
4 import { TypeCategory } from '../Model/TypeCategory';
5
6 @Injectable({
7   providedIn: 'root',
8 })
9 export class TypeCategoryService {
10
11   private apiUrl = 'http://localhost:9000/types';
12
13   constructor(
14     private http: HttpClient
15   ) {}
16
17   addTypeCategory(typeCategory: TypeCategory): Observable<any> {
18     return this.http.post(`${this.apiUrl}/add`, typeCategory);
19   }
20
21   getAllTypesCategories(): Observable<TypeCategory[]> {
22     return this.http.get<TypeCategory[]>(this.apiUrl);
23   }
24 }
```

Partie 2 : Création du Modèle `TypeCategory`

Le modèle `TypeCategory` représente les données envoyées par le formulaire.



```
src > app > Model > TS TypeCategory.ts > TypeCategory
1 export interface TypeCategory {
2   id: number;
3   typeNom: string;
4   status: number;
5   active: boolean;
6   description: string;
7 }
8
```

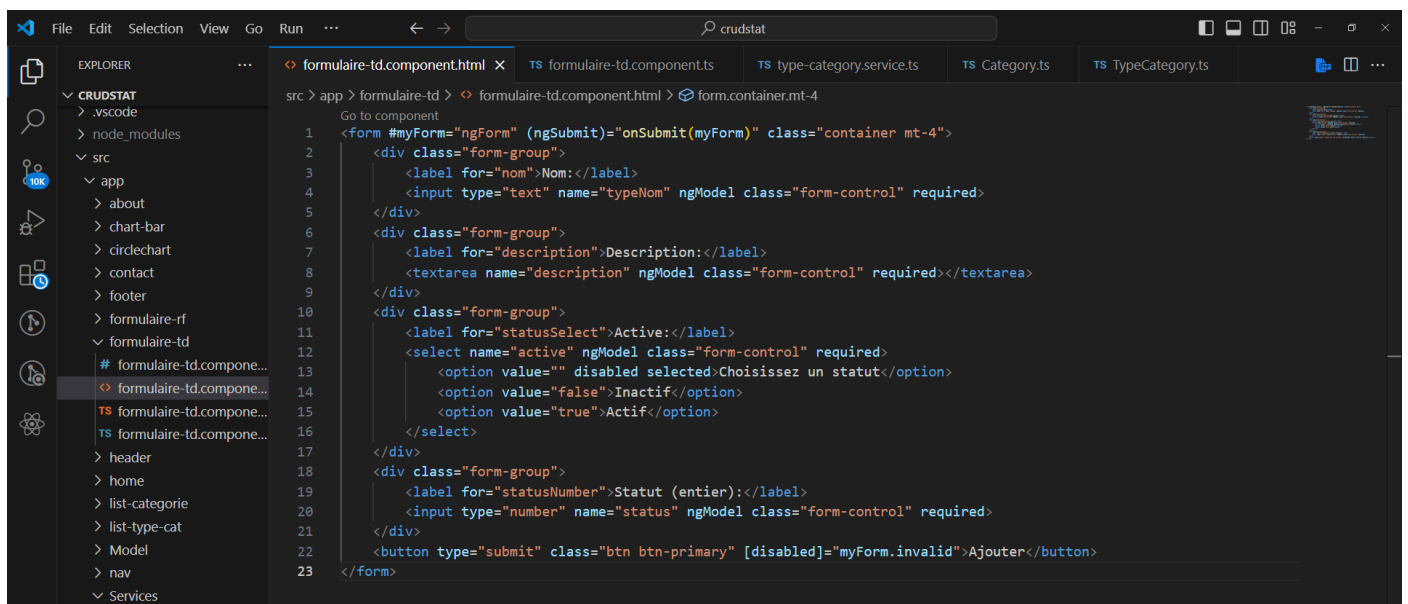
Partie 3 : Implémentation des Formulaires dans le Composant

1. Utilisation de la méthode Template-driven form

Objectif :

- Créer un formulaire utilisant `ngModel` pour collecter les données de type `TypeCategory`.

Code HTML du Template-driven Form :



```
src > app > formulaire-td > formulaire-td.component.html > form.container.mt-4
Go to component
1 <form #myForm="ngForm" (ngSubmit)="onSubmit(myForm)" class="container mt-4">
2   <div class="form-group">
3     <label for="nom">Nom:</label>
4     <input type="text" name="typeNom" ngModel class="form-control" required>
5   </div>
6   <div class="form-group">
7     <label for="description">Description:</label>
8     <textarea name="description" ngModel class="form-control" required></textarea>
9   </div>
10  <div class="form-group">
11    <label for="statusSelect">Active:</label>
12    <select name="active" ngModel class="form-control" required>
13      <option value="" disabled selected>Choisissez un statut</option>
14      <option value="false">Inactif</option>
15      <option value="true">Actif</option>
16    </select>
17  </div>
18  <div class="form-group">
19    <label for="statusNumber">Statut (entier):</label>
20    <input type="number" name="status" ngModel class="form-control" required>
21  </div>
22  <button type="submit" class="btn btn-primary" [disabled]="myForm.invalid">Ajouter</button>
23 </form>
```

Code du Composant TypeScript :

```
1 import { CommonModule } from '@angular/common';
2 import { HttpClientModule } from '@angular/common/http';
3 import { Component, inject } from '@angular/core';
4 import { FormsModule, NgForm } from '@angular/forms';
5 import { TypeCategoryService } from '../Services/type-category.service';
6 import { TypeCategory } from '../Model/TypeCategory';
7
8 @Component({
9   selector: 'app-formulaire-td',
10  standalone: true,
11  imports: [HttpClientModule, CommonModule, FormsModule],
12  templateUrl: './formulaire-td.component.html',
13  styleUrls: ['./formulaire-td.component.css']
14 })
15 export class FormulaireTDComponent {
16   typeCategoriesService = inject(TypeCategoryService); // Injection du service
17
18   // Méthode pour traiter la soumission du formulaire
19   onSubmit(form: NgForm) {
20     if (form.valid) {
21
22       // Préparez les données à envoyer
23       const formData: TypeCategory = {
24         id: form.value.id,
25         typeNom: form.value.typeNom,
26         description: form.value.description,
27         active: form.value.active === 'true', // Convertit '1' en true, '0' en false
28         status: form.value.status,
29       };
30
31       console.log(formData); // Affichez les données dans la console
32
33       // Appelez le service pour envoyer les données à l'API
34       this.typeCategoriesService.addTypeCategorie(formData).subscribe({
35         next: (response) => {
36           console.log('Données soumises avec succès', response);
37           form.reset(); // Réinitialise le formulaire après la soumission
38         },
39         error: (error) => {
40           console.error('Erreur lors de la soumission des données', error);
41         },
42       });
43     }
44   }
45 }
```

Explication :

- **ngForm** permet de lier le formulaire à la logique de soumission sans nécessiter d'un contrôle explicite des champs dans le code.
- Les données du formulaire sont envoyées à l'API via le service `TypeCategoryService`.

2. Utilisation de la méthode Reactive form

Objectif :

Créer un formulaire réactif en utilisant **Reactive Forms** et **FormGroup** pour valider et soumettre les données.

Code HTML du Reactive Form :

```
1 <form [formGroup]="myForm" (ngSubmit)="onSubmit()" class="container mt-4">
2   <div class="form-group">
3     <label for="nom">Nom:</label>
4     <input type="text" formControlName="typeNom" class="form-control" required>
5   </div>
6   <div class="form-group">
7     <label for="description">Description:</label>
8     <textarea formControlName="description" class="form-control" required></textarea>
9   </div>
10  <div class="form-group">
11    <label for="statusSelect">Active:</label>
12    <select formControlName="active" class="form-control" required>
13      <option value="" disabled selected>Choisissez un statut</option>
14      <option value="false">Inactif</option>
15      <option value="true">Actif</option>
16    </select>
17  </div>
18  <div class="form-group">
19    <label for="statusNumber">Statut (entier):</label>
20    <input type="number" formControlName="status" class="form-control" required>
21  </div>
22  <button type="submit" class="btn btn-primary" [disabled]="myForm.invalid">Ajouter</button>
23 </form>
24
```

Code du Composant TypeScript :

```
1 import { CommonModule } from '@angular/common';
2 import { HttpClientModule } from '@angular/common/http';
3 import { Component, inject } from '@angular/core';
4 import { FormBuilder, FormGroup, ReactiveFormsModule, Validators } from '@angular/forms';
5 import { TypeCategoryService } from '../Services/type-category.service';
6
7 @Component({
8   selector: 'app-formulaire-rf',
9   standalone: true,
10  imports: [HttpClientModule, ReactiveFormsModule, CommonModule],
11  templateUrl: './formulaire-rf.component.html',
12  styleUrls: ['./formulaire-rf.component.css'],
13 })
14 export class FormulaireRFComponent {
15   myForm!: FormGroup;
16
17   constructor(private fb: FormBuilder) {}
18   typeCategoryService = inject(TypeCategoryService);
19
20   ngOnInit() {
21     this.myForm = this.fb.group({
22       typeNom: ['', Validators.required],
23       description: ['', Validators.required],
24       status: ['', Validators.required],
25       active: ['', Validators.required],
26     });
27   }
28
29   onSubmit() {
30     if (this.myForm.valid) {
31       this.typeCategoryService.addTypeCategorie(this.myForm.value).subscribe({
32         next: (response) => {
33           console.log('Type de catégorie ajouté avec succès!', response);
34           this.myForm.reset(); // Réinitialiser le formulaire après l'envoi
35         },
36         error: (error) => {
37           console.error('Erreur lors de l\'ajout du type de catégorie', error);
38           alert('Erreur: ' + error.error.message);
39         }
40       });
41     }
42   }
43 }
44
```

Explication :

- **Reactive Forms** : Ici, on utilise `FormBuilder` et `FormGroup` pour créer le formulaire. On déclare les contrôles de formulaire dans le code, et Angular gère la validation de manière plus structurée.
- La logique de validation et soumission est plus explicite et gérable.

Conclusion du TP :

- **Template-driven form** est simple et rapide à mettre en place. Idéal pour les petits formulaires.
- **Reactive form** est plus puissant et flexible, idéal pour des formulaires plus complexes où on veut un contrôle complet sur la validation et l'interaction avec l'interface utilisateur