FormRecord in Angular Forms

Leave a Comment / 6 minutes of reading / April 14, 2024

⟵ Angular Tutorial Angular Forms ⟶

The `FormRecord` is a collection of FormControl. It is very similar to FormGroup and like FormGroup it tracks the value and validity state of a group of FormControl instances. Angular introduced the Typed Forms in Angular 14. With the typed forms it becomes difficult to add FormControl dynamically to a FormGroup. Hence the FormRecord is created for the sole purpose of adding a new FormControl dynamically with dynamic keys. In this article, we will learn what is FormRecord is and learn some of its important properties & methods.

Table of Contents

What is FormRecord

FormRecord is intended for situations where we want to add or remove controls dynamically with dynamic keys. FormRecord accepts one generic argument, which describes the type of the controls it contains. This will ensue type safety by restricting all controls in the FormRecord to have the same value type.

FormGroup and FormRecord are fairly similar. You can see from the definition of FormRecord that it extends the FormGroup. Hence we can use FormRecord in the same way we use FormGroup.

```

class FormRecord<TControl extends AbstractControl = AbstractControl, TControl>
  extends FormGroup<{ [key: string]: TControl;}>

```

FormRecord is a new class introduced in Angular 14 for typed reactive forms. It does not have a equivalent in Template Driven Forms.

Why it was needed

Angular introduced typed forms in Angular 14., which allowed us to assign type to our form.

To create a Typed Form, you can either initialize the form at the time of declaration or create a custom type.

The code below declares the `myForm` and initializes it at the same time. The Angular will infer its type from the initialization.

```

myForm = this.fb.group({
   key1: [''],
   key2: [''],
 });

constructor(private fb: FormBuilder) {}

```

Source Code

The `myForm` contains two `FormControl<string | null>` properties `key1` & `key2`.

Now, when we try to add a **new control at run time** with the key `test`, the Angular compiler throws an error. This is because the key `test` is not in the inferred type of `myForm`.

```

this.myForm.addControl('test', new FormControl(''));

```

Another way to create a custom type is by making use of Interfaces in TypeScript. The code below creates a new custom type `IForm`.

```

interface IForm {
  key1: FormControl<string | null>;
  key2: FormControl<string | null>;
 }

```

Use the newly created type to declare the `MyForm`.

```
 
myForm!: FormGroup<IForm>;
 
```

And then initialize the `MyForm` in ngOnInit.

```
 
  this.myForm = new FormGroup({
    key1: new FormControl<string | null>('', Validators.required),
    key2: new FormControl<string | null>('', Validators.required),
  });
 
```

Since, the key `test` is not part of the interface, the code below gives us an compile error.

```
 
this.myForm.addControl('test', new FormControl(''));
 
```

One way is to declare the **optional** key `test` in the interface `IForm`. That requires us to know the keys ahead of time, which is not possible when we add controls dynamically at run time.

We can also declare the type as `any` or `untypedFormGroup`.

```
 
myForm!: FormGroup<any>;
 
or
 
myForm!: UntypedFormGroup
 
```

In both the above cases, you are not using the types, which would defeat the very purpose of using the Typed forms.

This is where FormRecord steps in. It allows us to add new controls at runtime with dynamic keys and also keeps the benefit of the type system.

How to use FormRecord

The example uses the Angular 17 and standalone components

The code below creates an Angular Form of type `FormControl<string|null>`. We also suffix the mainForm with `!` other wise it will result in an `Property 'mainForm' has no initializer and is not definitely assigned in the constructor` error.

```

mainForm!: FormRecord<FormControl<string | null>>;

```

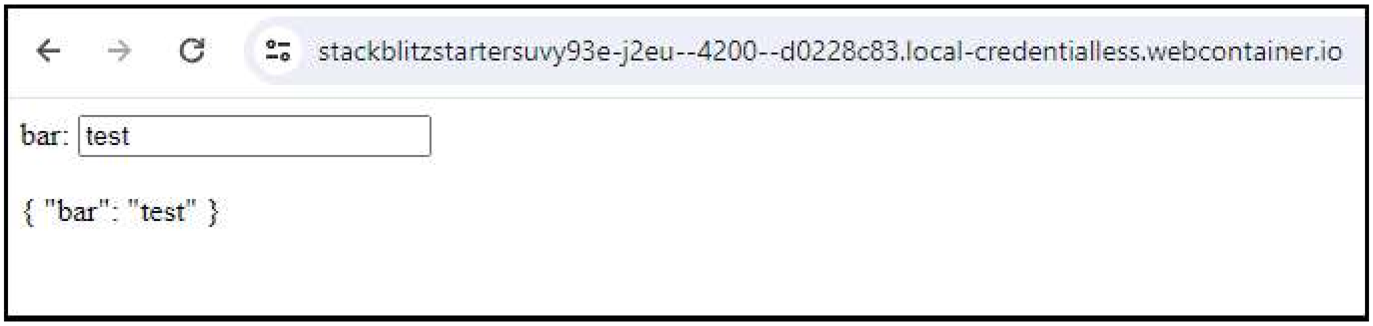
To add a new FormControl dynamically, we invoke the `addControl` method passing a new instance of `FromControl` with a key `bar`.

```

  ngOnInit() {
```

```
  this.mainForm.addControl('bar', new FormControl(''));
}

```

In the template assign the mainForm to formGroup directive to display the Form. We use the mainForm.value to read the values of the form just like we do it with the FormGroup.

```

<form [formGroup]="mainForm">
   bar: <input formControlName="bar" /><br />
 </form>
 <br />
 <div>{{ mainForm.value | json }}</div>

```


stackblitzstartersuvy93e-j2eu--4200--d0228c83.local-credentialless.webcontainer.io

bar: test

```
{ "bar": "test" }
```

Complete Source code is on stackblitz.com

We can add another control using the addControl method.

```

this.mainForm.addControl('foo', new FormControl(''));

```

Controls can be removed using the removeControl method and using the key

```

this.mainForm.removeControl('foo');

```

Our FormRecord is of type `FormControl<string|null>`. Hence adding FormControl with the value 0 will result in compile error

```

this.mainForm.addControl('raz', new FormControl(0));

   //Argument of type 'FormControl<number | null>' is not assignable to parameter of type
   //Type 'number | null' is not assignable to type 'string | null'.
   //Type 'number' is not assignable to type 'string'.

```

Dynamically adding Controls to FormRecord

In this example, we will show you how to add controls dynamically to a `FormRecord`.

Create a `FormRecord` and an array to hold designations.

```

  public mainForm: FormRecord = new FormRecord<FormControl<string | null>>({});
  public designations: string[] = ['CEO', 'Manager', 'Supervisor'];

```

In ngOnInit hook loop through the designations array and add the FormControl instance to the FormRecord using the addControl method.

```

ngOnInit() {
  this.designations.forEach((key) =>
    this.mainForm.addControl(key, new FormControl(''))
  );
}

```

Create a form to capture the new designation to add to our form.

```

    <input type="text" name="newDesignation" [(ngModel)]="newDesignation">
    <button (click)="addDesignation()">Add</button> <br/>

```

In the `addDesignation` method, push the new designation to the `designations` array and also add new control to the `mainForm`.

```

addDesignation() {
  this.designations.push(this.newDesignation)
  this.mainForm.addControl(this.newDesignation, new FormControl(''))
  this.newDesignation=""
}

```

Finally, in the template use the ngFor directive to loop through the designations and insert the input field for each control

```

    <form [formGroup]="mainForm">
      <div *ngFor="let key of designations">
        <b>{{ key }}: </b>
        <input [formControlName]="key" /><br />
      </div>
    </form>

```

You can download the Source Code.

We can further extend the above example to include option to remove the designation.

In the example below, removeDesignation method uses the removeControl method to remove the FormControl from the mainForm.

```

removeDesignation(key:string) {
  this.mainForm.removeControl(key)
}

```

We can also simplify the code and remove the need for maintaining the designation array, by directly reading the keys from the FormRecord

```

get designations() {
  return Object.keys(this.mainForm.controls)
}

```

Which will also simplifies the addDesignation method.

```

addDesignation() {
  if  (this.newDesignation !="") {
    this.mainForm.addControl(this.newDesignation, new FormControl(''))
    this.newDesignation=""
  }
}

```

Finally, add the option to remove the designation in the template by invoking the removeDesignation method using event binding.

```

    <form [formGroup]="mainForm">
      <div *ngFor="let key of designations">
        <b>{{ key }}: </b>
        <input [formControlName]="key" />
        <button (click)="removeDesignation(key)">X</button>
        <br />
      </div>
    </form>

```

Source Code

FormRecord Vs FormArray

Both FormRecord and FormArray allows us to add or remove FormControl ‘s at runtime. Both tracks the value & validity of their controls and help us the manage them.

The difference is how they are structured. In FormRecord controls becomes a property of the FormRecord . Each control is represented as key-value pair, while in FormArray , the controls become part of an array.

Hence we can manage the controls in the FormRecord using the keys, While in FormArray we need to track them using the array index or using some other technique.

This makes FormRecord ideal for situations where you need to add the new FormControl whose keys are known only at runtime.

FormRecord Vs FormGroup

FormRecord extends FormGroup and hence is very similar. Both Manage the FormControl instance as a key-value pair.

We cannot add FormControl to FormGroup at runtime as we need to declare the type of FormGroup upfront. FormRecord on the other hand is designed to add new FormControl as a key-value pair without breaking the type checking.