Reactive Forms

1. ✅ What Are Reactive Forms?

> **Reactive Forms** are a way of building forms in Angular using **code (TypeScript)** instead of writing logic directly in the template (HTML).
> They are more **scalable, testable, and powerful** than Template-Driven Forms.

Real-life comparison:

Feature	Template-Driven	Reactive
Code Location	Mostly in template	Mostly in TS file
Flexibility	Basic forms	Complex forms, dynamic
Validation	HTML-based	Programmatic
Debugging	Hard	Easy

2. 🧠 Core Concepts

- **FormControl** – Represents a single input element.
- **FormGroup** – A group of `FormControl`s (i.e., a whole form).
- **FormBuilder** – A helper service to create `FormControl`s and `FormGroup`s with less code.

Analogy:

> - "`FormControl` = a single field (like a name input)"
> - "`FormGroup` = the full form (like a student registration form)"
> - "`FormBuilder` = a factory to build controls and groups easily"

3. 🛠️ Setup: Creating a Simple Reactive Form

🔹 Step 1: Import `ReactiveFormsModule`

In your `app.module.ts`:

ts

Copy Edit

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

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

Step 2: Create a Form in the Component

ts

Copy Edit

```ts
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-student-form',
  templateUrl: './student-form.component.html',
})
export class StudentFormComponent {
  studentForm = new FormGroup({
    name: new FormControl(''),
    age: new FormControl(''),
  });

  onSubmit() {
    console.log(this.studentForm.value);
  }
}
```

Step 3: Form Template

html

Cop

```
<form [formGroup]="studentForm" (ngSubmit)="onSubmit()">
  <label>
    Name:
    <input type="text" formControlName="name">
  </label><br>

  <label>
    Age:
    <input type="number" formControlName="age">
  </label><br>

  <button type="submit">Submit</button>
</form>
```

4. 🦸 How to Use Validation

Update the component:

ts

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

studentForm = new FormGroup({
  name: new FormControl('', [Validators.required, Validators.minLength(3)]),
  age: new FormControl('', [Validators.required, Validators.min(18)]),
});
```

In the template:

html

```html
<div *ngIf="studentForm.get('name')?.invalid && studentForm.get('name')?.touched">
  Name is required and must be at least 3 characters.
</div>
```

```html
<!-- student-form.component.html -->
<form [formGroup]="studentForm" (ngSubmit)="onSubmit()">
  <input formControlName="name" placeholder="Name">
  <div *ngIf="studentForm.get('name')?.invalid && studentForm.get('name')?.touched">
    Name is required and should be 3+ chars.
  </div>

  <input formControlName="email" placeholder="Email">
  <div *ngIf="studentForm.get('email')?.invalid && studentForm.get('email')?.touched">
    Valid email required.
  </div>

  <button type="submit" [disabled]="studentForm.invalid">Submit</button>
</form>
```

✅ Summary

Concept	Example
`FormControl`	Single input field
`FormGroup`	Whole form (group of fields)
`formControlName`	Binds input to `FormControl`
`ReactiveFormsModule`	Required to use reactive forms

Level 2

🔄 1. Two-way Data Binding (Reactive Forms Version)

Reactive forms **do not use** `[(ngModel)]`, but we can achieve something similar via the `FormControl`.

Example:

ts

Copy Edit

```ts
this.studentForm.get('name')?.value  // to read value
this.studentForm.get('name')?.setValue('Ahmed')  // to set value
```

You can bind this logic to a button or lifecycle hook to simulate dynamic value changes.

📦 2. `FormBuilder` (Cleaner Syntax)

Without `FormBuilder`:

ts

```ts
new FormGroup({
  name: new FormControl(''),
  age: new FormControl(''),
});
```

With `FormBuilder`:

ts

```ts
constructor(private fb: FormBuilder) {}

this.studentForm = this.fb.group({
  name: [''],
  age: ['']
});
```

✅ **Advantage**: Less typing, easier to read, easier to apply validators.

✅ 3. Custom Validation (Like “Password Match”)

Let’s say you want to check if `password` and `confirmPassword` match.

🔧 Step 1: Create a Custom Validator Function

ts

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

export const confirmPasswordValidator: ValidatorFn = (control: AbstractControl) => {
  const password = control.get('password')?.value;
  const confirm = control.get('confirmPassword')?.value;
  return password === confirm ? null : { passwordMismatch: true };
};
```

🔧 Step 2: Apply It to the FormGroup

ts

Copy Edit

```ts
this.form = this.fb.group({
  password: [''],
  confirmPassword: [''],
}, { validators: confirmPasswordValidator });
```

🔧 Step 3: Show Error in HTML

html

Copy Edit

```html
<div *ngIf="form.errors?.passwordMismatch && form.touched">
  Passwords do not match!
</div>
```

🔁 4. `valueChanges` & `statusChanges`

Let's your form update something live as the user types.

ts

Copy Edit

```ts
this.studentForm.get('name')?.valueChanges.subscribe(value => {
  console.log('Name changed to:', value);
});

this.studentForm.statusChanges.subscribe(status => {
  console.log('Form status:', status);
});
```

✅ Use this to show live validation, change button colors, or auto-fill suggestions.

🔧 5. Dynamic Form Controls (Add/Remove Fields at Runtime)

Perfect for fields like: "Add more emails", "Add children", etc.

🔹 Step 1: Use `FormArray`

ts

Copy Edit

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

form = this.fb.group({
  emails: this.emails
});
```

🧠 Real-Life Analogy

Think of a form like a **Google Form**:

- **“You design it using code (`FormBuilder`)”**
- **“Every question is a `FormControl`”**
- **“Sections are `FormGroup`”**
- **“"Add more" buttons use `FormArray`”**
- **“You validate if the form is complete using `status` and `validators`”**

Is it Easy or too Easy?