Lecture 14: Angular Forms

What are Angular forms?

- **Angular Forms** enable the capture, validation, and processing of user inputs in web applications.

- Two main types:
  - **Template-Driven Forms –** Easier for simple use cases, heavily reliant on templates.
  - **Reactive Forms –** More powerful, programmatic, and suitable for complex use cases

Template-Driven Forms

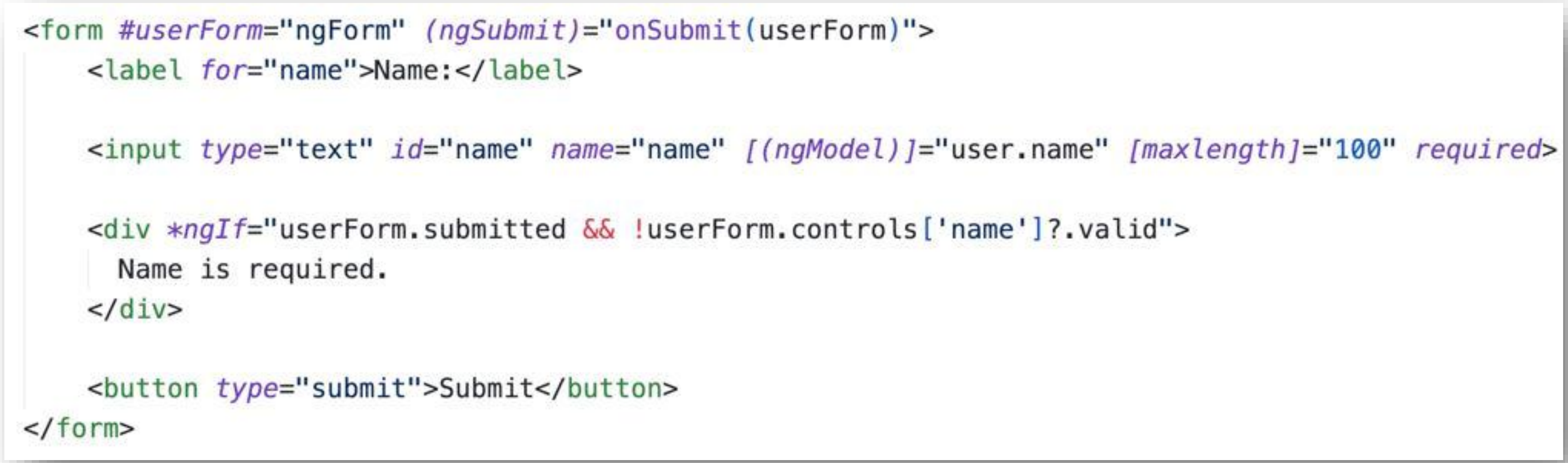
Characteristics

- Uses Angular directives like **ngModel** to bind data to the template
- Relies on two-way data binding and **FormsModule**
- Best for small scale, straightforward forms

Key Features

- Utilizes **[(ngModel)]** to bind input values to the component class
- Supports built-in validators (e.g. **required, minlength, maxlength, pattern**)
- **ngForm** – Tracks the overall form state
- **ngModel** – Tracks individual form controls

Template-Driven Forms



```html
<form #userForm="ngForm" (ngSubmit)="onSubmit(userForm)">
    <label for="name">Name:</label>

    <input type="text" id="name" name="name" [(ngModel)]="user.name" [maxlength]="100" required>

    <div *ngIf="userForm.submitted && !userForm.controls['name']?.valid">
      Name is required.
    </div>

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

Reactive Forms

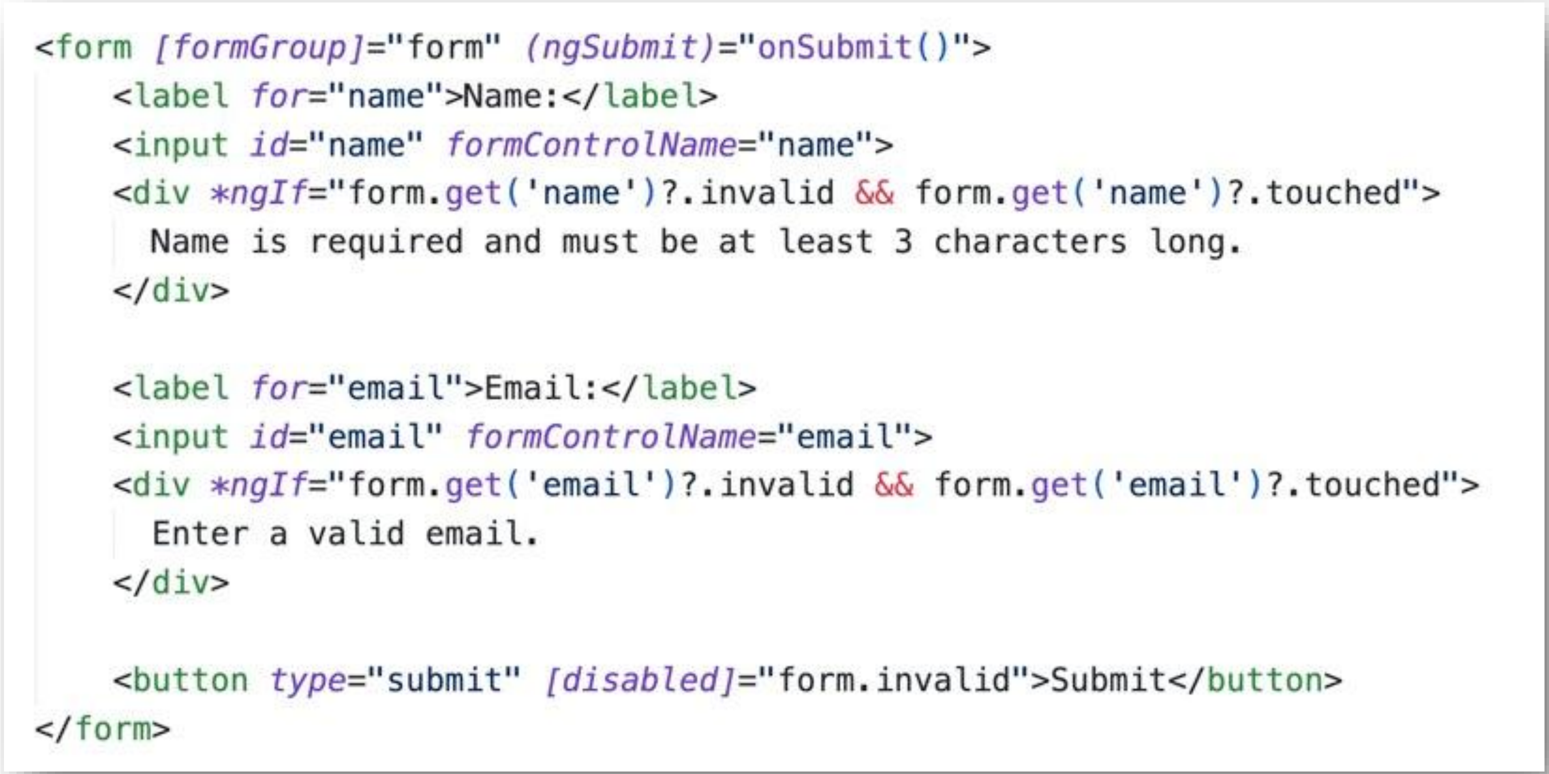
Characteristics

- More structured and scalable, using explicit and immutable form-state management
- Relies on **FormGroup, FormControl,** and **FormBuilder.**
- Requires importing **ReactiveFormsModule**

Key Features

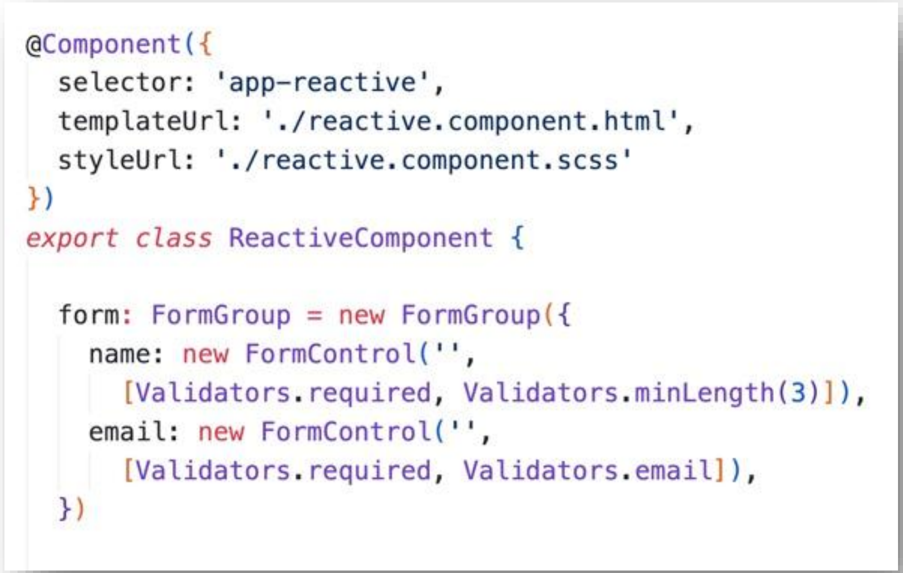
- **FormGroup** - a collection of **FormControl** objects that track the value and the state of a group of controls
- **FormControl** represents a single input field and its validation logic
- **FormBuilder –** simplifies the creation of form groups and controls

Reactive Form Template



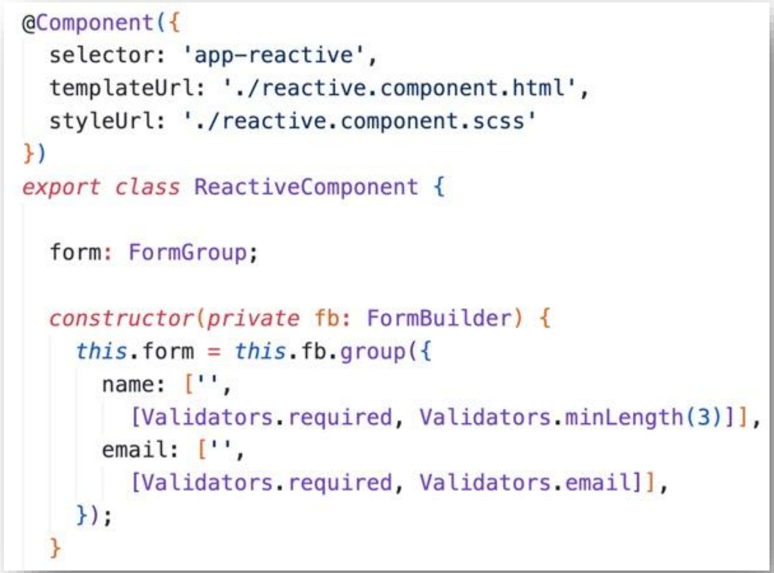
```
<form [formGroup]="form" (ngSubmit)="onSubmit()">
    <label for="name">Name:</label>
    <input id="name" formControlName="name">
    <div *ngIf="form.get('name')?.invalid && form.get('name')?.touched">
      Name is required and must be at least 3 characters long.
    </div>

    <label for="email">Email:</label>
    <input id="email" formControlName="email">
    <div *ngIf="form.get('email')?.invalid && form.get('email')?.touched">
      Enter a valid email.
    </div>

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

FormGroup Version



```
@Component({
  selector: 'app-reactive',
  templateUrl: './reactive.component.html',
  styleUrl: './reactive.component.scss'
})
export class ReactiveComponent {

  form: FormGroup = new FormGroup({
    name: new FormControl('',
      [Validators.required, Validators.minLength(3)]),
    email: new FormControl('',
      [Validators.required, Validators.email]),
  })
```

FormBuilder Version



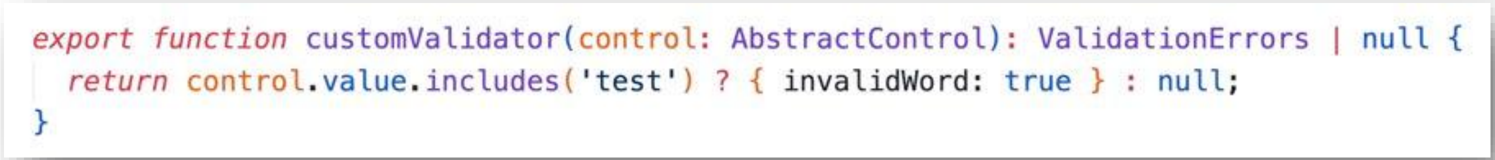
```
@Component({
  selector: 'app-reactive',
  templateUrl: './reactive.component.html',
  styleUrl: './reactive.component.scss'
})
export class ReactiveComponent {

  form: FormGroup;

  constructor(private fb: FormBuilder) {
    this.form = this.fb.group({
      name: ['',
        [Validators.required, Validators.minLength(3)]],
      email: ['',
        [Validators.required, Validators.email]],
    });
  }
```

Validation

Built-in Validators

- required
- minlength, maxlength
- email
- pattern

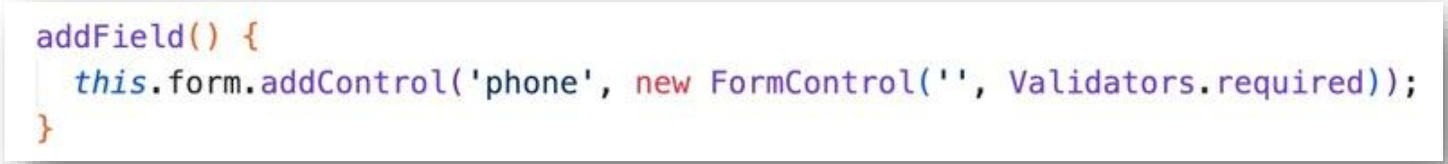
Custom Validation



```
export function customValidator(control: AbstractControl): ValidationErrors | null {
  return control.value.includes('test') ? { invalidWord: true } : null;
}
```

Dynamic Forms

- Add / remove form controls or groups dynamically



```
addField() {
  this.form.addControl('phone', new FormControl('', Validators.required));
}
```

Form States

- **Valid –** All controls are valid
- **Invalid –** At least one control is invalid
- **Touched –** Control has been focused and unfocused
- **Dirty –** Control value has been changed