Mobile Application Development

LECTURE 6

Forms in Angular

- A large category of frontend applications are very form-intensive, especially in the case of enterprise development. Many of these applications are basically just huge forms, spanning multiple tabs and dialogs and with non-trivial validation business logic.

- There are two types of forms in Angular. Template Driven Forms and Reactive / Model Driven Forms.

- Reactive Forms are arguably the best thing about Angular (biased opinion ☺)

Template Driven vs Reactive Forms

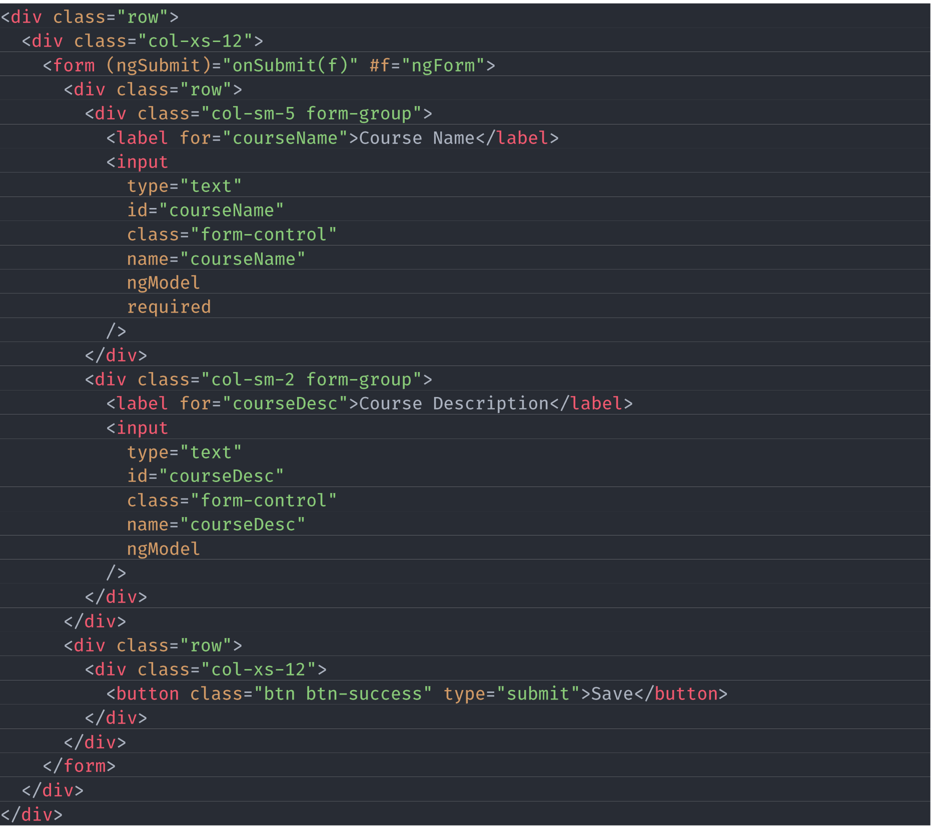
- In a template-driven approach, most of the logic is driven from the template, whereas in reactive-driven approach, the logic resides mainly in the component or typescript code.
- Template Driven forms involve writing less code however it can get complicated very quickly.
- Template Driven forms get complicated very quickly and are very difficult to handle when the forms get large.
- Reactive forms or Model-driven forms are more robust, scalable, reusable, and testable.

Template Driven vs Reactive Forms

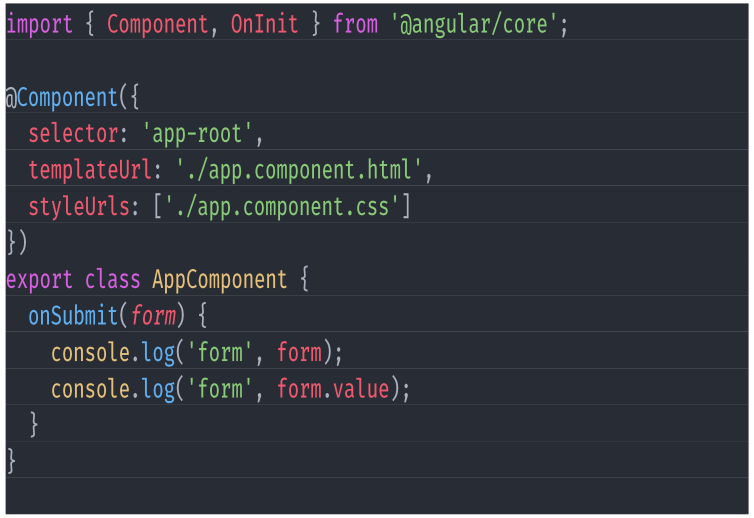
- Validations especially custom validations in Template Driven forms are very hard whereas they are a breeze to work with in Reactive Forms.
- Working on large Forms is very easier in Reactive Forms compared to Template Driven.
- Even React.js doesnot provide such easy form manipulation as Angular. With React.js, you would need to include a library like **Formik** however Angular offers such forms handling out of the box through Reactive Forms.

“You’ll stop worrying what others think about you when you realize how seldom they do.” ~ David Foster Wallace

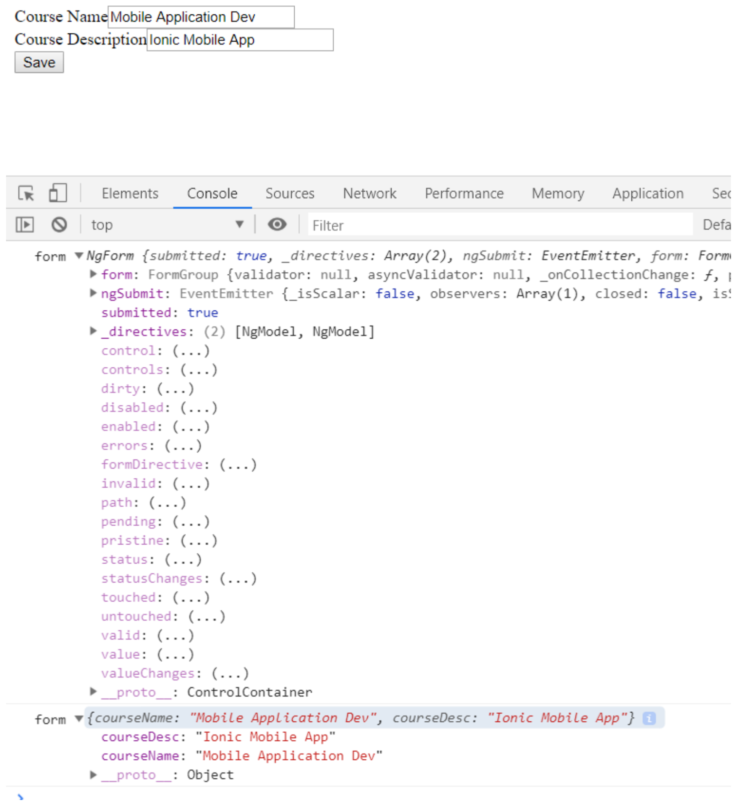
Template Driven Forms



```html
<div class="row">
  <div class="col-xs-12">
    <form (ngSubmit)="onSubmit(f)" #f="ngForm">
      <div class="row">
        <div class="col-sm-5 form-group">
          <label for="courseName">Course Name</label>
          <input
            type="text"
            id="courseName"
            class="form-control"
            name="courseName"
            ngModel
            required
          />
        </div>
        <div class="col-sm-2 form-group">
          <label for="courseDesc">Course Description</label>
          <input
            type="text"
            id="courseDesc"
            class="form-control"
            name="courseDesc"
            ngModel
          />
        </div>
      </div>
      <div class="row">
        <div class="col-xs-12">
          <button class="btn btn-success" type="submit">Save</button>
        </div>
      </div>
    </form>
  </div>
</div>
```



```typescript
import { Component, OnInit } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent {
  onSubmit(form) {
    console.log('form', form);
    console.log('form', form.value);
  }
}
```

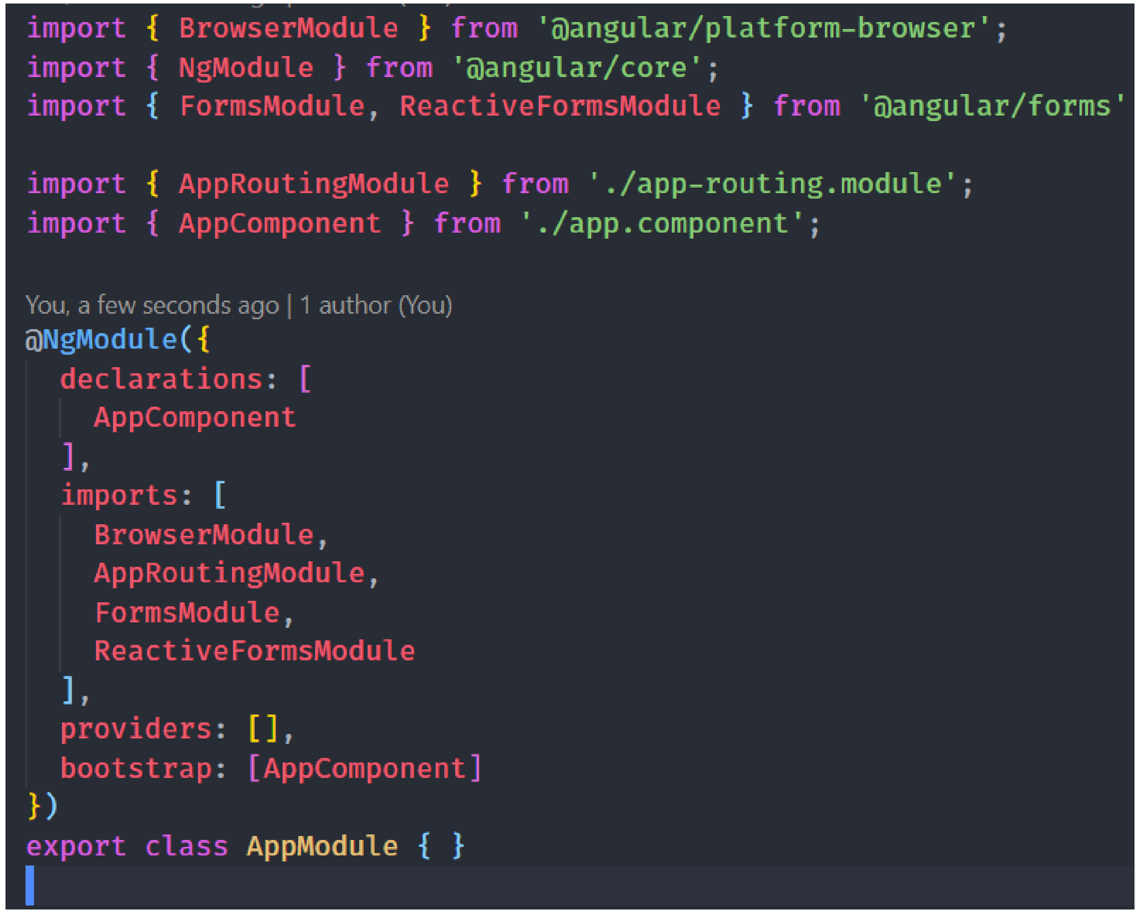


Course Name Mobile Application Dev
Course Description Ionic Mobile App

```
form ▼NgForm {submitted: true, _directives: Array(2), ngSubmit: EventEmitter, form: Form
      ▶form: FormGroup {validator: null, asyncValidator: null, _onCollectionChange: ƒ,
      ▶ngSubmit: EventEmitter {_isScalar: false, observers: Array(1), closed: false, is
       submitted: true
      ▶_directives: (2) [NgModel, NgModel]
       control: (...)
       controls: (...)
       dirty: (...)
       disabled: (...)
       enabled: (...)
       errors: (...)
       formDirective: (...)
       invalid: (...)
       path: (...)
       pending: (...)
       pristine: (...)
       status: (...)
       statusChanges: (...)
       touched: (...)
       untouched: (...)
       valid: (...)
       value: (...)
       valueChanges: (...)
      ▶__proto__: ControlContainer
form ▼{courseName: "Mobile Application Dev", courseDesc: "Ionic Mobile App"}
       courseDesc: "Ionic Mobile App"
       courseName: "Mobile Application Dev"
      ▶__proto__: Object
```

Reactive Forms

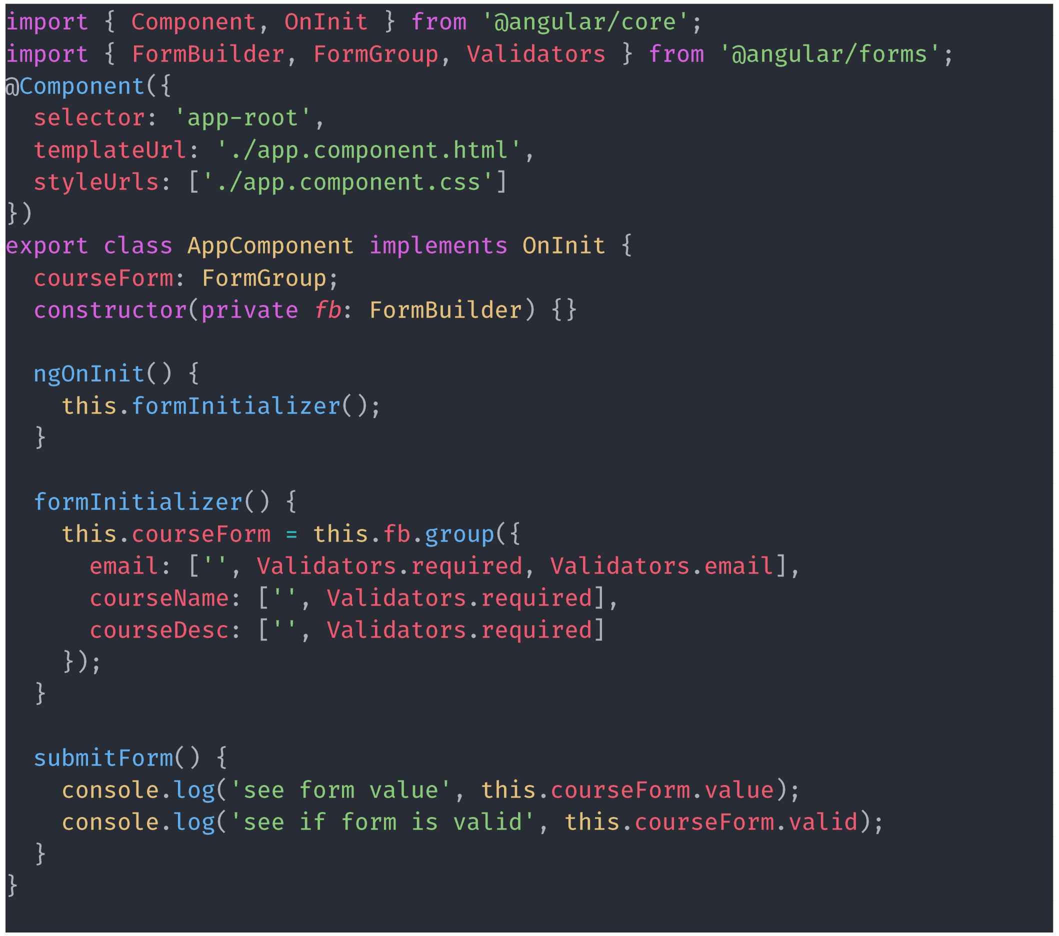
- In the model-driven form, we need to import a ReactiveFormsModule from @angular/forms and use the same in the imports array.



```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { FormsModule, ReactiveFormsModule } from '@angular/forms'

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';

You, a few seconds ago | 1 author (You)
@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    AppRoutingModule,
    FormsModule,
    ReactiveFormsModule
  ],
  providers: [],
  bootstrap: [AppComponent]
})
export class AppModule { }
```

Reactive Forms

- In the **app.component.ts** file, we need to import the few modules for the model-driven form. For example, import { FormGroup, FormBuilder, Validators } from ‘@angular/forms’.



```
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  courseForm: FormGroup;
  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.formInitializer();
  }

  formInitializer() {
    this.courseForm = this.fb.group({
      email: ['', Validators.required, Validators.email],
      courseName: ['', Validators.required],
      courseDesc: ['', Validators.required]
    });
  }

  submitForm() {
    console.log('see form value', this.courseForm.value);
    console.log('see if form is valid', this.courseForm.valid);
  }
}
```

Reactive Forms

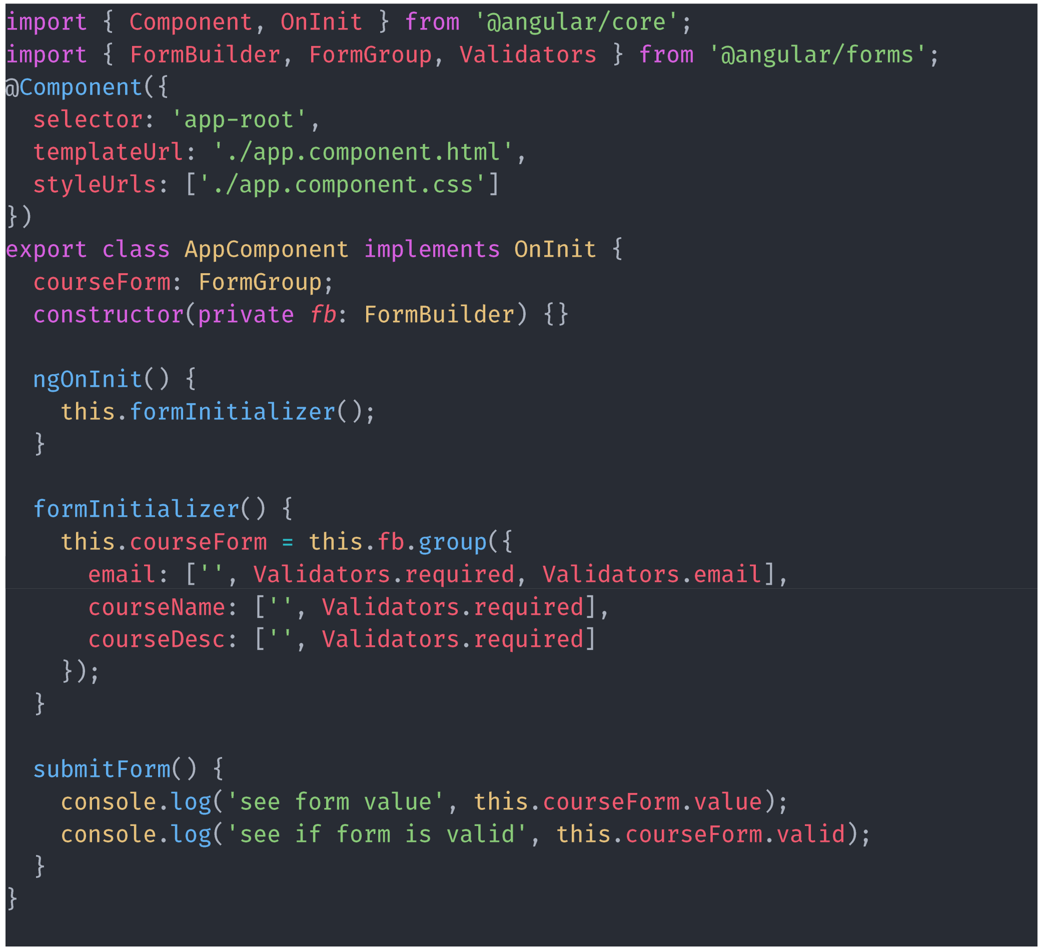
- We inject formbuilder instance in

  constructor.

- we initialize the form on ngOnInit.

- all the form logic resides inside .TS /

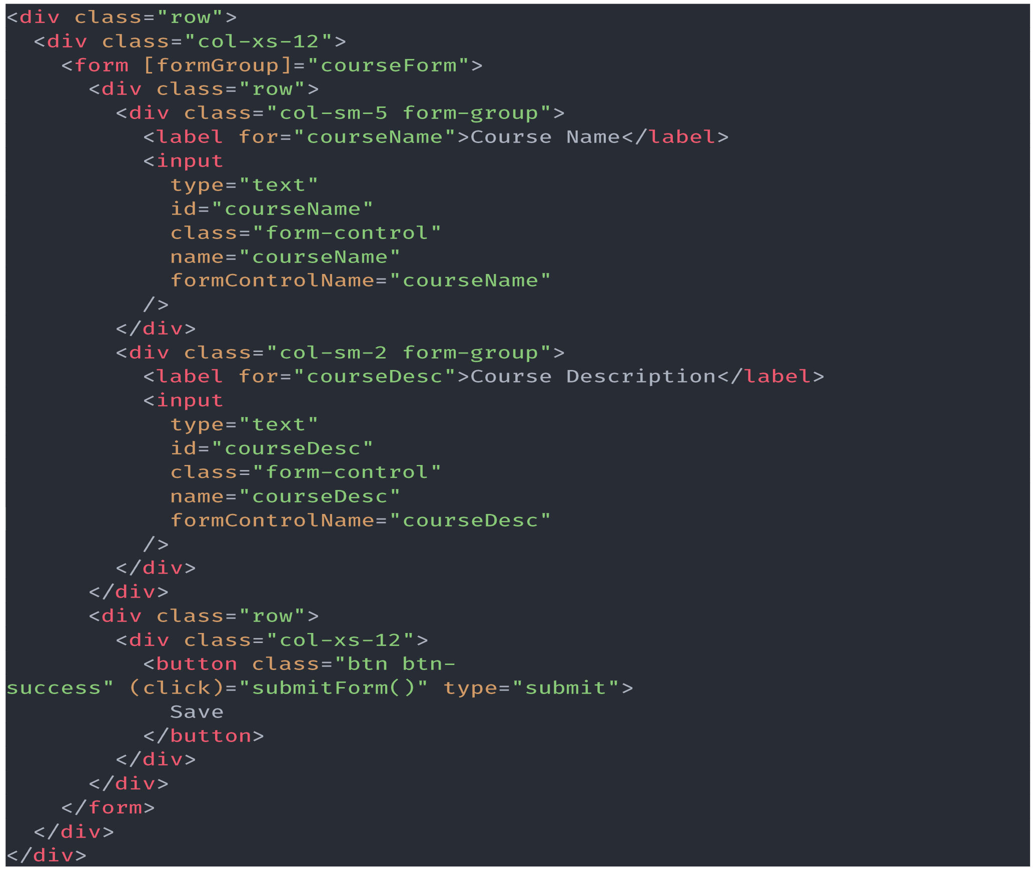
  component file



```typescript
import { Component, OnInit } from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.css']
})
export class AppComponent implements OnInit {
  courseForm: FormGroup;
  constructor(private fb: FormBuilder) {}

  ngOnInit() {
    this.formInitializer();
  }

  formInitializer() {
    this.courseForm = this.fb.group({
      email: ['', Validators.required, Validators.email],
      courseName: ['', Validators.required],
      courseDesc: ['', Validators.required]
    });
  }

  submitForm() {
    console.log('see form value', this.courseForm.value);
    console.log('see if form is valid', this.courseForm.valid);
  }
}
```

Reactive Forms

- HTML Code. Here we add [formGroup] and only formControlName to each fields. That’s it.

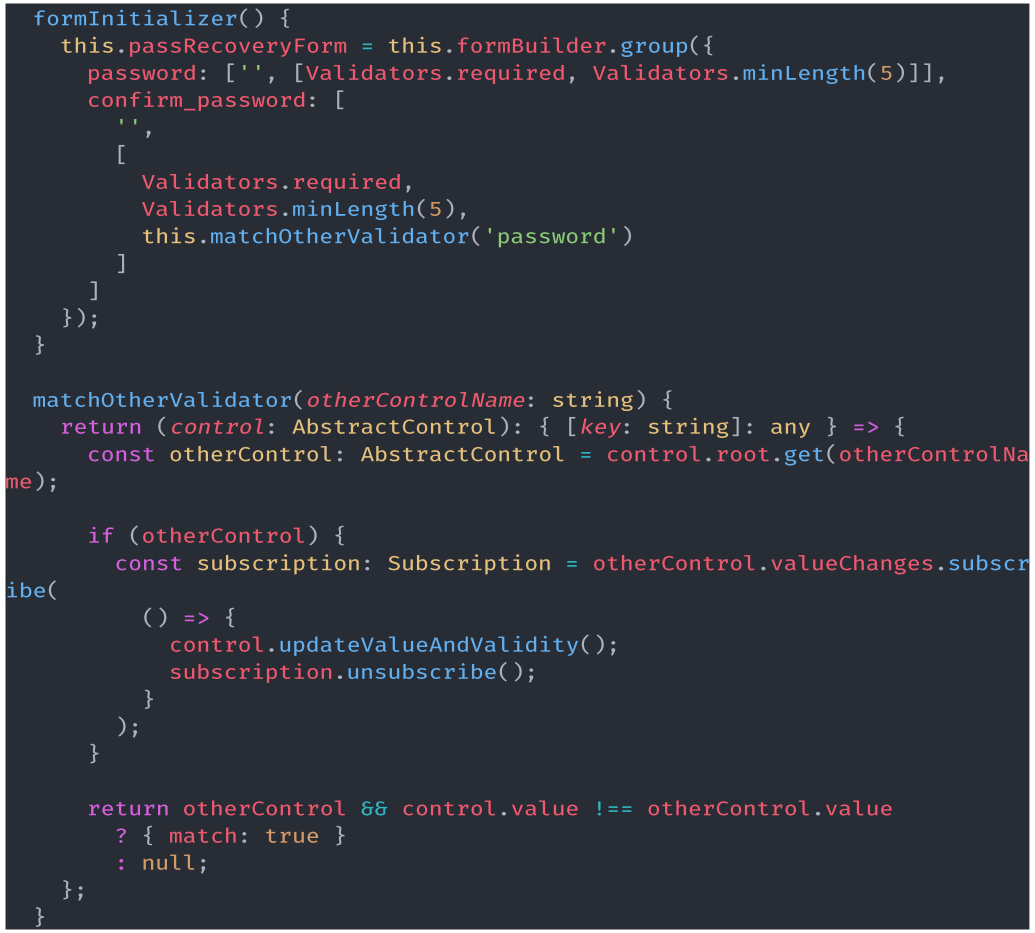


```
<div class="row">
  <div class="col-xs-12">
    <form [formGroup]="courseForm">
      <div class="row">
        <div class="col-sm-5 form-group">
          <label for="courseName">Course Name</label>
          <input
            type="text"
            id="courseName"
            class="form-control"
            name="courseName"
            formControlName="courseName"
          />
        </div>
        <div class="col-sm-2 form-group">
          <label for="courseDesc">Course Description</label>
          <input
            type="text"
            id="courseDesc"
            class="form-control"
            name="courseDesc"
            formControlName="courseDesc"
          />
        </div>
      </div>
      <div class="row">
        <div class="col-xs-12">
          <button class="btn btn-
success" (click)="submitForm()" type="submit">
            Save
          </button>
        </div>
      </div>
    </form>
  </div>
</div>
```

Reactive Forms

- Example of custom validations made easy

using Reactive Forms



```
formInitializer() {
  this.passRecoveryForm = this.formBuilder.group({
    password: ['', [Validators.required, Validators.minLength(5)]],
    confirm_password: [
      '',
      [
        Validators.required,
        Validators.minLength(5),
        this.matchOtherValidator('password')
      ]
    ]
  });
}

matchOtherValidator(otherControlName: string) {
  return (control: AbstractControl): { [key: string]: any } => {
    const otherControl: AbstractControl = control.root.get(otherControlNa
me);

    if (otherControl) {
      const subscription: Subscription = otherControl.valueChanges.subscr
ibe(
        () => {
          control.updateValueAndValidity();
          subscription.unsubscribe();
        }
      );
    }

    return otherControl && control.value !== otherControl.value
      ? { match: true }
      : null;
  };
}
```