Model Driven Forms or Reactive Forms

**Introduction to Forms in Angular:**

Forms are crucial part of web applications. Forms enable users to provide data input into the application in contexts like performing user registration, user sign-in, updating profile, entering sensitive information like payment information and for performing other data-entry based tasks.

**Forms in Angular:**

Angular has two different approaches in dealing with forms: *reactive forms* and *template-driven forms*.

Both reactive forms and template-driven forms:

* can capture user-provided data,
* can capture user input events,
* can validate the user input, etc.
* have their own approaches of processing and managing the form data:

In template-driven forms, you will create the form completely in the template and need to rely on directives to create and manipulate the underlying form object model. Since the template-driven forms do not scale that well, they are more suitable only when you want to add a simple small form to the application. For example: a signup form.

In reactive forms, you can control the form completely from the component class and hence you will get direct, explicit access to the underlying forms object model. Hence, reactive forms are also known as *'model-driven forms'*. As reactive forms are more robust and scalable, they are more suitable for creating all kind of forms in an application, irrespective of the size of form.

Angular automatically tracks the changes happening to the form and form controls as and when user provides input and thereby controls the state and the validity of the form/form controls. Angular does this by associating respective keywords automatically for the forms/form controls depending on the context. The below table describes the details:

|  |  |  |
| --- | --- | --- |
| **State detected for the form control** | **Context** | **Keyword associated by Angular** |
| valid | Element value is valid | valid.  The keyword becomes true if element is valid. |
| invalid | Element value is invalid | invalid.  The keyword becomes true if element is invalid |
| dirty | Element value is changed | dirty.  The keyword becomes true if element is dirty |
| pristine | Element value is unchanged | pristine.  The keyword becomes true if element is pristine |
| touched | Element gets focus | touched.  The keyword becomes true if element is touched |
| untouched | Element doesn't have a focus in it | untouched.  The keyword becomes true if element is untouched |

Angular also has the following built-in CSS classes which get auto-applied depending on the state. Code can be written inside these CSS classes which can change the appearance of the control suitably as per context.

|  |  |
| --- | --- |
| **CSS Class** | **Purpose** |
| ng-valid | Applied if control's value is valid |
| ng-invalid | Applied if control's value is invalid |
| ng-dirty | Applied if control's value is changed |
| ng-pristine | Applied if control's value is not changed |
| ng-touched | Applied if control is touched/gets focus |
| ng-untouched | Applied if control is not touched/doesn't get focus |

**Following are the advantages of Reactive Forms/Model Driven Forms:**

* Unit testing(using the Jasmine framework) on the validation logic can be performed, as it is written inside the component class.
* Form changes or events can be heard easily using reactive forms. Each FormGroup or FormControl has few events like valueChanges, statusChanges, etc.,  which can be subscribed to.
* Reactive forms are used in creating medium to large scale applications

Due to the advantages of Reactive forms, in most Angular applications, Reactive Forms Approach is chosen when creating forms

Step1:

To create a reactive form in Angular, **FormBuilder** class must be used. To make the FormBuilder class available, **ReactiveFormsModule**has to be imported in the root module.

Register the ReactiveFormsModule during bootstrapping, in the **app.module.ts**

1. import { BrowserModule } from '@angular/platform-browser';
2. import { NgModule } from '@angular/core';
3. import { ReactiveFormsModule } from '@angular/forms';
4. import { AppComponent } from './app.component';
5. import { RegistrationFormComponent } from './registration-form/registration-form.component';
6. @NgModule({
7. declarations: [
8. AppComponent,
9. RegistrationFormComponent
10. ],
11. imports: [
12. BrowserModule,
13. ReactiveFormsModule
14. ],
15. providers: [],
16. bootstrap: [AppComponent]
17. })
18. export class AppModule { }

Line 3: Import ReactiveFormsModule from @angular/forms module

Line 15: Add ReactiveFormsModule in the imports declaration to create reactive forms

Step 2:

Create a component called **RegistrationForm** using the following CLI command

1. ng generate component RegistrationForm

Step 3:

Add the below to **app.component.html**

1. <app-registration-form></app-registration-form>

Line 1: Loads RegistrationFormComponent in the root component

Step 4:

Bootstrap CSS framework (v3) is commonly used for adding basic structural layout and style for web applications. For structuring the RegistrationForm in this example, CSS classes from the Bootstrap CSS framework can be used.

To add Bootstrap CSS library to the application, install **bootstrap** as shown below:

1. npm install bootstrap@3.3.7 --save

Then, include boostrap.min.css file in angular.json file as shown below:

1. ...
2. "styles": [
3. "src/styles.css",
4. "node\_modules/bootstrap/dist/css/bootstrap.min.css"
5. ],
6. "scripts": [
7. "node\_modules/jquery/dist/jquery.min.js",
8. "node\_modules/bootstrap/dist/js/bootstrap.min.js"
9. ]
10. ...

**Note:**When angular.json is modified, restart the server to see the changes reflected.

​ ​

Step 5:

Include the below code to **registration-form.component.css**

.ng-valid[required]  {

    border-left: 5px solid #42A948; /\* green \*/

  }

  .ng-invalid:not(form)  {

    border-left: 5px solid #a94442; /\* red \*/

  }

Line 1-3: ng-valid CSS class changes left border of the textbox to green if form control has valid input

Line 5-7: ng-invalid CSS class changes left border of the textbox to red if form control has invalid data

**Building Reactive Forms (Angular v13)**

Add the following code in the **registration-form.component.ts** file

**Building Reactive Forms (Angular v13)**

Add the following code in the **registration-form.component.ts** file

1. import { Component, OnInit } from '@angular/core';
2. import { FormBuilder, FormGroup, Validators } from '@angular/forms';
3. @Component({
4. selector: 'app-registration-form',
5. templateUrl: './registration-form.component.html',
6. styleUrls: ['./registration-form.component.css']
7. })
8. export class RegistrationFormComponent implements OnInit {
9. registerForm!: FormGroup;
10. submitted!:boolean;
12. constructor(private formBuilder: FormBuilder) { }
13. ngOnInit() {
14. this.registerForm = this.formBuilder.group({
15. firstName: ['', Validators.required],
16. lastName: ['', Validators.required],
17. address: this.formBuilder.group({
18. street: [],
19. zip: [],
20. city: []
21. })
22. });
23. }
24. }

Line 2: Import FormBuilder class to create a reactive form. Also, import FormGroup class to create a group of form controls and Validators for validation

Line 11: Create a property registerForm of type FormGroup

Line 14: Inject a FormBuilder instance using constructor

Line 17: formBuilder.group() method creates a FormGroup. It takes an object whose keys are FormControl names and values are their definitions

Line 18-24: Create form controls such as firstName, lastName, and address as a subgroup with fields street, zip, and city. These fields are form controls.

For each form control:

* you can mention the default value as the first argument and
* the list of validators as the second argument:
* Validations can be added to the form controls using the built-in validators supplied by the Validators class.
* For example: Configure built-in required validator for each control using [' ', Validators.required] syntax.
* If multiple validators are to be applied, then use the syntax [' ', [Validators.required, Validators.maxlength(10)]].

**registration-form.component.html**

1. <div class="container">
2. <h1>Registration Form</h1>
3. <form [formGroup]="registerForm">
4. <div class="form-group">
5. <label>First Name</label>
6. <input type="text" class="form-control" formControlName="firstName">
7. <div \*ngIf="registerForm.controls['firstName'].errors" class="alert alert-danger">
8. Firstname field is invalid.
9. <p \*ngIf="registerForm.controls['firstName'].errors?.['required']">
10. This field is required!
11. </p>
12. </div>
13. </div>
14. <div class="form-group">
15. <label>Last Name</label>
16. <input type="text" class="form-control" formControlName="lastName">
17. <div \*ngIf="registerForm.controls['lastName'].errors" class="alert alert-danger">
18. Lastname field is invalid.
19. <p \*ngIf="registerForm.controls['lastName'].errors?.['required']">
20. This field is required!
21. </p>
22. </div>
23. </div>
24. <div class="form-group">
25. <fieldset formGroupName="address">
26. <legend>Address:</legend>
27. <label>Street</label>
28. <input type="text" class="form-control" formControlName="street">
29. <label>Zip</label>
30. <input type="text" class="form-control" formControlName="zip">
31. <label>City</label>
32. <input type="text" class="form-control" formControlName="city">
33. </fieldset>
34. </div>
35. <button type="submit" class="btn btn-primary" (click)="submitted=true">Submit</button>
36. </form>
37. <br/>
38. <div [hidden]="!submitted">
39. <h3> Employee Details </h3>
40. <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
41. <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
42. <p> Street: {{ registerForm.get('address.street')?.value }}</p>
43. <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
44. <p> City: {{ registerForm.get('address.city')?.value }}</p>
45. </div>
46. </div>

Line 3: formGroup is a directive that binds HTML form with the FormGroup property created inside a component class. A FormGroup has been created in component with the name registerForm. Here form tag is bound with FormGroup name called registerForm

Line 6, 16: Two text boxes for first name and last name are bound with the form controls created in the component using formControlName directive

Line 7-13:  A validation error message is displayed when the firstName is modified and has validation errors.

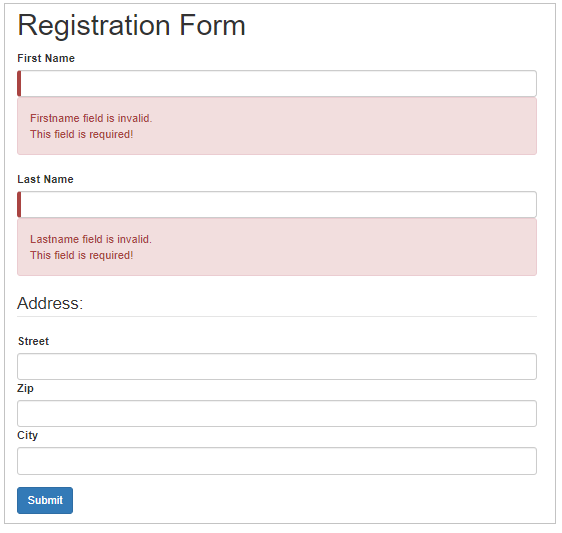
Line 17-22:  A validation error message is displayed when the lastName is modified and has validation errors.

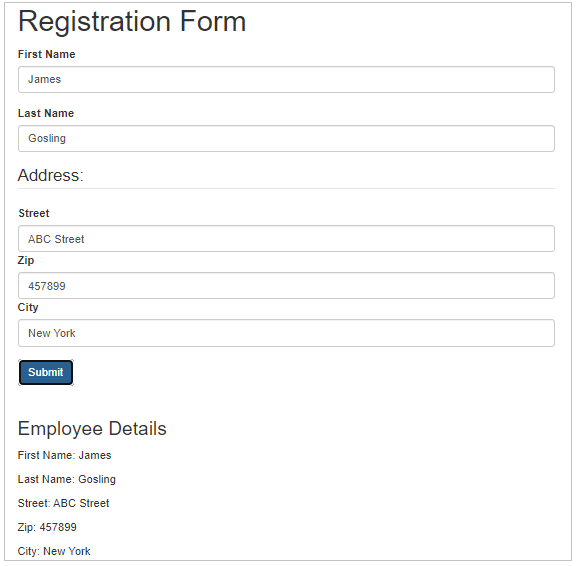
Line 35: When the submit button is clicked, it initializes the submitted property value to true

Line 38: div tag will be hidden if the form is not submitted

Line 39-44: Using the get() method of FormGroup, each FormControl value is fetched and rendered.

**Save all the files and observe the output:**





# Custom Validators in Reactive Forms

Need for custom validation:

While creating forms, there can be situations for which built-in validators are not available. Few such examples include validating a phone number, validating if the password and confirm password fields matches or not, etc.. In such situations, custom validators can be created to implement the required validation functionality.

Custom validation in Reactive Forms of Angular:

Custom validation can be applied to form controls of a Reactive Form in Angular.

* Custom validators are implemented as separate functions inside the component.ts file.
* these functions can be added to the list of other validators configured for a form control.

Implementing custom validation in Reactive Forms of Angular:

Add one more field called email inside the example used for ReactiveForms previously. The below are the validations to be applied to the 'email' field:

* required,
* checks for standard email pattern, for example: abc@something.com, abc@something.co.in, etc.

For implementing the for custom validation, add a separate function which checks for standard email pattern inside **registration-form.component.ts**as shown below:

1. import { Component, OnInit } from '@angular/core';
2. import { FormBuilder, FormControl, FormGroup, Validators } from '@angular/forms';
3. @Component({
4. selector: 'app-registration-form',
5. templateUrl: './registration-form.component.html',
6. styleUrls: ['./registration-form.component.css']
7. })
8. export class RegistrationFormComponent implements OnInit {
9. registerForm!: FormGroup;
10. submitted!:boolean;
12. constructor(private formBuilder: FormBuilder) { }
13. ngOnInit() {
14. this.registerForm = this.formBuilder.group({
15. firstName: ['', Validators.required],
16. lastName: ['', Validators.required],
17. address: this.formBuilder.group({
18. street: [],
19. zip: [],
20. city: []
21. }),
22. email: ['', validateEmail]
23. });
24. }
25. }
26. function validateEmail(c: FormControl): any {
27. let EMAIL\_REGEXP = /^([a-zA-Z0-9\_\-\.]+)@([a-zA-Z0-9\_\-\.]+)\.([a-zA-Z]{2,5})$/;
28. return EMAIL\_REGEXP.test(c.value) ? null : {
29. emailInvalid: {
30. message: "Invalid Format!"
31. }
32. };
33. }

Line 30-37: In this function, a regular expression pattern is taken for email and the input value of the form control is tested against the mentioned pattern. If the pattern matches, it means the entered input is valid and hence, the validation function returns null.  Otherwise, the function returns an object with name 'emailInvalid' with one property called 'message' set to appropriate string message.

Line 21: Binds the required validator and the custom validator named validateEmail to the email field.

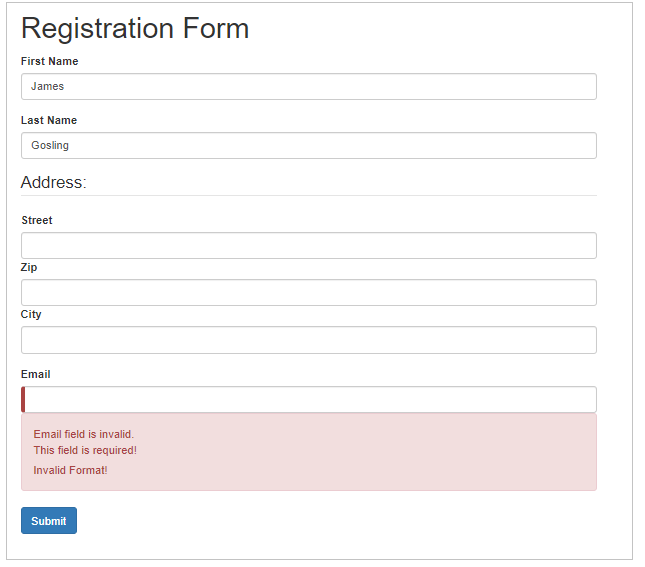
Custom Validators in ReactiveForms

For tracking the custom validators in the Reactive Form's template (Angular v13), add HTML controls for the email field in the registration-form.component.html file as shown below:

1. <div class="container">
2. <h1>Registration Form</h1>
3. <form [formGroup]="registerForm">
4. <div class="form-group">
5. <label>First Name</label>
6. <input type="text" class="form-control" formControlName="firstName">
7. <div \*ngIf="registerForm.controls['firstName'].errors" class="alert alert-danger">
8. Firstname field is invalid.
9. <p \*ngIf="registerForm.controls['firstName'].errors?.['required']">
10. This field is required!
11. </p>
12. </div>
13. </div>
14. <div class="form-group">
15. <label>Last Name</label>
16. <input type="text" class="form-control" formControlName="lastName">
17. <div \*ngIf="registerForm.controls['lastName'].errors" class="alert alert-danger">
18. Lastname field is invalid.
19. <p \*ngIf="registerForm.controls['lastName'].errors?.['required']">
20. This field is required!
21. </p>
22. </div>
23. </div>
24. <div class="form-group">
25. <fieldset formGroupName="address">
26. <legend>Address:</legend>
27. <label>Street</label>
28. <input type="text" class="form-control" formControlName="street">
29. <label>Zip</label>
30. <input type="text" class="form-control" formControlName="zip">
31. <label>City</label>
32. <input type="text" class="form-control" formControlName="city">
33. </fieldset>
34. </div>
35. <div class="form-group">
36. <label>Email</label>
37. <input type="text" class="form-control" formControlName="email" />
38. <div \*ngIf="registerForm.controls['email'].errors" class="alert alert-danger">
39. Email field is invalid.
40. <p \*ngIf="registerForm.controls['email'].errors?.['required']">
41. This field is required!
42. </p>
43. <p \*ngIf="registerForm.controls['email'].errors?.['emailInvalid']">
44. {{ registerForm.controls['email'].errors?.['emailInvalid'].message }}
45. </p>
46. </div>
47. </div>
48. <button type="submit" class="btn btn-primary" (click)="submitted=true">Submit</button>
49. </form>
50. <br/>
51. <div [hidden]="!submitted">
52. <h3> Employee Details </h3>
53. <p>First Name: {{ registerForm.get('firstName')?.value }} </p>
54. <p> Last Name: {{ registerForm.get('lastName')?.value }} </p>
55. <p> Street: {{ registerForm.get('address.street')?.value }}</p>
56. <p> Zip: {{ registerForm.get('address.zip')?.value }} </p>
57. <p> City: {{ registerForm.get('address.city')?.value }}</p>
58. <p>Email: {{ registerForm.get('email')?.value }}</p>
59. </div>
60. </div>

Line 44: Displays error message if email validation fails. errors object holds the error messages of all form controls

Output:



For tracking the custom validators in the Reactive Form's template (<Angular v13), add HTML controls for the email field in the registration-form.component.html file as shown below:

1. <div class="container">
2. <h1>Registration Form</h1>
3. <form [formGroup]="registerForm">
4. <div class="form-group">
5. <label>First Name</label>
6. <input type="text" class="form-control" formControlName="firstName" />
7. <div \*ngIf="registerForm.controls.firstName.errors" class="alert alert-danger">
8. Firstname field is invalid.
9. <p \*ngIf="registerForm.controls.firstName.errors?.required">
10. This field is required!
11. </p>
12. </div>
13. </div>
14. <div class="form-group">
15. <label>Last Name</label>
16. <input type="text" class="form-control" formControlName="lastName" />
17. <div \*ngIf="registerForm.controls.lastName.errors" class="alert alert-danger">
18. Lastname field is invaliddd.
19. <p \*ngIf="registerForm.controls.lastName.errors?.required">
20. This field is required!
21. </p>
22. </div>
23. </div>
24. <div class="form-group">
25. <fieldset formGroupName="address">
26. <label>Street</label>
27. <input type="text" class="form-control" formControlName="street" />
28. <label>Zip</label>
29. <input type="text" class="form-control" formControlName="zip" />
30. <label>City</label>
31. <input type="text" class="form-control" formControlName="city" />
32. </fieldset>
33. </div>
34. <div class="form-group">
35. <label>Email</label>
36. <input type="text" class="form-control" formControlName="email" />
37. <div \*ngIf="registerForm.controls.email.errors" class="alert alert-danger">
38. Email field is invalid.
39. <p \*ngIf="registerForm.controls.email.errors?.required">
40. This field is required!
41. </p>
42. <p \*ngIf="registerForm.controls.email.errors?.emailInvalid">
43. {{ registerForm.controls.email.errors?.emailInvalid.message }}
44. </p>
45. </div>
46. </div>
47. <button type="submit" class="btn btn-primary" (click)="submitted = true">
48. Submit
49. </button>
50. </form>
51. <br/>
52. <div [hidden]="!submitted">
53. <h3>Employee Details</h3>
54. <p>First Name: {{ registerForm.controls.firstName.value }}</p>
55. <p>Last Name: {{ registerForm.controls.lastName.value }}</p>
56. <p>Street: {{ registerForm.controls.address.value.street }}</p>
57. <p>Zip: {{ registerForm.controls.address.value.zip }}</p>
58. <p>City: {{ registerForm.controls.address.value.city }}</p>
59. <p>Email: {{ registerForm.controls.email.value }}</p>
60. </div>
61. </div>

Line 43: Displays error message if email validation fails. errors object holds the error messages of all form controls

Output:

