Angular forms

The Angular forms are used to collect the data from the user.

Angular forms module provides all the services out of the box.

- It binds the form field to the Angular component class.
- It tracks changes made to the form fields so that we can respond accordingly.
- provide the built-in validators to validate the inputs.
- presents the validation errors to the user.
- encapsulates all the input fields into an object structure when the user submits the form.

Angular takes two approaches to build the forms.

1. Template-driven forms approach
2. Reactive forms or model-driven forms approach

Template-driven forms approach

- is the easiest way to build the Angular forms.
- logic of the form is placed in the template.
- we specify behaviors/validations using directives and attributes in our template and let it work behind the scenes.
- All things happen in Templates hence very little code is required in the component class.
- To work with Template-driven forms, we must import the FormsModule. We usually import it in root module or in a shared module.

The Template-driven forms

1. The form is set up using ngForm directive
2. controls are set up using the ngModel directive
3. ngModel also provides the two-way data binding
4. The Validations are configured in the template via directives

Template-driven forms are

1. Contains little code in the component class
2. Easier to set up

While they are

1. Difficult to add controls dynamically
2. Unit testing is a challenge

Example: Component template

```
<form #contactForm="ngForm" (ngSubmit)="onSubmit(contactForm)">

 <p>
  <label for="firstname">First Name</label>
  <input type="text" name="firstname" ngModel>
 </p>

 <p>
  <label for="lastname">Last Name</label>
  <input type="text" name="lastname" ngModel>
 </p>

 <p>
  <label for="email">Email </label>
  <input type="text" id="email" name="email" ngModel>
 </p>

 <p>
  <label for="gender">Geneder</label>
  <input type="radio" value="male" name="gender" ngModel> Male
  <input type="radio" value="female" name="gender" ngModel> Female
 </p>

 <p>
  <label for="isMarried">Married</label>
  <input type="checkbox" name="isMarried" ngModel>
 </p>

 <select name="country" ngModel>
  <option [ngValue]="c.id" *ngFor="let c of countryList">
   {{c.name}}
  </option>
 </select>

 <p>
  <button type="submit">Submit</button>
 </p>

</form>
```

Component class

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

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

```
  title = 'Template driven forms';

  countryList:country[] = [
    new country("1", "India"),
    new country('2', 'USA'),
    new country('3', 'England')
  ];
}

export class country {
  id:string;
  name:string;

  constructor(id:string, name:string) {
    this.id=id;
    this.name=name;
  }
 onSubmit(contactForm) {
    console.log(contactForm.value);
  }

}
```

Template Driven form Validation

- Validations in Template-driven forms are provided by the Validation directives.
- The Angular Forms Module comes with several built-in validators.
- You can also create your own custom Validator.

Required Validation

The required validator returns true only if the form control has non-empty value entered. Let us add this validator to all fields

```
<input type="text" id="firstname" name="firstname" required minlength="10"
[(ngModel)]="contact.firstname">
```

Minlength Validation

This Validator requires the control value must not have less number of characters than the value specified in the validator.

```
<input type="text" id="firstname" name="firstname" required minlength="10"
[(ngModel)]="contact.firstname">
```

Maxlength Validation

This Validator requires that the number of characters must not exceed the value of the attribute.

```
<input type="text" id="lastname" name="lastname" required maxlength="15"
[(ngModel)]="contact.lastname">
```

Pattern Validation
This Validator requires that the control value must match the regex pattern provided in the attribute. For example, the pattern ^[a-zA-Z]+$ ensures that the only letters are allowed (even spaces are not allowed). Let us apply this pattern to the lastName

<input type="text" id="lastname" name="lastname" required maxlength="15" pattern="^[a-zA-Z]+$" [(ngModel)]="contact.lastname">

Email Validation
This Validator requires that the control value must be a valid email address. We apply this to the email field

<input type="text" id="email" name="email" required email [(ngModel)]="contact.email">

Disable Submit button
Now, we have successfully added the validators. You will notice that the click submit button still submits the form.

We need to disable the submit button if our form is not valid.

- Angular forms module keep track of the state of our form and each of its form elements. These states are exposed to the user through FormGroup, FormArray & FormControl objects.

- We get the reference to the top-level FormGroup instance by creating a template variable and bind it to ngForm. We have already done it when we had added the #contactForm="ngForm" in our form tag.

- The FormGroup has a valid property, which is set to true if all of its child controls are valid. We use it to set the disabled attribute of the submit button.

- So long as contactForm.valid remains false, the submit button remains disable

Displaying the Validation/Error messages

<input type="text" id="firstname" name="firstname" required minlength="10" #firstname="ngModel" [(ngModel)]="contact.firstname">

Now, we have a reference to the firstname FormControl instance, we can check its status. We use the valid property to check if the firstname has any errors.
valid: returns either invalid status or null which means a valid status

*<div *ngIf="!firstname?.valid && (firstname?.dirty || firstname?.touched)">*
Invalid First Name
* *

Model-driven forms approach

- the logic of the form is defined in the component as an object.
- The Model-driven approach has more benefits as it makes the testing of the component easier.
- the representation of the form is created in the component class.
- This form model is then bound to the HTML elements. it is done using the special markups.
- The model-driven forms are created in component class, where Form fields are created as properties of our component class. This makes it easier to test.

How to use Reactive Forms

1. Import ReactiveFormsModule
2. Create Form Model in component class using Form Group, Form Control & Form Arrays
3. Create the HTML Form resembling the Form Model.
4. Bind the HTML Form to the Form Model

- In Reactive Forms approach, it is our responsibility to build the Model using FormGroup, FormControl and FormArray.
- The FormGroup, FormControl & FormArray are the three building blocks of the Angular Forms. We learned about them in Angular Forms Tutorial.
- FormControl encapsulates the state of a *single form element* in our form. It stores the value and state of the form element and helps us to interact with them using properties & methods.
- FormGroup represents a collection of form Controls. It can also contain form groups and form arrays. In fact, an angular form is a FormGroup.

```
contactForm = new FormGroup({
  firstname: new FormControl(),
  lastname: new FormControl(),
  email: new FormControl(),
  gender: new FormControl(),
  isMarried: new FormControl(),
  country: new FormControl()
})
```

```
<form">

 <p>
  <label for="firstname">First Name </label>
  <input type="text" id="firstname" name="firstname">
 </p>
```

```
<p>
  <label for="lastname">Last Name </label>
  <input type="text" id="lastname" name="lastname">
</p>

<p>
  <label for="email">Email </label>
  <input type="text" id="email" name="email">
</p>

<p>
  <label for="gender">Geneder </label>
  <input type="radio" value="male" id="gender" name="gender"> Male
  <input type="radio" value="female" id="gender" name="gender"> Female
</p>

<p>
  <label for="isMarried">Married </label>
  <input type="checkbox" id="isMarried" name="isMarried">
</p>

<p>
  <label for="country">country </label>
  <select id="country" name="country">
    <option [ngValue]="c.id" *ngFor="let c of countryList">
      {{c.name}}
    </option>
  </select>
</p>

<p>
  <button type="submit">Submit</button>
</p>

</form>
```

we need to associate our model to the Template. We need to tell angular that we have a model for the form using formgroup and formControlName attributes.