o build reactive forms, first, we need to import ReactiveFormsModule. We then create the Form Model in component class using [Form Group](https://www.tektutorialshub.com/angular/formgroup-in-angular/), [Form Control](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) & [FormArrays](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/). Next, we will create the HTML form template and bind it to the Form Model.

00:19/03:20

**Table of Contents**

* [What are Reactive Forms?](https://www.tektutorialshub.com/angular/angular-reactive-forms/#what-are-reactive-forms)
* [How to use Reactive Forms](https://www.tektutorialshub.com/angular/angular-reactive-forms/#how-to-use-reactive-forms)
* [Reactive Forms Example Application](https://www.tektutorialshub.com/angular/angular-reactive-forms/#reactive-forms-example-application)
* [Import ReactiveFormsModule](https://www.tektutorialshub.com/angular/angular-reactive-forms/#import-reactiveformsmodule)
* [Model](https://www.tektutorialshub.com/angular/angular-reactive-forms/#model)
  + [Creating the Form Model](https://www.tektutorialshub.com/angular/angular-reactive-forms/#creating-the-form-model)
  + [Adding the Child Controls](https://www.tektutorialshub.com/angular/angular-reactive-forms/#adding-the-child-controls)
* [HTML Form](https://www.tektutorialshub.com/angular/angular-reactive-forms/#html-form)
  + [Binding the template to the model](https://www.tektutorialshub.com/angular/angular-reactive-forms/#binding-the-template-to-the-model)
  + [Submit form](https://www.tektutorialshub.com/angular/angular-reactive-forms/#submit-form)
* [Final Template](https://www.tektutorialshub.com/angular/angular-reactive-forms/#final-template)
  + [Receive the data in the Component class](https://www.tektutorialshub.com/angular/angular-reactive-forms/#receive-the-data-in-the-component-class)
  + [Test the form](https://www.tektutorialshub.com/angular/angular-reactive-forms/#test-the-form)
* [FormControl](https://www.tektutorialshub.com/angular/angular-reactive-forms/#formcontrol)
  + [Default Value](https://www.tektutorialshub.com/angular/angular-reactive-forms/#default-value)
  + [Sync Validator](https://www.tektutorialshub.com/angular/angular-reactive-forms/#sync-validator)
  + [Asynchronous validator](https://www.tektutorialshub.com/angular/angular-reactive-forms/#asynchronous-validator)
* [Grouping the controls using FormGroup](https://www.tektutorialshub.com/angular/angular-reactive-forms/#grouping-the-controls-using-formgroup)
  + [Summary](https://www.tektutorialshub.com/angular/angular-reactive-forms/#summary)

**What are Reactive Forms?**

Reactive forms are forms where we define the structure of the form in the component class. i.e. we create the form model with [Form Groups](https://www.tektutorialshub.com/angular/formgroup-in-angular/), [Form Controls](https://www.tektutorialshub.com/angular/formcontrol-in-angular/), and [FormArrays](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/).

We also define the validation rules in the component class. Then, we bind it to the HTML form in the template.

This is different from the template-driven forms, where we define the logic

**How to use Reactive Forms**

1. Import ReactiveFormsModule
2. Create Form Model in component class using FormGroup, FormControl & FormArrays
3. Create the HTML Form resembling the Form Model.
4. Bind the HTML Form to the Form Model

**Reactive Forms Example Application**

Use ng new to create a new application

|  |  |
| --- | --- |
| 1  2  3 | ng new mdf  --routing=true --style=css |

Run ng serve and verify if everything is installed correctly.

**Import ReactiveFormsModule**

To work with Reactive forms, we must import the ReactiveFormsModule. We usually import it in root module or in a shared module. The ReactiveFormsModule contains all the form directives and constructs for working with angular reactive forms.

|  |
| --- |
| import { BrowserModule } from '@angular/platform-browser';  import { NgModule } from '@angular/core';  import { ReactiveFormsModule } from '@angular/forms';    import { AppRoutingModule } from './app-routing.module';  import { AppComponent } from './app.component';    @NgModule({    declarations: [      AppComponent    ],    imports: [      BrowserModule,      AppRoutingModule,      ReactiveFormsModule    ],    providers: [],    bootstrap: [AppComponent]  })  export class AppModule { } |

**Model**

In the [template-driven approach](https://www.tektutorialshub.com/angular/angular-template-driven-forms/), we used ngModel & ngModelGroup directive on the HTML elements. The FormsModule automatically creates the [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) & [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/)instances from the HTML template. This happens behind the scene.

In the Reactive Forms approach, It is our responsibility to build the Model using [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/), [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) and [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/).

The [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/), [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) & [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) are the three [building blocks of the Angular Forms](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/#building-blocks-of-angular-forms). We learned about them in the [Angular Forms Tutorial](https://www.tektutorialshub.com/angular/angular-forms-fundamentals/).

[FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/) 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](https://www.tektutorialshub.com/angular/formgroup-in-angular/) represents a collection of form controls. It can also contain other FormGroups and FormArrays. In fact, an angular form is a FormGroup.

Let’s create the model for our Form.

First, we need to import FormGroup, FormControl & Validator from the @angular/forms. Open the app.component.ts and the add following import statement.

|  |  |
| --- | --- |
| 1  2  3 | import { FormGroup, FormControl, Validators } from '@angular/forms' |

**Angular**

**Creating the Form Model**

The FormGroup is created with the following syntax

|  |  |
| --- | --- |
|  | contactForm = new FormGroup({}) |

In the above, we have created an instance of a FormGroup and named it as contactForm. The contactForm is top level FormGroup and is the name of our form model.

The FormGroup takes 3 arguments. a collection of a child controls (which can be FormControl, FormArray, or another FormGroup), a validator, and an asynchronous validator. The validators are optional.

**Adding the Child Controls**

The next step is to add the child controls to the contactForm.

The first argument to FormGroup is the collection of controls. The Controls can be FormControl, FormArray or another FormGroup. It can be done by creating a new instance for FormControl (or FormGroup or FormArray)

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

In the above example, we have added five FormControl instances each representing the properties firstname. lastname.email, gender, ismarried & country.

The Other two arguments to FormGroup are Sync Validator & Async Validator. They are optional.

With this our model is ready.

**HTML Form**

The next task is to build an HTML form. The following is a regular HTML form. We enclose it in a <form> tag. We have included two text inputs (FirstName & LastName), an email field, a radio button (gender), a checkbox (isMarried), and a select list (country). These are simple HTML Form elements.

|  |  |
| --- | --- |
|  | <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 value="1">India</option>        <option value="2">USA</option>        <option value="3">England</option>        <option value="4">Singapore</option>      </select>    </p>      <p>      <button type="submit">Submit</button>    </p>    </form> |

**Binding the template to the model**

Now we need to associate our model with the above HTML Template. We need to tell angular that we have a model for the form.

This is done using the formGroup directive as shown below.

|  |  |
| --- | --- |
|  | <form [formGroup]="contactForm"> |

We use the square bracket ([one-way binding](https://www.tektutorialshub.com/angular/angular-data-binding/)) around FormGroup directive and assign our form model (i.e. contactForm name that we gave to our model in the component class) to it.

Next, we need to bind each form field to an instance of the FormControl models. We use the FormControlName directive for this. We add this directive to every form field element in our form. The value is set to the name of the corresponding FormControl instance in the component class.

|  |  |
| --- | --- |
|  | <input type="text" id="firstname" name="firstname" formControlName="firstname">  <input type="text" id="lastname" name="lastname" formControlName="lastname"> |

**Submit form**

We submit the form data to the component using the [Angular directive](https://www.tektutorialshub.com/angular/angular-directives/) named ngSubmit.  Note that we already have a submit button in our form. The ngSubmit directive binds itself to the click event of the submit button. We are using [event binding](https://www.tektutorialshub.com/angular/angular-data-binding/) (parentheses) to bind ngSubmit to OnSubmit method. When the user clicks on the submit button ngSubmit invokes the OnSubmit method on the Component class

|  |
| --- |
| <form [formGroup]="contactForm" (ngSubmit)="onSubmit()"> |

We are yet to add the onSubmit() method in the component class. Hence the Angular will throw an error here.

**Final Template**

Our Final Template is as shown below

|  |  |
| --- | --- |
| 1 | <form [formGroup]="contactForm" (ngSubmit)="onSubmit()">      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" formControlName="firstname">    </p>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" formControlName="lastname">    </p>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email" formControlName="email">    </p>      <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" formControlName="gender"> Male      <input type="radio" value="female" id="gender" name="gender" formControlName="gender"> Female    </p>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" formControlName="isMarried">    </p>        <p>      <label for="country">country </label>      <select id="country" name="country" formControlName="country">        <option value="1">India</option>        <option value="2">USA</option>        <option value="3">England</option>        <option value="4">Singapore</option>      </select>    </p>        <p>      <button type="submit">Submit</button>    </p>    </form> |

**Receive the data in the Component class**

The last step is to receive the form data in the component class. All we need to do is to create the onSubmit method in our component class.

|  |  |
| --- | --- |
|  | onSubmit() {    console.log(this.contactForm.value);  } |

We are using the console.log(this.contactForm.value) to send the value of our form data to the console window.

The final component class is shown below.

|  |  |
| --- | --- |
|  | **import { Component } from '@angular/core';**  **import { FormGroup, FormControl, Validators } from '@angular/forms'**    **@Component({**  **selector: 'app-root',**  **templateUrl: './app.component.html',**  **styleUrls: ['./app.component.css']**  **})**  **export class AppComponent {**  **title = 'mdf';**    **contactForm = new FormGroup({**  **firstname: new FormControl(),**  **lastname: new FormControl(),**  **email: new FormControl(),**  **gender: new FormControl(),**  **isMarried: new FormControl(),**  **country: new FormControl()**  **})**      **onSubmit() {**  **console.log(this.contactForm.value);**  **}**  **}** |

**Test the form**

Now you can run the app and see the result. Open the developer console and see the value returned by the contactForm.value. The values of the form are returned as JSON object, which you can pass it your backend API to persist the information to the database.

**FormControl**

A FormControl takes 3 arguments. a default value, a validator, and an asynchronous validator. All of them are optional.

**Default Value**

You can pass a default value either as a string or as an object of key-value pair. When you pass an object you can set both the value and the whether or not the control is disabled.

|  |  |
| --- | --- |
| 1  2  3  4 | //Setting Default value as string  firstname= new FormControl(‘Sachin’); |

|  |  |
| --- | --- |
| 1  2  3  4 | //Setting Default value & disabled state as object  firstname: new FormControl({value: ‘Rahul’, disabled: true}), |

**Sync Validator**

The second parameter is an array of sync Validators. Angular has some built-in Validators such as required and minLength etc.

You can pass the Validator function as shown below.

|  |  |
| --- | --- |
| 13 | firstname: new FormControl('', [Validators.required,Validators.minLength(10)]), |

**Asynchronous validator**

The third argument is the Async Validator. The syntax of Async Validators is similar to Sync Validators.

More on validation in our next tutorial Validations in Reactive forms.

**Grouping the controls using FormGroup**

We can group various FormControls together. For Example fields such as street, city, country and Pincode each will have their own FormControl but can be grouped together as an address FormGroup

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | contactForm = new FormGroup({    firstname: new FormControl(),    lastname: new FormControl(),    email: new FormControl(),    gender: new FormControl(),    isMarried: new FormControl(),    address:new FormGroup({      city: new FormControl(),      street: new FormControl(),      pincode:new FormControl(),      country: new FormControl(),    })  }) |

In the code above, we have created new FormGroup Address and added three form controls i.e city, street, country & Pincode

In the template use the formGroupName directive to enclose the control using a div element as shown below

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | <div formGroupName="address">        <div class="form-group">          <label for="city">City</label>          <input type="text" class="form-control" name="city" formControlName="city" >      </div>        <div class="form-group">          <label for="street">Street</label>          <input type="text" class="form-control" name="street" formControlName="street" >      </div>        <div class="form-group">          <label for="pincode">Pin Code</label>          <input type="text" class="form-control" name="pincode" formControlName="pincode">      </div>        <p>        <label for="country">country </label>        <select id="country" name="country" formControlName="country">          <option value="1">India</option>          <option value="2">USA</option>          <option value="3">England</option>          <option value="4">Singapore</option>        </select>      </p>      </div> |

**Summary**

We learned how to build Angular Reactive Forms in this tutorial. In the next tutorial, we will add validation rules to our application.

### What is a Validator

A Validator is a function that checks the instance of [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/), [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) or a [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) and returns a list of errors. If the Validator returns a null means that validation has passed

### How to add a Validator to Reactive Forms

We configure the validators as the second and third argument to the [FormControl](https://www.tektutorialshub.com/angular/formcontrol-in-angular/), [FormGroup](https://www.tektutorialshub.com/angular/formgroup-in-angular/) or [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) in the component class. The second argument is a collection of **sync validators** and the third argument is a collection of an **async validators**.

sync validators runs validations and returns immediately. They either return a list of errors or null if no errors found.

async validators: returns a **Promise** or **Observable**. They either return a list of errors or null if no errors are found.

### Built-in Validators

The Angular ReactiveForms Module provides several Built-in validators out of the box. They are required, minlength, maxlength & pattern etc.

## Reactive Forms Validation Example

W learned [how to create Angular Reactive Forms](https://www.tektutorialshub.com/angular/angular-reactive-forms/) in the previous tutorial. We will now add some of the built-in validators to that example.

### Model

Here is the contactForm model from the previous tutorial.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | contactForm = new FormGroup({      firstname: new FormControl(''),      lastname: new FormControl(''),      email:new FormControl(''),      gender: new FormControl(''),      isMarried: new FormControl(''),      country: new FormControl(''),      address:new FormGroup({        city: new FormControl(''),        street: new FormControl(''),        pincode:new FormControl('')      })    }) |

### Disabling the Browser validation

First, we need to disable browser validator by adding the [novalidate](https://developer.mozilla.org/en-US/docs/Web/HTML/Element/form) attribute to the *<form>* element as shown below. If this attribute is present then the form is not validated by the built-in HTML5 validation when submitted.

|  |  |
| --- | --- |
| 1  2  3 | <form [formGroup]="contactForm" (ngSubmit)="onSubmit()" novalidate> |

### Adding in Built-in Validators

The mentioned earlier, the Angular has provided several built-in validators out of the box.

#### Required Validator

The required validator is a sync validator, which returns true only if the formcontrol has a non-empty value entered. The second argument of the FormControl takes the Sync Validator.

|  |  |
| --- | --- |
| 1  2  3 | firstname: new FormControl('',[Validators.required]), |

#### Minlength Validator

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

For Example, minlength validator ensures that the firstname value has at least 10 characters.

|  |  |
| --- | --- |
| 1  2  3 | firstname: new FormControl('',[Validators.required,Validators.minLength(10)]), |

#### Maxlength Validator

This Validator requires that the number of characters must not exceed the value specified in the validator.

|  |  |
| --- | --- |
| 1  2  3 | lastname: new FormControl('',[Validators.maxLength(15)]), |

#### **Pattern Validator**

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

|  |  |
| --- | --- |
| 1  2  3 | lastname: new FormControl('',[Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$")]), |

#### Email Validator

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

|  |  |
| --- | --- |
| 1  2  3 | email:new FormControl('',[Validators.email,Validators.required]), |

After adding all the validators, our final contactForm will look like this.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | contactForm = new FormGroup({      firstname: new FormControl('',[Validators.required,Validators.minLength(10)]),      lastname: new FormControl('',[Validators.required, Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$")]),      email:new FormControl('',[Validators.email,Validators.required]),      gender: new FormControl('',[Validators.required]),      isMarried: new FormControl('',[Validators.required]),      country: new FormControl('',[Validators.required]),      address:new FormGroup({        city: new FormControl('',[Validators.required]),        street: new FormControl('',[Validators.required]),        pincode:new FormControl('',[Validators.required])      })    }) |

## Disable Submit button

We have successfully added the validators. Now, we need to disable the submit button if our form is not valid.

The Angular Forms API exposes the state of the forms through the FormGroup, FormControl & [FormArray](https://www.tektutorialshub.com/angular/angular-formarray-example-in-reactive-forms/) instances. The FormGroup control has a property valid, which is set to true if all of its child controls are valid.

The contactForm represents the top-level FormGroup. We use it to set the disabled attribute of the submit button.

|  |  |
| --- | --- |
| 1  2  3 | <button type="submit" [disabled]="!contactForm.valid">Submit</button> |

## Displaying the Validation/Error messages

We need to provide a short and meaningful error message to the user. We do that by using the error object returned by the FormControl instance

Every form element has a FormControl instance associated with it. It exposes the state of form element like valid, dirty, touched etc.

There are two ways in which you can get the reference to the FormControl.

One way is to use the contactForm variable. We can use contactForm.controls.firstname.valid to find out if the firstname is valid.

|  |  |
| --- | --- |
|  | <div      \*ngIf="!contactForm.controls.firstname?.valid && (contactForm.controls.firstname?.dirty      ||contactForm.controls.firstname?.touched)">        First Name is not valid    </div> |

The other way to is to define getter function for each FormControl instance in the component class.

|  |  |
| --- | --- |
| 1  2  3  4  5 | get firstname() {     return this.contactForm.get('firstname');  } |

and then use it in the template as follows

|  |  |
| --- | --- |
| 1  2  3  4  5 | <div \*ngIf="!firstname.valid && (firstname.dirty ||firstname.touched)">        First Name is not valid    </div> |

### Dirty & touched

Apart from checking valid we are also checking for the dirty & touched. Because we do not want the application to display the error when the form is displayed for the first time. We want to display errors only after the user has attempted to change the value. The dirty & touched properties help us do that.

dirty: A control is dirty if the user has changed the value in the UI.  
touched: A control is touched if the user has triggered a blur event on it.

### Error message

The error message “First Name is not valid ” is not helpful. The firstname has two validators. required and minlength

Any errors generated by the failing validation is updated in the errors object. The errors object returns the error object or null if there are no errors.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <div      \*ngIf="!firstname?.valid && (firstname?.dirty ||firstname?.touched)">      <div [hidden]="!firstname.errors.required">        First Name is required      </div>      <div [hidden]="!firstname.errors.minlength">        Min Length is 10      </div>    </div> |

## Final Code

**app.component.ts**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109 | import { Component, ViewChild, ElementRef } from '@angular/core';  import { FormGroup, FormControl, Validators } from '@angular/forms'      @Component({    selector: 'app-root',    templateUrl: './app.component.html',    styleUrls: ['./app.component.css']  })  export class AppComponent {    title = 'Angular Reactive forms';        contactForm = new FormGroup({      firstname: new FormControl('',[Validators.required,Validators.minLength(10)]),      lastname: new FormControl('',[Validators.required, Validators.maxLength(15), Validators.pattern("^[a-zA-Z]+$")]),      email:new FormControl('',[Validators.email,Validators.required]),      gender: new FormControl('',[Validators.required]),      isMarried: new FormControl('',[Validators.required]),      country: new FormControl('',[Validators.required]),      address:new FormGroup({        city: new FormControl('',[Validators.required]),        street: new FormControl('',[Validators.required]),        pincode:new FormControl('',[Validators.required])      })    })      get firstname() {      return this.contactForm.get('firstname');    }      get lastname() {      return this.contactForm.get('lastname');    }      get email() {      return this.contactForm.get('email');    }      get gender() {      return this.contactForm.get('gender');    }      get isMarried() {      return this.contactForm.get('isMarried');    }      get country() {      return this.contactForm.get('country');    }      get city() {      return this.contactForm.get("address").get('city');    }      get street() {      return this.contactForm.get("address").get('street');    }      get pincode() {      return this.contactForm.get("address").get('pincode');    }        countryList: country[] = [      new country("1", "India"),      new country('2', 'USA'),      new country('3', 'England')    ];            onSubmit() {      console.log(this.contactForm.value);    }        }      export class contact {    firstname:string;    lastname:string;    gender:string;    isMarried:boolean;    country:string;    address: {      city:string;      street:string;      pincode:string;    }  }      export class country {    id: string;    name: string;      constructor(id: string, name: string) {      this.id = id;      this.name = name;    }  } |

**app.component.html**

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152 | <form [formGroup]="contactForm" (ngSubmit)="onSubmit()" novalidate>      <p>      <label for="firstname">First Name </label>      <input type="text" id="firstname" name="firstname" formControlName="firstname">    </p>      <div      \*ngIf="!firstname?.valid && (firstname?.dirty ||firstname?.touched)">      <div [hidden]="!firstname.errors.required">        First Name is required      </div>      <div [hidden]="!firstname.errors.minlength">        Min Length is 10      </div>    </div>      <p>      <label for="lastname">Last Name </label>      <input type="text" id="lastname" name="lastname" formControlName="lastname">    </p>      <div \*ngIf="!lastname.valid && (lastname.dirty ||lastname.touched)">      <div [hidden]="!lastname.errors.pattern">        Only characters are allowed      </div>      <div [hidden]="!lastname.errors.maxLength">        Max length allowed is {{lastname.errors.maxlength?.requiredLength}}      </div>      <div [hidden]="!lastname.errors.required">        Last Name is required      </div>    </div>      <p>      <label for="email">Email </label>      <input type="text" id="email" name="email" formControlName="email">    </p>    <div \*ngIf="!email.valid && (email.dirty ||email.touched)">      <div [hidden]="!email.errors.required">        email is required      </div>      <div [hidden]="!email.errors.email">        invalid email id      </div>    </div>        <p>      <label for="gender">Geneder </label>      <input type="radio" value="male" id="gender" name="gender" formControlName="gender"> Male      <input type="radio" value="female" id="gender" name="gender" formControlName="gender"> Female    </p>    <div \*ngIf="!gender.valid && (gender.dirty ||gender.touched)">      <div [hidden]="!gender.errors.required">        gender is required      </div>    </div>      <p>      <label for="isMarried">Married </label>      <input type="checkbox" id="isMarried" name="isMarried" formControlName="isMarried">    </p>    <div \*ngIf="!isMarried.valid && (isMarried.dirty ||isMarried.touched)">      <div [hidden]="!isMarried.errors.required">        isMarried is required      </div>    </div>        <p>      <label for="country">country </label>      <select id="country" name="country" formControlName="country">        <option [ngValue]="c.id" \*ngFor="let c of countryList">          {{c.name}}        </option>      </select>    </p>    <div \*ngIf="!country.valid && (country.dirty ||country.touched)">      <div [hidden]="!country.errors.required">        country is required      </div>    </div>          <div formGroupName="address">        <div class="form-group">        <label for="city">City</label>        <input type="text" class="form-control" name="city" formControlName="city">      </div>      <div \*ngIf="!city.valid && (city.dirty ||city.touched)">        <div [hidden]="!city.errors.required">          city is required        </div>      </div>          <div class="form-group">        <label for="street">Street</label>        <input type="text" class="form-control" name="street" formControlName="street">      </div>      <div \*ngIf="!street.valid && (street.dirty ||street.touched)">        <div [hidden]="!street.errors.required">          street is required        </div>      </div>        <div class="form-group">        <label for="pincode">Pin Code</label>        <input type="text" class="form-control" name="pincode" formControlName="pincode">      </div>      <div \*ngIf="!pincode.valid && (pincode.dirty ||pincode.touched)">        <div [hidden]="!pincode.errors.required">          pincode is required        </div>      </div>      </div>      <p>{{contactForm.valid}} </p>      <p>      <button type="submit" [disabled]="!contactForm.valid">Submit</button>    </p>      </form>          <!-- <div ngModelGroup="address">      <p>      <label for="city">City</label>      <input type="text" id="city" name="city">    </p>      <p>      <label for="street">Street</label>      <input type="text" id="street" name="street">    </p>    <p>      <label for="pincode">Pin Code</label>      <input type="text" id="pincode" name="pincode">    </p>    </div> --> |