@angular/forms

building forms with angular

@angular/forms

- forms are used to capture user data in web applications
- @angular/forms we can:
  - easily capture user input
  - validate the inputs and the form
- For the sake of this tutorial we will build a registration form containing the following fields
  - username
  - email
  - phone
  - password
  - repeat password

Install @angular/forms

- create a new project with angular cli
- angular cli already installed the **@angular/forms** package
  - **npm install @angular/forms --save**
- you need to include the **FormsModule** in the **imports** array of the module you want to add it to.
- HTML has form validation built in, when adding **@angular/forms** the forms will have a **novalidate**
- form has a default behaviour when adding **@angular/forms** angular will prevent the default behaviour

Installing bootstrap

- bootstrap is a library containing css, js, and font files
- the library helps us deal with certain UI issues
- using bootstrap we can easily build a responsive app
- using bootstrap properly will cause our HTML structure to be more unified
- to build our registration we will provide global styling to the entire application
- install bootstrap using npm
  - **npm install bootstrap@4.0.0-beta --save**
- there is a file called **styles.scss** that you can use to import global styling
- let's build our form in **app.component.html** using bootstrap
  - some useful classes: **.container, .mt-5, .row, .col-*, .justify-content-center, .form-group, .form-control, .btn, .btn-primary**

Binding form input to class property

- In our registration form we want to grab the user input
- the easiest way to do that is with a directive in **@angular/forms** called **NgModel**
- **NgModel** defined **exportAs** in the directive metadata, so to attach template reference variable we use this syntax:
  - **<input [(ngModel)]=”someProperty” #directiveInstance=”ngModel” />**
- let’s try to examine the instance of **ngModel** directive by using template reference variable and the **ViewChild** decorator
- the **OnInit** hook will be called after the property will be populated with the **NgModel** instance

NgModel

- let’s examine common properties and methods on the **NgModel** instance
- **NgModel extends NgControl**
- **NgControl extends AbstractControlDirective**
- some common properties:
  - **dirty: boolean**
  - **errors: ValidationErrors**
  - **pristine: boolean**
  - **touched: boolean**
  - **untouched: boolean**
  - **reset**

adding validation

- NgModel added the following classes to the input
  - **ng-valid/ng-invalid**
  - **ng-dirty/ng-pristine**
  - **ng-touched/ng-untouched**
- adding the proper html attribute will add the validation to the input
- html validation attributes:
  - required
  - pattern
  - maxlength
  - minlength
- you can also add **email** attribute to validate email address

view the errors

- **NgModel** instance contains the errors in a property called: **errors**
- the errors are a dictionary
- the keys of the dictionary match the name of the validation property
- let’s try to view the errors on an input

display custom error messages

- most of the time we would like to add error message
- we can use the template reference variable containing the instance of the **ngModel** along with the errors dictionary to create custom messages

NgForm

- when adding **FormsModule** to the **imports** array of the module, angular will add the **NgForm** directive to every form
- let's use **ViewChild** to check the instance of the **NgForm** directive
- **NgForm extends ControlContainer**
- **abstract class ControlContainer extends AbstractControlDirective**
- the **NgForm** and **NgModel** have the same parent **AbstractControlDirective**
- a lot of the properties repeat in **NgForm**
- **NgForm** contains a property **controls** that is a dictionary with keys of the name of the inputs that has **NgModel** and the value is **FormControl,** the control will be populated **AfterViewInit**
- **AfterViewInit** is called after angular initializes the component view and the children view
- **NgForm** has **exportAs: ngForm**
- **NgForm** is valid if all the **FormControl** in it are valid

Summary

- using template language and directive from **FormsModule** we can add forms and forms validation to our forms
- **NgModel** can be used more then just two way binding and we can also use it to add validation to our forms
- **NgForm** contains validation to our entire form