Angular 2 Forms

Sang Shin
JPassion.com
"Code with Passion!"

Topics

- Angular Form
- Registering input form controls to Angular
- Validation
- Form state and Angular generated CSS classes
- Default values with Property Binding
- Two-way binding with ngModel
- ngModelGroup
- Radio and Select controls
- Conditionally display error message
- Conditionally disable Submit button

Angular Form

What is Angular Form?

- Angular generates an internal form object (NgForm type)
  - Coordinates a set of data-bound user controls
  - Tracks changes, validates input, and presents errors
  - Provide strong visual feedback using special CSS classes that track the state of the controls

Registering Form Controls

Angular recognizes <form> element

- When Angular encounters <form> element in a template, it creates Angular version of form object (NgForm type)
  - In other words, the <form> is now under control of Angular
- But it does not recognize the <input> control – you have to tell it via *ngModel* directive for each <input> control
- You also need to add name=”..” attribute

```
<form ...>
    <input type="text"
        class="form-control"
        id="username"
        ngModel
        name='username'>
```

Enable Form Submission

- We do not want to use *action* attribute in our form (because we are not sending the form to the server), instead we want to use *(ngSubmit)=”onSubmit(..)”* so that Angular receives the form

```
<form (ngSubmit)=”onSubmit(..)”>
    <input type="text"
        class="form-control"
        id="username"
        ngModel
        name='username'>
```

How to create a reference to NgForm object

- How can we pass the Angular managed form object (NgForm object) to the onSubmit(..) method?
- Make local reference with “ngForm” - this will reference to the Angular managed form object (not the DOM form element)

```
<form (ngSubmit)=”onSubmit(myform)” #myform =”ngForm”>
    <input type="text"
        class="form-control"
        id="username"
        ngModel
        name='username'>
```

How to receive NgForm object?

- You can then receive it as NgForm object in the component

```
import { Component, OnInit } from '@angular/core';
import { NgForm } from "@angular/forms";

@Component({
  moduleId: module.id,
  selector: 'app-form',
  templateUrl: 'form.component.html',
  styleUrls: ['form.component.css']
})
export class FormComponent {

  onSubmit(myForm: NgForm) {
    console.log(myForm);
  }
}
```

Lab: Form Submission

- Observe that NgForm object gets passed to the onSubmit method
  - Use console.log(myform);
- Study the following properties of NgForm Object
  - value
  - valid
  - controls
  - submitted
  - touched
  - ...

Validation

Angular recognizes HTML5 validator attrs

- You can use HTML5 native validator attributes such as
  - required
  - minlength
  - maxlength
  - pattern (email pattern http://emailregex.com/)

```
<form ...>
    <input type="text"
        class="form-control"
        id="email"
        ngModel
        name='email'
        required
        pattern="^[a-zA-Z0-9....">
```

Email regexp pattern

Form State and Angular Generated CSS Classes

Form State and CSS Classes

- Angular provides special CSS classes and associate them to the input fields automatically depending on the state of the form and input form fields
  - ng-pristine and ng-dirty
  - ng-touced and ng-untouched
  - ng-valid and ng-invalid
- For the form, it associate the following CSS classes
  - ng-pristine and ng-dirty
  - ng-valid and ng-invalid
  - ng-submitted

Lab: Form States and CSS Classes

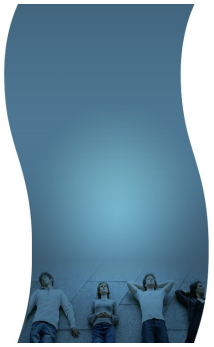
- Inspect the form and input fields
- Enter invalid data and check if ng-invalid CSS class gets added
- Use your own custom styling to these CSS classes

```
input.ng-dirty.ng-invalid {
    border: 3px red solid;
}

input.ng-dirty.ng-valid {
    border: 3px green solid;
}
```

Default Values with Property Binding

Default values can be set via Property Bindi



- In the form

```
<form ...>
    <input type="text"
        class="form-control"
        id="email"
        [ngModel]=”user.username”
        name='username'
        required
        pattern=”^[a-z0-9!#$%&'...”>
```

- In the class

```
user: User = {
  username: 'sang',
  email: 'sangshin@jpassion.com',
  password: 'xyz'
}
```

Two-way Binding with ngModel

Two-way Databinding

- In the form

```
<form ...>
    <input type="text"
        class="form-control"
        id="email"
        [(ngModel)]=”user.email”
        name='email'
        required
        pattern=”[a-z0-9!#$%&'...”>
```

- In the class - The user object now reflects the valued entered in the fields

```
onSubmit(form: NgForm){
   console.log(this.user);
}
```

ngModelGroup

ngModelGroup

- You can create a new form group object from multiple input form fields
- Use ngModelGroup=”userData”

```
<form ...>
    <div ngModelGroup=”userData” >
        <input type="text"
            class="form-control"
            id="username"
          . ...>
        <input type="text"
            class="form-control"
            id="email"
          . ...>
     <div>
```

Lab: ngModelGroup

- Observe that “userData” object now contains “username” and “email”

```
onSubmit(form: NgForm){
   console.log(form.value.userData);
}
```

Radio Control & Select Control

Radio Controls

- Create an array

```
genders = [
  'male',
  'female',
  'other'
]
```

- Then use the array in the template

```
<div class="radio" *ngFor="let gender of genders">
  <label>
    <input type="radio"
      name="gender"
      [(ngModel)] = 'user.gender'
      [value]="gender">
      {{gender}}
  </label>
</div>
```

Select Controls

- Create an array

```
hobbies = [
  'golf',
  'swimming',
  'jumping'
]
```

- Then use the array in the template

```
<div class="form-group">
  <select name="hobby" [(ngModel)]="user.hobby">
    <option *ngFor="let hobby of hobbies"
            [value]="hobby" >{{hobby}}</option>
  </select>
</div>
```

Conditional Error Message

Conditional Error message

- You can get a reference to input control that Angular manages via #email
- Then use *ngIf to display error message

```
<form ...>
    <input type="email"
        class="form-control"
        id="email"
        [(ngModel)]="user.email"
        name='email'
        required
        pattern="[a-z0-9!#$%&'..."
        #email="ngModel">

<div *ngIf="!email.valid">
  Email is invalid!
<div>
```

Conditionally Disabling Submit Button

Conditionally Disabling Submit Button

```
  ...
  <button type="submit" class="btn btn-primary"
          [disabled]="myform.invalid">Submit</button>

</form>
```

Code with Passion!
JPassion.com