ANGULAR 2

FORMS
DIRECTIVES

FORM WITH VALIDATION

```
<form #heroForm ="ngForm"  *ngIf="active"  (ngSubmit)="onSubmit()">
   <div class="form-group”> <label for="power">Hero Power</label>
      <select #power ="ngModel" class="form-control” name="power"
            [(ngModel)]="hero.power" required >
         <option *ngFor="let p of powers" [value]="p">{{p}}</option>
      </select>
      <div *ngIf="power.errors && power.touched" class="alert alert-danger">
         <div [hidden]="!power.errors.required">Power is required</div>
      </div>
   </div>
   <button type="submit" class="btn btn-default"
         [disabled]="!heroForm.form.valid">Submit</button>
</form>
```

```
@Component({selector: 'hero-form', templateUrl: 'hero-form.html'})
export class HeroFormComponent {
   powers = ['Really Smart', 'Super Flexible', 'Weather Changer'];
   hero = new Hero(18, 'Dr. WhatIsHisWayTooLongName', this.powers[0], 'Dr. What');
   onSubmit() {
              this.heroService.saveHero(this.hero).subscribe(res=>
                     router.navigateByUrl("/"))
   }
}
```

VALIDATION: BUILT IN VALIDATORS

- **required** - Requires a form control to have a non-empty value
- **minlength** - Requires a form control to have a value of a minimum length
- **maxlength** - Requires a form control to have a value of a maximum length
- **pattern** - Requires a form control’s value to match a given regex

```
<form novalidate>
   <input type="text" name="name" ngModel required>
   <input type="text" name="street" ngModel minlength="3">
   <input type="text" name="city" ngModel maxlength="10">
   <input type="text" name="zip" ngModel pattern="[A-Za-z]{5}">
</form>
```

FORM VALIDATION RESULTS

State	Class if true	Class if false
Control has been visited	ng-touched	ng-untouched
Control's value has changed	ng-dirty	ng-pristine
Control's value is valid	ng-valid	ng-invalid

```
.ng-valid[required] {
  border-left: 5px solid #42A948; /* green */
}
.ng-invalid.ng-touched {
  border-left: 5px solid #a94442; /* red */
}
```

SHOW VALIDATION MESSAGES

```
<input type="text" class="form-control" required
  [(ngModel)]="model.name"
    name="name"  #name>
<div [hidden]="name.valid" class="alert alert-danger">
  Name is required
</div>
```

INPUT FIELD AND VALIDATION MESSAGES

```
<input type="text" id="name" class="form-control"
      required minlength="4" maxlength="24“
      name="name" [(ngModel)]="hero.name"
      #name="ngModel" >

<div *ngIf="name.errors && (name.dirty || name.touched)"
     class="alert alert-danger">
   <div [hidden]="!name.errors.required">
      Name is required
   </div>
   <div [hidden]="!name.errors.minlength">
      Name must be at least 4 characters long.
   </div>
   <div [hidden]="!name.errors.maxlength">
      Name cannot be more than 24 characters long.
   </div>
</div>
```

DRAWBACKS OF VALIDATION MESSAGES IN THE CODE

- It takes a lot of HTML to represent all possible error conditions. This gets out of hand when there are many controls and many validation rules.

- We're not fond of so much JavaScript logic in HTML.

- The messages are static strings, hard-coded into the template. We often require dynamic messages that we should shape in code.

We can move the logic and the messages into the component with a few changes to the template and component.

VALIDATION RESULTS IN COMPONENT CODE

```
<form #heroForm="ngForm"  *ngIf="active"  (ngSubmit)="onSubmit()">
  <input type="text" id="name" class="form-control"
      required minlength="4" maxlength="24" forbiddenName="bob"
      name="name" [(ngModel)]="hero.name" >

  <div *ngIf="formErrors.name" class="alert alert-danger">
      {{ formErrors.name }}
  </div>
</form>

@ViewChild('heroForm') heroForm: NgForm;

ngOnInit() {
        this.heroForm.valueChanges
          .subscribe(data => this.onValueChanged(data));
}
```

FILL VALIDATION MESSAGES FROM CODE

```
onValueChanged(data?: any) {
    if (!this.heroForm) { return; }
    const form = this.heroForm.form;

    for (const field in this.formErrors) {
      // clear previous error message (if any)
      this.formErrors[field] = '';
      const control = form.get(field);

      if (control && control.dirty && !control.valid) {
        const messages = this.validationMessages[field];
        for (const key in control.errors) {
            this.formErrors[field] += messages[key] + ' ';
        }
      }
    }
}
```

```
validationMessages = {
'name': {
  'required':      'Name is required.',
  'minlength':     `Name must be
         at least 4 characters long.`,
   'maxlength':     `Name cannot be
         more than 24 characters long.`
    },
'power': {
    'required': 'Power is required.'
    }
};
```

```
formErrors = {
    'name': '',
    'power': ''
};
```

```
<div *ngIf="formErrors.name" >
     {{ formErrors.name }}
 </div>
```

REACTIVE FORM

```
<form novalidate [formGroup]="form">
   <input type="text" name="name" ngModel >
   <input type="text" name="street" ngModel >
   <input type="text" name="city" ngModel >
   <input type="text" name="zip" ngModel >
</form>

@Component()
class Cmp {
 constructor(private fb: FormBuilder) {}
 ngOnInit() {
   this.form = this.fb.group({
      name: ['', Validators.required],
      street: ['', Validators.minLength(3)],
      city: ['', Validators.maxLength(10)],
      zip: ['', Validators.pattern('[A-Za-z]{5}')]
   });
}}
```

Reactive form benefits: you can

- add, change, and remove validation functions on the fly
- manipulate the control model dynamically from within the component
- test validation and control logic with isolated unit tests.

IMPORT REACTIVE FORMS MODULE

```
@NgModule({
   imports:      [ SharedModule, ReactiveFormsModule ],
   declarations: [ HeroFormReactiveComponent ],
   exports:      [ HeroFormReactiveComponent ]
})
export class HeroFormReactiveModule { }
```

CUSTOM VALIDATION FUNCTION

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): {[key: string]: any} => {
    const name = control.value;
    const no = nameRe.test(name);
    return no ? {'forbiddenName': {name}} : null;
  };
}
```

Use in reactive form:

```
'name': [this.hero.name, [
  Validators.required,
  Validators.minLength(4),
  Validators.maxLength(24),
  forbiddenNameValidator(/bob/i)
 ]
],
```

CUSTOM VALIDATOR DIRECTIVE

```
@Directive({
  selector: '[startWith]',
  providers: [{provide: NG_VALIDATORS, useExisting: CustomValidatorDirective,
     multi: true}]
})
export class CustomValidatorDirective implements Validator{
  @Input('startWith') expr: string;

  validate(control: AbstractControl) {
    if(control.value && !control.value.startsWith(this.expr)){
      return {'startWith': control.value};
    }
    return null;
  }
}
```

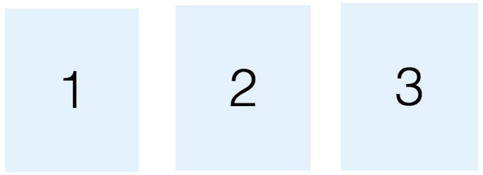
```
<input [startWith]="a" ngModel name=“name”>
```

ATTRIBUTE DIRECTIVE

```
<p myHighlight>Highlight me red</p>

<p [myHighlight]="color“ [size]=2>Highlight
me!</p>
```

```
@Directive({
   selector: '[myHighlight]'
})
export class HighlightDirective {
   private _defaultColor = 'red';
   constructor(private el: ElementRef) { }
   @Input('myHighlight') highlightColor: string;
   @Input('size') size: number;

   @HostListener('mouseenter') onMouseEnter() {
      this.highlight(this.highlightColor || this._defaultColor);
   }
   @HostListener('mouseleave') onMouseLeave() {
      this.highlight(null);
   }

   private highlight(color: string) {
      this.el.nativeElement.style.backgroundColor = color;
   }
}
```

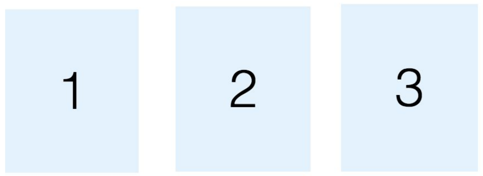
STRUCTURAL DIRECTIVE



```
@Directive({
  selector: '[delay]'
})
export class DelayDirective {
  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainerRef: ViewContainerRef
  ) { }

  @Input('delay')
  set delayTime(time: number): void {
    setTimeout(()=>{
      this.viewContainerRef
        .createEmbeddedView(
          this.templateRef);
    }, time);
  }
```

```
@Component({
  selector: 'app',
  template: `
  <div *ngFor="let item of [1,2,3]">
    <card *delay="500 * item">
        {{item}}
    </card>
   </template>
  </div>
 `
})
export class AppComponent {
}
@Component({
  selector: 'card',
  template: `
    <ng-content></ng-content>`})
export class CardComponent {}
```

STRUCTURAL DIRECTIVE



```
@Directive({
  selector: '[delay]'
})
export class DelayDirective {
  constructor(
    private templateRef: TemplateRef<any>,
    private viewContainerRef: ViewContainerRef
  ) { }

  @Input('delay')
  set delayTime(time: number): void {
    setTimeout(()=>{
      this.viewContainerRef
        .createEmbeddedView(
          this.templateRef);
    }, time);
  }
```

```
@Component({
  selector: 'app',
  template: `
  <div *ngFor="let item of [1,2,3]">
   <template [delay]="500 * item">
    <card >
     {{item}}
    </card>
   </template>
  </div>
 `
})
export class AppComponent {
}

@Component({
  selector: 'card',
  template: `
     <ng-content></ng-content>`})
export class CardComponent {}
```