Lesson 6

Forms

Agenda

- User input events
- Template-driven forms
- Reactive forms (dynamic forms)

User input events

```
@Component({
  selector: 'key-up1',
  template: `
    <input (keyup)="onKey($event)">
    <p>{{values}}</p>
  `
})
```

```
export class KeyUpComponent_v1 {
  values = '';

  /*
  onKey(event: any) { // without type info
    this.values += event.target.value + ' | ';
  }
  */

  onKey(event: KeyboardEvent) { // with type info
    this.values += (<HTMLInputElement>event.target).value + ' | ';
  }
}
```

User input events

```
@Component({
  selector: 'key-up3',
  template: `
    <input #box (keyup.enter)="onEnter(box.value)">
    <p>{{value}}</p>
  `
})
export class KeyUpComponent_v3 {
  value = '';
  onEnter(value: string) { this.value = value; }
}
```

Approaches to create forms

- Template-driven forms
- Reactive forms

Template-driven forms

- Requires FormsModule
- Works with POCO models
- NgForm directive is added automatically
- Inputs, bindings, validation are described in template

Template-driven forms. Register Module

```
import { NgModule }      from '@angular/core';
import { BrowserModule } from '@angular/platform-browser';
import { FormsModule }   from '@angular/forms';

import { AppComponent }  from './app.component';
import { HeroFormComponent } from './hero-form.component';

@NgModule({
  imports: [
    BrowserModule,
    FormsModule
  ],
  declarations: [
    AppComponent,
    HeroFormComponent
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Template-driven forms. Create model

```
export class Hero {

  constructor(
    public id: number,
    public name: string,
    public power: string,
    public alterEgo?: string
  ) {  }

}
```

Template-driven forms. Create template

```
<form (ngSubmit)="onSubmit()" #heroForm="ngForm">
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" class="form-control" id="name"
           required
           [(ngModel)]="model.name" name="name"
           #name="ngModel">
    <div [hidden]="name.valid || name.pristine"
         class="alert alert-danger">
      Name is required
    </div>
  </div>
```

...

```
    <button type="submit" class="btn btn-success"
[disabled]="!heroForm.form.valid">Submit</button>
    <button type="button" class="btn btn-default" (click)="newHero();
heroForm.reset()">New Hero</button>
  </form>
```

Template-driven forms. Control states

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

Template-driven forms. Validation

```
<input id="name" name="name" class="form-control"
       required minlength="4" forbiddenName="bob"
       [(ngModel)]="hero.name" #name="ngModel" >

<div *ngIf="name.invalid && (name.dirty || name.touched)"
     class="alert alert-danger">

  <div *ngIf="name.errors.required">
    Name is required.
  </div>
  <div *ngIf="name.errors.minlength">
    Name must be at least 4 characters long.
  </div>
  <div *ngIf="name.errors.forbiddenName">
    Name cannot be Bob.
  </div>
```

Template-driven forms. Custom validation

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): {[key: string]: any} => {
    const forbidden = nameRe.test(control.value);
    return forbidden ? {'forbiddenName': {value: control.value}} : null;
  };
}
@Directive({
  selector: '[forbiddenName]',
  providers: [{provide: NG_VALIDATORS, useExisting: ForbiddenValidatorDirective, multi: true}]
})
export class ForbiddenValidatorDirective implements Validator {
  @Input() forbiddenName: string;

  validate(control: AbstractControl): {[key: string]: any} {
    return this.forbiddenName ? forbiddenNameValidator(new RegExp(this.forbiddenName, 'i'))(control)
                              : null;
  }
}
```

Reactive forms

- Requires ReactiveFormsModule
- Works with FormBuilder groups
- FormGroup is applied manually, controls html is generated
- Inputs, bindings, validation are described in form group

Reactive forms. Register module

```
import { NgModule }            from '@angular/core';
import { BrowserModule }       from '@angular/platform-browser';
import { ReactiveFormsModule } from '@angular/forms';  // <-- #1 import module

import { AppComponent }        from './app.component';
import { HeroDetailComponent } from './hero-detail.component'; // <-- #1 import component

@NgModule({
  imports: [
    BrowserModule,
    ReactiveFormsModule // <-- #2 add to @NgModule imports
  ],
  declarations: [
    AppComponent,
    HeroDetailComponent, // <-- #3 declare app component
  ],
  bootstrap: [ AppComponent ]
})
export class AppModule { }
```

Reactive forms. Add form group

```
import { Component }             from '@angular/core';
import { FormControl, FormGroup } from '@angular/forms';
```

```
export class HeroDetailComponent2 {
  heroForm = new FormGroup ({
    name: new FormControl()
  });
}
```

```
<form [formGroup]="heroForm" novalidate>
  <div class="form-group">
    <label class="center-block">Name:
      <input class="form-control" formControlName="name">
    </label>
  </div>
</form>
```

Reactive forms. Form builder. Validation

```
import { Component }                          from '@angular/core';
import { FormBuilder, FormGroup, Validators } from '@angular/forms';
```

```
createForm() {
  this.heroForm = this.fb.group({
    name: ['', Validators.required ],
    street: '',
    city: '',
    state: '',
    zip: '',
    power: '',
    sidekick: ''
  });
}
```

Reactive forms. FormControl

```
<p>Street value: {{ heroForm.get('address.street').value}}</p>
```

Property	Description
`myControl.value`	the value of a `FormControl` .
`myControl.status`	the validity of a `FormControl` . Possible values: `VALID` , `INVALID` , `PENDING` , or `DISABLED` .
`myControl.pristine`	`true` if the user has *not* changed the value in the UI. Its opposite is `myControl.dirty` .
`myControl.untouched`	`true` if the control user has not yet entered the HTML control and triggered its blur event. Its opposite is `myControl.touched` .

Reactive forms. Custom validation

```
export function forbiddenNameValidator(nameRe: RegExp): ValidatorFn {
  return (control: AbstractControl): {[key: string]: any} => {
    const forbidden = nameRe.test(control.value);
    return forbidden ? {'forbiddenName': {value: control.value}} : null;
  };
}
```

```
this.heroForm = new FormGroup({
  'name': new FormControl(this.hero.name, [
    Validators.required,
    Validators.minLength(4),
    forbiddenNameValidator(/bob/i) // <-- Here's how you pass in the custom validator.
  ]),
  'alterEgo': new FormControl(this.hero.alterEgo),
  'power': new FormControl(this.hero.power, Validators.required)
});
```