Forms Design & Validaiton

Agenda

- Forms Introduction
- Template Driven Froms
- Reactive (Model Based) Froms
- Validating Forms
- Custom & Code Based Validation

Forms Introduction

Angular Forms

- Angular provides two Form Types
- Template Driven Froms
  - Rely on FormsModule imported from '@angular/forms'
- Reactive (Model bound) Forms
  - Rely on ReactiveFormsModule imported from '@angular/forms'
- Reactive Forms are easier to implement and more powerful ie. Testable

Template Driven Froms

Template Driven Froms

+ Easy to implement
+ Good for simple Forms
- Lot of logic in html
- Bad for cross field validation
- Not Unit Testable

Form Setup

- Template Driven Forms require FormModule
- ngForm – Adds Form Functionality to a Form using Local Ref
- ngSubmit – Handles Form Submission in Angular instead classic HTML Submit
- novalidate – Supresses HTML validation

```
<form #personForm="ngForm" (ngSubmit)="savePerson(personForm.value)" role="form" novalidate>
```

ngForm

- Adds additional Form Functionality to a Form like
  - controls collection {[key: string]}
  - addControl() / removeControl()
  - setValue()
  - onSubmit(), onReset()
  - ....


```
▼ NgForm {_submitted: true, ngSubmit: EventEmitter, form: FormGroup}
    control: (...)
  ▼ controls: Object
    ▶ personAge: FormControl {asyncValidator: null, _pristine: true, _touc
    ▶ personGender: FormControl {asyncValidator: null, _pristine: true, _t
    ▶ personName: FormControl {asyncValidator: null, _pristine: true, _tou
    ▶ __proto__: Object
    dirty: (...)
```

```
<form #personForm="ngForm" (ngSubmit)="savePerson(personForm.value)" role="form" novalidate>
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" class="form-control" id="name" name="personName" placeholder="Enter name" [(ngModel)]="person.name">
  </div>
  <div class="form-group">
    <label for="age">Age</label>
    <input type="number" class="form-control" id="age" name="personAge" placeholder="Enter age" [(ngModel)]="person.age">
  </div>
  <div class="form-group">
    <label>Gender</label><br/>
    <input type="radio" value="male" name="personGender" [(ngModel)]="person.gender" > Male
    <input type="radio" value="female" name="personGender" [(ngModel)]="person.gender" > Female
  </div>
  <div>
    <button type="submit" class="btn btn-primary">Submit</button>
  </div>
</form>
```

<input type="..">

- Inputs typically have the following related elements
  - <label> & id attr – work together
  - name – submitted key for field
  - ngModel – Databinding
- Can have:
  - HTML validation
  - Placeholder
  - Angular Validation
  - CSS

```
saving person with values:
▼ {personName: "Heinz", personAge: 12, personGender: "male"}
    personAge: 12
    personGender: "male"
    personName: "Heinz"
  ▶ __proto__: Object
```

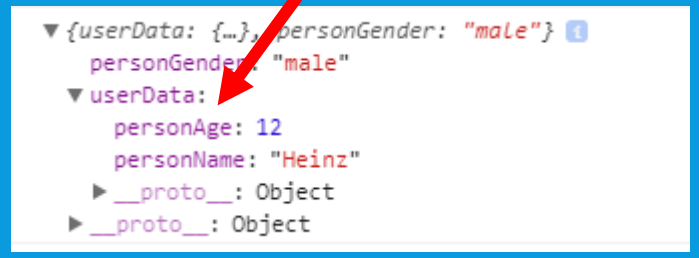
```
export class TemplateDrivenComponent implements OnInit {

  person : Person = {name: "Heinz", gender: "male", age: 12};
```

```
<div class="form-group">
<label for="name">Name</label>
<input type="text" class="form-control"
 id="name" name="personName" placeholder="Enter name" [(ngModel)]="person.name">
</div>
```

Grouping Form Controls

- Use ngModelGroup to structure data in large forms

```
<div id="user-data" ngModelGroup="userData">
<div class="form-group">
<label for="name">Name</label>
<input type="text" class="form-control" id="name" name="personName" placeholder="Enter name"
[(ngModel)]="person.name">
</div>
<div class="form-group">
<label for="age">Age</label>
<input type="number" class="form-control" id="age" name="personAge" placeholder="Enter age"
[(ngModel)]="person.age">
</div>
</div>
```



Radio, Select, Checkbox

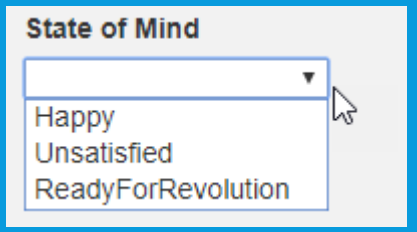
- Build using *ngIf – assign same name attr

```
export class TemplateDrivenComponent implements OnInit {

person : Person = <Person> {name: "Heinz", gender: "male", age: 12, wealth: "poor", …};
wealth = ['poor', 'rich', 'middle class'];
```

```
<div class="form-group">
<label>Wealth Radio Buttons</label><br/>
<div *ngFor="let rw of wealth">
   <input type="radio" value="rw" name="personGenderRB" [(ngModel)]="person.gender" > {{rw}}
</div>
</div>
```

Using Enums in Forms

- Get Enum values using Object.keys(ENUM)
- Object.key contrains all keys AND all values -> use splice()
- Otherwiese use like Radios, Select, …



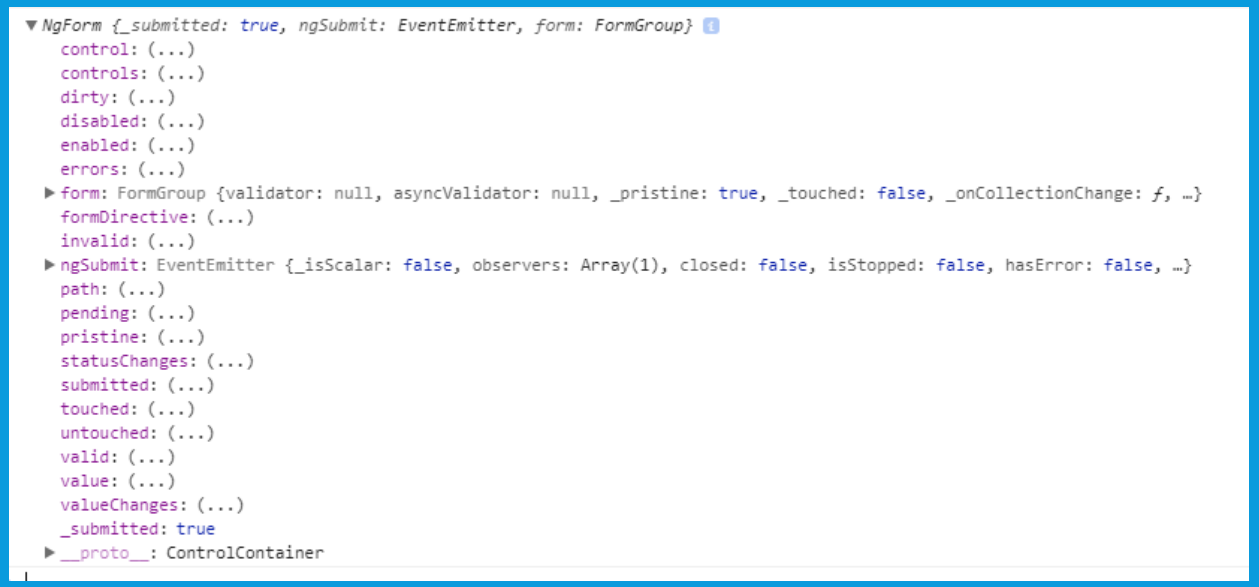

```
<select name="personState" [(ngModel)]="person.state">
<option *ngFor="let s of states" [ngValue]="s">{{s}}</option>
</select>
```

```
export enum WorkLifeBalance {
Happy = 0,
Unsatisfied = 1,
ReadyForRevolution = 2
}

export interface Person {
name: string;
gender: string;
age: number;
mail: string;
wealth?: string;
state?: WorkLifeBalance
}
```

```
states : string [] =
Object.keys(WorkLifeBalance).slice(Object.keys(WorkLifeBalance).length / 2);
```

Submitting

- Choose to
  - Just the data: personForm.value
  - Submit whole form: personForm



Reactive Forms

Reactive Forms

- Reactive forms are synchronous. Template-driven forms are asynchronous.
- Depends on ReativeFormsComponent import in Module
- Uses FormGroup, FormControl, FormArray elements
- FormGroups are used to Group FormControls

Initializing

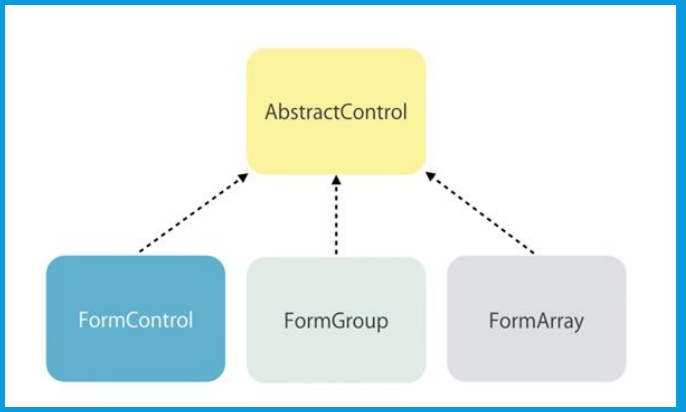
Reactive Forms do NOT CONTAIN ngModel & name attributes

```
personForm: FormGroup;
personName: FormControl;
personAge: FormControl;
personGender: FormControl;

person : Person = {name: "Heinz", gender: "male", age: 12};

constructor() { }

ngOnInit(){
  this.personName = new FormControl(this.person.name);
  this.personAge = new FormControl(this.person.age);
  this.personGender = new FormControl(this.person.gender);

  this.personForm = new FormGroup({
    pName: this.personName,
    pAge: this.personAge,
    pGender: this.personGender
  })
}
```

```
<form [formGroup]="personForm" (ngSubmit)="savePerson(personForm.value)" novalidate>
  <div class="form-group">
    <label for="name">Name</label>
    <input type="text" class="form-control" placeholder="Enter name"
     formControlName="pName" id="name" name="personName" >
  </div>
  <div class="form-group">
    <label for="age">Age</label>
    <input type="number" class="form-control" placeholder="Enter age"
     formControlName="pAge" id="age" name="personAge">
  </div>
  <div class="form-group">
    <label>Gender</label><br/>
    <input type="radio" value="male" id="gender" formControlName="pGender"> Male
    <input type="radio" value="female" id="gender" formControlName="pGender"> Female
  </div>
```

FormArray

- Helper Class that allows us to explicitly declare forms in our components
- Tracks the value and validity state of an array of FormControl / FormGroup / FormArray instances



Forms Builder

- Helper Class that allows us to explicitly declare forms in our components
- Makes reacitve Form building much more straight forward

```
export class FormsBuilderComponent implements OnInit {

personForm: FormGroup;
person : Person = {name: "Heinz", gender: "male", age: 12, mail: "derschoeneheinz@xyz.at"};
constructor(private fb: FormBuilder) {}

ngOnInit() {
 this.personForm = this.fb.group({
  personName: [this.person.name, Validators.required],
  personAge: [this.person.age],
  personGender: [this.person.gender]
 })
}
```

Validation

HTML Validation

- HTML5 provides input types that expect data in a specific format,
- You can also apply your own custom rules to many input fields by using a regular expression

```
Age: <input type="number" size="6" name="age" min="18" max="99" value="21">
Website: <input type="url" name="website" required placeholder="Insert URL">

//URL
<input type="url" pattern="https?://.+">
//IPv4 Address
<input type="text" pattern="\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}" />
//Price
<input type="text" pattern="\d+(\.\d{2})?"/>
//Long /Lat
<input type="text" pattern="-?\d{1,3}\.\d+"/>
```

```
input {
    border: solid 1px;
}
input:invalid {
    border-color: #f00;
}
input:valid {
    border-color: #0f0;
}
```

Angular Validation

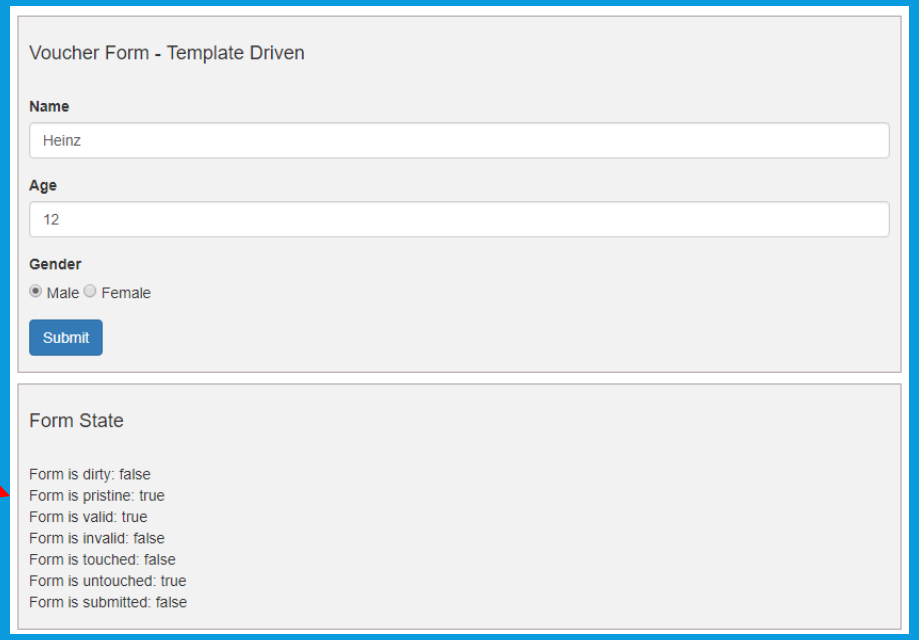
- Typically you want Angular to take over validation instead on HTML validation
- HTML Validation can be disabled using a novalidate (HTML 5) attribute
- We want Validaiton done by Angular – not HTML!

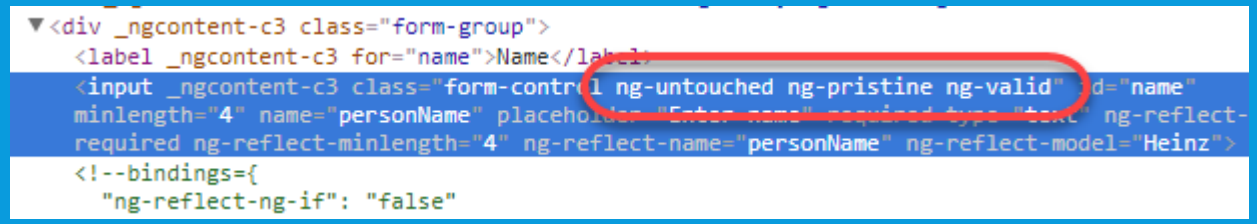
```
class Validators {
  static min(min: number): ValidatorFn
  static max(max: number): ValidatorFn
  static required(control: AbstractControl): ValidationErrors|null
  static requiredTrue(control: AbstractControl): ValidationErrors|null
  static email(control: AbstractControl): ValidationErrors|null
  static minLength(minLength: number): ValidatorFn
  static maxLength(maxLength: number): ValidatorFn
  static pattern(pattern: string|RegExp): ValidatorFn
  static nullValidator(c: AbstractControl): ValidationErrors|null
  static compose(validators: (ValidatorFn|null|undefined)[]|null): ValidatorFn|null
  static composeAsync(validators: (AsyncValidatorFn|null)[]): AsyncValidatorFn|null
}
```

Form | Control State

- Informs us about the current State of a Form | Control

```
<div class="section">
<h4>Form State</h4><br>
Form is dirty: {{personForm.dirty}}<br>
Form is pristine: {{personForm.pristine}}<br>
Form is valid: {{personForm.valid}}<br>
Form is invalid: {{personForm.invalid}}<br>
Form is touched: {{personForm.touched}}<br>
Form is untouched: {{personForm.untouched}}<br>
Form is submitted: {{personForm.submitted}}<br>
</div>
```



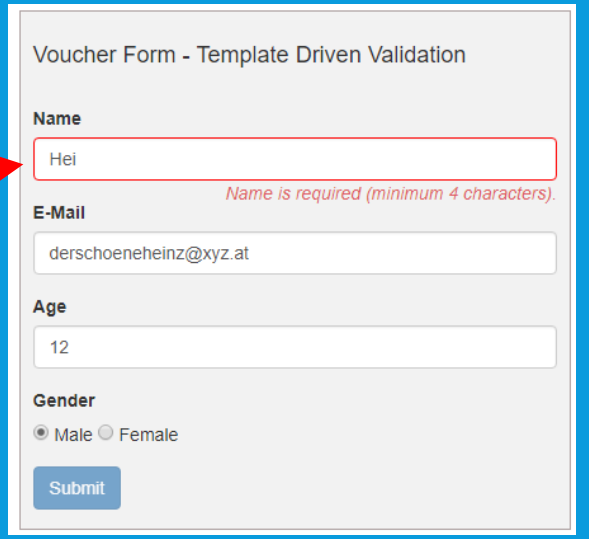


Validating Template Based Forms

- Access Form Controls using LokalRef.controls["Field"]
- To get a reference in code use @ViewChild

```
<div class="form-group">
<label for="name">Name</label>
<input type="text" class="form-control" placeholder="Enter name" id="name"
   name="personName" [(ngModel)]="person.name" required minlength="4">
<em *ngIf="personForm.controls['personName']?.invalid">
   Name is required (minimum 4 characters).
</em>
</div>
```

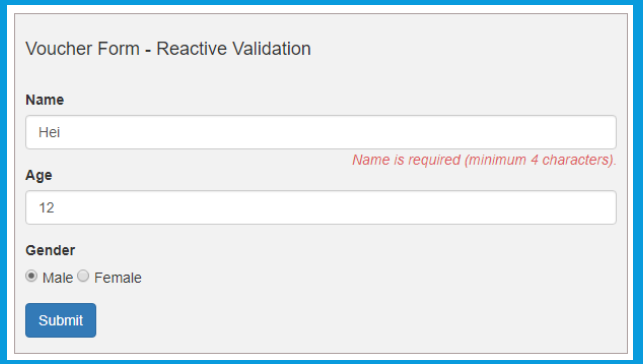
```
export class TemplateValidationComponent implements OnInit {
@ViewChild('personForm') form: NgForm;

savePerson(personForm):void {
console.log("Current personForm using ViewChild: ")
console.log(this.form);
console.log(this.form.controls["personName"].value)
```




```
Current personForm using ViewChild:
▶ NgForm {_submitted: true, ngSubmit: EventEmitter, form: FormGroup}
Heinz
```

Validating Reactive Forms

- Does not validate in HTML but uses Code instead
  - Get Logic out of HTML and Into code
  - More powerful & straight forward




```
<div class="form-group">
<label for="name">Name</label>
<input type="text" class="form-control" placeholder="Enter name"
formControlName="pName" id="name" name="personName" >
<em *ngIf="!validateName()"> … </em>
</div>
```

```
ngOnInit() {
 this.personName = new FormControl(this.person.name, [Validators.required, Validators.minLength(4)]);
 this.personForm = new FormGroup({pName: this.personName});}
...
validateName(){
 return this.personName.valid || this.personName.untouched
}
```

Custom Validation

Custom Validators

- Only available in Reactive Approach
- Function that returns Promis or Observable

```
validateNotHugo(control: FormControl): {[s: string]: boolean}{
if(control.value === "Hugo"){
 return {'hugoNotAllowed': true}
}
 return null;
}
```

```
ngOnInit() {
this.personName = new
FormControl(this.person.name,
[Validators.required,
Validators.minLength(4),
this.validateNotHugo],
this.validateNamesExist);
```

Async Validation

- Can be used if Validation includes an async operation
  - i.e. calling a service

```
validateNamesExist(control: FormControl): Promise<any> | Observable<any>{
//Mocking Http Call
const result = new Promise<any>((resolve, reject)=>{
setTimeout(()=>{
     if(control.value === "Alexander"){
      resolve({'UserExists': true});
     }
     else{
      resolve(null);
     }
}, 1500)
})
return result;
}
```

```
ngOnInit() {
this.personName = new
FormControl(this.person.name,
[Validators.required,
Validators.minLength(4),
this.validateNoHugo],
this.validateNamesExist);
```

Validation using Code

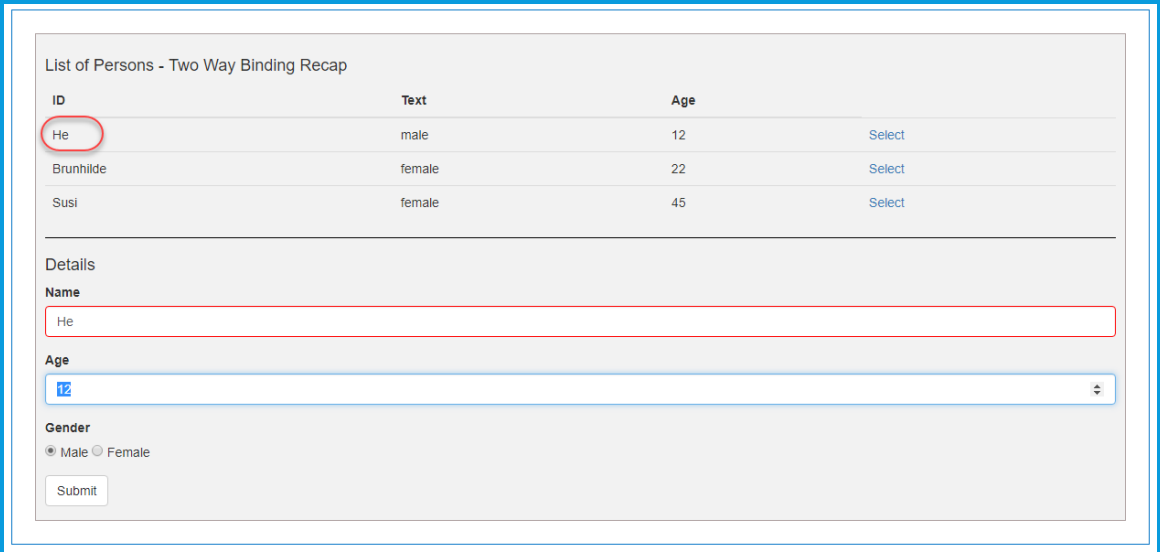
- Triggering Validation using code is easy
- Use .updateValueAndValidity() on form or control

```
<div>
  <button class="btn btn-primary" (click)="validateForm()" >Validate</button>
  <button type="submit" class="btn btn-primary">Submit</button>
</div>
```

```
validateForm(){
  this.personForm.updateValueAndValidity();
  this.personForm.controls['pName'].updateValueAndValidity();
}
```

Two Way Databinding Revisited

- Consider a scenario with two way databinding
- What happens in Master / Detail scenario with validation on person.name => 4 chars?
- Do we want to pass values from child comp. to parent comp before validation?




```
<div class="form-group">
<label for="name">Name</label>
<input type="text" class="form-control" id="name"
placeholder="Enter name"
name="name" [(ngModel)]="person.name"
required minlength="4">
</div>
```