

## Angular 2+

### Workshop. Forms.

#### Contents

Task 01. Template-Driven Form .....	2
Task 02. ReactiveComponent Class.....	6
Task 03. formGroup, FormControlName directives .....	7
Task 04. setValue(), patchValue().....	10
Task 05. FormBuilder.....	11
Task 06. Setting Built-in Validators .....	12
Task 07. Adjusting Validation Rules in Runtime.....	14
Task 08. Custom Validator .....	17
Task 09 Handle Validators Run Moment.....	19
Task 10. Custom Validator w/ Parameters.....	21
Task 11. Custom Validator Directive .....	22
Task 12. Custom Validator Directive w/ Parameters .....	24
Task 13. Async Custom Validator .....	25
Task 14. Async Custom Validator Directive.....	27
Task 15. Cross-Field Validation.....	29
Task 16. Adjusting Validation Rules .....	34
Task 16. Displaying Validation Messages .....	36
Task 17. Reactive Transformations .....	38
Task 18. Define the input element(s) to duplicate.....	39
Task 19. Define a FormGroup.....	40
Task 20. Refactor to Make Copies.....	41
Task 21. Create a FormArray .....	42
Task 22. Loop through the FormArray .....	43
Task 23. Duplicate the Input Elements.....	44

## Task 01. Template-Driven Form

1. Create file **app/models/user.model.ts**. Run the following command from a command line:

```
> ng g cl models/user --type model
```

2. Replace the content of the file with the following content:

```
export class UserModel {  
  
  constructor(public firstName = '',  
              public lastName = '',  
              public email = '',  
              public sendProducts = false,  
              public addressType = 'home',  
              public street1?: string,  
              public street2?: string,  
              public country = '',  
              public city?: string,  
              public zip?: string) { }  
  
}
```

3. Make changes to **SignupFormComponent**. Use the following snippet of code:

```
import { Component, OnInit } from '@angular/core';  
import { NgForm } from '@angular/forms';  
import { UserModel } from '../models/user.model';
```

4. Add properties:

```
countries: Array<string> = ['Ukraine', 'Armenia', 'Belarus', 'Hungary', 'Kazakhstan',  
                             'Poland', 'Russia'];  
user: UserModel = new UserModel();
```

5. Add method:

```
onSave(signupForm: NgForm) {  
  // Form model  
  console.log(signupForm.form);  
  // Form value  
  console.log(`Saved: ${JSON.stringify(signupForm.value)}`);  
}
```

6. Make changes to **SignupFormComponent template**. Use the following snippet of HTML:

```
// 1  
<form class="form-horizontal"  
      #signupForm="ngForm"  
      (ngSubmit)="onSave(signupForm)">  
  
// 2  
<div class="form-group"  
      [ngClass]="{'has-error': (firstNameVar.touched || firstNameVar.dirty) &&  
      firstNameVar.invalid }">  
  
<input class="form-control"  
      id="firstNameId"  
      type="text"
```

```

        placeholder="First Name (required)"
        required
        minlength="3"
        [(ngModel)]="user.firstName"
        name="firstName"
        #firstNameVar="ngModel" />
<span class="help-block" *ngIf="(firstNameVar.touched || firstNameVar.dirty) &&
firstNameVar.errors">
    <span *ngIf="firstNameVar.hasError('required')">
        Please enter your first name.
    </span>
    <span *ngIf="firstNameVar.hasError('minlength')">
        The first name must be longer than 3 characters.
    </span>
</span>

// 3
<div class="form-group"
    [ngClass]="{'has-error': (lastNameVar.touched || lastNameVar.dirty) &&
lastNameVar.invalid }">

    <input class="form-control"
        id="lastNameId"
        type="text"
        placeholder="Last Name (required)"
        required
        maxlength="50"
        [(ngModel)]="user.lastName"
        name="lastName"
        #lastNameVar="ngModel" />
    <span class="help-block" *ngIf="(lastNameVar.touched || lastNameVar.dirty) &&
lastNameVar.errors">
        <span *ngIf="lastNameVar.hasError('required')">
            Please enter your last name.
        </span>
    </span>
</div>

// 4
<div class="form-group"
    [ngClass]="{'has-error': (emailVar.touched || emailVar.dirty) && emailVar.invalid
}">

    <input class="form-control"
        id="emailId"
        type="email"
        placeholder="Email (required)"
        required
        pattern="[a-z0-9._%+-]+@[a-z0-9.-]+"
        [(ngModel)]="user.email"
        name="email"
        #emailVar="ngModel" />
    <span class="help-block" *ngIf="(emailVar.touched || emailVar.dirty) &&
emailVar.errors">
        <span *ngIf="emailVar.hasError('required')">

```

```

        Please enter your email address.
    </span>
    <span *ngIf="emailVar.hasError('pattern')">
        Please enter a valid email address.
    </span>
</span>

// 5
<input id="sendProductsId"
    type="checkbox"
    [(ngModel)]="user.sendProducts"
    name="sendProducts" >

// 6
<div *ngIf="user.sendProducts">
    <div class="form-group" >
        <label class="col-md-2 control-label">Address Type</label>

// 7
<input type="radio" id="addressType1Id" value="home"
    [(ngModel)]="user.addressType"
    name="addressType">Home

<input type="radio" id="addressType2Id" value="work"
    [(ngModel)]="user.addressType"
    name="addressType">Work

<input type="radio" id="addressType3Id" value="other"
    [(ngModel)]="user.addressType"
    name="addressType">Other

// 8
<select class="form-control"
    id="countryId"
    [(ngModel)]="user.country"
    name="country">
        <option value="">Select a Country...</option>
        <option *ngFor="let country of countries"
            [value]="country">{{country}}</option>
</select>

// 9
<input type="text"
    class="form-control"
    id="cityId"
    placeholder="City"
    [(ngModel)]="user.city"
    name="city">

<input type="number"
    class="form-control"
    id="zipId"
    placeholder="Zip Code"

```

```

        [(ngModel)]="user.zip"
        name="zip">

<input type="text"
        class="form-control"
        id="street1Id"
        placeholder="Street address"
        [(ngModel)]="user.street1"
        name="street1">

<input type="text"
        class="form-control"
        id="street2Id"
        placeholder="Street address (second line)"
        [(ngModel)]="user.street2"
        name="street2">

// 10
<button class="btn btn-primary"
        type="submit"
        [(disabled)]="!signupForm.valid">
    Save
</button>

```

7. Add the following snippet of code at the end of template

```

<br>Dirty: {{ signupForm.dirty }}
<br>Touched: {{ signupForm.touched }}
<br>Valid: {{ signupForm.valid }}
<br>Value: {{ signupForm.value | json }}

```

8. Make changes to the file **signup-form.component.css**. Use the following snippet of code:

```

input.ng-valid.ng-touched{
    background: lightgreen;
}

```

## Task 02. ReactiveComponent Class

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
import { FormGroup, FormControl } from '@angular/forms';
import { UserModel } from '../models/user.model';

countries: Array<string> = ['Ukraine', 'Armenia', 'Belarus', 'Hungary', 'Kazakhstan',
'Poland', 'Russia'];
// data model
user: UserModel = new UserModel();

// form model
userForm: FormGroup;

private createForm() {
  this.userForm = new FormGroup({
    firstName: new FormControl(),
    lastName: new FormControl(),
    email: new FormControl(),
    sendProducts: new FormControl(true)
  });
}

ngOnInit() {
  this.createForm();
}

onSave() {
  // Form model
  console.log(this.userForm);
  // Form value w/o disabled controls
  console.log(`Saved: ${JSON.stringify(this.userForm.value)}`);
  // Form value w/ disabled controls
  console.log(`Saved: ${JSON.stringify(this.userForm.getRawValue())}`);
}
```

2. Make changes to **SignupReactiveFormComponent Template**. Use the following snippet of html:

```
<select class="form-control"
id="countryId">
  <option value="">Select a Country...</option>
  <option *ngFor="let country of countries"
    value="{{country}}">{{country}}
  </option>
</select>
```

## Task 03. formGroup, FormControlName directives

1. Make changes to **SignupReactiveFormComponent** template. Use the following snippet of HTML:

```
<form class="form-horizontal"
      (ngSubmit)="onSave()"
      [formGroup]="userForm">

  <button class="btn btn-primary"
          type="submit"
          [disabled]="!userForm.valid">
    Save
  </button>

  <br>Dirty: {{ userForm.dirty }}
  <br>Touched: {{ userForm.touched }}
  <br>Valid: {{ userForm.valid }}
  <br>Value: {{ userForm.value | json }}
  <br>Form Status: {{userForm.status }}

  <input class="form-control"
         id="firstNameId"
         type="text"
         placeholder="First Name (required)"
         required
         minlength="3"
         FormControlName="firstName"/>

  <input class="form-control"
         id="lastNameId"
         type="text"
         placeholder="Last Name (required)"
         required
         maxlength="50"
         FormControlName="lastName"/>

  <input class="form-control"
         id="emailId"
         type="email"
         placeholder="Email (required)"
         required
         pattern="[a-z0-9._%+-]+@[a-z0-9.-.]+\"
         FormControlName="email" />

  <label>
    <input id="sendProductsId"
          type="checkbox"
          FormControlName="sendProducts" />
    Send me your products
  </label>
```

2. Comment template from

```

<div>
  <div class="form-group" >
    <label class="col-md-2 control-label">Address Type</label>

```

to the closed tag

3. Add attribute [ngClass] for div with class "form-group"

For firstName

```

[ngClass]="{'has-error': (userForm.get('firstName').touched ||
userForm.get('firstName').dirty) && !userForm.get('firstName').valid }"

```

For lastName

```

[ngClass]="{'has-error': (userForm.get('lastName').touched ||
userForm.get('lastName').dirty) && !userForm.get('lastName').valid }"

```

For email

```

[ngClass]="{'has-error': (userForm.get('email').touched || userForm.get('email').dirty)
&& !userForm.get('email').valid }"

```

4. Add span block after each input – block for displaying validation error messages

For firstName

```

<span class="help-block" *ngIf="(userForm.get('firstName').touched ||
userForm.get('firstName').dirty) && userForm.get('firstName').errors">
  <span *ngIf="userForm.get('firstName').hasError('required')">
    Please enter your first name.
  </span>
  <span *ngIf="userForm.get('firstName').hasError('minlength')">
    The first name must be longer than 3 characters.
  </span>
</span>

```

For lastName

```

<span class="help-block" *ngIf="(userForm.get('lastName').touched ||
userForm.get('lastName').dirty) && userForm.get('lastName').errors">
<span *ngIf="userForm.get('lastName').hasError('required')">
  Please enter your last name.
</span>
</span>

```

For email

```

<span class="help-block" *ngIf="(userForm.get('email').touched ||
userForm.get('email').dirty) && userForm.get('email').errors">
  <span
*ngIf="userForm.get('email').hasError('required')">
    Please enter your email address.
  </span>
  <span *ngIf="userForm.get('email').hasError('pattern')">
    Please enter a valid email address.
  </span>
</span>

```



```
<span *ngIf="userForm.get('email').hasError('email')">
    Please enter a valid email address.
</span>
</span>
```

## Task 04. setValue(), patchValue()

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
user: UserModel = new UserModel(
  'Vitaliy',
  'Zhyrytskyy',
  'v.zhiritskiy@gmail.com',
  false
);

// 2
private setFormValues() {
  this.userForm.setValue({
    firstName: this.user.firstName,
    lastName: this.user.lastName,
    email: this.user.email,
    sendProducts: this.user.sendProducts
  });
}

// 3
private patchFormValues() {
  this.userForm.patchValue({
    firstName: this.user.firstName,
    lastName: this.user.lastName
  });
}

// 4
ngOnInit() {
  this.createForm();
  this.setFormValues();
  // this.patchFormValues();
}
```

2. Look at the result. Comment method **this.setFormValues()**; in ngOnInit() and uncomment method **this.patchFormValues()**; Look at the result.

## Task 05. FormBuilder

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

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

constructor(
  private fb: FormBuilder
) { }

private buildForm() {
  this.userForm = this.fb.group({
    firstName: '',
    lastName: {value: 'Zhyrytskyy', disabled: true},
    email: [''],
    sendProducts: true
  });
}

ngOnInit() {
  this.createForm();
  this.buildForm();
  this.setFormValues();
  this.patchFormValues();
}
```

## Task 06. Setting Built-in Validators

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
import { FormBuilder, FormGroup, FormControl, Validators } from '@angular/forms';

// 2
private buildForm() {
  this.userForm = this.fb.group({
    firstName: '',
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: {value: 'Zhyrytskyy', disabled: true},
    email: [''],
    sendProducts: true
  });
}
```

2. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<input class="form-control"
  id="firstNameId"
  type="text"
  placeholder="First Name (required)"
  required
  minlength="3"
  formControlName="firstName"/>
```

3. Check the validation for the field First Name

4. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
private buildForm() {
  this.userForm = this.fb.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: {value: 'Zhyrytskyy', disabled: true},
    lastName: [
      { value: 'Zhyrytskyy', disabled: false },
      [Validators.required, Validators.maxLength(50)]
    ],
    email: [''],
    email: [
      '',
      [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
      Validators.email
    ],
    sendProducts: true
  });
}
```

5. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<input class="form-control"
  id="lastNameId"
  type="text"
  placeholder="Last Name (required)"
  required
  maxlength="50"
  formControlName="lastName"/>
```

```
<input class="form-control"
      id="emailId"
      type="email"
      placeholder="Email (required)"
      required
      pattern="[a-z0-9._%+-]+@[a-z0-9.-]+"
      formControlName="email" />
```

6. Check the validation of the fields: FirtsName, LastName, Email

## Task 07. Adjusting Validation Rules in Runtime

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
placeholder = {
  email: 'Email (required)',
  phone: 'Phone'
};

// 2
private buildForm() {
  this.userForm = this.fb.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: [
      { value: 'Zhyrytskyy', disabled: false },
      [Validators.required, Validators.maxLength(50)]
    ],
    email: [
      '',
      [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')]
    ],
    phone: '',
    notification: 'email',
    sendProducts: true
  });
}
```

2. Make changes for the Email input in **SignupReactiveFormComponent template**. Add the following snippet of HTML **after the field email**

```
<input class="form-control"
  id="emailId"
  type="email"
  placeholder="Email (required)"
  placeholder={{placeholder.email}}
  formControlName="email" />
```

3. Make changes to **SignupReactiveFormComponent template**. Add the following snippet of HTML **after the field email**

```
<div class="form-group"
  [ngClass]="{'has-error': (userForm.get('phone').touched ||
    userForm.get('phone').dirty) && userForm.get('phone').invalid }">
  <label class="col-md-2 control-label"
    for="phoneId">Phone</label>

  <div class="col-md-8">
    <input class="form-control"
      id="phoneId"
      type="tel"
      placeholder={{placeholder.phone}}
      formControlName="phone" />
    <span class="help-block" *ngIf="(userForm.get('phone').touched
      || userForm.get('phone').dirty) && userForm.get('phone').errors">
      <span *ngIf="userForm.get('phone').hasError('required')">
```

```

        Please enter your phone number.
    </span>
</span>
</div>
</div>

<div class="form-group">
  <label class="col-md-2 control-label">Send Notifications</label>
  <div class="col-md-8">
    <label class="radio-inline">
      <input type="radio"
        value="email"
        formControlName="notification">Email
    </label>
    <label class="radio-inline">
      <input type="radio"
        value="text"
        formControlName="notification">Text
    </label>
  </div>
</div>

```

4. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```

onSetNotification(notifyVia: string) {
  const phoneControl = this.userForm.get('phone');
  const emailControl = this.userForm.get('email');

  if (notifyVia === 'text') {
    phoneControl.setValidators(Validators.required);
    emailControl.clearValidators();
    this.placeholder.email = 'Email';
    this.placeholder.phone = 'Phone (required)';
  }
  else {
    emailControl.setValidators( [
      Validators.required,
      Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+'),
      Validators.email
    ]);
    phoneControl.clearValidators();
    this.placeholder.email = 'Email (required)';
    this.placeholder.phone = 'Phone';
  }
  phoneControl.updateValueAndValidity();
  emailControl.updateValueAndValidity();
}

```

5. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```

<label class="radio-online">
  <input type="radio"
    value="email"
    formControlName="notification"
    (click)="onSetNotification('email')">Email
</label>
<label class="radio-online">

```

```
        <input type="radio"
              value="text"
              formControlName="notification"
              (click)="onSetNotification('text')">Text
    </label>
```



## Task 08. Custom Validator

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
private buildForm() {
  this.userForm = this.fb.group({
    firstName: ['', [Validators.required, Validators.minLength(3)]],
    lastName: [
      { value: 'Zhyrytskyy', disabled: false },
      [Validators.required, Validators.maxLength(50)]
    ],
    email: [
      '',
      [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
      Validators.email
    ],
    phone: '',
    notification: 'email',
    serviceLevel: '',
    sendProducts: true
  });
}
```

2. Make changes to **SignupReactiveFormComponent template**. Add the following snippet of HTML after the block div for the element «Send Notifications»:

```
<div class="form-group"
  [ngClass]="{'has-error': (userForm.get('serviceLevel').touched ||
userForm.get('serviceLevel').dirty) && !userForm.get('serviceLevel').valid }">
  <label class="col-md-2 control-label"
    for="serviceLevelId">Service Level</label>

    <div class="col-md-8">
      <input class="form-control"
        id="serviceLevelId"
        type="number"
        formControlName="serviceLevel" />
      <span class="help-block"
        *ngIf="(userForm.get('serviceLevel').touched || userForm.get('serviceLevel').dirty) &&
userForm.get('serviceLevel').errors">
        <span
          *ngIf="userForm.get('serviceLevel').hasError('serviceLevel')">
            Please enter correct number from 1 to 5.
          </span>
        </span>
      </div>
    </div>
  </div>
```

3. Create file **app/validators/custom.validators.ts**. Use the following snippet of code:

```
import { AbstractControl, ValidationErrors } from '@angular/forms';

export class CustomValidators {
  static serviceLevel(c: AbstractControl): ValidationErrors | null {
    if (c.value !== undefined && (Number.isNaN(c.value) || c.value < 1 || c.value > 5))
    {
```

```

        return {
          serviceLevel: true
        };
      }
      return null;
    }
  }
}

```

4. Create file **app/validators/index.ts**. Use the following snippet of code:

```
export * from './custom.validators';
```

5. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```

// 1
import { CustomValidators } from './../../validators';

// 2
phone: '',
notification: 'email',
serviceLevel: '',
serviceLevel: ['', CustomValidators.serviceLevel],
sendProducts: true

```

## Task 09 Handle Validators Run Moment

1. Make changes to the file **app/validators/custom.validators.ts**. Use the following snippet of code:

```
static serviceLevel(c: AbstractControl): ValidationErrors | null {
  console.log('Validator: serviceLevel is called');
  if (c.value !== undefined && (Number.isNaN(c.value) || c.value < 1 || c.value > 5))
  {
    return {
      serviceLevel: true
    };
  }
  return null;
}
```

2. Look at the console. You can see that validator runs very often.
3. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
private createForm() {
  this.userForm = new FormGroup({
    firstName: new FormControl(),
    firstName: new FormControl('', {
      validators: [Validators.required, Validators.minLength(3)],
      updateOn: 'blur'
    }),
    lastName: new FormControl(),
    email: new FormControl(),
    phone: new FormControl(),
    notification: new FormControl('email'),
    serviceLevel: new FormControl('', {
      validators: [CustomValidators.serviceLevel],
      updateOn: 'blur'
    }),
    sendProducts: new FormControl(true)
  });
}

// 2
ngOnInit() {
  // this.buildForm();
  this.createForm();
}
```

4. Look at the console. You can see that validator runs more rarely.
5. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
private buildForm() {
  this.userForm = this.fb.group({
    // firstName: ['', [Validators.required, Validators.minLength(3)]],
    // It works!
    firstName: new FormControl('', {validators: [Validators.required,
Validators.minLength(3)], updateOn: 'blur'}),
    // It works since v7
    // firstName: this.fb.control('', { validators: [Validators.required,
Validators.minLength(3)], updateOn: 'blur' }),
  });
}
```

```

        lastName: [
            { value: 'Zhyrytskyy', disabled: false },
            [Validators.required, Validators.maxLength(50)]
        ],
        email: [
            '',
            [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
            Validators.email
        ],
        phone: '',
        notification: 'email',
        serviceLevel: ['', CustomValidators.serviceLevel],
        sendProducts: true
    });
}

// 2
ngOnInit() {
    // this.buildForm();
    // this.createForm();
}

```

6. Look at the console. How often does validator run?

## Task 10. Custom Validator w/ Parameters

1. Make changes to the file **custom.validators.ts**. Use the following snippet of code:

```
// 1
import { AbstractControl, ValidationErrors, ValidatorFn } from '@angular/forms';

// 2
static serviceLevelRange(min: number, max: number): ValidatorFn {
  return (c: AbstractControl): ValidationErrors | null => {
    if (c.value !== undefined && (Number.isNaN(c.value) || c.value < min || c.value >
max)) {
      return {
        serviceLevel: true
      };
    }
    return null;
  }
}
```

2. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1. add properties
rMin = 1;
rMax = 3;

// 2
serviceLevel: ['', CustomValidators.serviceLevel],
serviceLevel: ['', CustomValidators.serviceLevelRange(this.rMin, this.rMax)],
```

3. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
Please enter correct number from 1 to 5.
Please enter correct number from {{rMin}} to {{rMax}}.
```

## Task 11. Custom Validator Directive

1. Create file **ValidatorsModule**. Run the following command from the command line:

```
ng g m validators -m app.module
```

2. Make changes to file **custom.validators.ts**. Use the following snippet of code:

```
// 1
export function checkServiceLevel(
  c: AbstractControl,
  min: number = 1,
  max: number = 5
): ValidationErrors | null {
  if (
    c.value !== undefined &&
    (Number.isNaN(c.value) || c.value < min || c.value > max)
  ) {
    return {
      serviceLevel: true
    };
  }
  return null;
}

// 2
static serviceLevel(c: AbstractControl): ValidationErrors | null {
  console.log('Validator: serviceLevel is called');
  if (
    c.value !== undefined &&
    (Number.isNaN(c.value) || c.value < 1 || c.value > 5)
  ) {
    return {
      serviceLevel: true
    };
  }
  return null;
  return checkServiceLevel(c);
}

// 3
static serviceLevelRange(min: number, max: number): ValidatorFn {
  return (c: AbstractControl): ValidationErrors | null => {
    if (c.value !== undefined && (Number.isNaN(c.value) || c.value < min || c.value >
max)) {
      return {
        'serviceLevel': true
      };
    }
    return null;
    return checkServiceLevel(c, min, max);
  };
}
```

3. Create **ServiceLevelDirective**. Run the following command from the command line:

```
ng g d validators/service-level --skip-tests true --export true -m validators.module
```

4. Replace the content of **ServiceLevelDirective**. Use the following snippet of code:

```

import { Directive } from '@angular/core';
import { Validator, AbstractControl, ValidationErrors, NG_VALIDATORS } from
 '@angular/forms';

import { checkServiceLevel } from './custom.validators';

@Directive({
  selector: '[appServiceLevelValidator]',
  providers: [{
    provide: NG_VALIDATORS,
    useExisting: ServiceLevelDirective,
    multi: true
  }]
})
export class ServiceLevelDirective implements Validator {

  validate(c: AbstractControl): ValidationErrors | null {
    return checkServiceLevel(c, 1, 3);
  }
}

```

5. Make changes to file **app/validators/index.ts**. Use the following snippet of code:

```

export * from './custom.validators';
export * from './service-level.directive';

```

4. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```

serviceLevel: ['', CustomValidators.serviceLevelRange(this.rMin, this.rMax)],
serviceLevel: [''],

```

5. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```

<input class="form-control"
  id="serviceLevelId"
  type="number"
  formControlName="serviceLevel"
  appServiceLevelValidator />

```

## Task 12. Custom Validator Directive w/ Parameters

1. Make changes to **ServiceLevelDirective**. Use the following snippet of code:

```
// 1
import { Directive, Input } from '@angular/core';

// 2
export class ServiceLevelDirective implements Validator {
  @Input() rMin = 1;
  @Input() rMax = 3;

  validate(c: AbstractControl): ValidationErrors | null {
    return checkServiceLevel(c, 1this.rMin, 3this.rMax);
  }
}
```

2. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<input class="form-control"
  id="serviceLevelId"
  type="number"
  formControlName="serviceLevel"
  appServiceLevelValidator rMin="{{rMin}}" rMax="{{rMax}}" />
```



## Task 13. Async Custom Validator

1. Make changes to **custom.validators.ts**. Use the following snippet of code:

```
// 1
// rxjs
import { Observable } from 'rxjs';

// 2
static asyncEmailPromiseValidator(
  c: AbstractControl
):
  | Promise<ValidationErrors | null> | Observable<ValidationErrors | null> {
  const email = c.value;

  return new Promise(resolve => {
    setTimeout(() => {
      if (email === 'existsemail@example.com') {
        resolve({
          asyncEmailInvalid: true
        });
      } else {
        resolve(null);
      }
    }, 2000);
  });
}
```

2. Make changes to **SignupReactiveFormComponent** Use the following snippet of code:

```
// 1 buildForm()
email: [
  '',
  [
    Validators.required,
    Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+'),
    Validators.email
  ],
  [CustomValidators.asyncEmailPromiseValidator]
],

// 2 onSetNotification
emailControl.clearValidators();
emailControl.clearAsyncValidators();

// 3 onSetNotification
emailControl.setValidators([
  Validators.required,
  Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+'),
  Validators.email
]);
emailControl.setAsyncValidators(
  CustomValidators.asyncEmailPromiseValidator
);
```

3. Make changes to **SignupReactiveFormComponent Template** Use the following snippet of HTML:

```
<span class="help-block" *ngIf="(userForm.get('email').touched ||
userForm.get('email').dirty) && userForm.get('email').errors">
  <span *ngIf="userForm.get('email').hasError('required')">
    Please enter your email address.
  </span>
  <span *ngIf="userForm.get('email').hasError('pattern')">
    Please enter a valid email address.
  </span>
  <span *ngIf="userForm.get('email').hasError('email')">
    Please enter a valid email address.
  </span>
  <span *ngIf="userForm.get('email').hasError('asyncEmailInvalid')">
    This email already exists. Please enter other email address.
  </span>
</span>
```

## Task 14. Async Custom Validator Directive

1. Create **AsyncEmailValidatorDirective**. Run the following command from the command line:

```
ng g d validators/async-email-validator --skip-tests true --export true -m validators.module
```

2. Replace the content of **AsyncEmailValidatorDirective**. Use the following snippet of code:

```
import { Directive } from '@angular/core';
import { NG_ASYNC_VALIDATORS, Validator, AbstractControl } from '@angular/forms';

import { Observable } from 'rxjs';
import { debounceTime, distinctUntilChanged, first } from 'rxjs/operators';

import { CustomValidators } from '../custom.validators';

@Directive({
  selector: '[appAsyncEmailValidator][formControlName],
[appAsyncEmailValidator][ngModel]',
  providers: [
    {
      provide: NG_ASYNC_VALIDATORS,
      useExisting: AsyncEmailValidatorDirective,
      multi: true
    }
  ]
})
export class AsyncEmailValidatorDirective implements Validator {
  validate(c: AbstractControl): Promise<{ [key: string]: any}>|Observable < {[key:
string]: any}> {
    return CustomValidators.asyncEmailPromiseValidator(c);
    // return this.validateEmailObservable(c.value)
    // .pipe(
    //   debounceTime(1000),
    //   distinctUntilChanged(),
    // // The observable returned must be finite, meaning it must complete at some point.
    // // To convert an infinite observable into a finite one, pipe the observable through a
    // // filtering operator such as first, last, take, or takeUntil
    //   first()
    // );
  }

  private validateEmailObservable(email: string) {
    return new Observable(observer => {
      if (email === 'existsemail@example.com') {
        observer.next({asyncEmailInvalid: true});
      } else {
        observer.next(null);
      }
    });
  }
}
```

3. Make changes to file **validators/index.ts**. Use the following snippet of code:

```
export * from './async-email-validator.directive';
```

4. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
email: [
  '',
  [
    Validators.required,
    Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+'),
    Validators.email
  ],
  // [CustomValidators.asyncEmailPromiseValidator]
],
```

5. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<input class="form-control"
  id="emailId"
  type="email"
  placeholder="Email (required)"
  formControlName="email"
  appAsyncEmailValidator />
```

1. Enter the value **existsemail@example.com** to the field and ensure that validation runs.
2. Make changes to file **validators/async-email-validator.directive.ts** and ensure that validation runs.

```
validate(c: AbstractControl): Promise<{ [key: string]: any}>|Observable < {[key:
string]: any}> {
  // return CustomValidators.asyncEmailPromiseValidator(c);
  // return this.validateEmailObservable(c.value)
  // .pipe(
  //   debounceTime(1000),
  //   distinctUntilChanged(),
  //   first()
  // );
}
```

## Task 15. Cross-Field Validation

1. Make changes to **SignupReactiveFormComponent template**. Add the following snippet of HTML after the block of email:

```
<div class="form-group"
  [ngClass]='{\'has-error\': (userForm.get(\'confirmEmail\').touched ||
                                userForm.get(\'confirmEmail\').dirty) &&
                                userForm.get(\'confirmEmail\').invalid }'>

  <label class="col-md-2 control-label"
    for="confirmEmailId">Confirm Email</label>

  <div class="col-md-8">
    <input class="form-control"
      id="confirmEmailId"
      type="email"
      placeholder={{placeholder.confirmEmail}}
      formControlName="confirmEmail" />
    <span class="help-block" *ngIf="(userForm.get(\'confirmEmail\').touched ||
                                    userForm.get(\'confirmEmail\').dirty) &&
                                    userForm.get(\'confirmEmail\').errors">
      <span *ngIf="userForm.get(\'confirmEmail\').hasError(\'required\')">
        Please confirm your email address.
      </span>
    </span>
  </div>
</div>
```

2. Make changes to **SignupReactiveFormComponent**. Ensure that the validation of the field `confirmEmail` runs.

```
// 1
placeholder = {
  email: 'Email (required)',
  confirmEmail: 'Confirm Email (required)',
  phone: 'Phone'
};

// 2
email: ['',
  [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')], ,
  Validators.email],
// [CustomValidators.asyncEmailPromiseValidator]
],
confirmEmail: ['', Validators.required],
```

3. Make changes to **SignupReactiveFormComponent template**. Add the following snippet of HTML before the block of email and put into it blocks for email and `confirmEmail`:

```
<div formGroupName="emailGroup">
  <!-- Put here email and confirmEmail blocks -->
</div>
```

4. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```

email: [
  '',
  [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
  Validators.email],
// [CustomValidators.asyncEmailPromiseValidator]
],
confirmEmail: ['', Validators.required],
emailGroup: this.fb.group({
  email: ['',
    [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
    Validators.email],
  // [CustomValidators.asyncEmailPromiseValidator]
  ],
  confirmEmail: ['', Validators.required],
}),

```

5. Make changes to **SignupReactiveFormComponent** template. Use the following snippet of HTML:

```

<div class="form-group"
  [ngClass]="{'has-error': (userForm.get('emailGroup.email').touched ||
    userForm.get('emailGroup.email').dirty) &&
    userForm.get('emailGroup.email').invalid }">

  <label class="col-md-2 control-label"
    for="emailId">Email</label>

  <div class="col-md-8">
    <input class="form-control"
      id="emailId"
      type="email"
      placeholder="{{placeholder.email}}"
      formControlName="email"
      appAsyncEmailValidator />
    <span class="help-block"
      *ngIf="(userForm.get('emailGroup.email').touched ||
        userForm.get('emailGroup.email').dirty) &&
        userForm.get('emailGroup.email').errors">
      <span
        *ngIf="userForm.get('emailGroup.email').hasError('required')">
        Please enter your email address.
      </span>
      <span
        *ngIf="userForm.get('emailGroup.email').hasError('pattern')">
        Please enter a valid email address.
      </span>
      <span
        *ngIf="userForm.get('emailGroup.email').hasError('email')">
        Please enter a valid email address.
      </span>
      <span
        *ngIf="userForm.get('emailGroup.email').hasError('asyncEmailInvalid')">
        This email already exists. Please enter other email
        address.
      </span>
    </span>
  </div>
</div>

```

```

        </div>

<div class="form-group"
      [ngClass]="{'has-error':
(userForm.get('emailGroup.confirmEmail').touched ||
userForm.get('emailGroup.confirmEmail').dirty) &&
userForm.get('emailGroup.confirmEmail').invalid }">
    <label class="col-md-2 control-label"
          for="confirmEmailId">Confirm Email</label>

    <div class="col-md-8">
        <input class="form-control"
              id="confirmEmailId"
              type="email"
              placeholder={{placeholder.confirmEmail}}
              formControlName="confirmEmail" />
        <span class="help-block"
*ngIf="(userForm.get('emailGroup.confirmEmail').touched ||
userForm.get('emailGroup.confirmEmail').dirty) &&
userForm.get('emailGroup.confirmEmail').errors">
            <span
*ngIf="userForm.get('emailGroup.confirmEmail').hasError('required')">
                Please confirm your email address.
            </span>
        </span>
    </div>
</div>
</div>

```

6. Make changes to file **custom.validators.ts**. Use the following snippet of code:

```

static emailMatcher(c: AbstractControl): ValidationErrors | null {
    const emailControl = c.get('email');
    const emailConfirmControl = c.get('confirmEmail');

    if (emailControl.pristine || emailConfirmControl.pristine) {
        return null;
    }

    if (emailControl.value === emailConfirmControl.value) {
        return null;
    }

    return { emailMatch: true };
}

```

7. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```

// 1
emailGroup: this.fb.group({
    email: ['',
        [Validators.required, Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+')],
        Validators.email],

```

```

//[CustomValidators.asyncEmailPromiseValidator]
    ],
    confirmEmail: ['', Validators.required],
  }, {validator: CustomValidators.emailMatcher}),

// 2 Replace method onSetNotification with new one
onSetNotification(notifyVia: string) {
  const controls = new Map();
  controls.set('phoneControl', this.userForm.get('phone'));
  controls.set('emailGroup', this.userForm.get('emailGroup'));
  controls.set('emailControl', this.userForm.get('emailGroup.email'));
  controls.set(
    'confirmEmailControl',
    this.userForm.get('emailGroup.confirmEmail')
  );

  if (notifyVia === 'text') {
    controls.get('phoneControl').setValidators(Validators.required);
    controls.forEach(
      (control, index) => {
        if (index !== 'phoneControl') {
          control.clearValidators();
          control.clearAsyncValidators();
        }
      }
    );

    this.placeholder = {
      phone: 'Phone (required)',
      email: 'Email',
      confirmEmail: 'Confirm Email'
    };
  } else {
    const emailControl = controls.get('emailControl');
    emailControl.setValidators([
      Validators.required,
      Validators.pattern('[a-z0-9._%+-]+@[a-z0-9.-]+'),
      Validators.email
    ]);
    emailControl.setAsyncValidators(CustomValidators.asyncEmailPromiseValidator);
    controls.get('confirmEmailControl').setValidators([Validators.required]);
    controls.get('emailGroup').setValidators([CustomValidators.emailMatcher]);
    controls.get('phoneControl').clearValidators();

    this.placeholder = {
      phone: 'Phone',
      email: 'Email (required)',
      confirmEmail: 'Confirm Email (required)'
    };
  }
  controls.forEach(control => control.updateValueAndValidity());
}

```

8. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<div formGroupName="emailGroup">
```



```

[ngClass]="{'has-error': userForm.get('emailGroup').errors}">
<span class="help-block" *ngIf="(userForm.get('emailGroup.confirmEmail').touched ||
    userForm.get('emailGroup.confirmEmail').dirty) &&
    userForm.get('emailGroup.confirmEmail').errors">
    <span *ngIf="userForm.get('emailGroup.confirmEmail').hasError('required')">
        Please confirm your email address.
    </span>
</span>
<span class="help-block" *ngIf="(userForm.get('emailGroup.confirmEmail').touched ||
    userForm.get('emailGroup.confirmEmail').dirty) &&
    (userForm.get('emailGroup.confirmEmail').errors ||
    userForm.get('emailGroup').errors) ">
    <span *ngIf="userForm.get('emailGroup.confirmEmail').hasError('required')">
        Please confirm your email address.
    </span>
    <span *ngIf="userForm.get('emailGroup').hasError('emailMatch')">
        The confirmation does not match the email address.
    </span>
</span>

```

## Task 16. Adjusting Validation Rules

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
import { Component, OnInit, OnDestroy } from '@angular/core';
import { Subscription } from 'rxjs';

// 2
export class SignupReactiveFormComponent implements OnInit, OnDestroy

// 3
private sub: Subscription;

// 4
private watchValueChanges() {
  this.sub = this.userForm.get('notification').valueChanges
    .subscribe(value => console.log(value));
}

// 5
ngOnInit() {
  this.buildForm();
  this.watchValueChanges();
}

// 6
ngOnDestroy() {
  this.sub.unsubscribe();
}
```

2. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<div class="form-group">
  <label class="col-md-2 control-label">Send Notifications</label>
  <div class="col-md-8">
    <label class="radio-online">
      <input type="radio"
        value="email"
        formControlName="notification"
        (click)="onSetNotification('email')">Email
    </label>
    <label class="radio-online">
      <input type="radio"
        value="text"
        formControlName="notification"
        (click)="onSetNotification('text')">Text
    </label>
  </div>
</div>
```

3. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
```

```
private watchValueChanges() {  
    this.sub = this.userForm.get('notification').valueChanges  
        .subscribe(value => console.log(value));  
        .subscribe(value => this.setNotification(value));  
}  
  
// 2  
private onSsetNotification(notifyVia: string) {
```

## Task 17. Displaying Validation Messages

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
import { FormGroup, FormControl, FormBuilder, Validators, AbstractControl } from
 '@angular/forms';

// 2
userForm: FormGroup;
validationMessage: string;

// 3
private sub: Subscription;
private validationMessagesMap = {
  email: {
    required: 'Please enter your email address.',
    pattern: 'Please enter a valid email address.',
    email: 'Please enter a valid email address.',
    asyncEmailInvalid:
      'This email already exists. Please enter other email address.'
  }
};

// 4
private setValidationMessage(c: AbstractControl, controlName: string) {
  this.validationMessage = '';

  if ((c.touched || c.dirty) && c.errors) {
    this.validationMessage = Object.keys(c.errors)
      .map(key => this.validationMessagesMap[controlName][key])
      .join(' ');
  }
}

// 5
private watchValueChanges() {
  ...

  const emailControl = this.userForm.get('emailGroup.email');
  const sub = emailControl.valueChanges.subscribe(value =>
    this.setValidationMessage(emailControl, 'email')
  );
  this.sub.add(sub);
}

// 6
onBlur() {
  const emailControl = this.userForm.get('emailGroup.email');
  this.setValidationMessage(emailControl, 'email');
}
```

2. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
<div class="form-group"
```

```

[ngClass]="{'has-error':
(userForm.controls.emailGroup.controls.email.touched ||
userForm.get('emailGroup.email').dirty)
&&
userForm.get('emailGroup.email').invalid
}">
[ngClass]="{'has-error': validationMessage}">
  <label class="col-md-2 control-label"
    for="emailId">Email</label>

  <div class="col-md-8">
    <input class="form-control"
      id="emailId"
      type="email"
      placeholder={{placeholder.email}}
      formControlName="email"
      asyncEmailValidator
      (blur)="onBlur()" />
    <span class="help-block"
      *ngIf="(userForm.get('emailGroup.email').touched ||
userForm.get('emailGroup.email').dirty) &&
userForm.get('emailGroup.email').errors">
      <span
        *ngIf="userForm.get('emailGroup.email').hasError('required')">
          Please enter your email address.
        </span>
        <span
          *ngIf="userForm.get('emailGroup.email').hasError('pattern')">
            Please enter a valid email address.
          </span>
          <span
            *ngIf="userForm.get('emailGroup.email').hasError('email')">
              Please enter a valid email address.
            </span>
            <span *ngIf="userForm.get('emailGroup.email').hasError('asyncEmailInvalid')">
              This email already exists. Please enter other email
              address.
            </span>
          </span>

    </span>
    <span class="help-block" *ngIf="validationMessage">
      {{ validationMessage }}
    </span>
  </div>
</div>

```

## Task 18. Reactive Transformations

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
import { Subscription } from 'rxjs';  
import { debounceTime } from 'rxjs/operators';
```

2. Внесите изменения в метод **watchValueChanges**

```
private watchValueChanges() {  
  ...  
  const sub = emailControl.valueChanges  
    .pipe(  
      debounceTime(1000)  
    )  
    .subscribe(value => this.setValidationMessage(emailControl, 'email'));  
  this.sub.add(sub);  
}
```

## Task 19. Define the input element(s) to duplicate

1. Make changes to method **buildForm**. Use the following snippet of code:

```
sendProducts: true,  
addressType: 'home',  
country: '',  
city: '',  
zip: '',  
street1: '',  
street2: ''
```

2. Uncomment HTML of **SignupReactiveFormComponent**, which starts from  
<!--<div >

3. Replace the div element. Use the following snippet of code:

```
<div >  
<div *ngIf="userForm.get('sendProducts').value">
```

4. Make changes to the markup of the fields **Home, Work, Other**

```
formControlName="addressType"
```

5. Make changes to the markup of the fields **Country, City, Zip Code, Street Address 1, Street Address 2**

```
// Country  
formControlName="country"
```

```
// City  
formControlName="city"
```

```
// Zip Code  
formControlName="zip"
```

```
// Street Address 1  
formControlName="street1"
```

```
// Street Address 2  
formControlName="street2"
```

## Task 20. Define a FormGroup

1. Make changes to the method **buildForm**. Use the following snippet of code:

```
addressType: 'home',  
country: '',  
city: '',  
zip: '',  
street1: '',  
street2: ''  
addresses: this.fb.group({  
  addressType: 'home',  
  country: '',  
  city: '',  
  zip: '',  
  street1: '',  
  street2: ''  
})
```

2. Wrap 4 div blocks, which are in the block  
`<div *ngIf="userForm.get('sendProducts').value">` and contain the fields for address. Use the following snippet of HTML:

```
<div formGroupName="addresses">...</div>
```



## Task 21. Refactor to Make Copies

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
private buildAddress(): FormGroup {
  return this.fb.group({
    addressType: 'home',
    country: '',
    city: '',
    zip: '',
    street1: '',
    street2: ''
  });
}

// 2
addresses: this.buildAddress()
addresses: this.fb.group({
  addressType: 'home',
  country: '',
  city: '',
  zip: '',
  street1: '',
  street2: ''
})
```

## Task 22. Create a FormArray

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
// 1
import { FormGroup, FormControl, FormArray, FormBuilder, Validators, AbstractControl }
from '@angular/forms';

// 2
get addresses(): FormArray {
  return this.userForm.get('addresses') as FormArray;
}
```

2. Make changes to the method **buildForm**. Use the following snippet of code:

```
addresses: this.buildAddress()
addresses: this.fb.array([this.buildAddress()])
```

3. Make changes to **SignupReactiveFormComponent template**. Create wrapper block for the block `<div formGroupName="addresses">`. Use the following snippet of HTML:

```
<div formArrayName="addresses">
</div>
```

4. Replace the expression of attribute **formGroupName="addresses"**. Use the following snippet of HTML:

```
<div formGroupName="addresses">
<div formGroupName="0">
```

## Task 23. Loop through the FormArray

1. Make changes to **SignupReactiveFormComponent** template. Use the following snippet of HTML:

```
<div formArrayName="addresses">
  <div formGroupName="0">
    <div *ngFor="let address of addresses.controls; let i = index" [formGroupName]="i"
  >
```

2. Replace the value of attribute **id** for the fields **Home, Work, Other**. Use the following snippet of HTML:

```
id="addressType1Id"
id="{{ 'addressType1Id' + i }}"
```

3. Replace the value of attribute **for** for the label elements «**Country, City, Zip Code**», «**Street Address 1**», «**Street Address 2**». Use the following snippet of HTML:

```
// Country, City, Zip Code
for="cityId"
attr.for="{{ 'countryId' + i }}"

// Street Address 1
for="street1Id"
attr.for="{{ 'street1Id' + i }}"

// Street Address 2
for="street2Id"
attr.for="{{ 'street2Id' + i }}"
```

4. Replace the value of the attribute **id** for the fields **Country, City, Zip Code, Street Address 1, Street Address 2**. Use the following snippet of HTML:

```
// Country
id="countryId"
id="{{ 'countryId' + i }}"

// City
id="cityId"
id="{{ 'cityId' + i }}"

// Zip Code
id="zipId"
id="{{ 'zipId' + i }}"

// Street Address 1
id="street1Id"
id="{{ 'street1Id' + i }}"

// Street Address 2
id="street2Id"
id="{{ 'street2Id' + i }}"
```

## Task 24. Duplicate the Input Elements

1. Make changes to **SignupReactiveFormComponent**. Use the following snippet of code:

```
onAddAddress(): void {  
    this.addresses.push(this.buildAddress());  
}
```

2. Make changes to **SignupReactiveFormComponent template**. Use the following snippet of HTML:

```
// 1  
<div formArrayName="addresses">  
...  
</div>  
  
    <div class="form-group">  
        <div class="col-md-4 col-md-offset-2">  
            <button class="btn btn-primary"  
                type="button"  
                (click)="onAddAddress()">  
                Add Another Address  
            </button>  
        </div>  
    </div>  
  
// 2  
<br>userForm.get('emailGroup').errors {{userForm.get('emailGroup').errors | json}}  
<br>Street: {{addresses.get('0.street1')?.value}}
```