

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	33
Task 17. Displaying Validation Messages	34
Task 18. Define the input element(s) to duplicate.....	38
Task 19. Define a FormGroup.....	39
Task 20. Refactor to Make Copies.....	40
Task 21. Create a FormArray	41
Task 22. Loop through the FormArray	42
Task 23. Duplicate the Input Elements.....	43
Task 24. Remove Input Elements	44
Task 25. Control Value Accessor Interface.....	45

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. Create **app/data/countries.ts**. Use the following snippet of code:

```
export const COUNTRIES = ['Ukraine', 'Armenia', 'Belarus', 'Hungary', 'Kazakhstan', 'Poland'];
```

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

```
// 1  
import { UserModel } from '../models/user.model';  
import { COUNTRIES } from 'src/app/data/countries';  
import { JsonPipe, NgClass, NgForOf, NgIf } from '@angular/common';  
  
// 2  
imports: [FormsModule, NgIf, NgForOf, NgClass, JsonPipe],  
  
// 3  
countries: Array<string> = COUNTRIES;  
user: UserModel = new UserModel();  
  
// 4  
onSave(signupForm: NgForm): void {  
  // Form model  
  console.log(signupForm.form);  
  // Form value  
  console.log(`Saved: ${JSON.stringify(signupForm.value)}`);  
}
```

5. 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"
```

```

        [class.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>

// 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>

```

6. 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 }}

```

7. 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:

```
// 1
import { ReactiveFormsModule, FormGroup, FormControl } from '@angular/forms';
import { UserModel } from '../models/user.model';
import { COUNTRIES } from 'src/app/data/countries';
import { NgForOf } from '@angular/common';

// 2
imports: [ReactiveFormsModule, NgForOf],

// 3
countries: Array<string> = COUNTRIES;

// data model
user: UserModel = new UserModel();

// form model
userForm = new FormGroup({
  firstName: new FormControl(''),
  lastName: new FormControl(''),
  email: new FormControl(''),
  sendProducts: new FormControl(true)
});

onSave(): void {
  // 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. Make changes to the following import

```
// 1
import { AbstractControl, ReactiveFormsModule, FormGroup, FormControl } from
 '@angular/forms'; import { JsonPipe, NgClass, NgIf, NgForOf } from '@angular/common';

// 2
imports: [ReactiveFormsModule, NgIf, NgForOf, NgClass, JsonPipe],
```

4. Add getters to the class of **SignupReactiveFormComponent**

```
get firstName(): AbstractControl {
  return this.userForm.get('firstName')!;
}

get lastName(): AbstractControl {
  return this.userForm.get('lastName')!;
}

get email(): AbstractControl {
  return this.userForm.get('email')!;
}
```

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

For firstName

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

For lastName

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

For email

```
[ngClass]="{'has-error': (email.touched || email.dirty) && email.invalid }"
```

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

For firstName

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

For lastName


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

For email

```
<span class="help-block" *ngIf="(email.touched || email.dirty) && email.errors">
  <span *ngIf="email.hasError('required')">
    Please enter your email address.
  </span>
  <span *ngIf="email.hasError('pattern')">
    Please enter a valid email address.
  </span>
  <span *ngIf="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(): void {
  this.userForm.setValue({
    firstName: this.user.firstName,
    lastName: this.user.lastName,
    email: this.user.email,
    sendProducts: this.user.sendProducts
  });
}

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

// 4
ngOnInit(): void {
  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 { AbstractControl, ReactiveFormsModule, FormGroup, FormControl, FormBuilder,
NullableFormBuilder } from '@angular/forms';
import { Component, OnInit, inject } from '@angular/core';

private fb = inject(FormBuilder);

// form model
// userForm = new FormGroup({
//   firstName: new FormControl(''),
//   lastName: new FormControl(''),
//   email: new FormControl(''),
//   sendProducts: new FormControl(true)
// });

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

ngOnInit(): void {
  this.setFormValues();
  this.patchFormValues();
}

onReset(): void {
  this.userForm.reset();
}
```

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

```
<button class="btn btn-primary"
  type="submit"
  [disabled]="!userForm.valid">
  Save
</button>
<button class="btn btn-primary"
  type="button"
  (click)="onReset()">
  Reset
</button>
```

3. Look at the result. Click "Reset" button. Replace this.fb.group with this.fb.nonNullable.group. Click "Reset" button. Look at the result.
4. Replace this.fb.nonNullable.group with this.fb.group. Replace FormBuilder with NullableFormBuilder Click "Reset" button. Look at the result.

Task 06. Setting Built-in Validators

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

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

// 2
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:

```
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
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
});

// 3
get phone(): AbstractControl {
  return this.userForm.get('phone')!;
}

// 4
// private setFormValues(): void {
//   this.userForm.setValue({
//     firstName: this.user.firstName,
//     lastName: { value: this.user.lastName, disabled: false },
//     email: this.user.email,
//     sendProducts: this.user.sendProducts
//   });
// }
```

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"
```

```

phone.invalid }">
    [ngClass]="{'has-error': (phone.touched || phone.dirty) &&
    <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="(phone.touched || phone.dirty)
&& phone.errors">
        <span *ngIf="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): void {
  const phoneControl = this.phone;
  const emailControl = this.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:

```
// 1
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
get serviceLevel(): AbstractControl {
  return this.userForm.get('serviceLevel')!;
}
```

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': (serviceLevel.touched ||
serviceLevel.dirty) && !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="(serviceLevel.touched ||
serviceLevel.dirty) && serviceLevel.errors">
      <span *ngIf="serviceLevel.hasError('serviceLevel')">
        Please enter correct number from 1 to 5.
      </span>
    </span>
  </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({ value }: AbstractControl): ValidationErrors | null {
  if (value !== undefined && (Number.isNaN(value) || value < 1 || 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({ value }: AbstractControl): ValidationErrors | null {
  console.log('Validator: serviceLevel is called');
  if (value !== undefined && (Number.isNaN(value) || value < 1 || 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
userForm = new FormGroup({
  firstName: new FormControl(''),
  firstName: new FormControl('', {
    validators: [Validators.required, Validators.minLength(3)],
    updateOn: 'blur',
    nonNullable: true
  }),
  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)
});
```

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

```
// 1
userForm = this.fb.group({
  // firstName: ['', [Validators.required, Validators.minLength(3)]],
  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  
  });  
}
```

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 ({ value }: AbstractControl): ValidationErrors | null => {
    if (value !== undefined && (Number.isNaN(value) || value < min || 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 a file **validators/checkServiceLevel.function.ts**. Use the following snippet of code:

```
import { ValidationErrors } from "@angular/forms";

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

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

```
// 1
import { checkServiceLevel } from './checkServiceLevel.function';

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

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

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

```
ng g d validators/service-level --skip-tests true --standalone true
```

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 "../checkServiceLevel.function";

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

  validate(c: AbstractControl): ValidationErrors | null {
    return checkServiceLevel(c.value, 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';

```

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

```

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

// 2
imports: [ReactiveFormsModule, NgIf, NgForOf, NgClass, JsonPipe, ServiceLevelDirective],

// 3
serviceLevel: ['', CustomValidators.serviceLevelRange(this.rMin, this.rMax)],
serviceLevel: [''],

```

7. 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.value, 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" />
```

3. Make changes to SignupReactiveFormComponent class. Use the following snippet of code:
rMax = 34;

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
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="(email.touched || email.dirty) && email.errors">
  <span *ngIf="email.hasError('required')">
    Please enter your email address.
  </span>
  <span *ngIf="email.hasError('pattern')">
    Please enter a valid email address.
  </span>
  <span *ngIf="email.hasError('email')">
    Please enter a valid email address.
  </span>
  <span *ngIf="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 --standalone true
```

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

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

import { Observable, debounceTime, distinctUntilChanged, first } from 'rxjs';
import { CustomValidators } from '../custom.validators';

@Directive({
  selector: '[appAsyncEmailValidator][formControlName],
[appAsyncEmailValidator][ngModel]',
  standalone: true,
  providers: [
    {
      provide: NG_ASYNC_VALIDATORS,
      useExisting: AsyncEmailValidatorDirective,
      multi: true
    }
  ]
})
export class AsyncEmailValidatorDirective implements Validator {
  validate(c: AbstractControl): Promise<ValidationErrors | null> |
  Observable<ValidationErrors | null> {
    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): Observable<ValidationErrors | null> {
    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:

```
// 1
import { CustomValidators, ServiceLevelDirective, AsyncEmailValidatorDirective } from
'../../validators';

// 2
imports: [ReactiveFormsModule, NgIf, NgForOf, NgClass, JsonPipe, ServiceLevelDirective,
AsyncEmailValidatorDirective],

// 3
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 />
```

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

```
validate(c: AbstractControl): Promise< ValidationErrors | null >|Observable <
ValidationErrors | null > {
  // 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\': (confirmEmail.touched ||
                        confirmEmail.dirty) &&
                        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="(confirmEmail.touched ||
                                confirmEmail.dirty) &&
                                confirmEmail.errors">
      <span *ngIf="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
get confirmEmail(): AbstractControl {
  return this.userForm.get('confirmEmail')!;
}
```

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 confirmPassword blocks -->
</div>
```

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

```
// 1
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],
}),

// 2
get email(): AbstractControl {
  return this.userForm.get('emailGroup.email')!;
}

get confirmEmail(): AbstractControl {
  return this.userForm.get('emailGroup.confirmEmail')!;
}
```

5. 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 };
}
```

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

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

// 2
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} as AbstractControlOptions),

// 3
get emailGroup(): AbstractControl {
    return this.userForm.get('emailGroup')!;
}

// 4 Replace method onSetNotification with new one
onSetNotification(notifyVia: string): void {
    const controls = new Map();
    controls.set('phoneControl', this.phone);
    controls.set('emailGroup', this.emailGroup);
    controls.set('emailControl', this.email);
    controls.set('confirmEmailControl', this.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());
}

```

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

```
<div formGroupName="emailGroup"
    [ngClass]="{'has-error': emailGroup.errors}">

<span class="help-block" *ngIf="(confirmEmail.touched || confirmEmail.dirty) &&
                                (confirmEmail.errors || emailGroup.errors)">
    <span *ngIf="confirmEmail.hasError('required')">
        Please confirm your email address.
    </span>
    <span *ngIf="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
get notification(): AbstractControl {
    return this.userForm.get('notification')!;
}

// 2
private watchValueChanges(): void {
    this.notification.valueChanges.subscribe(value => console.log(value));
}

// 3
ngOnInit(): void {
    this.watchValueChanges();
    // this.setFormValues();
    // this.patchFormValues();
}
```

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(): void {
    this.notification.valueChanges
        .subscribe(value => console.log(value));
        .subscribe(value => this.setNotification(value));
}

// 2
private onSetNotification(notifyVia: string): void {
```

Task 17. Displaying Validation Messages

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

```
// 1
// map includes all controls,
// even though they don't have validators
validationMessagesMap = new Map([
  ['firstName', {
    message: '', // <== message for user
    required: 'Please enter your first name.',
    minlength: 'The first name must be longer than 3 characters.'
  }],
  ['lastName', {
    message: '',
    required: 'Please enter your last name.'
  }],
  ['email', {
    message: '',
    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.'
  }],
  ['confirmEmail', {
    message: '',
    required: 'Please confirm your email address.'
  }],
  ['emailGroup', {
    message: '',
    emailMatch: 'The confirmation does not match the email address.'
  }],
  ['phone', {
    message: '',
    required: 'Please enter your phone number.'
  }],
  ['serviceLevel', {
    message: '',
    serviceLevel: `Please enter correct number from ${this.rMin} to ${this.rMax}.`
  }],
  ['notification', {
    message: ''
  }],
  ['sendProducts', {
    message: ''
  }],
]);

// 2
get sendProducts(): AbstractControl {
  return this.userForm.get('sendProducts')!;
}

// 3
ngOnInit(): void {
```

```

        this.watchValueChanges();
        this.setValidationMessages();
    }

    // 4
    isShowValidationMessage(controlName: string): boolean {
        return this.validationMessagesMap.get(controlName)!.message && (this as {[index: string]: any})[controlName].touched;
    }

    // 5
    private buildValidationMessages(controlName: string): void {
        const c: AbstractControl = (this as {[index: string]: any})[controlName]; // вызов
        return getpa
        this.validationMessagesMap.get(controlName)!.message = '';

        if (c.errors) {
            this.validationMessagesMap.get(controlName)!.message = Object.keys(c.errors)
                .map(key => {
                    const value = this.validationMessagesMap.get(controlName)!.
                    return (value as {[index: string]: any})[key];
                })
                .join(' ');
        }
    }

    // 6
    private setValidationMessages(): void {
        this.validationMessagesMap.forEach((control, cntrlName) => {
            this.buildValidationMessages(cntrlName);
        });
    }

    // 7
    private watchValueChanges(): void {
        ...

        this.userForm.valueChanges.subscribe(ignorValue =>
            this.setValidationMessages()
        );
    }
}

```

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

```

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

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

```

```

        </span>
    </span>
    <span class="help-block" *ngIf="isShowValidationMessage('firstName') ">
        {{validationMessagesMap.get('firstName').message}}
    </span>

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

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

    // emailGroup
    [ngClass]="{'has-error': emailGroup.errors}"
    [ngClass]="{'has-error': isShowValidationMessage('emailGroup') }"

    // email
    [ngClass]="{'has-error': (email.touched || email.dirty) && email.invalid }"
    [ngClass]="{'has-error': isShowValidationMessage('email') }"

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

    // confirmEmail
    [ngClass]="{'has-error': (confirmEmail.touched || confirmEmail.dirty) &&
    confirmEmail.invalid }"
    [ngClass]="{'has-error': isShowValidationMessage('confirmEmail') }"

    <span class="help-block" *ngIf="(confirmEmail.touched || confirmEmail.dirty) &&
    (confirmEmail.errors || emailGroup.errors) ">
        <span *ngIf="confirmEmail.hasError('required')">
            Please confirm your email address.
        </span>
        <span *ngIf="emailGroup.hasError('emailMatch')">

```

```

        The confirmation does not match the email address.
    </span>
</span>
<span class="help-block" *ngIf="isShowValidationMessage('confirmEmail') ">
    {{validationMessagesMap.get('confirmEmail')?.message}}
</span>
<span class="help-block" *ngIf="isShowValidationMessage('emailGroup') ">
    {{validationMessagesMap.get('emailGroup')?.message}}
</span>

// phone
[ngClass]="{'has-error': (phone.touched || phone.dirty) && phone.invalid }"
[ngClass]="{'has-error': isShowValidationMessage('phone') }"

<span class="help-block" *ngIf="(phone.touched || phone.dirty) && phone.errors">
    <span *ngIf="phone.hasError('required')">
        Please enter your phone number.
    </span>
</span>
<span class="help-block" *ngIf="isShowValidationMessage('phone') ">
    {{validationMessagesMap.get('phone')?.message}}
</span>

// serviceLevel
[ngClass]="{'has-error': (serviceLevel.touched || serviceLevel.dirty) &&
!serviceLevel.valid }"
[ngClass]="{'has-error': isShowValidationMessage('serviceLevel') }"

<span class="help-block" *ngIf="(serviceLevel.touched || serviceLevel.dirty) &&
serviceLevel.errors">
    <span *ngIf="serviceLevel.hasError('serviceLevel')">
        Please enter correct number from {{rMin}} to {{rMax}}.
    </span>
</span>
<span class="help-block" *ngIf="isShowValidationMessage('serviceLevel') ">
    {{validationMessagesMap.get('serviceLevel')?.message}}
</span>

```

Task 18. Define the input element(s) to duplicate

1. Make changes to the model of form. 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="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 19. 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="sendProducts.value">` and contain the fields for address. Use the following snippet of HTML:

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

Task 20. Refactor to Make Copies

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

```
// 1
private buildAddress() {
  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 21. Create a FormArray

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

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

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

2. Make changes to the form model. 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 22. 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 23. 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}}
```

Task 24. Remove Input Elements

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

```
onRemoveAddress(index: number): void {  
    this.addresses.removeAt(index);  
}
```

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

```
<div *ngFor="let address of addresses.controls; let i = index" [formGroupName]="i">  
    <div class="form-group" >  
        <label class="col-md-2 control-label">Address Type</label>  
        <div class="col-md-8">  
            <label class="radio-inline">  
                <input type="radio" id="{{ 'addressType1Id' + i }}" value="home"  
                    formControlName="addressType">Home  
            </label>  
            <label class="radio-inline">  
                <input type="radio" id="{{ 'addressType2Id' + i }}" value="work"  
                    formControlName="addressType">Work  
            </label>  
            <label class="radio-inline">  
                <input type="radio" id="{{ 'addressType3Id' + i }}" value="other"  
                    formControlName="addressType">Other  
            </label>  
        </div>  
        <div class="col-md-1 text-right" *ngIf="i>0">  
            <button class="btn btn-danger" (click)="onRemoveAddress(i)">X</button>  
        </div>  
    </div>
```

Task 25. Control Value Accessor Interface

1. Create **AddressInfoComponent**. Run the following command from the command line

ng g c reactive-forms/signup-reactive-form/components/address-info --standalone true

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

```
import { Component, OnInit, forwardRef, Input, Output, EventEmitter, inject } from
 '@angular/core';
import {
  Validators,
  ControlValueAccessor,
  NG_VALUE_ACCESSOR,
  AbstractControl,
  ValidationErrors,
  NG_VALIDATORS,
  Validator,
  NonNullableFormBuilder
} from '@angular/forms';

import { COUNTRIES } from 'src/app/data/countries';

@Component({
  selector: 'app-address-info',
  standalone: true,
  imports: [ReactiveFormsModule, NgForOf, NgIf, NgClass],
  templateUrl: './address-info.component.html',
  styleUrls: ['./address-info.component.css'],
  providers: [
    {
      provide: NG_VALUE_ACCESSOR,
      useExisting: forwardRef(() => AddressInfoComponent),
      multi: true
    },
    {
      provide: NG_VALIDATORS,
      useExisting: forwardRef(() => AddressInfoComponent),
      multi: true
    }
  ]
})
export class AddressInfoComponent
  implements OnInit, ControlValueAccessor, Validator {
  private fb = inject(NonNullableFormBuilder);

  countries: Array<string> = COUNTRIES;

  validationMessagesMap = new Map([
    ['city', {
      message: '', // <== сформированное сообщение для пользователя
      required: 'Please enter city',
    }]
  ]);

  addressInfoForm = this.buildAddress();
```

```

@Input('index') i = 0;
@Output() removeAddress = new EventEmitter<number>();

get city(): AbstractControl {
    return this.addressInfoForm.get('city')!;
}

ngOnInit() {
    this.watchValueChanges();
    this.setValidationMessages();
}

onRemoveAddress(index: number): void {
    this.removeAddress.emit(index);
}

isShowValidationMessage(controlName: string): boolean {
    return this.validationMessagesMap.get(controlName)!.message && (this as {[index: string]: any})[controlName].touched;
}

private buildAddress() {
    return this.fb.group({
        addressType: 'home',
        country: '',
        city: ['', Validators.required],
        zip: '',
        street1: '',
        street2: ''
    });
}

private watchValueChanges() {
    this.city.valueChanges
        .subscribe(value => this.setValidationMessages());
}

private setValidationMessages() {
    this.validationMessagesMap.forEach((control, cntrlName) => {
        this.buildValidationMessages(cntrlName);
    });
}

private buildValidationMessages(controlName: string) {
    const c: AbstractControl = (this as {[index: string]: any})[controlName]; // вызов
// метода
    this.validationMessagesMap.get(controlName)!.message = '';

    if (c.errors) {
        this.validationMessagesMap.get(controlName)!.message = Object.keys(c.errors)
            .map(key => {
                const value = this.validationMessagesMap.get(controlName)!.message;
                return (value as {[index: string]: any})[key];
            })
            .join(' ');
    }
}

```

```
// ***** CONTROL_VALUE_ACCESSOR INTERFACE METHODS ***** /

public onTouched: () => void = () => {};

// model => DOM
writeValue(val: any): void {
  if (val) {
    this.addressInfoForm.setValue(val, { emitEvent: false });
  }
}

// DOM => model
registerOnChange(fn: any): void {
  console.log('on change');
  this.addressInfoForm.valueChanges.subscribe(fn);
}

registerOnTouched(fn: any): void {
  console.log('on blur');
  this.onTouched = fn;
}

setDisabledState?(isDisabled: boolean): void {
  isDisabled ? this.addressInfoForm.disable() : this.addressInfoForm.enable();
}

validate(c: AbstractControl): ValidationErrors | null {
  console.log('Adress Info validation', c);
  return this.addressInfoForm.valid
    ? null
    : {
        invalidForm: {
          valid: false,
          message: 'addressInfoForm fields are invalid'
        }
      };
}
}
```

3. Replace the content of **AddressInfoComponent Template**. Use the following snippet of HTML:

```
<div [formGroup]="addressInfoForm">
  <div class="form-group" >
    <label class="col-md-2 control-label">Address Type</label>
    <div class="col-md-7">
      <label class="radio-inline">
        <input type="radio" id="{{'addressType1Id' + i}}" value="home"
          formControlName="addressType">Home
      </label>
      <label class="radio-inline">
        <input type="radio" id="{{'addressType2Id' + i}}" value="work"
          formControlName="addressType">Work
      </label>
      <label class="radio-inline">
        <input type="radio" id="{{'addressType3Id' + i}}" value="other"
          formControlName="addressType">Other
      </label>
    </div>
  </div>
</div>
```

```

        </label>
    </div>
    <div class="col-md-1 text-right" *ngIf="i>0">
        <button class="btn btn-danger" (click)="onRemoveAddress(i)">X</button>
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label"
        attr.for="{{ 'countryId' + i }}">Country, City, Zip Code</label>

    <div class="col-md-3">
        <select class="form-control"
            id="{{ 'countryId' + i }}"
            formControlName="country">
            <option value="">Select a Country...</option>
            <option *ngFor="let country of countries"
                [value]="country">{{country}}</option>
        </select>
    </div>
    <div class="col-md-3"
        [ngClass]="{'has-error': isShowValidationMessage('city')}" >
        <input type="text"
            class="form-control"
            id="{{ 'cityId' + i }}"
            placeholder="City"
            formControlName="city">
        <span class="help-block" *ngIf=" isShowValidationMessage('city')">
            {{validationMessagesMap.get('city')?.message}}
        </span>
    </div>
    <div class="col-md-2">
        <input type="number"
            class="form-control"
            id="{{ 'zipId' + i }}"
            placeholder="Zip Code"
            formControlName="zip">
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label"
        attr.for="{{ 'street1Id' + i }}">Street Address 1</label>
    <div class="col-md-8">
        <input type="text"
            class="form-control"
            id="{{ 'street1Id' + i }}"
            placeholder="Street address"
            formControlName="street1">
    </div>
</div>

<div class="form-group">
    <label class="col-md-2 control-label"
        attr.for="{{ 'street2Id' + i }}">Street Address 2</label>
    <div class="col-md-8">
        <input type="text"

```



```

        class="form-control"
        id="{{ 'street2Id' + i }}"
        placeholder="Street address (second line)"
        formControlName="street2">
    </div>
</div>
</div>

```

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

```

// 1
import { AddressInfoComponent } from './components/address-info/address-info.component';

imports: [ReactiveFormsModule, NgIf, NgForOf, NgClass, JsonPipe, ServiceLevelDirective,
AsyncEmailValidatorDirective, AddressInfoComponent],

// 1
countries: Array<string> = COUNTRIES;

// 2
private buildAddress() {
    return this.fb.group({
        addressType: 'home',
        country: '',
        city: '',
        zip: '',
        street1: '',
        street2: ''
    });
    return this.fb.control('');
}

```

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

```

// 1
<div *ngFor="let address of addresses.controls; let i = index" [formGroupName]="i">
    <div class="form-group">
        <label class="col-md-2 control-label">Address Type</label>
        <div class="col-md-7">
            <label class="radio-inline">
                <input type="radio" id="{{ 'addressType1Id' +
i }}" value="home"
                                formControlName="addressType">Home
            </label>
            <label class="radio-inline">
                <input type="radio" id="{{ 'addressType2Id' +
i }}" value="work"
                                formControlName="addressType">Work
            </label>
            <label class="radio-inline">
                <input type="radio" id="{{ 'addressType3Id' +
i }}" value="other"
                                formControlName="addressType">Other
            </label>
        </div>
    </div>

```

```

        <div class="col-md-1 text-right" *ngIf="i>0">
            <button class="btn btn-danger"
(click)="onRemoveAddress(i)">X</button>
        </div>
    </div>

    <div class="form-group">
        <label class="col-md-2 control-label"
            attr.for="{{ 'countryId' + i }}">Country, City, Zip
Code</label>

        <div class="col-md-3">
            <select class="form-control"
                id="{{ 'countryId' + i }}"
                formControlName="country">
                <option value="">Select a Country...</option>
                <option *ngFor="let country of countries"
                    [value]="country">{{country}}</option>
            </select>
        </div>
        <div class="col-md-3">
            <input type="text"
                class="form-control"
                id="{{ 'cityId' + i }}"
                placeholder="City"
                formControlName="city">
        </div>
        <div class="col-md-2">
            <input type="number"
                class="form-control"
                id="{{ 'zipId' + i }}"
                placeholder="Zip Code"
                formControlName="zip">
        </div>
    </div>

    <div class="form-group">
        <label class="col-md-2 control-label"
            attr.for="{{ 'street1Id' + i }}">Street Address 1</label>
        <div class="col-md-8">
            <input type="text"
                class="form-control"
                id="{{ 'street1Id' + i }}"
                placeholder="Street address"
                formControlName="street1">
        </div>
    </div>

    <div class="form-group">
        <label class="col-md-2 control-label"
            attr.for="{{ 'street2Id' + i }}">Street Address 2</label>
        <div class="col-md-8">
            <input type="text"
                class="form-control"
                id="{{ 'street2Id' + i }}"
                placeholder="Street address (second line)"
                formControlName="street2">
        </div>
    </div>

```

```
        </div>
      </div>
    </div>
  </div>
```

```
// 2
<div formArrayName="addresses">
  <app-address-info *ngFor="let address of addresses.controls; let i = index"
    [formControlName]="i"
    [index]="i"
    (removeAddress)="onRemoveAddress($event)">
  </app-address-info>
```