Table of Contents

[Current Resources with Changes 3](#_Toc115736590)

[New Resources Added 3](#_Toc115736591)

[/core/models 3](#_Toc115736592)

[userTokenDto.model.ts 3](#_Toc115736593)

[siteRegister.model.ts 3](#_Toc115736594)

[/core/validators 3](#_Toc115736595)

[only-char.validator.ts 3](#_Toc115736596)

[password-strength.validator.ts 4](#_Toc115736597)

[must-match.validator.ts 5](#_Toc115736598)

[/core/pipes 6](#_Toc115736599)

[validators-transform.pipe.ts 6](#_Toc115736600)

[Alternate Code as function Component 6](#_Toc115736601)

[Add Reactive Forms Module 6](#_Toc115736602)

[/core/modules/shared.modules.ts 6](#_Toc115736603)

[Components 6](#_Toc115736604)

[/site/nav 6](#_Toc115736605)

[/site/errors/test-errors 7](#_Toc115736606)

[test-arrors.component.ts 7](#_Toc115736607)

[/site/errors/display-formgroup-errors 7](#_Toc115736608)

[display-formgroup-errors.component.ts 7](#_Toc115736609)

[display-formgroup-errors.component.html 7](#_Toc115736610)

[/site/\_form-controls [Reusable Controls] 7](#_Toc115736611)

[text-input Reusable Form Controls 7](#_Toc115736612)

[date-input Reusable Form Control 8](#_Toc115736613)

[/site/members/photo-editor 9](#_Toc115736614)

[photo-editor.component.ts 9](#_Toc115736615)

[/site/site-register 10](#_Toc115736616)

[site-register.component.ts 10](#_Toc115736617)

[site-register.component.html local implementation [DO NOT USE] 13](#_Toc115736618)

[site-register.component.html 15](#_Toc115736619)

# Current Resources with Changes

1. /core/modules/shared.modules
2. /site/site-register
3. /core/models/userTokenDto
4. /core/models/siteRegisterDto
5. /site/nav
6. /site/errors/test-errors
7. /site/members/photo-editor

# New Resources Added

1. /core/validators
2. /core/pipes/validators-transform.pipe.ts
3. /site/\_form-controls/text-input Reusable Form Controls
4. /site/errors/displayFormgroupErrors

# /core/models

## userTokenDto.model.ts

Add additional property for the display Name

export class UserTokenDto {

    constructor(public userName: string = "",

        public guId: string = "",

        public token: string = "",

        public mainPhotoUrl = "",

        public displayName = "") { }

}

## siteRegister.model.ts

Add the additional properties

export class SiteRegisterDto {

    constructor(public userName: string = "",

        public password: string = "",

        public gender: string = "",

        public displayName: string = "",

        public dateOfBirth: Date,

        public city: string = "",

        public country: string = ""

     ) {}

}

# /core/validators

Create a new folder under /core and name it validators

## only-char.validator.ts

This validator has two validators created under same class since the core functionality is the same. These are

1. onlyChar
2. onlycharWithSpace

import { AbstractControl, ValidatorFn } from "@angular/forms";

/\*

1. only one input argument is expected, which is of type AbstractControl

2. the validator function can obtain the value to be validated via the control.value property

3. the validator function needs to return null if no errors were found in the field value, meaning that the value is valid

4. if any validation errors are found, the function needs to return an object of type ValidationErrors

5. the ValidationErrors object can have as properties the multiple errors found (usually just one), and as values the details about each error.

6. the value of the ValidationErrors object can be an object with any properties that we want, allowing us to provide a lot of useful information about the error if needed

\*/

export class OnlyCharValidator{

    constructor() { }

    //to check: controlName.errors?.onlyChar

    static onlyChar(): ValidatorFn {

        return (control: AbstractControl): { [key: string]: boolean } | null => {

            const value = control.value;

            //empty good

            if (value === '') return null;

            const regEx = new RegExp('^[a-zA-z]\*$');

            const isChars = regEx.test(value);

            //match good

            if (isChars) return null;

            //no match bad

            return { onlyChar: true };

        };

    }

    //to check: controlName.errors?.onlyCharWithSpace

    static onlyCharWithSpace(): ValidatorFn {

        return (control: AbstractControl): { [key: string]: boolean } | null => {

            const value = control.value;

            //empty good

            if (value === '') return null;

            const regEx = new RegExp('^[a-zA-z ]\*$');

            const isChars = regEx.test(value);

            //match good

            if (isChars) return null;

            //no match bad

            return { onlyCharWithSpace: true };

        };

    }

}

## password-strength.validator.ts

import { AbstractControl, ValidationErrors, ValidatorFn } from "@angular/forms";

/\*

1. only one input argument is expected, which is of type AbstractControl

2. the validator function can obtain the value to be validated via the control.value property

3. the validator function needs to return null if no errors were found in the field value, meaning that the value is valid

4. if any validation errors are found, the function needs to return an object of type ValidationErrors

5. the ValidationErrors object can have as properties the multiple errors found (usually just one), and as values the details about each error.

6. the value of the ValidationErrors object can be an object with any properties that we want, allowing us to provide a lot of useful information about the error if needed

picked from blog angular

\*/

export class PasswordStrengthValidator{

    constructor() { }

    //to check: password.errors?.passwordStrengthValidator

    static passwordStrength(): ValidatorFn {

        return (control: AbstractControl): ValidationErrors | null => {

            const value = control.value;

            //empty good

            if (!value) return null;

            const hasUpperCase = /[A-Z]+/.test(value);

            const hasLowerCase = /[a-z]+/.test(value);

            const hasNumeric = /[0-9]+/.test(value);

            const passwordValid = hasUpperCase && hasLowerCase && hasNumeric;

            //good

            if (passwordValid) return null;

            //bad

            //return { passwordStrengthValidator: true };

            return {

                passwordStrength: {

                    hasUpperCase: hasUpperCase,

                    hasLowerCase: hasLowerCase,

                    hasNumeric: hasNumeric

                }

            };

        };

    }

}

## must-match.validator.ts

import { AbstractControl, FormGroup, ValidationErrors, ValidatorFn } from "@angular/forms";

/\*

1. only one input argument is expected, which is of type AbstractControl

2. the validator function can obtain the value to be validated via the control.value property

3. the validator function needs to return null if no errors were found in the field value, meaning that the value is valid

4. if any validation errors are found, the function needs to return an object of type ValidationErrors

5. the ValidationErrors object can have as properties the multiple errors found (usually just one), and as values the details about each error.

6. the value of the ValidationErrors object can be an object with any properties that we want, allowing us to provide a lot of useful information about the error if needed

used for any thing where we need to match two fields like password and confirm password

\*/

export class MustMatchValidator{

    constructor() { }

    //apply at the formGroup level: MustMatchValidator.mustMatch('password', 'confirmPassword'

    //to check: password.errors?.mismatch

    static mustMatch(source: string, target: string) : ValidatorFn {

        return (control: AbstractControl) : ValidationErrors | null => {

            const sourceControl = control.get(source);

            const targetControl = control.get(target);

            if (!targetControl || !sourceControl)

                return null;

            if (targetControl.errors && !targetControl.errors["mustMatch"])

                return null;

            //check

            if (sourceControl && targetControl && sourceControl.value !== targetControl.value) {

                //return { mismatch: true };

                targetControl.setErrors({ mustMatch: true });

            }

            else {

                targetControl.setErrors(null);

            }

            return null;

        };

    }

}

# /core/pipes

## validators-transform.pipe.ts

Create a pipe with the command

* ng generate pipe /core/pipes/validatorsTransform --module=/core/modules/shared.module.ts --skip-tests

This will also add the pipe to the declarations array of the /core/modules/shared.module.ts

import { Pipe, PipeTransform } from '@angular/core';

@Pipe({

  name: 'validatorsTransform'

})

export class ValidatorsTransformPipe implements PipeTransform {

  transform(value: any, ...args: any[]): any {

    if (!value) return value;

    return Object.keys(value);

  }

}

### Alternate Code as function Component

Alternatively the pipe can also be written as a function in the component.ts file. Pipe is better approach since the code is reusable

  getErrorList(errorObject: any) {

    if (!errorObject)

      return [];

    return Object.keys(errorObject);

  }

# Add Reactive Forms Module

## /core/modules/shared.modules.ts

Add to both imports and exports array ReactiveFormsModule.

Also make sure that it gets added to the imports from @angular/forms

# Components

## /site/nav

nav.component.html

* use displayName to display the logged in users name in the nav
* also, when mainphoto is missing then display the user.png as default

<button type="button" class="btn btn-info">

     <img src="{{ userInfo.mainPhotoUrl || './assets/user.png' }}" alt="{{ userInfo.userName | titlecase }}" class="me-2">

                        Welcome {{userInfo.displayName || userInfo.userName | titlecase}}</button>

## /site/errors/test-errors

### test-arrors.component.ts

Change the first line under function getValidationError to fill in additional properties

  getValidationError() {

    const register: SiteRegisterDto = new SiteRegisterDto("", "pas", "", "", new Date(), "", "");

## /site/errors/display-formgroup-errors

Displays the form group validators that do not pass.

* ng g c /site/errors/displayFormgroupErrors --module=/core/modules/shared.module.ts --skip-tests

### display-formgroup-errors.component.ts

import { Component, Input, OnInit } from '@angular/core';

import { ValidationErrors } from '@angular/forms';

@Component({

  selector: 'app-display-formgroup-errors',

  templateUrl: './display-formgroup-errors.component.html',

  styleUrls: ['./display-formgroup-errors.component.css']

})

export class DisplayFormgroupErrorsComponent implements OnInit {

  @Input() key!: string;

  @Input() errors: ValidationErrors | null | undefined;

  constructor() { }

  ngOnInit(): void {

  }

}

### display-formgroup-errors.component.html

<div class="" \*ngIf="key && errors">

    <ul class="list-inline">

        <li class="list-inline-item"><strong>{{ key }} Validator(s): </strong></li>

        <li class="list-inline-item" \*ngFor="let item of errors | validatorsTransform">{{ item }}</li>

    </ul>

</div>

## /site/\_form-controls [Reusable Controls]

### text-input Reusable Form Controls

This is a reusable input control which can be used in all the forms

* ng g c /site/\_form-controls/textInput --module=/core/modules/shared.module.ts --skip-tests

This will also add to the /core/modules/shared.module.ts

#### text-input.component.ts

import { Component, Input, Self } from '@angular/core';

import { ControlValueAccessor, NgControl } from '@angular/forms';

@Component({

  selector: 'app-text-input',

  templateUrl: './text-input.component.html',

  styleUrls: ['./text-input.component.css']

})

export class TextInputComponent implements ControlValueAccessor {

  @Input() label: string = '';

  @Input() labelMustMatch: string = '';

  @Input() placeHolder: string = '';

  @Input() type: string = 'text';

  constructor(@Self() public ngControl: NgControl) {

    this.ngControl.valueAccessor = this;

  }

  writeValue(obj: any): void {}

  registerOnChange(fn: any): void {}

  registerOnTouched(fn: any): void {}

  //setDisabledState?(isDisabled: boolean): void {}

}

#### text-input.component.html

<!-- rf['userName'].errors && (rf['userName'].touched && rf['userName'].dirty) -->

<!--use $any for the form control or it will complain about

        Type 'AbstractControl | null' is not assignable to type 'FormControl'.Type 'null' is not assignable to type 'FormControl'

-->

<input

[class.is-invalid]="ngControl.invalid && ngControl.touched && ngControl.dirty"

type="{{ type }}"

class="form-control"

[formControl]="$any(ngControl.control)"

placeholder="{{ placeHolder }}"

>

<!--required-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['required']">

    {{label}} is required

</div>

<!--minlength-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['minlength']">

    {{label}} must be minimum {{ngControl.control?.errors?.['minlength']['requiredLength']}} characters

</div>

<!--maxlength-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['maxlength']">

    {{label}} must be maximum {{ngControl.control?.errors?.['maxlength']['requiredLength']}} characters

</div>

<!--onlyChar-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['onlyChar']">

    {{label}} must be only characters

</div>

<!--onlyCharWithSpace-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['onlyCharWithSpace']">

    {{label}} must be only characters and space

</div>

<!--mustMatch-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['mustMatch']">

    {{labelMustMatch}} do not match

</div>

<!--passwordStrength-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['passwordStrength']">

    {{label}} must have an upper case, lower case and a number

</div>

### date-input Reusable Form Control

* ng g c /site/form-controls/dateInput --module=/core/modules/shared.module.ts --skip-tests

the component will get added to the shared module declarations array

#### date-input.component.ts

import { Component, Input, Self } from '@angular/core';

import { ControlValueAccessor, NgControl } from '@angular/forms';

import { BsDatepickerConfig } from 'ngx-bootstrap/datepicker';

@Component({

  selector: 'app-date-input',

  templateUrl: './date-input.component.html',

  styleUrls: ['./date-input.component.css']

})

export class DateInputComponent implements ControlValueAccessor {

  @Input() label: string = '';

  @Input() placeHolder: string = '';

  @Input() maxDate!: Date;

  //Partial means every property inside BsDatepickerConfig is optional.

  bsConfig!: Partial<BsDatepickerConfig>;

  constructor(@Self() public ngControl: NgControl) {

    this.ngControl.valueAccessor = this;

    this.bsConfig = {

      containerClass: 'theme-red',

      dateInputFormat: 'YYYY-MM-DD',

    }

  }

  writeValue(obj: any): void {

  }

  registerOnChange(fn: any): void {

  }

  registerOnTouched(fn: any): void {

  }

}

#### date-input.component.html

<input

type="text"

[class.is-invalid]="ngControl.invalid && ngControl.touched"

class="form-control"

[formControl]="$any(ngControl.control)"

placeholder="{{ placeHolder }}"

bsDatepicker

[bsConfig]="bsConfig"

[maxDate]="maxDate"

>

<!--required-->

<div class="invalid-feedback" \*ngIf="ngControl.control?.errors?.['required']">

    {{label}} is required

</div>

## /site/members/photo-editor

### photo-editor.component.ts

We have a bug where the phot is not updated in al places when the first photo for the user gets uploaded.

Here is the updated photo-editor success method

    this.uploader.onSuccessItem = (item, response, status, headers) => {

      if (response) {

        const photo = <PhotoDto>JSON.parse(response);

        this.member.photos.push(photo);

        if (photo.isMain) {

          this.currentUser.mainPhotoUrl = photo.url;

          this.member.photoUrl = photo.url;

          this.accountService.setAndFireCurrentUser(this.currentUser);

        }

      }

    }

## /site/site-register

This was originally built with template driven form. Now it is redone with reactive forms. All the original pieces are in there.

* The register method has been renamed
* The html is being conditionally put depending upon reactiveForm is enabled or disabled.

### site-register.component.ts

import { Component, EventEmitter, OnDestroy, OnInit, Output } from '@angular/core';

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

import { Router } from '@angular/router';

import { Subscription } from 'rxjs';

import { ToastrService } from 'ngx-toastr';

import { SiteRegisterDto } from '../../core/models/siteRegisterDto.model';

import { environment } from '../../../environments/environment';

import { AccountService } from '../../core/services/account.service';

import { ErrorMessageService } from '../../core/services/error-message.service';

import { MustMatchValidator } from '../../core/validators/must-match.validator';

import { PasswordStrengthValidator } from '../../core/validators/password-strength.validator';

import { OnlyCharValidator } from '../../core/validators/only-char.validator';

@Component({

  selector: 'app-site-register',

  templateUrl: './site-register.component.html',

  styleUrls: ['./site-register.component.css']

})

export class SiteRegisterComponent implements OnInit, OnDestroy {

  //to tell the site home to hide the register form since cancel has been clicked

  @Output() cancelRegister = new EventEmitter();

  //reactive form

  registerForm!: FormGroup;

  isReactiveForm: boolean = true;

  siteRegister: SiteRegisterDto = <SiteRegisterDto>{};

  //note use of ! or will see a compiler error

  registerSubscription!: Subscription;

  registerReactiveSubscription!: Subscription;

  //after implementation on error interceptor

  validationErrors: string[] = [];

  //must be minimum 18 years old

  maxDate!: Date;

  //gender list

  genderList = [

    { id: 'female', value: 'female', label: 'Female' },

    { id: 'male', value: 'male', label: 'Male' },

  ];

  constructor(private accountService: AccountService,

    private errorMsgService: ErrorMessageService,

    private toastrService: ToastrService,

    private fb: FormBuilder,

    private router: Router) { }

  ngOnInit(): void {

    //reactive form

    this.initializeForm();

    this.maxDate = new Date();

    this.maxDate.setFullYear(this.maxDate.getFullYear() - 18);

  }

  ngOnDestroy(): void {

    if (this.registerSubscription) this.registerSubscription.unsubscribe();

    if (this.registerReactiveSubscription) this.registerReactiveSubscription.unsubscribe();

  }

  //reactive form

  initializeForm() {

    //we have a formGroup and formFroup has formControl

    /\*

    this.registerForm = new FormGroup({

      userName: new FormControl('', [Validators.required, Validators.minLength(5), OnlyCharValidator.onlyChar()]),

      password: new FormControl('', [Validators.required, Validators.minLength(4), Validators.maxLength(8), PasswordStrengthValidator.passwordStrength()]),

      confirmPassword: new FormControl('', Validators.required)

      },

      MustMatchValidator.mustMatch("password", "confirmPassword")

    );

    \*/

    this.registerForm = this.fb.group({

      gender: ['male', Validators.required],

      displayName: ['', [Validators.required, Validators.minLength(5)]],

      dateOfBirth: ['', Validators.required],

      city: ['', [Validators.required, OnlyCharValidator.onlyChar()]],

      country: ['', [Validators.required, OnlyCharValidator.onlyChar()]],

      userName: ['', [Validators.required, Validators.minLength(5), OnlyCharValidator.onlyChar()]],

      password: ['', [Validators.required, Validators.minLength(4), Validators.maxLength(8), PasswordStrengthValidator.passwordStrength()]],

      confirmPassword: ['', Validators.required]

      },

      {validators: [MustMatchValidator.mustMatch("password", "confirmPassword")]} as AbstractControlOptions

    );

    console.log(this.registerForm.controls);

  }

  getErrorList(errorObject: any) {

    if (!errorObject)

      return [];

    return Object.keys(errorObject);

  }

  //convenience getter for easy access to form fields

  get rf() {

    return this.registerForm.controls;

  }

  rf2(key: string) {

    return this.registerForm.get(key) as FormControl;

  }

  get registerFormUsernameControl(): FormControl {

    return this.registerForm.get('userName') as FormControl;

    //return this.rf['userName'] as FormControl;

  }

  get registerFormPasswordControl(): FormControl {

    return this.registerForm.get('password') as FormControl;

    //return this.rf['password'] as FormControl;

  }

  get registerFormConfirmPasswordControl(): FormControl {

    return this.registerForm.get('confirmPassword') as FormControl;

    //return this.rf['confirmPassword'] as FormControl;

  }

  //per template driven form

  onRegisterTemplateDriven() {

    this.validationErrors = [];//reset

    this.doRegisteration(this.siteRegister);

  }

  //show erros on submit

  private showErrorsOnSubmit() {

    Object.keys(this.registerForm.controls).forEach(field => {

      const control = this.registerForm.get(field);

      if (control?.errors)

        control.markAsTouched({onlySelf: true});

    });

  }

  //check reactive form

  private isReactiveFormGood(): boolean {

    if (environment.displayConsoleLog) console.log(this.registerForm.value);

    if (this.registerForm.invalid) {

      this.showErrorsOnSubmit();

      this.toastrService.error("Please fix errors and try again", "Validation Error(s)")

      return false;

    }

    return true;

  }

  //per reactive form

  onRegisterReactiveForm() {

    this.validationErrors = [];//reset

    //when invalid do not proceed further

    if (!this.isReactiveFormGood()) return;

    //convert to siteRegister

    const registerUser = new SiteRegisterDto(this.registerForm.value['userName'],

                                            this.registerForm.value['password'],

                                            this.registerForm.value['gender'],

                                            this.registerForm.value['displayName'],

                                            this.registerForm.value['dateOfBirth'],

                                            this.registerForm.value['city'],

                                            this.registerForm.value['country']);

    this.doRegisteration(registerUser)

  }

  private doRegisteration(registerUser: SiteRegisterDto) {

    if (environment.displayConsoleLog) console.log(registerUser);

    //register

    this.registerSubscription = this.accountService.register(registerUser).subscribe({

      next: r => {

        if (environment.displayConsoleLog) {

          console.log("RegisterUserBack: ");

          console.log(r);

        }

        //go to the members page

        this.router.navigateByUrl('/members/list');

      }, error: e => {

        //due to error intercepter we are getting a flat array of validation items so for modal validation need to check that

        //check array and length > 0

        //other cases the error interceptor is displaying the error

        if (e?.length) {

          if (environment.displayConsoleLog) console.log("\*\*\*inside model validation errors\*\*\*");

          this.validationErrors = e;

        }

        if (environment.displayConsoleLog) console.log(e);

        //this.displayError(e, "Registeration");

      }, complete: () => {

      }

    });

  }

  private displayError(error: any, from: string) {

    const errormsg = this.errorMsgService.getHttpErrorMessage(error);

    if(environment.displayConsoleLog) console.log(`displayError-${from} Error: ${errormsg}`);

    //alert(`displayError-${from} Error: ${errormsg}`);

    this.toastrService.error(errormsg);

  }

  onCancel() {

    this.validationErrors = [];//reset

    if (environment.displayConsoleLog) console.log('cancelled');

    this.cancelRegister.emit(false);

    //reset form

    this.siteRegister = <SiteRegisterDto>{};

  }

}

### site-register.component.html local implementation [DO NOT USE]

Everything is local in the html. This is too much code. Later will use the reusable form controls as created above in the html.

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OLD TEMPLATE DRIVEN FORM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<ng-container \*ngIf="!isReactiveForm">

    <!--template driven form-->

    <form #registerForm="ngForm" (ngSubmit)="onRegisterTemplateDriven()" autocomplete="off">

        <h2 class="text-center text-primary">Sign up</h2>

        <hr>

        <div class="form-group mt-2">

            <input [(ngModel)]="siteRegister.userName" #username="ngModel" required minlength="5" placeholder="username" id="username" name="username" type="text" class="form-control">

        </div>

        <div class="form-group mt-2">

            <input [(ngModel)]="siteRegister.password" #password="ngModel" required placeholder="password" id="password" name="password" type="password" class="form-control">

        </div>

        <div class="form-group text-center mt-2">

            <button class="btn btn-success mr-2" type="submit" [disabled]="username.invalid || password.invalid">Register</button>&nbsp;

            <button class="btn btn-danger" (click)="onCancel()" type="button">Cancel</button>

        </div>

        <div class="row mt-5" \*ngIf="validationErrors && validationErrors.length > 0">

            <ul class="text-danger">

                <li \*ngFor="let error of validationErrors">{{ error }}</li>

            </ul>

        </div>

    </form>

</ng-container>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NEW REACTIVE FORM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<ng-container \*ngIf="isReactiveForm && registerForm">

    <!--Reactive Form-->

    <form [formGroup]="registerForm" (ngSubmit)="onRegisterReactiveForm()" autocomplete="off">

        <h2 class="text-center text-primary">Sign up</h2>

        <hr>

        <div class="form-group mt-2">

            <!--

                we can eihter use registerForm.get("userName")

                or can use convenience getter rf which stands for registerForm

                in this case not using rf['userName'].pristine, which means the value has not yet changed in the UI

            -->

            <input [class.is-invalid]="rf['userName'].errors && (rf['userName'].touched && rf['userName'].dirty)"

                type="text" class="form-control" formControlName="userName"  placeholder="username">

            <div class="invalid-feedback" \*ngIf="rf['userName'].errors && (rf['userName'].touched || rf['userName'].dirty) && rf['userName'].errors['required']">User name is required</div>

            <div class="invalid-feedback" \*ngIf="rf['userName'].errors && (rf['userName'].touched || rf['userName'].dirty) && !(rf['userName'].errors['required'] || rf['userName'].errors['onlyChar'])">Username must be minimum 5 characters</div>

            <div class="invalid-feedback" \*ngIf="rf['userName'].errors && (rf['userName'].touched || rf['userName'].dirty) && rf['userName'].errors['onlyChar']">Username must be only characters</div>

        </div>

        <div class="form-group mt-2">

            <input [class.is-invalid]="rf['password'].errors && (rf['password'].touched && rf['password'].dirty)"

                type="password" class="form-control" formControlName="password" placeholder="password" maxlength="8">

            <div class="invalid-feedback" \*ngIf="rf['password'].errors && (rf['password'].touched || rf['password'].dirty) && rf['password'].errors['required']">Password is required</div>

            <div class="invalid-feedback" \*ngIf="rf['password'].errors && (rf['password'].touched || rf['password'].dirty) && !(rf['password'].errors['required'] || rf['password'].errors['passwordStrength'])">Password must be between 5 and 8 characters long</div>

            <div class="invalid-feedback" \*ngIf="rf['password'].errors && (rf['password'].touched || rf['password'].dirty) && rf['password'].errors['passwordStrength']">Password must have an upper case, lower case and a number</div>

        </div>

        <div class="form-group mt-2">

            <input [class.is-invalid]="rf['confirmPassword'].errors && (rf['confirmPassword'].touched && rf['confirmPassword'].dirty)"

                type="password" class="form-control" formControlName="confirmPassword" placeholder="confirm password" maxlength="8">

            <div class="invalid-feedback" \*ngIf="rf['confirmPassword'].errors && (rf['confirmPassword'].touched || rf['confirmPassword'].dirty) && rf['confirmPassword'].errors['required']">Confirm Password is required</div>

            <div class="invalid-feedback" \*ngIf="rf['confirmPassword'].errors && (rf['confirmPassword'].touched || rf['confirmPassword'].dirty) && rf['confirmPassword'].errors['mustMatch']">Password and Confirm do not match</div>

        </div>

        <!-- [disabled]="registerForm.invalid"-->

        <div class="form-group text-center mt-2">

            <button class="btn btn-success me-2" type="submit">Register</button>

            <button class="btn btn-danger" (click)="onCancel()" type="button">Cancel</button>

        </div>

        <!--########## Validators being assigned in real time ##########-->

        <div class="card mt-2" \*ngIf="rf['userName'].errors || rf['password'].errors || rf['confirmPassword'].errors">

            <div class="card-header">Validator(s) being assinged to the controls by angular</div>

            <div class="card-body">

                <app-display-formgroup-errors [key]="'UserName'" [errors]="rf['userName'].errors"></app-display-formgroup-errors>

                <app-display-formgroup-errors [key]="'Password'" [errors]="rf['password'].errors"></app-display-formgroup-errors>

                <app-display-formgroup-errors [key]="'ConfirmPassword'" [errors]="rf['confirmPassword'].errors"></app-display-formgroup-errors>

            </div>

        </div>

        <div class="row mt-5" \*ngIf="validationErrors && validationErrors.length > 0">

            <ul class="text-danger">

                <li \*ngFor="let error of validationErrors">{{ error }}</li>

            </ul>

        </div>

    </form>

    <div class="card mt-2">

        <div class="card-header">Instructions</div>

        <div class="card-body">

            <ul>

                <li>Username must be only characters</li>

                <li>Username must be minimum 5 characters</li>

                <li>Password must be between 4 and 8 characters</li>

                <li>Password must have an upper case, lower case and a number in it</li>

            </ul>

        </div>

    </div>

</ng-container>

### site-register.component.html

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* OLD TEMPLATE DRIVEN FORM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<ng-container \*ngIf="!isReactiveForm">

    <!--template driven form-->

    <form #registerForm="ngForm" (ngSubmit)="onRegisterTemplateDriven()" autocomplete="off">

        <h2 class="text-center text-primary">Sign up</h2>

        <hr>

        <div class="form-group mt-2">

            <input [(ngModel)]="siteRegister.userName" #username="ngModel" required minlength="5" placeholder="username" id="username" name="username" type="text" class="form-control">

        </div>

        <div class="form-group mt-2">

            <input [(ngModel)]="siteRegister.password" #password="ngModel" required placeholder="password" id="password" name="password" type="password" class="form-control">

        </div>

        <div class="form-group text-center mt-2">

            <button class="btn btn-success mr-2" type="submit" [disabled]="username.invalid || password.invalid">Register</button>&nbsp;

            <button class="btn btn-danger" (click)="onCancel()" type="button">Cancel</button>

        </div>

        <div class="row mt-5" \*ngIf="validationErrors && validationErrors.length > 0">

            <ul class="text-danger">

                <li \*ngFor="let error of validationErrors">{{ error }}</li>

            </ul>

        </div>

    </form>

</ng-container>

<!--\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* NEW REACTIVE FORM \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*-->

<ng-container \*ngIf="isReactiveForm && registerForm">

    <!--Reactive Form-->

    <form [formGroup]="registerForm" (ngSubmit)="registerForm.valid && onRegisterReactiveForm()" autocomplete="off">

        <h2 class="text-center text-primary">Sign up</h2>

        <hr>

        <!--

            we can eihter use registerForm.get("userName")

            or can use convenience getter rf which stands for registerForm

            in this case not using rf['userName'].pristine, which means the value has not yet changed in the UI

        -->

        <fieldset class="reset">

            <legend class="reset">Who you are?</legend>

                <div class="form-group">

                    <label class="control-label" style="margin-right: 10px;">I am a: </label>

                    <label class="radio-inline" \*ngFor="let gender of genderList; let i = index;">

                        <!--apply the marging to the left side of the second radio button :) -->

                        <input type="radio" id="gender-{{gender.id}}" value="{{gender.value}}" formControlName="gender" [class.ms-3]="i == 1"> {{gender.label}}

                    </label>

                </div>

                <div class="form-group mt-2"><app-text-input [formControl]="rf2('displayName')" [label]="'Display name'" [placeHolder]="'display name'"></app-text-input></div>

                <div class="form-group mt-2"><app-date-input [formControl]="rf2('dateOfBirth')" [label]="'Date Of Birth'" [placeHolder]="'date of birth MM/DD/YYYY'" [maxDate]="maxDate"></app-date-input></div>

                <div class="form-group mt-2"><app-text-input [formControl]="rf2('city')" [label]="'city'" [placeHolder]="'city'"></app-text-input></div>

                <div class="form-group mt-2"><app-text-input [formControl]="rf2('country')" [label]="'country'" [placeHolder]="'country'"></app-text-input></div>

        </fieldset>

        <fieldset class="mt-2 reset">

            <legend class="reset">Login info?</legend>

                <div class="form-group"><app-text-input [formControl]="rf2('userName')" [label]="'User name'" [placeHolder]="'user name'"></app-text-input></div>

                <div class="form-group mt-2"><app-text-input [formControl]="rf2('password')" [type]="'password'" [label]="'Password'" [placeHolder]="'password'"></app-text-input></div>

                <div class="form-group mt-2"><app-text-input [formControl]="rf2('confirmPassword')" [type]="'password'" [label]="'Confirm password'" [placeHolder]="'confirm password'" [labelMustMatch]="'Password and confirm password'"></app-text-input></div>

        </fieldset>

        <div class="form-group text-center mt-2">

            <button class="btn btn-success me-2" type="submit" [disabled]="registerForm.invalid">Register</button>

            <button class="btn btn-danger" (click)="onCancel()" type="button">Cancel</button>

        </div>

        <!--########## Validators being assigned in real time ##########-->

        <div class="card mt-2">

            <div class="card-header">Validator(s) being assinged to the controls by angular</div>

            <div class="card-body">

                <!--<app-display-formgroup-errors [key]="'UserName'" [errors]="rf['userName'].errors"></app-display-formgroup-errors>-->

                <div \*ngFor="let item of registerForm.controls | validatorsTransform">

                    <app-display-formgroup-errors [key]="item" [errors]="rf[item].errors"></app-display-formgroup-errors>

                </div>

            </div>

        </div>

        <div class="row mt-5" \*ngIf="validationErrors && validationErrors.length > 0">

            <ul class="text-danger">

                <li \*ngFor="let error of validationErrors">{{ error }}</li>

            </ul>

        </div>

    </form>

    <div class="card mt-2">

        <div class="card-header">Instructions</div>

        <div class="card-body">

            <ul>

                <li>Username must be only characters</li>

                <li>Username must be minimum 5 characters</li>

                <li>Password must be between 4 and 8 characters</li>

                <li>Password must have an upper case, lower case and a number in it</li>

            </ul>

        </div>

    </div>

</ng-container>