# Project

* Site-11-Api-Ng-reactiveforms-reusable-input-validators
  + Copied from: Site-10-Api-Ng-upload-photos-cloudinary
* For the “MySocialConnect-SPA” issue npm install command to install all packages
  + ng serve : to run the spa

# New & Updates

|  |  |
| --- | --- |
| New | Updates |
| /core/models-interfaces/spinner-not-allowed.model.ts | /core/modules/shared.module.ts |
| /core/validators/input-validators.ts | /site/register/register-reactiveform.component |
| /site/register-reactiveform-reusablecontrols.component | /site/home.component |
|  | styles.css |
|  | /core/services/helper.service.ts |
|  | /core/modules/shared.module.ts |
|  | /core/models-interfaces/user-register-dto.models.ts |
|  |  |
|  |  |
|  |  |

# Clear Database

If you want to clear database then drop it and recreate it

dotnet ef database drop

dotnet ef database update

# Important

1. <https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/>
2. <https://blog.angular-university.io/angular-custom-validators/> without service and using validators
3. <https://www.thisdot.co/blog/using-custom-async-validators-in-angular-reactive-forms>
4. also check TemplateFormValidatorsService

Check [003 Project4 -Ng-login-reg-nav-services-templateforms.docx](003%20Project4%20-Ng-login-reg-nav-services-templateforms.docx) as well.

# ReactiveFormsModule

Make sure that ReactiveFormsModule is added to the imports/exports array of the /core/modules/shared.module.ts

# styles.css

## Change from

.ng-valid[required], .ng-valid.required, .ng-valid:not(form)  {

    border-left: 5px solid #42A948; /\* green \*/

}

.ng-invalid:not(form)  {

    border-left: 5px solid #a94442; /\* red \*/

}

## Change to

input.ng-valid[required], input.ng-valid.required, input.ng-valid:not(form)  {

  border-left: 5px solid #42A948; /\* green \*/

}

input.ng-invalid:not(form)  {

  border-left: 5px solid #a94442; /\* red \*/

}

# /core/models-interfaces

## user-register-dto.model.ts

Add the additional properties

export class UserRegisterDto {

    constructor(public userName: string = "",

        public password: string = "",

        public confirmPassword: string = "",

        public gender: string = "",

        public displayName: string = "",

        public dateOfBirth: Date,

        public city: string = "",

        public country: string = ""

     ) {}

}

# Loading Interceptor – Not Allowed Paths Implementation

## /core/models-interfaces/spinner-not-allowed.models.ts

Class to hold the urls for which the spinner is not allowed.

export class SpinnerNotAllowed {

    constructor(public url: string = "", public method: string = ""){}

}

## /core/services/helper.service.ts

Create array of not allowed paths and implement a method to check and return false if spinner is not allowed.

  //paths for which the spinner is not needed

  private spinnerNotAllowedUrls: SpinnerNotAllowed[] = [

    new SpinnerNotAllowed(`${this.urlAccount}/checkUser/`, 'Get')

  ];

  public isSpinnerallowed(url: string, method: string): boolean{

    let isSpinnerAllowed: boolean = true;

    if(!this.spinnerNotAllowedUrls || this.spinnerNotAllowedUrls.length <= 0) return isSpinnerAllowed;

    if(!url) return isSpinnerAllowed;

    for(let item of this.spinnerNotAllowedUrls){

      if(item.url.toLowerCase().includes(url.toLowerCase()) && item.method.toLowerCase() === method.toLowerCase()){

        isSpinnerAllowed = false;

        break;

      }

    }

    return isSpinnerAllowed;

  }

## /core/interceptors/loading.interceptor.ts

Implement isSpinnerallowed

import { Injectable } from '@angular/core';

import {

  HttpRequest,

  HttpHandler,

  HttpEvent,

  HttpInterceptor

} from '@angular/common/http';

import { Observable, delay, finalize } from 'rxjs';

import { SpinnerBusyService } from '../services/spinner-busy.service';

import { HelperService } from '../services/helper.service';

@Injectable()

export class LoadingInterceptor implements HttpInterceptor {

  constructor(private spinnerBusyService: SpinnerBusyService,

              private helperService: HelperService

  ) {}

  intercept(request: HttpRequest<unknown>, next: HttpHandler): Observable<HttpEvent<unknown>> {

    const isSpinnerAllowed:boolean = this.helperService.isSpinnerallowed(request.url, request.method);

    if(isSpinnerAllowed)

      this.spinnerBusyService.busy();

    //return next.handle(request);

    return next.handle(request).pipe(

      delay(this.helperService.LoadingSpinnerDelayMiliSec),

      finalize(() => {

        if(isSpinnerAllowed)

          this.spinnerBusyService.idle();

      })

    );

  }

}

# Validators

## Local validator

Register reactive form has a local validator code that is commented. Can use like that as well.

  initializeFormWithLocalValidator(){

    this.registerForm = new FormGroup({

      username: new FormControl('', [Validators.required, Validators.minLength(5)]),

      password: new FormControl('', [Validators.required, Validators.minLength(10)]),

      confirmpassword: new FormControl('', [Validators.required, this.matchValues('password')]),

    });

    //above will only do validation when the confirm passwors is changed. We need to do the same when password is changed as well

    this.registerForm.controls['password'].valueChanges.subscribe({

      next: () => this.registerForm.controls['confirmpassword'].updateValueAndValidity()

    });

  }

  //using local validator

  matchValues(matchTo: string): ValidatorFn{

    return (control: AbstractControl) => {

      return control.value === control.parent?.get(matchTo)?.value ? null : {notMatching: true}

    }

  }

## /code/validators/input-validators.ts

All new validators are in this one file

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

import { AccountService } from "../services/account.service";

import { Observable, map, of } from "rxjs";

export class InputValidator{

    constructor(){}

    //two functions to match values, check /site/register-reactiveform.component.ts

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

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

            const sourceControl = control.get(source);

            const targControl = control.get(target);

            if(!sourceControl || !targControl) return null;

            if(targControl.errors && !targControl.hasError('notMatching')) return null;

            if(sourceControl.value !== targControl.value){

                targControl.setErrors({ notMatching: true });

                return ({ notMatching: true });

            }

            targControl.setErrors(null);

            return null;

        }

    }

    static matchValuesTwo(matchTo: string): ValidatorFn{

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

            const matchToControl = control.parent?.get(matchTo);

            if(!matchToControl) return null;

            if(control.errors && !control.hasError('notMatchingTwo')) return null;

            if(control.value !== matchToControl.value){

                matchToControl.setErrors({ notMatchingTwo: true });

                return ({ notMatchingTwo: true });

            }

            matchToControl.setErrors(null);

            return null;

        }

    }

    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 hasSpecialChar = /[\*@!#%&()^~{}]+/.test(value);

            const hasLength = value.toString().length >= 10 && value.toString().length <= 16;

            const valid = hasUpperCase && hasLowerCase && hasNumeric && hasSpecialChar && hasLength;

            if(valid) return null;

            //bad

            //return { passwordStrengthValidator: true };

            return {

                passwordStrength: {

                    hasUpperCase: hasUpperCase,

                    hasLowerCase: hasLowerCase,

                    hasNumeric: hasNumeric,

                    hasSpecialChar: hasSpecialChar,

                    hasLength: hasLength

                }

            };

            \*/

            //if(control.errors && !control.hasError('passwordStrength')) return null;

            //single regex

            const regex = new RegExp(/^(?=.\*[0-9])(?=.\*[a-z])(?=.\*[A-Z])(?=.\*\W)(?!.\* ).{10,16}$/);

            const valid = regex.test(control.value);

            if(!valid){

                control.setErrors({ passwordStrength: true });

                return ({ passwordStrength: true });

            }

            control.setErrors(null);

            return null;

        }

    }

    //async validator

    static userNameExistAsync(accountService: AccountService): AsyncValidatorFn{

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

            if(!control.value) return of(null);

            if(control.errors && !control.hasError("userNameExistAsync")) return of(null);

            return accountService.checkUser(control.value).pipe(

                map((result: boolean) =>{

                    return result ? {userNameExistAsync: true } : null

                })

            );

        }

    }

    static userNameAllowed(): ValidatorFn{

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

            const value = control.value;

            //empty good

            if (!value) return null;

            if(control.errors && !control.hasError('userNameAllowed')) return null;

            const regex = new RegExp(/^[a-zA-Z][A-Za-z0-9]+(?:[\_-][A-Za-z0-9]+)\*$/);

            const valid = regex.test(control.value);

            if(!valid){

                control.setErrors({ userNameAllowed: true });

                return ({ userNameAllowed: true });

            }

            control.setErrors(null);

            return null;

        }

    }

    static onlyAlphaNumeric(): ValidatorFn {

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

            const value = control.value;

            //empty good

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

            //if(control.errors && !control.hasError('onlyAlphaNumeric')) return null;

            const regex = new RegExp(/^[a-zA-z0-9]\*$/);

            const valid = regex.test(control.value);

            if(!valid){

                control.setErrors({ onlyAlphaNumeric: true });

                return ({ onlyAlphaNumeric: true });

            }

            control.setErrors(null);

            return null;

        };

    }

    static onlyChar(): ValidatorFn {

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

            const value = control.value;

            //empty good

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

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

            const valid = regex.test(control.value);

            if(!valid){

                control.setErrors({ onlyChar: true });

                return ({ onlyChar: true });

            }

            control.setErrors(null);

            return null;

        };

    }

    static onlyCharWithSpace(): ValidatorFn {

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

            const value = control.value;

            //empty good

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

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

            const valid = regex.test(control.value);

            if(!valid){

                control.setErrors({ onlyCharWithSpace: true });

                return ({ onlyCharWithSpace: true });

            }

            control.setErrors(null);

            return null;

        };

    }

}

# ReactiveForm Simple

## /site/home/home.component.html

Comment the app register and instead use the register-reactiveform

    <div \*ngIf="registerMode" class="conatiner">

        <div class="row justify-content-center">

            <div class="col-4">

                <!--<app-register (cancelRegister)="onCancelRegisterMode($event)"></app-register>-->

                <app-register-reactiveform (cancelRegister)="onCancelRegisterMode($event)"></app-register-reactiveform>

            </div>

        </div>

    </div>

## /site/register-reactiveform

### Step 1

Make the form to behave just like /site/register/register.component.ts, the template driven form

So copy paste the code from the register.component.ts and register.componenet.html accordingly.

Change the h2 form title

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

### Step 2

#### register-reactiveform.component.ts

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

import { ToastrService } from 'ngx-toastr';

import { Subscription } from 'rxjs';

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

import { HelperService } from '../../core/services/helper.service';

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

import { UserRegisterDto } from '../../core/models-interfaces/user-register-dto.model';

import { InputValidator } from '../../core/validators/input-validators';

@Component({

  selector: 'app-register-reactiveform',

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

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

})

export class RegisterReactiveformComponent implements OnInit, OnDestroy {

  //tell the parent homeComponent that cancel is clicked

  @Output('cancelRegister') cancelRegister = new EventEmitter();

  register: UserRegisterDto = <UserRegisterDto>{};

  //reactive form

  registerForm!: FormGroup;

  registerSubscription!: Subscription;

  constructor(private helperService: HelperService, private accountService: AccountService,

    private toastr: ToastrService){}

  ngOnDestroy(): void {

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

  }

  ngOnInit(): void {

    //this.initializeFormWithLocalValidator();

    this.initializeForm(); //this uses /code/validators

  }

  //convenience getter for easy access to form fields

  get rf() {

    return this.registerForm.controls;

  }

  rf2(key: string) {

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

  }

  initializeFormWithLocalValidator(){

    this.registerForm = new FormGroup({

      username: new FormControl('', [Validators.required, Validators.minLength(5)]),

      password: new FormControl('', [Validators.required, Validators.minLength(10)]),

      confirmpassword: new FormControl('', [Validators.required, this.matchValues('password')]),

    });

    //above will only do validation when the confirm passwors is changed. We need to do the same when password is changed as well

    this.registerForm.controls['password'].valueChanges.subscribe({

      next: () => this.registerForm.controls['confirmpassword'].updateValueAndValidity()

    });

  }

  //using local validator

  matchValues(matchTo: string): ValidatorFn{

    return (control: AbstractControl) => {

      return control.value === control.parent?.get(matchTo)?.value ? null : {notMatching: true}

    }

  }

  //two different approaches used here for the password and conform password match check

  //either use matchValue or matchValueTwo do not use both

  //password strength returns multiple items check back

  initializeForm(){

    this.registerForm = new FormGroup({

        username: new FormControl('',

                                    [Validators.required, Validators.minLength(5), InputValidator.userNameAllowed()],

                                    [InputValidator.userNameExistAsync(this.accountService)]),

        password: new FormControl('', [Validators.required, Validators.minLength(10), InputValidator.passwordStrength()]),

        confirmpassword: new FormControl('', [

          Validators.required //, InputValidator.matchValuesTwo('password')

        ]),

      },

      {validators: [InputValidator.matchValues('password','confirmpassword')]} as AbstractControlOptions,

    );

    //second approach for MatchValuesTwo

    //above will only do validation when the confirm passwors is changed. We need to do the same when password is changed as well

    /\*

    this.registerForm.controls['password'].valueChanges.subscribe({

      next: () => this.registerForm.controls['confirmpassword'].updateValueAndValidity()

    });

    \*/

  }

  onCancel(){

    this.helperService.logIfFrom("Register Cancelled", "OnCancel TemplateForm");

    this.cancelRegister.emit(false);

  }

  onRegister(){

    this.helperService.logIfFrom(this.registerForm?.value, "Register Reactive");

    /\*

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

      next: () => {

        this.helperService.logIfFrom("", "register template driven");

        this.onCancel();

      },

      error: e => {

        this.helperService.logIfError(e, "register error template driven");

        this.toastr.error(e.error);

      },

      complete: () => {

        this.helperService.logIfFrom("register complete", "register template driven");

      }

    });

    \*/

  }

}

#### register-reactiveform.component.html

<div>

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

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

        <hr>

        <div class="form-group mb-3">

            <input type="text" class="form-control" formControlName="username" placeholder="Username" [class.is-invalid]="rf['username'].errors && rf['username'].touched">

            <!--can use rf or rf2-->

            <div \*ngIf="rf['username'].touched && rf['username'].pristine && rf['username'].dirty && rf['username'].invalid && rf['username'].hasError('required')" class="alert alert-danger py-1 small">User name is required</div>

            <div \*ngIf="rf2('username').touched && rf2('username').invalid && rf2('username').hasError('minlength')" class="alert alert-danger py-1 small">User name must be minimum 5 characters</div>

            <div \*ngIf="rf2('username').touched && rf2('username').invalid &&

                        !(rf2('username').hasError('required') || rf2('username').hasError('minlength')) &&

                        rf2('username').hasError('userNameAllowed')" class="alert alert-danger py-1 small">User name must be alpha numeric with \_ and - are allowed. Number cannot be at start and  \_ - should be in the middle</div>

            <div \*ngIf="rf2('username').touched && rf2('username').invalid &&

                        !(rf2('username').hasError('required') || rf2('username').hasError('minlength') || rf2('username').hasError('userNameAllowed')) &&

                        rf2('username').hasError('userNameExistAsync')" class="alert alert-danger py-1 small">User name already taken</div>

        </div>

        <div class="form-group mb-3">

            <input type="password" class="form-control" formControlName="password" placeholder="Password" [class.is-invalid]="rf['password'].errors && rf['password'].touched">

            <!--can use rf or rf2-->

            <div \*ngIf="rf['password'].touched && rf['password'].invalid && rf['password'].hasError('required')" class="alert alert-danger py-1 small">Password is required</div>

            <div \*ngIf="rf2('password').touched && rf2('password').invalid && rf2('password').hasError('minlength')" class="alert alert-danger py-1 small">Password must be minimum 10 characters</div>

            <div \*ngIf="rf2('password').touched && rf2('password').invalid && !(rf['password'].hasError('required') || rf2('password').hasError('minlength')) &&

                        rf['password'].hasError('passwordStrength')" class="alert alert-danger py-1 small">Password must have an upper case, lower case, special character, a number and max length 30</div>

        </div>

        <div class="form-group mb-3">

            <input type="password" class="form-control" formControlName="confirmpassword" placeholder="Confirm password" [class.is-invalid]="rf['confirmpassword'].errors && rf['confirmpassword'].touched">

            <!--can use rf or rf2-->

            <div \*ngIf="rf['confirmpassword'].touched && rf['confirmpassword'].invalid && rf['confirmpassword'].hasError('required')" class="alert alert-danger py-1 small">Confirm password is required</div>

            <div \*ngIf="rf['confirmpassword'].touched && rf['confirmpassword'].invalid && !rf['confirmpassword'].hasError('required') && rf['confirmpassword'].hasError('notMatching')" class="alert alert-danger py-1 small">Passwords do not match</div>

            <div \*ngIf="rf['confirmpassword'].touched && rf['confirmpassword'].invalid && !rf['confirmpassword'].hasError('required') && rf['confirmpassword'].hasError('notMatchingTwo')" class="alert alert-danger py-1 small">Passwords do not match - notMatching2</div>

        </div>

        <div class="text-center">

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

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

        </div>

    </form>

    <div class="card mt-2">

        <div class="card-header">Register Form</div>

        <div class="card-body">

            <p>Status: <code>{{ registerForm.status | json }}</code></p>

            <p>Value: <br>

                <code>{{ registerForm.value | json }}</code>

            </p>

        </div>

    </div>

    <div class="card mt-2">

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

        <div class="card-body">

            <ul>

                <li>Username must be Alpha numeric with \_ and - are allowed. Number cannot be at start and  \_ - should be in the middle</li>

                <li>User name must be minimum 5 characters</li>

                <li>Password must be minimum 10 characters</li>

                <li>Password must have an upper case, lower case, special character, a number and max length 30</li>

                <li>Password and confirm password must match</li>

            </ul>

        </div>

    </div>

</div>

# Reactive Form with Reusable Controls

## Setup

> ng g c site/registerReactiveformReusablecontrols --module=core/modules/shared --skip-tests

This will add the componenet to the shared.module as well

## Step 1 – [home.componenet.html](http://home.componenet.html) update

Open /site/home/home.component.html and call the new component for registering.

    <div \*ngIf="registerMode" class="conatiner">

        <div class="row justify-content-center">

            <div class="col-4">

                <!--<app-register (cancelRegister)="onCancelRegisterMode($event)"></app-register>-->

                <!--<app-register-reactiveform (cancelRegister)="onCancelRegisterMode($event)"></app-register-reactiveform>-->

                <app-register-reactiveform-reusablecontrols (cancelRegister)="onCancelRegisterMode($event)"></app-register-reactiveform-reusablecontrols>

            </div>

        </div>

    </div>

## Step 2 – Fill the new Control .ts and .html

Open the /site/register-reactiveform-reusablecontrols .ts and .html and copy paste the content from /site/register-reactiveform .html and .ts files.

Change the title to

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

Do ng serve and make sure the app is running without any issues

## Step 3 – Create ReusableControls

### /site/formControls/text-input.component

#### Setup

> ng g c site/formControls/textInput --module=core/modules/shared --skip-tests

This will create the component and will also add to the shared module

#### text-input.component.ts

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

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

@Component({

  selector: 'app-text-input',

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

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

})

export class TextInputComponent implements ControlValueAccessor {

  //properties

  @Input() label: string = '';

  @Input() labelMustMatch: string = '';

  @Input() placeHolder: string = '';

  @Input() type: string = 'text';

  //Self

  //angular will check if it is used recently and will reuse as kept in memory

  //when it comes to inputs we do not want to resue any other control that was already in memory

  //we want to make sure that this NgControl is unique to the inputs that we are updating to in the DOM

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

    this.ngControl.valueAccessor = this; //this represents TextInputComponent class

  }

  //no need to write code in any of the below nethods that got implemented due to ControlValueAccessor

  writeValue(obj: any): void {}

  registerOnChange(fn: any): void {}

  registerOnTouched(fn: any): void {}

  //optional so commented

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

  get getcontrol() : FormControl{

    return this.ngControl.control as FormControl;

  }

}

#### text-input.component.html

<div class="mb-3">

    <!-- 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"

        type="{{ type }}"

        class="form-control"

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

        placeholder="{{ placeHolder }}"

        >

    -->

    <input

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

        type="{{ type }}"

        class="form-control"

        [formControl]="getcontrol"

        placeholder="{{ placeHolder }}"

        >

    <!--required-->

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

        {{label}} is required

    </div>

    <!--alphaNumeric-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['onlyAlphaNumeric'] && !getcontrol.errors?.['required']">

        {{label}} must be only alpha numeric

    </div>

    <!--onlyChar-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['onlyChar'] && !getcontrol.errors?.['required']">

        {{label}} must be only characters

    </div>

    <!--onlyCharWithSpace-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['onlyCharWithSpace'] && !getcontrol.errors?.['required']">

        {{label}} must be only characters and space

    </div>

    <!--minlength-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['minlength'] &&

                                        !(getcontrol.errors?.['maxlength'] || getcontrol.errors?.['required'] || getcontrol.errors?.['onlyChar'] || getcontrol.errors?.['onlyCharWithSpace'] || getcontrol.errors?.['onlyAlphaNumeric'])">

        <!--.requiredLength will pick up the length specified in validators-->

        <!--{{label}} must be minimum {{getcontrol.errors?.['minlength']['requiredLength']}} characters-->

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

    </div>

    <!--maxlength-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['maxlength'] &&

                                        !(getcontrol.errors?.['minlength'] || getcontrol.errors?.['required'] || getcontrol.errors?.['onlyChar'] || getcontrol.errors?.['onlyCharWithSpace'] || getcontrol.errors?.['onlyAlphaNumeric'])">

        <!--.requiredLength will pick up the length specified in validators-->

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

    </div>

    <!--password strength-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['passwordStrength'] &&

                                        !(getcontrol.errors?.['minlength'] || getcontrol.errors?.['maxlength'] || getcontrol.errors?.['required'] || getcontrol.errors?.['onlyChar'] || getcontrol.errors?.['onlyCharWithSpace'])">

        {{label}} must have an upper case, lower case, special character, a number and max length 30

    </div>

    <!--not matching-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['notMatching'] && !getcontrol.errors?.['required']">

        {{labelMustMatch}} do not match

    </div>

    <!--not matching two-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['notMatchingTwo'] && !getcontrol.errors?.['required']">

        {{labelMustMatch}} do not match

    </div>

    <!--userNameAllowed-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['userNameAllowed'] &&

                                        !(getcontrol.errors?.['minlength'] || getcontrol.errors?.['maxlength'] || getcontrol.errors?.['required'] || getcontrol.errors?.['onlyChar'] || getcontrol.errors?.['onlyCharWithSpace'])">

        {{label}} must be alpha numeric with \_ and - are allowed. Number cannot be at start and  \_ - should be in the middle

    </div>

    <!--userNameExistAsync-->

    <div class="invalid-feedback" \*ngIf="getcontrol.errors?.['userNameExistAsync'] &&

                                        !(getcontrol.errors?.['userNameAllowed'] || getcontrol.errors?.['minlength'] || getcontrol.errors?.['maxlength'] || getcontrol.errors?.['required'] || getcontrol.errors?.['onlyChar'] || getcontrol.errors?.['onlyCharWithSpace'])">

        {{label}} already taken

    </div>

</div>

### /site/formControls/date-input.component

#### Setup

> ng g c site/formControls/dateInput --module=core/modules/shared --skip-tests

This will create the component and will also add to the shared module

We will be using:

<https://valor-software.com/ngx-bootstrap/#/components/datepicker?tab=overview>

If not added then add it like

>ng add ngx-bootstrap --component datepicker

Add it to the shared module imports and exports array

Imports: BsDatepickerModule.forRoot(),

Exports: BsDatepickerModule

#### date-input.component.ts

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

import { ControlValueAccessor, FormControl, 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;

  //config for datepicker, partial infront makes all the properties optional

  bsConfig: Partial<BsDatepickerConfig> | undefined;

  //Self

  //angular will check if it is used recently and will reuse as kept in memory

  //when it comes to inputs we do not want to resue any other control that was already in memory

  //we want to make sure that this NgControl is unique to the inputs that we are updating to in the DOM

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

    this.ngControl.valueAccessor = this; //this represents DateInputComponent class

    this.bsConfig = {

      containerClass: 'theme-red', //default is theme-green

      dateInputFormat: 'YYYY-MM-DD', //DD MMMM YYYY

    }

  }

  writeValue(obj: any): void {}

  registerOnChange(fn: any): void {}

  registerOnTouched(fn: any): void {}

  //optional so commented

  setDisabledState?(isDisabled: boolean): void {}

  get getcontrol(): FormControl{

    return this.ngControl.control as FormControl;

  }

}

#### date-input.component.html

<div class="mb-3">

    <input

        type="text"

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

        class="form-control"

        [formControl]="getcontrol"

        placeholder="{{ placeHolder }}"

        bsDatepicker

        [bsConfig]="bsConfig"

        [maxDate]="maxDate"

        >

    <!--required-->

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

        {{label}} is required

    </div>

</div>

### /site/register-reactiveform-reusablecontrols

#### register-reactiveform-reusablecontrols.css

/\*

.reset {

    all: revert;

}

\*/

#### register-reactiveform-reusablecontrols.ts

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

import { ToastrService } from 'ngx-toastr';

import { Subscription } from 'rxjs';

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

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

import { HelperService } from '../../core/services/helper.service';

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

import { UserRegisterDto } from '../../core/models-interfaces/user-register-dto.model';

import { InputValidator } from '../../core/validators/input-validators';

@Component({

  selector: 'app-register-reactiveform-reusablecontrols',

  templateUrl: './register-reactiveform-reusablecontrols.component.html',

  styleUrls: ['./register-reactiveform-reusablecontrols.component.css']

})

export class RegisterReactiveformReusablecontrolsComponent implements OnInit, OnDestroy {

   //tell the parent homeComponent that cancel is clicked

   @Output('cancelRegister') cancelRegister = new EventEmitter();

   //register: UserRegisterDto = <UserRegisterDto>{};

   //reactive form

   registerForm!: FormGroup;

   registerSubscription!: Subscription;

   //must be minimum 18 years old

    maxDate!: Date;

    validationErrors: string[] = [];

    //gender list

    genderList = [

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

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

    ];

   constructor(private helperService: HelperService, private accountService: AccountService,

     private toastr: ToastrService,

     private fb: FormBuilder,

     private router: Router){}

   ngOnDestroy(): void {

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

   }

   ngOnInit(): void {

     //this.initializeFormWithLocalValidator();

     //this.initializeForm(); //this uses /code/validators

     this.initializeForm\_FormBuilder();

     this.maxDate = new Date();

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

   }

   //convenience getter for easy access to form fields

   get rf() {

     return this.registerForm.controls;

   }

   rf2(key: string) {

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

   }

   initializeFormWithLocalValidator(){

     this.registerForm = new FormGroup({

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

       password: new FormControl('', [Validators.required, Validators.minLength(10)]),

       confirmPassword: new FormControl('', [Validators.required, this.matchValues('password')]),

     });

     //above will only do validation when the confirm passwors is changed. We need to do the same when password is changed as well

     this.registerForm.controls['password'].valueChanges.subscribe({

       next: () => this.registerForm.controls['confirmpassword'].updateValueAndValidity()

     });

   }

   //using local validator

   matchValues(matchTo: string): ValidatorFn{

     return (control: AbstractControl) => {

       return control.value === control.parent?.get(matchTo)?.value ? null : {notMatching: true}

     }

   }

   //two different approaches used here for the password and conform password match check

   //either use matchValue or matchValueTwo do not use both

   //password strength returns multiple items check back

   initializeForm(){

     this.registerForm = new FormGroup({

         userName: new FormControl('',

                                     [Validators.required, Validators.minLength(5), InputValidator.userNameAllowed()],

                                     [InputValidator.userNameExistAsync(this.accountService)]),

         password: new FormControl('', [Validators.required, Validators.minLength(10), InputValidator.passwordStrength()]),

         confirmPassword: new FormControl('', [

           Validators.required //, InputValidator.matchValuesTwo('password')

         ]),

       },

       {validators: [InputValidator.matchValues('password','confirmpassword')]} as AbstractControlOptions,

     );

     //second approach for MatchValuesTwo

     //above will only do validation when the confirm passwors is changed. We need to do the same when password is changed as well

     /\*

     this.registerForm.controls['password'].valueChanges.subscribe({

       next: () => this.registerForm.controls['confirmpassword'].updateValueAndValidity()

     });

     \*/

   }

   initializeForm\_FormBuilder(){

      this.registerForm = this.fb.group({

        userName: ['',

                  [Validators.required, Validators.minLength(5), InputValidator.userNameAllowed()],

                  [InputValidator.userNameExistAsync(this.accountService)]],

        password: ['', [Validators.required, Validators.minLength(10), InputValidator.passwordStrength()]],

        confirmPassword: ['', [Validators.required ]],

        gender: ['male'],

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

        dateOfBirth: ['', [Validators.required]],

        city: ['', [Validators.required, InputValidator.onlyCharWithSpace()]],

        country: ['', [Validators.required, InputValidator.onlyCharWithSpace()]],

      },

      {validators: [InputValidator.matchValues('password','confirmpassword')]} as AbstractControlOptions,

      );

   }

   onCancel(){

     this.helperService.logIfFrom("Register Cancelled", "OnCancel TemplateForm");

     this.cancelRegister.emit(false);

   }

   onRegister(){

     this.helperService.logIfFrom(this.registerForm?.value, "Register Reactive");

     this.validationErrors = [];//reset

     //when invalid do not proceed further

     if (!this.isReactiveFormGood()) return;

    //do registration

     this.doRegisteration();

   }

   private isReactiveFormGood(): boolean {

    if (this.registerForm.invalid) {

      this.showErrorsOnSubmit();

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

      return false;

    }

    return true;

  }

  private showErrorsOnSubmit() {

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

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

      if (control?.errors)

        control.markAsTouched({onlySelf: true});

    });

  }

  private getDateOnly(dob: string | undefined){

    if(!dob) return;

    let theDob = new Date(dob);

    let isoDateString = new Date(theDob.setMinutes(theDob.getMinutes()-theDob.getTimezoneOffset())).toISOString();

    let theDate = isoDateString.slice(0, 10); //get the characters from 0 to 10

    this.helperService.logIfFrom(isoDateString, "getDateOnly isoDateString");

    this.helperService.logIfFrom(theDate, "getDateOnly theDate");

    return theDate;

  }

  private getControlValue(key: string): any{

    //this.registerForm.value['userName']

    //or

    //this.registerForm.controls['userName'].value

    return this.registerForm.controls[key].value;

  }

  private mapFormToUserRegisterDto(): UserRegisterDto{

    const registerUser = new UserRegisterDto(this.getControlValue('userName'),

                                              this.getControlValue('password'),

                                              this.getControlValue('confirmPassword'),

                                              this.getControlValue('gender'),

                                              this.getControlValue('displayName'),

                                              this.getControlValue('dateOfBirth'),

                                              this.getControlValue('city'),

                                              this.getControlValue('country')

                                            );

    this.helperService.logIfFrom(registerUser, "mapFormToUserRegisterDto registerUser");

    return registerUser;

  }

  private mapRawValuesToUserRegisterDto(rawvalues: any): UserRegisterDto{

    const registerUser = new UserRegisterDto(rawvalues.userName,

                                              rawvalues.password,

                                              rawvalues.confirmPassword,

                                              rawvalues.gender,

                                              rawvalues.displayName,

                                              rawvalues.dateOfBirth,

                                              rawvalues.city,

                                              rawvalues.country

                                            );

    this.helperService.logIfFrom(registerUser, "mapRawValuesToUserRegisterDto registerUser");

    return registerUser;

  }

  private doRegisteration() {

    const dob = this.getDateOnly(this.getControlValue('dateOfBirth'));

    //fill

    /\*

    1. can use this.registerForm.value

    2. const registerUser = this.mapFormToUserRegisterDto();

    3. use spread operator, which we are doing below as we ned to overwite the dob

    \*/

    var rawvalues = {...this.registerForm.value, dateOfBirth: dob};

    this.helperService.logIfFrom(rawvalues, "doRegisteration rawvalues");

    //can call like

    //this.registerSubscription = this.accountService.register(rawvalues)

    //or like

    const registerUser2 = this.mapRawValuesToUserRegisterDto(rawvalues);

    this.registerSubscription = this.accountService.register(registerUser2).subscribe({

      next: () => {

        //go to members page

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

      },

      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){

          this.toastr.error("An error occured");

          this.validationErrors = e;

        }

      }

    });

  }

}

#### register-reactiveform-reusablecontrols.html

<!--

    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

-->

<div>

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

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

        <hr>

        <!--reset style in local .css is commented-->

        <fieldset class="reset mb-3">

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

            <div class="mb-3">

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

                <label class="form-check-label" \*ngFor="let gl of genderList; let i = index">

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

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

                </label>

            </div>

            <app-text-input [formControl]="rf2('displayName')" [label]="'Display name'" [placeHolder]="'display name'"></app-text-input>

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

            <app-text-input [formControl]="rf2('city')" [label]="'City'" [placeHolder]="'city'"></app-text-input>

            <app-text-input [formControl]="rf2('country')" [label]="'Country'" [placeHolder]="'country'"></app-text-input>

        </fieldset>

        <fieldset class="reset mb-3">

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

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

            <app-text-input [formControl]="rf2('password')" [type]="'password'" [label]="'Password'" [placeHolder]="'password'"></app-text-input>

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

        </fieldset>

        <!--display errors returned from the server-->

        <fieldset class="reset mb-3" \*ngIf="validationErrors && validationErrors.length > 0">

            <legend class="reset">Server Validation Errors</legend>

            <ul class="text-danger">

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

            </ul>

        </fieldset>

        <div class="text-center">

            <!--registerForm.pending check is due to use of AsycValidator for unique userName-->

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

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

        </div>

    </form>

    <div class="card mt-2">

        <div class="card-header">Register Form</div>

        <div class="card-body">

            <p>Status: <code>{{ registerForm.status | json }}</code></p>

            <p>Value: <br>

                <code>{{ registerForm.value | json }}</code>

            </p>

        </div>

    </div>

    <div class="card mt-2">

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

        <div class="card-body">

            <ul>

                <li>User name must be Alpha numeric with \_ and - are allowed. Number cannot be at start and  \_ - should be in the middle</li>

                <li>User name must be minimum 5 characters</li>

                <li>Password must be minimum 10 characters</li>

                <li>Password must have an upper case, lower case, special character, a number and max length 30</li>

                <li>Password and confirm password must match</li>

            </ul>

        </div>

    </div>

</div>