# Project

* Site-04-Ng-login-reg-nav-services-templateforms-Api-MiddlewareException
  + Copied from: Site-03-Api-Auth-JWT-Reg-Login-Extensions
* For the “MySocialConnect-SPA” issue npm install command to install all packages
  + ng serve : to run the spa

# New & Updates

|  |  |
| --- | --- |
| New | Updates |
| /site/nav | App.module.ts |
| /core/models-interfaces/login-dto.model.ts | App.component.ts |
| /coe/models-interfaceslogged-in-user-dto.model.ts | App.component.html |
| /core/servides/account.service.ts | Core/services/helper.service.ts |
| /core/services/local-storage.service.ts | Styles.ss |
| /core/services/template-form.validators.service.ts |  |
| /core/directives/must-match.directive.ts |  |

# 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. also check TemplateFormValidatorsService

Check [010 Project11 -Ng-reactiveforms-validators-reusable-controls.docx](010%20Project11%20-Ng-reactiveforms-validators-reusable-controls.docx) as well

# styles.css

/\* You can add global styles to this file, and also import other style files \*/

.title {color: #FF4500;}

.api-url { color: #FF8C00; }

.main-container {

    margin-top: 60px;

    margin-bottom: 65px; /\*with footer\*/

}

a.welcome-user, a.welcome-user:link, a.welcome-user:visited, a.welcome-user:hover, a.welcome-user:active { text-decoration: none; color: #FF7F50 !important; }

a.welcome-user:hover { color: #FF4500 !important; cursor: pointer; }

.dropdown-toggle, .dropdown-item {cursor: pointer;}

footer {

    position: fixed;

    bottom: 0;

    width: 100%;

    height: 60px;

    border-top-width: 5px;

    border-top-style: solid;

    border-top-color: #FF7F50;

    display: block;

    padding: 0 10px;

    margin-top: 10px;

    text-align: center;

    background-color: #fff;

 }

# app.module.ts

Add the FormsModule and ReactiveFormsModule

# AppComponent

## app.component.ts

comment out the two calls that are fetching the users data

## app.component.html

comment out the current html

# models-interfaces

## Global

### Location

Core/models-interfaces

### login-dto.model.ts

> ng g interface core/models-interfaces/loginDto --type=model

export interface LoginDto {

    userName: string;

    password: string;

}

### logged-in-user-dto.model.ts

> ng g interface core/models-interfaces/loggedInUserDto --type=model

export interface LoggedInUserDto {

    userName: string;

    guid: string;

    token: string;

    mainPhotoUrl: string;

    displayName: string;

    gender: string;

}

### User-register-dto.models.ts

> ng g class core/models-interfaces/userRegisterDto --type=model --skip-tests

CREATE src/app/core/models-interfaces/user-register-dto.model.ts (33 bytes)

This one created as class and not an interface

export class UserRegisterDto {

    constructor(public userName: string = "",

        public password: string = "",

        public confirmPassword: string = ""

     ) {}

}

# Services

## Global

### Location

Core/services

### account.service.ts

> ng g s core/services/account --skip-tests

Check persisting the user and update to this service under “[Persisting Logged in User](#_Persisting_the_Logged)”

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

import { map } from 'rxjs';

import { HttpClientService } from './http-client.service';

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

import { LocalStorageService } from './local-storage.service';

import { LoginDto } from '../models-interfaces/login-dto.model';

import { LoggedInUserDto } from '../models-interfaces/logged-in-user-dto.model';

@Injectable({

  providedIn: 'root'

})

export class AccountService {

  private serviceName: string = "AccountService";

  //for all http requests use the HttpClientService rather than making the call locally

  //helper service has the urls etc, this will build the base url properly

  constructor(private httpClientService: HttpClientService,

              private helperService: HelperService,

              private localStorageService: LocalStorageService) { }

  login(loginDto: LoginDto){

    this.localStorageService.removeUser();

    const url = this.helperService.urlAccountLogin;

    this.helperService.logIf(`${this.serviceName}.Login Url: ${url}`);

    //we are getting back LoggedInUserDto

    return this.httpClientService.post<LoggedInUserDto>(url, loginDto)

    .pipe(

      map((r: LoggedInUserDto) => {

        const user:LoggedInUserDto  = r;

        if(user)

          this.setAndFireCurrentUser(user);

        return user;

      })

    );

  }

  fireCurrentUser(user: LoggedInUserDto){

  }

  setAndFireCurrentUser(user: LoggedInUserDto){

    this.localStorageService.setUser(user);

    this.fireCurrentUser(user);

  }

  getAndFireCurrentUser(){

    const user: LoggedInUserDto = this.localStorageService.getUser();

    this.fireCurrentUser(user);

  }

}

### local-storage.service.ts

> ng g s core/services/localStorage --skip-tests

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

import { LoggedInUserDto } from '../models-interfaces/logged-in-user-dto.model';

@Injectable({

  providedIn: 'root'

})

export class LocalStorageService {

  //setup the keys for different items

  public \_keyUser: string = "MySocialConnectUser";

  constructor() { }

  getStorage(): any[]{

    let s: any[] = [];

    for(let i=0; i<localStorage.length; i++){

      const key: string = localStorage.key(i)!;

      if(key){

        const value: string = localStorage.getItem(key)!;

        s.push({key: key, value: value});

      }

    }

    return s;

  }

  getItemJson(key: string): any|null{

    const item = this.getItem(key);

    if(!item) return null;

    return JSON.parse(item);

  }

  getUser(): LoggedInUserDto{

    return this.getItemJson(this.\_keyUser)

  }

  getItem(key: string){

    return localStorage.getItem(key);

  }

  setItemJson(key: string, value: any){

    const item = JSON.stringify(value);

    this.setItem(key, item);

  }

  setUser(user: LoggedInUserDto){

    this.setItemJson(this.\_keyUser, user);

  }

  setItem(key: string, value: string){

    localStorage.setItem(key, value)

  }

  removeItem(key: string){

    const item = this.getItem(key);

    if(!item) return;

    localStorage.removeItem(key);

  }

  removeUser(){

    this.removeItem(this.\_keyUser)

  }

  //helper items to get some of the common pieces

  public getUserName: string = this.getUser()?.userName;

  public getUserToken: string = this.getUser()?.token;

  public getUserGuid: string = this.getUser()?.guid;

  public getUserGender: string = this.getUser()?.gender;

  public getUserPhotoUrl: string = this.getUser()?.mainPhotoUrl;

  public getUserDisplayName: string = this.getUser()?.displayName;

}

### helper.service.ts

Added the link to the checkUser url

  public urlAccountCheckUser: string = `${this.urlAccount}/checkUser/${this.keyName}`;

# Persisting the Logged In User

## Core > services > account-service.ts

This also includes the new method to check user name taken or available

And also the register method

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

import { BehaviorSubject, map } from 'rxjs';

import { HttpClientService } from './http-client.service';

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

import { LocalStorageService } from './local-storage.service';

import { LoginDto } from '../models-interfaces/login-dto.model';

import { LoggedInUserDto } from '../models-interfaces/logged-in-user-dto.model';

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

@Injectable({

  providedIn: 'root'

})

export class AccountService {

  private serviceName: string = "AccountService";

  //behavioral subject observable for loggedInUser

  private loggedInUserSource = new BehaviorSubject<LoggedInUserDto | null>(null);

  currentLoggedInUser$ = this.loggedInUserSource.asObservable();

  //for all http requests use the HttpClientService rather than making the call locally

  //helper service has the urls etc, this will build the base url properly

  constructor(private httpClientService: HttpClientService,

              private helperService: HelperService,

              private localStorageService: LocalStorageService) { }

  checkUser(userName: string){

    const url = this.helperService.replaceKeyValue(this.helperService.urlAccountCheckUser, this.helperService.keyName, userName);

    return this.httpClientService.get<boolean>(url);

  }

  //fire the user as null so that any one subscribing to it can update

  logout(){

    //remove the user

    this.localStorageService.removeUser();

    this.fireCurrentUser(null);

  }

  //set the user in local storage and fire the user so that any one subscribing to it can action

  login(loginDto: LoginDto){

    this.localStorageService.removeUser();

    const url = this.helperService.urlAccountLogin;

    this.helperService.logIfFrom(url, `${this.serviceName}.Login Url`);

    //we are getting back LoggedInUserDto

    //persist the loggedinUser info in the local browser storage

    return this.httpClientService.post<LoggedInUserDto>(url, loginDto)

    .pipe(

      map((r: LoggedInUserDto) => {

        const user:LoggedInUserDto  = r;

        if(user)

          this.setAndFireCurrentUser(user);

        //return user;

      })

    );

  }

  register(user: UserRegisterDto){

    this.localStorageService.removeUser();

    const url = this.helperService.urlAccountRegister;

    this.helperService.logIfFrom(url, `${this.serviceName}.Register Url`);

    //we are getting back LoggedInUserDto

    //persist the loggedinUser info in the local browser storage

    return this.httpClientService.post<LoggedInUserDto>(url, user)

    .pipe(

      map((r: LoggedInUserDto) => {

        const user:LoggedInUserDto  = r;

        if(user)

          this.setAndFireCurrentUser(user);

        //return user;

      })

    );

  }

  private fireCurrentUser(user: LoggedInUserDto | null){

    this.loggedInUserSource.next(user);

  }

  private setAndFireCurrentUser(user: LoggedInUserDto){

    this.localStorageService.setUser(user);

    this.fireCurrentUser(user);

  }

  getAndFireCurrentUser(){

    const user: LoggedInUserDto = this.localStorageService.getUser();

    if(!user) {

      this.fireCurrentUser(null);

      return;

    }

    this.fireCurrentUser(user);

  }

}

## app.component.ts

Fire the getAndFireCurrentUser method from inside

  constructor(private helperService: HelperService, private http: HttpClient, private accountService: AccountService){}

  ngOnInit(): void {

    this.title = this.helperService.Title;

    this.webApiUrl = this.helperService.BaseUrlServer;

    //this.getUsersPreferred();

    //this.getUsersNonPreferred();

    this.getAndSetLoggedInUser();

  }

  getAndSetLoggedInUser(){

    this.accountService.getAndFireCurrentUser();

  }

# Nav Setup

Go to bootstrap <https://getbootstrap.com/docs/5.3/examples/carousel/>

Right click nav area and copy the “nav” element

<nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

## Add Nav Component

|  |  |
| --- | --- |
| DryRun **> ng g c site/nav --skip-tests --dry-run**  CREATE src/app/site/nav/nav.component.html (18 bytes)  CREATE src/app/site/nav/nav.component.ts (190 bytes)  CREATE src/app/site/nav/nav.component.css (0 bytes)  UPDATE src/app/app.module.ts (656 bytes) | Actual **> ng g c site/nav --skip-tests**  CREATE src/app/site/nav/nav.component.html (18 bytes)  CREATE src/app/site/nav/nav.component.ts (190 bytes)  CREATE src/app/site/nav/nav.component.css (0 bytes)  UPDATE src/app/app.module.ts (656 bytes) |

This will add the NavComponent to app.module.ts as well.

## nav.component.ts

Final update is [here…](#_nav.component.ts)

import { AfterViewInit, Component, ElementRef, OnInit, ViewChild } from '@angular/core';

import { LoginDto } from '../../core/models-interfaces/login-dto.model';

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

@Component({

  selector: 'app-nav',

  templateUrl: './nav.component.html',

  styleUrls: ['./nav.component.css']

})

export class NavComponent implements OnInit, AfterViewInit {

  title = "";

  constructor(private helperService: HelperService) { }

  ngAfterViewInit(): void {

    //put a focus in userName element

    this.userNameElement.nativeElement.focus();

  }

  ngOnInit(): void {

    this.title = this.helperService.Title;

  }

}

## nav.component.html

Paste copied html and replace the content. We are not doing responsive design in this tutorial so not showing that

Updated code for implementation purposes

Final update is [here…](#_nav.component.html)

<nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

    <div class="container"> <!--container-fluid to container-->

      <a class="navbar-brand" href="#"><span class="title">{{ title }}</span></a>

      <!--removing the navbar toggler, this is for responsive design-->

      <!--

      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

      </button>

      -->

      <!--<div class="collapse navbar-collapse" id="navbarCollapse">-->

        <ul class="navbar-nav me-auto mb-2 mb-md-0">

          <li class="nav-item"><a class="nav-link" href="#">Matches</a></li>

          <li class="nav-item"><a class="nav-link" href="#">Lists</a></li>

          <li class="nav-item"><a class="nav-link" href="#">Messages</a></li>

        </ul>

        <form \*ngIf="!isLoggedIn" #loginForm="ngForm" class="d-flex" (ngSubmit)="onLogin()" autocomplete="off">

          <input name="username" id="username" #username [(ngModel)]="loginDto.userName"

            class="form-control me-2" type="text" placeholder="Username" aria-label="Username">

          <input name="password" id="password" #password [(ngModel)]="loginDto.password"

            class="form-control me-2" type="password" placeholder="Password" aria-label="Password">

          <button class="btn btn-outline-info" type="submit">Login</button>

        </form>

      <!--</div>-->

    </div>

  </nav>

## Add Nav to AppComponent

### app.component.ts

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

import { HttpClient } from '@angular/common/http';

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

@Component({

  selector: 'app-root',

  templateUrl: './app.component.html',

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

  title = '';

  webApiUrl = '';

  constructor(private helperService: HelperService, private http: HttpClient){}

  ngOnInit(): void {

    this.title = this.helperService.Title;

    this.webApiUrl = this.helperService.BaseUrlServer;

  }

}

### app.component.html

<app-nav></app-nav>

<div class="container main-container">

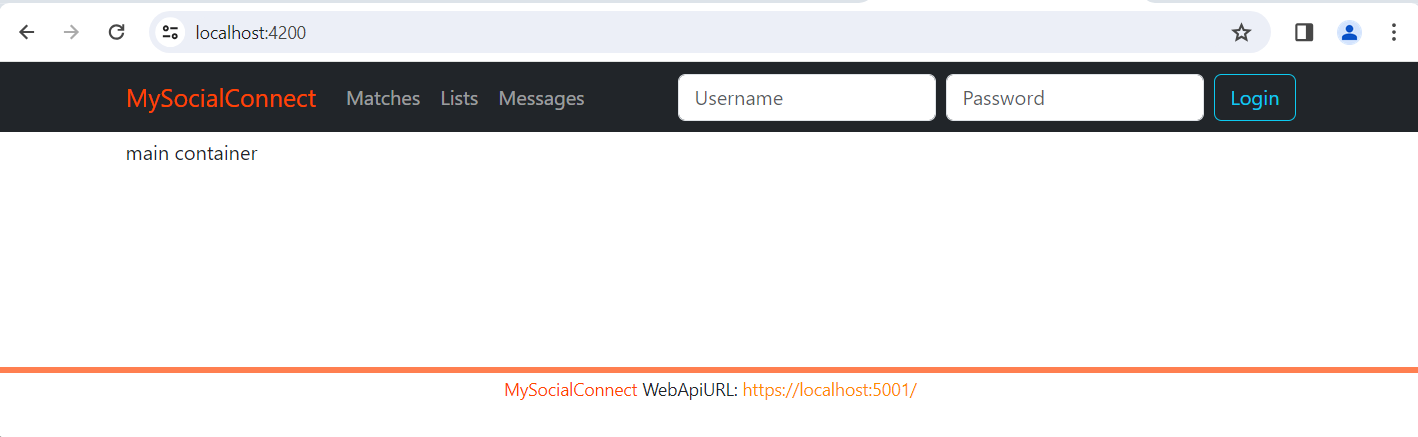
    main container

</div>

<footer><small><span class="title">{{ title }}</span> WebApiURL: <span class="api-url">{{webApiUrl}}</span></small></footer>

## Run the App

Run the app and you should see the landing page



# Login

## Dropdown section when loggedin

### Add ngx-bootstrap Dropdown component

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

>ng add ngx-bootstrap --component dropdowns

The command will update angular.json and app module

#### App.module.ts

import { BsDropdownModule } from 'ngx-bootstrap/dropdown';

  imports: [

    BrowserModule,

    AppRoutingModule,

    BrowserAnimationsModule,

    HttpClientModule,

    FormsModule,

    ReactiveFormsModule,

    BsDropdownModule.forRoot()

  ],

### Styles.css

Check following in styles.css

.dropdown-toggle, .dropdown-item {cursor: pointer;}

## nav.component.html

Pick up the “Trigger by tag <a>” from overview tab.

<nav class="navbar navbar-expand-md navbar-dark fixed-top bg-dark">

    <div class="container"> <!--container-fluid to container-->

      <a class="navbar-brand" href="#"><span class="title">{{ title }}</span></a>

      <!--removing the navbar toggler, this is for responsive design-->

      <!--

      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarCollapse" aria-controls="navbarCollapse" aria-expanded="false" aria-label="Toggle navigation">

        <span class="navbar-toggler-icon"></span>

      </button>

      -->

      <!--<div class="collapse navbar-collapse" id="navbarCollapse">-->

        <!--logged in links section-->

        <ul class="navbar-nav me-auto mb-2 mb-md-0" \*ngIf="isLoggedIn">

          <li class="nav-item"><a class="nav-link" href="#">Matches</a></li>

          <li class="nav-item"><a class="nav-link" href="#">Lists</a></li>

          <li class="nav-item"><a class="nav-link" href="#">Messages</a></li>

        </ul>

        <!--logged in dropped down section-->

        <div class="dropdown" \*ngIf="isLoggedIn" dropdown>

          <a class="dropdown-toggle text-light text-decoration-none" dropdownToggle>

            Welcome {{loggedInUser?.userName | titlecase}}

          </a>

          <div class="dropdown-menu" \*dropdownMenu>

            <a class="dropdown-item">Edit Profile</a>

            <a class="dropdown-item" (click)="onLogout()">Logout</a>

          </div>

        </div>

        <!--not logged in section-->

        <form \*ngIf="!isLoggedIn" #loginForm="ngForm" class="d-flex" (ngSubmit)="onLogin()" autocomplete="off">

          <input name="username" id="username" #username [(ngModel)]="loginDto.userName"

            class="form-control me-2" type="text" placeholder="Username" aria-label="Username">

          <input name="password" id="password" #password [(ngModel)]="loginDto.password"

            class="form-control me-2" type="password" placeholder="Password" aria-label="Password">

          <button class="btn btn-outline-info" type="submit">Login</button>

        </form>

      <!--</div>-->

    </div>

  </nav>

## nav.component.ts

import { AfterViewInit, Component, ElementRef, OnDestroy, OnInit, ViewChild } from '@angular/core';

import { Subscription } from 'rxjs';

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

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

import { LoginDto } from '../../core/models-interfaces/login-dto.model';

import { LoggedInUserDto } from '../../core/models-interfaces/logged-in-user-dto.model';

@Component({

  selector: 'app-nav',

  templateUrl: './nav.component.html',

  styleUrls: ['./nav.component.css']

})

export class NavComponent implements OnInit, AfterViewInit, OnDestroy {

  //put focus on page load in username

  @ViewChild('username', { static: false }) userNameElement!: ElementRef;

  private isExecutingLogin: boolean = false;

  putFocus = false;

  title = "";

  isLoggedIn: boolean = false;

  loggedInUser: LoggedInUserDto | null = <LoggedInUserDto>{};

  loginDto: LoginDto = <LoginDto>{};

  //subscriptions

  loginSubscription!: Subscription;

  loggedInUserSubscription!: Subscription;

  constructor(private helperService: HelperService, private accountService: AccountService) { }

  ngAfterViewInit(): void {

    this.putFocus = true;

    this.focusUserName();

  }

  ngOnInit(): void {

    this.title = this.helperService.Title;

    this.getCurrentLoggedInUser(); //persisted user in local storage

  }

  ngOnDestroy(): void {

    if(this.loginSubscription) this.loginSubscription.unsubscribe();

    if(this.loggedInUserSubscription) this.loggedInUserSubscription.unsubscribe();

  }

  private focusUserName(){

    //put a focus in userName element

    if(!this.isLoggedIn && this.putFocus && this.userNameElement.nativeElement){

      this.userNameElement.nativeElement.focus();

    }

  }

  //login method will fire a behavioral subject from inside the account service so tap into it

  //alternatively in login and logout can handle as well.

  getCurrentLoggedInUser(){

    this.loggedInUserSubscription = this.accountService.currentLoggedInUser$.subscribe({

      next: r => {

        this.isLoggedIn = !!r;

        this.loggedInUser = r;

      },

      error: e => {

        this.helperService.logIfError(e, "getCurrentLoggedInUser")

        this.isLoggedIn = false;

        this.loggedInUser = <LoggedInUserDto>{};

      }

    })

  }

  onLogin(){

    this.helperService.logIfFrom(this.loginDto, 'Login')

    //set executing login true

    this.isExecutingLogin = true;

    this.loginSubscription = this.accountService.login(this.loginDto).subscribe({

      next: r => {

        this.helperService.logIfFrom(r, 'Login.User Response');

        //clear the login form

        this.loginDto = <LoginDto>{};

        //this.isLoggedIn = true;

        //this.loggedInUser = r;

      }, error: e => {

        this.helperService.logIfError(e, 'NavComponent.OnLogin');

        this.focusUserName();

      }, complete: () => {

        //set executing login false

        this.isExecutingLogin = false;

      }

    })

  }

  onLogout(){

    //logout the user

    //this.isLoggedIn = false;

    this.accountService.logout();

    setTimeout(() => {

      this.focusUserName();

    }, 500);

  }

}

# Home Page

> ng g c site/home --skip-tests

CREATE src/app/site/home/home.component.html (19 bytes)

CREATE src/app/site/home/home.component.ts (194 bytes)

CREATE src/app/site/home/home.component.css (0 bytes)

UPDATE src/app/app.module.ts (941 bytes)

This adds the home to the app.module.ts as well

## home.component.ts

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

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

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

@Component({

  selector: 'app-home',

  templateUrl: './home.component.html',

  styleUrls: ['./home.component.css']

})

export class HomeComponent implements OnInit {

  title = "";

  registerMode = false;

  constructor(private helperService: HelperService, public accountService: AccountService){}

  ngOnInit(): void {

    this.title = this.helperService.Title;

  }

  onRegisterToggle(){

    this.registerMode = !this.registerMode;

  }

}

## home.component.html

Using async pipe and accountService in html

<div class="container mt-5">

    <h1><span class="title">{{title}}</span> Home</h1>

</div>

<div \*ngIf="!(accountService.currentLoggedInUser$ | async)" class="container mt-5">

    <div \*ngIf="!registerMode" style="text-align: center">

        <h3>Find your <span class="title">{{title}}</span> matches!</h3>

        <p class="lead">Come on in to see who you know or want to connect with... all you need to do is sign up!</p>

        <div class="text-center">

            <button class="btn btn-primary btn-lg me-2" (click)="onRegisterToggle()">Register</button>&nbsp;

            <button class="btn btn-info btn-lg">Learn More</button>

        </div>

    </div>

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

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

            <div class="col-4">

                Register Form

            </div>

        </div>

    </div>

</div>

## Adding home component to app.component.ts

<app-nav></app-nav>

<div class="container main-container">

    <app-home></app-home>

</div>

<footer><small><span class="title">{{ title }}</span> WebApiURL: <span class="api-url">{{webApiUrl}}</span></small></footer>

# Register Form – Template Driven

## Setup register Component

> ng g c site/register --skip-tests

CREATE src/app/site/register/register.component.html (23 bytes)

CREATE src/app/site/register/register.component.ts (210 bytes)

CREATE src/app/site/register/register.component.css (0 bytes)

## Child/Parent Communication – Output

On the registeration forma we’ll have the cancel button. When that is clicked then the home component will receive that and hide the register form

### register.component.ts

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

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

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

@Component({

  selector: 'app-register',

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

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

})

export class RegisterComponent implements OnInit {

  //tell the parent homeComponent that cancel is clicked

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

  register: UserRegisterDto = <UserRegisterDto>{};

  constructor(private helperService: HelperService){}

  ngOnInit(): void {

  }

  onCancel(){

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

    this.cancelRegister.emit(false);

  }

  onRegister(){

    this.helperService.logIfFrom(this.register, "onRegister TemplateForm");

  }

}

### register.component.html

<div>

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

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

        <hr>

        <div class="form-group mb-3">

            <input type="text" class="form-control"

            name="username" id="username" placeholder="Username"

            [(ngModel)]="register.userName" #username="ngModel >

        </div>

        <div class="form-group mb-3">

            <input type="text" class="form-control"

            name="password" id="password" placeholder="Password"

            [(ngModel)]="register.password" #password="ngModel>

        </div>

        <div class="form-group mb-3">

            <input type="text" class="form-control"

            name="confirmpassword" id="confirmpassword" placeholder="Confirm password"

            [(ngModel)]="register.confirmPassword" #confirmpassword="ngModel>

        </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>

### home.component.ts

Put the code to toggle the registration form when cancel is clicked on the register form

  onCancelRegisterMode(event: boolean){

    this.registerMode = event;

  }

### home.component.html

Add the register form and also receive the cancel button clicked event

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

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

            <div class="col-4">

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

            </div>

        </div>

    </div>

## Validatiors

Validators for the template driven are setup via Service and Directive

<https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/>

### core/services/template-form-validators-service.ts

> ng g s core/services/templateFormValidators --skip-tests

Used by directives for template driven form validation

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

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

import { Observable } from 'rxjs';

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

//https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/

@Injectable({

  providedIn: 'root'

})

export class TemplateFormValidatorsService {

  constructor(private accountService: AccountService) { }

  //assign via directive appMustMatch. Check register form - TemplateDriven one

  MustMatchValidator(source: string, target: string) {

    return(formGroup: FormGroup) => {

      const sourceControl = formGroup.get(source);

      const targetControl = formGroup.get(target);

      if (!targetControl || !sourceControl)

          return null;

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

          return null;

      //check

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

          targetControl.setErrors({ mustMatch: true });

          return ({ mustMatch: true });

      }

      targetControl.setErrors(null);

      return null;

    }

  }

  //assign via directive appPasswordStrength

  PasswordStrengthValidator(): ValidatorFn {

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

      if (!control.value) {

        return null;

      }

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

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

      return valid ? null : { passwordStrength: true };

    };

  }

  //assign via appUserNameAllowed directive

  UserNameAllowedValidator(): ValidatorFn {

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

      if (!control.value) {

        return null;

      }

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

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

      return valid ? null : { userNameAllowed: true };

    };

  }

  //assign via appUserNameCheck directive

  CheckUserNameTakenValidator(control: AbstractControl): Promise<{} | null> | Promise<{ [key: string]: any }> | Observable<{ [key: string]: any }> {

    return new Promise(resolve => {

      if(control.errors?.['required'] || control.errors?.['minlength'] || control.errors?.['userNameAllowed'])

        resolve(null);

      else if(!control.value)

        resolve(null);

      else{

        this.accountService.checkUser(control.value).subscribe({

          next: response => {

            if(response){

              resolve({ userNameNotAvailable: true }); //tap into this

            }

            else{

              resolve(null);

            }

          },

          error: e => {

            resolve(null);

          },

          complete: () => {}

        });

      }

    });

  }

}

### core/directives/must-match.directive.ts

> ng g d core/directives/mustMatch --skip-tests

MustMatch directive will also get add to the app.module.ts, declarations array

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

import { AbstractControl, FormGroup, NG\_VALIDATORS, ValidationErrors, Validator } from '@angular/forms';

import { TemplateFormValidatorsService } from '../services/template-form-validators.service';

//https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/

//also check TemplateFormValidatorsService

@Directive({

  selector: '[appMustMatch]',

  providers: [{ provide: NG\_VALIDATORS, useExisting: MustMatchDirective, multi: true }]

})

export class MustMatchDirective implements Validator {

  @Input('appMustMatch') MustMatch: string[] = []

  constructor(private tfvs: TemplateFormValidatorsService) { }

  validate(formGroup: FormGroup): ValidationErrors | null {

    const result = this.tfvs.MustMatchValidator(this.MustMatch[0], this.MustMatch[1])(formGroup);

    return result;

  }

}

### core/directives/password-strength.directive.ts

> ng g d core/directives/passwordStrength --skip-tests

PassordStrengthDirective directive will also get add to the app.module.ts, declarations array

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

import { AbstractControl, NG\_VALIDATORS, ValidationErrors, Validator } from '@angular/forms';

import { TemplateFormValidatorsService } from '../services/template-form-validators.service';

//https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/

//also check TemplateFormValidatorsService

@Directive({

  selector: '[appPasswordStrength]',

  providers: [{ provide: NG\_VALIDATORS, useExisting: PasswordStrengthDirective, multi: true }]

})

export class PasswordStrengthDirective implements Validator {

  constructor(private tfvs: TemplateFormValidatorsService) { }

  validate(control: AbstractControl): { [key: string]: any } | null {

    const result = this.tfvs.PasswordStrengthValidator()(control);

    return result;

  }

}

### core/directives/user-name-allowed.directive.ts

> ng g d core/directives/userNameAllowed --skip-tests

UserNameAllowedDirective directive will also get add to the app.module.ts, declarations array

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

import { AbstractControl, NG\_VALIDATORS, Validator } from '@angular/forms';

import { TemplateFormValidatorsService } from '../services/template-form-validators.service';

//https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/

//also check TemplateFormValidatorsService

@Directive({

  selector: '[appUserNameAllowed]',

  providers: [{ provide: NG\_VALIDATORS, useExisting: UserNameAllowedDirective, multi: true }]

})

export class UserNameAllowedDirective implements Validator {

  constructor(private tfvs: TemplateFormValidatorsService) { }

  validate(control: AbstractControl): { [key: string]: any } | null {

    const result = this.tfvs.UserNameAllowedValidator()(control);

    return result;

  }

}

### core/directives/user-name-check.directive.ts

> ng g d core/directives/userNameCheck --skip-tests

UserNamecheckDirective directive will also get add to the app.module.ts, declarations array

import { Directive, forwardRef } from '@angular/core';

import { AbstractControl, NG\_ASYNC\_VALIDATORS, Validator } from '@angular/forms';

import { TemplateFormValidatorsService } from '../services/template-form-validators.service';

import { Observable } from 'rxjs';

//https://www.freecodecamp.org/news/how-to-validate-angular-template-driven-forms/

//also check TemplateFormValidatorsService

@Directive({

  selector: '[appUserNameCheck]',

  providers: [{ provide: NG\_ASYNC\_VALIDATORS, useExisting: forwardRef(() => UserNameCheckDirective), multi: true }]

})

export class UserNameCheckDirective implements Validator {

  constructor(private  tfvs: TemplateFormValidatorsService) { }

  validate(control: AbstractControl): Promise<{} | null> | Promise<{ [key: string]: any }> | Observable<{ [key: string]: any }> {

    const result = this.tfvs.CheckUserNameTakenValidator(control);

    return result;

  }

}

## register.component.ts

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

import { Subscription } from 'rxjs';

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

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

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

@Component({

  selector: 'app-register',

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

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

})

export class RegisterComponent implements OnInit, OnDestroy {

  //tell the parent homeComponent that cancel is clicked

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

  register: UserRegisterDto = <UserRegisterDto>{};

  registerSubscription!: Subscription

  constructor(private helperService: HelperService, private accountService: AccountService){}

  ngOnInit(): void {

  }

  ngOnDestroy(): void {

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

  }

  onCancel(){

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

    this.cancelRegister.emit(false);

  }

  onRegister(){

    this.helperService.logIfFrom(this.register, "onRegister TemplateForm");

    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");

      },

      complete: () => {

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

      }

    });

  }

}

## register.component.html

<div>

    <form #registerForm="ngForm" (ngSubmit)="onRegister()" autocomplete="off" [appMustMatch]="['password','confirmpassword']">

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

        <hr>

        <div class="form-group mb-3">

            <input type="text" class="form-control"

            name="username" id="username" placeholder="Username"

            [(ngModel)]="register.userName" #username="ngModel" required minlength="5" appUserNameAllowed appUserNameCheck>

            <div \*ngIf="username.touched && username.invalid && username.errors?.['required']"

                class="alert alert-danger">User name is required</div>

            <div \*ngIf="username.touched && username.invalid && username.errors?.['minlength']"

                class="alert alert-danger">User name must be minimum 5 characters</div>

            <div \*ngIf="username.touched && username.invalid && username.errors?.['userNameAllowed'] && !username.errors?.['minlength']"

                class="alert alert-danger">Alpha numeric with \_ and - are allowed. Number cannot be at start and  \_ - should be in the middle</div>

            <div \*ngIf="username.touched && username.invalid && username.errors?.['userNameNotAvailable'] && !username.errors?.['required'] && !username.errors?.['minlength'] && !username.errors?.['userNameAllowed']"

                class="alert alert-danger">The user name is already taken</div>

        </div>

        <div class="form-group mb-3">

            <input type="password" class="form-control"

            name="password" id="password" placeholder="Password"

            [(ngModel)]="register.password" #password="ngModel" required minlength="10" appPasswordStrength>

            <div \*ngIf="password.touched && password.invalid && password.errors?.['required']"

            class="alert alert-danger">Password is required</div>

            <div \*ngIf="password.touched && password.invalid && password.errors?.['minlength']"

            class="alert alert-danger">Password must be minimum 10 characters</div>

            <div \*ngIf="password.touched && password.invalid && password.errors?.['passwordStrength'] && !password.errors?.['minlength']"

            class="alert alert-danger">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"

            name="confirmpassword" id="confirmpassword" placeholder="Confirm password"

            [(ngModel)]="register.confirmPassword" #confirmpassword="ngModel" required >

            <div \*ngIf="confirmpassword.touched && confirmpassword.invalid && password.errors?.['required']"

            class="alert alert-danger">Confirm password is required</div>

            <div \*ngIf="confirmpassword.touched && confirmpassword.invalid && !password.errors?.['required'] && !confirmpassword.errors?.['required'] && confirmpassword.errors?.['mustMatch'] "

            class="alert alert-danger">Passwords do not match</div>

            <!--

            <div \*ngIf="confirmpassword.touched && !confirmpassword.errors?.['required'] && confirmpassword.value !== password.value"

            class="alert alert-danger">Password do not match</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>

# Clear Database

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

dotnet ef database drop

dotnet ef database update