Table of Contents

[Current Resources with Changes 2](#_Toc112098576)

[New Resources Added 2](#_Toc112098577)

[Environment 2](#_Toc112098578)

[Styles 2](#_Toc112098579)

[app.component 3](#_Toc112098580)

[app.component.ts 3](#_Toc112098581)

[app.component.html 3](#_Toc112098582)

[Nav 4](#_Toc112098583)

[Creating the Component 4](#_Toc112098584)

[Adding the Nav html 4](#_Toc112098585)

[Adding bootstrap js 4](#_Toc112098586)

[Adding app-nav to app component 4](#_Toc112098587)

[Models/DTOs 4](#_Toc112098588)

[LoginDto Model 4](#_Toc112098589)

[UserTokenDto Model 5](#_Toc112098590)

[Services 5](#_Toc112098591)

[ErrorMessageService 5](#_Toc112098592)

[Important 5](#_Toc112098593)

[HttpClient Service 5](#_Toc112098594)

[ApiUrl Service 6](#_Toc112098595)

[LocalStorage Service 6](#_Toc112098596)

[Account Service 6](#_Toc112098597)

[App.module.ts 6](#_Toc112098598)

[Nav Component TS and HTML updates 6](#_Toc112098599)

[nav.component.ts 6](#_Toc112098600)

[nav.component.html 10](#_Toc112098601)

# Current Resources with Changes

1. environments/environment.prod.ts
2. environments/environment.ts
3. app/styles.css
4. app/app.component.html
5. app.module.ts

# New Resources Added

1. app/site/nav component
2. app/core/models/loginDto.model.ts
3. app/core/models/userTokenDto.model.ts
4. app/core/services/error-message.service.ts
5. app/core/services/http-client.service.ts
6. app/core/services/api-url.service.ts
7. app/core/services/local-storage.service.ts
8. app/core/services/account.service.ts

# Environment

Add the following two properties to both the environment files. We’ll display console log conditionally using displayConsoleLog property. Title is the site title, check app.component.ts for details.

  title: "MySocialConnect",

  displayConsoleLog: true,

# Styles

File styles.css

.title {color: #FF4500;}

.api-url { color: #FF8C00; }

.main-container {margin-top: 60px;}

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; }

footer {

    position: absolute;

    bottom: 0;

    height: 60px;

    border-top-width: 5px;

    border-top-style: solid;

    border-top-color: #FF7F50;

    width: 100%;

    display: block;

    padding: 0 10px;

    margin-top: 10px;

    text-align: center;

 }

# app.component

## app.component.ts

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

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

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

@Component({

  selector: 'app-root',

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

  styleUrls: ['./app.component.css']

})

export class AppComponent implements OnInit {

  title = '';

  webApiUrl: string = "";

  constructor(private accountService: AccountService){}

  ngOnInit() {

    this.title = environment.title;

    this.webApiUrl = environment.usebaseUrlHttps ? environment.webApiBaseUrlHttps : environment.webApiBaseUrlHttp;

    //get the user from local storage if available and set it. This is persistence

    this.setCurrentUser();

  }

  //on website load get the user and fire it. The subscribers then will pick it, nav bar component in this case

  setCurrentUser() {

    this.accountService.getAndFireCurrentUser();

  }

}

## app.component.html

for app-nav, row number 1 make sure to have the nav component created with input param. Check nav component [ts](#_nav.component.ts) and [html](#_nav.component.html) below first.

<app-nav [title]="title"></app-nav>

<div class="container main-container" >

    <ul>

        <li>Tahir</li>

    </ul>

</div>

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

# Nav

## Creating the Component

Create a nav component by using the following command. Nav will be placed inside the site folder.

* ng g c site/nav --skip-tests

1. Above command will create 3 files for us.
2. It will also add to app.module.ts, the import and declaration array.

## Adding the Nav html

Go to bootstrap website (installed version is Bootstrap v5.1.3)

1. Then go to example: <https://getbootstrap.com/docs/5.1/examples/>
2. Need a navbar with form so click Carousel
3. Right click and inspect
4. Select nav element and copy the element
5. Paste in the new create site/nav/nav.component.html
6. Also receive the title from the app.component.html where the nav will be placed so create an input decorator to receive the title
   1. Input decorator moves pass the data from parent 🡺 child
7. Then change the site/nav/nav.component.html as per the changes in the file

## Adding bootstrap js

Open angular.json and add the bootstrap js to it. Important: must do ng serve after this step

"scripts": [

              "node\_modules/bootstrap/dist/js/bootstrap.min.js"

            ]

## Adding app-nav to app component

Then go to app.component.html and add app-nav to it.

1. We are passing the title to the nav as well.
   1. <app-nav [title]="title"></app-nav>
   2. First [title] is the Input decorator
   3. Second title inside double quotes is the variable defined in app.componet.ts
2. Current title has been commented
3. Save and ng serve and you should see the app-nav

# Models/DTOs

## LoginDto Model

add a app/core/models/LoginDto.model.ts class in with two properties username and password.

* ng g class core/models/loginDto --type=model

export class LoginDto {

constructor(public userName: string = "", public password: string = "") {}

}

## UserTokenDto Model

add a app/core/models/userTokenDto.model.ts class in with two properties username and token.

* ng g class core/models/userTokenDto --type=model

export class UserTokenDto {

constructor(public userName: string = "", public token: string = "") { }

}

# Services

## ErrorMessageService

Create an http client error message reading service. It will receive the error and will return the message from it

* ng g s core/services/errorMessage --skip-tests

This has one method which is reading the error returned by observable and translates it into a message to display.

Check the method inside the service for details

### Important

Keep note of this since it will not be covered when creating other services. The basic structure of the service will be

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

@Injectable({

  providedIn: 'root'

})

export class ErrorMessageService{

  constructor() { }

}

* The important thing to note here is @Injectable. This means that this service will be injected into other components and services in our application.
* The metadata providedIn: 'root' means we do not need to add it to the providers array in app.module.ts.

## HttpClient Service

Create a service which with generic methods to make http calls.

* ng g s core/services/httpClient --skip-tests
* It will house the get, post, put and delete calls
* Check the service code for more details

## ApiUrl Service

Create an api url service in core/services folder

* ng g s core/services/apiUrl --skip-tests
* All api end point urls will be created here
* Public properties will be created to fetch the url from the service
* Check the service code for more details

## LocalStorage Service

Create a local storage service in core/services folder

* ng g s core/services/localStorage --skip-tests
* Will have methods to interact wit local storage
* Public properties will be created to get, set and remove item from local storage
* Check the service code for more details

## Account Service

Create an account service in core/services folder

* ng g s core/services/account --skip-tests
* Will use the ApiUrl Service to fetch the urls to use
* This service will use the httpClient service to make the http calls

# App.module.ts

Go to app.module.ts and add the following. These will be needed by the functionality in nav component.

* FormsModule to the imports array and import it as well.
* BsDropdownModule to the imports array and import it as well

Imports

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

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

Imports array:

FormsModule,

BsDropdownModule.forRoot()

# Nav Component TS and HTML updates

We have one form in the nav to login. It has a username, password and login button. We’ll convert this to Angular Template form

## nav.component.ts

* setup title with Input to be received from app.component.html where nav is being displayed
  + working with @Input decorator
* methods to login and logout
  + variables for storing the logged in user and login state
  + also ngDestroy implemented to unsubscribe the subscription
* Using @ViewChild decorator to work with the html elements
* method to close the navbar when it is open in mobile mode after
  + successful login
  + or other links in the navbar are clicked
  + working with @ViewChild decorator to target the div with id navbarCollapse

Here is the full code for it

import { Component, ElementRef, Input, OnDestroy, OnInit, Renderer2, ViewChild } from '@angular/core';

import { Subscription } from 'rxjs';

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

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

import { LoginDto } from '../../core/models/loginDto.model';

import { UserTokenDto } from '../../core/models/userTokenDto.model';

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

@Component({

  selector: 'app-nav',

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

  styleUrls: ['./nav.component.css']

})

export class NavComponent implements OnInit, OnDestroy {

  //getting passed in from app.component.html

  @Input() title = '';

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

  @ViewChild('navbarCollapse') navbarCollapseElement!: ElementRef;

  loginInfo: LoginDto = <LoginDto>{};

  userInfo: UserTokenDto = <UserTokenDto>{};

  error: string = "";

  isLoggedIn: boolean = false;

  private isExecutingLogin: boolean = false;

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

  loginSubscription!: Subscription;

  loginAlternateSubscription!: Subscription;

  currentUserSubscription!: Subscription;

  constructor(private accountService: AccountService, private errorMsgService: ErrorMessageService,

    private el: ElementRef, private renderer: Renderer2) { }

  ngOnInit(): void {

    //subscribe to the observable being fired from the account service

    this.getCurrentUser();

  }

  ngOnDestroy(): void {

    //unsubscribe

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

    if (this.loginAlternateSubscription) this.loginAlternateSubscription.unsubscribe();

    if (this.currentUserSubscription) this.currentUserSubscription.unsubscribe();

  }

  onLogin() {

    this.isExecutingLogin = true;

    if(environment.displayConsoleLog) console.log(this.loginInfo);

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

      next: r => {

        this.setUser(r, "onLogin");

        //clear login

        this.loginInfo = <LoginDto>{};

        //hide navbar in mobile mode

        this.onNavBarItemClickCloseNavBar();

      }, error: e => {

        this.displayError(e, "onLogin");

      }, complete: () => {

        //do something on complete

        this.isExecutingLogin = false;

      }

    });

  }

  onLogout() {

    //logout, due to persistence remove the user from the local storage as well

    this.accountService.logout();

    //hide navbar in mobile mode

    this.onNavBarItemClickCloseNavBar();

    //flag

    this.isLoggedIn = false;

  }

  //hide the navbar in mobile mode after an action has been performed

  onNavBarItemClickCloseNavBar() {

    //view child method to access the html elemnt

    const classToRemove = "show";

    if (this.navbarCollapseElement && this.navbarCollapseElement.nativeElement.classList.contains(classToRemove))

      this.navbarCollapseElement.nativeElement.classList.remove(classToRemove);

  }

  getCurrentUser() {

    //subscribe to the observable being fired from the account service

    this.currentUserSubscription = this.accountService.currentUser$.subscribe({

      next: user => {

        //getting fired on login as well but that is handled above in login so do not set the user again

        if (this.isExecutingLogin) return;

        this.setUser(user, "getCurrentUser");

      }, error: e => {

        this.displayError(e, "getCurrentUser");

      }, complete: () => {

        //do something on complete

      }

    });

  }

  setUser(user: UserTokenDto, from: string) {

    this.userInfo = user;

    if (environment.displayConsoleLog && this.userInfo) {

      console.log(`setUser-${from} UserName: ${this.userInfo?.userName}`);

      console.log(`setUser-${from} Token: ${this.userInfo?.token}`);

    }

    this.isLoggedIn = !!user;

  }

  displayError(error: any, from: string) {

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

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

    alert(`displayError-${from} Error: ${this.error}`);

  }

}

## nav.component.html

1. Use of Template driven form for login
2. Login button will be disabled when username or password are empty
3. Structural directive \*ngIf use to conditionally display content
4. Use of ngbootstrap dropdown
5. Calling the login and logout methods
6. Calling the onNavBarItemClickCloseNavBar to hide the navbar in mobile mode

Here is the full code

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

    <div class="container">

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

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

            <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" #navbarCollapse>

                <!--only show links when logged in-->

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

                    <li class="nav-item">

                        <a class="nav-link" aria-current="page" routerLink="/" (click)="onNavBarItemClickCloseNavBar()">Matches</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" routerLink="/" (click)="onNavBarItemClickCloseNavBar()">Lists</a>

                    </li>

                    <li class="nav-item">

                        <a class="nav-link" routerLink="/" (click)="onNavBarItemClickCloseNavBar()">Messages</a>

                    </li>

                </ul>

                <!--only show in logged in mode,

                    apply directives dropdown, dropdownToggle

                            and structural directive \*dropdownMenu-->

                <div \*ngIf="isLoggedIn" class="btn-group" dropdown>

                    <button type="button" class="btn btn-info">Welcome {{userInfo.userName}}</button>

                    <button id="button-split" type="button" dropdownToggle class="btn btn-info dropdown-toggle dropdown-toggle-split" aria-controls="dropdown-split">

                      <span class="caret"></span>

                      <span class="sr-only visually-hidden">Welcome {{userInfo.userName}}</span>

                    </button>

                    <ul id="dropdown-split" \*dropdownMenu class="dropdown-menu" role="menu" aria-labelledby="Welcome">

                      <li role="menuitem"><a class="dropdown-item" routerLink="/" (click)="onNavBarItemClickCloseNavBar()">Edit Profile</a></li>

                      <li class="divider dropdown-divider"></li>

                      <li role="menuitem"><a class="dropdown-item bg-danger text-white" (click)="onLogout()">Logout</a></li>

                    </ul>

                  </div>

                <!--show in not logged in mode-->

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

                    <input name="username" id="username" #username="ngModel" [(ngModel)]="loginInfo.userName" required class="form-control me-2" type="text" placeholder="username" aria-label="Username">

                    <input name="password" id="password" #password="ngModel" [(ngModel)]="loginInfo.password" required class="form-control me-2" type="password" placeholder="password" aria-label="Password">

                    <button class="btn btn-outline-success" type="submit" [disabled]="username.invalid || password.invalid">Login</button>

                </form>

            </div>

        <!--</div>-->

    </div>

  </nav>