# Angular Basics

## Important Angular classes

app. module.ts

bootstrap: [AppComponent]

Root of all components for this application. All components roll up here

imports: [

[other, modules] to make them globally usable throughout the application

component - Provides data for the view templates

package. json - Manages dependencies + versions. One level higher than src

angular. json - The app configuration

node\_modules - 100's of listed dependencies listed out

webpack -Compiles code to JS + injects JS into index.html (5 files)

## How to install bootstrap

npm install bootstrap font-awesome (installs both)

src>styles.css

@import "../node\_modules/bootstrap/dist/css/bootstrap.min.css";

@import "../node\_modules/font-awesome/css/font-awesome.min.css";

## Angular 3rd party libraries

Wrapping 3rd party libraries as an Angular service. Get a 3rd party app for cool alerts and turn it into an

Angular service. Do install, then make a service (so components can inject to use). make teh methods of

this service (success, fail, warn). based on the imported libraries, when used, altertify makes cool pop ups

for succes + message you pass. You use these DI this service into a component, then calling their methods

and passing in the custom message

Alertifyjs.com - Notifications (sucess/failures) from browser

npm install alertifyjs DatingApp > spa

spa > src > styles @import "../node\_modules/alertifyjs/build/css/alertify.min.css"

@import "../node\_modules/alertifyjs/build/css/themes/bootstrap.min.css";

spa> src > app > services ng g service alterify

import all alertify methods/libraries import \* as alertify from 'alertifyjs'

spa> src > new file typings.d.ts declare module 'alertifyjs'

spa> tsconfig.json

alertify.service.ts

Add confirm/success/error methods in this .ts then use them in other components

Import AlertifyService into constructor

login(){

this.service.login(this.model)

.subscribe(

next=>{this.alertify.success("")},

err =>{this.alertify.error(err)}

);

}

Angular JWT - Token helpers - 3rd party library to help manage tokens https://github.com/auth0/angular2-jwt

npm install @auth0/angular-jwt

import {JwtHelperService} from "@auth0/angular-jwt"

--Building this in service, this way you can check JWT status by injection vs in each component

jwtHelper = new JwtHelperService();

loggedin(){ //comming in as a service

const token = localStorage.getItem("token");

return !this.jwtHelper.isTokenExpired(token);

loggedIn(): boolean { --consumed by a component

return this.authService.loggedin();

NGX Bootstrap - 3rd party library to integrate bootstrap with Angular. Best practice is not to use jQuery but,

pure Angular and integrate Angular based libraries. https://valor-software.com/ngx-bootstrap/#/dropdowns

-basically after installed, just add some NGX specific key words to elements to use features.

install at the spa level ng add ngx-bootstrap

add NGX module to module root import { BrowserAnimationsModule } from '@angular/platform-browser/animations'

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

register module in imports import: [BrowserAnimationsModule,

BsDropdownModule.forRoot()]

<div dropdown> Keyword needed on surounding elem to indicate this is a dropdown

<a dropdowntoggle>Welcome Kw when using <a>, what you click on to display dropdown options (on-off)

<div \*dropdownMenu> The inner items are <a> link as drop down options.

<a>item1</> This Kw just groups these items that will be shown as drop-down options

<a>item2</> \* this is a structural directive that changes the DOM directly

-minor fixes in CSS of the nav bar we added a pointer when using the dropdown, dropdowntoggle, the items of teh drop down

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

-Hid the NavBar menu items to only show if a user is logged in. Targeted <ul> holding <li> each menu item

<ul class="navbar-nav mr-auto" \*ngIf="loggedIn()">

<li class="nav-item active">

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

-Drop down items were too close to the "dropdowntoggle" which shows >Welcome Username<

The "dropdownMenu" sourounding each item needed some margin-top so, they werent mashed together...to effect all grouped items

<div class="dropdown-menu mt-3" \*dropdownMenu>

Bootswatch - Change theme of design free (through Bootstrap) https://bootswatch.com/help/

npm install bootswatch install Bootswatch at the spa level

spa>node\_modules These are the project dependencies-> will show "bootswatch"

bootswatch>dist This holds all the avaialble themes, so you know path to point to

spa>app>src>styles.css @import "../node\_modules/bootswatch/dist/flatly/bootstrap.min.css";

from bootswatch, dropdown to theme used "flatly". <> means check out the code. Only one change was made

<nav class="navbar navbar-expand-lg navbar-dark bg-primary"> changed to bg-primary to match theme better

## FormsModule - Login <form>

<form> is just html until you activate the Angular capabilities for:

change tracking -Angular tracks changes for forms and fields behind the scenes

two-way binding - Turn on FormsModule and use [(ngModel)]

validation and error handling -You can test in your form for validation and state values

NgForm = a directive exported from FormsModule once you import it. It is the way to turn on Angular functionality for a form and use ng directives like (\*ngIf, [(ngModel)] ):

1. import FormsModule from the @angular/forms into spa>src>app>AppModule

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

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

@NgModule({

  declarations: [],

imports: [

    FormsModule,

…]

})

export class AppModule {}

1. Make a template reference inside of the <form> … This allows you to reference the template as a variable that, can be access outside of the form tag

<! --If not logged in...user/pass login fields will appear.  Removed once logged in-->

-- Template reference #loginForm = variable to assign "ngForm" directive

<form #loginForm="ngForm" \*ngIf= “! loggedIn()" class="form-inline my-2 my-lg-0">

--name= attribute used to assign and register the input + required

            <input class="form-control mr-sm-2”

 type="text" placeholder="UserName" required name="username"

                [(ngModel)] ="model.username">

            <input class="form-control mr-sm-2”

type="password" placeholder="Password" required name="password"

                [(ngModel)] ="model.password"> <--2-way binding to model:any={} in component

            <button [disabled]= “! loginForm.valid"

class="btn btn-success my-2 my-sm-0" type="submit"

                (click)="login()">Login</button>

</form>

## Interfaces in TypeScript

Interfaces in TS == classes in C# but, structured differently …TS does not have constructors. You just make the fields + optional fields … Make TS Interface to match C# class structure

xxx?: any = Optional filed must come after all required fields

Make a new folder “\_models” src>app>\_models

ng g i users to hold the Interfaces to match up with the classes

Example: In our case, we are using Dto's to only pass selected fields between the Spa < - > Api. So, we would replicate each Dto. For Dto's that are basically the same +/- a few fields.

public class UserForDetailedDto

    {

public int Id {get; set;}

public string UserName {get; set;}

public string Gender {get; set;}

public int Age {get; set;

public string KnownAs {get; set;}

public DateTime Created {get; set;}

public DateTime LastActive {get; set;}

public string City {get; set;}

public string Country {get; set;}

public string Introduction {get; set;}

public string LookingFor {get; set;}

public string Interests {get; set;}

public string PhotoUrl {get; set;}

public ICollection<PhotosForDetailedDto> Photos {get; set;}

src>app>\_models>user.ts

export interface User {

    id: number;

    userName: string;

    gender: string;

    age: number;

    knownAs: string;

    created: Date;

    lastActive: Date;

    city: string;

    country: string;

    photUrl: string;

    introduction?: string;

    lookingFor?: string;

    interests?: string;

    photos?: Photo[];

}

## Day XX:

Angular @Input @Output Property binding

This is how you pass data from one componet to another so that its avaialble

for the other components template

1. Transfer data from p->c receiving @input within the c component

(p)

P\_input:string;

<input type="text" [(ngModel)]="P\_input">

v

<app-child [parentTxBx] ="P\_input"> </app-child>

(C)

@Input("parentTxBx")parentTxBxVal: string;

<p>Value of Parent TextBox: {{parentTxBxVal}}

2. Transfer data from p->c receiving @input within the c component

(p)

pClick: Subject<void> =new Subject<void>();

OnPclick(){this.pClick.next();

v

<button (click)="OnPclick()">Parent Click</button>

v

<app-child [pBclick]="pClick" >

(c)

v

@Input("pBclick") parentClick:Subject<void>;

clickCount:number=0;

//On-click Subject observable is fed a new Subject

//Increment is called on clickCount

ngOnInit(): void {

this.parentClick.subscribe(()=>

this.incrementValue() );

}

incrementValue(): void{

this.clickCount= this.clickCount+1;

}

<div> Parent click count : {{parentClick}} </div>

3. Transfer data from c->p receiving @Output within the p component

src>app>members>member-list>member-list.component.html

 <div class="row">

        <div \*ngFor="let u of users" class="col-lg-2 col-md-3 col-sm-6">

 <! --Parent passes "input" to the [@Input variable name] of the child-->

            <app-member-card [user]="u"> </app-member-card>

        </div>

src>app>members>member-card>member-card.component.ts

@Component ({

  selector: 'app-member-card',

  templateUrl: './member-card.component.html',

  styleUrls: ['./member-card.component.css']

})

export class MemberCardComponent implements OnInit {

@Input() user: User;

-------------

## Angular validation

Testing the states (field & form states) that Angular is keeping track of behind the scenes. I am using a <div> on the template so, I can see real-time. For example, the field states will change when I put in information vs. having the fields blank.

1. Add reference var to the end of each <input> field … This allows us to access the tag in our div below #password="ngModel" - Allows for 2-way binding so, you can see the field states
2. The form already has its own template reference so, we can access the form for validation in the below div by accessing #loginForm
   1. A form will be considered “invalid” if the “required” fields are not populated
   2. This test shows our form is only valid when fields are populated but, we need a way to stop the user from logging in with invalid (fill out required fields).
      1. **Disable the <button> if form is not valid**
      2. <button [disabled]= “! loginForm.valid"
      3. <button [attribute] = “expression” > …</button>
      4. Button is only available once the form is valid (input all required fields)
3. This <div> can be removed after testing is complete

--This <div> is just an example of testing state/validation inside of the template

<! -- form-states that Angular is keeping track of -->

<div>

Form Valid: {{#loginForm.valid}} --Required fields populated entire form?  T/F

Form Values: {{#loginForm.value | json}}  --Values populate as you type

<! -- field-states that Angular is keeping track of -->

Password Valid: {{password. valid}} --valid = required field is populated

Password Value: {{password.value}} --2-way binding value will be printed

</div>

### (click) vs (ngSubmit) – Handlin a click event

These two do the exact same thing. The only difference is that one is done on the <button> while the other is within the <form> see the difference in syntax below

<form #loginForm="ngForm"  (ngSubmit)="login()">

-OR-

<button …(click)="login()"> Login </button>

### Reading data from the browser

Console: Message you send to the log

Network: Where you see your Req/resp info .. Click on indiv. req to see req/resp info

Application: Storage- Local/Session/Cookies, Cache-



## The \*ngIf structural directive

### \*ngIf directive on a method

True = Add objects inside of this <div> to the DOM

False = Items in this <div> will not be displayed in the DOM

<! --Drop Down menu will display if the User is logged in ...using NGXBootstrap-->

        <div class="dropdown" \*ngIf="loggedIn()" dropdown>

Conditional method above in the html template “loggedIn()” is being used as a switch for this \*ngIf=" …" The login form is triggering this method on-click event when user signs in

Below is the component.ts that contains the method that is being executed in the template

<!—-Does the “token” exists in the local storage (T/F) -->

loggedIn (): boolean {

      const token = localStorage.getItem("token");

        return !!token;  !!== If (token) -> true || else (! token) -> false

  }

template reference

The \*ngIf=” …” is constantly running and listening. So, when conditions changes, the values are updated and the switch if flipped (True / False)

### \*ngIf directive on a field value

Alternate pattern for toggling

1. make a variable in the component, set as False to be initially hidden

export class HomeComponent implements OnInit {

  registerMode: boolean = false;

1. set this variable inside the \*ngIf="…"

    <div class="container" \*ngIf="registerMode"> --hidden since currently false

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

            <div class="col-4">

                <app-register> </app-register> --Once \*ngIf becomes true this child

            </div> component will be displayed

        </div>

</div>

1. set an event (click)="toggle()" that will toggle a variables value (True -or- False) on-click

<div class="container mt-5">

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

        <h1>Find your match</h1>

        <p class="lead">View your matches... Sign up! <p>

        <div class="text-center">

            <button class="btn btn-primary btn-lg mr-2" (click)="registerToggle()">Register</button> --Toggle changes “registerMode” F/T

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

        </div>

    </div>

>

# Angular Services

Service = Centralizing the API req/resp to a single location vs. duplicating code in every class. Use services to create methods that communicate with API as a stream

Service Methods = Used to send Http Req (get, post, put, delete) -> get back a stream as the Response

1. Make a folder "\_services" to hold all of your services spa>src>app>\_services

2. ng g service auth -> component: auth.service.ts export class: AuthService

Components are auto-injectable but Services are not that is why “services” need @Injectable decorator

>src>app>\_services>auth.service.ts

@Injectable({

  providedIn: 'root' -- Which module is providing this service? The root module is so

}) this will be made available throughout the entire

src>app

export class AuthService { -- You can inject this instance to another class as DI

3. The service should get auto-registered in spa>app>**app.module.ts**

providers: [AuthService, ErrorInterceptorProvider], --src>app is a root level module

4. Import {Observable} from 'rxjs';

import {map} from "rxjs/operators";

Services use Observables because they use HttpClient to communicate with REST API's

In order to do something with a Resp (from server). You need to use rxjs operators

You pass these operators(filter/map/reduce) through a pipe method so, you can chain rxjs operators to the request.

The. pipe returns Observable<void> ...remember, Observables are lazy so, they don't return anything. You must subscribe to them to elicit the data

export class AuthService {

  baseUrl: string = environment.apiUrl + "auth/";

  jwtHelper = new JwtHelperService();

  decodedToken: any;

  constructor(private http: HttpClient) { }

login(model: any): Observable<void> { -- interprets the body as JSON and

returns it as an Observable in JSON

    return this.http.post(this.baseUrl + "login", model) -- return this.http.post("url", obj)

      .pipe( pipe= transform the response to your request as Observable

        map((response: any) => {   = transform incoming values on-at-a-time

          const user = response; -- incoming resp is a bearer (“token”: "token string")

          if (user) {

            localStorage.setItem("token", user.token);

//decode (v) token string and hold as var

            this.decodedToken = this.jwtHelper.decodeToken(user.token);

//to see the fields of jwt decode method in browser

            console.log(this.decodedToken);

/\* console.log("user: " + this.decodedToken.unique\_name);    --> fields comes from authController ...claims + tokenDescriptor

             this.decodedToken.exp , this.decodedToken.iat, this.decodedToken.nbf,

this.decodedToken.nameid\*/

          }

        }) //--map transform one-at-a-time

      ); //--pipe transform to Observable

  }

--localstorage = store (k, v) in a web browser.

--So, data is persisted when you close the browser or, refresh the page.

--The data is still there (persisted). (-) You can only store strings, store as json

import { map } from "rxjs/operators";

### rxjs Operators Examples

These are examples of how you might use rxjs operators in a pipe to transform an observable

const filterOutWithEvens = filter(x => x % 2)

const doubleByValue = x => map(value => value \* x);

const sumValue = reduce((acc, next) => acc + next, 0);

const source$ = Observable.range(0, 10)

source$.pipe(

filterOutWithEvens,

doubleByValue(2),

sumValue)

.subscribe(console.log); // 50

## Base Url

Base Url = Http address to connect to the Api. It is just a property that you want to use throughout the entire app

Create base Url property for entire spa

src>environments>environment.ts (for non-production environment ...for now)

src>environments>environment.prod.ts (production ...for later)

src>environments>environment.ts

export const environment = {

  production: false,

  apiUrl: "http://localhost:5000/api/"

};

--This gives us an option to switch the api source across the entire application

When we run in production. We can just update the env.prod file with the final location.

Just change Url for production api. Therefore, we don’t have to change any of the "services" we have already created.

Since the base Url “apiUrl” is in src>environment. You import the object to use its properties. Here is an example of how you would access the Api Url in this “service”

src>app>\_services>user.services.ts

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

export class UserService {

  //apiUrl: "http://localhost:5000/api/"

  baseUrl: string = environment.apiUrl;

## Manually adding headers to Requests being sent to the Api

Here is what we have done so far:

1. Injected HttpClient into the constructor

2. Make methods that return an observable. One of the param options allows for headers that we can just pass into the parameter of the method

Example: This service methods is calling the Api’s endpoint to get back something of a certain type. In this case we are expecting a collection of “user objects” (actually user.Interface <-> UserDTO ). In the method parameter, we are passing the (endpointUrl, headers)

3. Manually create a new header

a) Create a variable below imports section to hold Header info to pass with the req

b) Create a new HttpHeader (K:V) this must be a legit header name

purpose: After login a user has populated their username/pass that are being stored in the browser's local storage. We want to grab that authentication info which is the saved token. Remember that the server passed the token as a Jwt Bearer (token : string) which had user details and a server signing key. We want to pass this info back to the server with this request so that we are authenticated with a valid token

c) Add the headers to the method issuing the request just add at end of method params

src>app>\_services>user.services.ts

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

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

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

import { Observable } from 'rxjs';

import { User } from "../\_models/user";

const httpOptions = {

  headers: new HttpHeaders({

    "Authorization": "Bearer " + localStorage.getItem("token")

  })

}

@Injectable({

  providedIn: 'root'

})

export class UserService {

  //apiUrl: "http://localhost:5000/api/"

  baseUrl: string = environment.apiUrl;

  constructor(private http: HttpClient) { }

  getUsers(): Observable<User[]> {

    return this.http.get<User[]>(this.baseUrl + "users", httpOptions);

  }

  getUser(id): Observable<User[]> {

    return this.http.get<User[]>(this.baseUrl + "users/" + id, httpOptions);

  }

}

# Angular routing

How to set up

We need somewhere to route when people/click on links in the navbar to go to the appropriate

page (Angular component). Here are a few examples of components that are auto-generated when they are created as declarations: [components that roll up to app root] in app.module

@NgModule ({

  declarations: [

    AppComponent,

    ValueComponent,

    NavComponent]

Routes should be made in the app level src>app

echo > routes.ts Makes a file that will contains routes for our application

You must import Routes from @angular/router'

src>app>appRoutes.ts

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

export const appRoutes: Routes = []

How to set route paths?

Define Routes= [] as an array of objects {}. Each object is a Url paths that map to your -> components

Order is important.

export const appRoutes: Routes = [

{path: "", component: HomeComponent},

{path: "members", component: MemberListComponent},

{path: "\*\*", redirectTo: "", pathMatch: "full"}

]

]

Register your routing in the app at the app root level "app.module". This allows the entire app to use any registered modules + auto-registers components to roll up to this app-root src>app>app.module.ts

src>app>app.module.ts

import {RouterModule} from "@angular/router";

@NgModule({

  declarations: [

    someComponent,

],

imports: [

    BsDropdownModule.forRoot(),

    RouterModule.forRoot(appRoutes) --configure this module with any routes we provide from our

angular routing “appRoutes.ts”

  ],

### How to use [routerLink] in html templates for <a> navigation

My Nav component has <a> that drop down for navigation. I want to connect those to my routing to direct people to the appropriate URL a- is a shortcut [routerLink]="['/routePath']" routerLinkActive="router-link-active"

--Adding [routerLink] to the nav.html active = light-up on click/hover

<ul class="navbar-nav mr-auto" \*ngIf="loggedIn ()">

<li class="nav-item" routerLinkActive="router-link-active">

    <a class="nav-link" [routerLink]="['/members']">Matches </a>

  </li>

--<a> on click will take you to the Url http://host/members

--Change app.component.html structure to accommodate routing -> RouterOutlet structure

src>app>app.component.html

<app-nav></app-nav>

<router-outlet></router-outlet>

Here you can see that only the nav bar is being displayed on the site but, in our routing (Routes path), we have any non-specific path “\*\*” to default to our home component. This means that both “app-nav” and “home” will be displayed when the application starts

The way to navigate around the rest of the site is to click the <a> which will have “routerLinks” with rout paths that change the Url and take you to other components.

### Using methods to Navigate to new components

When a customer logs in. The login () is called. When this happens, we want the routing to change

When a customer logs out. The logout () is called. When this happens, we want the routing to change

To use routing for a method inside of a class. We need to inject Routing for that class

Inject routing as a service

Make routing change when method called

Subscribe to a service’s observable (api call) …call routing from inside the .subscribe() ( w/ subscribe param)

src>app>nav.component.ts

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

export class NavComponent implements OnInit {

  model: any = {};

 constructor (public authService: AuthService, private alertify: AlertifyService,

    private router: Router) {} <----inject routing

  ngOnInit (): void {

  }

  login(): void {

    this.authService.login(this.model)

      .subscribe(

        next => { //next=Req is successful + Resp has data from api

          this.alertify.success("Logged in successfully");

        },

        error => {this.alertify.error(error); },

--route to this Url after all successful req/resp ----

        () => {this.router.navigate(['/members']); }    <-- Subscribe param used

   Effect routing change

--Do this after all next’s have been called ----------

      );

  }

//alternate use complete (): void after success req/resp

1. Effect routing from an observable method by subscribing to it

Subscribing to a service api call. Here are the subscription parameters:

next = Req is successful, response from api has token (our request has our log in details)

error = data stream error that we are grabbing/passing and returning to browser via alertify

complete= () => {…after all next's are called, …do this, …returns void}

2. Effect routing from regular method

Inject routing as a service

this.router.navigate(['…']);

src>app>nav.component.ts

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

export class NavComponent implements OnInit {

  model: any = {};

 constructor (public authService: AuthService, private alertify: AlertifyService,

    private router: Router) {} <----inject routing

  ngOnInit (): void {

  }

logout(): void {

    localStorage.removeItem("token");

    this.alertify.message("logged out");

    this.router.navigate(['']); <-- call routing to change when this method called

  }

\*Router change in methods = router.navigate(['…']) in html = [routerLink]="['/lists']"

### Routeguards

Routeguards protect routes from unauthorized users. implements CanActivate ... can this route

-Make Folder + guard class: be activated (t/f)

src>app mdir \_guards Make a dir to hold auth guards for routes

cd \_guards ng g guard auth --skipTests Make guard with no tests(spects.ts)

-which interfaces to use...space to select

-select "CanActivate" return

auth.guard.ts is kind of like a separate component. I stripped off most return options since this is just a bool to

see if, a user is logged in or not. If not logged in, they can only access the home page. Inject the guard into the constructor of routes.ts to use this protection.

src>app>\_guards>auth.guard.ts

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

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

@Injectable({

  providedIn: 'root'

})

Protection for url path routes.  Has a single method (t/f) to verify if this path can be accessed or not

//

export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService,

    private alertify: AlertifyService, private router: Router) { }

  canActivate(): boolean {       //Method call of canActivate true = user can proceed

//Call service's method to check if tokens exist for login

    if (this.authService.loggedin())

      return true;                   //if true this means user has logged in

    else {

      this.alertify.error("Area:  Access denied");

    }

  }

}

Example using a guard in the appRoute configuration class routes.ts to effect a single route routes.ts

src>app>routes.ts

import {AuthGuard} from "./\_guards/auth.guard";

export const appRoutes: Routes = [ --can only access this route if guard = true

{path: "members", component: MemberListComponent, canActivate: [AuthGuard]}

\*canActivate: [] Is a type that can hold an array of guards

Holds an array of dependency-injection tokens used to look up CanActivate() handlers, to see if, the rout is T or F.

### Using guards to protect multiple routes

Routes are treated as separate objects {path: “”, component}

Here we want to take a rout and give it a bunch of children {path: “…”, guard stuff, children: [\_\_{c1}, {c2} \_\_] }

This accomplishes nesting multiple routes as children inside an object rout

src>app>routes.ts

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

import {HomeComponent} from "./home/home.component";

import {MemberListComponent} from "./member-list/member-list.component";

import {MessagesComponent} from "./messages/messages.component";

import {ListComponent} from "./list/list.component";

import {AuthGuard} from "./\_guards/auth.guard";

export const appRoutes: Routes = [

    {path: "", component: HomeComponent},

    {

        path: "“, //localhost:4200 + “…” path + add Url path

//always run guards for these children. verify if this path can be accessed or not

runGuardsAndResolvers: "always”,

        canActivate: [AuthGuard], //This single guard (one of many)

        children: [

            {path: "members", component: MemberListComponent},

            {path: "messages", component: MessagesComponent},

            {path: "lists", component: ListComponent}

        ]

    },

    {path: "\*\*", redirectTo: "", pathMatch: "full"}

]

# AuthO JwtModule

<https://github.com/auth0/angular2-jwt>

This works in conjunction with JwtHelperService import which helps us manage our tokens. Used to decode the token and see if it is expired or not. In this library you also get an HttpInterceptor which attaches the web token to the client request. In our application we are using a token for validation and a request every-time we interact with the api except

-Login () which sends username, password and gets back a new token

-Register () which send new user information

In all other activities we are requesting interaction with the api and need token info

Usage Injection - How set-up AuthO

src>app.module

import {JwtModule} from "@auth0/angular-jwt";

import {HttpClientModule} from "@angular/common/http";

export function tokenGetter () {

return localStorage.getItem("your\_token\_name\_here");

}

@NgModule({

bootstrap: [AppComponent],

imports: [

// ...

HttpClientModule,

JwtModule.forRoot({

config: {

tokenGetter: tokenGetter, --This is the method we just made above as an export

allowedDomains: ["example.com", “localhost:5000”],

disallowedRoutes: ["http://example.com/examplebadroute/", “needs full path/api/auth”],

--disallowed is optional

},

}),

],

})

export class AppModule {}

Previously we had manually added a variable to hold a new HttpRequestHeader … the header was passing our token info and putting at the end of the request params….in “service” as we sent requests to the api. We no longer need this http option. Because

**This library relies on the URL interface which is not supported in IE11.** To solve the IE11 compatibility, you can add a polyfill.

* run  npm i --save url-polyfill
* add  import 'url-polyfill'; to  polyfills.ts   in your project

# Dotnet Stuff

## Add Coors policy

CORS - Browser security feature that protects you from accessing data from an unknown origin.

Our API is a foreign site (localhost:5000) being Requested by angular (localhost:4200)

A cors policy is needed else an ERROR - "No Access-Control-Allow-Origin" header is coming from request

app.UseRouting();

            app.UseAuthentication();

            app.UseAuthorization();

            app.UseCors(req => req.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

## Global exception handling

properties -> launchSetting.json "Development" -or- switch to "Production"

you want to use global exception handling to limit try/catch blocks in ea class

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

if (env.IsDevelopment()) Exception Handler adds middleware to the

{ pipeline. That catches the exceptions

app.UseDeveloperExceptionPage(); logs it-> re-executes the request in an

} alternate pipeline...so, no unhandled, this

else is handling ...I.E equivenant of global try/catch

{

app.UseExceptionHandler(builder => This needs to be configured and returns IApplicationBuilder

} Set all exceptions as 500, output our custom Extensions class method "AddApplicationError"

passes our error to the response. Return s status 500 w/ message in the headers

## System.Security.Crytography

These are Objects you can create that can scramble inputs like passwords

using Crypto object method like computeHash can encode passwords to a hash of bytes.

These Crypto objects have this secret hash output and also provide a key.

With this key you can replicate the hash given another Crypto object

The unique key is called a "Salt”, use this key to reverse-engineer. Just create another Crypto object and pass it the previous key. Now It's output will be the same as your original Crypto object

/// <summary>

        /// Convenience method to verify If a given password matches the encryted password in the DB

        /// </summary>

        /// <param name="passWord">User supplied Login password</param>

        /// <param name="passwordHash">Encripted password from the DB</param>

        /// <param name="passwordSalt"> Previous Key given from Security. Cryptography object that generates same hash, given same password...from same key</param>

        /// <returns>True = Encrypted passwords match</returns>

        /// ----

        private bool VerifyPasswordHash(string passWord,

byte[] passwordHash, byte[] passwordSalt)

        {

            using (var hmac = new System.Security.Cryptography.HMACSHA512(passwordSalt))

            {

                var computedHash = hmac.ComputeHash(System.Text.Encoding

.UTF8.GetBytes(passWord));

                for (int i = 0; i < computedHash.Length; i++)

                {

                    if (computedHash[i] != passwordHash[i])

                        return false;

                }

                return true;

            }

        }

### Passwords

Passwords in Db are stored as byte [] password hash + byte [] salts(key) ... not visible text

Even if two passwords are the same, they will have a different Hash values because, a new security crypto object (unique key) is created making a new key

That new key will output a completely different hash specific to that object. Each and every time a user registers

--New User with password flow

Controller --> Register method--> authRepo method (encrypts the password + track + save) --> DB

authRepo method (returns created object) --> Controller’s Register method (returns status code 201)

## Registering Services in the Startup class

By registering Services, you make them injectable throughout the App

public void ConfigureServices(IServiceCollection services)

{

services.AddSingleton<I\_Repo, Impl\_Repo>(); = Single instance created on 1st request

This instance is shared across all new requests. Problem when you have concurrent requests

services.AddTransient<I\_Repo, Impl\_Repo>(); = New instance created of repo for each http request

Good for stateless services

services.AddScoped<I\_Repo, Impl\_Repo>(); = Created only once per Req w/i the same scope

Uses this single instance when other calls made as part of this Request

The user's initial Request is grouped as a scope so, same instance is used throughout

Good for an Authentication request because they are tied to a single session

## API Controllers

IActionResult = Http Response

async Task= A task that doesn't block other Req’s in thread while waiting for returned results

Task<User> = Just a User obj that is wrapped in a Task for Threading (uses async await pattern)

using Microsoft.AspNetCore.Mvc; =We are using MVC

[Route("api/[Controller]")]

    [ApiController] =We are enforcing attribute routing | remove =std

    public class UsersController: ControllerBase

    {

:ControllerBase = MVC (C) with NO Views : Controller = MVC (C) Allows Views

[ApiController] = Does a lot of work for us. The [] annotations that tell the server

That the param is an object vs. null is no longer required. Therefore, we do not need

the below ModelState validation

        [HttpPost("register")]

        public async Task<IActionResult> Register([FromBody] ObjType varNm)

        {

If (! ModelState.IsValid()) return BadRequest (ModelState) -- no longer needed

# Entity Framework Core

## Entity Framework commands

dotnet ef -h dotnet ef migrations -h

dotnet ef migrations add <migrationName> Adds updates to existing migration file <update name>

dotnet ef database update Update these changes to the DB

dotnet new -h listing of all available projects to be built

dotnet new webapi -n DatingApp.API ->Creates new project. NetCore as type web API called …

dotnet --info Display all your sdk's + info

## When Class Fields || Properties are changed in you App

\*Assume that new properties have been added or deleted

--> The DB entities are now different than the Models in our application classes

--> Required: Make updates in EF Core to account for these changes

Option 1: Normal flow

dotnet ef migrations add <someNameOfMigration> Adds Migration

dotnet ef database update Commits to the DB

Option 2: Oops I need to adjust something 1st

dotnet ef migrations add <someNameOfMigration> Adds Migration

ef migrations remove Removes recent most migration

--Make any fixes you need 1st, add another migration

dotnet ef database update Commits to the DB

"dotnet ef migrations list" shows all existing migrations

Option 3: Oops I already committed to the DB

Once you commit to the DB, you cannot undo the commit so, you will have to wipe the DB which is ok in development because you are just using dummy data

dotnet ef database drop gets rid of our DB and all tables + data

--> Migrations are still intact but, the actual Db is gone

dotnet ef migrations remove removes most recent migration (back to previous migration)

dotnet ef database update Recreates the schema (DB) up to the existing migrations

--> You are now back to the previous committed migration

## EF Core Foreign Keys

Navigation Property = Manually define the relationship by placing an instance as a member field

Manually map this relationship to the other class = object field + id field

public class User

{

//Property Navigation

public ICollection<Photo> Photos {get; set;}

}

public class Photo

{

//Property navigation resolution to solve for cascading deletes

        public User Users {get; set;}

        public int UserId {get; set;}

}

-If the relationship is not mapped (obj + id) = Restricted delete (onDelete: ReferentialAction.Restrict)

EF Core will still make the relationship but, will hold the FK as nullable. What this means is that, if we delete one. The other is orphaned. They float around and still exist regardless of which (User or connected photo) is deleted

-When relationship has a Navigation property + mapping

EF Core will create a cascading delete so, when a User is deleted, all photos will be deleted. onDelete: ReferentialAction.Restrict); vs. onDelete: ReferentialAction.Cascade);

If using SQLite you need to add these NuGet packages to your csproj file

Microsoft.EntityFrameworkCore

Microsoft.EntityFrameworkCore.Design

Microsoft.EntityFrameworkCore.Sqlite

## Repository Pattern

The Repo\_Interface is the Layer between Repo and Controllers that gets injected into the Controller.

Repo exposes its methods for the (C) to use so, the (C) does not directly talk with the DB

private readonly IMapper \_mapper;

        private readonly IDatingRepository \_repo;

        public UsersController(IDatingRepository repo, IMapper mapper)

        {

            \_mapper = mapper;

            \_repo = repo;

        }

Since the Repo interface is injected into the (C), the controllers logic does not need to change. All changes about talking to the DB happen in the in the Repository.

This is called “Separation of concerns” so, (C) only deals with (Http Request-Response)

Repo deals solely with talking to DB. You register Repo as a service in the Startup file so, it can be injected anywhere in the App.

\_context = Middleware (Program <-> DB) The repo Implementation will communicate with the DB

Before entities are saved to the DB, all actions need to be tracked (all changes in the entities state)

Else it cannot be persisted(saved) … state needs to be added … in order to be saved

Examples of tracking entity state -> Save to DB

await \_context. User.AddAsynch (userObject); --ADD command Tracks changes to the domain. model

await \_context. SaveChangesAsynch (); -- SAVECHANGES () command to persist any tracked changes

without the Asych part, an Exception would be thrown vs. Null

FirstOrDefaultAsynch (x => x. Id == id) returns null if not found

## Seeding data using JSON---- https://www.json-generator.com/

file.json goes into the data folder (Repo’s live here) app>data

make new class Seed.cs

Method -->Requires Json serialization text->object obj->DB

Seeding only works w/ empty DB "dotnet ef database drop"

public class Seed

    {

        public async static void SeedUsers(DataContext context)

        {

            //Check if DB is empty ... if so, lets seed the data

            //

            if (!context.Users.Any())

            {

                //Read the .json file in Data folder, hold as text

                //

                var userData = System.IO.File.ReadAllText("Data/UserSeedData.json");

                //Use Newtonsoft to convert Json <-> .Net

                //Read json data and convert into .NET User objects

                //Hold Users as a List ... pass text as a param

                //

                var users = JsonConvert.DeserializeObject<List<User>>(userData);

                foreach (var u in users)

                {

                    //Hord code the passwords for dummy data

                    byte[] passwordHash; byte[] passwordSalt;

                    CreatePasswordHash("1234", out passwordHash, out passwordSalt);

                    u.PasswordHash = passwordHash;

                    u.PasswordSalt = passwordSalt;

                    await context.Users.AddAsync(u);

                }

                await context.SaveChangesAsync();

            }

        }

In Program.cs call static Seed.SeedUsers(context) to save dummy data to the DB

ASP.NET core 2.2 vs. 3.0 ---Difference in the way JSON serialization happens and returns data

previously Newtonsoft.json was used to serialize obj responses to text -> defaults to System.Text.Json

3.0 requires that you register Microsoft.AspNetCore.Mvc.NewtonsoftJson w/ NuGet to register in csproj

<PackageReference Include="Microsoft.AspNetCore.Mvc.NewtonsoftJson" Version="3.1.7"/>

NewstonsotJson also needs to be registered as a service in Startup.cs

// Called at runtime. Shell to add + hold services for DI.

        public void ConfigureServices(IServiceCollection services)

        {

            services.AddControllers().AddNewtonsoftJson(

                opt =>

                {

                    opt.SerializerSettings.ReferenceLoopHandling =

                    Newtonsoft.Json.ReferenceLoopHandling.Ignore;

                }

            );

# DTO’s

Used to map our classes (coming from DB as entities) into more simple objects

Keep Dto’s in a separate folder

When you make a Dto class, end its name with Dto nameDto.cs

public class Photo

    {

 public int Id {get; set;}

 public string Url {get; set;}

 public string Description {get; set;}

 public DateTime DateAdded {get; set;}

 public bool IsMain {get; set;}

//Property navigation resolution to solve for cascading deletes

        public User Users {get; set;}

        public int UserId {get; set;}

    }

public class PhotosForDetailedDto

{

public int Id {get; set;}

public string Url {get; set;}

public string Description {get; set;}

public DateTime DateAdded {get; set;}

public bool IsMain {get; set;}

//Only difference is that DTO does not include the Property navigation.

--This DTO is used to flatten out the data. Client should not see details of the User who the Photo belongs to

}

Dto’s are generally caught/sent by the (Controller) as an object –Since they have the info we want from form/user

We convert Dto to the corresponding object entity in the (Controller)

This conversion Dto -> Entity needs to happen … otherwise we would be unable to … save to the DB

Day 4: https://jwt.io/

## Server-Side Validation

Data Annotations done on the model that is interacting with the Client

public class UserForRegisterDto

    {

[Required]

[MinLength(4, ErrorMessage = "Must be at least 4 characters")]

        public string Username {get; set;}

[Required]

[StringLength(50, MinimumLength = 4, ErrorMessage = "Must be between 4-50 characters")]

        public string Password {get; set;}

    }

## Using An AutoMapper---- map Entities/models <-> DTO's

1. Use Nuget to install AutoMapper. This gets installed in the csproj file

<PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="8.0.1"/>

1. Register AutoMapper into the Services container. This allows you to inject AutoMapper into the app

public void ConfigureServices(IServiceCollection services)

        {

--//target of Automapper= Repo...translation of entities to DTO (shaped data)

services.AddAutoMapper(typeof(DatingRepository).Assembly);

3. Inject the AutoMapper in the Controller ...b/c Controller deals w/ shaped data

OUT - Shaped data to client as Resp

IN - Client only interacts/sees what we want them to

NOT exact fields as they exist in DB

using AutoMapper;

[Route("api/[Controller]")]

    [ApiController]

    public class UsersController: ControllerBase

    {

        private readonly IMapper \_mapper;

        private readonly IDatingRepository \_repo;

        public UsersController(IDatingRepository repo, IMapper mapper)

        {

            \_mapper = mapper;

            \_repo = repo;

        }

4. Returning Mapped values from the DTO

[HttpGet("{id}")]

        public async Task<IActionResult> GetUser(int id)

        {

            var user = await \_repo.GetUser(id);

--Transform to return DTO < Destination\_Out\_theDTO > (input\_source\_IN)

            var userToReturn = \_mapper.Map<UserForDetailedDto>(user);

            return Ok(userToReturn);

        }

5. Tell AutoMapper about the mappings we need it to support

Separate folder called "Helpers" this folder is stuff that assists other classes I.E. Extensions for global error handling -or- Mapper for creating a relationship between the entity/Model <-> DTO

A helper class is needed to extend Profiles. this class's purpose is to understand the source(IN) and the destination(OUT) of what is being mapped. That’s why we are extending "Profiles"

namespace DatingApp.API.Helpers

{

    /// <summary>

    /// This AutoMapper defines what can be mapped as a source & output

    /// The entity class is the source

    /// The DTO is the destination/output

    /// </summary>

    public class AutoMapperProfiles: Profile

    {

// Destination mappings are done in the constructor

        //

        public AutoMapperProfiles ()

        {

-- We now have two DTO mapping that are supporting the User entity / class

CreateMap<User, UserForListDto> ();

CreateMap<User, UserForDetailedDto> ();

## Fixing DTO output to Client

**How to fix common Navigational Property issues (part 1):**

**Passing DTO’s with DTO member fields**

At current the User...DTO is returning the full user Object inside of the Photo.cs since Photo has

Navigational Property of (User + UserId ...to establish FK) which return a lot of data we don't want

In our Controller method (step 4) we made adjustments so that, the DTO would be returned vs. the entity object.

We need to make sure that the DTO is only pulling data from the PhotoDTO vs. the entity object. Since the PhotoDTO does not have mapping properties (obj + id). The data returned will be flattened. Otherwise full User details will be displayed showing passwords, salts etc.…

Notice that one DTO has an instance of the other DTO. Now only these DTO properties will be passed, not the full entity object details

The Controller is now only passing the data we want to show from our DTO’s

public class UserForDetailedDto

    {

public string PhotoUrl {get; set;}

 public ICollection<PhotosForDetailedDto> Photos {get; set;}

}

public class PhotosForDetailedDto

    {

public int Id {get; set;}

public string Url {get; set;}

public string Description {get; set;}

public DateTime DateAdded {get; set;}

public bool IsMain {get; set;}

    }

**How to fix common Navigational Property issues (part 2):**

How to configure AutoMapping relationships using .ForMember() Helpers>AutoMapperProfile

Remember that our Mapping relationship is between User object and UserDto

problem1: At current the UserDto that is being sent as the response from the Controller. Is not showing the "PhotoUrl" property = null

public class User

    {

        //Property Navigation

        public ICollection<Photo> Photos {get; set;}

--User.cs -> Photo.cs -> a Photo property

public class UserForDetailedDto

    {

public string PhotoUrl {get; set;}

 public ICollection<PhotosForDetailedDto> Photos {get; set;}

Reason: The User entity object has an ICollection<Photo> so, it has access to Photo class. However, User obj does not have a field that exists in DTO "PhotoUrl" (only exists in DTO). The User obj cannot pass info it does not have.

solve: Configure the mapping, between User entity and Dto to assign the “PhotoUrl” property from Photos entity -> PhotosDto.

I.E. find a way to get the "PhotoUrl" prop from Photo.cs instance that's inside of User object.

var Url = \_context.Users.Include(u=> u.Photos).FirstOrDefault(p => p.IsMain).url

Mapper is between two classes so, does not make sense to make call from DB like above

The solution needs to be between the DTO grabbing data from the Entity. This code needs to happen inside of the AutoMapper.

Using. ForMember () to solve for specific Property fields which Transforms a single member = Pass single property from the entity -> dto

ForMember (destMbr, mbrOptions) dest=dto mbrOption= use opts to get User obj details

--add a configuration to the existing mapping

CreateMap<User, UserForListDto>()

                .ForMember(dest => dest.PhotoUrl,     //destination member = dto.prop

                    opt => opt.MapFrom(                     //opt to map src =User obj

                        src => src.Photos.FirstOrDefault(    //All Photos for this User

                            p => p.IsMain == true            //Return 1st where IsMain

                        ).Url                        //Of all those a single will return

                    )                             //For that one, give me url prop field

                )

problem2: DateTime DOB property in User obj is null when it is passed to Dto as int Age.

public DateTime DateOfBirth {get; set;}

public int Age {get; set;}

reason: can’t pass User's DateTime DOB property -> Dto's int Age property. They are different types so, the Automapper cannot convert w/o more instructions

solution: We want this property to take CurrDate (Yr) - DOB(Yr) = Age

Convert User's DateTime DOB through a custom "extension method" that can take a DateTime as an argument. Do the above calculation and return an int. When an int is returned, the AutoMapper can now pass this int value to the DTO int Age value. Basically, we need to map another individual property.

-->append existing CreateMap<src, dest>() .ForMbr(...) w/ another .ForMbr(...) to get this field and pass the updated value. Since this mapping is already solving user->userDto we make additions to this existing CreateMap relationship

Making custom extension method ... add new method to existing extension class Helpers>Extensions

new method "CalculateAge" will extend DateTime ... extension methods should be static. The parameter will take a (this DateTime dt) which means that this method can accept the DateTime as a chain vs. direct param

public static class Extensions

    {

public static int CalculateAge(this DateTime theDateTime) -- extends this DateTime

        {

       var age = DateTime.Today.Year - theDateTime.Year; --return int Yr but,has dt info

       if (DateTime.Now.DayOfYear < theDateTime.DayOfYear) --if dob>cur doy not bday yet

       {

           age--;            //cannot combine calc + return, must be on sep lines

           return age; //Else will not calc the age= age-1

       }

       else return age;

        }

# Token Authentication

JWT = Json Web Tokens (industry standard) https://jwt.io/

Token are used to authenticate the client against the API server.

Users cannot make up their own token and send to the server. Even if they have the correct username b/c the server has its own signature. Each token must be exact to verify against the server

The server uses a crypto object with a key to ensure that the token is valid...The token cannot be manipulated as it will not pass check. The token doesn't go to DB to validate, it calls the token itself

JwTokens are passed back from client to the server to authenticate the user.

Since JWT are signed with a secure key, no DB call to server is needed to check if authenticated

Because, they are sending request with the Token we have already provided, this ensures authentication for a given period of time

Tokens have self-contained properties (Credentials, Claims to the application + more)

JWT is a text file in this format "Token" : "Token string" (k, v)

1.Header ("algoUsed”: encryted algorithm “type”: "JWT")

What kind of security is used?

2.Payload (info stored inside the token) \*\*careful - easily decoded

Body that has some info about response

3.Secret (Used to Hash 1+2) stored on the server and never revealed to the client

This is a signature. Basically, an encryted string that comes from server. Only this exact string can be sent back to verify each token.

The request to the server must come with this exact signature + user log-in credentials

Secret being passed needs to match ... else not a valid token … not authorized

## Identity Claims in the Controller method

Claims are how we build and approve the identity of a User. Claims we are making about their identity

-->Login credentials are received in a method of the Controller

--> The username and password are verified against the DB

-->Since UserName + Password are correct we start building the Claims

var claims = new []

            {

                new Claim (ClaimTypes.NameIdentifier, userFromRepo.Id.ToString() ),

                new Claim (ClaimTypes.Name, userFromRepo.UserName)

            };

Authentication Middleware you will need - Once you have authorization set up in your Controller

1. using Microsoft.IdentityModel.Tokens this Controller
2. using System.IdentityModel.Tokens.Jwt this Controller
3. using Microsoft.AspNetCore.Authentication.JwtBearer Startup

## Adding Authentication as a service

The application needs to know which Type of Authorization you are using. So, the authentication needs to be registered as a service in “Startup.cs” to be used throughout the application

        // Called at runtime. Shell to add + hold services for DI.

        public void ConfigureServices(IServiceCollection services)

        {

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

                .AddJwtBearer(options =>

                {

                    options.TokenValidationParameters = new TokenValidationParameters

                    {

                        //--options to validate against our JWT auth

                        ValidateIssuerSigningKey = true,

                        IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII

                            .GetBytes(Configuration.GetSection("AppSettings:Token").Value)),

                        ValidateIssuer = false,

                        ValidateAudience = false

                    };

                });

The Http pipeline also needs to be set up in the “Startup.cs”

// This is middleware to interact w/ Req on its journey to deliver a Resp

        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

        {

. . .

app.UseRouting();

            app.UseAuthentication(); --Can this Request(User) be authorized

            app.UseAuthorization(); --If so, what can they do

-- This must happen before app.UseEndpoint / UseMvc(). Because, we want to short circuit the http pipeline and protect the Controller and its methods from unauthorized users

app.UseCors(req => req.AllowAnyOrigin().AllowAnyMethod().AllowAnyHeader());

app.UseEndpoints(endpoints => // Endpoint API's rout incomming Req to Controllers

            {

                endpoints.MapControllers();  // Controllers map to the endpoints

            });

## Testing Authentication in Postman

Security Annotations that can be put on a Controller class or, individual method

[Authorize]  = class attribute  = Each request requires authorization Token

[AllowAnonymous]  = method attribute =  No authorization token required

Postman … Paste in the request url

-> Log in by sending “post” with username + password in the body { "username" : "Lola", "password" : "1234"}

-> Grab the token from the response body "token":” …”

-> Make a new “get” request

-> Go to Headers

-> Add a new Key value “Authorization”

-> Input the value “Bearer stringValueOftheToken”

## (1) Hiding the token key from being pushed to GitHub

The normal way would be to stage the changes and publish them to GitHub ...secrets are pushed

= Source control -> stage (+) -> "message" -> commit (enter)

How to hide a file using app>. gitIgnore file

--> add the file -or- file type you want to omit from being pushed to Git

--> git rm appsettings.json –cached Removes from staging and keeps any changes as local. Does not go with push request

.vscode --This is the .gitIgnore file

bin

obj

\*.db

//appsettings.json --This file is holding the secret string that our app is encrypting …

{ --This is the appsettings.json file

  "AppSettings": {

    "Token": "…secret value here…” <-- This should not be on a public server like GitHub

  },

  "ConnectionStrings": {

    "defaultConnection": "Data Source =datingapp.db"

  },  "Logging": {

## (2) Hiding the token key from being pushed to GitHub

Use this option for DEV mode only. In production you must use ENV Variables in production

1. dotnet user-secrets init Adds secrets GUID to <PropertyGroup> in csproj

<PropertyGroup>

    <TargetFramework>netcoreapp3.1</TargetFramework>

    <UserSecretsId>d316194c-5298-416c-84f9-1611dc803cc2</UserSecretsId>

  </PropertyGroup>

("K" "V") - from appsettings.json set as a secret locally

1. dotnet user-secrets set "AppSettings:Token" "secret string value"

dotnet user-secrets list Retrieves token value from local

1. Delete the string value from “AppSettings": {“Token": "..."} from appsettings.json because, it is now being saved as a secret locally on your computer

------------------