34 Урок

Добавляем [Authorize] в самом начале

В ValuesController и подключаем библиотеку к нему.

В методе Getvalue подключаем сверху [AllowAnonymous]

В класс Startup подключаем сервис:

services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options => {

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuerSigningKey = true,

IssuerSigningKey = new SymmetricSecurityKey(Encoding.ASCII.GetBytes(Configuration.GetSection("AppSettings:Token").Value)),

ValidateIssuer = false,

ValidateAudience = false,

};

});

И приложение, которое сможет его использовать ( перед app.UseMVC()):

app.UseAuthentication();

Проверяем в постмане, как приходит запрос. Если хотим получить values – то видим unathorization, если конкретное значение – видим его.

Переходим в header и в значение key подставляем Authorization, в parameters – вводим Bearer и через пробел наш токен, после чего получаем все значения.

Коммитим.

35 Урок

Копируем nav-bar из темы bootstrap <https://getbootstrap.com/docs/4.1/examples/jumbotron/>

В DatingApp-SPA в src/app создаем новый компонент – Generate Component назовем его nav

Подключаем этот компонент через добавление его селектора в app.component.html и настраиваем сам компонент в его файле nav.component.html

36 Урок

Переходим в nav.component.ts и добавляем в наш класс модель:

model: any = {};

и также добавляем метод:

login() {

console.log(this.model);

}

В app.module.ts импортируем:

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

и вставляем в декоратор @ngModule import: FormsModule

в nav.component.html в форм добавляем 2 параметра:

<form #loginForm="ngForm" class="form-inline my-2 my-lg-0" (ngSubmit)="login()">

А в input такие параметры:

<name="username" required [(ngModel)]="model.username">

<name="password" required [(ngModel)]="model.password">

37 Урок

Создаем папку \_services в src/app,

Генерируем в ней generate service – называем его auth,

Импортируем и подключаем наш сервис в app.module.ts в providers: [] ( подключаем по названию класса)

Создаем переменную в классе: baseUrl = 'http://localhost:5000/api/auth/';

В конструктор constructor(private http: HttpClient) { }

Под конструктором создаем метод login:

login(model: any) {

return this.http.post(this.baseUrl + 'login', model).pipe(map((response: any) => {

const user = response;

if (user) {

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

}

})

);

38 Урок

В nav.component.ts добавляем в конструктор:

constructor(private authService: AuthService) { }

и импортируем:

import { AuthService } from '../\_services/auth.service';

в методе login заменяем тело:

login() {

this.authService.login(this.model).subscribe(next => {

console.log('Logged in successfully');

}, error => {

console.log('Failed to login');

});

}

И добавляем еще 2 метода:

loggedIn() {

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

return !!token;

}

logout() {

localStorage.removeItem('token');

console.log('logged out');

}

Поместим наши методы внутрь template:

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

И

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

Добавим еще один лист для logout:

<li class="nav-item">

<a class="nav-link" (click)="logout()">Logout</a>

</li>

39 Урок

Создаем 2 компонента в папке src/app – generate component – home, generate component – register,

Редактируем [home.component.html](http://home.component.html)

<div class="container mt-5">

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

<h1>Find your match</h1>

<p class=lead>Come on in to view your matches... All you need to do is dign up!</p>

<div class="text-center">

<button class="btn btn-primary btn-lg mr-2" (click)="registerToggle()">Register</button>

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

</div>

</div>

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

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

<div class="col-4">

<app-register></app-register>

</div>

</div>

</div>

</div>

Добавляем наш selector в app.component.html

После этого делаем похожие манипуляции с папкой register, в register.component.html вставляем:

<form #registerForm="ngForm" (ngSubmit)="register()">

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

<hr>

<div class="form-group">

<input type="text" class="form-control" required name="username" [(ngModel)]="model.username" placeholder="Username">

</div>

<div class="form-group">

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

</div>

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

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

<button class="btn btn-default" type="button">Cancel</button>

</div>

</form>

После чего в register.component.ts, в классе создаем 2 метода и модель:

model: any = {};

register() {

console.log(this.model);

}

cancel() {

console.log('cancelled');

}

И изменяем [home.component.ts](http://home.component.ts), добавив:

registerMode = false;

и

registerToggle() {

this.registerMode = !this.registerMode;

}

40 Урок

Копируем их value.component.ts метод getValues() в наш [home.component.ts](http://home.component.ts),

В конструктор инжектируем http:

constructor(private http: HttpClient) { }

Импортируем его:

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

Добавляем values в класс:

values: any;

Определяем его в ngOnInit:

ngOnInit() {

this.getValues();

}

Удаляем компоненты Value ( и папку и из app.module)

Также в [home.component.html](http://home.component.html) редактируем:

<app-register [valuesFromHome]="values"></app-register>

В register.component.ts добавляем:

@Input() valuesFromHome: any;

И импортируем его:

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

И в app.component.html удаляем <app-value></app-value>

В register.component.html добавляем:

<div class="form-group">

<label for="favouriteValue">What is your favourite value?</label>

<select class="form-control" id="favouriteValue">

<option \*ngFor="let value of valuesFromHome" [value]="value">{{value.name}}</option>

</select>

</div>

41 Урок

В register.component.ts добавляем @Output() и импортируем его вместе с EventEmitter:

@Output() cancelRegister = new EventEmitter();

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

В методе cancel() добавляем:

this.cancelRegister.emit(false);

в [home.component.html](http://home.component.html) редактируем:

<app-register [valuesFromHome]="values" (cancelRegister)="cancelRegisterMode($event)"></app-register>

И добавляем этот метод в [home.component.ts](http://home.component.ts):

cancelRegisterMode(registerMode: boolean) {

this.registerMode = registerMode;

}

Редактируем :

registerToggle() {

this.registerMode = true;

}

42 Урок

В auth.service.ts создаем регистрационный метод:

register(model: any) {

return this.http.post(this.baseUrl + 'register', model);

}

In register.component.ts constructor arguments add:

private authService: AuthService

and import

import { AuthService } from '../\_services/auth.service';

and change register methods to:

register() {

this.authService.register(this.model).subscribe(() => {

console.log('registration successful');

}, error => {

console.log(error);

});

}

In [home.component.ts](http://home.component.ts) remove getValues methods, remove it in ngInit, remove from class our values: any, from register.component.html – we need to remove from group :

<div class="form-group">

<label for="favouriteValue">What is your favourite value?</label>

<select class="form-control" id="favouriteValue">

<option \*ngFor="let value of valuesFromHome" [value]="value">{{value.name}}</option>

</select>

</div>

And inside [home.component.html](http://home.component.html) we need to remove:

[valuesFromHome]="values"

Inside <app-register>, because we add that for demonstrate.

Inside register.component.ts we remove @input property,

Секция 5 ( обработчики ошибок )

43 Урок

Для примера в AuthController.cs in to [HttpPost("login")]

Add

throw new Exception("Computer says no!");

if we see the combinations 500 internal server error and No “Access-Control-Allow-origin” – we look only at the first error.

So if we want handled this error – we can use try…catch block.

44 Lesson ( Создаем описание ошибки в Headers для Production mode)

In Startup.cs Configure method we add to block else :

app.UseExceptionHandler(builder => {

builder.Run(async context => {

context.Response.StatusCode = (int)HttpStatusCode.InternalServerError;

var error = context.Features.Get<IExceptionHandlerFeature>();

if (error != null)

{

context.Response.AddApplicationError(error.Error.Message);

await context.Response.WriteAsync(error.Error.Message);

}

});

});

And add :

using System.Net;

using Microsoft.AspNetCore.Diagnostics;

using Microsoft.AspNetCore.Http;

using DatingApp.API.Helpers;

После чего создаем папку в корневом Dating.API с названием Helpers, в которой создаем C# class Extensions.cs, который делаем статическим, чтобы не создавать new instance, а можно было сразу вызывать.

Class выглядит следующим образом

using Microsoft.AspNetCore.Http;

namespace DatingApp.API.Helpers

{

public static class Extensions

{

public static void AddApplicationError(this HttpResponse response, string message)

{

response.Headers.Add("Application-Error", message);

response.Headers.Add("Access-Control-Expose-Headers", "Application-Error");

response.Headers.Add("Access-Control-Allow-Origin", "\*");

}

}

}

45 lessons (handling errors in Angular)

В DatingApp-Spa – src – app \_services создаем error.interceptor.ts

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

import { HttpInterceptor, HttpRequest, HttpHandler, HttpEvent, HttpErrorResponse, HTTP\_INTERCEPTORS } from '@angular/common/http';

import { Observable, throwError } from 'rxjs';

import { catchError } from 'rxjs/operators';

@Injectable()

export class ErrorInterceptor implements HttpInterceptor {

intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>>{

return next.handle(req).pipe(

catchError(error => {

if (error instanceof HttpErrorResponse) {

if (error.status === 401) {

return throwError(error.statusText);

}

const applicationError = error.headers.get('Application-Error');

if (applicationError) {

console.error(applicationError);

return throwError(applicationError);

}

const serverError = error.error;

let modalStateErrors = '';

if (serverError && typeof serverError === 'object') {

for (const key in serverError) {

if (serverError[key]) {

modalStateErrors += serverError[key] + '\n';

}

}

}

return throwError(modalStateErrors || serverError || 'Server Error');

}

})

);

}

}

export const ErrorInterceptorProvider = {

provide: HTTP\_INTERCEPTORS,

useClass: ErrorInterceptor,

multi: true

};

Добавляем его в app.module.ts – providers

providers: [

AuthService,

ErrorInterceptorProvider

],

import { ErrorInterceptorProvider } from './\_services/error.interceptor';

переходим в nav.component.ts и заменяем login:

login() {

this.authService.login(this.model).subscribe(next => {

console.log('Logged in successfully');

}, error => {

console.log(error);

});

}

И таким образом получаем сообщение ошибки в nav.component.ts при попытке зайти под bob – password.

Переходим в AuthController.cs и удаляем throw new Exception(“Computer says no!”);

В результате при неправильном логине получаем ошибки, при пустой регистрации получаем ошибку.

Module 6

Wraping 3rd party libraries as an Angular service

Install Alertify.js – библиотека по отображению всплывающих окон

Устанавливаем в datingApp-SPA\ через npm

npm install alertifyjs -–save

В angular.json в scripts добавляем:

"scripts": [

"node\_modules/alertifyjs/build/alertify.min.js"

]

А в src/app – style.css добавляем 2 зависимости:

@import '../node\_modules/alertifyjs/build/css/alertify.min.css';

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

После чего генерируем новый сервис:

Src/app/\_services – generate Service, называем его alertify, в alertify.service.ts прописываем:

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

declare let alertify: any;

@Injectable({

providedIn: 'root'

})

export class AlertifyService {

constructor() { }

confirm(message: string, okCallback: () => any) {

alertify.confirm(message, function(e) {

if (e) {

okCallback();

} else {}

});

}

success(message: string) {

alertify.success(message);

}

error(message: string) {

alertify.error(message);

}

warning(message: string) {

alertify.warning(message);

}

message(message: string) {

alertify.message(message);

}

}

И в app.module.ts в providers добавляем наш класс:

providers: [

AuthService,

ErrorInterceptorProvider,

AlertifyService

],

В nav.component.ts добавляем в constructor:

constructor(private authService: AuthService, private alertify: AlertifyService) { }

и метод login заменяем на:

login() {

this.authService.login(this.model).subscribe(next => {

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

}, error => {

this.alertify.error(error);

});

}

И logout:

logout() {

localStorage.removeItem('token');

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

}

Далее переходим в register.component.ts и также заменяем constructor:

constructor(private authService: AuthService, private alertify: AlertifyService) { }

и редактируем методы :

register() {

this.authService.register(this.model).subscribe(() => {

this.alertify.success('registration successful');

}, error => {

this.alertify.error(error);

});

}

cancel() {

this.cancelRegister.emit(false);

}

Пробуем зарегистрироваться, зайти и выйти и видимо, что все работает.

**Using the Angular JWT library to improve token handling**

Устанавливаем npm install @auth0/angular-jwt@2.0.0 в DatingApp-SPA

Переходим в auth.service.ts и добавляем под baseURL переменную:

jwtHelper = new JwtHelperService();

импортируем его:

import {JwtHelperService} from '@auth0/angular-jwt';

и создаем новый метод:

loggedIn() {

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

return !this.jwtHelper.isTokenExpired(token);

}

Переходим в nav.component.ts и редактируем loggedIn метод:

loggedIn() {

return this.authService.loggedIn();

}

Проверяем, регистрация все также работает.

**Using the Angular JWT library to decode token**

Переходим в auth.service.ts и добавляем под baseURL переменную:

decodedToken: any;

после чего редактируем метод login:

login(model: any) {

return this.http.post(this.baseUrl + 'login', model).pipe(map((response: any) => {

const user = response;

if (user) {

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

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

console.log(this.decodedToken);

}

})

);

}

Переходим в nav.component.html и в нем заменяем welcome user на

Welcome {{ authService.decodedToken?.unique\_name | titlecase}}

Переходим в nav.component.ts и заменяем authService в constructor’e private на public:

constructor(public authService: AuthService, private alertify: AlertifyService) { }

открываем app.component.ts и редактируем его:

import { JwtHelperService } from '@auth0/angular-jwt';

import { AuthService } from './\_services/auth.service';

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

@Component({

selector: 'app-root',

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

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

})

export class AppComponent implements OnInit {

jwtHelper = new JwtHelperService();

constructor(private authService: AuthService) {}

ngOnInit() {

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

if (token) {

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

}

}

}

**Adding Ngx Bootstrap to power our Bootstrap component**

Переходим на сайт: https://valor-software.com/ngx-bootstrap/#/ v.3.0.1 – get started

npm install ngx-bootstrap@3.0.1 --save

По ссылке:

<https://valor-software.com/ngx-bootstrap/old/3.0.1/#/dropdowns>

по мануалу вставляем в imports – app.module.ts:

BsDropdownModule.forRoot()

И вставляем импорт:

import { BsDropdownModule } from 'ngx-bootstrap';

После чего в nav.component.html добавляем, в класс dropdown добавляем еще dropdown, dropdownToggle в класс dropdown-toggle text-light и \*dropdownMenu, в класс dropdown-menu mt-3

После чего все со 2ой строчки обернем в div class container, чтобы сделать отступы.

Переходим в nav.component.css и создаем:

.dropdow-toggle, .dropdown-item {

cursor: pointer;

}

А в nav.component.html в итоге все выглядит следующим образом:

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

<div class="container">

<a class="navbar-brand" href="#">Dating App</a>

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

<li class="nav-item active">

<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>

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

<a class="dropdown-toggle text-light" dropdownToggle>

Welcome {{authService.decodedToken?.unique\_name | titlecase}}

</a>

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

<a class="dropdown-item" href="#"> <i class="fa fa-user"></i> Edit Profile</a>

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

<a class="dropdown-item" (click)="logout()"> <i class="fa fa-sign-out"></i> Logout</a>

</div>

</div>

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

<input class="form-control mr-sm-2" type="text" name="username" placeholder="Username" required [(ngModel)]="model.username">

<input class="form-control mr-sm-2" type="password" name="password" placeholder="Password" required [(ngModel)]="model.password">

<button [disabled]="!loginForm.valid" class="btn btn-success my-2 my-sm-0" type="submit">Login</button>

</form>

</div>

</nav>

**Bringing some color to our app with a theme from bootswatch**

Устанавливаем bootswatch - npm install [bootswatch@4.1.1](mailto:bootswatch@4.1.1)

В style.css, после boostrap импортируем:

@import '../node\_modules/bootswatch/dist/united/bootstrap.min.css';

And in nav.component.html need change this line on **bg-primary**:

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

Section 7

Setting up routing in Angular

Dating-App-Spa – src – app – Generate Component – member-list

Dating-App-Spa – src – app – Generate Component – lists

Dating-App-Spa – src – app – Generate Component – messages

Dating-App-Spa – src – app –Create file – routes.ts, в который пишем:

import { ListsComponent } from './lists/lists.component';

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

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

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

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

export const appRoutes: Routes = [

{path: 'home', component: HomeComponent},

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

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

{path: 'lists', component: ListsComponent},

{path: '\*\*', redirectTo: 'home', pathMatch: 'full'}

];

In app.module.ts :

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

import { appRoutes } from './routes';

imports: [

BrowserModule,

HttpClientModule,

FormsModule,

BsDropdownModule.forRoot(),

RouterModule.forRoot(appRoutes)

Setting up our links in the nav component

In nav.component.html we change <ul> and <a>:

<a class="navbar-brand" [routerLink]="['/home']">Dating App</a>

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

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

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

</li>

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

<a class="nav-link" [routerLink]="['/lists']">Lists</a>

</li>

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

<a class="nav-link" [routerLink]="['/messages']">Messages</a>

</li>

</ul>

In app.component.html we need change :

<app-home></app-home>

To

<router-outlet></router-outlet>

**Using routing in our component (при входе и выходе переходим на другие страницы)**

In nav.component.ts – we add to contructor:

constructor(public authService: AuthService, private alertify: AlertifyService, private router: Router) { }

and

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

change login method (add routing):

login() {

this.authService.login(this.model).subscribe(next => {

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

}, error => {

this.alertify.error(error);

}, () => {

this.router.navigate(['/members']);

});

}

And change logout method (add routing)::

logout() {

localStorage.removeItem('token');

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

this.router.navigate(['/home']);

}

**Protect our routes with a route guard (защищаем наш роутинг, от несанкционированного доступа)**

Dating-App-Spa – src – app – add – ne folder \_guards

cd .\src\app\\_guard

ng g guard auth --spec=false

в результате у нас появляется auth.guard.ts, в который мы внесем свои изменения, что в результате у нас получится:

import { AuthService } from '../\_services/auth.service';

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

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

import { AlertifyService } from '../\_services/alertify.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

constructor(private authService: AuthService, private router: Router, private alertify: AlertifyService) {}

canActivate(): boolean {

if (this.authService.loggedIn()) {

return true;

}

this.alertify.error('You shall not pass!!!');

this.router.navigate(['/home']);

return false;

}

}

Далее переходим в app.module.ts и:

providers: [

AuthService,

ErrorInterceptorProvider,

AlertifyService,

AuthGuard

],

и

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

перейдем в routes.ts и изменим одно поле path на:

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

пробуем зайти на эту страницу не будучи залогинеными и получаем нашу ошибку!

**Protecting multiple routes with a single route guard using dummy routes**

Переходим в routes.ts и создаем новый раздел, в котором в children вставляем наши path:

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

import { ListsComponent } from './lists/lists.component';

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

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

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

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

export const appRoutes: Routes = [

{path: '', component: HomeComponent},

{

path: '', // localhost:4200/thispathmembers

runGuardsAndResolvers: 'always',

canActivate: [AuthGuard],

children: [

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

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

{path: 'lists', component: ListsComponent}

]

},

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

];

Module 8 API

Extending the User class

DatingApp.API – models - users.cs и добавляем в него свойства:

using System;

using System.Collections.Generic;

namespace DatingApp.API.Models

{

public class User

{

public int Id { get; set; }

public string Username { get; set; }

public byte[] PasswordHash { get; set; }

public byte[] PasswordSalt { get; set; }

public string Gender { get; set; }

public DateTime DateOfBirth { get; set; }

public string KnownAs { get; set; }

public DateTime Created { get; set; }

public DateTime LastActive { get; set; }

public string Introduction { get; set; }

public string LookingFor { get; set; }

public string Interests { get; set; }

public string City { get; set; }

public string Country { get; set; }

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

}

}

Также импортируем 2 библиотеки и создаем класс Photo -> в models

using System;

namespace DatingApp.API.Models

{

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; }

}

}

Переходим в DataContext.cs и добавляем

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

Останавливаем сервер dotnet в терминале, ипишем :

dotnet ef migrations add ExtendedUserClass

**Exploring Entity Framework migrations ( решаем вопрос с удалением фотографий)**

Если мы хотим удалить нашу миграцию, мы можем это сделать командой:

Dotnet ef migrations remove

Добавляем еще раз нашу миграцию и апдейтим ее:

dotnet ef migrations add ExtendedUserClass

dotnet ef database update

Если вдруг мы захотим вернуться назад после апдейта –> делаем дроп датабейз, миграции остаются доступными:

dotnet ef database drop

после чего можем выполнить:

Dotnet ef migrations remove

И теперь можем проапдейтить нашу базу данных:

Dotnet ef database update

**Entity framework relationship**

Переходим в photo.cs и добавляем 2 свойства:

public User User { get; set; }

public int UserId { get; set; }

После чего делаем:

dotnet ef migrations add ExtendedUserClass

и

dotnet ef database update

Теперь у нас в файле миграции name: photos - удаление каскадом и мы сможем нормально удалять фото и userId nullable:false;

**Seeding Data to the Database- part1**

Go to - <https://www.json-generator.com/>

Вставляем нашу заготовку для json для female из 8 урока

Идем в Data и создаем UserSeedData.json – вставляем то, что сгенерировало на сайте

Возвращяемся на сайт и делаем тоже самое, только меняем female на male в имени, gender и ссылке.

**Seeding Data to the Database- part2**

In Data we need to create new c# class, call Seed.cs

В который заполняем:

using System.Collections.Generic;

using DatingApp.API.Models;

using Newtonsoft.Json;

namespace DatingApp.API.Data

{

public class Seed

{

private readonly DataContext \_context;

public Seed(DataContext context)

{

\_context = context;

}

public void SeedUsers()

{

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

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

foreach (var user in users)

{

byte[] passwordHash, passwordSalt;

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

user.PasswordHash = passwordHash;

user.PasswordSalt = passwordSalt;

user.Username = user.Username.ToLower();

\_context.Users.Add(user);

}

\_context.SaveChanges();

}

private void CreatePasswordHash(string password, out byte[] passwordHash, out byte[] passwordSalt)

{

using(var hmac = new System.Security.Cryptography.HMACSHA256())

{

passwordSalt = hmac.Key;

passwordHash = hmac.ComputeHash(System.Text.Encoding.UTF8.GetBytes(password));

}

}

}

}

Go to Startup.cs and add service:

services.AddTransient<Seed>();

and in configure args add:

public void Configure(IApplicationBuilder app, IHostingEnvironment env, Seed seeder)

and add seader over app:

seeder.SeedUsers();

пишем:

dotnet run и обновляем нашу БД – у нас появились записи ( Если выдает ошибку – сначала дропаем БД dotnet ef database drop, потом апдейтим ее dotnet ef database update, после чего снова запускаем dotnet run).

После чего комментируем:

// seeder.SeedUsers();

**Creating a new repository for our API**

In Data folder create IDatingRepository c# interface:

using System.Collections.Generic;

using System.Threading.Tasks;

using DatingApp.API.Models;

namespace DatingApp.API.Data

{

public interface IDatingRepository

{

void Add<T>(T entity) where T: class;

void Delete<T>(T entity) where T: class;

Task<bool> SaveAll();

Task<IEnumerable<User>> GetUsers();

Task<User> GetUser(int id);

}

}

Create new c# class in Data – call name DatingRepository and implement interface IDatingRepository:

using System.Collections.Generic;

using System.Threading.Tasks;

using DatingApp.API.Models;

using Microsoft.EntityFrameworkCore;

namespace DatingApp.API.Data

{

public class DatingRepository : IDatingRepository

{

private readonly DataContext \_context;

public DatingRepository(DataContext context)

{

\_context = context;

}

public void Add<T>(T entity) where T : class

{

\_context.Add(entity);

}

public void Delete<T>(T entity) where T : class

{

\_context.Remove(entity);

}

public async Task<User> GetUser(int id)

{

var user = await \_context.Users.Include(p => p.Photos).FirstOrDefaultAsync(u => u.Id == id);

return user;

}

public async Task<IEnumerable<User>> GetUsers()

{

var users = await \_context.Users.Include(p => p.Photos).ToListAsync();

return users;

}

public async Task<bool> SaveAll()

{

return await \_context.SaveChangesAsync() > 0;

}

}

}

Go to startup.cs and add services:

services.AddScoped<IDatingRepository, DatingRepository>();

**Creating the users controller**

Add new c# class in Controllers folder – UsersController.cs

using System.Threading.Tasks;

using DatingApp.API.Data;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace DatingApp.API.Controllers

{

[Authorize]

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

[ApiController]

public class UsersController : ControllerBase

{

private readonly IDatingRepository \_repo;

public UsersController(IDatingRepository repo)

{

\_repo = repo;

}

[HttpGet]

public async Task<IActionResult> GetUsers()

{

var users = await \_repo.GetUsers();

return Ok(users);

}

[HttpGet("{id}")]

public async Task<IActionResult> Getuser(int id)

{

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

return Ok(user);

}

}

}

Go to launchsetting.js and change enviroments to development:

"ASPNETCORE\_ENVIRONMENT": "Development"

In startup.cs change service to:

services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version\_2\_1)

.AddJsonOptions(opt => {

opt.SerializerSettings.ReferenceLoopHandling =

Newtonsoft.Json.ReferenceLoopHandling.Ignore;

});

Then check in postman <http://localhost:5000/api/users/2> or <http://localhost:5000/api/users>:

In headers – we set Autharization and in key add Bearer and add the token:

Bearer eyJhbGciOiJIUzUxMiIsInR5cCI6IkpXVCJ9.eyJuYW1laWQiOiIxIiwidW5pcXVlX25hbWUiOiJyb3NhIiwibmJmIjoxNTQ3Mzc4MzI2LCJleHAiOjE1NDc0NjQ3MjYsImlhdCI6MTU0NzM3ODMyNn0.5VeH857bmPaZpYEqhKVents7awdb5DUFESsfmohk-e7Kd12ls1OY5fxX-tR5DSWGm7c1yFSNJ5wPogI4arqIdw

Then send and take the result.

**Shaping the data to return with DTOs (Уменьшение полученной информации через API)**

Go to DTOs folder and add new c# class call – UserForListDto.cs

using System;

namespace DatingApp.API.Dtos

{

public class UserForListDto

{

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 PhotoUrl { get; set; }

}

}

Then in DTOs folder create new c# class – UserForDetailedDto.cs

using System;

using System.Collections.Generic;

using DatingApp.API.Models;

namespace DatingApp.API.Dtos

{

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 Introduction { get; set; }

public string LookingFor { get; set; }

public string Interests { get; set; }

public string City { get; set; }

public string Country { get; set; }

public string PhotoUrl { get; set; }

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

}

}

**Using AutoMapper part1 (Приводим из юзер класса в юзер DTOs list и detailed по названию поля)**

Install AutoMapper - Ctrl+shift+p – NugetPackage manager – Add package – Automapper and press return – Automapper.extensions.Microsoft.DependencyInjection – 4.0.1

And we see in DatingApp.Api.csproj add new column:

<PackageReference Include="AutoMapper.Extensions.Microsoft.DependencyInjection" Version="4.0.1"/>

Go to startup.cs and add service under AddCors():

services.AddAutoMapper();

and import:

using AutoMapper;

Go to UsersController.cs and change to:

using System.Collections.Generic;

using System.Threading.Tasks;

using AutoMapper;

using DatingApp.API.Data;

using DatingApp.API.Dtos;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace DatingApp.API.Controllers

{

[Authorize]

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

[ApiController]

public class UsersController : ControllerBase

{

private readonly IDatingRepository \_repo;

private readonly IMapper \_mapper;

public UsersController(IDatingRepository repo, IMapper mapper)

{

\_mapper = mapper;

\_repo = repo;

}

[HttpGet]

public async Task<IActionResult> GetUsers()

{

var users = await \_repo.GetUsers();

var usersToReturn = \_mapper.Map<IEnumerable<UserForListDto>>(users);

return Ok(usersToReturn);

}

[HttpGet("{id}")]

public async Task<IActionResult> Getuser(int id)

{

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

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

return Ok(userToReturn);

}

}

}

Go to Helpers folder and create new c# class call – AutoMapperProfiles.cs

using AutoMapper;

using DatingApp.API.Dtos;

using DatingApp.API.Models;

namespace DatingApp.API.Helpers

{

public class AutoMapperProfiles : Profile

{

public AutoMapperProfiles()

{

CreateMap<User, UserForListDto>();

CreateMap<User, UserForDetailedDto>();

}

}

}

Перезапускаем наш сервер – dotnet watch run

**Using AutoMapper part2 (фиксим вывод фотос и нормализуем параметр age)**

In Dtos folder we’re create new c# class – PhotosForDetailedDto.cs

using System;

namespace DatingApp.API.Dtos

{

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; }

}

}

Go to UserForDetailedDto.cs and change –

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

Go to AutoMapperProfile and create new:

CreateMap<Photo, PhotosForDetailedDto>();

После этого нам нужно нормально отображать поле PhotoUrl, для этого изменим наш первый метод :

CreateMap<User, UserForListDto>();

На :

CreateMap<User, UserForListDto>()

.ForMember(dest => dest.PhotoUrl, opt => {

opt.MapFrom(src => src.Photos.FirstOrDefault(p => p.IsMain).Url);

});

Также импотируем:

using System.Linq;

по аналогии делаем и со вторым методом UserForDetailedDto:

CreateMap<User, UserForDetailedDto>()

.ForMember(dest => dest.PhotoUrl, opt => {

opt.MapFrom(src => src.Photos.FirstOrDefault(p => p.IsMain).Url);

});

In extension.cs add new method:

public static int CalculateAge(this DateTime theDateTime)

{

var age = DateTime.Today.Year - theDateTime.Year;

if (theDateTime.AddYears(age) > DateTime.Today)

age--;

return age;

}

And import:

using System;

in AutoMapperProfile change the first CreateMap method to:

CreateMap<User, UserForListDto>()

.ForMember(dest => dest.PhotoUrl, opt => {

opt.MapFrom(src => src.Photos.FirstOrDefault(p => p.IsMain).Url);

})

.ForMember(dest => dest.Age, opt => {

opt.ResolveUsing(d => d.DateOfBirth.CalculateAge());

});

And change second CreateMap method:

CreateMap<User, UserForDetailedDto>()

.ForMember(dest => dest.PhotoUrl, opt => {

opt.MapFrom(src => src.Photos.FirstOrDefault(p => p.IsMain).Url);

})

.ForMember(dest => dest.Age, opt => {

opt.ResolveUsing(d => d.DateOfBirth.CalculateAge());

});