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.