INF 354 Notes

Application Security

Contents

[API 2](#_Toc138256741)

[Login 2](#_Toc138256742)

[Register 3](#_Toc138256743)

[JWT Token 4](#_Toc138256744)

[Front-end 5](#_Toc138256745)

[Login 5](#_Toc138256746)

[Register 7](#_Toc138256747)

[Api.service.ts 8](#_Toc138256748)

[Logout 8](#_Toc138256749)

[Example (with roles) 9](#_Toc138256750)

# API

## Login

[HttpPost]

[Route("Login")]

public async Task<ActionResult> Login(UserViewModel uvm)

{

var user = await \_userManager.FindByNameAsync(uvm.emailaddress);

if (user != null && await \_userManager.CheckPasswordAsync(user, uvm.password))

{

try

{

var principal = await \_claimsPrincipalFactory.CreateAsync(user);

return GenerateJWTToken(user);

}

catch (Exception)

{

return StatusCode(StatusCodes.Status500InternalServerError, "Internal Server Error. Please contact support.");

}

}

else

{

return NotFound("Does not exist");

}

}

## Register

[HttpPost]

[Route("Register")]

public async Task<IActionResult> Register(UserViewModel uvm)

{

var user = await \_userManager.FindByIdAsync(uvm.emailaddress);

if (user == null)

{

user = new AppUser

{

Id = Guid.NewGuid().ToString(),

UserName = uvm.emailaddress,

Email = uvm.emailaddress

};

var result = await \_userManager.CreateAsync(user, uvm.password);

if (result.Errors.Count() > 0) return StatusCode(StatusCodes.Status500InternalServerError, "Internal Server Error. Please contact support.");

}

else

{

return Forbid("Account already exists.");

}

return Ok();

}

## JWT Token

[HttpGet]

private ActionResult GenerateJWTToken(AppUser user)

{

// Create JWT Token

var claims = new[]

{

new Claim(JwtRegisteredClaimNames.Sub, user.Email),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

new Claim(JwtRegisteredClaimNames.UniqueName, user.UserName)

};

var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(\_configuration["Tokens:Key"]));

var credentials = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

var token = new JwtSecurityToken(

\_configuration["Tokens:Issuer"],

\_configuration["Tokens:Audience"],

claims,

signingCredentials: credentials,

expires: DateTime.UtcNow.AddHours(3)

);

return Created("", new

{

token = new JwtSecurityTokenHandler().WriteToken(token),

user = user.UserName

});

}

# Front-end

## Login

export class LoginComponent implements OnInit {

loginFormGroup: FormGroup = this.fb.group({

emailaddress: ['', [Validators.required, Validators.email]],

password: ['', Validators.required],

})

isLoading:boolean = false

constructor(private router: Router, private apiService: APIService, private fb: FormBuilder, private snackBar: MatSnackBar) { }

ngOnInit(): void {

}

async LoginUser(){

if(this.loginFormGroup.valid)

{

this.isLoading = true

await this.apiService.LoginUser(this.loginFormGroup.value).subscribe(result => {

localStorage.setItem('User', JSON.stringify(result))

this.loginFormGroup.reset();

this.router.navigateByUrl('productListing');

})

}

}

}

OR (simpler example)

Login(){

this.dataService.Login().subscribe((result: any) =>

localStorage.setItem('Token', JSON.stringify(result))

)

}

## Register

export class RegisterComponent implements OnInit {

registerFormGroup: FormGroup = this.fb.group({

emailaddress: ['', [Validators.required, Validators.email]],

password: ['', [Validators.required, Validators.minLength(6), Validators.maxLength(16)]],

})

constructor(private router: Router, private apiService: APIService, private fb: FormBuilder, private snackBar: MatSnackBar) {

}

ngOnInit(): void {

}

RegisterUser(){

if(this.registerFormGroup.valid)

{

this.apiService.RegisterUser(this.registerFormGroup.value).subscribe(() => {

this.registerFormGroup.reset();

this.router.navigate(['']).then((navigated: boolean) => {

if(navigated) {

this.snackBar.open(`Registered successfully`, 'X', {duration: 5000});

}

});

})

}

## Api.service.ts

export class APIService {

apiUrl = 'http://localhost:5240/api/'

httpOptions ={

headers: new HttpHeaders({

ContentType: 'application/json'

})

}

constructor(private httpClient: HttpClient) {

}

RegisterUser(registerUser: RegisterUser){

return this.httpClient.post(`${this.apiUrl}Authentication/Register`, registerUser, this.httpOptions)

}

LoginUser(loginUser: LoginUser){

return this.httpClient.post<User>(`${this.apiUrl}Authentication/Login`, loginUser, this.httpOptions)

}

## Logout

logout(){

if(localStorage.getItem('User'))

{

localStorage.removeItem('User')

this.router.navigateByUrl('login');

}

}

# Example (with roles)

security.component.ts:

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

interface User {

username: string;

password: string;

role: string;

}

@Component({

selector: 'app-security',

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

styleUrls: ['./security.component.css'],

})

export class SecurityComponent implements OnInit {

users: User[] = [

{ username: 'Ziel', password: 'Ziel123', role: 'Admin' },

{ username: 'Phil', password: 'Phil123', role: 'Employee' },

{ username: 'Jacques', password: 'Jacques123', role: 'Employee' },

];

isLoggedIn = false;

currentUser: User | null = null;

ngOnInit() {

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

if (token) {

this.isLoggedIn = true;

const user = this.users.find((u) => u.username === token);

if (user) {

this.currentUser = user;

}

}

}

login(username: string, password: string) {

const user = this.users.find(

(u) => u.username === username && u.password === password

);

if (user) {

this.isLoggedIn = true;

this.currentUser = user;

localStorage.setItem('token', username);

} else {

alert('Invalid username or password');

}

}

logout() {

this.isLoggedIn = false;

this.currentUser = null;

localStorage.removeItem('token');

}

isAdmin(): boolean {

return this.isLoggedIn && this.currentUser?.role === 'Admin';

}

isEmployee(): boolean {

return this.isLoggedIn && this.currentUser?.role === 'Employee';

}

AdminRole() {

if (this.isAdmin()) {

alert('Admin content is visible');

} else {

alert('You do not have permission to view this content');

}

}

EmployeeRole() {

alert('Employee content is visible');

}

}

In the above code, the User interface is defined to represent the user object with properties username, password, and role. The users array is updated to include the role property for each user.

The login() function is modified to store the role of the logged-in user in the local storage as the access token. The isAdmin() and isEmployee() functions are implemented to check if the current user has the role of Admin or Employee, respectively.

The AdminRole() function checks if the current user has the role of Admin and displays a corresponding alert message. The EmployeeRole() function displays an alert message for the employee role.