To build an Angular 17 project with login authentication integrated with a .NET Core API using a model-driven approach, you can follow the steps below. This example will cover the creation of a simple login form in Angular, authentication against a .NET Core API, and how to handle authentication tokens.

**1. Setting Up the .NET Core API for Authentication**

**1.1 Create a New .NET Core Web API Project**

bash

Copy code

dotnet new webapi -n AuthApi

cd AuthApi

**1.2 Add Authentication Logic**

1. **Create a User Model:**

Inside the Models folder, create a User.cs file:

csharp

Copy code

namespace AuthApi.Models

{

public class User

{

public string Username { get; set; }

public string Password { get; set; }

}

}

1. **Create a Token Service:**

Inside the Services folder, create a TokenService.cs file to handle JWT generation:

csharp

Copy code

using System;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.IdentityModel.Tokens;

namespace AuthApi.Services

{

public class TokenService

{

private const string SecretKey = "your\_secret\_key\_here";

public string GenerateToken(string username)

{

var tokenHandler = new JwtSecurityTokenHandler();

var key = Encoding.ASCII.GetBytes(SecretKey);

var tokenDescriptor = new SecurityTokenDescriptor

{

Subject = new ClaimsIdentity(new Claim[]

{

new Claim(ClaimTypes.Name, username)

}),

Expires = DateTime.UtcNow.AddHours(1),

SigningCredentials = new SigningCredentials(new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256Signature)

};

var token = tokenHandler.CreateToken(tokenDescriptor);

return tokenHandler.WriteToken(token);

}

}

}

1. **Create an Authentication Controller:**

Inside the Controllers folder, create an AuthController.cs file:

csharp

Copy code

using AuthApi.Models;

using AuthApi.Services;

using Microsoft.AspNetCore.Mvc;

namespace AuthApi.Controllers

{

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

[ApiController]

public class AuthController : ControllerBase

{

private readonly TokenService \_tokenService;

public AuthController()

{

\_tokenService = new TokenService();

}

[HttpPost("login")]

public IActionResult Login([FromBody] User user)

{

// Dummy authentication: replace with your actual authentication logic

if (user.Username == "testuser" && user.Password == "password")

{

var token = \_tokenService.GenerateToken(user.Username);

return Ok(new { Token = token });

}

return Unauthorized("Invalid username or password");

}

}

}

1. **Run the API:**

bash

Copy code

dotnet run

The API will be available at https://localhost:5001/api/auth/login.

**2. Setting Up the Angular 17 Project**

**2.1 Create a New Angular Project**

bash

Copy code

ng new AngularAuth

cd AngularAuth

**2.2 Generate a Login Component**

bash

Copy code

ng generate component components/login

**2.3 Generate a User Model**

bash

Copy code

ng generate class models/user --type=model

**2.4 Create an Authentication Service**

Generate a service to handle authentication:

bash

Copy code

ng generate service services/auth

**2.5 Implement the User Model**

Update user.model.ts with the following:

typescript

Copy code

export class User {

username: string = '';

password: string = '';

}

**2.6 Implement the Authentication Service**

Update auth.service.ts:

typescript

Copy code

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

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

import { Observable } from 'rxjs';

import { User } from '../models/user.model';

@Injectable({

providedIn: 'root'

})

export class AuthService {

private apiUrl = 'https://localhost:5001/api/auth/login';

constructor(private http: HttpClient) {}

login(user: User): Observable<any> {

return this.http.post(this.apiUrl, user);

}

saveToken(token: string): void {

localStorage.setItem('auth\_token', token);

}

getToken(): string | null {

return localStorage.getItem('auth\_token');

}

isLoggedIn(): boolean {

return this.getToken() !== null;

}

logout(): void {

localStorage.removeItem('auth\_token');

}

}

**2.7 Implement the Login Component**

Update login.component.ts:

typescript

Copy code

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

import { FormBuilder, FormGroup, Validators } from '@angular/forms';

import { AuthService } from '../../services/auth.service';

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

@Component({

selector: 'app-login',

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

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

})

export class LoginComponent {

loginForm: FormGroup;

errorMessage: string = '';

constructor(

private fb: FormBuilder,

private authService: AuthService,

private router: Router

) {

this.loginForm = this.fb.group({

username: ['', Validators.required],

password: ['', Validators.required]

});

}

onSubmit(): void {

if (this.loginForm.valid) {

this.authService.login(this.loginForm.value).subscribe({

next: (response) => {

this.authService.saveToken(response.token);

this.router.navigate(['/dashboard']); // Redirect to a dashboard or another component

},

error: () => {

this.errorMessage = 'Invalid username or password';

}

});

}

}

}

**2.8 Create the Login Component Template**

Update login.component.html:

html

Copy code

<form [formGroup]="loginForm" (ngSubmit)="onSubmit()">

<div>

<label for="username">Username:</label>

<input id="username" formControlName="username" type="text" />

</div>

<div>

<label for="password">Password:</label>

<input id="password" formControlName="password" type="password" />

</div>

<button type="submit">Login</button>

<p \*ngIf="errorMessage">{{ errorMessage }}</p>

</form>

**3. Integrating the Login Component into the App**

**3.1 Update app-routing.module.ts**

Set up routing to include the login component:

typescript

Copy code

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

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

import { LoginComponent } from './components/login/login.component';

const routes: Routes = [

{ path: 'login', component: LoginComponent },

{ path: '', redirectTo: '/login', pathMatch: 'full' }

];

@NgModule({

imports: [RouterModule.forRoot(routes)],

exports: [RouterModule]

})

export class AppRoutingModule {}

**3.2 Update app.module.ts**

Ensure that HttpClientModule and ReactiveFormsModule are imported:

typescript

Copy code

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

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

@NgModule({

declarations: [/\* your components here \*/],

imports: [

HttpClientModule,

ReactiveFormsModule,

// other imports

],

bootstrap: [/\* your main component \*/]

})

export class AppModule {}

**4. Running the Application**

1. **Run the Angular Project:**

bash

Copy code

ng serve

1. **Access the Application:**

Navigate to http://localhost:4200/login to see your Angular login form.

**5. Securing Angular Routes**

To secure routes in Angular, use route guards that check if the user is authenticated before allowing access.

**5.1 Create an Auth Guard**

Generate an auth guard:

bash

Copy code

ng generate guard guards/auth

Update auth.guard.ts:

typescript

Copy code

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

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

import { AuthService } from '../services/auth.service';

@Injectable({

providedIn: 'root'

})

export class AuthGuard implements CanActivate {

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

canActivate(): boolean {

if (this.authService.isLoggedIn()) {

return true;

} else {

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

return false;

}

}

}

**5.2 Apply the Auth Guard**

Update your routing module to protect routes:

typescript

Copy code

import { AuthGuard } from './guards/auth.guard';

const routes: Routes = [

{ path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] },

{ path: 'login', component: LoginComponent },

{ path: '', redirectTo: '/login', pathMatch: 'full' }

];

**Conclusion**

This setup provides a basic foundation for integrating Angular 17 with a .NET Core API for login authentication using a model-driven approach. You can expand upon this by adding features such as token refresh, user roles, and more.