**DN 4.0 Dotnet FSE**

**Name: Aditya Mohanty**

**Superset ID:6361043**

**Week 5**

**Microservices**

**[Kindly Note:Question 6 of Web API was done in last week’s hands-on]**

Task-Implement JWT Authentication in ASP.NET Core Web API

Package imported from nugget package manager:**Microsoft.AspNetCore.Authentication.JwtBearer**

Code for Program.cs:

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

namespace Week5

{

public class Program

{

public static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

In = Microsoft.OpenApi.Models.ParameterLocation.Header,

Description = "Please enter JWT with Bearer prefix (e.g., Bearer {token})",

Name = "Authorization",

Type = Microsoft.OpenApi.Models.SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new Microsoft.OpenApi.Models.OpenApiSecurityRequirement

{

{

new Microsoft.OpenApi.Models.OpenApiSecurityScheme

{

Reference = new Microsoft.OpenApi.Models.OpenApiReference

{

Type = Microsoft.OpenApi.Models.ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme).AddJwtBearer(options =>

{

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = builder.Configuration["Jwt:Issuer"],

ValidAudience = builder.Configuration["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(builder.Configuration["Jwt:Key"]))

};

});

builder.Services.AddAuthorization(options =>

{

options.AddPolicy("AdminOnly", policy => policy.RequireClaim("role", "admin"));

options.AddPolicy("UserOnly", policy => policy.RequireClaim("role", "user"));

});

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

}

Code for appsettings.json:

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Jwt": {

"Key": "ThisIsASecretKeyForJwtToken32Char",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

Code for models/loginmodel.cs:

namespace Week5.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

Code for controllers/authController.cs:

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Week5.Models;

[ApiController]

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

public class AuthController : ControllerBase

{

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized();

}

private bool IsValidUser(LoginModel model)

{

// Simple user validation (use real validation in production!)

return model.Username == "testuser" && model.Password == "password";

}

private string GenerateJwtToken(string username)

{

var claims = new[] { new Claim(ClaimTypes.Name, username) };

var key = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes("ThisIsASecretKeyForJwtToken32Char"));

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

var token = new JwtSecurityToken(

issuer: "MyAuthServer",

audience: "MyApiUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

public class SecretController : ControllerBase

{

[Authorize]

[HttpGet]

public IActionResult GetSecret()

{

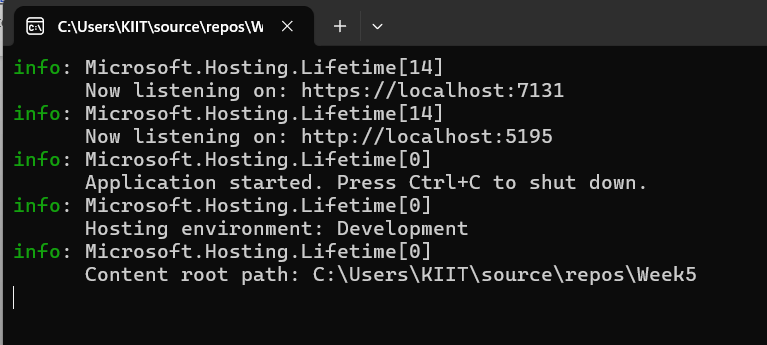
return Ok("This is a protected endpoint.");

}

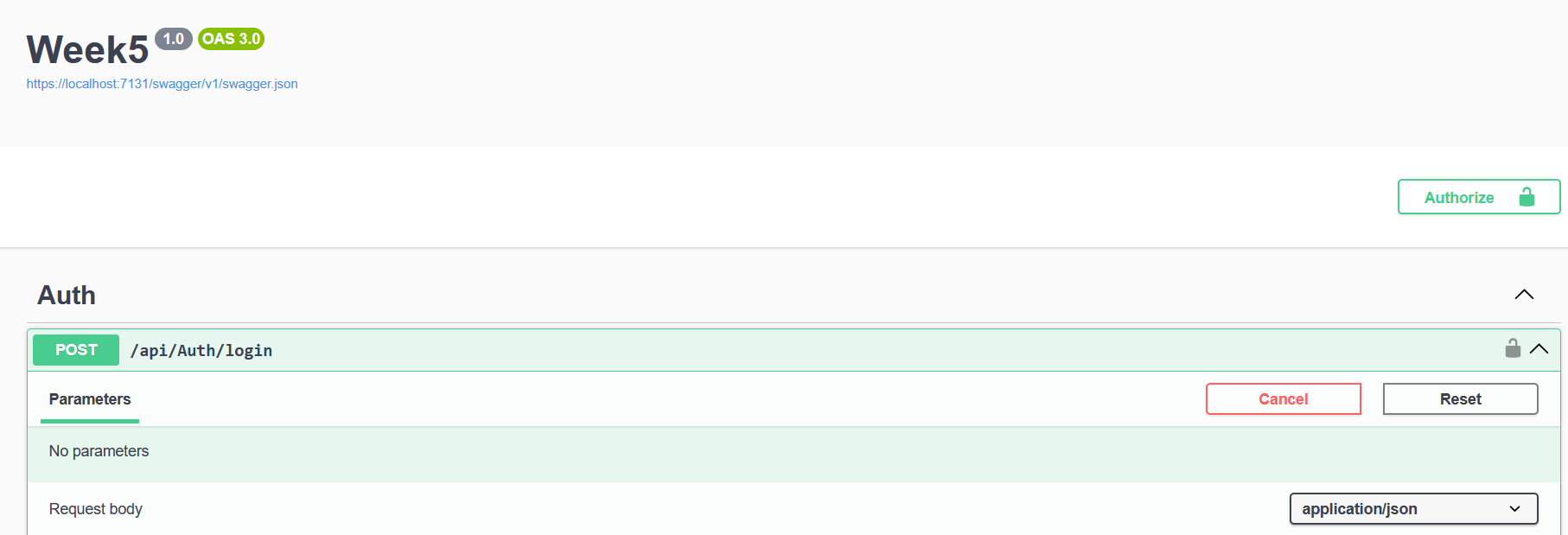
}

**OUPUT**

Terminal after build(next page)

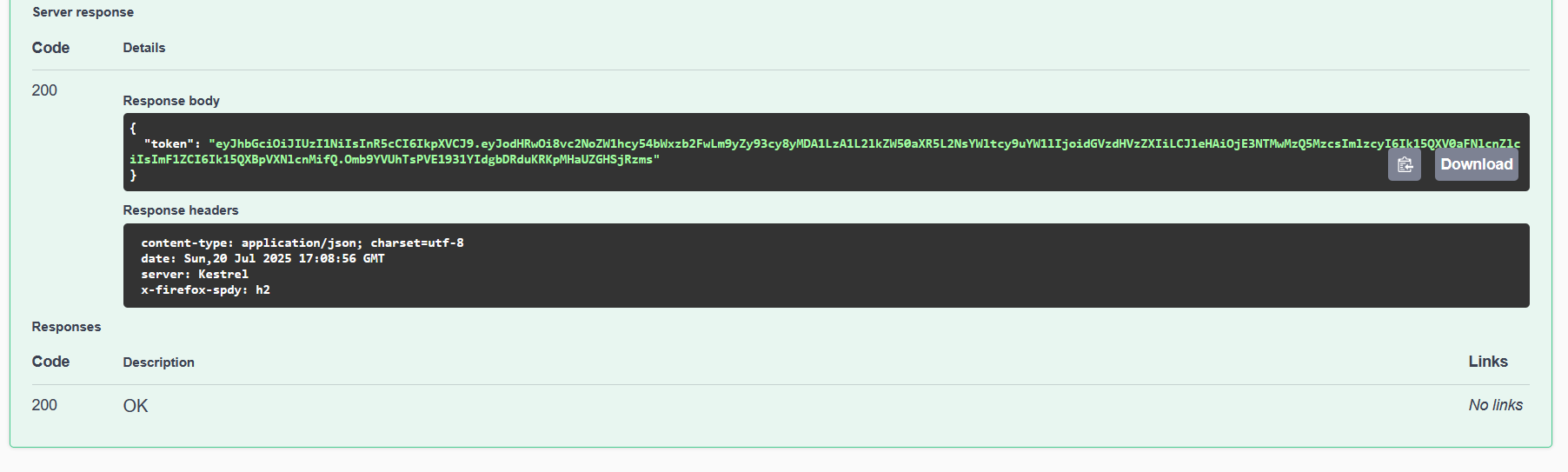


Authorize button-



POST REQUEST





Bearer after authorization(next page)

