**JsonWebToken**

There are various modes of authenticating a request. Json Web Token(JWT) authentication is one among them. It is a methodology of passing a token in the Authorization header value in the request so that it can be checked at the WebAPI and validated. If not there, then ‘**Unauthorized**’ status message with status code 401 should be thrown.

Use the below code in Startup.cs

* In ConfigureServices method

string securityKey = "mysuperdupersecret";

var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

//what to validate

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

//setup validate data

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

* In Configure method

app.UseAuthentication();

This is to enable the JWT authentication in .Net core

Create a new controller ‘AuthController’ in the Web API application. Add **AllowAnonymous** attribute to the controller. Create a private method GenerateJSONWebToken as shown thru the code below.

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

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

var claims = new List<Claim>

{

new Claim(ClaimTypes.Role, userRole),

new Claim("UserId", userId.ToString())

};

var token = new JwtSecurityToken(

issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(10),

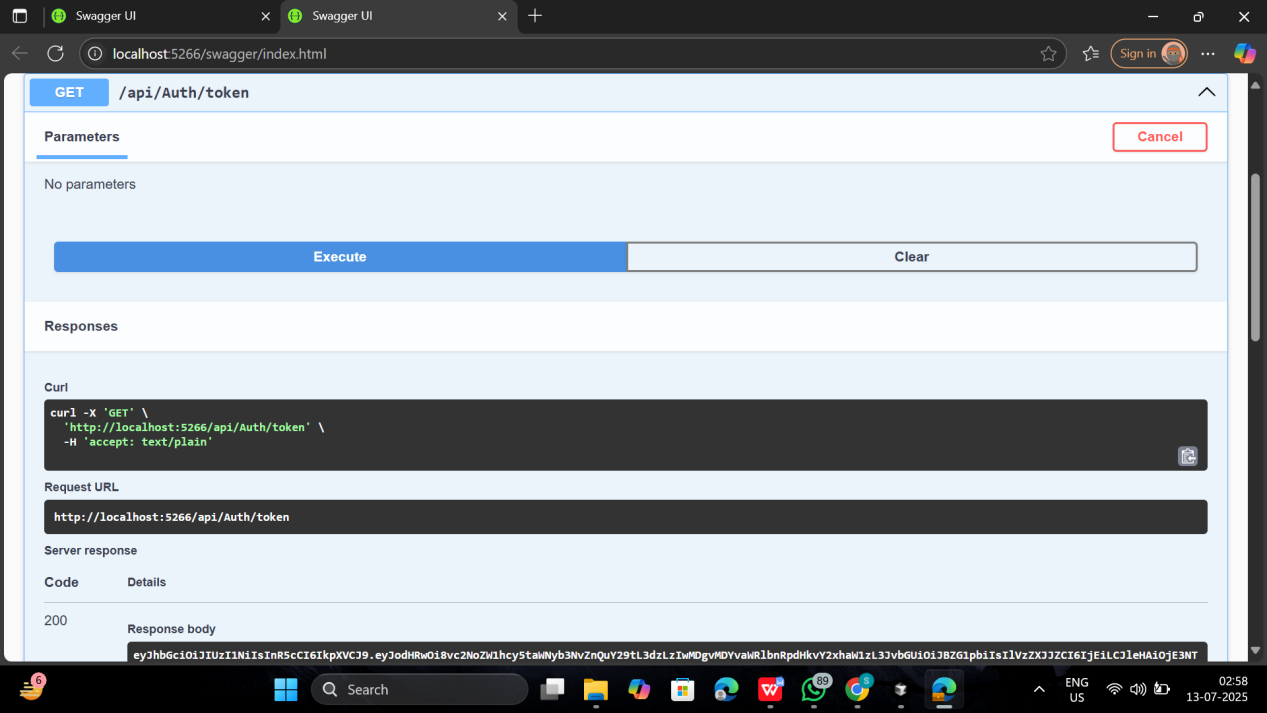
signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

Note that the issuer, audience and the securitykey defined in the Startup.cs code and method code shown above is the same and should match.

Invoke the GenerateJSONWebToken in the GET action method by sending some value for user id and ‘Admin’ for the user role. This is to set Claims information to check the user role



**Use the JWT generated thru the AuthController to be used in POSTMAN request.**

Remove the ‘CustomAuthFilter’ that is currently mapped to the Employee controller(if already done)

Use **Authorize** attribute in the Employee controller to check if the request header contains ‘Authorization’ key with ‘Bearer’ token or not. If the token is unavailable, then ‘Unauthorized’ status message whose code 401 will be thrown. Use POSTMAN to hit a GET action method call. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

Use the AuthController to generate the JWT. Use that in the GET action method call thru POSTMAN if the request is authenticated or not.

Modify the token value in the POSTMAN tool and check if ‘Unauthorized’ status message is thrown. Note the ‘Status’ attribute in the ‘Headers’ section in the output window.

using Microsoft.AspNetCore.Mvc;

using EmployeeWebApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace EmployeeWebApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> \_employees = new List<Employee>

        {

            new Employee

            {

                Id = 1,

                Name = "John Doe",

                Salary = 50000,

                Permanent = true,

                Department = new Department { Id = 1, Name = "IT", Location = "NY" },

                Skills = new List<Skill>

                {

                    new Skill { Id = 1, Name = "C#", Description = "Programming" }

                },

                DateOfBirth = new DateTime(1990, 1, 1)

            },

            new Employee

            {

                Id = 2,

                Name = "Jane Smith",

                Salary = 60000,

                Permanent = false,

                Department = new Department { Id = 2, Name = "HR", Location = "LA" },

                Skills = new List<Skill>

                {

                    new Skill { Id = 2, Name = "Recruitment", Description = "HR Skill" }

                },

                DateOfBirth = new DateTime(1992, 2, 2)

            }

        };

        [HttpPut("{id}")]

        public ActionResult<Employee> Put(int id, [FromBody] Employee employee)

        {

            if (id <= 0)

            {

                return BadRequest("Invalid employee id");

            }

            var existing = \_employees.FirstOrDefault(e => e.Id == id);

            if (existing == null)

            {

                return BadRequest("Invalid employee id");

            }

            existing.Name = employee.Name;

            existing.Salary = employee.Salary;

            existing.Permanent = employee.Permanent;

            existing.Department = employee.Department;

            existing.Skills = employee.Skills;

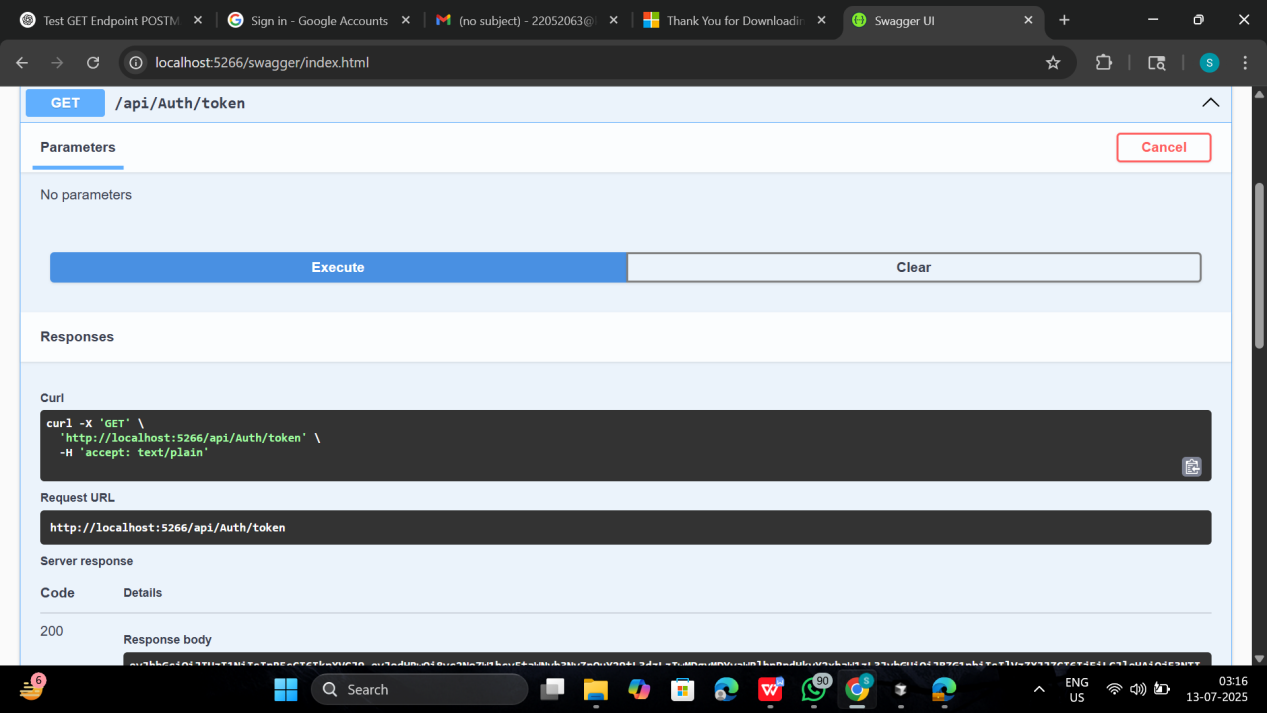
            existing.DateOfBirth = employee.DateOfBirth;

            return Ok(existing);

        }

    }

}

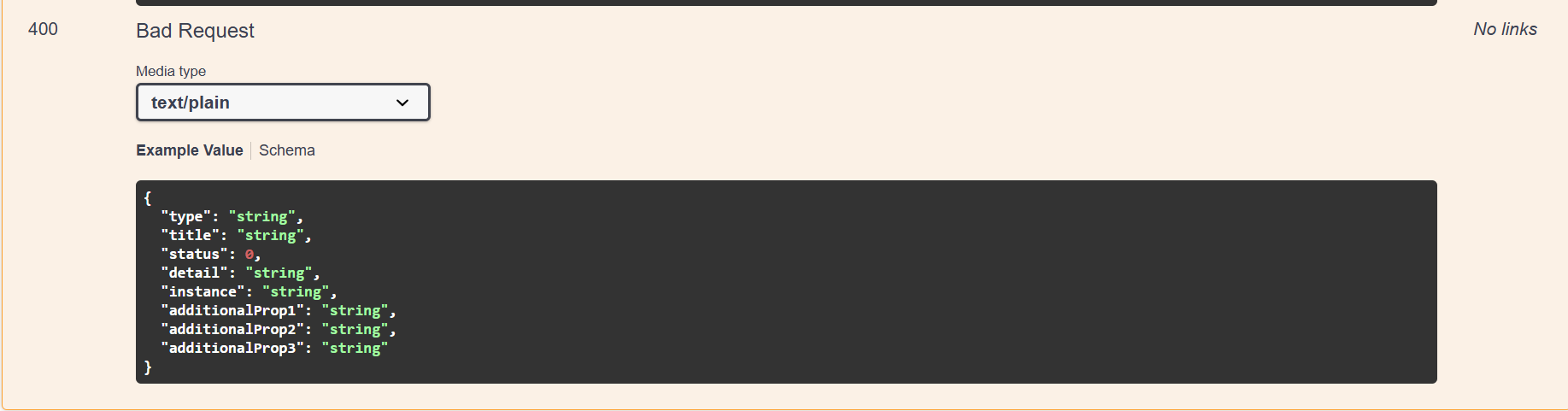


**Check for JWT expiration**

A JWT token has an attribute which can be set to determine how long the token is valid to be used.

In the GenerateJSONWebToken method in AuthController, the ‘expires’ attribute of the JwtSecurityToken object denotes the time in minutes for which the token would be valid.

Modify the duration for ‘expires’ attribute to 2 minutes. Check the POSTMAN request for GET call **AFTER** 2 minutes of generation of the JWT, which should yield ‘Unauthorized’ message with Http status code 401.



**Add the roles to be authorized in the Authorize attribute.**

The **Authorize** attribute supports the roles to be used to filter the controller action method access.

The GenerateJSONWebToken method of AuthController uses the role ‘Admin’ set in the claims.

Include the role ‘**POC**’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is ‘Unauthorized’ with status code 401

Include the role ‘**Admin**’ along with ‘POC’ in the Authorize attribute in the Employee controller. Hit the GET action method of the Employee controller thru POSTMAN. Verify if the response status is OK with status code 200

using Microsoft.AspNetCore.Mvc;

using EmployeeWebApi.Models;

using System;

using System.Collections.Generic;

using System.Linq;

namespace EmployeeWebApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> \_employees = new List<Employee>

        {

            new Employee

            {

                Id = 1,

                Name = "John Doe",

                Salary = 50000,

                Permanent = true,

                Department = new Department { Id = 1, Name = "IT", Location = "NY" },

                Skills = new List<Skill>

                {

                    new Skill { Id = 1, Name = "C#", Description = "Programming" }

                },

                DateOfBirth = new DateTime(1990, 1, 1)

            },

            new Employee

            {

                Id = 2,

                Name = "Jane Smith",

                Salary = 60000,

                Permanent = false,

                Department = new Department { Id = 2, Name = "HR", Location = "LA" },

                Skills = new List<Skill>

                {

                    new Skill { Id = 2, Name = "Recruitment", Description = "HR Skill" }

                },

                DateOfBirth = new DateTime(1992, 2, 2)

            }

        };

        [HttpPut("{id}")]

        public ActionResult<Employee> Put(int id, [FromBody] Employee employee)

        {

            if (id <= 0)

            {

                return BadRequest("Invalid employee id");

            }

            var existing = \_employees.FirstOrDefault(e => e.Id == id);

            if (existing == null)

            {

                return BadRequest("Invalid employee id");

            }

            existing.Name = employee.Name;

            existing.Salary = employee.Salary;

            existing.Permanent = employee.Permanent;

            existing.Department = employee.Department;

            existing.Skills = employee.Skills;

            existing.DateOfBirth = employee.DateOfBirth;

            return Ok(existing);

        }

    }

}