**Ashutosh Dash  
Superset ID – 6365404  
WebAPI - 5**

**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;

namespace CustomWebApi.Controllers

{

[ApiController]

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

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet]

public IActionResult GetToken()

{

try

{

var token = GenerateJSONWebToken(101, "Admin");

return Ok(token);

}

catch (Exception ex)

{

return StatusCode(500, $"Token generation failed: {ex.Message}");

}

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperduperultrasecretkey!@123456789"));

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(1),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**EmployeeCpntroller.cs**

using CustomWebApi.Filters;

using CustomWebApi.Models;

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace CustomWebApi.Controllers

{

[Authorize(Roles = "Admin,POC")]

[ApiController]

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

public class EmployeeController : ControllerBase

{

private readonly List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

[HttpGet("GetStandard")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> GetStandard()

{

try

{

return Ok(\_employees);

}

catch (Exception ex)

{

return StatusCode(500, $"Error occurred: {ex.Message}");

}

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> CreateEmployee([FromBody] Employee newEmp)

{

if (newEmp == null || newEmp.Id <= 0)

{

return BadRequest("Invalid employee data");

}

\_employees.Add(newEmp);

return CreatedAtAction(nameof(GetStandard), new { id = newEmp.Id }, newEmp);

}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> UpdateEmployee(int id, [FromBody] Employee updatedEmployee)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

return Ok(existingEmployee);

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public IActionResult DeleteEmployee(int id)

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (emp == null)

{

return BadRequest("Employee not found");

}

\_employees.Remove(emp);

return Ok($"Employee with ID {id} deleted");

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Ashutosh Dash",

Salary = 50000,

Permanent = true,

DateOfBirth = new DateTime(1999, 8, 15),

Department = new Department { Id = 101, Name = "IT" },

Skills = new List<Skill>

{

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

new Skill { Id = 2, Name = ".NET Core" }

}

},

new Employee

{

Id = 2,

Name = "Simran Sharma",

Salary = 60000,

Permanent = false,

DateOfBirth = new DateTime(1998, 11, 20),

Department = new Department { Id = 102, Name = "HR" },

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "Excel" },

new Skill { Id = 4, Name = "Recruitment" }

}

}

};

}

}

}

**Program.cs**

using CustomWebApi.Filters;

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers(options =>

{

options.Filters.Add<CustomExceptionFilter>();

});

builder.Services.AddScoped<CustomAuthFilter>();

builder.Services.AddScoped<CustomExceptionFilter>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo { Title = "CustomWebApi", Version = "v1" });

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

In = ParameterLocation.Header,

Description = "Please enter a valid token with Bearer",

Name = "Authorization",

Type = SecuritySchemeType.ApiKey,

Scheme = "Bearer"

});

c.AddSecurityRequirement(new OpenApiSecurityRequirement {

{

new OpenApiSecurityScheme {

Reference = new OpenApiReference {

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

new string[] {}

}

});

});

var securityKey = "mysuperduperultrasecretkey!@123456789";

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

builder.Services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey,

ClockSkew = TimeSpan.Zero

};

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

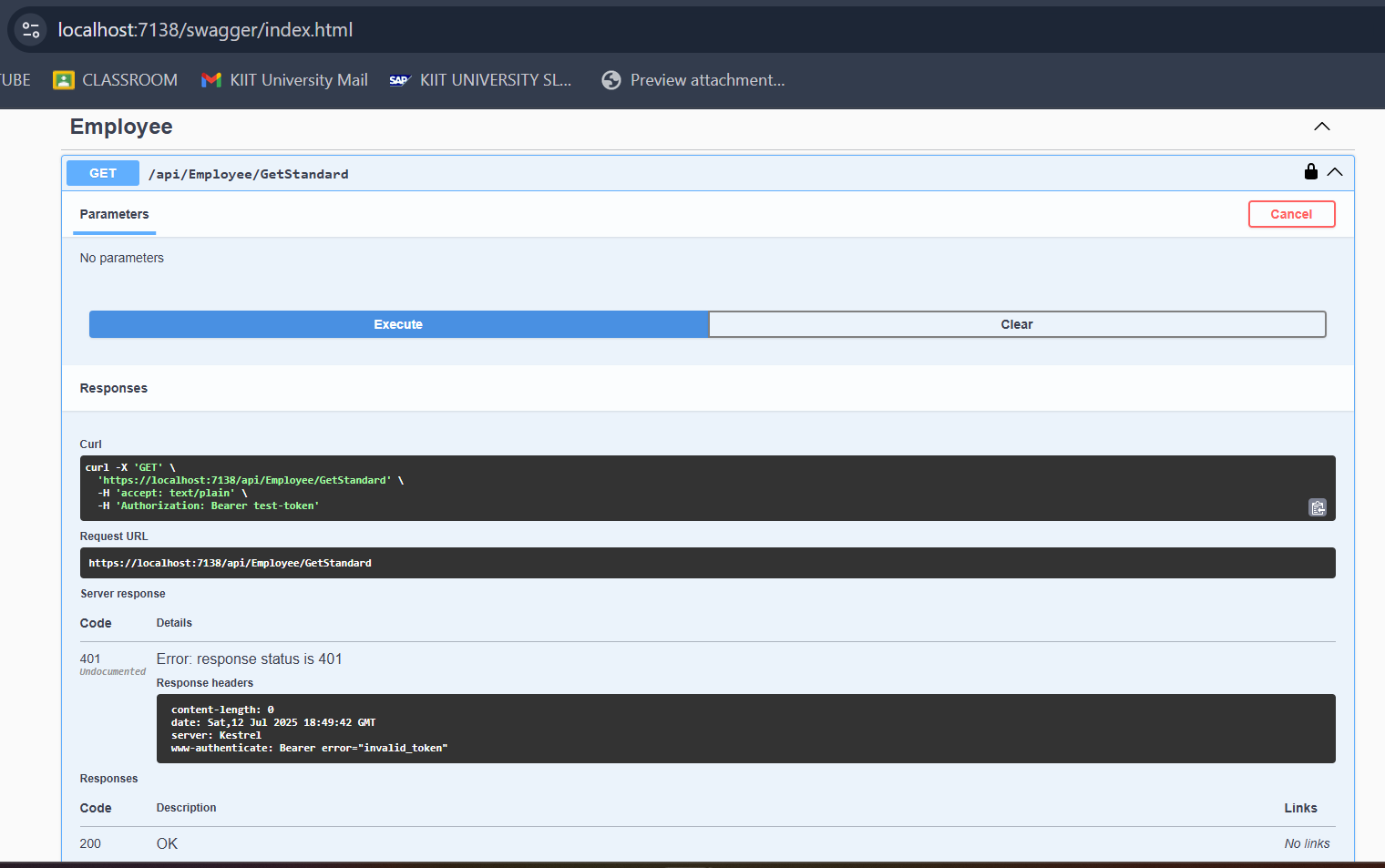
app.UseAuthentication();

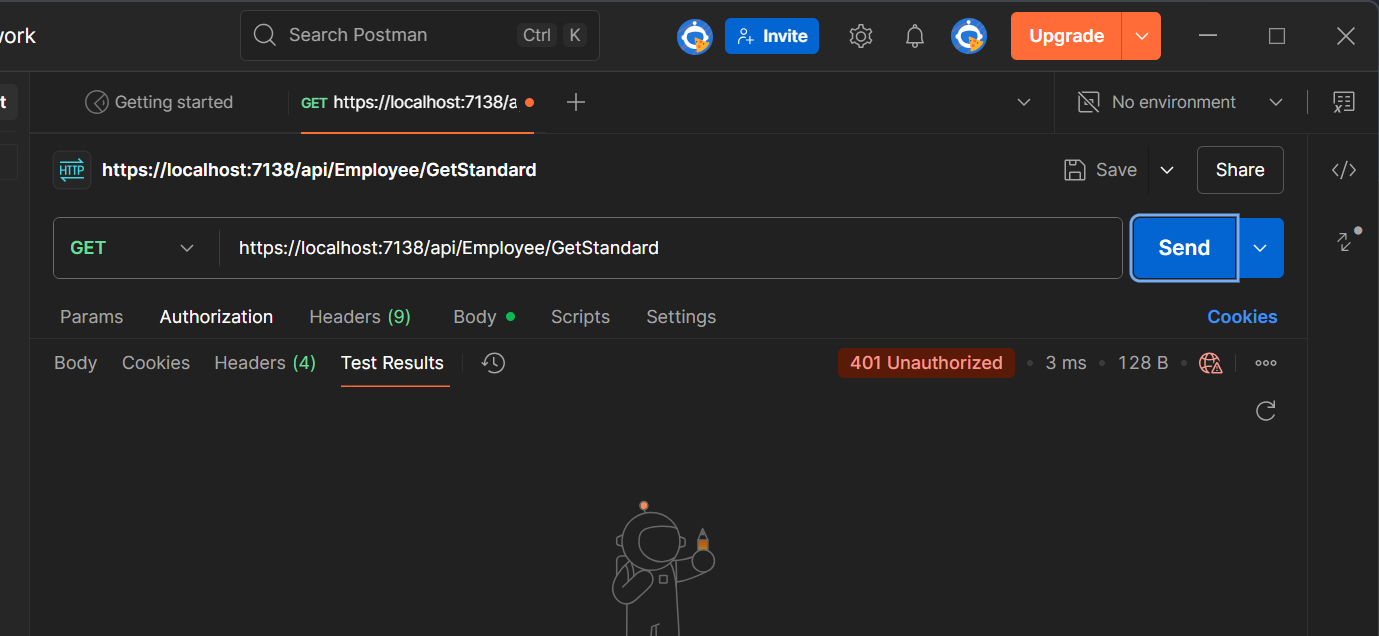
app.UseAuthorization();

app.MapControllers();

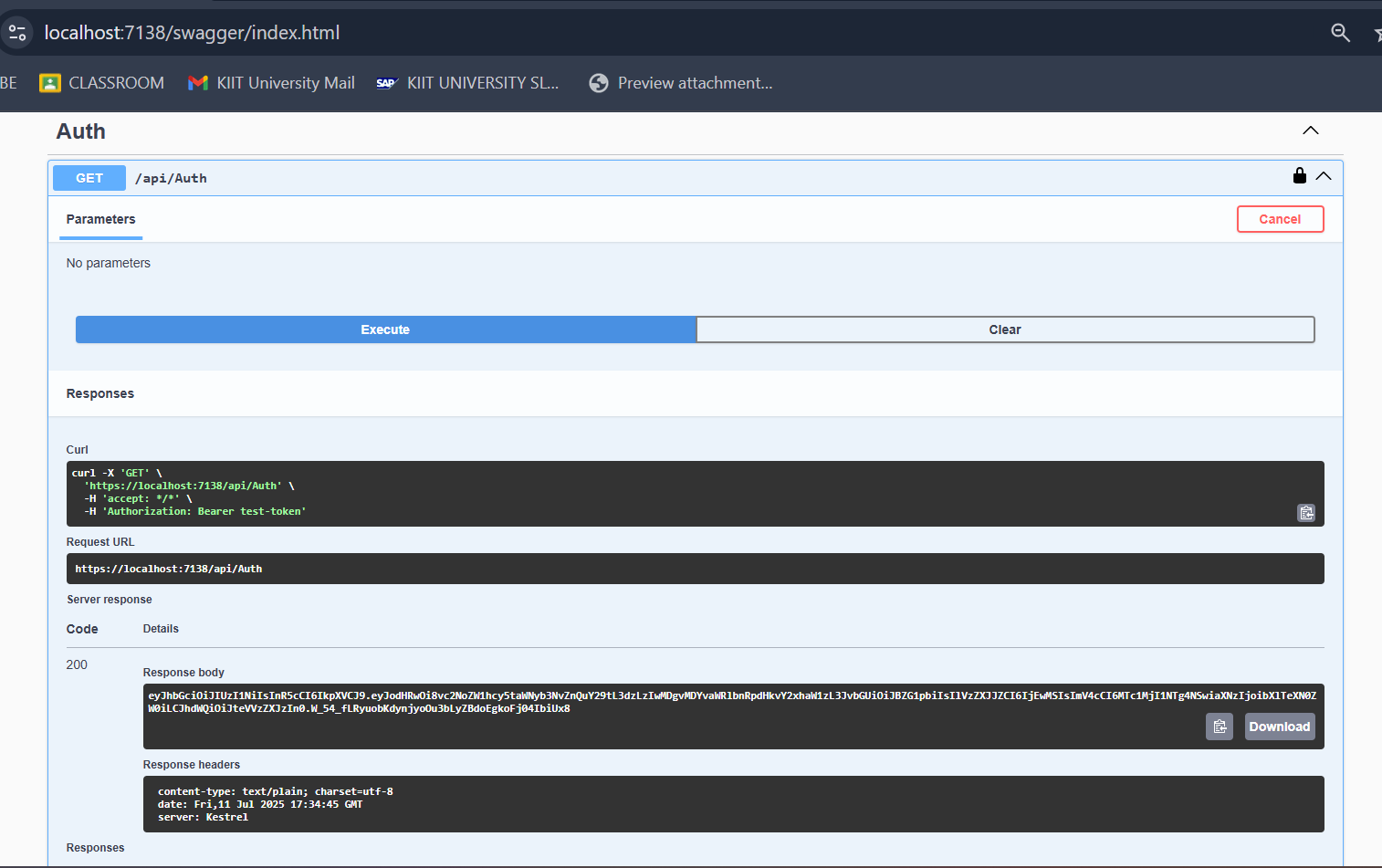
app.Run();

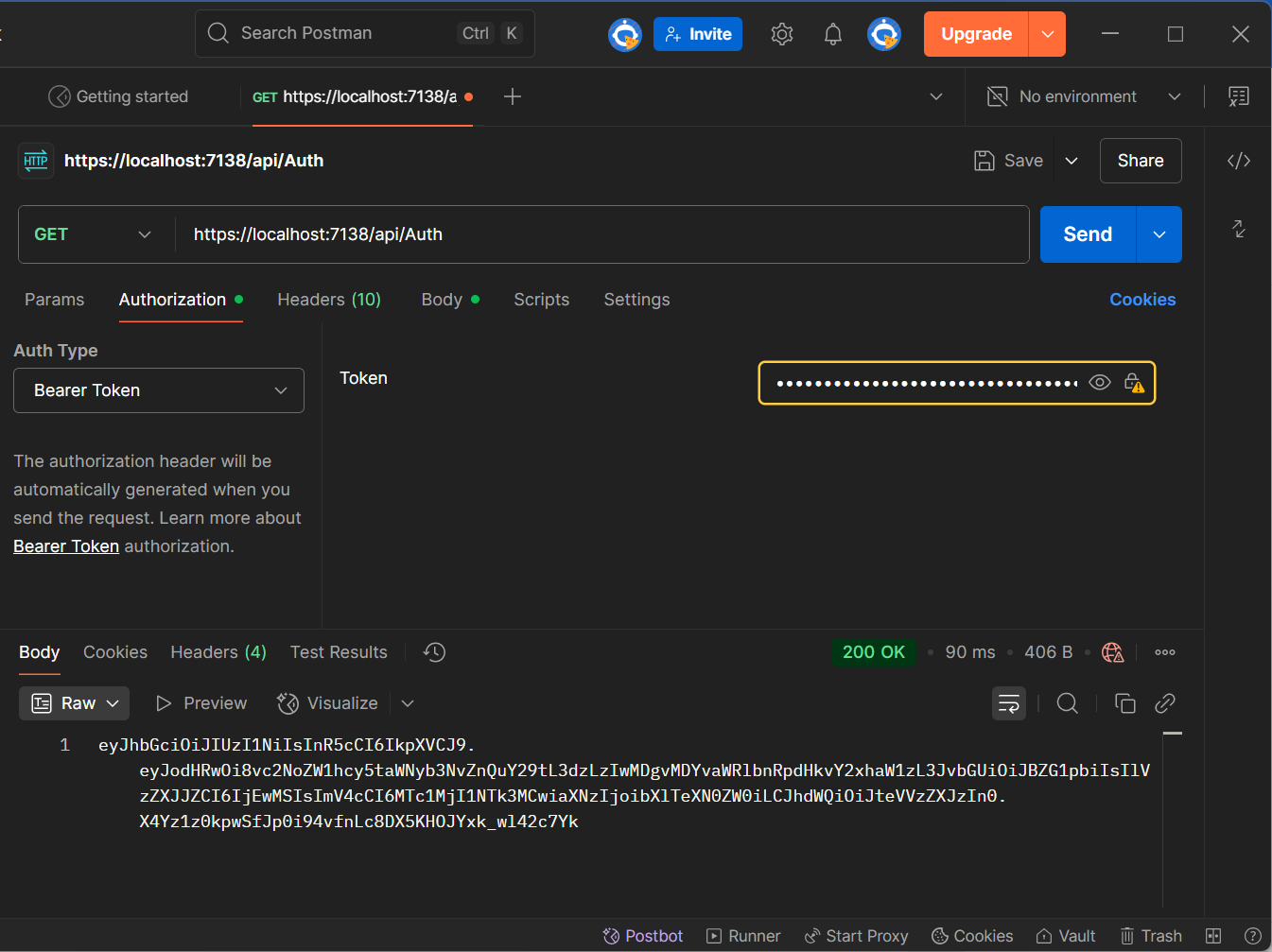
**Without authorization**

****

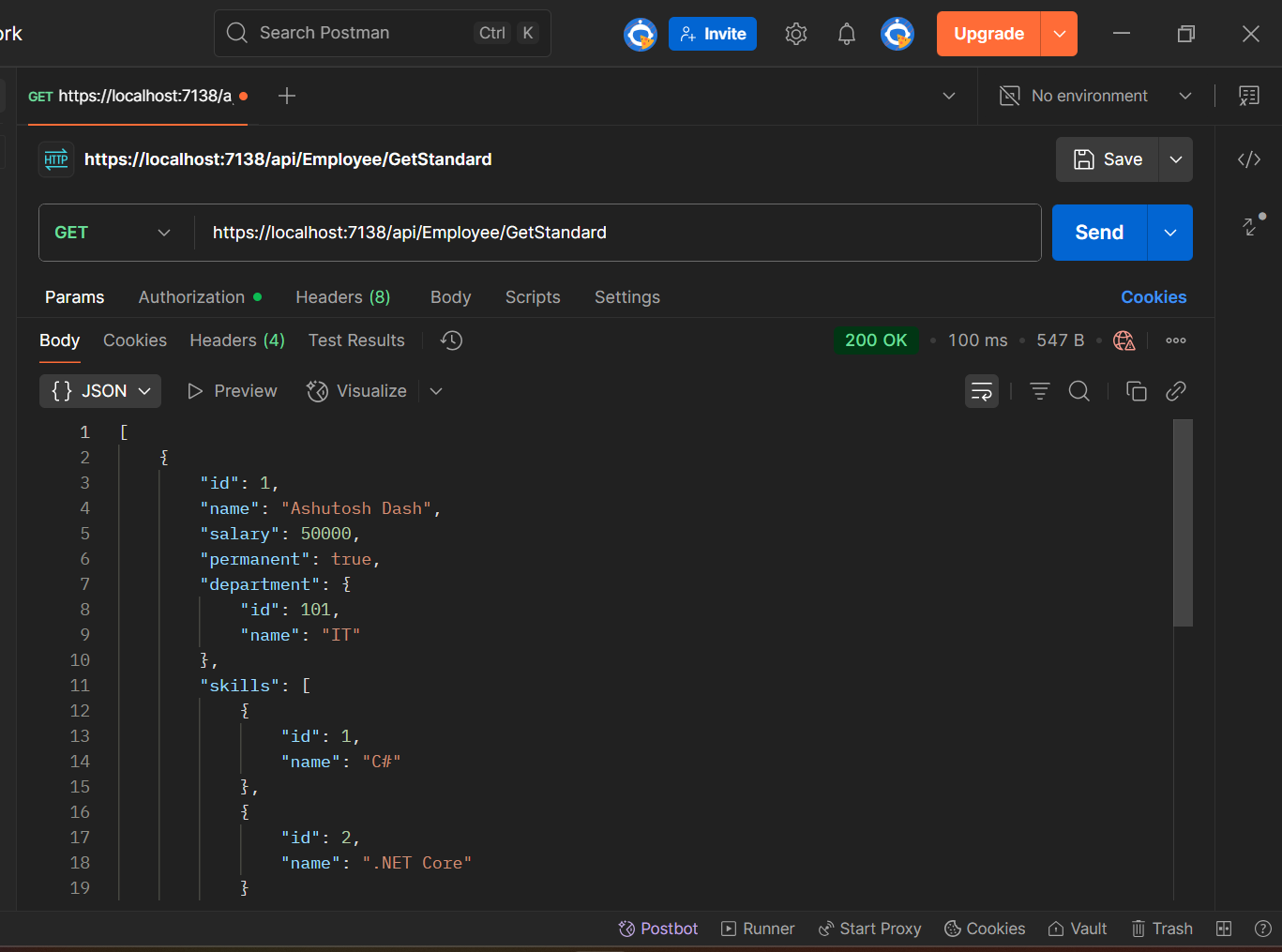


**After Authorization**





**GetStandard**



**After 1 minutes (expiration of token)**

var token = new JwtSecurityToken(

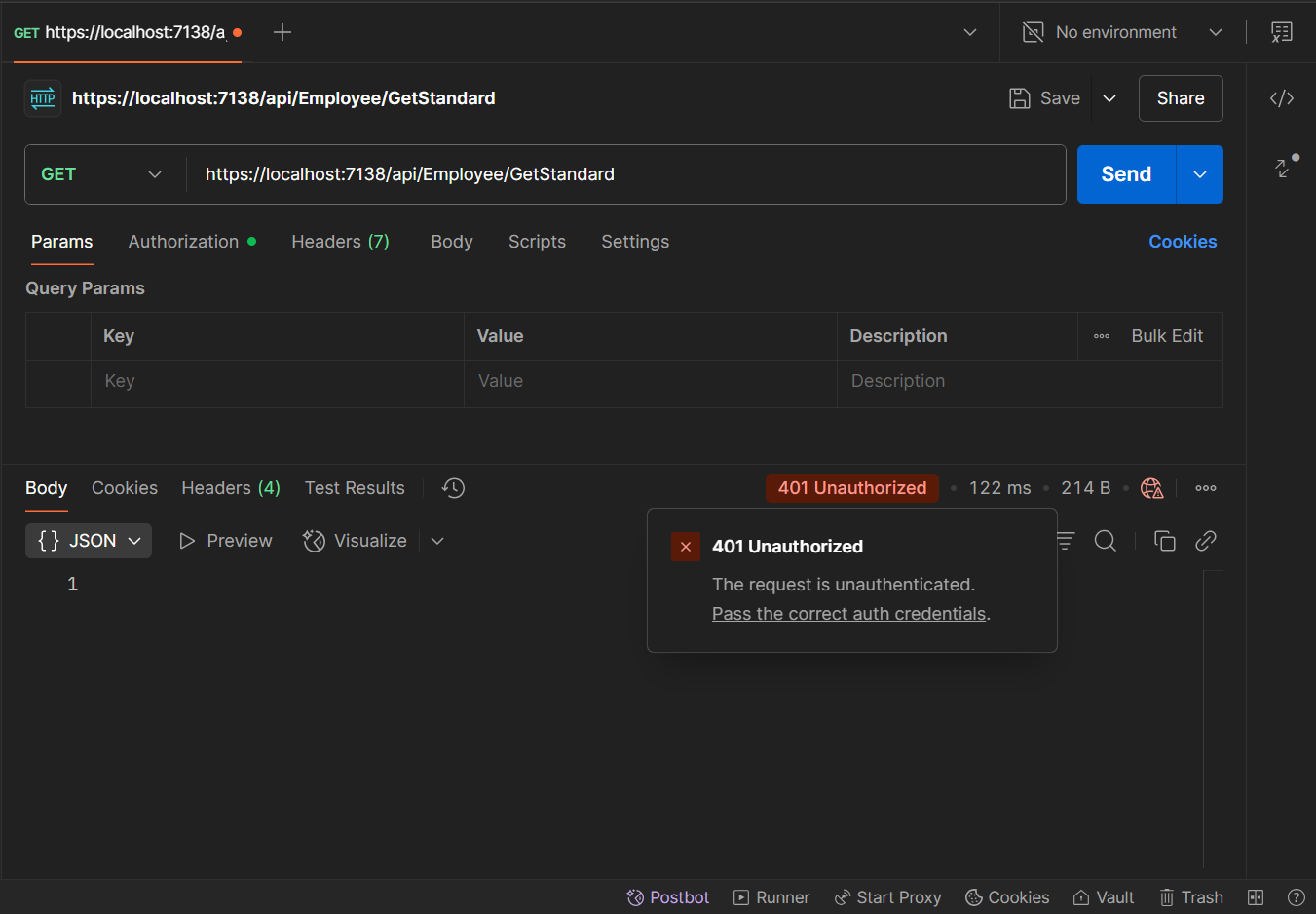
issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(1),

signingCredentials: credentials);



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

[Authorize(Roles = "POC")]

[ApiController]

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

