**Microservices Architecture using ASP.NET Core Web API**

**Suman Sourabh**

**6363866**

**Question 1: Implement JWT Authentication in ASP.NET Core Web API**

Scenario:

You are building a microservice that requires secure login. You need to implement JWT

based authentication.

Steps:

1. Create a new ASP.NET Core Web API project.

2. Add a `User` model and a login endpoint.

3. Generate a JWT token upon successful login.

4. Secure an endpoint using `[Authorize]`.

**Solutions:**

**Appsettings.json**

{

"Jwt": {

//"Key": "ThisIsASecretKeyForJwtToken",

"Key": "ThisIsAStrongSecretKeyWithMoreThan32Chars!",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

**User.cs**

namespace JwtAuthDemo.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**SecureDataController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class SecureDataController : ControllerBase

{

[HttpGet]

[Authorize]

public IActionResult GetSecureData()

{

var username = User.Identity?.Name;

return Ok(new { Message = $"Hello {username}, this is protected data." });

}

}

}

**AuthController.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using JwtAuthDemo.Models;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly IConfiguration \_config;

public AuthController(IConfiguration config)

{

\_config = config;

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

if (IsValidUser(model))

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized();

}

// Simple user check (demo only)

private bool IsValidUser(LoginModel model)

{

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

}

// Generate JWT token using config key

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

var key = Encoding.UTF8.GetBytes(\_config["Jwt:Key"]);

//if (key.Length < 32)

//{

// throw new Exception("JWT Key must be at least 32 characters for HS256.");

//}

var securityKey = new SymmetricSecurityKey(key);

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

var token = new JwtSecurityToken(

issuer: \_config["Jwt:Issuer"],

audience: \_config["Jwt:Audience"],

claims: claims,

expires: DateTime.Now.AddMinutes(Convert.ToDouble(\_config["Jwt:DurationInMinutes"])),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**Program.cs**

**using Microsoft.AspNetCore.Authentication.JwtBearer;**

**using Microsoft.IdentityModel.Tokens;**

**using System.Text;**

**internal class Program**

**{**

**private static void Main(string[] args)**

**{**

**var builder = WebApplication.CreateBuilder(args);**

**// Add services to the container**

**builder.Services.AddControllers();**

**builder.Services.AddEndpointsApiExplorer(); // Swagger**

**//builder.Services.AddSwaggerGen(); // Swagger**

**builder.Services.AddSwaggerGen(options =>**

**{**

**options.AddSecurityDefinition("Bearer", new Microsoft.OpenApi.Models.OpenApiSecurityScheme**

**{**

**Name = "Authorization",**

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

**Scheme = "Bearer",**

**BearerFormat = "JWT",**

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

**Description = "Enter 'Bearer' [space] and then your JWT token.\n\nExample: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."**

**});**

**options.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();**

**var app = builder.Build();**

**// Enable Swagger only in Development**

**if (app.Environment.IsDevelopment())**

**{**

**app.UseSwagger();**

**app.UseSwaggerUI();**

**}**

**app.UseRouting();**

**app.UseAuthentication();**

**app.UseAuthorization();**

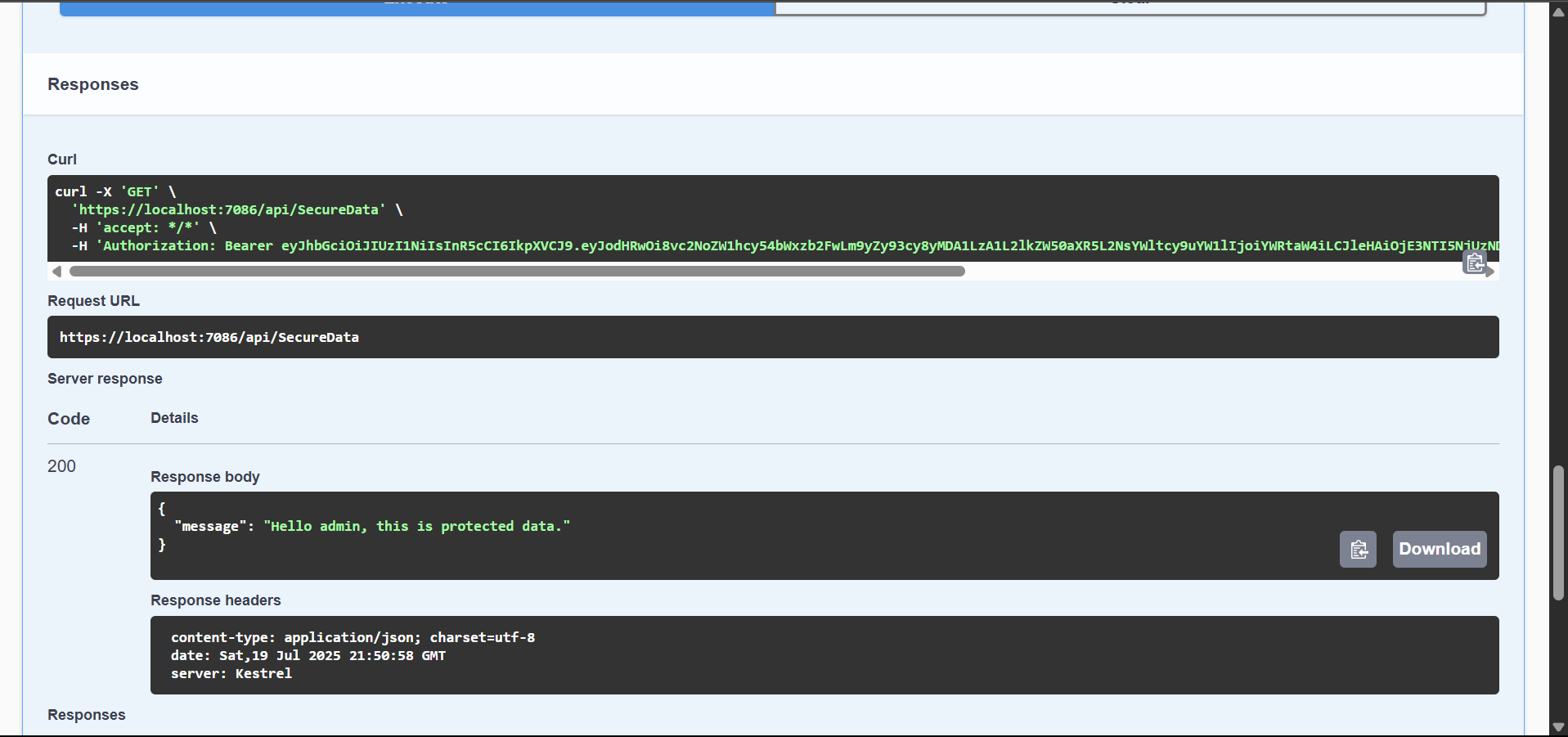
**app.MapControllers();**

**app.Run();**

**}**

**}**





**Question 2: Secure an API Endpoint Using JWT**

Scenario:

You want to restrict access to a sensitive endpoint using JWT authentication.

Steps:

1. Add `[Authorize]` to a controller.

2. Test access with and without a valid token.

**Solution:**

**SecureDataController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

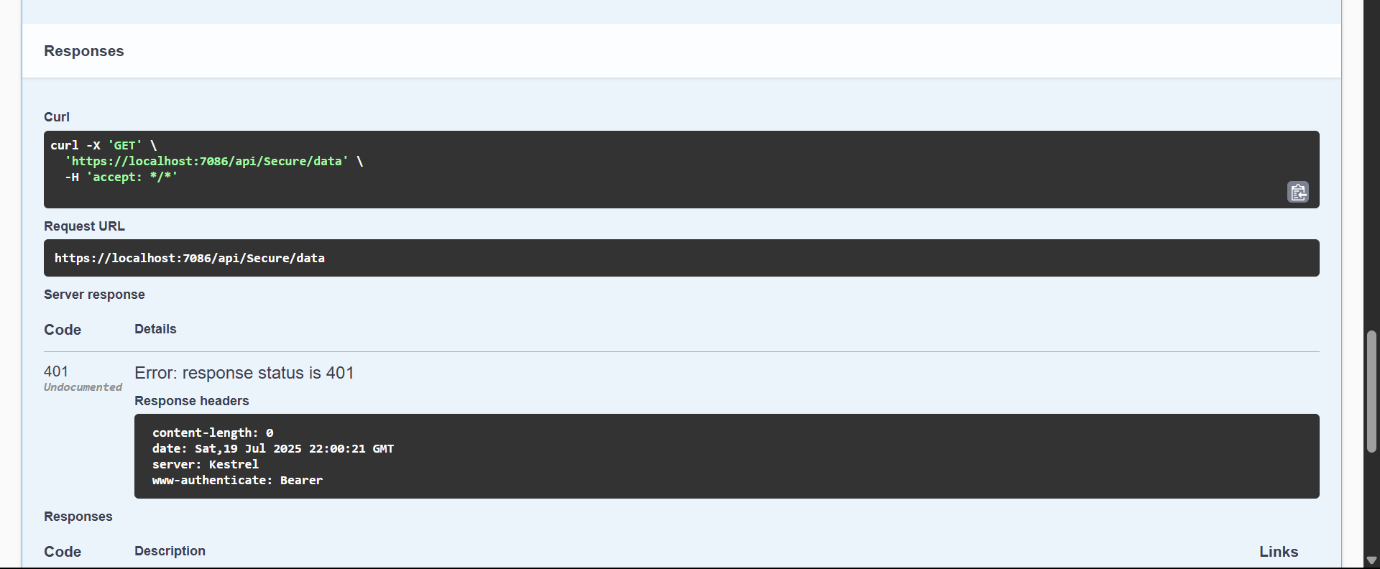
return Ok("This is protected data.");

}

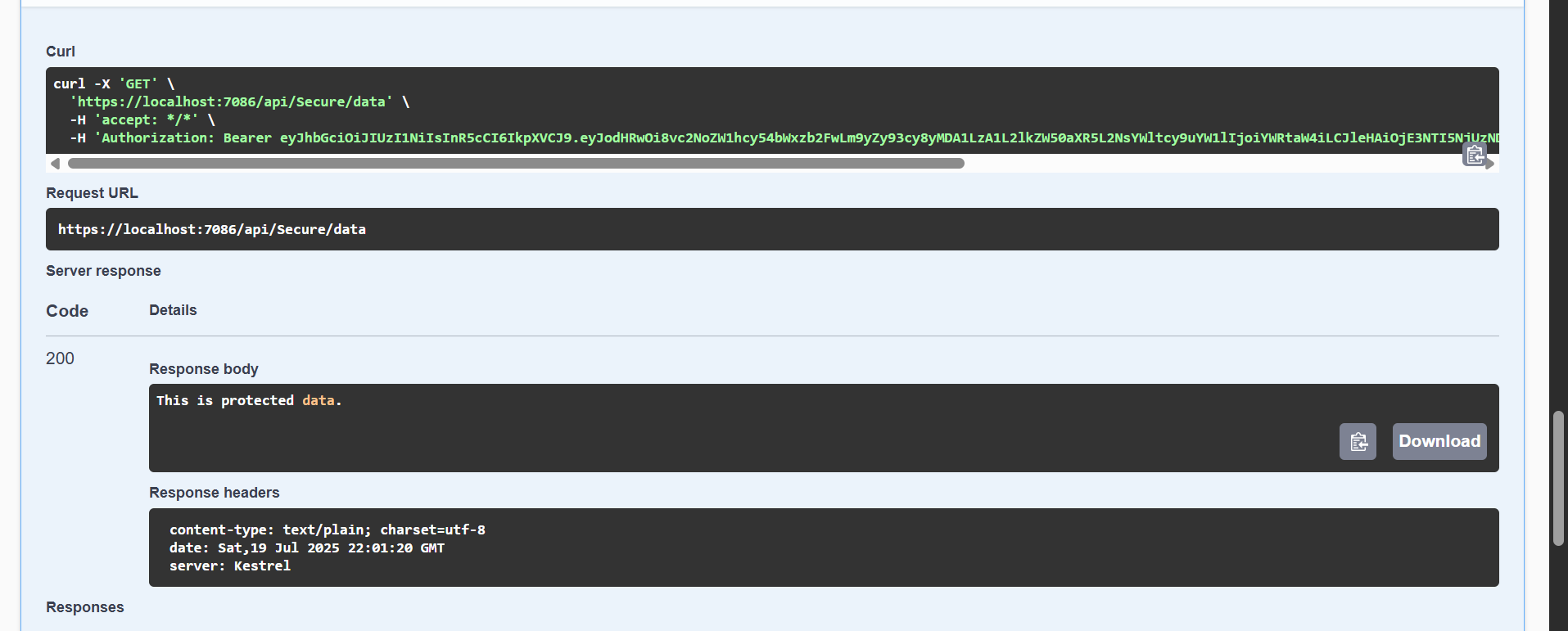
}

}

Without token



With token



**Question 3: Add Role-Based Authorization**

Scenario:

You want to allow only users with the "Admin" role to access certain endpoints.

Steps:

1. Add roles to JWT claims.

2. Use `[Authorize(Roles = "Admin")]`.

**Solution:**

**AdminController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class AdminController : ControllerBase

{

[HttpGet("dashboard")]

[Authorize(Roles = "Admin")]

public IActionResult GetAdminDashboard()

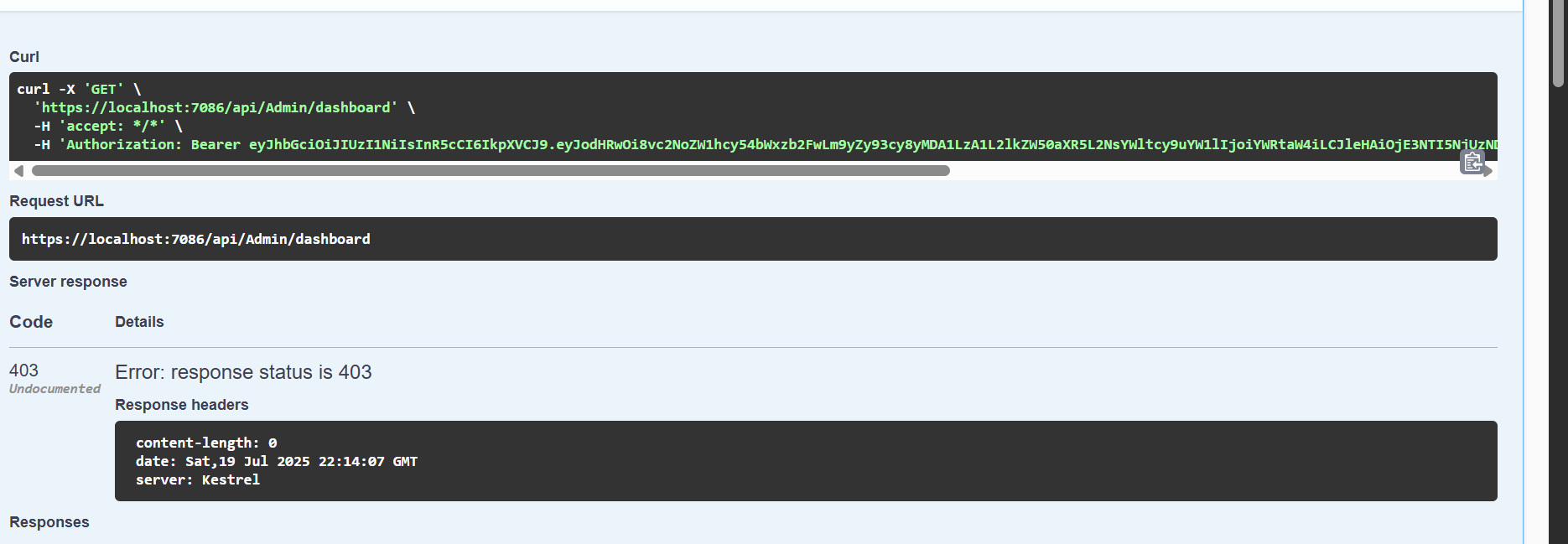
{

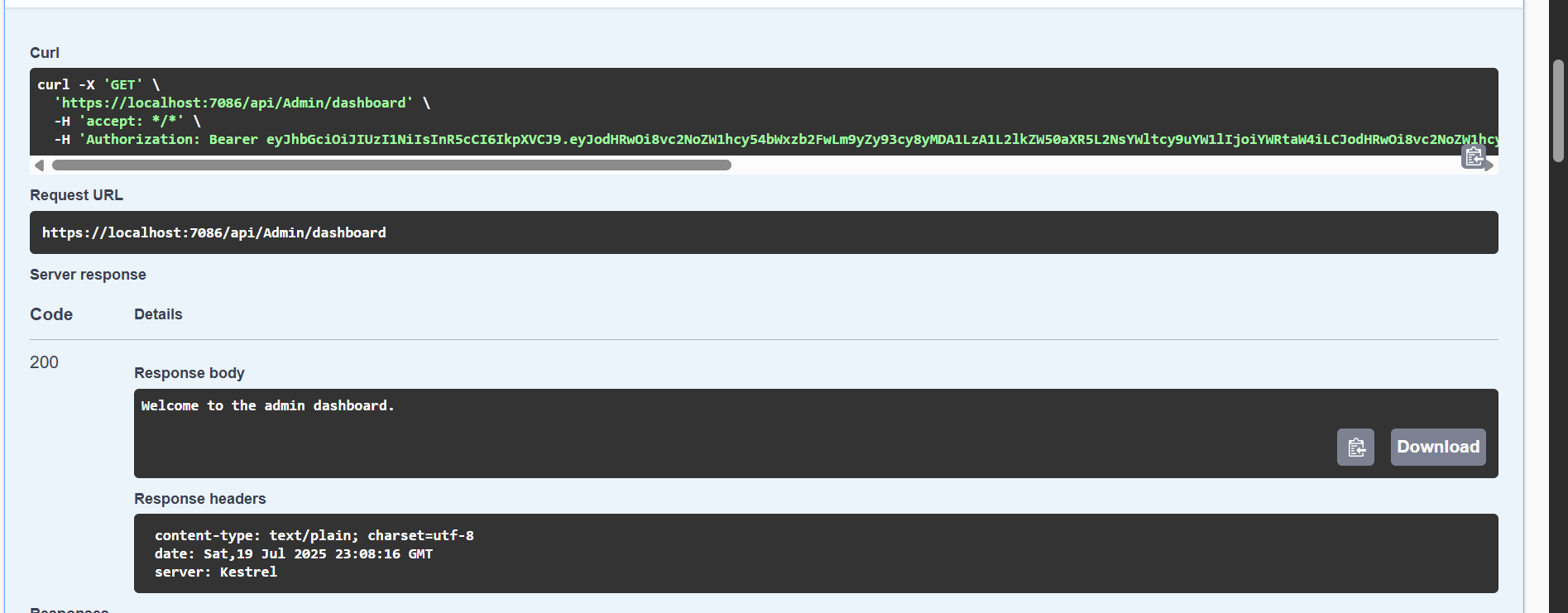
return Ok("Welcome to the admin dashboard.");

}

}

}





**Question 4: Validate JWT Token Expiry and Handle Unauthorized Access**

Scenario:

You want to handle expired or invalid tokens gracefully.

Steps:

1. Configure JWT bearer events.

2. Return custom messages for unauthorized access.

**Solutions:**

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

internal class Program

{

private static void Main(string[] args)

{

var builder = WebApplication.CreateBuilder(args);

// Add services to the container

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer(); // Swagger

//builder.Services.AddSwaggerGen(); // Swagger

builder.Services.AddSwaggerGen(options =>

{

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

{

Name = "Authorization",

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

Scheme = "Bearer",

BearerFormat = "JWT",

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

Description = "Enter 'Bearer' [space] and then your JWT token.\n\nExample: Bearer eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."

});

options.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.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"]))

};

// Handle token errors gracefully

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception.GetType() == typeof(SecurityTokenExpiredException))

{

// Add custom header for token expiration

context.Response.Headers.Add("Token-Expired", "true");

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization();

var app = builder.Build();

// Enable Swagger only in Development

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

}

}

