Aman Raj

Superset ID: 6358186

**Authentication and Authorization in ASP.NET Core Web API Microservices**

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

"Jwt": {

"Key": "ThisIsA32CharUltraSecretKeyJwt1234",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

},

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"AllowedHosts": "\*"

}

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

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

// JWT setup

builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)

.AddJwtBearer(options =>

{

var config = builder.Configuration;

options.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = config["Jwt:Issuer"],

ValidAudience = config["Jwt:Audience"],

IssuerSigningKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(config["Jwt:Key"]))

};

// Optional: Token expired response

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

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

{

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

}

return Task.CompletedTask;

}

};

});

builder.Services.AddAuthorization();

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthentication(); // Add this before UseAuthorization

app.UseAuthorization();

app.MapControllers();

app.Run();

**AuthController.cs**using JwtAuthDemo.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

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)

{

// Hardcoded credentials for testing

if (model.Username == "admin" && model.Password == "password")

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized(new

{

message = "Invalid username or password."

});

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

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

var creds = new SigningCredentials(key, 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

);

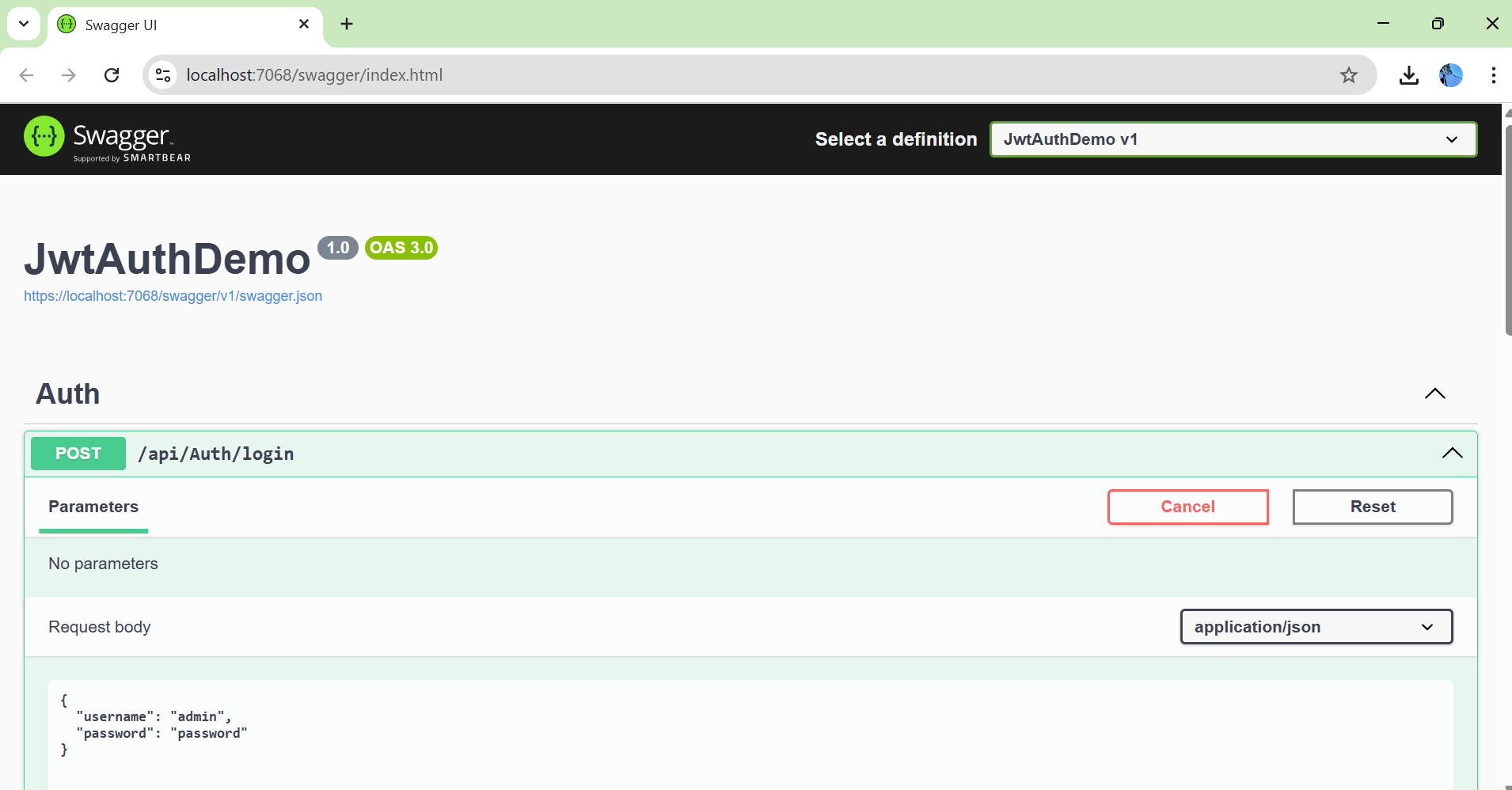
return new JwtSecurityTokenHandler().WriteToken(token);

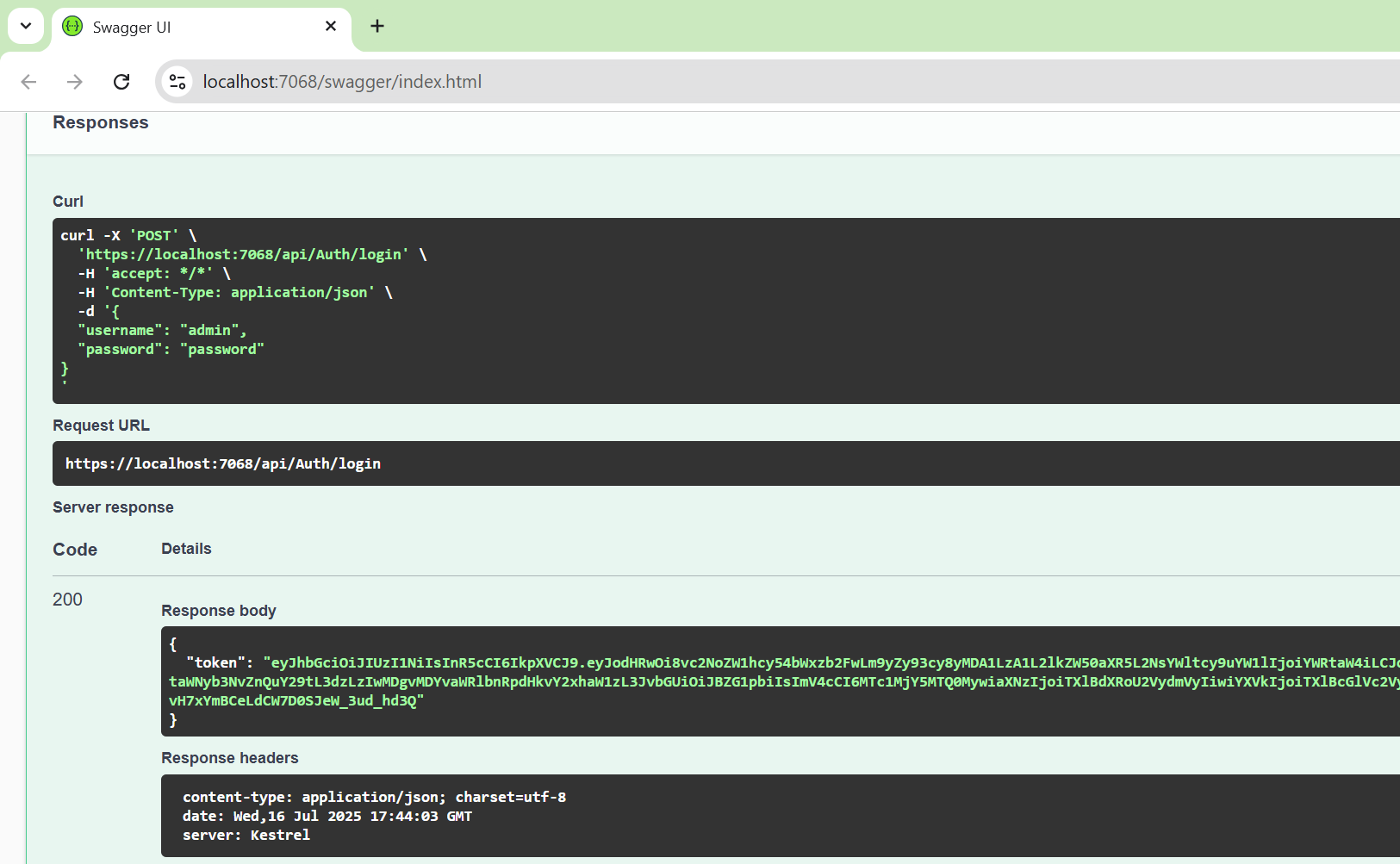
}

}

}

**Output:-**





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

**SecureController.cs**using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtAuthDemo.Controllers

{

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize] // 🔐 Requires a valid token

public IActionResult GetSecureData()

{

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

}

}

}

**AuthController.cs**

using JwtAuthDemo.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

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)

{

// Hardcoded credentials for testing

if (model.Username == "admin" && model.Password == "password")

{

var token = GenerateJwtToken(model.Username);

return Ok(new { Token = token });

}

return Unauthorized(new

{

message = "Invalid username or password."

});

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

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

var creds = new SigningCredentials(key, 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

);

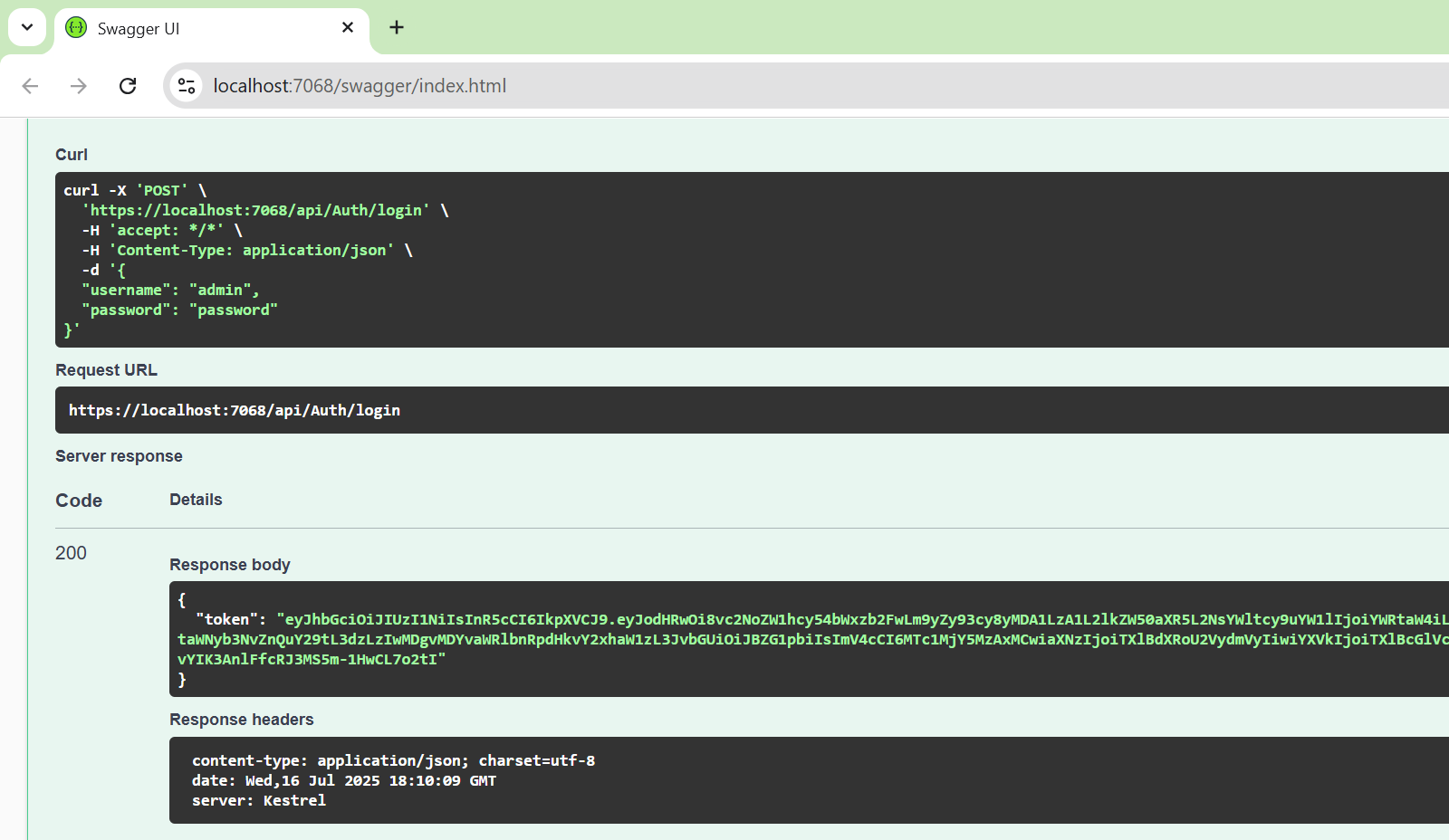
return new JwtSecurityTokenHandler().WriteToken(token);

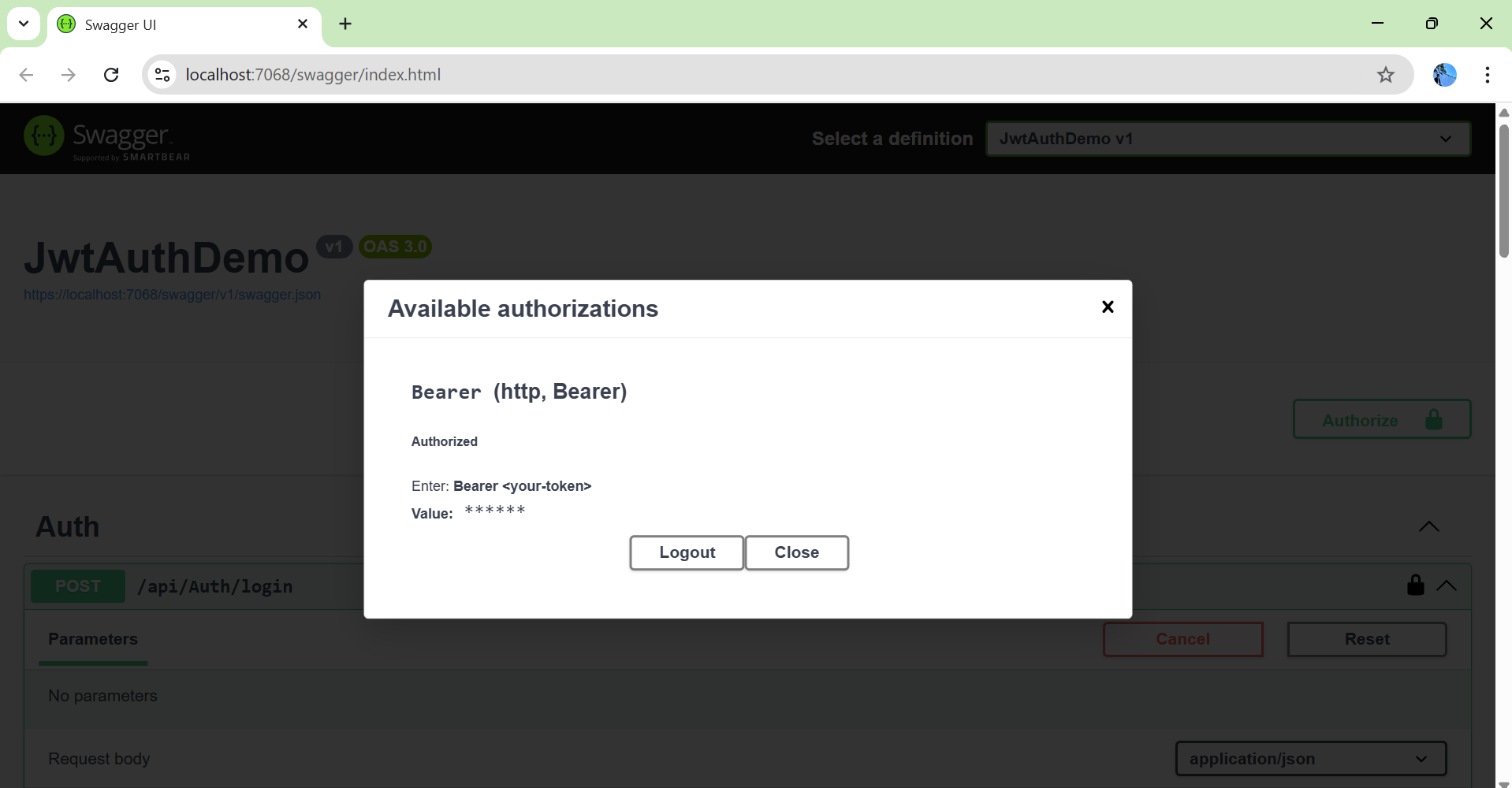
}

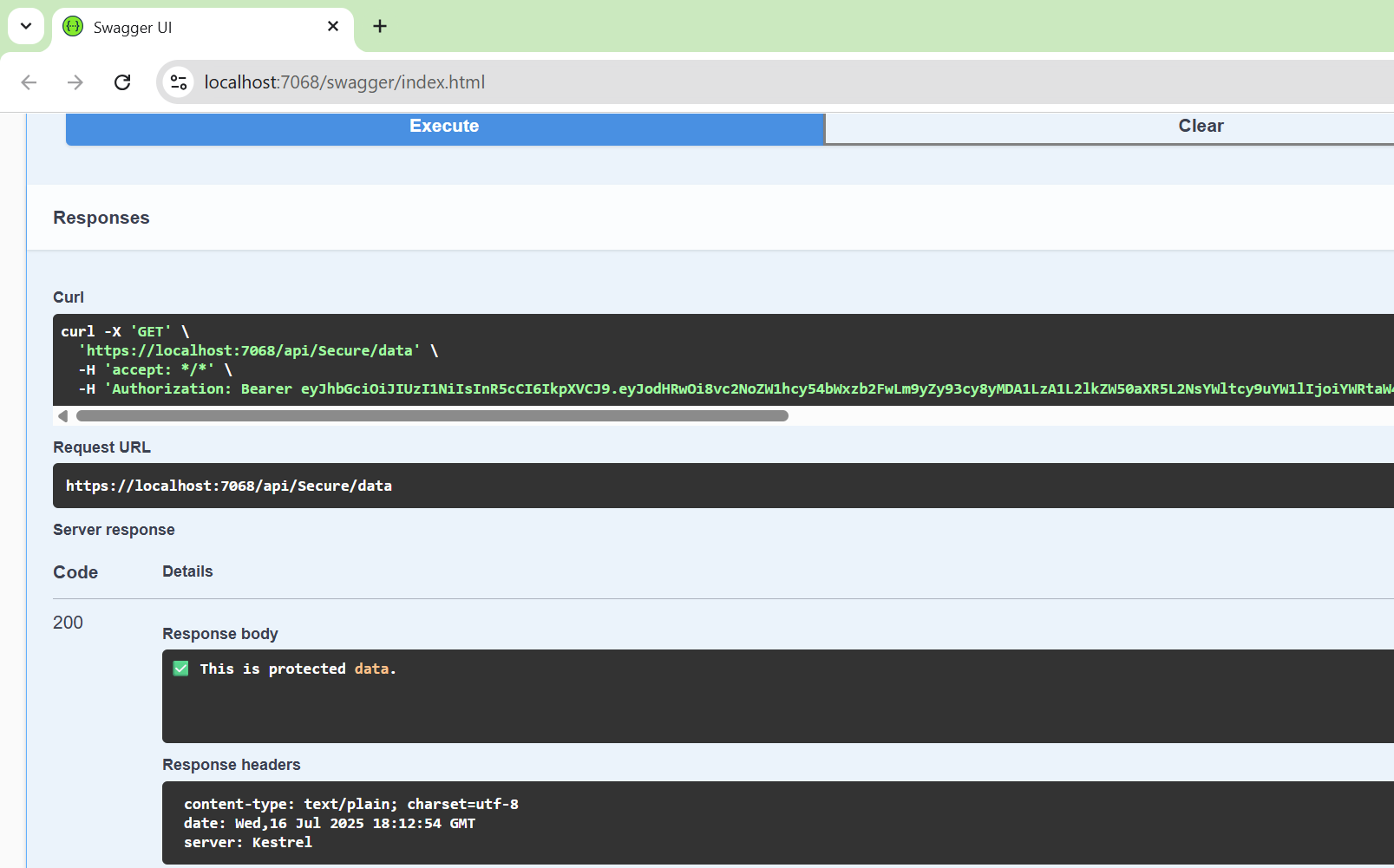
}

}

**Outputs:-**







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

**AuthController.cs**using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using Microsoft.Extensions.Configuration;

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 });

}

return Unauthorized(new { message = "Invalid username or password." });

}

private bool IsValidUser(LoginModel model)

{

// Simple hardcoded check

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

}

// THIS method must be INSIDE the class!

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin") // 👑 Role-based claim

};

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

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

var token = new JwtSecurityToken(

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

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

claims: claims,

expires: DateTime.Now.AddMinutes(60),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

// You can place this class in the same file or separately

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**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")] // 👮 Only Admins allowed

public IActionResult GetAdminDashboard()

{

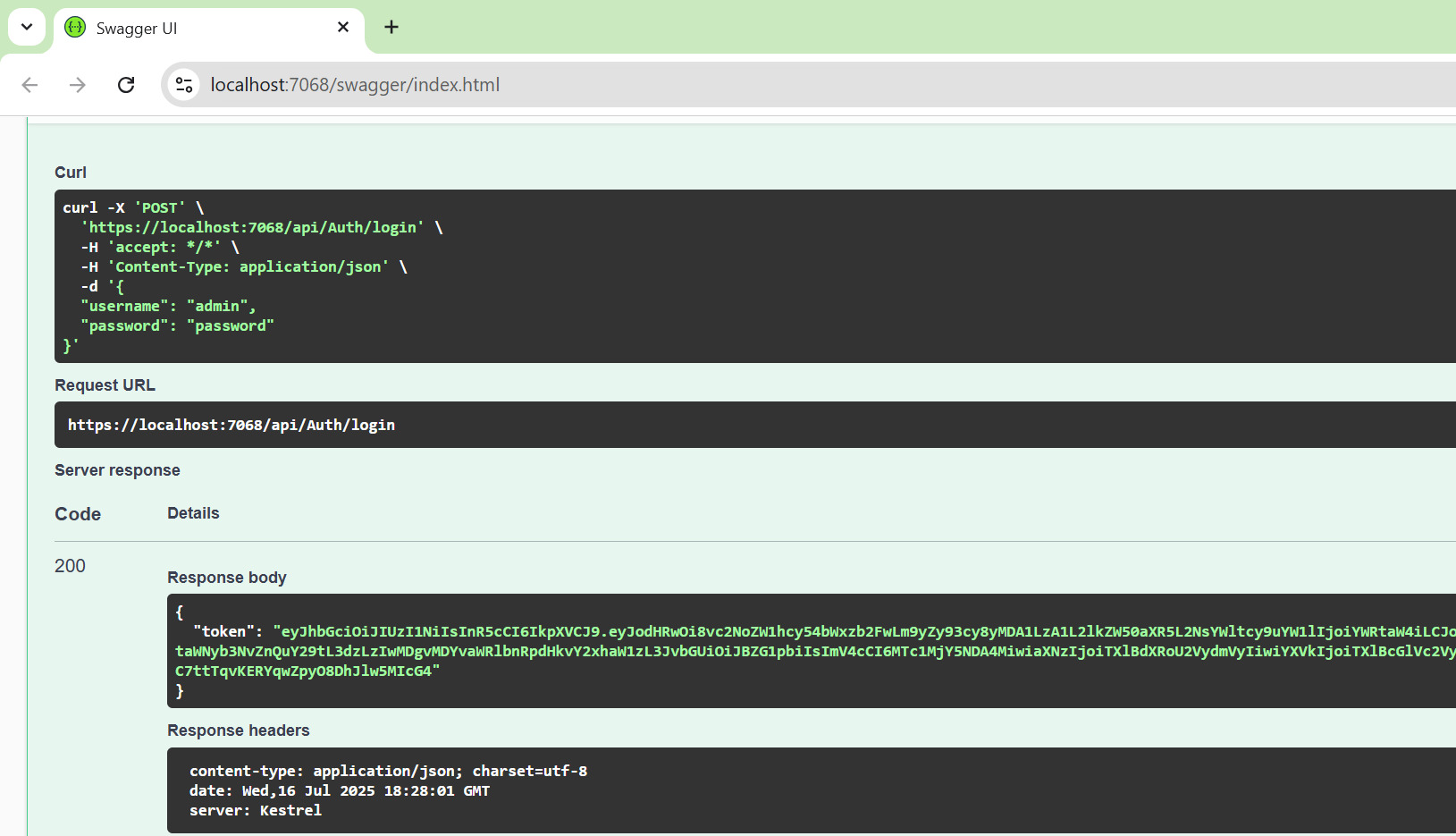
return Ok("📊 Welcome to the admin dashboard.");

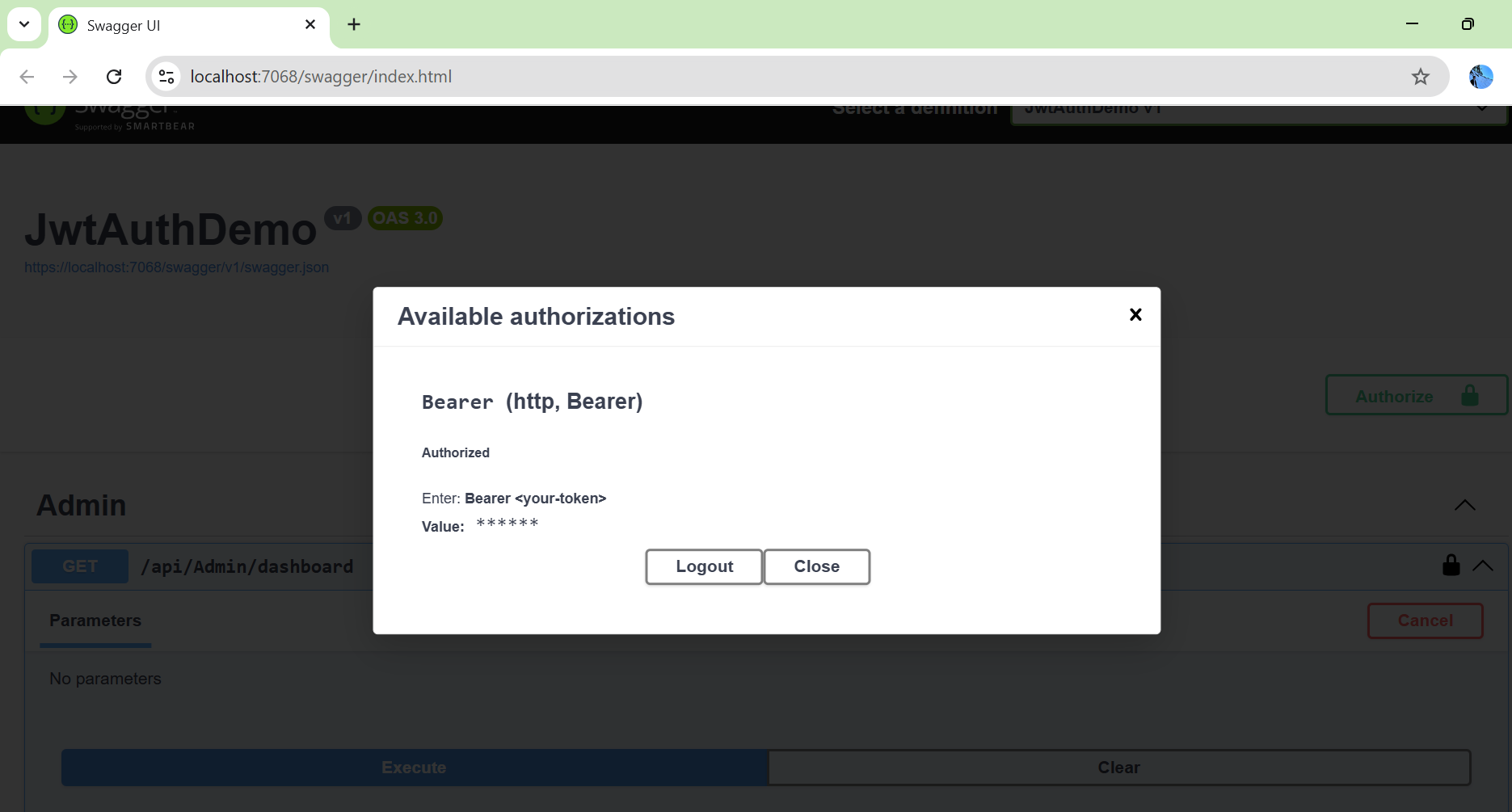
}

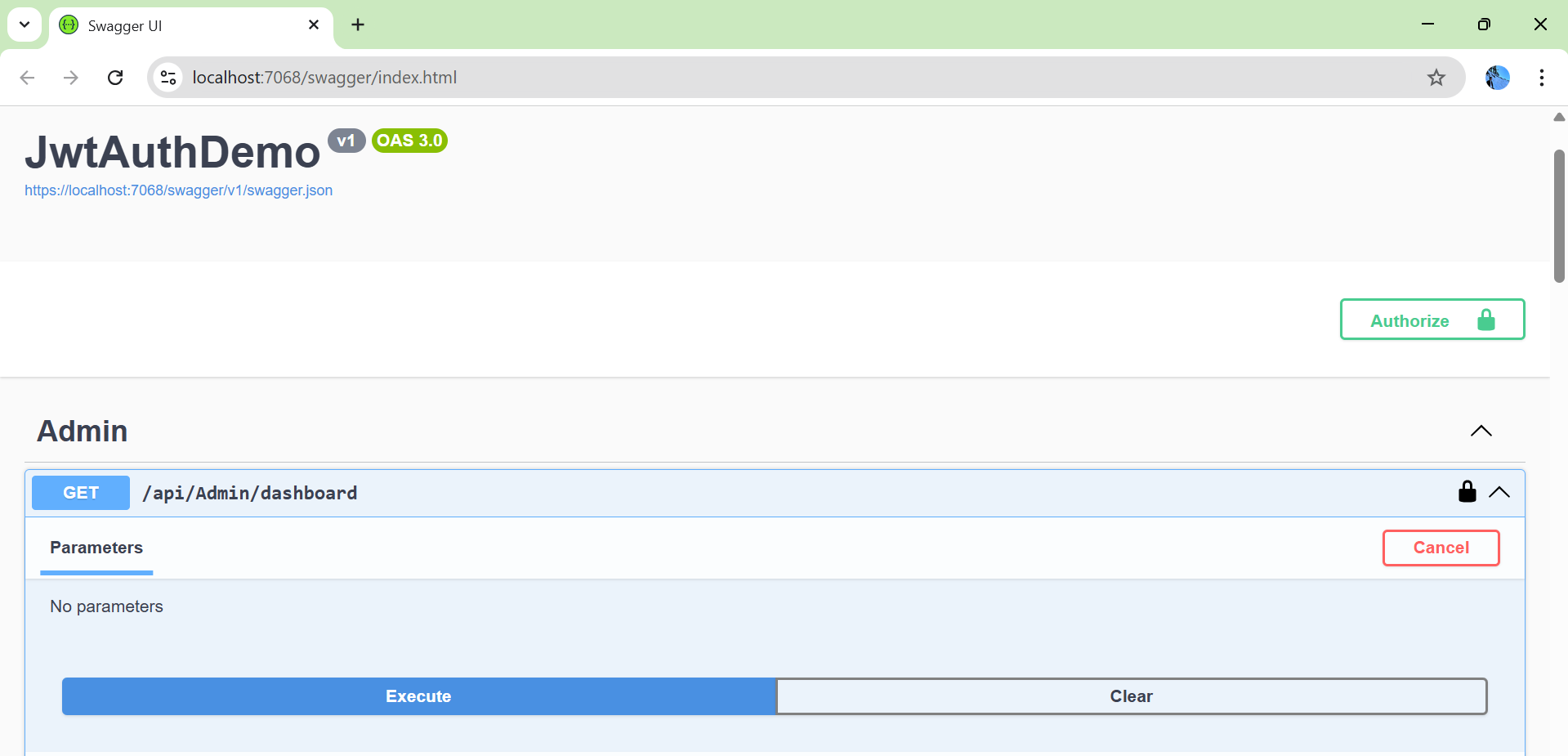
}

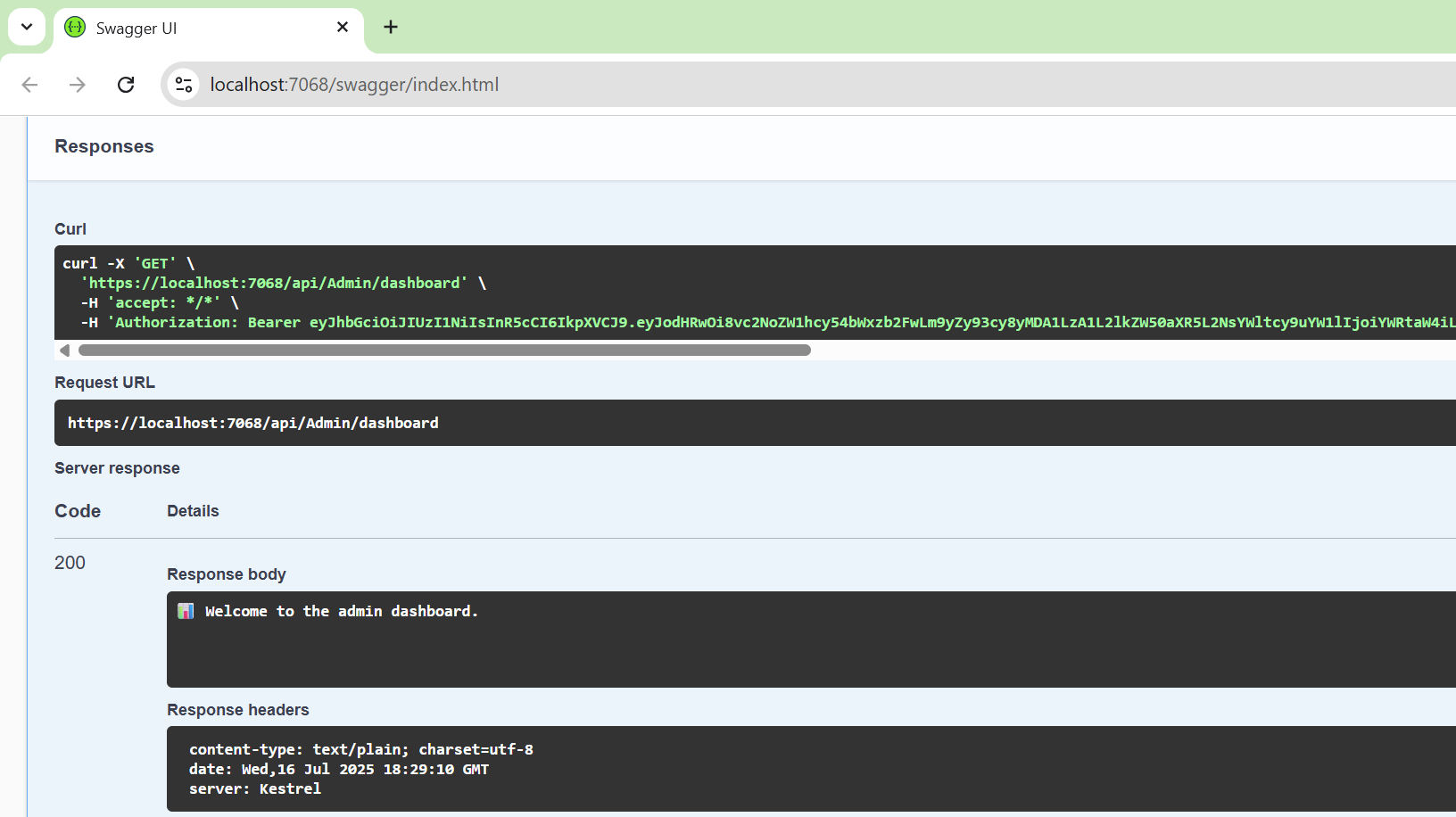
}

**Outputs:-**









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

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

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add services

builder.Services.AddControllers();

builder.Services.AddAuthentication(options =>

{

options.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

options.DefaultChallengeScheme = 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 expired tokens

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception is SecurityTokenExpiredException)

{

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

}

return Task.CompletedTask;

}

};

});

// Add role-based policy if needed

builder.Services.AddAuthorization();

// Add Swagger support with JWT UI

builder.Services.AddSwaggerGen(c =>

{

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

var jwtSecurityScheme = new OpenApiSecurityScheme

{

BearerFormat = "JWT",

Name = "Authorization",

In = ParameterLocation.Header,

Type = SecuritySchemeType.Http,

Scheme = "bearer",

Description = "Enter: Bearer <your JWT token>",

Reference = new OpenApiReference

{

Id = JwtBearerDefaults.AuthenticationScheme,

Type = ReferenceType.SecurityScheme

}

};

c.AddSecurityDefinition(jwtSecurityScheme.Reference.Id, jwtSecurityScheme);

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{ jwtSecurityScheme, Array.Empty<string>() }

});

});

var app = builder.Build();

// Configure middleware

app.UseSwagger();

app.UseSwaggerUI();

app.UseHttpsRedirection();

app.UseAuthentication(); // 🔒 Must come before Authorization

app.UseAuthorization();

app.MapControllers();

app.Run();

**AuthController.cs**using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

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 });

}

return Unauthorized(new { message = "Invalid username or password." });

}

private bool IsValidUser(LoginModel model)

{

// Hardcoded user for demo

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

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin")

};

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

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

var token = new JwtSecurityToken(

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

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

claims: claims,

// ⚠️ Expire in 30 seconds (short time for test)

expires: DateTime.Now.AddSeconds(30),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**Outputs:-**

