**Hands-On Exercises: Authentication and Authorization in ASP.NET Core**

**Web API Microservices**

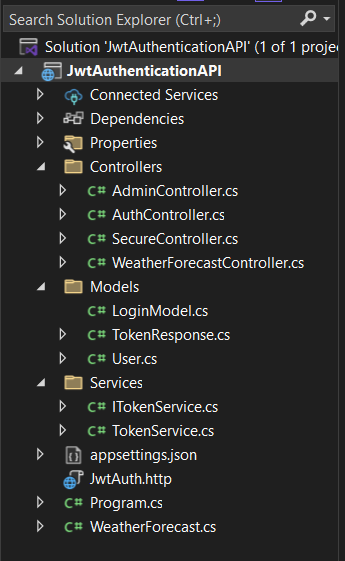
Question 1: Implement JWT Authentication in ASP.NET Core Web API

Question 2: Secure an API Endpoint Using JWT

Question 3: Add Role-Based Authorization

Question 4: Validate JWT Token Expiry and Handle Unauthorized Access

**File Structure:**



**Models/LoginModel.cs**

using System.ComponentModel.DataAnnotations;

namespace JwtAuthenticationAPI.Models

{

public class LoginModel

{

[Required]

[EmailAddress]

public string Email { get; set; } = string.Empty;

[Required]

[MinLength(6)]

public string Password { get; set; } = string.Empty;

}

}

**Models/TokenResponse.cs**

namespace JwtAuthenticationAPI.Models

{

public class TokenResponse

{

public string Token { get; set; } = string.Empty;

public DateTime Expires { get; set; }

public string Username { get; set; } = string.Empty;

public string Role { get; set; } = string.Empty;

}

}

**Models/User.cs**

namespace JwtAuthenticationAPI.Models

{

public class User

{

public int Id { get; set; }

public string Email { get; set; } = string.Empty;

public string Username { get; set; } = string.Empty;

public string Password { get; set; } = string.Empty;

public string Role { get; set; } = "User";

public DateTime CreatedAt { get; set; } = DateTime.UtcNow;

}

}

**Controller/AdminController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

namespace JwtAuthenticationAPI.Controllers

{

[ApiController]

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

public class AdminController : ControllerBase

{

[HttpGet("dashboard")]

[Authorize(Roles = "Admin")]

public IActionResult GetAdminDashboard()

{

var username = User.FindFirst(ClaimTypes.Name)?.Value;

return Ok(new

{

message = "Welcome to the admin dashboard",

adminUser = username,

adminData = new

{

totalUsers = 150,

activeUsers = 120,

todaysRegistrations = 5

},

timestamp = DateTime.UtcNow

});

}

[HttpGet("users")]

[Authorize(Roles = "Admin")]

public IActionResult GetAllUsers()

{

return Ok(new

{

message = "All users data (Admin only)",

users = new[]

{

new { id = 1, name = "John Doe", email = "john@example.com", role = "User" },

new { id = 2, name = "Jane Smith", email = "jane@example.com", role = "Admin" }

}

});

}

[HttpPost("promote-user/{userId}")]

[Authorize(Roles = "Admin")]

public IActionResult PromoteUser(int userId)

{

return Ok(new

{

message = $"User {userId} promoted to Admin successfully",

timestamp = DateTime.UtcNow

});

}

}

}

**Controller/AuthController.cs**

using Microsoft.AspNetCore.Mvc;

using JwtAuthenticationAPI.Models;

using JwtAuthenticationAPI.Services;

namespace JwtAuthenticationAPI.Controllers

{

[ApiController]

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

public class AuthController : ControllerBase

{

private readonly ITokenService \_tokenService;

private static readonly List<User> Users = new()

{

new User

{

Id = 1,

Email = "admin@example.com",

Username = "admin",

Password = "admin123",

Role = "Admin"

},

new User

{

Id = 2,

Email = "user@example.com",

Username = "user",

Password = "user123",

Role = "User"

}

};

public AuthController(ITokenService tokenService)

{

\_tokenService = tokenService;

}

[HttpPost("register")]

public IActionResult Register([FromBody] LoginModel model)

{

if (Users.Any(u => u.Email == model.Email))

{

return BadRequest(new { message = "User already exists" });

}

var user = new User

{

Id = Users.Count + 1,

Email = model.Email,

Username = model.Email.Split('@')[0],

Password = model.Password, Role = "User"

};

Users.Add(user);

var tokenResponse = \_tokenService.GenerateTokenResponse(user);

return Ok(new

{

message = "User registered successfully",

token = tokenResponse

});

}

[HttpPost("login")]

public IActionResult Login([FromBody] LoginModel model)

{

var user = AuthenticateUser(model);

if (user == null)

{

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

}

var tokenResponse = \_tokenService.GenerateTokenResponse(user);

return Ok(new

{

message = "Login successful",

token = tokenResponse

});

}

[HttpGet("users")]

public IActionResult GetUsers()

{

var usersWithoutPasswords = Users.Select(u => new

{

u.Id,

u.Email,

u.Username,

u.Role,

u.CreatedAt

}).ToList();

return Ok(usersWithoutPasswords);

}

private User? AuthenticateUser(LoginModel model)

{

return Users.FirstOrDefault(u =>

u.Email == model.Email && u.Password == model.Password);

}

}

}

**Controller/SecureController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Security.Claims;

namespace JwtAuthenticationAPI.Controllers

{

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

var username = User.FindFirst(ClaimTypes.Name)?.Value;

var email = User.FindFirst(ClaimTypes.Email)?.Value;

var role = User.FindFirst(ClaimTypes.Role)?.Value;

return Ok(new

{

message = "This is protected data accessible only with valid JWT token",

user = new

{

username,

email,

role

},

timestamp = DateTime.UtcNow

});

}

[HttpGet("user-profile")]

[Authorize]

public IActionResult GetUserProfile()

{

var claims = User.Claims.Select(c => new { c.Type, c.Value }).ToList();

return Ok(new

{

message = "User profile data",

claims = claims

});

}

}

}

**Services/ITokenService.cs**

using JwtAuthenticationAPI.Models;

namespace JwtAuthenticationAPI.Services

{

public interface ITokenService

{

string GenerateToken(User user);

TokenResponse GenerateTokenResponse(User user);

}

}

**Services/TokenService.cs**

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using JwtAuthenticationAPI.Models;

namespace JwtAuthenticationAPI.Services

{

public class TokenService : ITokenService

{

private readonly IConfiguration \_configuration;

public TokenService(IConfiguration configuration)

{

\_configuration = configuration;

}

public string GenerateToken(User user)

{

var claims = new[]

{

new Claim(ClaimTypes.NameIdentifier, user.Id.ToString()),

new Claim(ClaimTypes.Name, user.Username),

new Claim(ClaimTypes.Email, user.Email),

new Claim(ClaimTypes.Role, user.Role),

new Claim(JwtRegisteredClaimNames.Jti, Guid.NewGuid().ToString()),

new Claim(JwtRegisteredClaimNames.Iat,

new DateTimeOffset(DateTime.UtcNow).ToUnixTimeSeconds().ToString(),

ClaimValueTypes.Integer64)

};

var key = new SymmetricSecurityKey(

Encoding.UTF8.GetBytes(\_configuration["Jwt:Key"]!));

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

var token = new JwtSecurityToken(

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

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

claims: claims,

expires: DateTime.UtcNow.AddMinutes(

Convert.ToDouble(\_configuration["Jwt:DurationInMinutes"])),

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

public TokenResponse GenerateTokenResponse(User user)

{

var token = GenerateToken(user);

var expires = DateTime.UtcNow.AddMinutes(

Convert.ToDouble(\_configuration["Jwt:DurationInMinutes"]));

return new TokenResponse

{

Token = token,

Expires = expires,

Username = user.Username,

Role = user.Role

};

}

}

}

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using JwtAuthenticationAPI.Services;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

// Configure JWT Authentication

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"]!)),

ClockSkew = TimeSpan.Zero

};

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

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

{

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

}

return Task.CompletedTask;

},

OnChallenge = context =>

{

context.HandleResponse();

context.Response.StatusCode = 401;

context.Response.ContentType = "application/json";

var result = System.Text.Json.JsonSerializer.Serialize(new

{

error = "You are not authorized to access this resource.",

details = "Please provide a valid JWT token."

});

return context.Response.WriteAsync(result);

}

};

});

builder.Services.AddAuthorization();

builder.Services.AddScoped<ITokenService, TokenService>();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

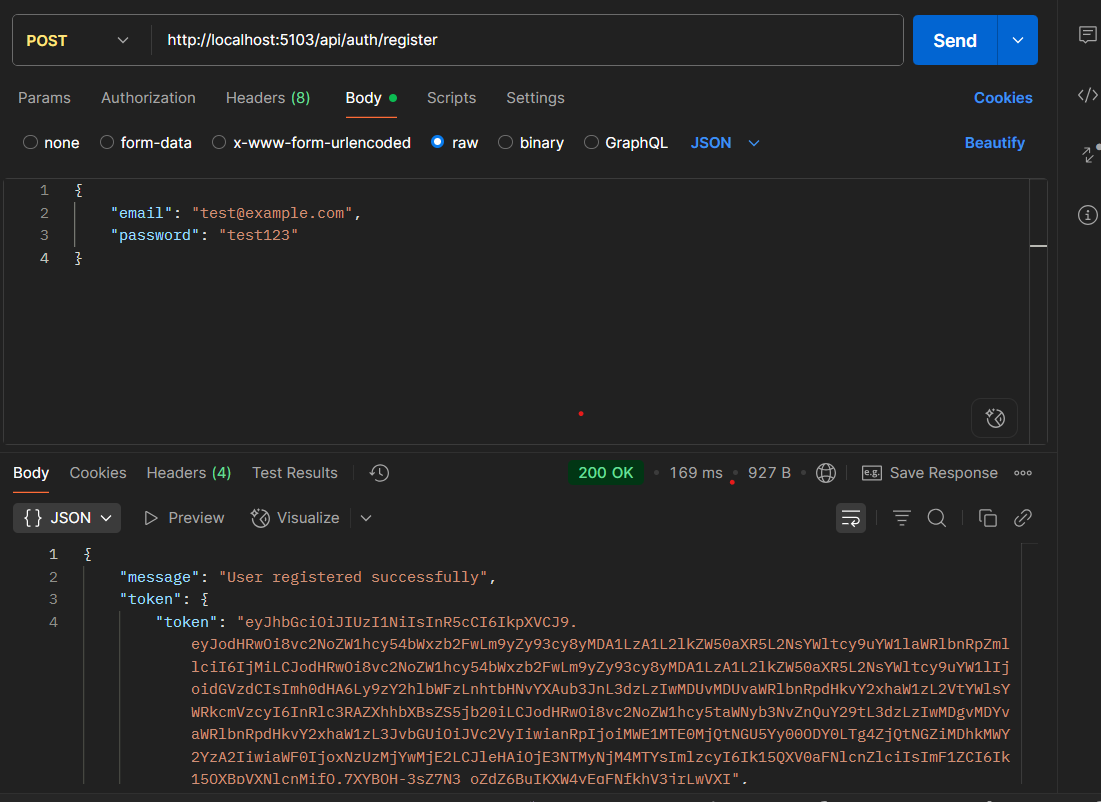
app.UseAuthentication();

app.UseAuthorization();

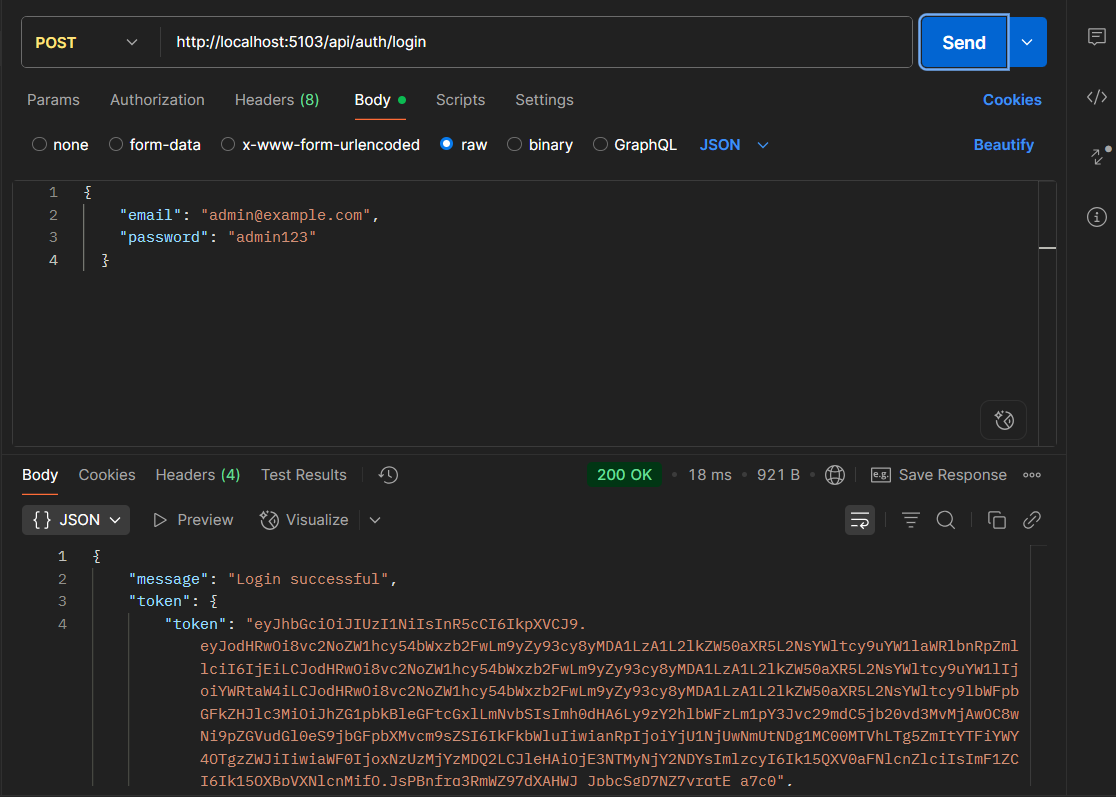
app.MapControllers();

app.Run();

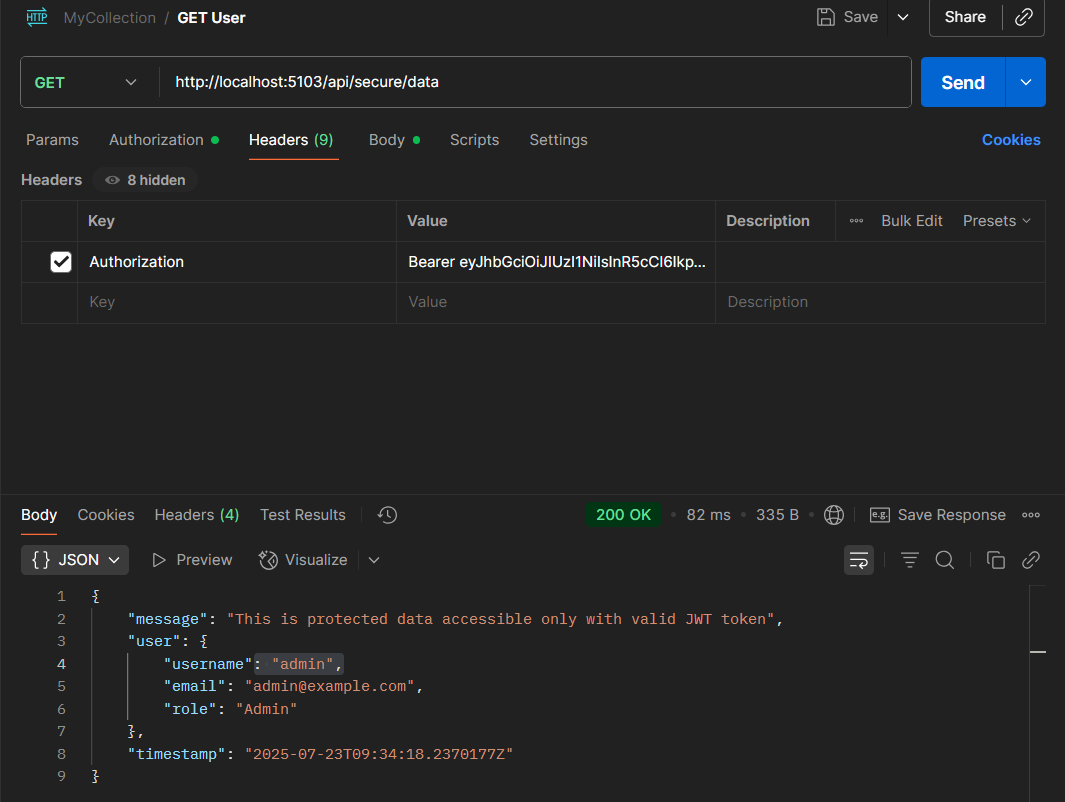
1. Test User Registration:



1. Test Login:



1. Test Secured Endpoint (replace TOKEN with actual token):



1. Test Admin Endpoint:

