# **Name- Sayandeep Dey (SupersetID:** **6363427)**

# **WEEK – 5 (Handson- Exercises)**

1. **ASP.NET Core 8.0 Web API:**

**LAB-6:WebApi\_Handson:-**

### **Code:**

**In kafkaChatApp:**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Confluent.Kafka" Version="1.9.2" />

</ItemGroup>

</Project>

**In Program.cs:**

using Confluent.Kafka;

Console.WriteLine("Enter message to send to Kafka (type 'exit' to quit):");

var config = new ProducerConfig { BootstrapServers = "localhost:9092" };

using var producer = new ProducerBuilder<Null, string>(config).Build();

while (true)

{

var input = Console.ReadLine();

if (input?.ToLower() == "exit") break;

await producer.ProduceAsync("test-topic", new Message<Null, string> { Value = input });

Console.WriteLine("Message sent!");

}

**Adding Second Project As KafkaConsumerApp:**

<Project Sdk="Microsoft.NET.Sdk">

<PropertyGroup>

<OutputType>Exe</OutputType>

<TargetFramework>net8.0</TargetFramework>

<ImplicitUsings>enable</ImplicitUsings>

<Nullable>enable</Nullable>

</PropertyGroup>

<ItemGroup>

<PackageReference Include="Confluent.Kafka" Version="1.9.2" />

</ItemGroup>

</Project>

**In Program.cs:**

using Confluent.Kafka;

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-consumer-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

using var consumer = new ConsumerBuilder<Ignore, string>(config).Build();

consumer.Subscribe("test-topic");

Console.WriteLine("Listening to messages. Press Ctrl+C to exit.");

try

{

while (true)

{

var result = consumer.Consume();

Console.WriteLine($"Received: {result.Message.Value}");

}

}

catch (OperationCanceledException)

{

consumer.Close();

}

**Output:**

**In Terminal Output (CMD):**

**In Producer side:**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**In Consumer Side:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**In Visual Studio Output:**

**After Running KafkaChatApp:**

**A computer screen with a black screen

AI-generated content may be incorrect.**

**After running KafkaConsumerApp:**

**A screenshot of a computer

AI-generated content may be incorrect.**

1. **Microservices Architecture using ASP.NET Core Web API:**
2. **Microservices – JWT**

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

**Code:**

**In Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// Add controller support

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"]))

};

});

// Add role/claims-based authorization

builder.Services.AddAuthorization();

var app = builder.Build();

// Use middleware in correct order

app.UseHttpsRedirection();

app.UseAuthentication(); // Always before UseAuthorization

app.UseAuthorization();

// Map endpoints

app.MapControllers();

app.Run();

**In appsettings.json:**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Jwt": {

"Key": "ThisIsA32CharLongSecretKeyValue!!",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": "60"

}

}

**In Models/ LoginModels.cs:**

namespace JwtAuth.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**In Models/ User.cs:**

namespace JwtAuth.Models

{

public class User

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**In Controllers/SecureController.cs:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System.Linq;

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("data")]

[Authorize]

public IActionResult GetSecureData()

{

var username = User.Identity?.Name ?? "Unknown";

return Ok($"Hello {username}, you have accessed a protected endpoint!");

}

}

**In Controllers/AuthController.cs:**

using JwtAuth.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuth.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();

}

private bool IsValidUser(LoginModel model)

{

// Dummy validation logic

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

}

private string GenerateJwtToken(string username)

{

var claims = new[]

{

new Claim(ClaimTypes.Name, username)

};

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(double.Parse(\_config["Jwt:DurationInMinutes"])),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**Output:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**After Copying Token and run GET** [**https://localhost:7025/api/secure/data**](https://localhost:7025/api/secure/data)**:**

**Click on Authorization🡪 Bearer Token 🡪 Paste the token and showing output as:**

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**Code:**

**In SecureController.cs:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

[Authorize] // Entire controller is protected

public class SecureController : ControllerBase

{

[HttpGet("data")]

public IActionResult GetSecureData()

{

var username = User.Identity?.Name ?? "Unknown";

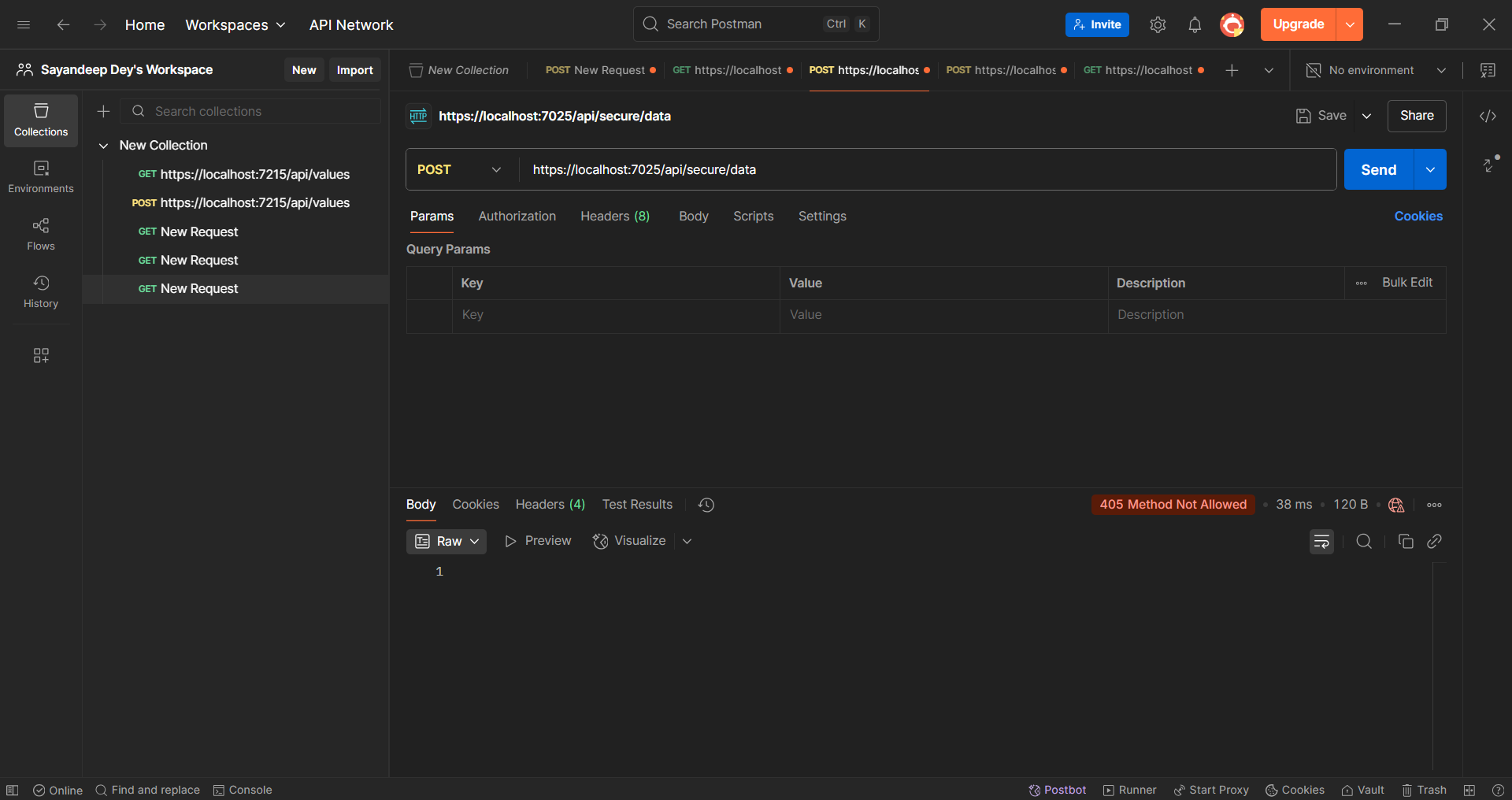
return Ok($"Hello {username}, you have accessed a protected endpoint!");

}

}

**Rest All the codes are same as Question 1 and Build and Run:**

**Output:**

****

**A screenshot of a computer

AI-generated content may be incorrect.**

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**Code:**

**AdminController.cs:**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

[ApiController]

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

public class AdminController : ControllerBase

{

[HttpGet("dashboard")]

[Authorize(Roles = "Admin")]

public IActionResult GetAdminDashboard()

{

return Ok("Admin dashboard accessed.");

}

[HttpGet("manager-report")]

[Authorize(Roles = "Manager")]

public IActionResult GetManagerReport()

{

return Ok("Manager report accessed.");

}

[HttpGet("combined")]

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

public IActionResult GetForEitherRole()

{

return Ok("Access granted to Admin or Manager.");

}

}

**AuthController.cs:**

using JwtAuth.Models;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace JwtAuth.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();

}

private bool IsValidUser(LoginModel model)

{

// Dummy validation logic

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

}

private string GenerateJwtToken(string username)

{

// Add multiple roles: Admin and Manager

var claims = new List<Claim>

{

new Claim(ClaimTypes.Name, username),

new Claim(ClaimTypes.Role, "Admin"),

new Claim(ClaimTypes.Role, "Manager")

};

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(double.Parse(\_config["Jwt:DurationInMinutes"])),

signingCredentials: creds);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**Output:**

**Generating Token:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Access for Admin Dashboard:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Access for Only Manager- Report:**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Access for Both Admin and Manager:**

**A screenshot of a computer

AI-generated content may be incorrect.**

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

**Code:**

**In apsetting.json:**

**Adding DurationInMinutes: 1**

{

"Logging": {

"LogLevel": {

"Default": "Information",

"Microsoft.AspNetCore": "Warning"

}

},

"Jwt": {

"Key": "ThisIsA32CharLongSecretKeyValue!!",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": "1"

}

}

**In Program.cs:**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

using System.IdentityModel.Tokens.Jwt;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddAuthentication("Bearer")

.AddJwtBearer("Bearer", 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"])),

// 🔥 This disables the 5-minute grace period

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;

}

};

});

builder.Services.AddAuthorization();

var app = builder.Build();

app.UseHttpsRedirection();

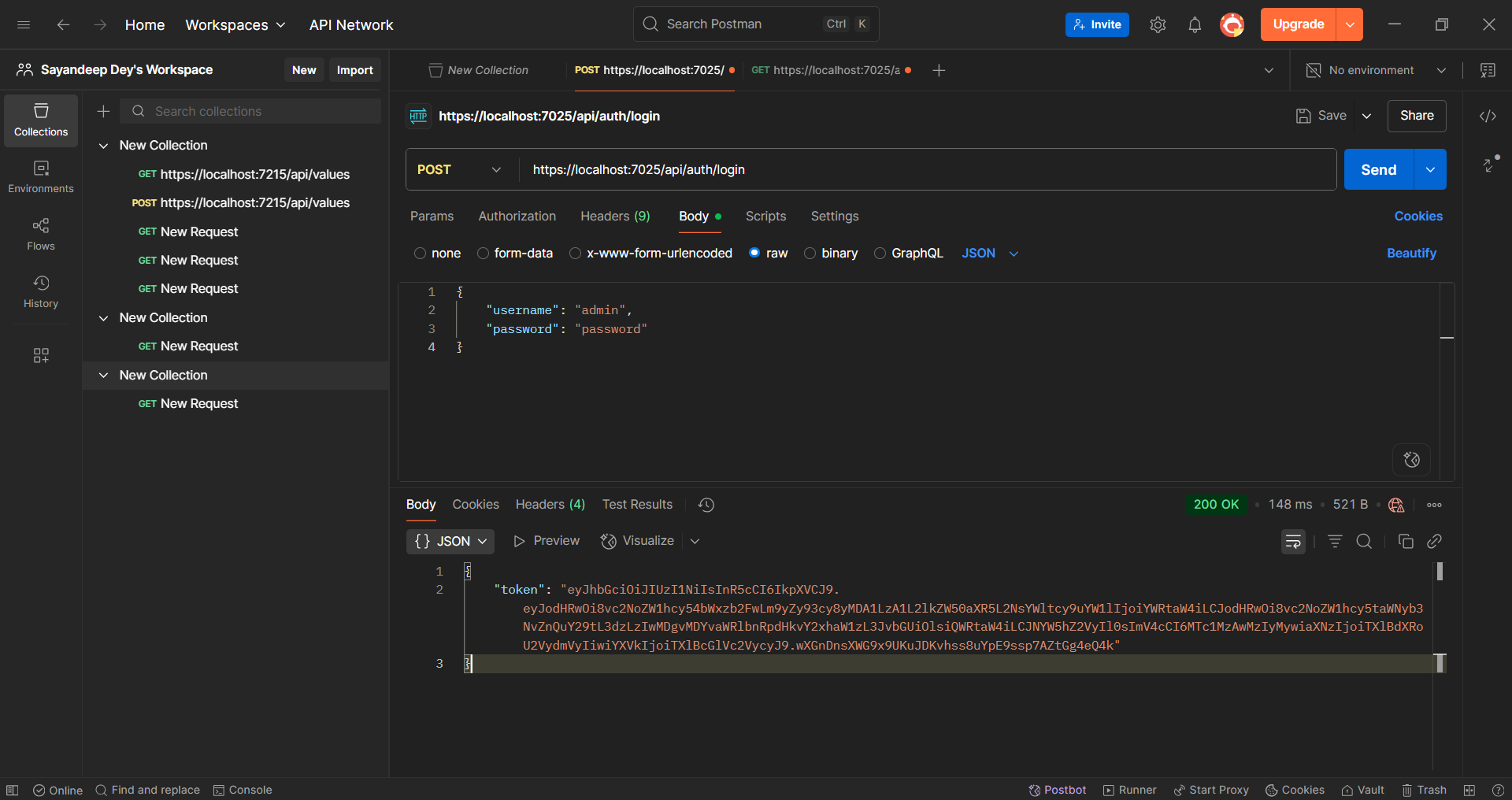
app.UseAuthentication();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Output:**

****

**For Safe-Side 5 minutes later Calling GET Method:**

**A screenshot of a computer

AI-generated content may be incorrect.**