1. **WebApi\_Handson**

**chatConsumer.cs**

using Confluent.Kafka;

using System;

using System.Threading;

public class ChatConsumer

{

public static void Start()

{

var config = new ConsumerConfig

{

GroupId = "chat-group",

BootstrapServers = "localhost:9092",

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

Console.WriteLine("Waiting for incoming messages...\n");

while (true)

{

var result = consumer.Consume(CancellationToken.None);

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

}

}

}

**ChatProducer.cs**

using Confluent.Kafka;

using System;

using System.Threading.Tasks;

public class ChatProducer

{

public static async Task StartAsync()

{

var config = new ProducerConfig

{

BootstrapServers = "localhost:9092"

};

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

Console.WriteLine("Chat started. Type your message (type 'exit' to quit):");

while (true)

{

Console.Write("You: ");

var input = Console.ReadLine();

if (input == "exit") break;

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

}

}

}

Program.cs

using System;

using System.Threading.Tasks;

class Program

{

static async Task Main(string[] args)

{

Console.WriteLine("Enter 'send' to send messages or 'receive' to receive messages:");

var mode = Console.ReadLine();

if (mode == "send")

await ChatProducer.StartAsync();

else if (mode == "receive")

ChatConsumer.Start();

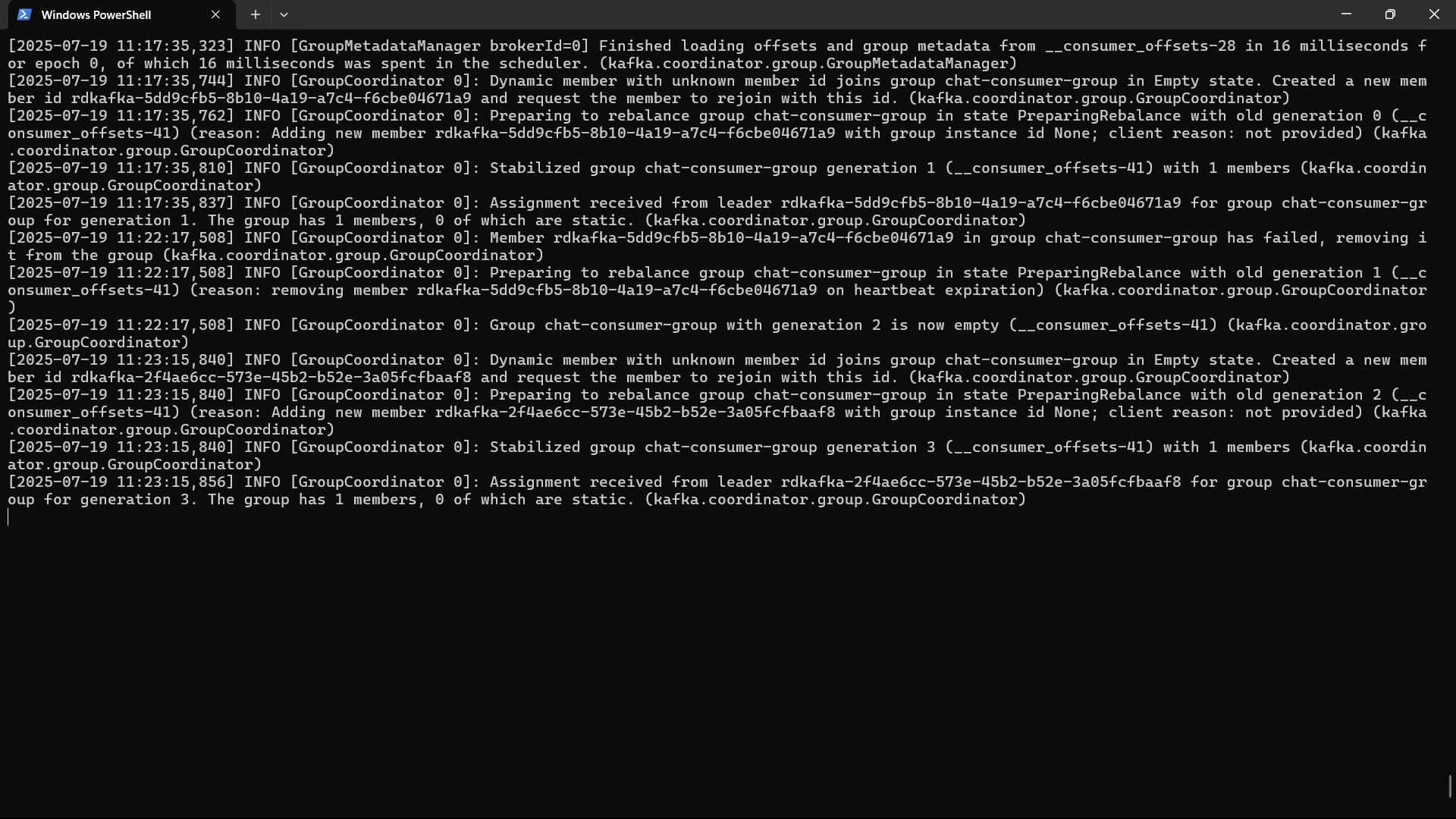
else

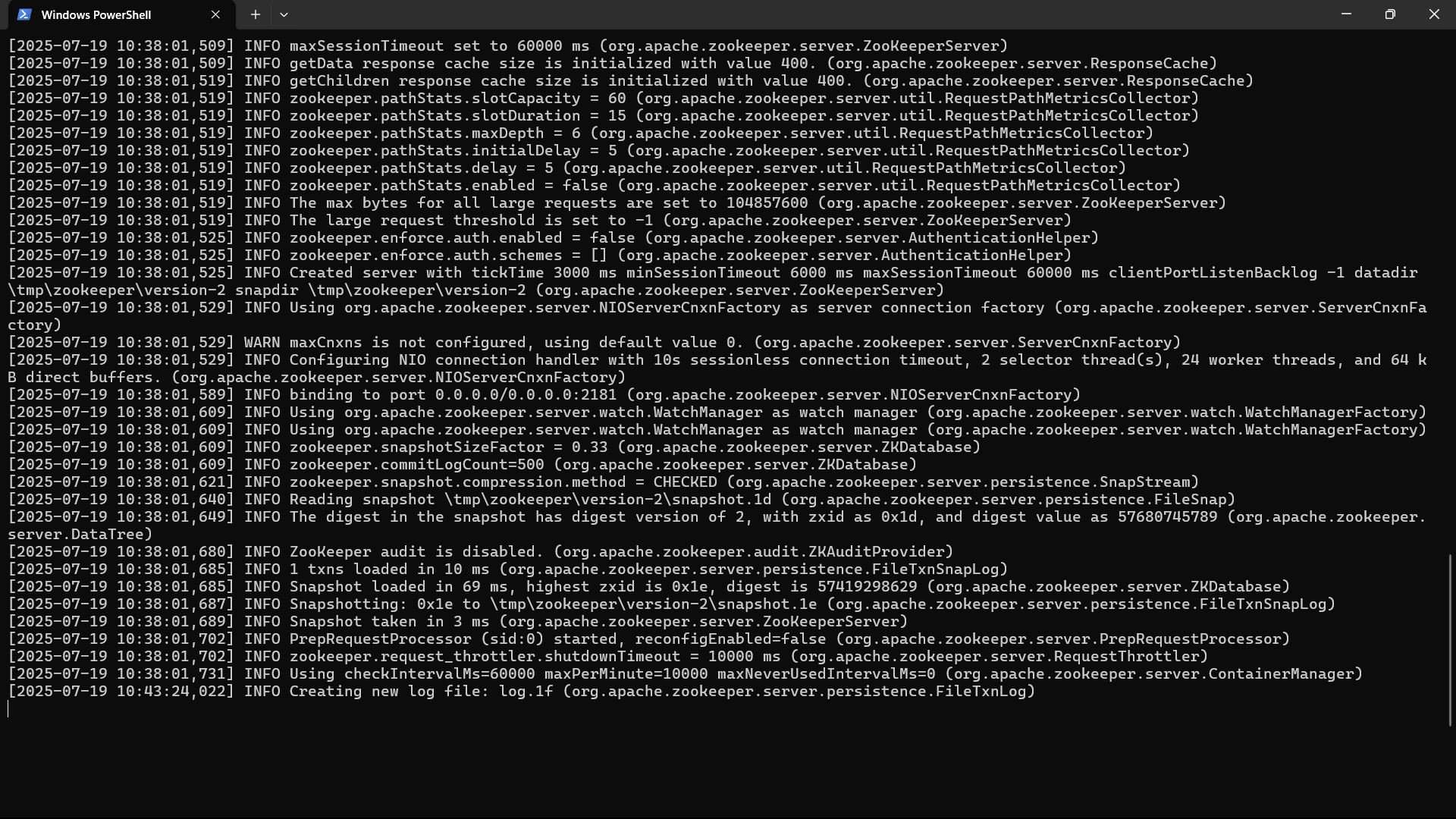
Console.WriteLine("Invalid input.");

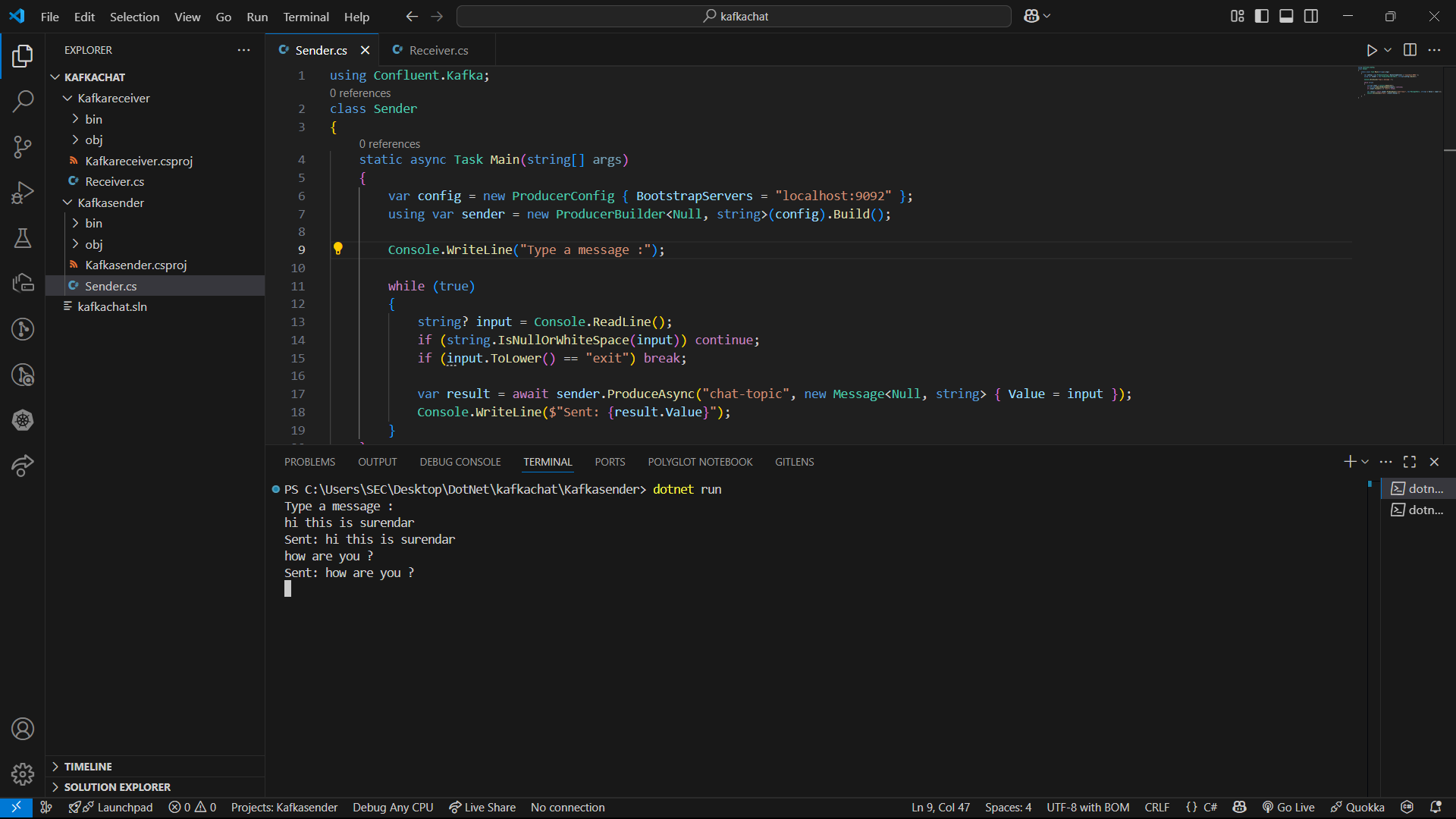
}

}

OUTPUT







1. **Microservices - JWT**

Program.cs

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using Microsoft.OpenApi.Models;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// JWT Authentication Configuration

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"]))

};

});

// Authorization Middleware

builder.Services.AddAuthorization();

// Add Controllers

builder.Services.AddControllers();

// 📘 Swagger Configuration with JWT Bearer Support

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen(c =>

{

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

// ✅ Define the BearerAuth scheme

c.AddSecurityDefinition("Bearer", new OpenApiSecurityScheme

{

Name = "Authorization",

Type = SecuritySchemeType.Http,

Scheme = "Bearer",

BearerFormat = "JWT",

In = ParameterLocation.Header,

Description = "Enter 'Bearer' [space] and then your valid token.\n\nExample: Bearer eyJhbGciOiJIUzI1NiIs..."

});

// Apply it globally to all operations

c.AddSecurityRequirement(new OpenApiSecurityRequirement

{

{

new OpenApiSecurityScheme

{

Reference = new OpenApiReference

{

Type = ReferenceType.SecurityScheme,

Id = "Bearer"

}

},

Array.Empty<string>()

}

});

});

var app = builder.Build();

// Middleware Pipeline

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthentication(); // Important: MUST be before UseAuthorization

app.UseAuthorization();

app.MapControllers();

app.Run();

SecureController.cs

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

namespace JwtLoginAPI.Controllers

{

[ApiController]

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

public class SecureController : ControllerBase

{

[HttpGet("secret")]

[Authorize]

public IActionResult GetSecret()

{

var username = User.Identity?.Name;

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

}

}

}

AuthController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

using JwtLoginAPI.Models;

namespace JwtLoginAPI.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("Invalid credentials");

}

private bool IsValidUser(LoginModel model)

{

// Dummy validation

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

signingCredentials: creds

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

LoginModel.cs

namespace JwtLoginAPI.Models

{

public class LoginModel

{

public string Username { get; set; }

public string Password { get; set; }

}

}

**OUTPUT**

