ASP.NET Core 8.0 Web API

6. WebApi\_Handson

Kafka Setup

# Zookeeper

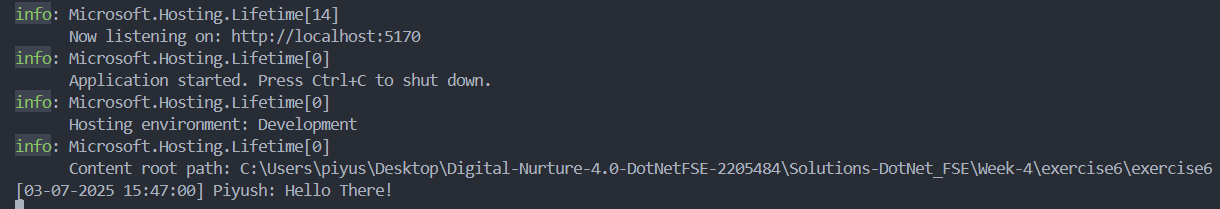
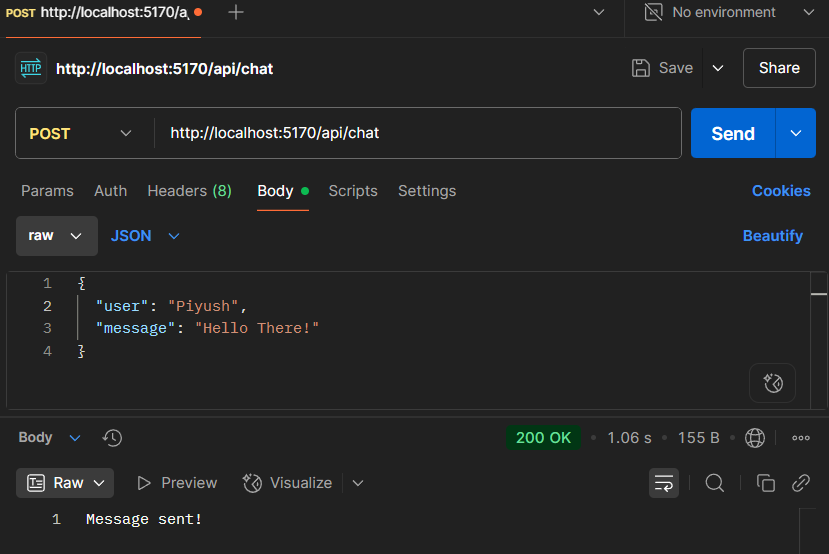
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

# Kafka

.\bin\windows\kafka-server-start.bat .\config\server.properties

# Create topic

.\bin\windows\kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1



# Create project

dotnet new winforms -n KafkaChatClient

cd KafkaChatClient

# Add Kafka dependency

dotnet add package Confluent.Kafka

# Replace Form1.cs with provided Kafka chat logic

# Add TextBox, Button, ListBox in Form1.Designer.cs

dotnet run

Kafka Chat Logic

Form1.cs

using System;

using System.Text.Json;

using System.Threading;

using System.Threading.Tasks;

using System.Windows.Forms;

using Confluent.Kafka;

namespace task2

{

public partial class Form1 : Form

{

private CancellationTokenSource \_cts = new();

public Form1()

{

InitializeComponent();

StartConsumer();

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private async void btnSend\_Click(object sender, EventArgs e)

{

var chat = new ChatMessage

{

User = txtUser.Text,

Message = txtMessage.Text

};

await SendMessageAsync(chat);

txtMessage.Clear();

}

private async Task SendMessageAsync(ChatMessage chat)

{

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

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

string json = JsonSerializer.Serialize(chat);

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

}

private void StartConsumer()

{

Task.Run(() =>

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = Guid.NewGuid().ToString(),

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

try

{

while (!\_cts.Token.IsCancellationRequested)

{

var result = consumer.Consume(\_cts.Token);

var chat = JsonSerializer.Deserialize<ChatMessage>(result.Message.Value);

Invoke(new Action(() =>

{

lstMessages.Items.Add($"[{chat.Timestamp:T}] {chat.User}: {chat.Message}");

}));

}

}

catch (OperationCanceledException) { consumer.Close(); }

});

}

}

public class ChatMessage

{

public string User { get; set; }

public string Message { get; set; }

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

}

}

Form1.Design.cs

namespace task2

{

partial class Form1

{

private System.Windows.Forms.TextBox txtUser;

private System.Windows.Forms.TextBox txtMessage;

private System.Windows.Forms.Button btnSend;

private System.Windows.Forms.ListBox lstMessages;

/// <summary>

/// Required method for Designer support

/// </summary>

private void InitializeComponent()

{

this.txtUser = new System.Windows.Forms.TextBox();

this.txtMessage = new System.Windows.Forms.TextBox();

this.btnSend = new System.Windows.Forms.Button();

this.lstMessages = new System.Windows.Forms.ListBox();

this.SuspendLayout();

// txtUser

this.txtUser.Location = new System.Drawing.Point(12, 12);

this.txtUser.Name = "txtUser";

this.txtUser.PlaceholderText = "Enter your name";

this.txtUser.Size = new System.Drawing.Size(200, 23);

// txtMessage

this.txtMessage.Location = new System.Drawing.Point(12, 41);

this.txtMessage.Name = "txtMessage";

this.txtMessage.PlaceholderText = "Type your message";

this.txtMessage.Size = new System.Drawing.Size(300, 23);

// btnSend

this.btnSend.Location = new System.Drawing.Point(318, 41);

this.btnSend.Name = "btnSend";

this.btnSend.Size = new System.Drawing.Size(75, 23);

this.btnSend.Text = "Send";

this.btnSend.Click += new System.EventHandler(this.btnSend\_Click);

// lstMessages

this.lstMessages.FormattingEnabled = true;

this.lstMessages.ItemHeight = 15;

this.lstMessages.Location = new System.Drawing.Point(12, 70);

this.lstMessages.Name = "lstMessages";

this.lstMessages.Size = new System.Drawing.Size(381, 169);

// Form1

this.ClientSize = new System.Drawing.Size(405, 250);

this.Controls.Add(this.txtUser);

this.Controls.Add(this.txtMessage);

this.Controls.Add(this.btnSend);

this.Controls.Add(this.lstMessages);

this.Name = "Form1";

this.Text = "Kafka Chat Client";

this.Load += new System.EventHandler(this.Form1\_Load);

this.ResumeLayout(false);

this.PerformLayout();

}

}

}

Program.cs

using System;

using System.Windows.Forms;

namespace task2

{

static class Program

{

[STAThread]

static void Main()

{

Application.SetHighDpiMode(HighDpiMode.SystemAware);

Application.EnableVisualStyles();

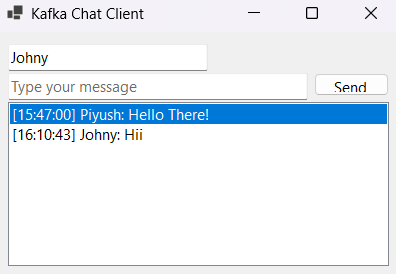
Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}



Kafka & Zookeeper Setup (Windows CLI)

Open CMD in Kafka directory

Start Zookeeper:

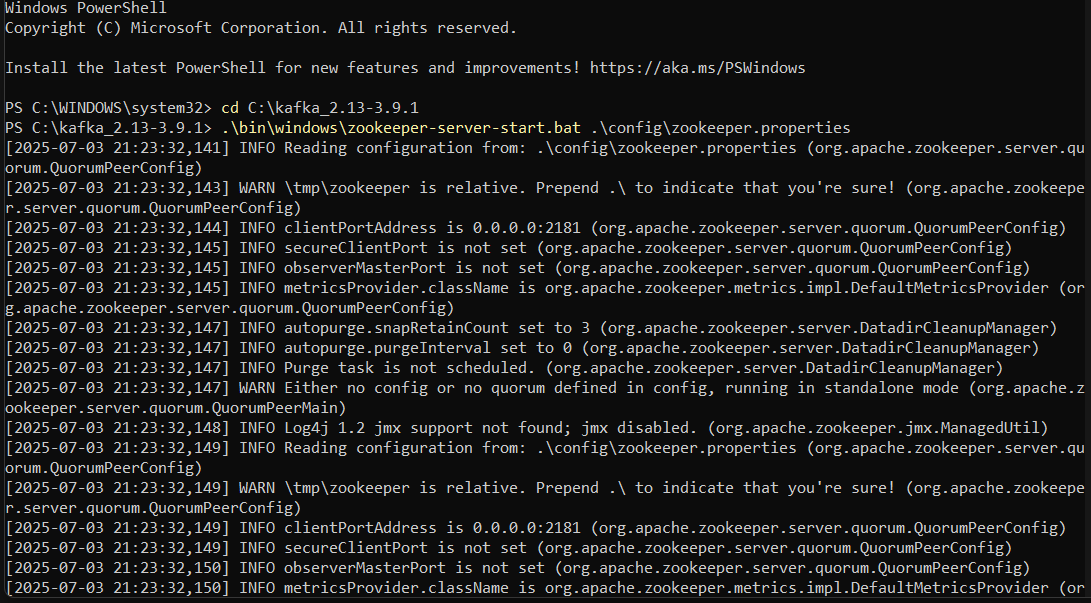
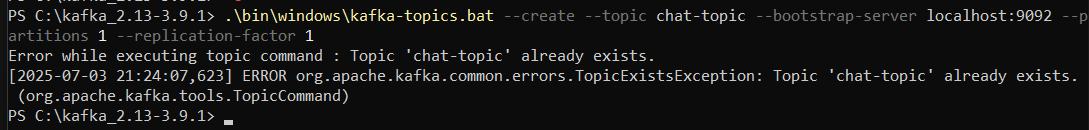
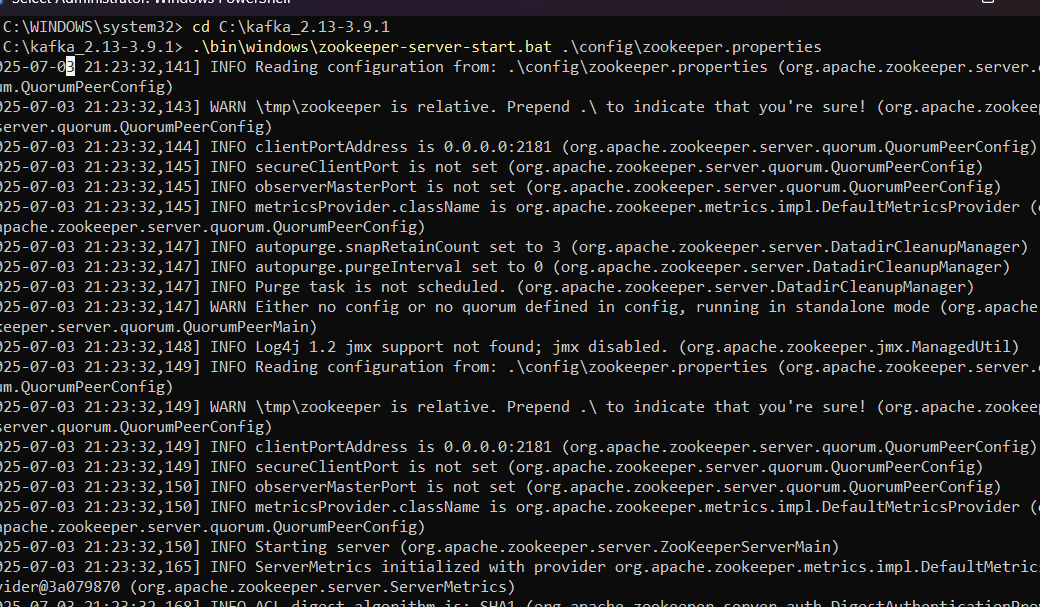
.\bin\windows\zookeeper-server-start.bat .\config\zookeeper.properties

Open new CMD => Start Kafka Broker:

.\bin\windows\kafka-server-start.bat .\config\server.properties

Create a topic:

.\bin\windows\kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1



Microservices Architecture using ASP.NET Core Web API

1. Microservices – JWT

Configuration

appsettings.json

{

"Jwt": {

"Key": "<YOUR\_SECRET\_KEY>",

"Issuer": "MyAuthServer",

"Audience": "MyApiUsers",

"DurationInMinutes": 60

}

}

**Question 1: Implement JWT Authentication**

**GOAL**

Issue a JWT token after successful login.

**Steps**

* Add LoginModel.cs for login credentials.
* Create IJwtService and JwtService using DI.
* Implement /api/auth/login endpoint in AuthController.
* Use SymmetricSecurityKey to sign the token.

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

**GOAL**

Restrict access to an endpoint with [Authorize].

**Steps**

* Add SecureController.cs.
* Use [Authorize] attribute.

*Example:*

[Authorize]

[HttpGet("data")]

public IActionResult GetSecureData()

{

var username = User.Identity?.Name;

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

}

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

**GOAL**

Allow only Admin role users to access a specific route.

**Steps:**

* Add "Admin" to token claims during login.
* Add [Authorize(Roles = "Admin")] to controller.

*Example:*

[Authorize(Roles = "Admin")]

[HttpGet("dashboard")]

public IActionResult GetAdminDashboard()

{

return Ok("Welcome to the Admin Dashboard");

}

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

**GOAL**

Detect expired tokens and return custom headers.

**Steps:** In Program.cs, configure JWT Bearer events:

options.Events = new JwtBearerEvents

{

OnAuthenticationFailed = context =>

{

if (context.Exception is SecurityTokenExpiredException)

{

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

}

return Task.CompletedTask;

}

};

