**WEBAPI**

**RESTful Web Services**

**REST (Representational State Transfer)** is an architectural style for designing networked applications.

RESTful services use **HTTP methods** and are **stateless** – each request contains all the information needed to process it.

Typically works with **JSON** or **XML** data over HTTP.

**Features of REST Architecture**

**Stateless** – Server doesn’t store client context.

**Client-Server Architecture** – Separation of concerns.

**Cacheable** – Responses can be cached.

**Uniform Interface** – Use of standard HTTP methods like GET, POST, PUT, DELETE.

**Representations** – Resources are represented in JSON/XML, etc.

**Messages** – Communication via HTTP messages.

**Web API**

A **Web API** is a framework for building HTTP services.

Used to expose data and services to clients like mobile apps, browsers, other services.

Supports REST principles.

**Microservices**

An **architecture style** that structures an application as a collection of loosely coupled, independently deployable services.

Each service is responsible for **a single business capability**.

Enables **scalability, resilience**, and **independent deployment**.

**HttpRequest & HttpResponse**

**HttpRequest**

Represents the incoming **request from client to server**.

Contains data like:

URL

HTTP Method (GET/POST)

Headers

Body (for POST/PUT)

**HttpResponse**

Represents the **response from server to client**.

Contains:

Status Code (e.g., 200, 404)

Headers

Response body (data)

1. First Web Api using .Net core Create a .Net core web application with API template. Use the option to create controller with Read Write permissions. Notice the ValuesController creation with Action methods corresponding to the Action verbs. On creation of the Web API, execute the application and check if the GET action method result is returned as expected.

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace FirstWebApiDemo.Controllers

{

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

[ApiController]

public class ValuesController : ControllerBase

{

// GET: api/values

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

return new string[] { "value1", "value2" };

}

// GET: api/values/5

[HttpGet("{id}")]

public ActionResult<string> Get(int id)

{

return $"value {id}";

}

// POST: api/values

[HttpPost]

public void Post([FromBody] string value)

{

}

// PUT: api/values/5

[HttpPut("{id}")]

public void Put(int id, [FromBody] string value)

{

}

// DELETE: api/values/5

[HttpDelete("{id}")]

public void Delete(int id)

{

}

}

}

* Demonstrate Swagger installation to WebAPI and WebAPI listing on browser
  + Nuget package to download Swashbuckle.AspNetCore, Usage of ProducesResponseType to Web API method, AddSwaggerGen, UseSwaggerUI
* Demonstrate the usage of Postman tool to hit WebAPI methods
  + Structure in Postman tool, Headers with Authorization, Body as JSON, Option to choose the type of request, Request collection and how to add a new request in the collection, Tabs in the center pane that corresponds to the request
* Demonstrate the usage of Route and Explain Name attribute in Http requests
  + Importance of user friendly name to action method, Explain the usage of ActionName to have more than 1 method with the same Action verbDemonstrate creation of a simple WebAPI - With Read, Write actions

SWAGGER SETUP

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Enable Swagger middleware

app.UseSwagger();

app.UseSwaggerUI();

app.UseAuthorization();

app.MapControllers();

app.Run();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = new Uri("https://example.com"),

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

Url = new Uri("https://www.example.com")

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

Url = new Uri("https://www.example.com")

}

});

});

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

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

[ApiController]

public class EmployeeController : ControllerBase

{

static List<string> employees = new List<string> { "Alice", "Bob", "Charlie" };

[HttpGet]

public IActionResult Get()

{

return Ok(employees);

}

}

[Route("api/emp")]

[ApiController]

public class EmployeeController : ControllerBase

{

static List<string> employees = new List<string> { "Alice", "Bob", "Charlie" };

[HttpGet]

public IActionResult Get()

{

return Ok(employees);

}

}

**Web Api using custom model class**

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public int Salary { get; set; }

public bool Permanent { get; set; }

public Department Department { get; set; }

public List<Skill> Skills { get; set; }

public DateTime DateOfBirth { get; set; }

}

READ AND WRITE OPERATIONS

using Microsoft.AspNetCore.Mvc;

using WebApiDemo.Models; // adjust the namespace as per your project

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

[ApiController]

[AllowAnonymous]

public class EmployeeController : ControllerBase

{

private List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 60000,

Permanent = true,

DateOfBirth = new DateTime(1990, 1, 1),

Department = new Department { Id = 1, Name = "IT" },

Skills = new List<Skill>

{

new Skill { Id = 1, Name = "C#" },

new Skill { Id = 2, Name = "SQL" }

}

}

};

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> Get()

{

// throw new Exception("Simulated error for testing custom exception filter.");

return Ok(\_employees);

}

[HttpPost]

public IActionResult Post([FromBody] Employee emp)

{

\_employees.Add(emp);

return Ok("Employee added");

}

[HttpPut("{id}")]

public IActionResult Put(int id, [FromBody] Employee updatedEmp)

{

var emp = \_employees.FirstOrDefault(e => e.Id == id);

if (emp == null) return NotFound();

emp.Name = updatedEmp.Name;

emp.Salary = updatedEmp.Salary;

emp.Permanent = updatedEmp.Permanent;

emp.Department = updatedEmp.Department;

emp.Skills = updatedEmp.Skills;

emp.DateOfBirth = updatedEmp.DateOfBirth;

return Ok("Updated");

}

}

Authorization

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var headers = context.HttpContext.Request.Headers;

if (!headers.ContainsKey("Authorization"))

{

context.Result = new BadRequestObjectResult("Invalid request - No Auth token");

return;

}

var authHeader = headers["Authorization"].ToString();

if (!authHeader.Contains("Bearer"))

{

context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");

}

}

}

EXECEPTION FILTER

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System.IO;

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

string logPath = "C:\\Logs\\exception\_log.txt";

Directory.CreateDirectory(Path.GetDirectoryName(logPath));

File.AppendAllText(logPath, $"[{DateTime.Now}] {exception.Message} {Environment.NewLine}");

context.Result = new ObjectResult("An internal server error occurred")

{

StatusCode = 500

};

}

}

CREATE AND UPDATE

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

using System.Linq;

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

[ApiController]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 60000,

Permanent = true,

DateOfBirth = new DateTime(1990, 1, 1),

Department = new Department { Id = 1, Name = "IT" },

Skills = new List<Skill> {

new Skill { Id = 1, Name = "C#" }

}

},

new Employee

{

Id = 2,

Name = "Bob",

Salary = 55000,

Permanent = false,

DateOfBirth = new DateTime(1985, 5, 15),

Department = new Department { Id = 2, Name = "HR" },

Skills = new List<Skill> {

new Skill { Id = 2, Name = "Recruitment" }

}

}

};

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> UpdateEmployee(int id, [FromBody] Employee updatedEmp)

{

if (id <= 0)

return BadRequest("Invalid employee id");

var existingEmp = \_employees.FirstOrDefault(e => e.Id == id);

if (existingEmp == null)

return BadRequest("Invalid employee id");

// Update the existing employee

existingEmp.Name = updatedEmp.Name;

existingEmp.Salary = updatedEmp.Salary;

existingEmp.Permanent = updatedEmp.Permanent;

existingEmp.Department = updatedEmp.Department;

existingEmp.Skills = updatedEmp.Skills;

existingEmp.DateOfBirth = updatedEmp.DateOfBirth;

return Ok(existingEmp);

}

}

TESTING

{

"id": 1,

"name": "Alice Updated",

"salary": 70000,

"permanent": true,

"department": {

"id": 1,

"name": "IT"

},

"skills": [

{ "id": 1, "name": "C#" },

{ "id": 3, "name": "Angular" }

],

"dateOfBirth": "1990-01-01T00:00:00"

}

AWT authentication

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

string securityKey = "mysuperdupersecret";

var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

CREATE CONTROLLER

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

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

[ApiController]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("token")]

public IActionResult GetToken()

{

var token = GenerateJSONWebToken(101, "Admin");

return Ok(new { token });

}

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

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

var claims = new List<Claim>

{

new Claim(ClaimTypes.Role, userRole),

new Claim("UserId", userId.ToString())

};

var token = new JwtSecurityToken(

issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(10), // Change to 2 for expiration test

signingCredentials: credentials

);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

AUTHORIZE ATTRIBUTE

using Microsoft.AspNetCore.Authorization;

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

[ApiController]

[Authorize] // Secures entire controller

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

return Ok("Access granted to GET Employee list.");

}

}

USE ROLES IN AUTHORIZATION

[Authorize(Roles = "POC")]

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

return Ok("This requires POC role");

}

}

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

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

return Ok("Access allowed for Admin or POC roles");

}

}

**1. Introduction to Kafka**

**Apache Kafka** is a distributed event streaming platform used for high-performance data pipelines, streaming analytics, and real-time data integration.

It allows publishing and subscribing to streams of records (like a messaging queue but more robust).

Kafka is horizontally scalable, fault-tolerant, and highly available.

**2. Kafka Architecture**

**Producer**: Sends data to Kafka topics.

**Consumer**: Reads data from topics.

**Broker**: Kafka server that stores and serves messages.

**Zookeeper**: Manages Kafka brokers (metadata, cluster coordination).

**3. Topics**

Logical channel to which records are sent.

Think of a topic like a database table or log file.

**4. Partitions**

A topic is split into **partitions** for scalability and parallel processing.

Each partition is an **ordered**, **immutable sequence** of records.

**5. Brokers**

Kafka runs on one or more servers called **brokers**.

Each broker holds a subset of topic partitions.

**6. Kafka Plugin in .NET**

Use **Confluent.Kafka** NuGet package for Kafka integration in .NET apps.

Supports .NET Core, .NET Framework, and Windows Forms.

**7. Installation of Kafka**

**Pre-requisites**: Java 8+, Zookeeper

**Steps**:

Download Kafka from: <https://kafka.apache.org/downloads>

Extract and navigate to the folder.

Start Zookeeper:

bash

CopyEdit

bin/zookeeper-server-start.sh config/zookeeper.properties

Start Kafka server:

bash

CopyEdit

bin/kafka-server-start.sh config/server.properties

**8. Basics of Zookeeper**

Manages **Kafka metadata** (brokers, topics, partitions).

Needed for **leader election** and **configuration management**.

Not needed in Kafka 2.8+ (KRaft mode removes dependency).

**9. Demo Plan**

**Producer app**: Publishes messages to a topic.

**Consumer app**: Subscribes to the topic and prints messages in real time.

PRODUCER

// Program.cs

using Confluent.Kafka;

class Program

{

static async Task Main()

{

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

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

Console.WriteLine("Enter messages (type 'exit' to quit):");

string message;

while ((message = Console.ReadLine()) != "exit")

{

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

Console.WriteLine("Message sent.");

}

}

}

CONSUMER

// Program.cs

using Confluent.Kafka;

class Program

{

static async Task Main()

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-group",

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

Console.WriteLine("Listening to chat messages...");

while (true)

{

var cr = consumer.Consume();

Console.WriteLine($"Message received: {cr.Message.Value}");

}

}

}

PRODUCER UI

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

{

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

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

string message = txtMessage.Text;

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

MessageBox.Show("Message sent");

}

CONSUMER UI

private void btnStart\_Click(object sender, EventArgs e)

{

Task.Run(() =>

{

var config = new ConsumerConfig

{

BootstrapServers = "localhost:9092",

GroupId = "chat-group-ui",

AutoOffsetReset = AutoOffsetReset.Earliest

};

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

consumer.Subscribe("chat-topic");

while (true)

{

var cr = consumer.Consume();

this.Invoke(new Action(() => lstMessages.Items.Add(cr.Message.Value)));

}

});

}