**WEEK 4 ASSIGNMENT**

# **Web API Hands-On Assignment**

## **Lab 1: First Web API Using .NET Core**

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

[ApiController]

public class ValuesController : ControllerBase

{

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

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

}

[HttpGet("{id}")]

public ActionResult<string> Get(int id)

{

return "value";

}

[HttpPost]

public void Post([FromBody] string value)

{

}

[HttpPut("{id}")]

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

{

}

[HttpDelete("{id}")]

public void Delete(int id)

{

}

}

## **Lab 2: Web API with Swagger**

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

services.AddControllers();

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

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

Contact = new OpenApiContact { Name = "John Doe", Email = "john@xyzmail.com", Url = new Uri("https://www.example.com") },

License = new OpenApiLicense { Name = "License Terms", Url = new Uri("https://www.example.com") }

});

});

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

app.UseRouting();

app.UseEndpoints(endpoints => endpoints.MapControllers());

}

}

Navigate to https://localhost:[port]/swagger, select GET method, click "Try it out", and "Execute" to verify response.

## **Lab 3: Web API Using Custom Model Class**

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; }

}

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

public class Skill

{

public string Name { get; set; }

}

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

[ApiController]

public class EmployeeController : ControllerBase

{

private readonly List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

[HttpGet]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

return Ok(\_employees);

}

[HttpPost]

[AllowAnonymous]

public ActionResult<Employee> Post([FromBody] Employee employee)

{

if (employee == null || employee.Id <= 0)

return BadRequest("Invalid employee data");

\_employees.Add(employee);

return CreatedAtAction(nameof(GetStandard), new { id = employee.Id }, employee);

}

[HttpPut("{id}")]

[AllowAnonymous]

public ActionResult<Employee> Put(int id, [FromBody] Employee employee)

{

if (id <= 0 || employee == null || employee.Id != id)

return BadRequest("Invalid employee id");

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

if (existing == null)

return BadRequest("Invalid employee id");

existing.Name = employee.Name;

existing.Salary = employee.Salary;

existing.Permanent = employee.Permanent;

existing.Department = employee.Department;

existing.Skills = employee.Skills;

existing.DateOfBirth = employee.DateOfBirth;

return Ok(existing);

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee { Id = 1, Name = "John Doe", Salary = 50000, Permanent = true, Department = new Department { Id = 1, Name = "IT" }, Skills = new List<Skill> { new Skill { Name = "C#" } }, DateOfBirth = new DateTime(1990, 1, 1) },

new Employee { Id = 2, Name = "Jane Smith", Salary = 60000, Permanent = false, Department = new Department { Id = 2, Name = "HR" }, Skills = new List<Skill> { new Skill { Name = "Java" } }, DateOfBirth = new DateTime(1985, 5, 10) }

};

}

}

## **Lab 4: Custom Action Filter for Authorization**

namespace YourNamespace.Filters

{

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

var authHeader = context.HttpContext.Request.Headers["Authorization"].ToString();

if (string.IsNullOrEmpty(authHeader))

{

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

return;

}

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

{

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

return;

}

base.OnActionExecuting(context);

}

}

}

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

[ApiController]

[CustomAuthFilter]

public class EmployeeController : ControllerBase

{

// Existing code from Lab 3

}

## **Lab 5: Custom Exception Filter**

Install Microsoft.AspNetCore.Mvc.WebApiCompatShim.

namespace YourNamespace.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

var exception = context.Exception;

var errorMessage = $"Error: {exception.Message}\nStackTrace: {exception.StackTrace}";

File.WriteAllText("error\_log.txt", errorMessage);

context.Result = new ObjectResult("Internal Server Error")

{

StatusCode = 500

};

context.ExceptionHandled = true;

}

}

}

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

[ApiController]

[CustomAuthFilter]

[CustomExceptionFilter]

public class EmployeeController : ControllerBase

{

[HttpGet]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status500InternalServerError)]

public ActionResult<List<Employee>> GetStandard()

{

throw new Exception("Test exception");

return Ok(\_employees);

}

// Other methods from Lab 3

}

## **Lab 6: Web API CRUD Operations**

[HttpPut("{id}")]

[AllowAnonymous]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status400BadRequest)]

public ActionResult<Employee> Put(int id, [FromBody] Employee employee)

{

if (id <= 0 || employee == null || employee.Id != id)

return BadRequest("Invalid employee id");

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

if (existing == null)

return BadRequest("Invalid employee id");

existing.Name = employee.Name;

existing.Salary = employee.Salary;

existing.Permanent = employee.Permanent;

existing.Department = employee.Department;

existing.Skills = employee.Skills;

existing.DateOfBirth = employee.DateOfBirth;

return Ok(existing);

}

Test in Swagger: Use PUT method at api/Employee/{id} with valid/invalid id. Test in Postman: Send PUT to https://localhost:[port]/api/Employee/1 with JSON body.

## **Lab 7: JSON Web Token (JWT) Authentication**

public class Startup

{

public void ConfigureServices(IServiceCollection services)

{

string securityKey = "mysuperdupersecret";

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

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

services.AddControllers();

}

public void Configure(IApplicationBuilder app, IWebHostEnvironment env)

{

app.UseRouting();

app.UseAuthentication();

app.UseAuthorization();

app.UseEndpoints(endpoints => endpoints.MapControllers());

}

}

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

[ApiController]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet]

public ActionResult<string> Get()

{

return GenerateJSONWebToken(1, "Admin");

}

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),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

## **Lab 8: JWT Testing**

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

[ApiController]

[Authorize]

public class EmployeeController : ControllerBase

{

// Existing code from Lab 3

}

In Postman, generate JWT from api/Auth, use in Authorization: Bearer <JWT> for api/Employee GET. Test invalid/missing token for 401 Unauthorized.

## **Lab 9: JWT Expiration**

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(2),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

Generate JWT, wait 2 minutes, test api/Employee GET in Postman for 401 Unauthorized.

## **Lab 10: Role-Based Authorization**

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

[ApiController]

[Authorize(Roles = "POC")]

public class EmployeeController : ControllerBase

{

// Existing code

}

Test with "Admin" JWT in Postman for 401 Unauthorized. Update to [Authorize(Roles = "Admin,POC")] and test for 200 OK.

## **Lab 11: Kafka Chat Application (Console)**

Install Kafka, Zookeeper, and Confluent.Kafka.

// Producer

using Confluent.Kafka;

using System;

class Program

{

static async Task Main(string[] args)

{

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

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

string topic = "chat-topic";

while (true)

{

Console.Write("Enter message: ");

string message = Console.ReadLine();

if (string.IsNullOrEmpty(message)) break;

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

Console.WriteLine($"Delivered '{result.Value}' to {result.TopicPartitionOffset}");

}

}

}

// Consumer

using Confluent.Kafka;

using System;

class Program

{

static void Main(string[] args)

{

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");

while (true)

{

var consumeResult = consumer.Consume();

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

}

}

}

Run Zookeeper: zookeeper-server-start.bat ../../config/zookeeper.properties. Run Kafka: kafka-server-start.bat ../../config/server.properties. Create topic: kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1.

## **Lab 12: Kafka Chat Application (Windows Forms)**

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

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

{

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

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

string message = messageTextBox.Text;

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

messageTextBox.Clear();

}

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

{

Task.Run(() => StartConsumer());

}

private void StartConsumer()

{

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");

while (true)

{

var consumeResult = consumer.Consume();

Invoke((Action)(() => messageListBox.Items.Add(consumeResult.Message.Value)));

}

}

}