WEB API HANDSON

1.

1. Concepts of RESTful Web Services, Web API, and Microservices

REST is an architectural style for designing networked applications. It relies on stateless, client-server communication, typically over HTTP.

Features of REST:

Stateless: Each request from a client contains all information needed; no session is maintained on the server.

Resource-based: Everything is a resource (represented by URLs).

Uses HTTP methods (GET, POST, PUT, DELETE).

Response Format: Not restricted to XML. Supports JSON, plain text, etc.

Client-Server: Separation of concerns.

Cacheable: Responses can be cached for better performance.

2. HttpRequest & HttpResponse

HttpRequest: Carries the client's request to the server (URL, headers, method, body).

HttpResponse: The server’s response to the request (status code, headers, body).

3. Types of HTTP Action Verbs

| Verb | Purpose |
| --- | --- |
| HttpGet | Retrieve data |
| HttpPost | Submit data (create) |
| HttpPut | Update data |
| HttpDelete | Delete data |

4. HTTP Status Codes Used in Web API

| Status Code | Meaning |
| --- | --- |
| 200 OK | Successful request |
| 400 BadRequest | Malformed request |
| 401 Unauthorized | Authentication needed |
| 500 InternalServerError | Server-side issue |

Demonstration: First Web API in .NET Core

ValuesController.cs:

[ApiController]

[Route("[controller]")]

public class ValuesController : ControllerBase

{

[HttpGet]

public IEnumerable<string> Get() => new string[] { "value1", "value2" };

[HttpPost]

public void Post([FromBody] string value) { }

[HttpPut("{id}")]

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

[HttpDelete("{id}")]

public void Delete(int id) { }

}

WebApiConfig.cs:

public static class WebApiConfig

{

public static void Register(HttpConfiguration config)

{

config.MapHttpAttributeRoutes();

config.Routes.MapHttpRoute(

name: "DefaultApi",

routeTemplate: "api/{controller}/{id}",

defaults: new { id = RouteParameter.Optional }

);

}

}

2.

Add Swagger to Your .NET Core Web API

Step 1: Install Swagger NuGet Package

dotnet add package Swashbuckle.AspNetCore

Step 2: Modify Startup.cs

using Microsoft.OpenApi.Models;

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/terms"),

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")

}

});

});

}

Step 3: Run the App

<https://localhost:<port>/swagger>

3.

Create a Custom Model and EmployeeController

Department.cs:

public class Department

{

public int Id { get; set; }

public string Name { get; set; }

}

Skill.cs:

public class Skill

{

public int Id { get; set; }

public string Name { get; set; }

}

Employee.cs:

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; }

}

EmployeeController.cs:

using Microsoft.AspNetCore.Mvc;

using YourAppNamespace.Models;

[ApiController]

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

[AllowAnonymous]

public class EmployeeController : ControllerBase

{

private readonly List<Employee> \_employees;

public EmployeeController()

{

\_employees = GetStandardEmployeeList();

}

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "John Doe",

Salary = 50000,

Permanent = true,

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

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

Skills = new List<Skill>

{

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

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

}

}

};

}

[HttpGet]

[ProducesResponseType(typeof(List<Employee>), 200)]

[ProducesResponseType(500)]

public ActionResult<List<Employee>> Get()

{

throw new Exception("Simulated server error"); // For exception testing

return Ok(\_employees);

}

[HttpPost]

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

{

return Ok(emp); // Echo back the posted employee

}

[HttpPut("{id}")]

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

{

return Ok($"Updated employee with ID = {id}");

}

}

CustomAuthFilter.cs:

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

public class CustomAuthFilter : ActionFilterAttribute

{

public override void OnActionExecuting(ActionExecutingContext context)

{

if (!context.HttpContext.Request.Headers.TryGetValue("Authorization", out var authHeader))

{

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

return;

}

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

{

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

}

}

}

CustomExceptionFilter.cs:

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;

File.AppendAllText("exceptions.log", $"[{DateTime.Now}] {exception.Message}\n");

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

{

StatusCode = 500

};

context.ExceptionHandled = true;

}

}

Run your app

<https://localhost:<port>/swagger>

4.

Implement the PUT (Update) Operation

[HttpPut("{id}")]

[ProducesResponseType(typeof(Employee), 200)]

[ProducesResponseType(400)]

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

{

if (id <= 0)

{

return BadRequest("Invalid employee id");

}

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

if (existingEmployee == null)

{

return BadRequest("Invalid employee id");

}

// Update values

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

return Ok(existingEmployee);

}

Test the PUT API

<https://localhost:<port>/swagger>

5.

Enable CORS in Web API

dotnet add package Microsoft.AspNetCore.Cors

Startup.cs

services.AddCors(options =>

{

options.AddPolicy("AllowAll",

builder => builder

.AllowAnyOrigin()

.AllowAnyMethod()

.AllowAnyHeader());

});

ConfigureServices

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

};

});

AuthController.cs

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

[ApiController]

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

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet("generate")]

public ActionResult<string> GenerateToken()

{

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

return Ok(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), // Modify to 2 for expiry test

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

EmployeeController

[ApiController]

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

[Authorize(Roles = "Admin,POC")] // Change this to test different roles

public class EmployeeController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

return Ok("You are authorized!");

}

}

Testing with Postman

<https://localhost:<port>/api/auth/generate>