# WebApi Hands-on Exercises

**QUESTION 1)**

**Objectives:**

* Explain the concept of RESTful web service, Web API & Microservice
  + Features of REST architecture - Representational State Transfer, Stateless, Messages, Concept of Microservice, Difference between WebService & WebAPI, Not restricted to send XML as response
* Explain what is HttpRequest & HttpResponse
* List the types of Action Verbs
  + HttpGet, HttpPost, HttpPut, HttpDelete - Meaning of action verbs and how that should be declared as attributes for Web API
* List the types of HttpStatusCodes used in WebAPI
  + Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types
* Demonstrate creation of a simple WebAPI - With Read, Write actions
  + Structure of a web api - Controller & its inheritance from ApiController, Action verbs, Action method
* Explain the types of Configuration files of WebAPI
  + Startup.cs with depdency injection, appSettings.json, launchSettings.json, Explain Route.config & WebAPI.config in .Net 4.5 framework

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.

### **Ans) **1. RESTful Web Service:****

A RESTful Web Service is a web service that follows the REST (Representational State Transfer) architectural style. It allows communication between systems using standard HTTP methods like GET, POST, PUT, and DELETE. RESTful services use URIs to access resources and typically exchange data in JSON or XML format.

### ****2. Features of REST Architecture:****

**Representational State Transfer (REST):** REST uses resource-based URLs and transfers data in representations like JSON or XML.

**Stateless:** Each request from the client to the server must contain all necessary information. The server does not maintain any session state.

**Messages:** Communication is done via HTTP requests and responses.

**Uniform Interface:** Standard HTTP verbs (GET, POST, PUT, DELETE) are used to perform operations on resources.

**Scalability and Simplicity:** REST is easy to implement and allows services to scale easily.

**Not restricted to XML:** Unlike SOAP, REST can return responses in multiple formats such as JSON, XML, plain text, etc.

### ****3. Web API:****

A Web API (Web Application Programming Interface) is a framework that allows access to the functionalities of an application via the web. It allows communication between different software systems using HTTP. RESTful Web APIs are commonly used for web, mobile, and cloud applications and usually return data in JSON format.

### ****4. Microservice:****

A microservice is a small, independent service that performs a specific business function. Microservices architecture divides an application into multiple loosely coupled services, which can be developed, deployed, and scaled independently. These services communicate with each other using REST APIs or messaging protocols.

### ****5. Difference between Web Service and Web API:****

| **Feature** | **Web Service** | **Web API** |
| --- | --- | --- |
| Protocol | Supports HTTP, SOAP, XML-RPC | Works on HTTP/HTTPS only |
| Data Format | XML only | JSON, XML, plain text, etc. |
| Performance | Slower due to SOAP overhead | Faster and lightweight |
| Accessibility | Requires more setup/configuration | Easy to access via URLs |
| Usage | Used in enterprise systems | Used in modern apps (web, mobile, IoT) |

### ****Conclusion:****

RESTful Web Services, Web APIs, and Microservices are modern architectural approaches that enable efficient, scalable, and interoperable software systems. REST simplifies communication using HTTP, Web APIs provide access to app features via the web, and Microservices enhance flexibility by dividing the system into smaller, focused services.

**First Web Api using .Net core**

**Ans) Code:-**

using Microsoft.AspNetCore.Mvc;

// For more information on enabling Web API for empty projects, visit https://go.microsoft.com/fwlink/?LinkID=397860

namespace AspCoreRestApi\_Wk4.Controllers

{

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

[ApiController]

public class ValuesController : ControllerBase

{

// GET: api/<ValuesController>

[HttpGet]

public IEnumerable<string> Get()

{

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

}

// GET api/<ValuesController>/5

[HttpGet("{id}")]

public string Get(int id)

{

return "value";

}

// POST api/<ValuesController>

[HttpPost]

public void Post([FromBody] string value)

{

}

// PUT api/<ValuesController>/5

[HttpPut("{id}")]

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

{

}

// DELETE api/<ValuesController>/5

[HttpDelete("{id}")]

public void Delete(int id)

{

}

}

}

**Ouput:-**



**QUESTION 2)**

**Objectives:**

* 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

**ANS)**

**CODE:-(Objective 1)**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// ✅ Swagger config here

builder.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://example.com")

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

// Configure the HTTP request pipeline.

// ✅ Add Swagger middleware

app.UseSwagger();

app.UseSwaggerUI(c =>

{

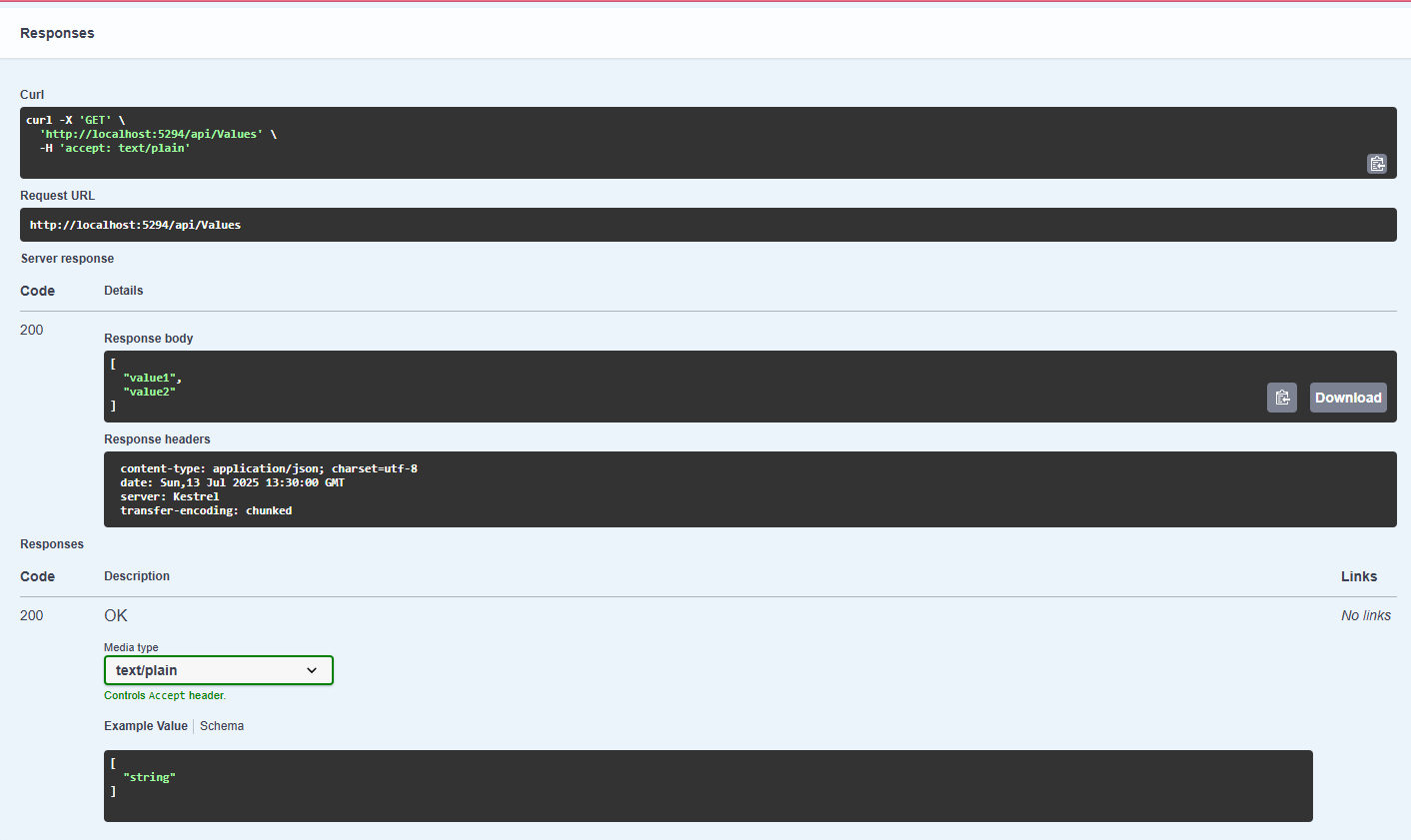
c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

});

app.UseAuthorization();

app.MapControllers();

app.Run();



**CODE:-(Objective 2)**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace AspCoreRestApi\_Wk4.Controllers

{

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

[ApiController]

public class EmployeeController : ControllerBase

{

[HttpGet]

public IEnumerable<string> Get()

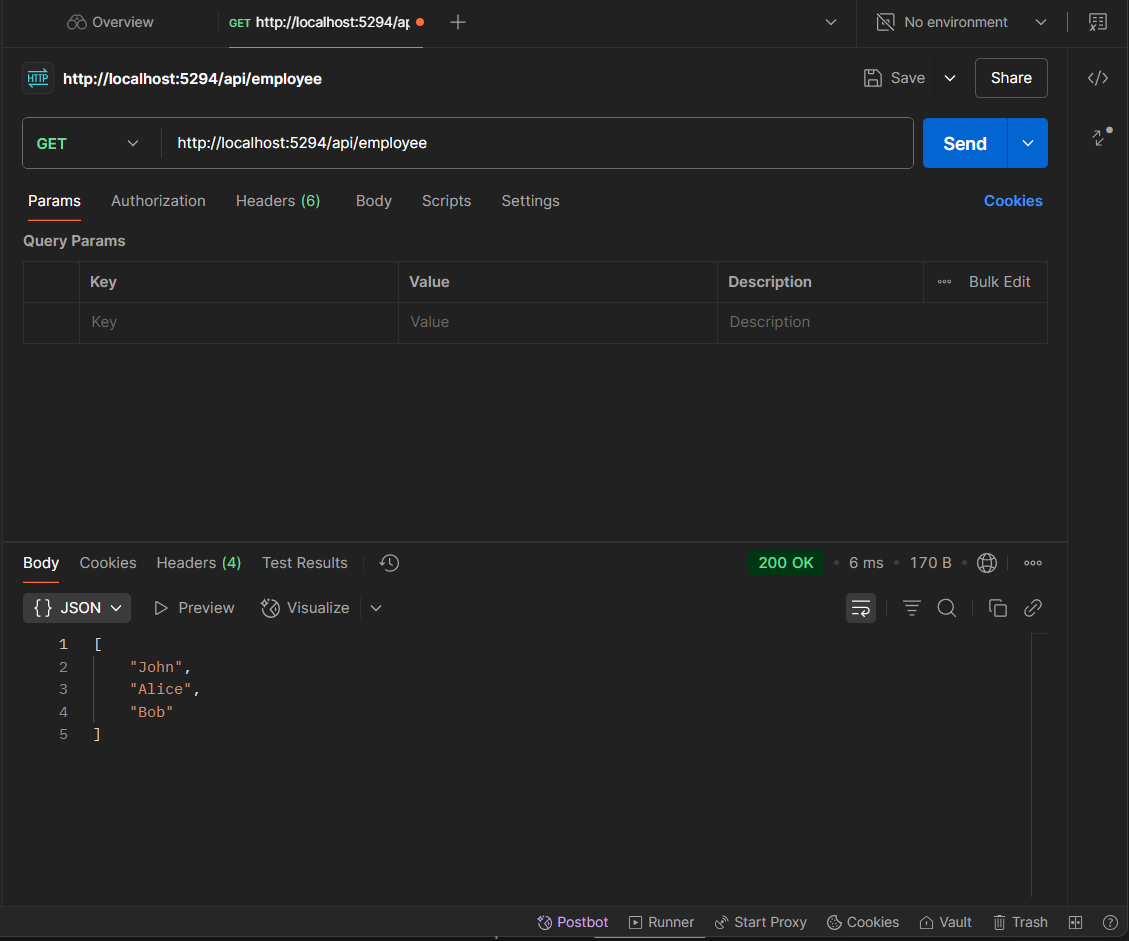
{

return new string[] { "John", "Alice", "Bob" };

}

}

}



**CODE:-(Objective 3)**

using Microsoft.AspNetCore.Mvc;

using System.Collections.Generic;

namespace AspCoreRestApi\_Wk4.Controllers

{

[Route("api/emp")]

[ApiController]

public class EmployeeController : ControllerBase

{

[HttpGet]

public IEnumerable<string> Get()

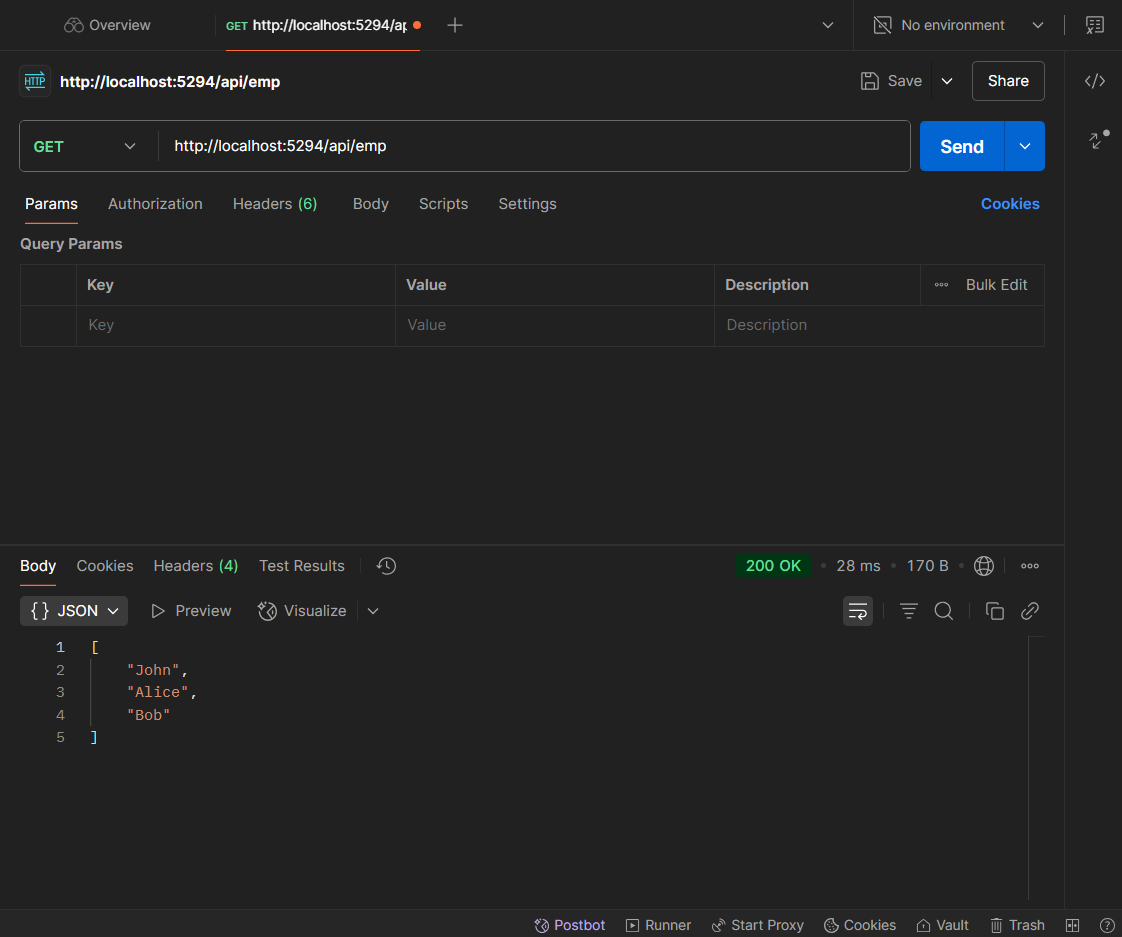
{

return new string[] { "John", "Alice", "Bob" };

}

}

}

****

**QUESTION 3)**

**Objectives:**

* Demonstrate creation of an Action method to return list of custom class entity
  + Model class creation, Use AllowAnonymous attribute, Use HttpGet action method
* Explain the usage of FromBody attribute
  + Read the model object from request, other than the query string parameter
* Demonstrate Custom filter
  + Usage of ActionFilterAttribute, OnActionExecuting method to intercept the request, Create filter for Custom exception - Need to install Microsoft.AspNetCore.Mvc.WebApiCompatShim package

**ANS)**

**CODE:-(Objective 1)**

**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; }

}

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; }

}

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;

[Route("api/emp")]

[ApiController]

public class EmployeeController : ControllerBase

{

// Create a few sample records

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Asish Sarangi",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill>

{

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

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

},

DateOfBirth = new DateTime(2004, 5, 16)

}

};

}

// GET api/emp

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

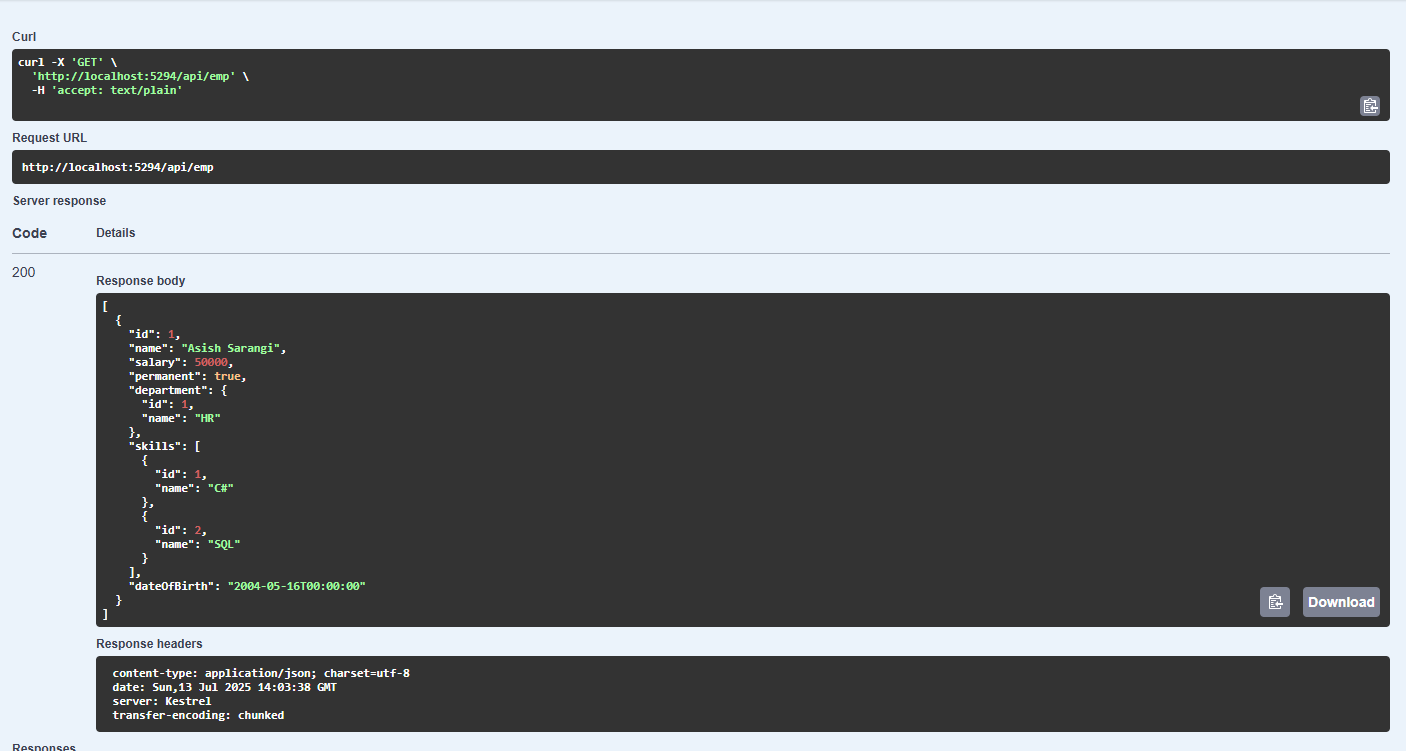
public ActionResult<List<Employee>> GetStandard()

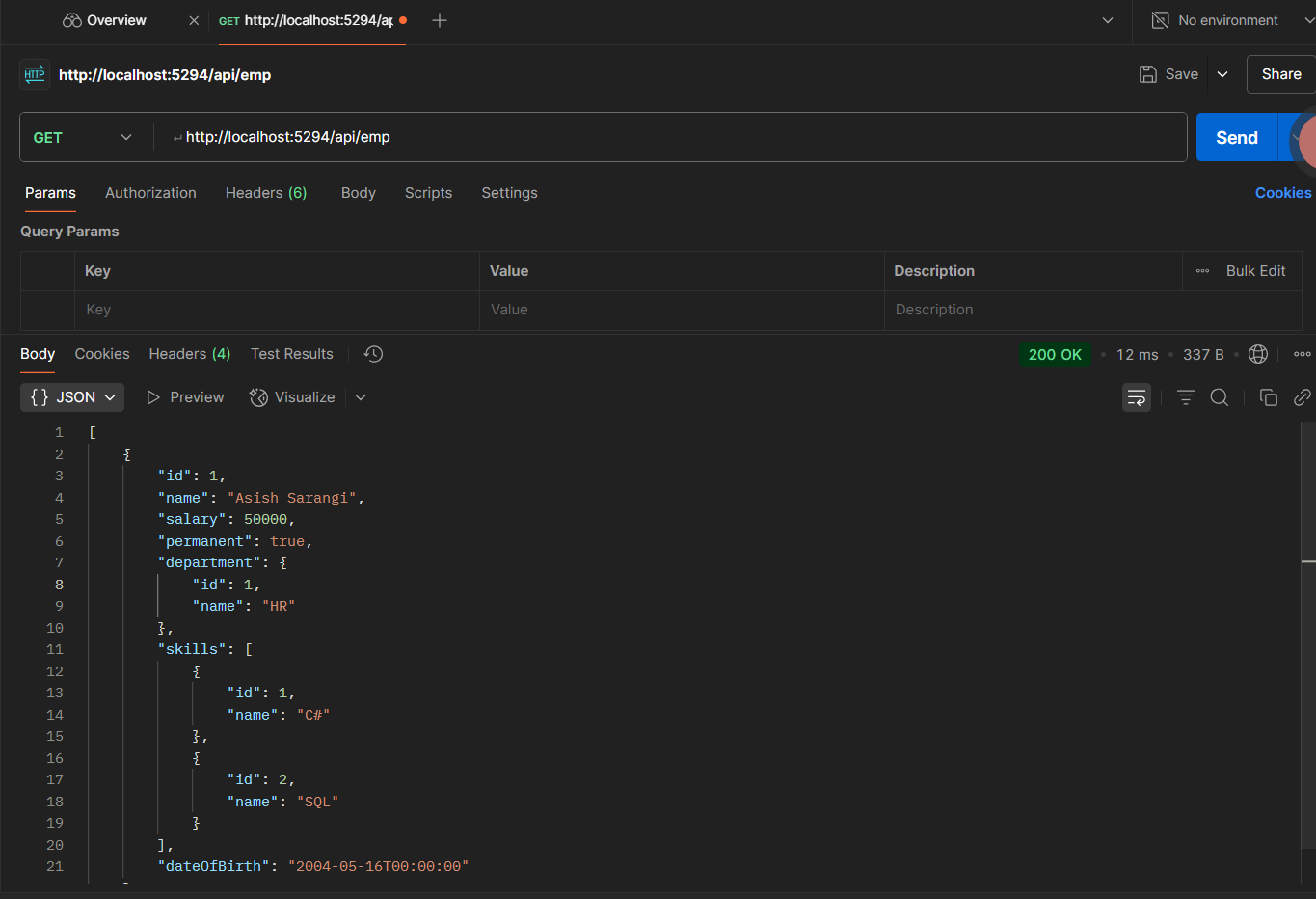
{

return GetStandardEmployeeList();

}

}





**CODE:-(Objective 2)**

**CustomAuthFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

namespace AspCoreRestApi\_Wk4.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 token = headers["Authorization"].ToString();

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

{

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

return;

}

base.OnActionExecuting(context);

}

}

}

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;

using AspCoreRestApi\_Wk4.Filters;

using System;

using System.Collections.Generic;

[Route("api/emp")]

[ApiController]

[CustomAuthFilter] // ✅ corrected name

public class EmployeeController : ControllerBase

{

private List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Asish Sarangi",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill>

{

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

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

},

DateOfBirth = new DateTime(2004, 5, 16)

}

};

}

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

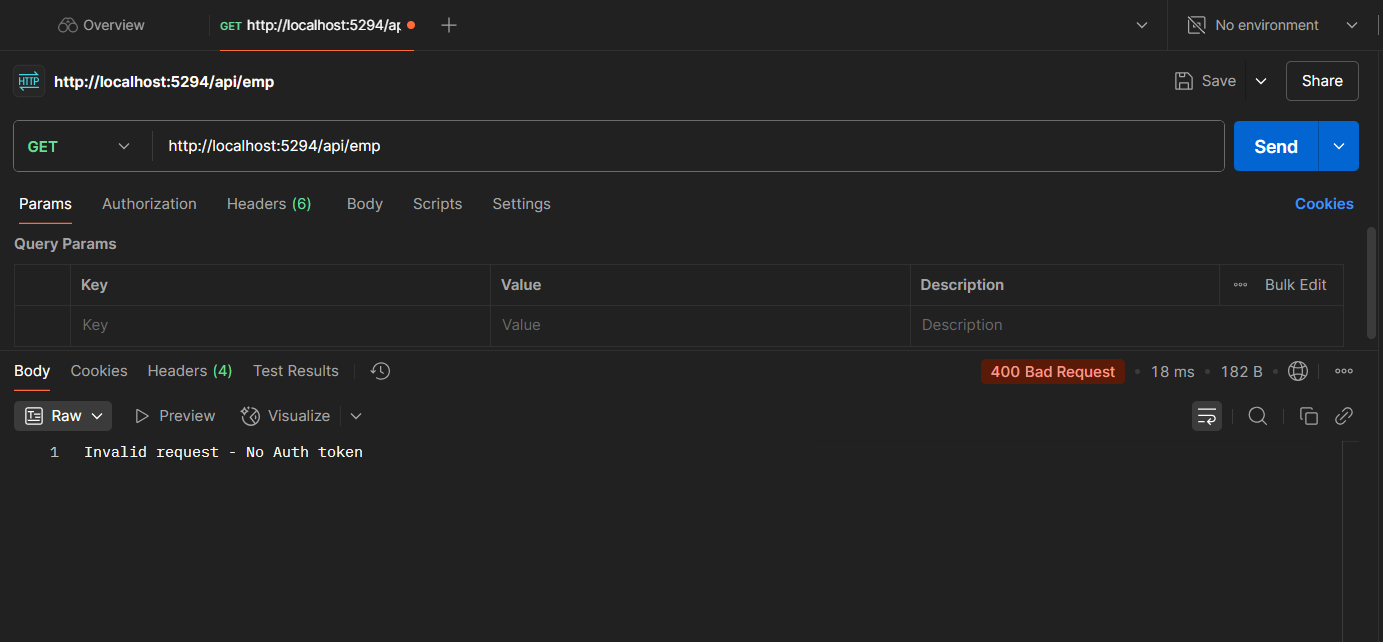
{

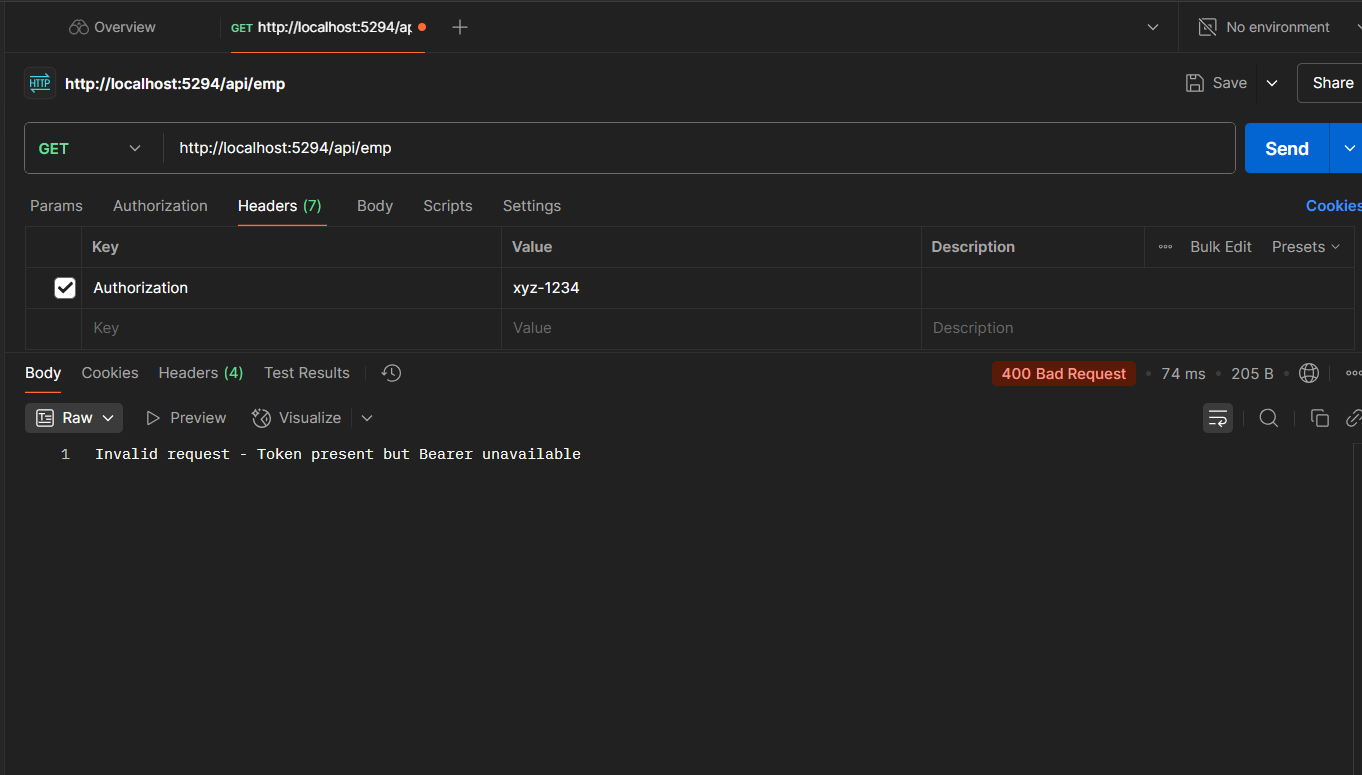
return GetStandardEmployeeList();

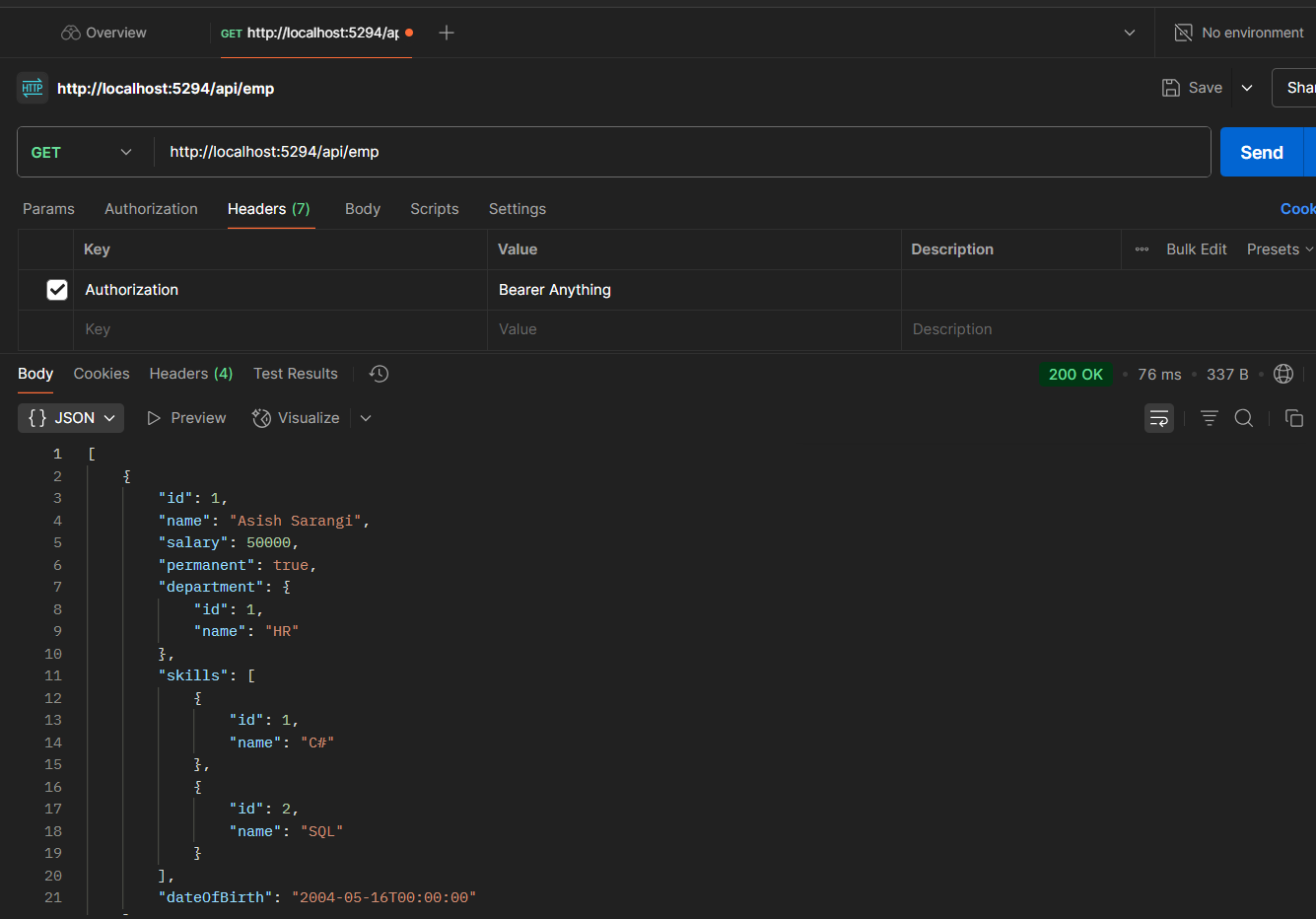
}

}

**OUTPUT:-**

****





**CODE:-(Objective 3)**

**CustomerExceptionFilter.cs**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Mvc.Filters;

using System.IO;

using System;

namespace AspCoreRestApi\_Wk4.Filters

{

public class CustomExceptionFilter : IExceptionFilter

{

public void OnException(ExceptionContext context)

{

string filePath = "exceptions.txt";

File.AppendAllText(filePath,

$"[{DateTime.Now}] {context.Exception.Message} {Environment.NewLine}");

context.Result = new ObjectResult("An error occurred. Please try again later.")

{

StatusCode = 500

};

}

}

}

**Program.cs**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers(options =>

{

options.Filters.Add<AspCoreRestApi\_Wk4.Filters.CustomExceptionFilter>(); // 👈 Add this line

});

builder.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://example.com")

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

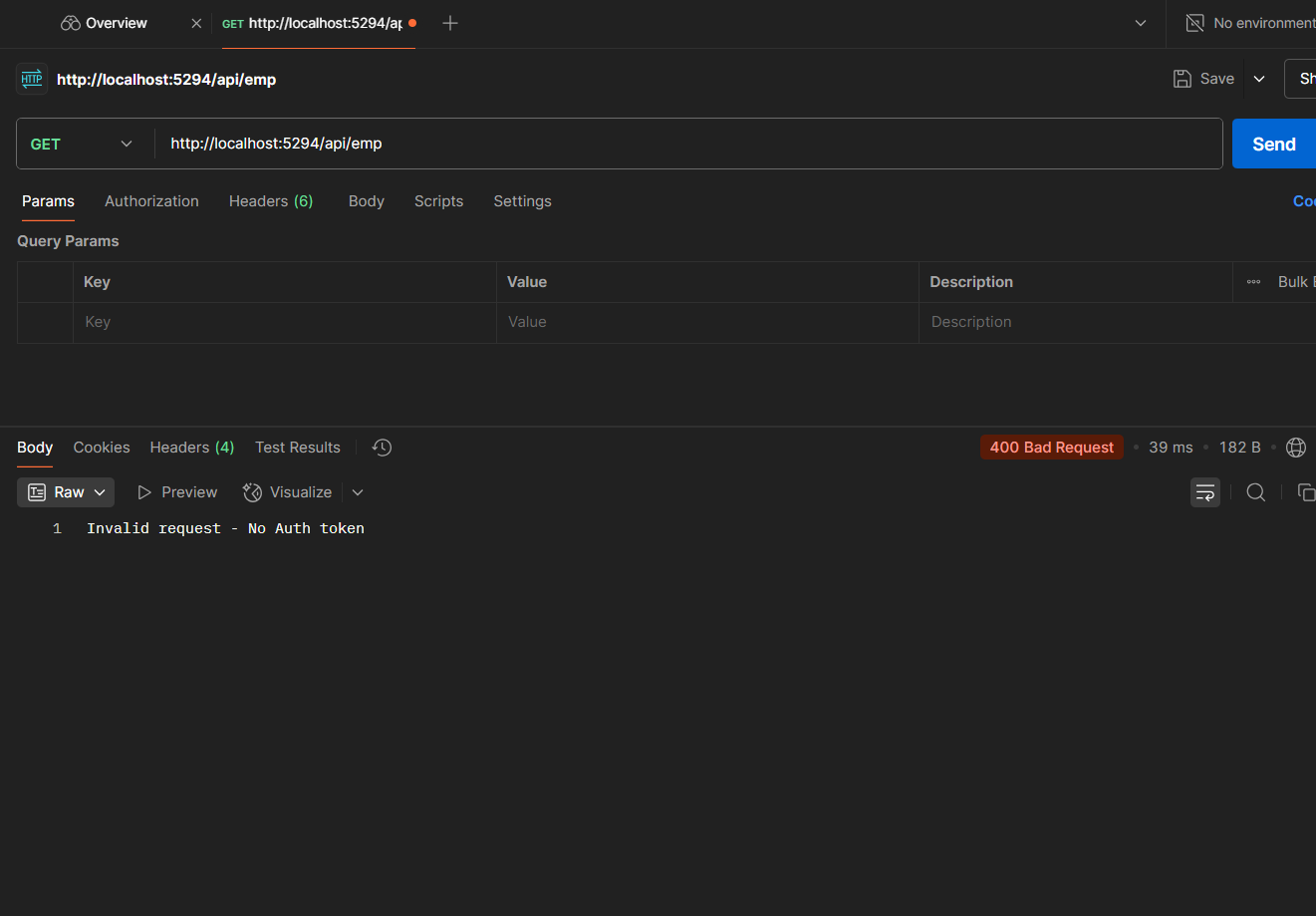
});

app.UseAuthorization();

app.MapControllers();

app.Run();

**OUPUT:-**

****

**QUESTION 4)**

**Objectives:**

* Demonstrate creation of an Action method to perform data create, update & delete operation
  + Use FromBody attribute, extract data to custom model class using FromBody attribute, use hardcoded data to update & delete data, Use Swagger and POSTMAN to test

**ANS)**

**CODE:-**

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;

using AspCoreRestApi\_Wk4.Filters;

using System;

using System.Collections.Generic;

namespace AspCoreRestApi\_Wk4.Controllers

{

[Route("api/emp")]

[ApiController]

[CustomAuthFilter]

public class EmployeeController : ControllerBase

{

// 🔹 Hardcoded static employee list

private static List<Employee> employeeList = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Asish Sarangi",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill>

{

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

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

},

DateOfBirth = new DateTime(2004, 5, 16)

}

};

// 🔹 GET: api/emp

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

return Ok(employeeList);

}

// 🔹 PUT: api/emp/{id}

[HttpPut("{id}")]

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

{

if (id <= 0)

return BadRequest("Invalid employee id");

var existingEmployee = employeeList.Find(e => e.Id == id);

if (existingEmployee == null)

return BadRequest("Invalid employee id");

// Update fields

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

return Ok(existingEmployee);

}

}

// 🔸 Models below (same file for now)

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

{

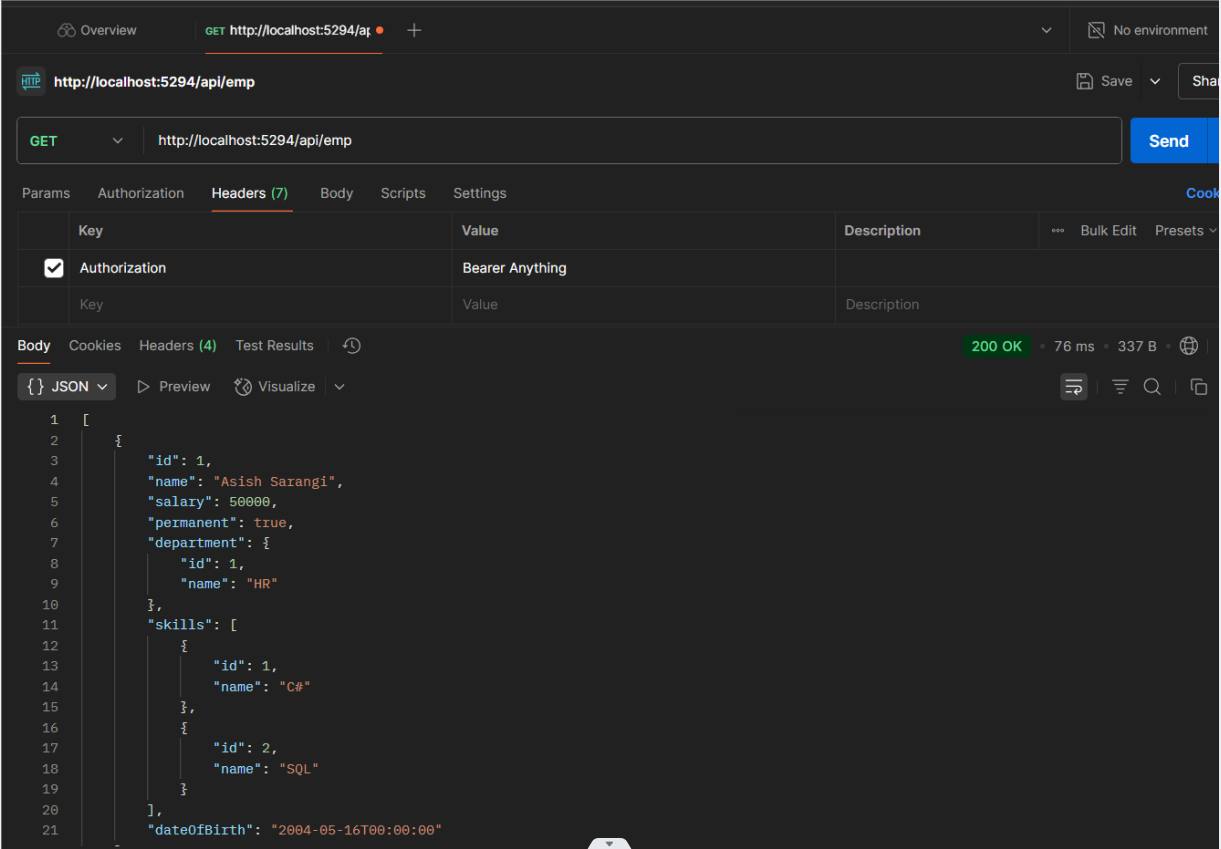
public int Id { get; set; }

public string Name { get; set; }

}

}

**OUTPUT:-**



**Question 5**

**Objectives:**

* Explain CORS enablement for Web API access for local application
  + What is CORS?, How to enable CORS thru Startup.cs, Install Cors nuget package to Web API application
* Demonstrate security in WebAPI
  + Bearer and Jwt token authentication, Use Authorize attribute & send roles in Jwt token, Setting in Startup.cs for AddAuthentication and AddJwtBearer with validation attributes, UseAuthentication, AllowAnonymous to AuthController to generate token, Claims

**Ans) (Objective 1):-**

**CORS (Cross-Origin Resource Sharing)** is a security feature that allows a Web API to be accessed from a different domain or port, such as a local frontend application running on a separate server. To enable CORS in an ASP.NET Core Web API, the Microsoft.AspNetCore.Cors NuGet package is installed first. Then, a policy is added in Program.cs using builder.Services.AddCors(), typically with options to allow any origin, header, and method. Finally, app.UseCors("AllowAll") is added to the middleware pipeline before authorization. This setup ensures that local applications (like React or Angular) can interact with the Web API without browser restrictions.

**CODE:-(Objective 2)**

**Authcontroller.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using Microsoft.IdentityModel.Tokens;

using System.IdentityModel.Tokens.Jwt;

using System.Security.Claims;

using System.Text;

namespace AspCoreRestApi\_Wk4.Controllers

{

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

[ApiController]

[AllowAnonymous]

public class AuthController : ControllerBase

{

[HttpGet]

public IActionResult GetToken()

{

var token = GenerateJSONWebToken(1, "Admin"); // "POC" or "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(2),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

}

}

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using AspCoreRestApi\_Wk4.Filters;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

// 🔹 Configure CORS

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowAll",

builder =>

{

builder.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

// 🔹 Configure JWT Authentication

string securityKey = "mysuperdupersecret";

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

builder.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

};

});

// 🔹 Add Controllers + Exception Filter (Optional if used)

builder.Services.AddControllers(options =>

{

options.Filters.Add<CustomExceptionFilter>();

});

// 🔹 Add Swagger

builder.Services.AddSwaggerGen(c =>

{

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

{

Title = "Swagger Demo",

Version = "v1",

Description = "JWT and CORS Enabled Web API",

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

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

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

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

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

}

});

});

var app = builder.Build();

// 🔹 Middleware Pipeline

app.UseCors("AllowAll"); // 🔸 CORS before Auth

app.UseAuthentication(); // 🔸 JWT Auth

app.UseAuthorization(); // 🔸 Role-based access

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

app.MapControllers();

app.Run();

**EmployeeController.cs**

using Microsoft.AspNetCore.Authorization;

using Microsoft.AspNetCore.Mvc;

using System;

using System.Collections.Generic;

namespace AspCoreRestApi\_Wk4.Controllers

{

[Route("api/emp")]

[ApiController]

[Authorize(Roles = "Admin,POC")] // ✅ Requires token + role

public class EmployeeController : ControllerBase

{

private static List<Employee> employeeList = new List<Employee>

{

new Employee

{

Id = 1,

Name = "Asish Sarangi",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill>

{

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

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

},

DateOfBirth = new DateTime(2004, 5, 16)

}

};

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<List<Employee>> GetStandard()

{

return Ok(employeeList);

}

[HttpPut("{id}")]

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

{

if (id <= 0)

return BadRequest("Invalid employee id");

var existingEmployee = employeeList.Find(e => e.Id == id);

if (existingEmployee == null)

return BadRequest("Invalid employee id");

existingEmployee.Name = updatedEmployee.Name;

existingEmployee.Salary = updatedEmployee.Salary;

existingEmployee.Permanent = updatedEmployee.Permanent;

existingEmployee.Department = updatedEmployee.Department;

existingEmployee.Skills = updatedEmployee.Skills;

existingEmployee.DateOfBirth = updatedEmployee.DateOfBirth;

return Ok(existingEmployee);

}

}

// Model classes

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 int Id { get; set; }

public string Name { get; set; }

}

}

**OUTPUT:-**

