**DN 4.0 Dotnet FSE**

**Name: Sparsh Guha**

**Superset ID:6361106**

**Week 4**

**Question 1:**

**RESTful Web Service**  
REST (Representational State Transfer) is an architectural pattern for designing networked applications, especially web services12.  
Key REST features:  
  
Client-Server Architecture: Separation between client and server2.  
Stateless: Each request contains all info needed; server does not store session state2.  
Uniform Interface: Standardized way to interact with resources (usually via HTTP/HTTPS).  
Cacheable: Responses can be cached for performance.  
Layered System: Architecture can be composed of hierarchical layers.  
Code on Demand (optional): Servers can extend client functionality by transferring executable code2.  
RESTful APIs typically use HTTP verbs (GET, POST, PUT, DELETE) and can return data in multiple formats (JSON, XML, etc.)

**Web API vs. Web Service**  
Web Service: A specialized API, often using SOAP/XML, for system-to-system communication4.  
Web API: A broader concept, often RESTful, supports multiple data formats (JSON, XML), and is simpler and more modern43.  
Web API is not limited to XML—it can return JSON, plain text, or other formats3.

**Microservice**  
An architectural style where an application is composed of small, independent services, each handling a specific business function. Each microservice can expose its own Web API.  
HttpRequest & HttpResponse  
HttpRequest: Represents the incoming request from a client. Contains method (GET, POST, etc.), URI, headers, and body/content5.  
HttpResponse: Represents the server’s reply. Contains status code (e.g., 200 OK), headers, and response body5.  
Action Verbs in Web API  
HttpGet: Retrieves data (read-only)67.  
HttpPost: Creates a new resource.  
HttpPut: Updates an existing resource.  
HttpDelete: Deletes a resource.  
These are declared as attributes above action methods in controllers, e.g., [HttpGet], [HttpPost]78.  
HttpStatusCodes in Web API  
200 OK: Request succeeded.  
400 BadRequest: Client sent an invalid request.  
401 Unauthorized: Authentication required.  
500 InternalServerError: Server encountered an error910.  
In ASP.NET Core, you return these using helper methods like Ok(), BadRequest(), Unauthorized(), etc.

**Input:**

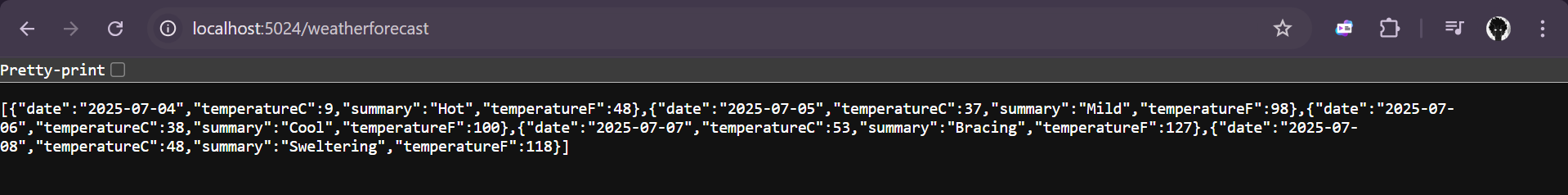
**Program.cs**

var builder = WebApplication.CreateBuilder(args);  
  
  
builder.Services.AddOpenApi();  
  
var app = builder.Build();  
  
  
if (app.Environment.IsDevelopment())  
{  
app.MapOpenApi();  
}  
  
app.UseHttpsRedirection();  
  
var summaries = new[]  
{  
"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"  
};  
  
app.MapGet("/weatherforecast", () =>  
{  
var forecast = Enumerable.Range(1, 5).Select(index =>  
new WeatherForecast  
(  
DateOnly.FromDateTime(DateTime.Now.AddDays(index)),  
Random.Shared.Next(-20, 55),  
summaries[Random.Shared.Next(summaries.Length)]  
))  
.ToArray();  
return forecast;  
})  
.WithName("GetWeatherForecast");  
  
app.Run();  
  
record WeatherForecast(DateOnly Date, int TemperatureC, string? Summary)  
{  
public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);  
}

**Creating a Simple Web API in .NET Core**  
1. Create a New Project (Terminal)  
  
dotnet new webapi -n MyFirstWebApi  
cd MyFirstWebApi  
  
**2. Structure**  
Controllers folder contains controller classes.  
Each controller inherits from ControllerBase (or ApiController in .NET 4.5)11.  
Action methods are decorated with HTTP verb attributes.

1. **Run the API**  
   Run in terminal:  
   dotnet run

**Output:**

****

**QUESTION: 2**

**Web Api using .Net core with Swagger**

**Input:**

**Code:**

**Program.cs**

using Microsoft.AspNetCore.Builder;

using Microsoft.Extensions.DependencyInjection;

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddEndpointsApiExplorer();

builder.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") }

});

});

var app = builder.Build();

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

app.MapControllers();

app.Run();

**Employees.cs**

public class Employee

{

public int Id { get; set; }

public string Name { get; set; }

public string Department { get; set; }

}

## ****EmployeeController.cs****

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Http;

using System.Collections.Generic;

using System.Linq;

[Route("api/emp")]

[ApiController]

public class EmployeeController : ControllerBase

{

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

{

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

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

};

[HttpGet]

[ProducesResponseType(StatusCodes.Status200OK)]

public ActionResult<IEnumerable<Employee>> GetAll()

{

return Ok(employees);

}

[HttpGet("{id}")]

[ProducesResponseType(StatusCodes.Status200OK)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public ActionResult<Employee> GetById(int id)

{

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

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

return Ok(emp);

}

[HttpPost]

[ProducesResponseType(StatusCodes.Status201Created)]

public ActionResult<Employee> Create(Employee employee)

{

employee.Id = employees.Max(e => e.Id) + 1;

employees.Add(employee);

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

}

[HttpPut("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Update(int id, Employee employee)

{

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

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

emp.Name = employee.Name;

emp.Department = employee.Department;

return NoContent();

}

[HttpDelete("{id}")]

[ProducesResponseType(StatusCodes.Status204NoContent)]

[ProducesResponseType(StatusCodes.Status404NotFound)]

public IActionResult Delete(int id)

{

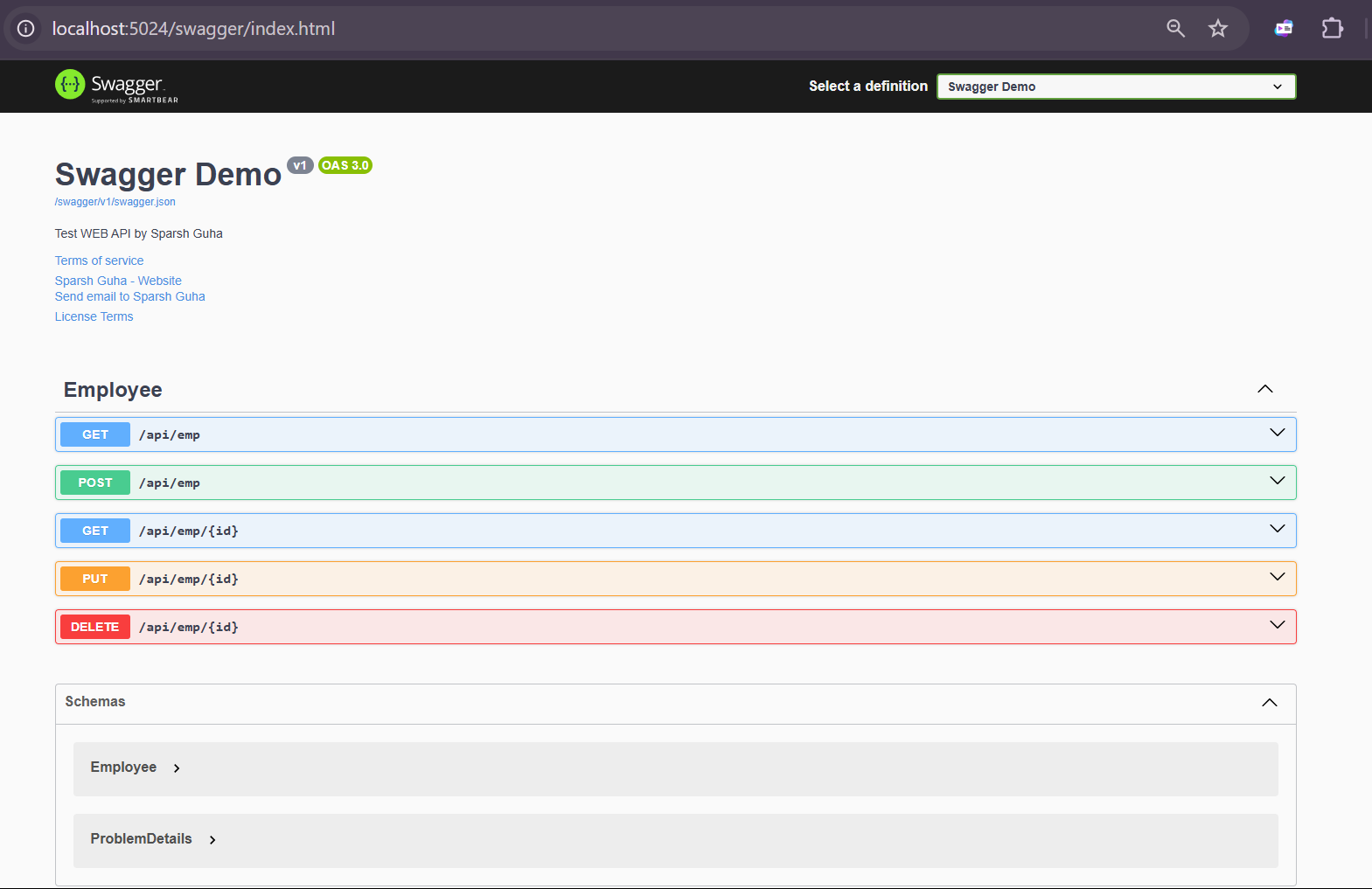
var emp = employees.FirstOrDefault(e => e.Id == id);

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

employees.Remove(emp);

return NoContent(); }}

**OUTPUT:**

****

**QUESTION-3**

**Web Api using custom model class**

**Create a Custom action filter for Authorization.**

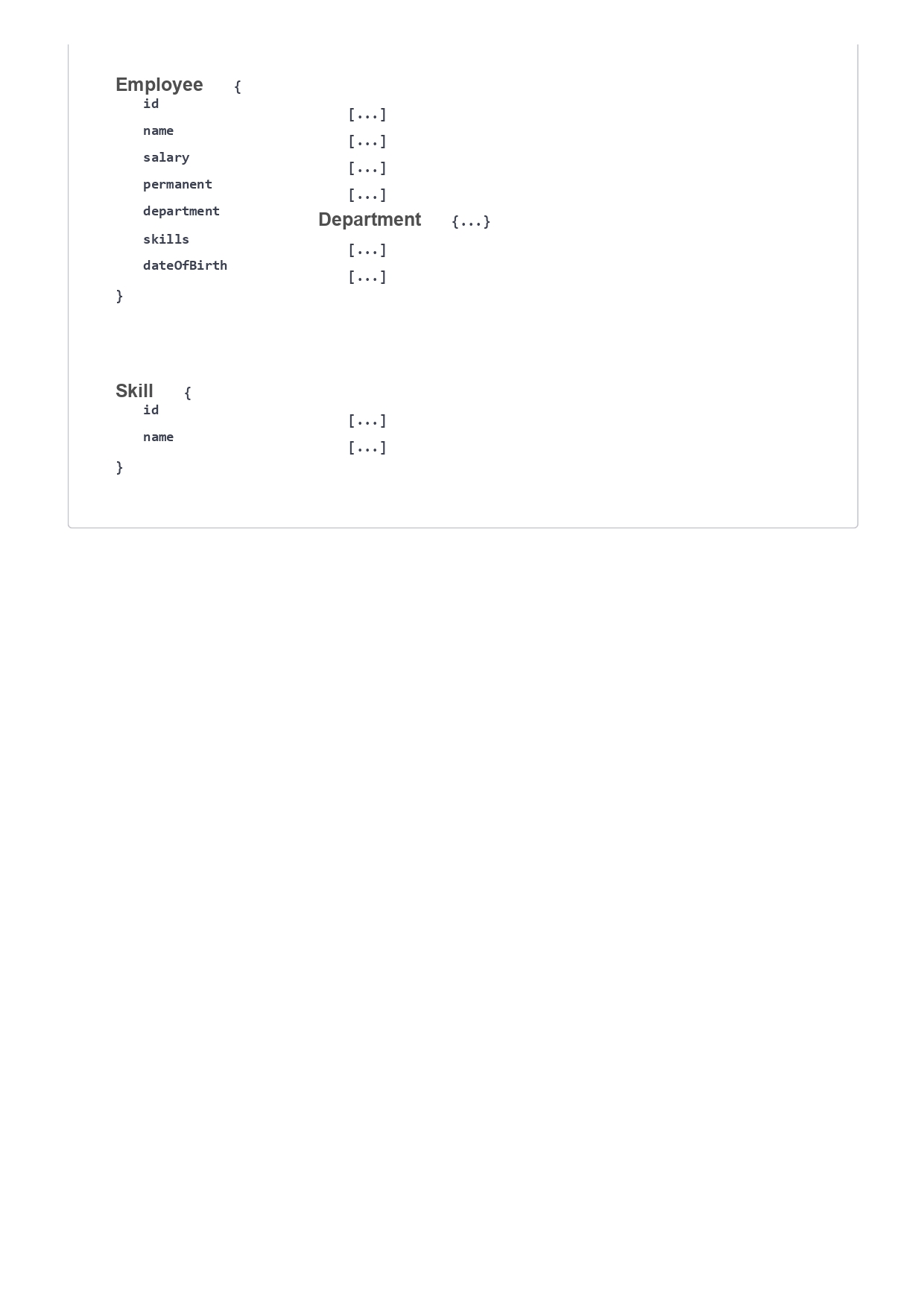
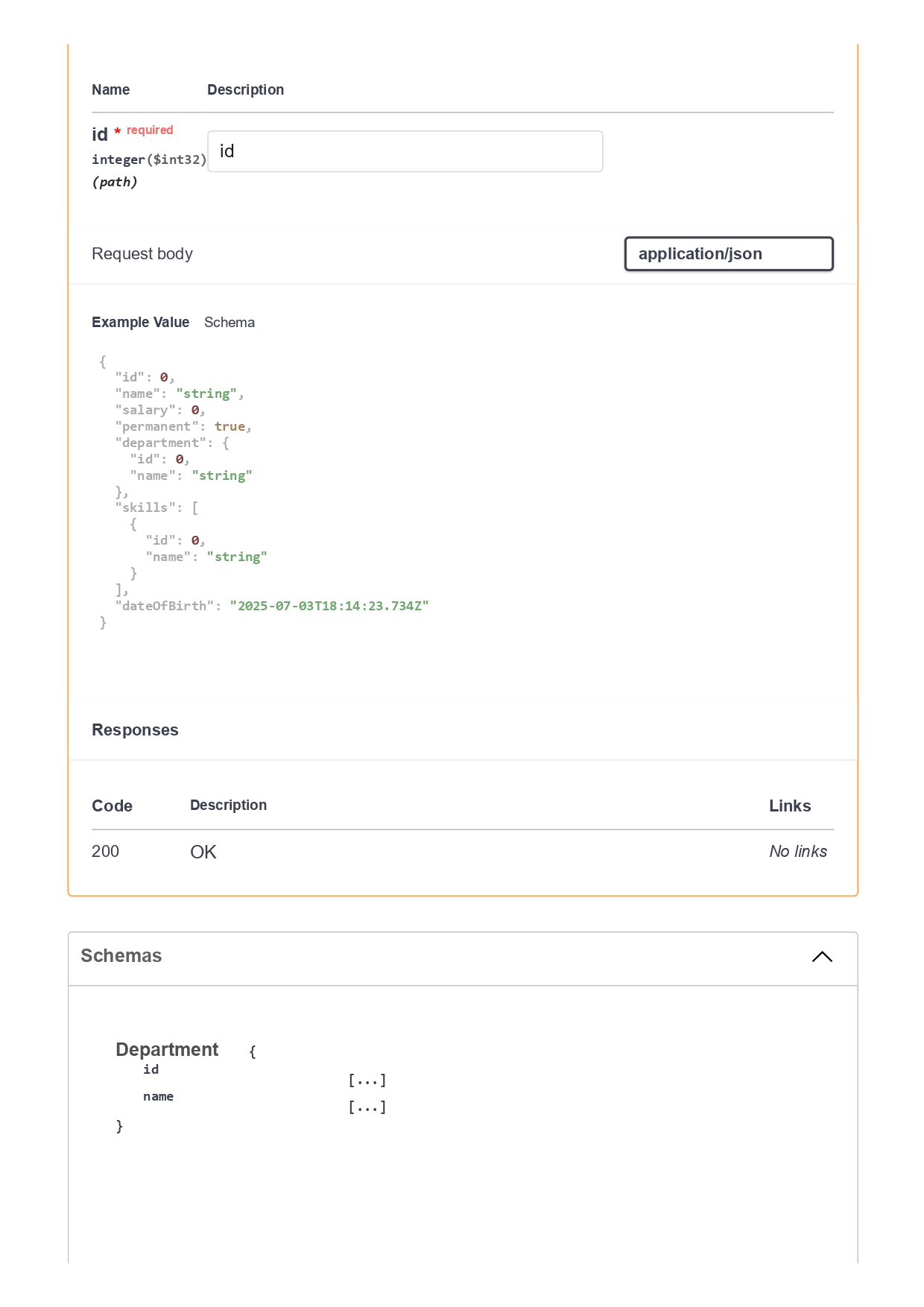
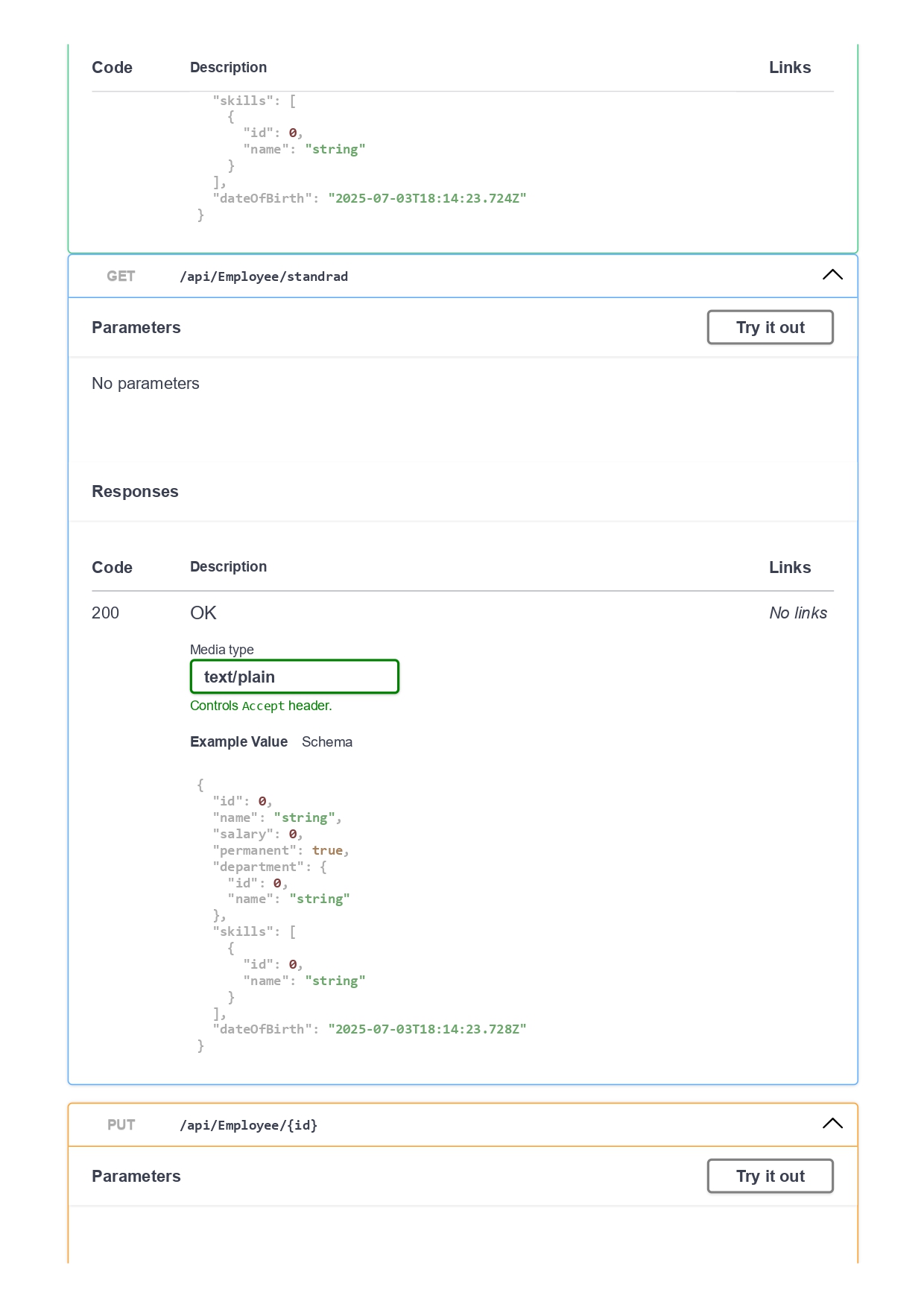
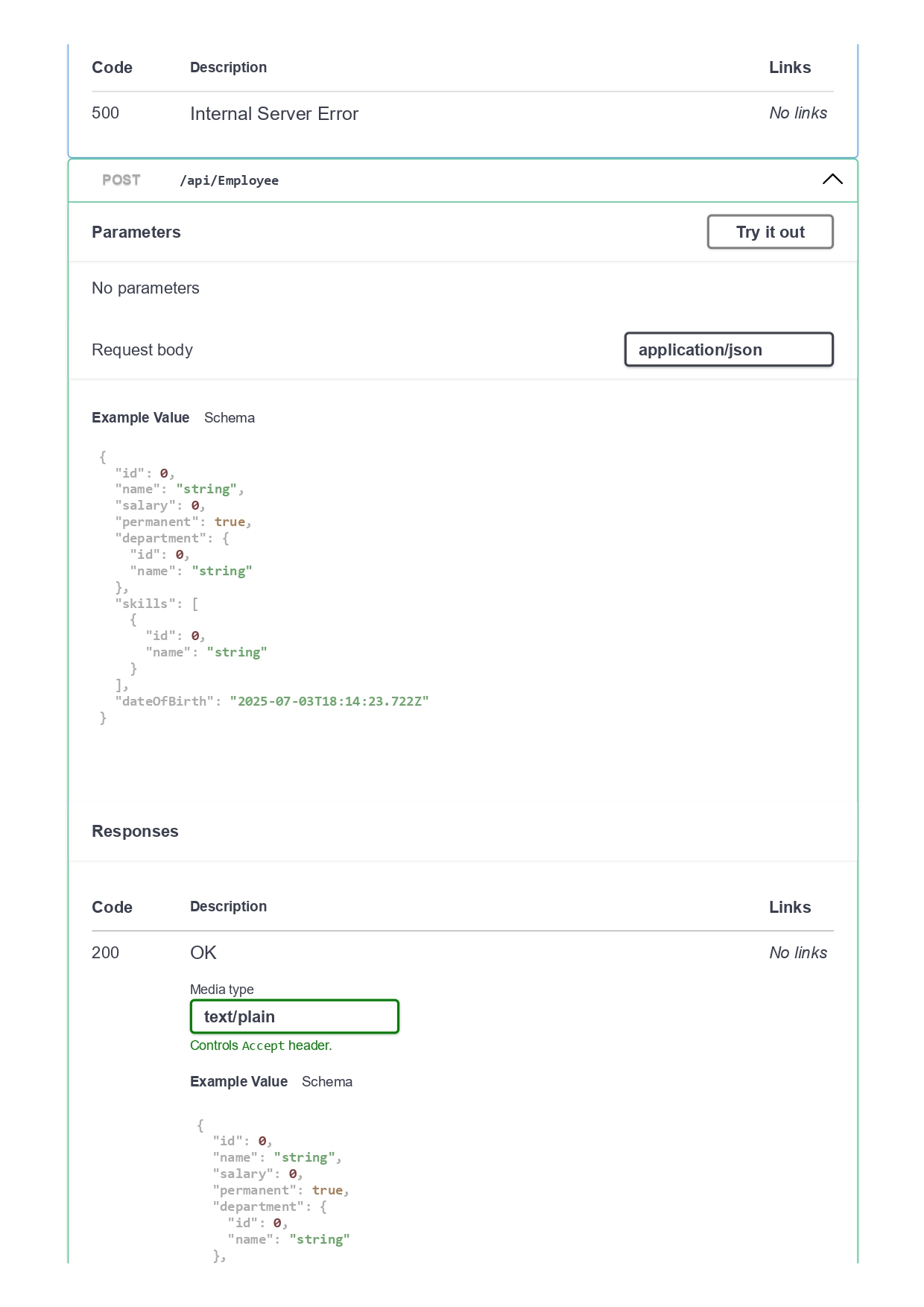
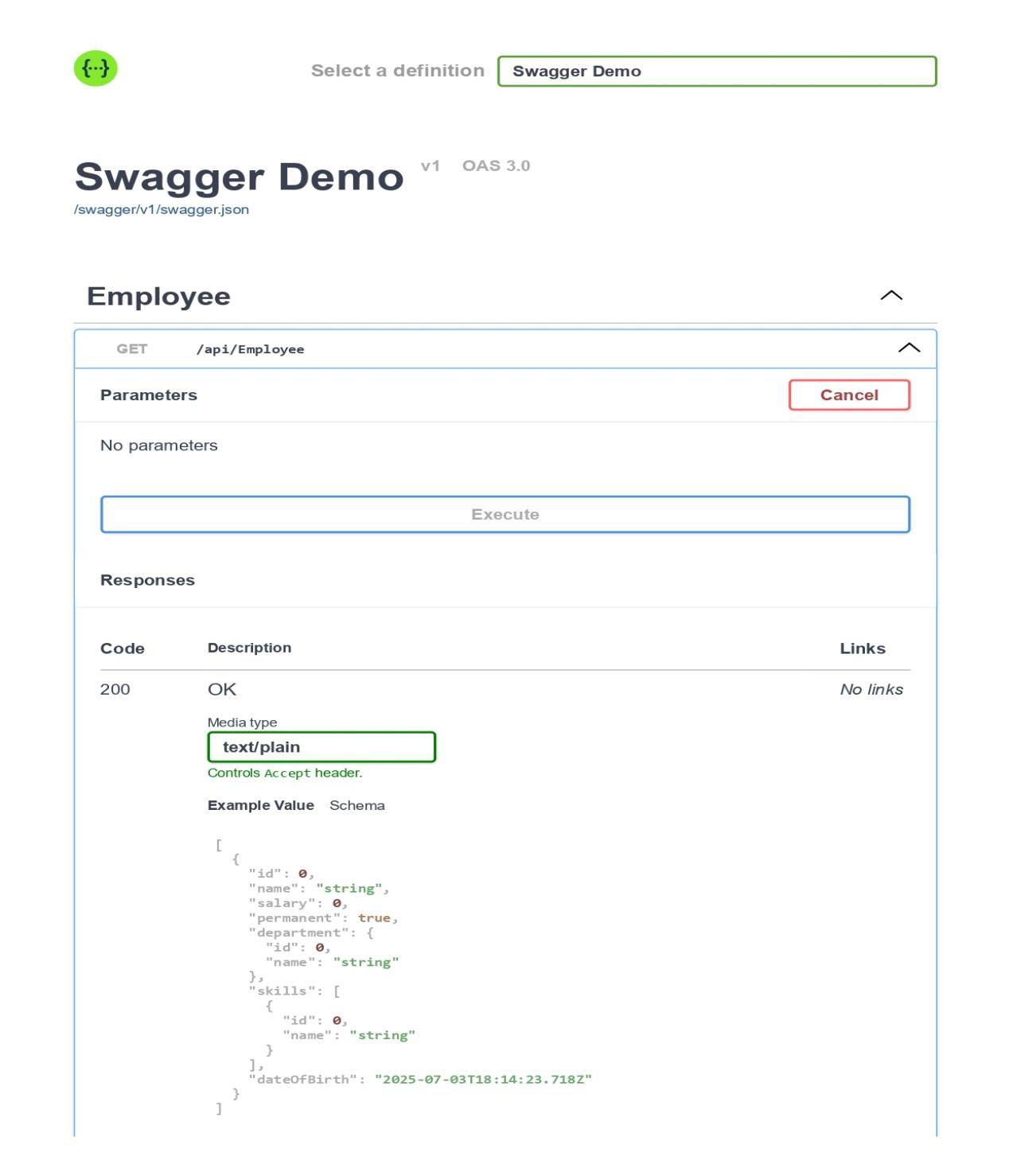
**Custom Exception filter**

**Input:**

**Code:**

**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; }  
}  
Filters/CustomAuthFilter.cs  
csharp  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Mvc.Filters;  
using System.Linq;  
public class CustomAuthFilter : ActionFilterAttribute  
{  
public override void OnActionExecuting(ActionExecutingContext context)  
{  
var hasAuth = context.HttpContext.Request.Headers.TryGetValue("Authorization", out var token);  
if (!hasAuth)  
{  
context.Result = new BadRequestObjectResult("Invalid request - No Auth token");  
return;  
}  
if (!token.ToString().Contains("Bearer"))  
{  
context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");  
return;  
}  
base.OnActionExecuting(context);  
}}  
Filters/CustomExceptionFilter.cs  
csharp  
using Microsoft.AspNetCore.Mvc.Filters;  
using Microsoft.AspNetCore.Mvc;  
using System;  
using System.IO;  
  
public class CustomExceptionFilter : IExceptionFilter  
{  
public void OnException(ExceptionContext context)  
{  
string message = $"{DateTime.Now}: {context.Exception.Message}{Environment.NewLine}";  
string filePath = Path.Combine(Directory.GetCurrentDirectory(), "exceptionlog.txt");  
File.AppendAllText(filePath, message);  
context.Result = new ObjectResult("Internal server error") { StatusCode = 500 };  
context.ExceptionHandled = true;  
}}  
**EmployeeController.cs**  
  
using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Authorization;  
using System.Collections.Generic;  
using System.Linq;  
[ApiController]  
[Route("api/[controller]")]  
[ServiceFilter(typeof(CustomAuthFilter))]  
[ServiceFilter(typeof(CustomExceptionFilter))]  
public class EmployeeController : ControllerBase  
{  
private static List<Employee> employees = GetStandardEmployeeList();  
[HttpGet]  
[AllowAnonymous]  
[ProducesResponseType(typeof(List<Employee>), 200)]  
[ProducesResponseType(500)]  
public ActionResult<List<Employee>> Get()  
{  
throw new System.Exception("Test exception");  
return employees;  
}  
[HttpGet("standrad")]  
[AllowAnonymous]  
public ActionResult<Employee> GetStandrad()  
{  
return employees.First();  
}  
[HttpPost]  
public ActionResult<Employee> Post([FromBody] Employee emp)  
{  
emp.Id = employees.Max(e => e.Id) + 1;  
employees.Add(emp);  
return emp;  
}  
[HttpPut("{id}")]  
public IActionResult Put(int id, [FromBody] Employee emp)  
{  
var e = employees.FirstOrDefault(x => x.Id == id);  
if (e == null) return NotFound();  
e.Name = emp.Name;  
e.Salary = emp.Salary;  
e.Permanent = emp.Permanent;  
e.Department = emp.Department;  
e.Skills = emp.Skills;  
e.DateOfBirth = emp.DateOfBirth;  
return NoContent();  
}  
private static List<Employee> GetStandardEmployeeList()  
{  
return new List<Employee>  
{  
new Employee  
{  
Id = 1,  
Name = "Alice",  
Salary = 50000,  
Permanent = true,  
Department = new Department { Id = 1, Name = "IT" },  
Skills = new List<Skill>  
{  
new Skill { Id = 1, Name = "C#" },  
new Skill { Id = 2, Name = "SQL" }  
},  
DateOfBirth = new DateTime(1990, 1, 1)  
},  
new Employee  
{  
Id = 2,  
Name = "Bob",  
Salary = 40000,  
Permanent = false,  
Department = new Department { Id = 2, Name = "HR" },  
Skills = new List<Skill>  
{  
new Skill { Id = 3, Name = "Excel" }  
},  
DateOfBirth = new DateTime(1992, 5, 20)  
}};}}  
**Program.cs**  
using Microsoft.AspNetCore.Builder;  
using Microsoft.Extensions.DependencyInjection;  
using Microsoft.OpenApi.Models;  
var builder = WebApplication.CreateBuilder(args);  
builder.Services.AddControllers();  
builder.Services.AddScoped<CustomAuthFilter>();  
builder.Services.AddScoped<CustomExceptionFilter>();  
builder.Services.AddEndpointsApiExplorer();  
builder.Services.AddSwaggerGen(c =>  
{  
c.SwaggerDoc("v1", new OpenApiInfo { Title = "Swagger Demo", Version = "v1" });  
});  
var app = builder.Build();  
app.UseSwagger();  
app.UseSwaggerUI();  
app.MapControllers();  
app.Run();

**OUTPUT:**

****

**QUESTION -4**

**Web Api CRUD operation**

**Code:**

**Input: (Updated)**

using Microsoft.AspNetCore.Mvc;

using Microsoft.AspNetCore.Authorization;

using System.Collections.Generic;

using System.Linq;

[ApiController]

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

[ServiceFilter(typeof(CustomAuthFilter))]

[ServiceFilter(typeof(CustomExceptionFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> employees = GetStandardEmployeeList();

[HttpGet]

[AllowAnonymous]

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

[ProducesResponseType(500)]

public ActionResult<List<Employee>> Get()

{

throw new System.Exception("Test exception");

//return employees;

}

[HttpGet("standrad")]

[AllowAnonymous]

public ActionResult<Employee> GetStandrad()

{

return employees.First();

}

[HttpPost]

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

{

emp.Id = employees.Max(e => e.Id) + 1;

employees.Add(emp);

return emp;

}

// INTEGRATED: Your required PUT method

[HttpPut("{id}")]

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

{

if (id <= 0)

return BadRequest("Invalid employee id");

var e = employees.FirstOrDefault(x => x.Id == id);

if (e == null)

return BadRequest("Invalid employee id");

e.Name = emp.Name;

e.Salary = emp.Salary;

e.Permanent = emp.Permanent;

e.Department = emp.Department;

e.Skills = emp.Skills;

e.DateOfBirth = emp.DateOfBirth;

return Ok(e);

}

private static List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 50000,

Permanent = true,

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

Skills = new List<Skill>

{

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

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

},

DateOfBirth = new DateTime(1990, 1, 1)

},

new Employee

{

Id = 2,

Name = "Bob",

Salary = 40000,

Permanent = false,

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

Skills = new List<Skill>

{

new Skill { Id = 3, Name = "Excel" }

},

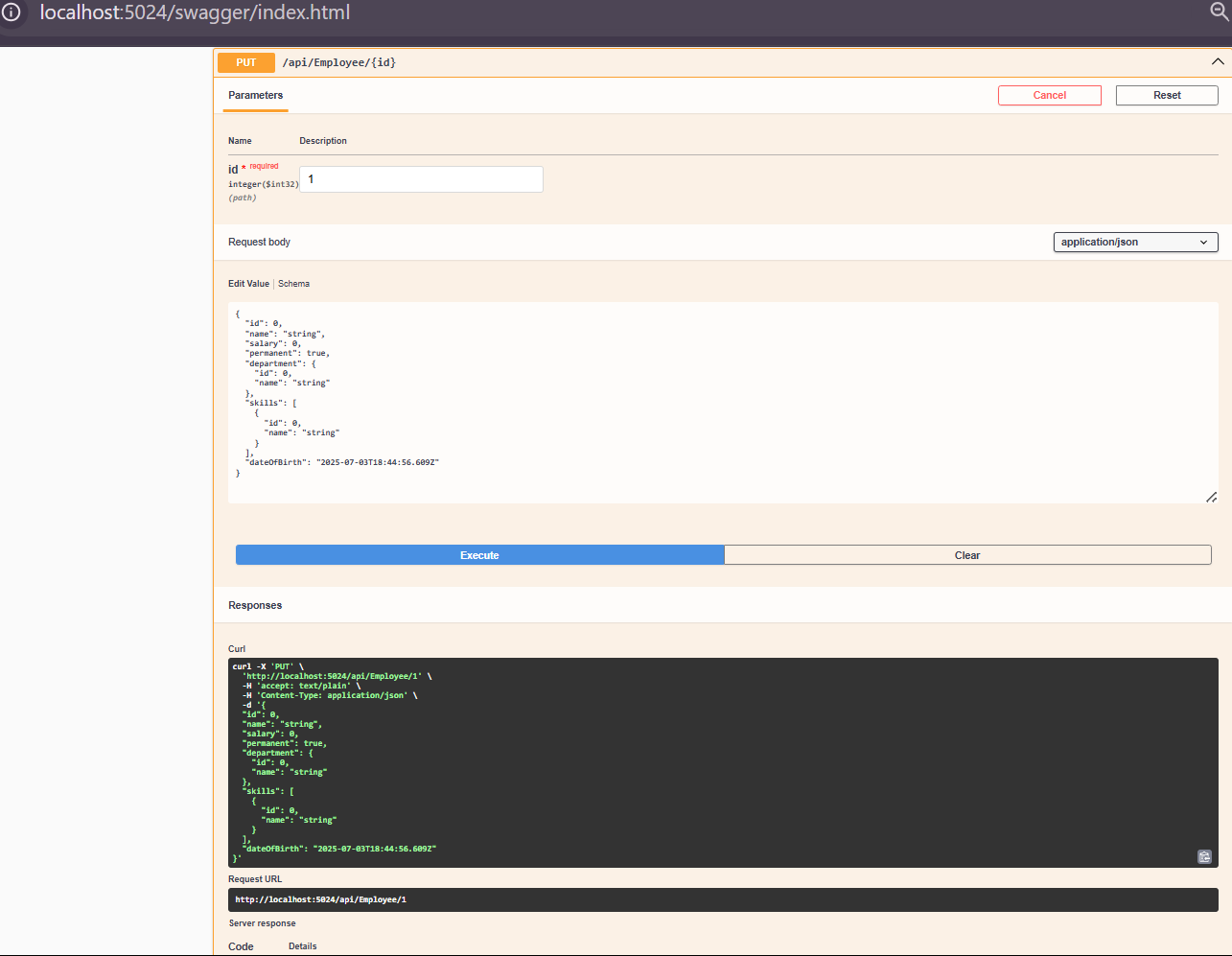
DateOfBirth = new DateTime(1992, 5, 20)

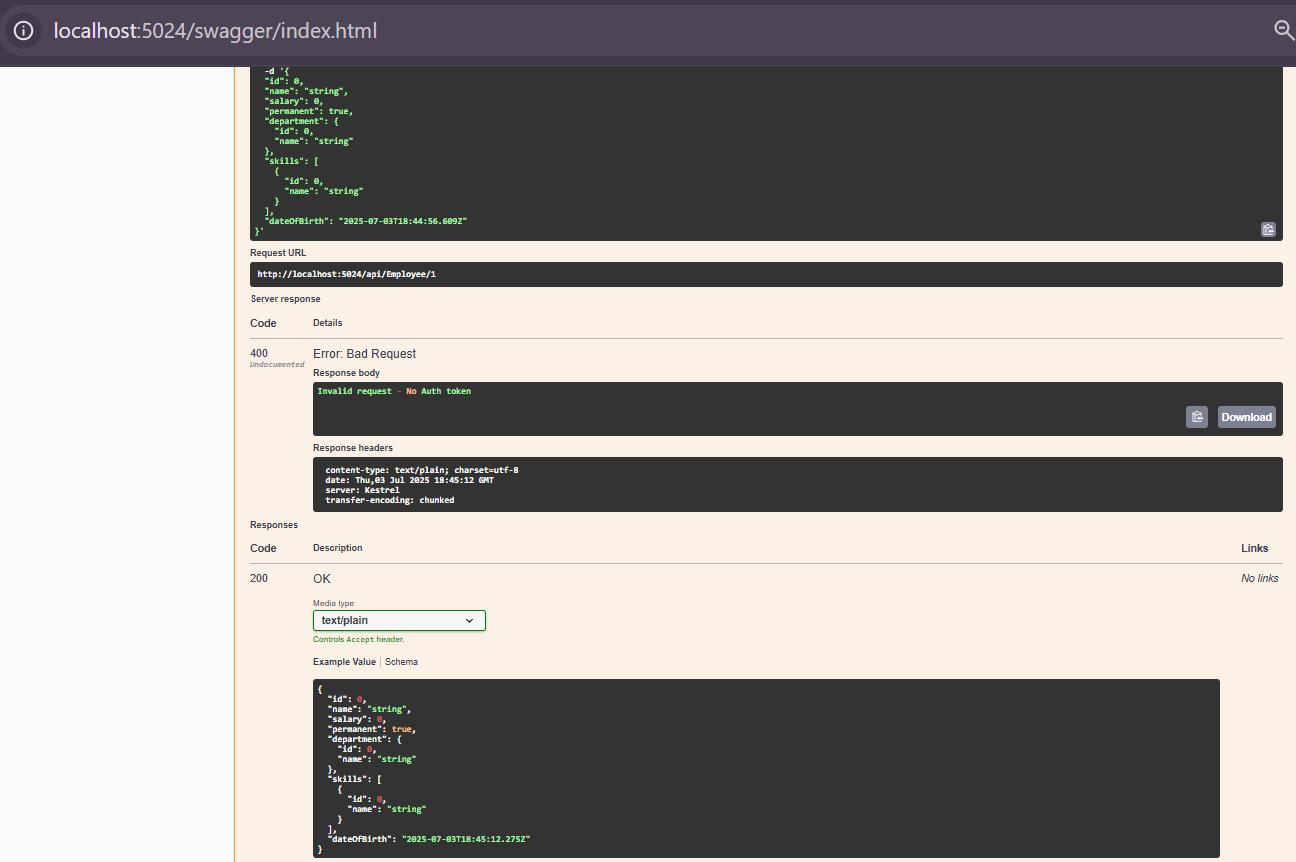
}

};

}

}





**QUESTION-5**

1. **JsonWebToken**
2. **Use the JWT generated thru the AuthController to be used in POSTMAN request.**
3. **Check for JWT expiration**
4. **Add the roles to be authorized in the Authorize attribute.**

**INPUT:**

**CODE:**

**Program.cs**

using Microsoft.AspNetCore.Authentication.JwtBearer;

using Microsoft.IdentityModel.Tokens;

using System.Text;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddCors(options =>

{

options.AddPolicy("AllowAll", policy =>

{

policy.AllowAnyOrigin()

.AllowAnyHeader()

.AllowAnyMethod();

});

});

var 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

};

});

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

app.UseCors("AllowAll");

app.UseAuthentication();

app.UseAuthorization();

app.UseSwagger();

app.UseSwaggerUI();

app.MapControllers();

app.Run();

## ****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("token")]

public IActionResult GetToken(int userId = 1, string userRole = "Admin")

{

var token = GenerateJSONWebToken(userId, userRole);

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);

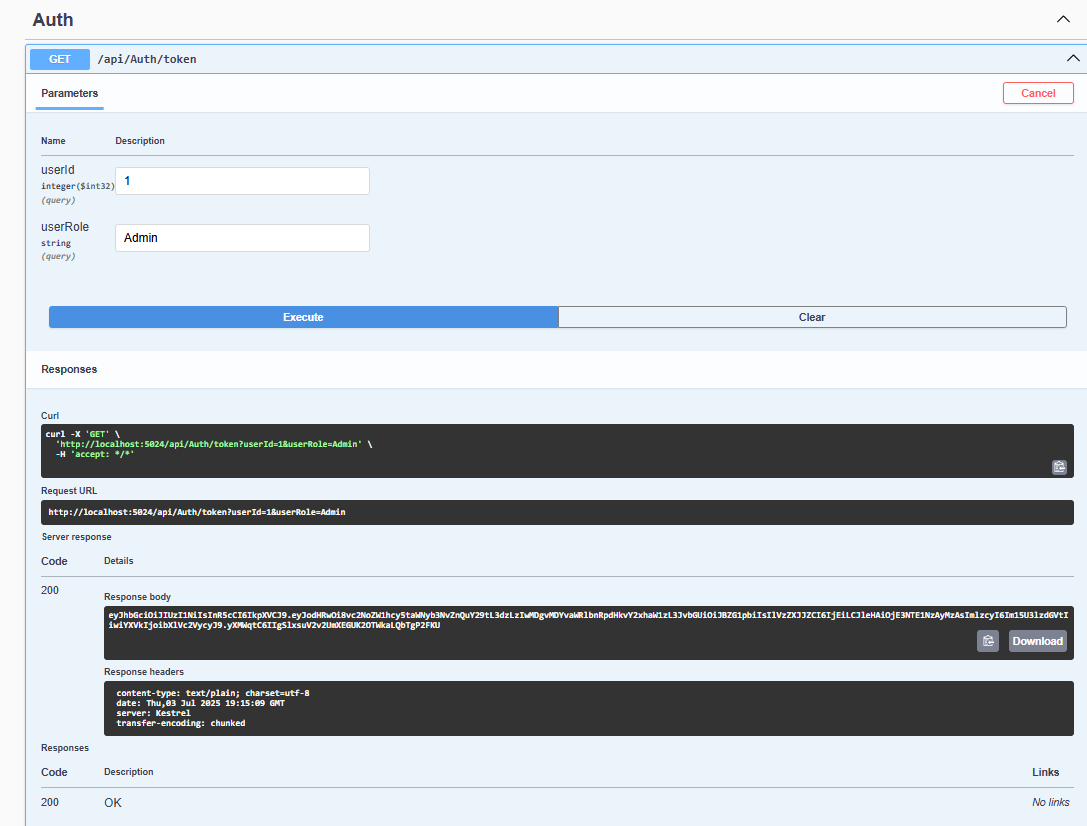
}

}

**EmployeeController.cs**

using Microsoft.AspNetCore.Mvc;  
using Microsoft.AspNetCore.Authorization;  
using Microsoft.AspNetCore.Authorization;  
using System.Collections.Generic;  
using System.Linq;  
using System;  
  
[ApiController]  
[Route("api/[controller]")]  
[Authorize(Roles = "Admin,POC")]  
public class EmployeeController : ControllerBase  
{  
private static List<Employee> employees = new List<Employee>  
{  
new Employee  
{  
Id = 1,  
Name = "Alice",  
Salary = 50000,  
Permanent = true,  
Department = new Department { Id = 1, Name = "IT" },  
Skills = new List<Skill>  
{  
new Skill { Id = 1, Name = "C#" },  
new Skill { Id = 2, Name = "SQL" }  
},  
DateOfBirth = new DateTime(1990, 1, 1)  
},  
new Employee  
{  
Id = 2,  
Name = "Bob",  
Salary = 40000,  
Permanent = false,  
Department = new Department { Id = 2, Name = "HR" },  
Skills = new List<Skill>  
{  
new Skill { Id = 3, Name = "Excel" }  
},  
DateOfBirth = new DateTime(1992, 5, 20)  
}  
};  
  
[HttpGet]  
public ActionResult<List<Employee>> Get()  
{  
return employees;  
}  
  
[HttpPut("{id}")]  
public ActionResult<Employee> Put(int id, [FromBody] Employee emp)  
{  
if (id <= 0)  
return BadRequest("Invalid employee id");  
var e = employees.FirstOrDefault(x => x.Id == id);  
if (e == null)  
return BadRequest("Invalid employee id");  
e.Name = emp.Name;  
e.Salary = emp.Salary;  
e.Permanent = emp.Permanent;  
e.Department = emp.Department;  
e.Skills = emp.Skills;  
e.DateOfBirth = emp.DateOfBirth;  
  
return Ok(e);  
}  
}

**OUTPUT:**



**QUESTION - 16  
Kafka Integration with C#:**

1. **Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.**
2. **Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications.**

**Introduction to Kafka**  
Apache Kafka is a distributed event streaming platform for high-throughput, low-latency data pipelines and streaming apps  
It enables real-time publish/subscribe messaging, storage, and processing of streams.

**Kafka Architecture**  
Producer: Publishes messages to Kafka topics.

Consumer: Subscribes and reads messages from topics.

Broker: Kafka server that stores messages and handles requests.

Multiple brokers form a cluster for scalability and fault tolerance

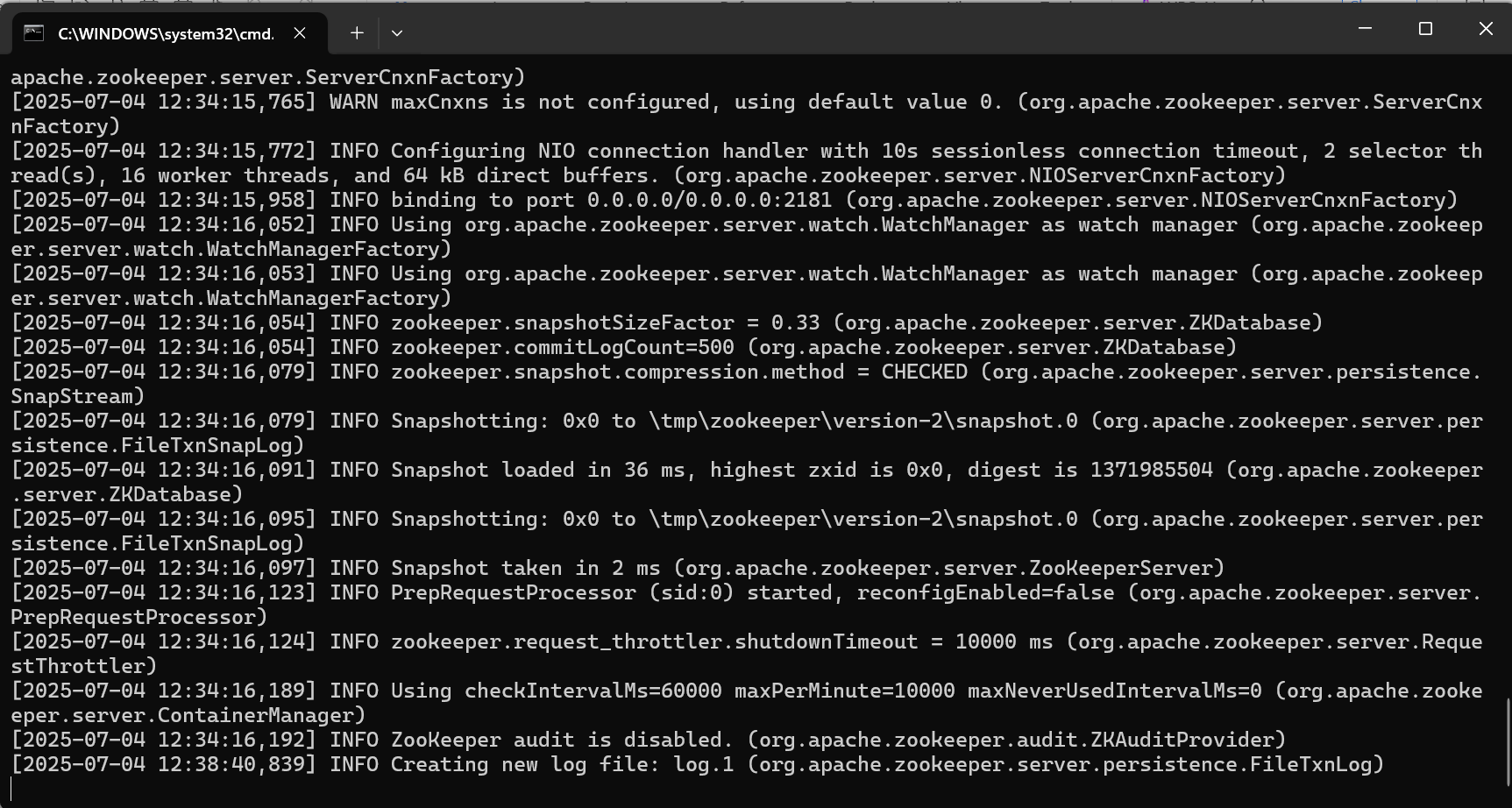
Topic: Logical channel for organizing messages. Producers write to topics, consumers read from them  
Partition: Each topic is split into partitions for parallelism and scalability. Each partition is an ordered, immutable sequence of messages

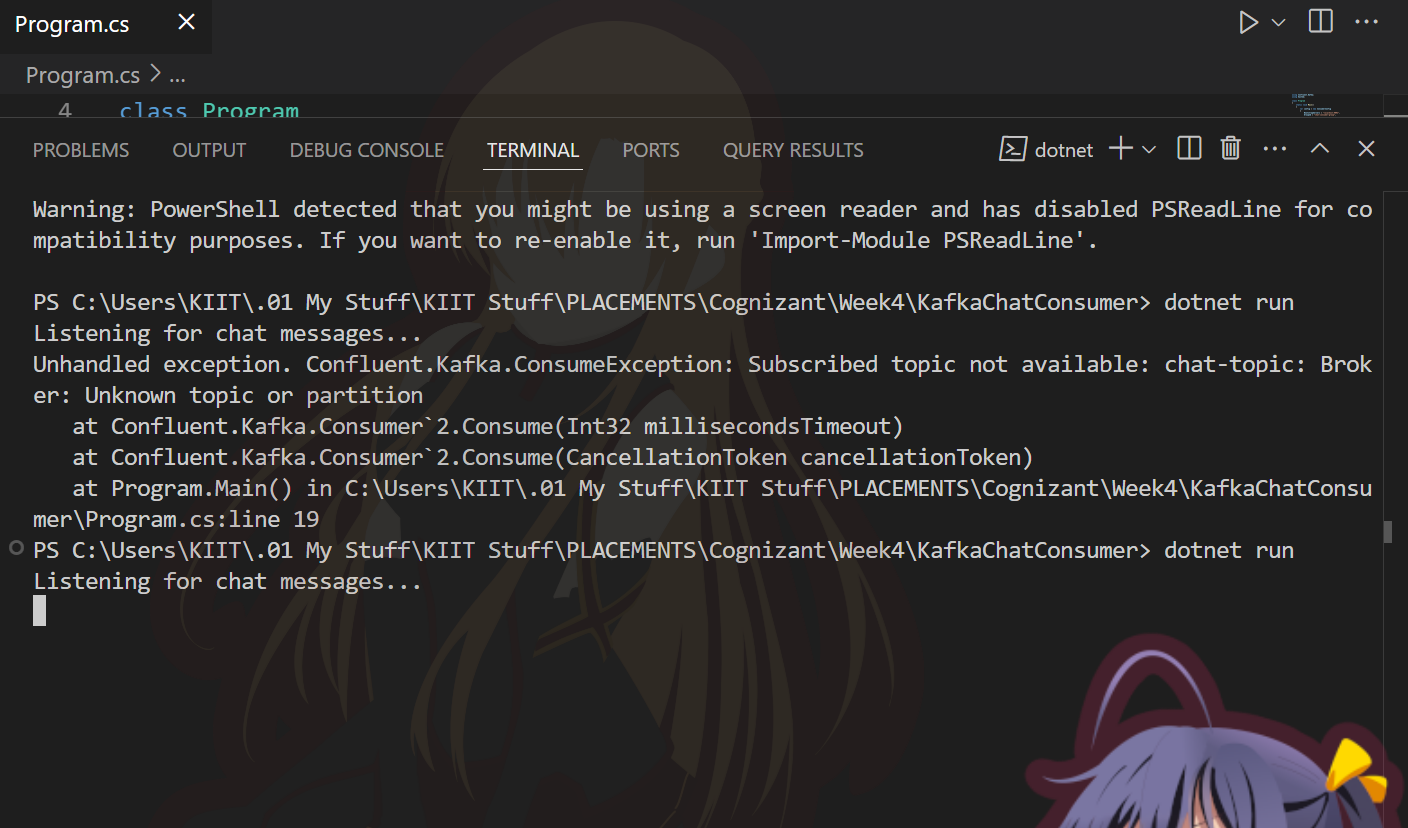
ZooKeeper: Coordinates brokers and manages cluster state (required for older Kafka versions)

Replication: Partitions are replicated across brokers for fault tolerance.  
Consumer Group: Multiple consumers sharing the workload for a topic.

**Input:**  
**Kafka Chat Application: Step-by-Step Commands**  
  
**1  
Start ZooKeeper**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\zookeeper-server-start.bat config\zookeeper.properties  
  
**| 2 | Start Kafka Broker |**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\kafka-server-start.bat config\server.properties  
  
**| 3 | Create Kafka Topic (if not already created) |**  
cd C:\kafka\_2.13-3.7.2  
bin\windows\kafka-topics.bat --create --topic chat-topic --bootstrap-server localhost:9092 --partitions 1 --replication-factor 1  
 **| 4 | Run Kafka Consumer Console App |**  
cd path\_to\_KafkaChatConsumer\_project  
dotnet run  
 **| 5 | Run Kafka Producer Console App |**  
cd path\_to\_KafkaChatProducer\_project  
dotnet run  
  
**| 6 | Send and Receive Messages |**  
Type chat messages in the producer console and see them appear in the consumer console.

**Output:**

****

****