**WEB API HANDS-ON**

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.

WeatherForecastController.cs

using Microsoft.AspNetCore.Mvc;

namespace FirstWebAPI.Controllers

{

[ApiController]

[Route("[controller]")]

public class ValuesController : ControllerBase

{

[HttpGet]

public IActionResult Get()

{

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

}

[HttpGet("{id}")]

public IActionResult Get(int id)

{

return Ok("value " + id);

}

[HttpPost]

public IActionResult Post([FromBody] string value)

{

return Ok("Posted: " + value);

}

[HttpPut("{id}")]

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

{

return Ok($"Updated id {id} with value: {value}");

}

[HttpDelete("{id}")]

public IActionResult Delete(int id)

{

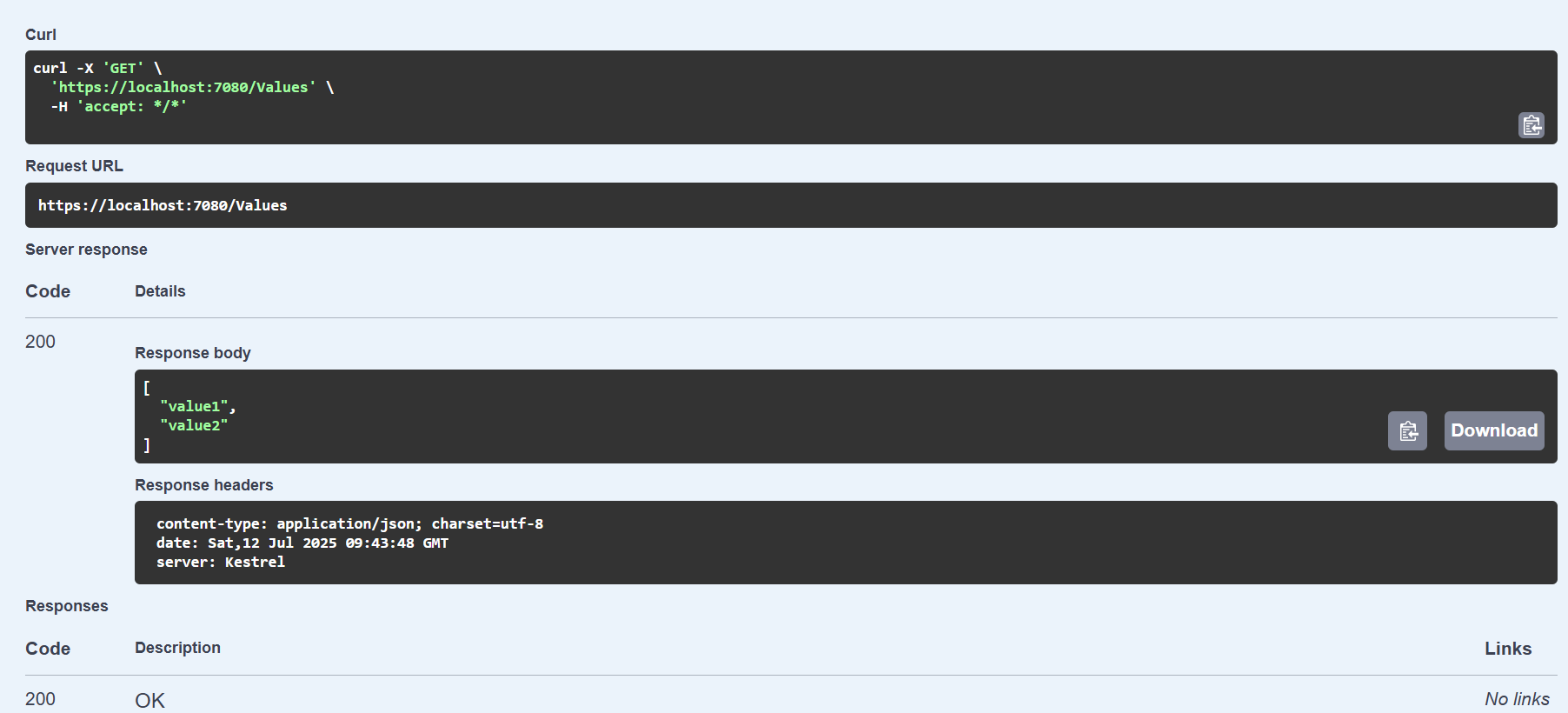
return Ok($"Deleted id {id}");

}

}

}

Output:



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

(i)Create a .Net core web application with API template. (Use existing application if created). Install Swashbuckle.AspNetCore Nuget package. Post this do the following steps in Startup.cs

* In ConfigureServices method, add the code provided below.

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Info

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = "None",

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

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

});

* In Configure method, add the code provided below.

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

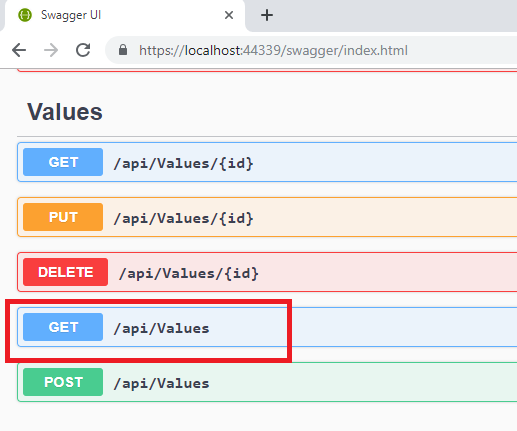
});

Execute the application which will load the default ‘Values’ controller(Settings as per launchSettings.json) GET action method. Change the url to <https://localhost:[port> number]/swagger

Notice the Title, Version, Contact detail provided shown on the top of the page

Notice the Values controller HttpVerb action methods getting listed.

Click the ‘GET’ action verb method(Without the parameter).



It opens a panel which has ‘Try it out’ button. Click that and Click ‘Execute’ button.

**Program.cs**

using Microsoft.OpenApi.Models;

var builder = WebApplication.CreateBuilder(args);

builder.Services.AddControllers();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = new Uri("https://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")

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

}

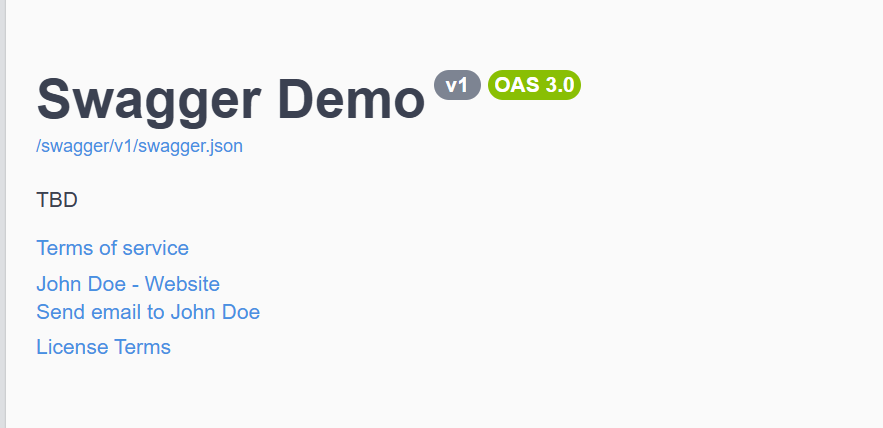
app.UseHttpsRedirection();

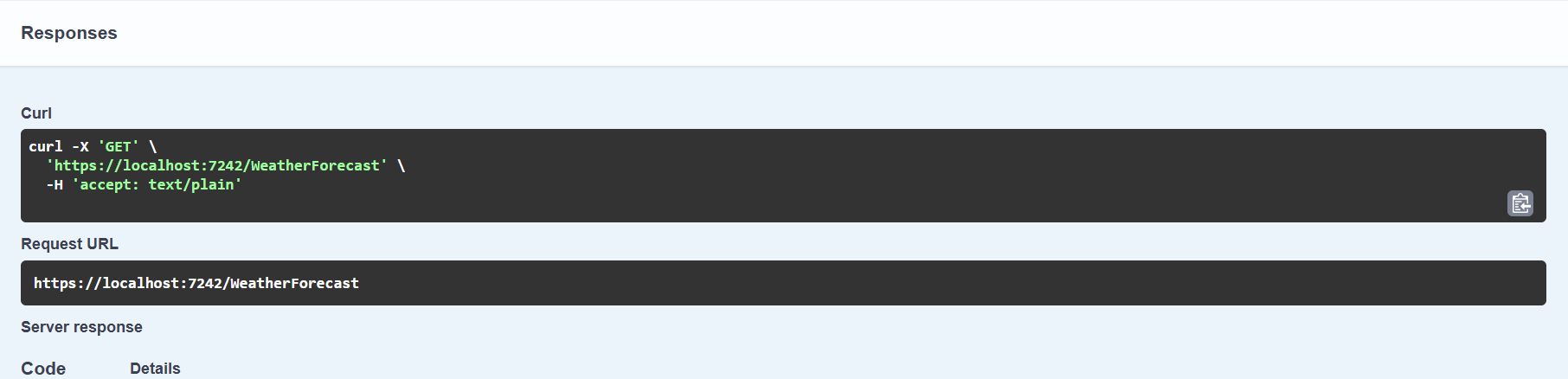
app.UseAuthorization();

app.MapControllers();

app.Run();

Output:



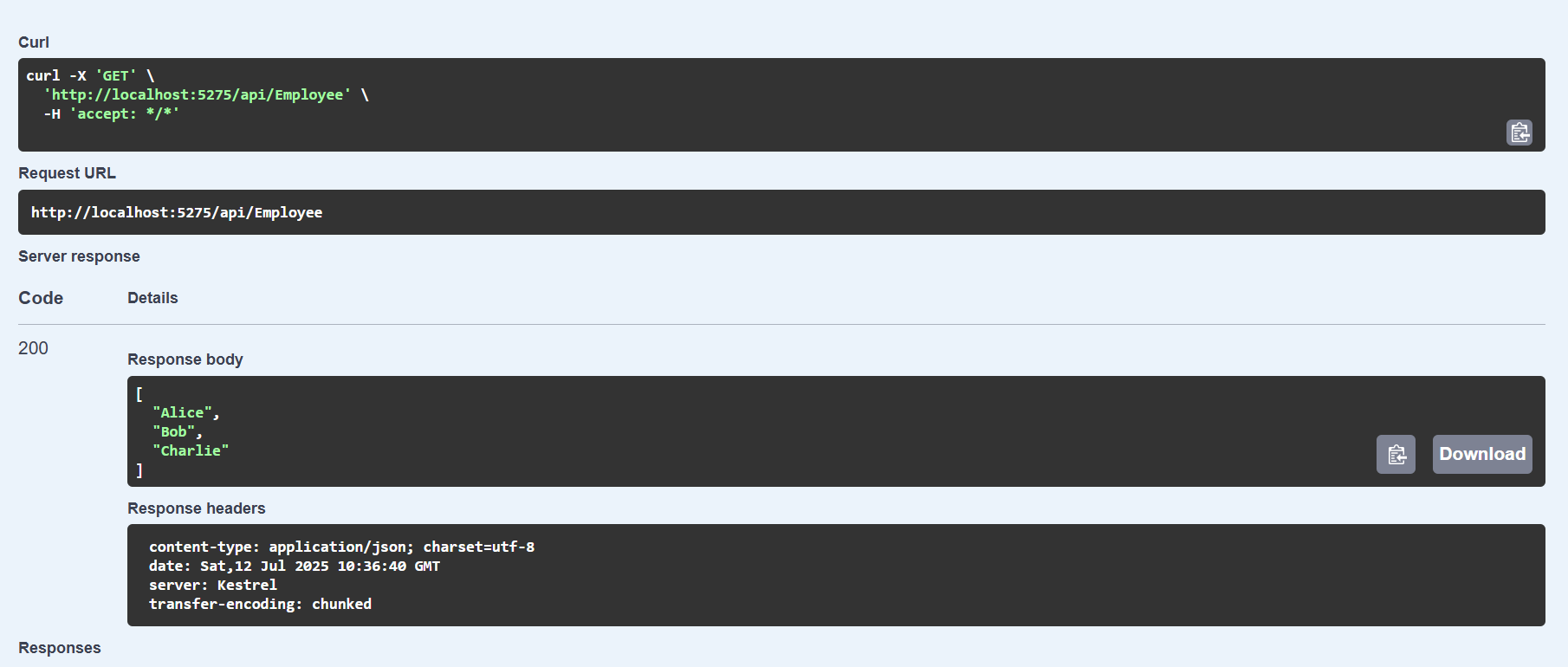


(ii) Use POSTMAN tool, to point to the local Web API that was created with Employee controller. Test the GET action method using POSTMAN.

Verify the output if the List of employees are listed in the ‘Body’ part of the GET window on POSTMAN tool.

Verify the Status on the right side of the output pane on POSTMAN tool.

Output:



**3.Web Api using custom model class**

i)Create a Custom class ‘Employee’ of the below defined structure

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; }

}

Create a new controller - EmployeeController with Read Write actions

Constructor: Create few records, HTTPGet, HTTPPost/HTTPPut

Create a Private method GetStandardEmployeeList that returns a List of Employee class. Invoke this method in the Get action method of the EmployeeController that was created in the previous step.

Public ActionResult<Employee> GetStandrad()

Modify the return type of the Get action method(without parameter) to return List of Employee class object

Add ProducesResponseType to the GET action method for Status code 200

Check the Swagger description for the GET method for success status code

EmployeeController.cs

using Microsoft.AspNetCore.Mvc;

using MyWebApiDemo.Models;

using MyWebApiDemo.Filters;

namespace MyWebApiDemo.Controllers

{

[ApiController]

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

[ServiceFilter(typeof(CustomAuthFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = GetStandardEmployeeList();

[HttpGet]

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

[ProducesResponseType(500)]

public ActionResult<List<Employee>> Get()

{

return Ok(\_employees);

}

[HttpPost]

public IActionResult Post([FromBody] Employee employee)

{

\_employees.Add(employee);

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

}

[HttpPut("{id}")]

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

{

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

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

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 NoContent();

}

private static List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 50000,

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" }

}

}

};

}

}

}

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

The requirement is to intercept incoming requests and check if there is a key ‘Authorization’ in the request header or not. If it is there, then to check if it contains a value ‘Bearer’ or not.

Create a folder ‘Filters’ in the application solution. Create a class ‘**CustomAuthFilter**’ to filter requests. Inherit ActionFilterAttribute. Override OnActionExecuting method to check if the request object has Header ‘Authorization’ or not. If not, throw BadRequestResult with the message

Invalid request - No Auth token

If the header is present, then check if the value contains the word ‘Bearer’. If not, throw BadRequestResult with the message

Invalid request - Token present but Bearer unavailable

Add an attribute **CustomAuthFilter** to the Employee controller to filter any request to check for the Authorization token in the request header.

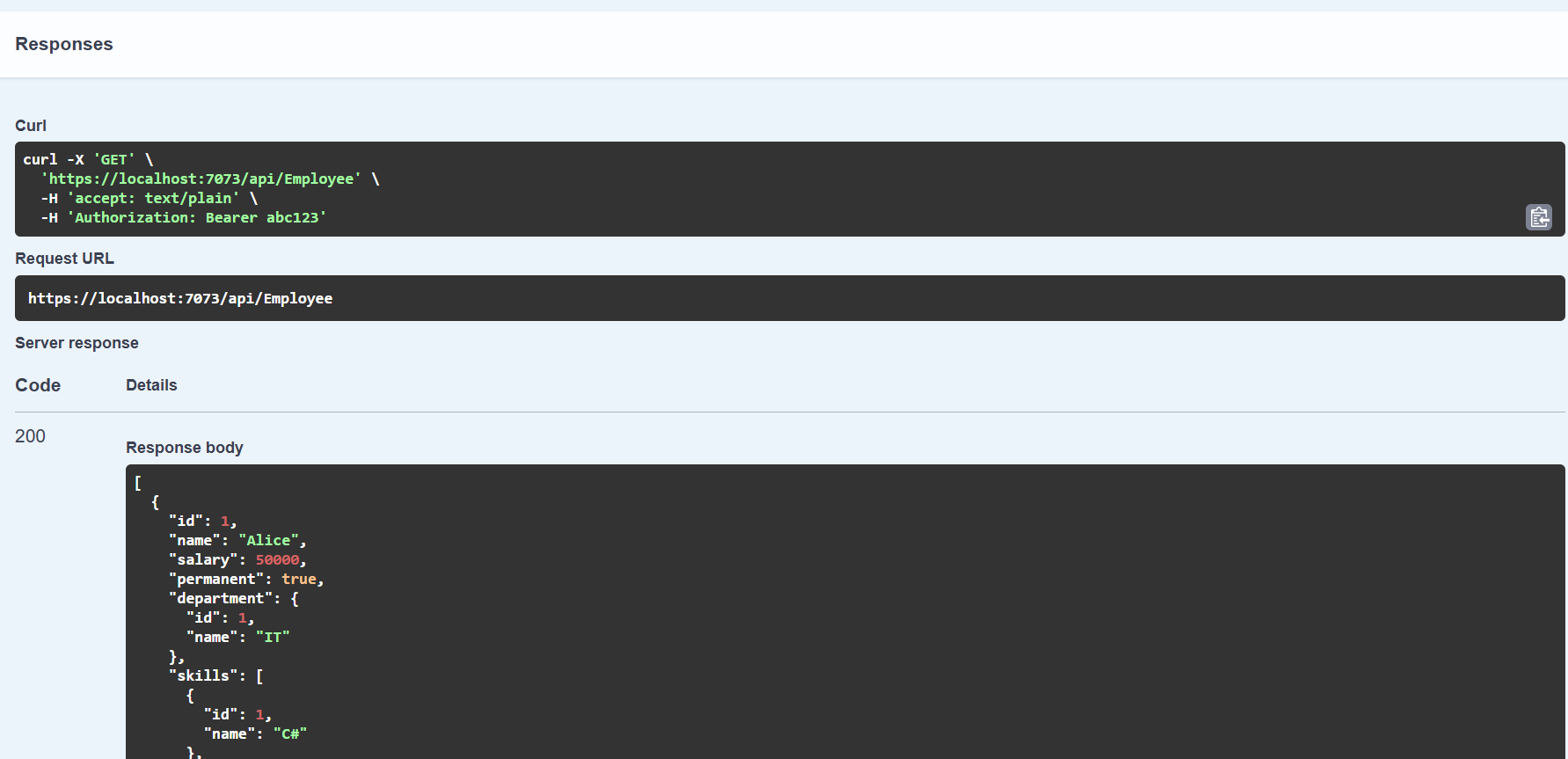
Output:



**Iii)Custom Exception filter**

Create a class ‘CustomExceptionFilter’ to catch the exceptions occuring the application. Implement IExceptionFilter thru the OnException method  
  
Use the exception context to fetch the exception detail. Capture that and write it to a File in the system.  
  
Set the Result property of the exception context to ExceptionResult.  
  
Throw an exception in GET action method.  
Ensure that the GET action method has ProducesResponseType for 500 - Internal server error  
  
Use Swagger to test the exception and message being thrown.  
  
Note: This needs WebApiCompatShim NuGet package installation

Output:



1. **Web Api CRUD operation**

Update Employee data as per the input thru Web API PUT action method call

Employee information has to be updated based on the user input. Use Swagger tool to invoke the action method mapped with Http PUT action verb to update an employee data.

Modify the action method to return Employee data thru ActionResult.

Check if the id value is lesser than or equal to 0. If true, throw BadRequest action result with the message ‘Invalid employee id’

If the value is greater than 0 but not available in the list of employee ids that is there in the hardcoded list of employees, throw BadRequest action result with the same message as stated above.

If the id value is valid, use the JSON data from the input body and update the hardcoded list. Filter the employee list data for the input id and return that as the output.

EmployeeController.cs

using Microsoft.AspNetCore.Mvc;

using MyWebApiDemo.Models;

using MyWebApiDemo.Filters;

namespace MyWebApiDemo.Controllers

{

[ApiController]

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

[ServiceFilter(typeof(CustomAuthFilter))]

public class EmployeeController : ControllerBase

{

private static List<Employee> \_employees = GetStandardEmployeeList();

[HttpGet]

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

[ProducesResponseType(500)]

public ActionResult<List<Employee>> Get()

{

return Ok(\_employees);

}

[HttpPost]

public IActionResult Post([FromBody] Employee employee)

{

\_employees.Add(employee);

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

}

[HttpPut("{id}")]

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

{

updatedEmployee.Id = id;

if (id <= 0)

return BadRequest("Invalid employee id");

var existingEmployee = \_employees.FirstOrDefault(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);

}

private static List<Employee> GetStandardEmployeeList()

{

return new List<Employee>

{

new Employee

{

Id = 1,

Name = "Alice",

Salary = 50000,

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" }

}

},

new Employee

{

Id = 2,

Name = "Bob",

Salary = 60000,

Permanent = false,

DateOfBirth = new DateTime(1988, 8, 15),

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

Skills = new List<Skill>

{

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

new Skill { Id = 4, Name = "Recruitment" }

}

}

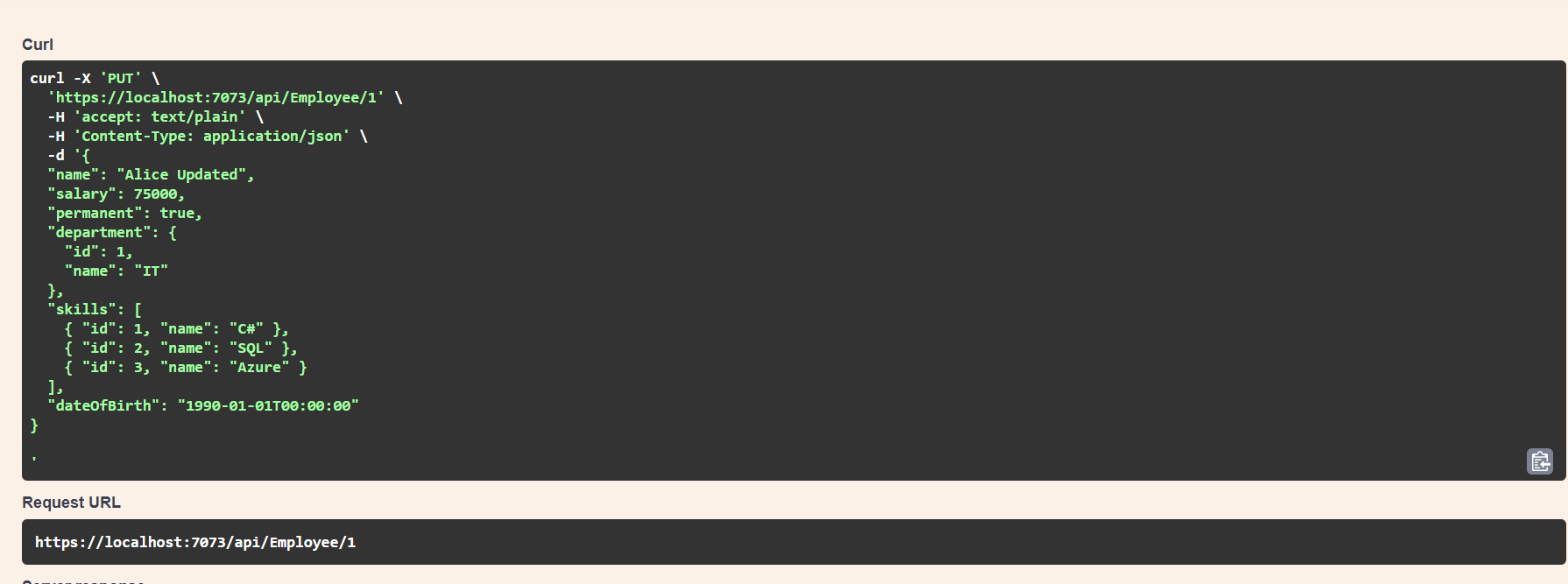
};

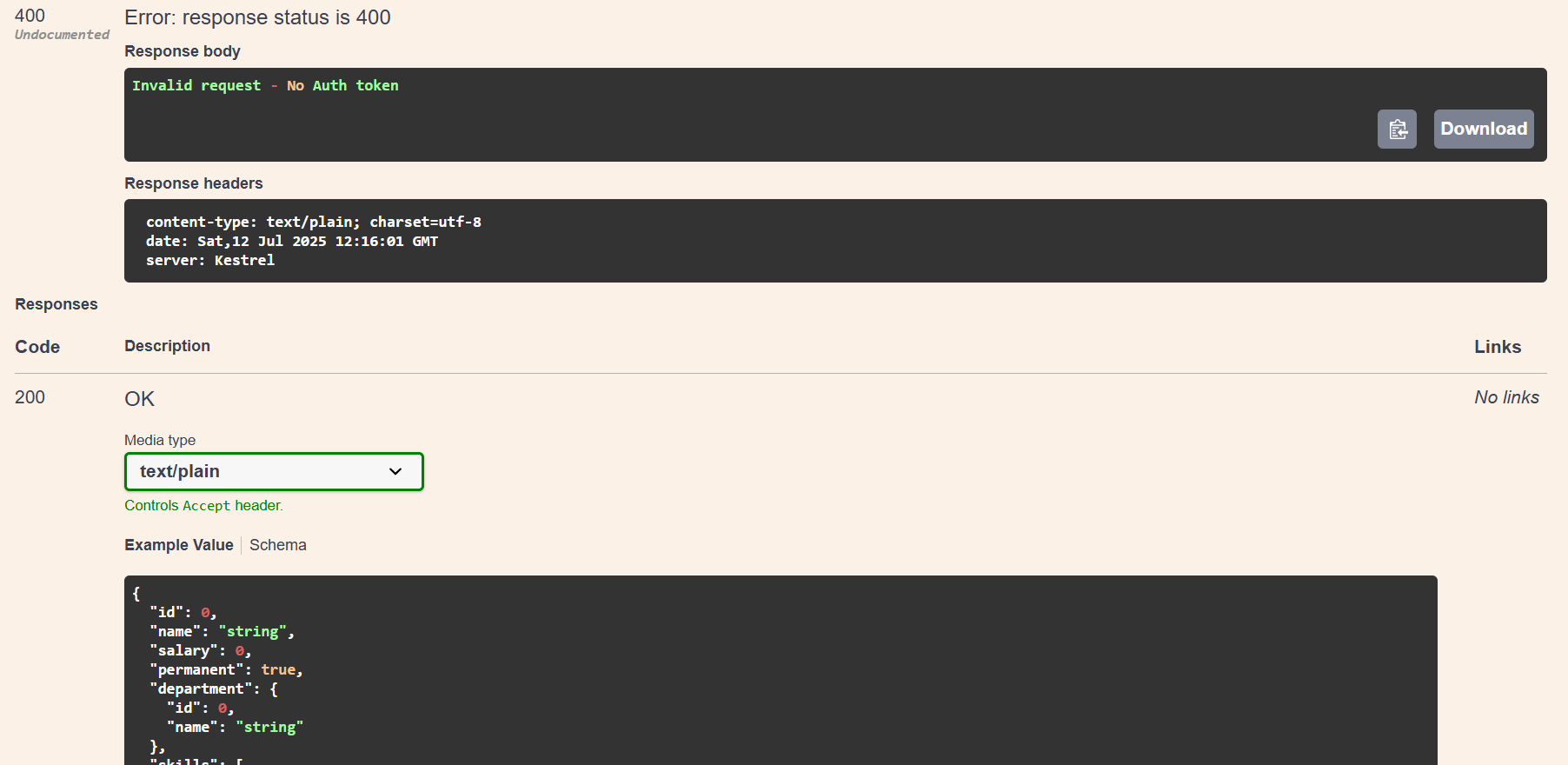
}

}

}

Output:



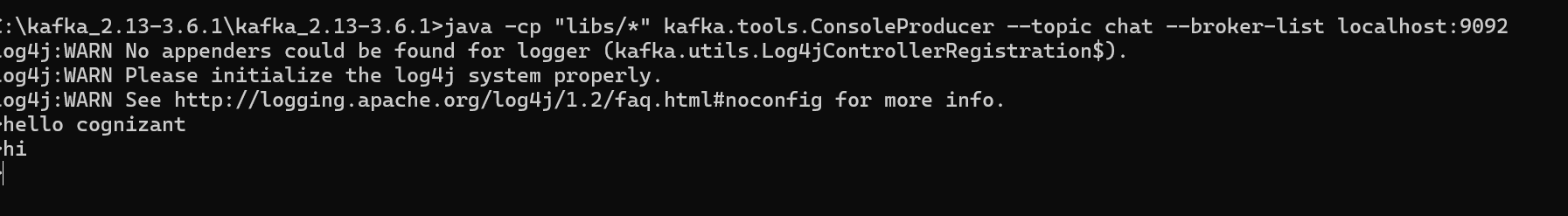


**6. Kafka Integration with C#:**

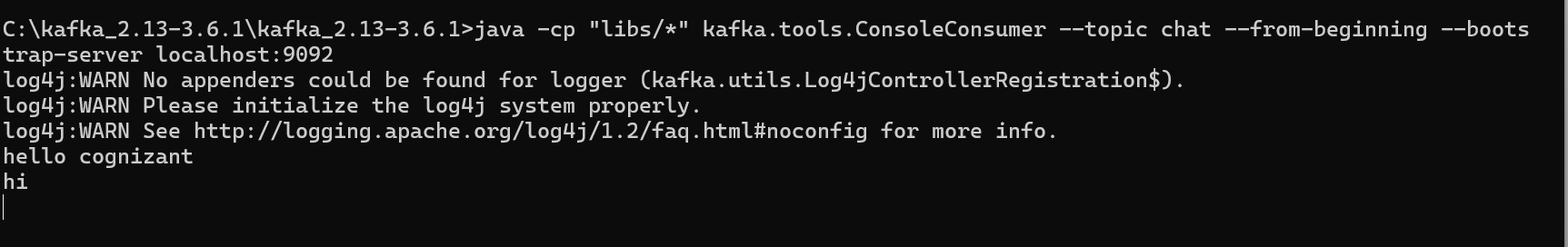
1.Create a Chat Application which uses Kafka as a streaming platform and consume the chat messages in the command prompt.

**Output:**

**Producer window:**



**Consumer window:**



1. Create a Chat Application using C# Windows Application using Kafka and consume the message in different client applications.

**Output:**

