WEEK\_4-Handson

Rajesh A

6384182

Lab 1: Building First Web API in .NET 9.0

# Objectives

- Explain RESTful web services, Web API, and Microservices  
- REST architecture features: Stateless, Representational State Transfer, JSON/XML messages  
- Difference between WebService & WebAPI  
- Understand HttpRequest & HttpResponse  
- Learn Http action verbs: GET, POST, PUT, DELETE and their usage in WebAPI  
- Learn about HttpStatusCodes: OK, BadRequest, InternalServerError, Unauthorized  
- Build a simple Web API with Read/Write actions  
- Understand configuration files like appSettings.json, launchSettings.json

# Step 1: Create Project

dotnet new webapi -n FirstWebApi  
cd FirstWebApi  
code .

# 

# Step 2: Add BooksController.cs

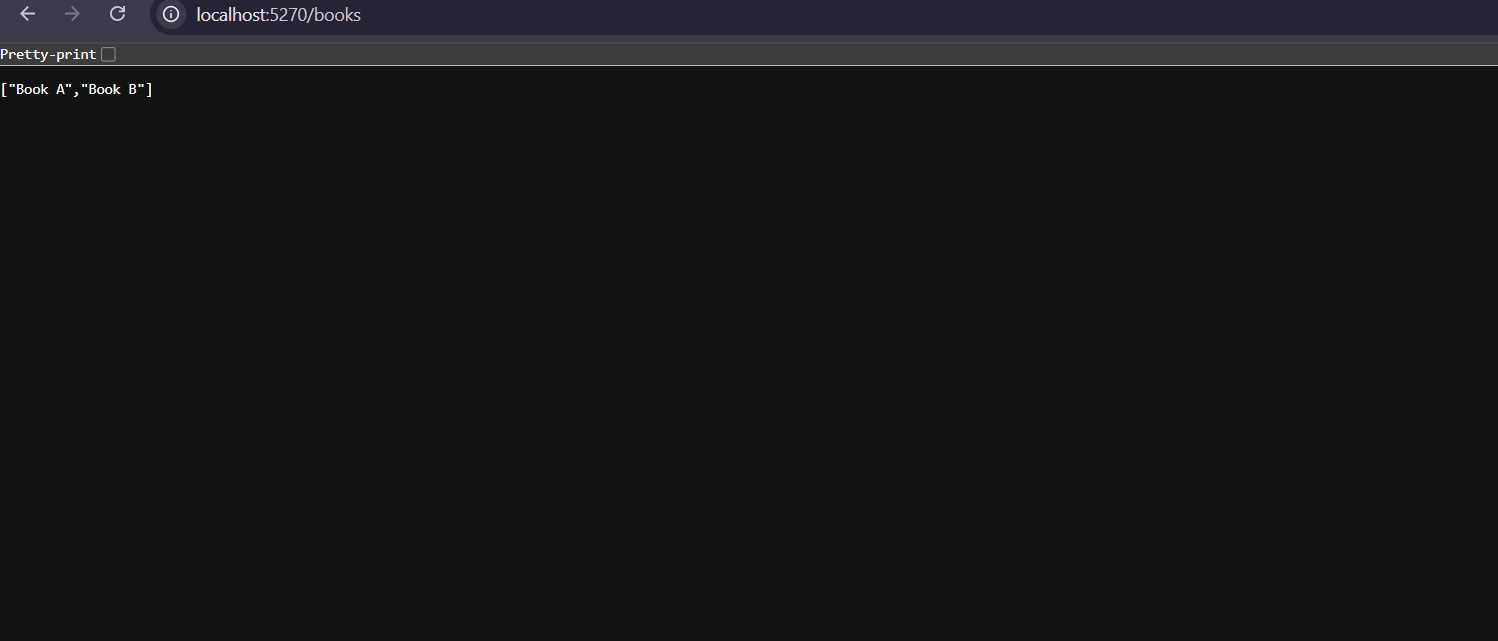
using Microsoft.AspNetCore.Mvc;  
  
namespace FirstWebApi.Controllers;  
  
[ApiController]  
[Route("books")]  
public class BooksController : ControllerBase  
{  
 static List<string> books = new() { "Book A", "Book B" };  
  
 [HttpGet]  
 public ActionResult<IEnumerable<string>> GetBooks() => Ok(books);  
  
 [HttpPost]  
 public ActionResult AddBook([FromBody] string book)  
 {  
 books.Add(book);  
 return Ok("Book added");  
 }  
  
 [HttpPut("{index}")]  
 public ActionResult UpdateBook(int index, [FromBody] string book)  
 {  
 if (index < 0 || index >= books.Count) return BadRequest("Invalid index");  
 books[index] = book;  
 return Ok("Book updated");  
 }  
  
 [HttpDelete("{index}")]  
 public ActionResult DeleteBook(int index)  
 {  
 if (index < 0 || index >= books.Count) return NotFound("Invalid index");  
 books.RemoveAt(index);  
 return Ok("Book deleted");  
 }  
}

# Step 3: Modify Program.cs

var builder = WebApplication.CreateBuilder(args);  
  
builder.Services.AddControllers();  
builder.Services.AddEndpointsApiExplorer();  
var app = builder.Build();  
  
app.UseHttpsRedirection();  
app.UseAuthorization();  
app.MapControllers();  
  
app.Run();

# Step 4: Run and Test in Browser

dotnet run → Visit <https://localhost:PORT/books>



# Theory and Concepts

RESTful Web Services:

REST (Representational State Transfer) is an architectural style used for designing networked applications. It uses HTTP methods like GET, POST, PUT, DELETE.   
Key features:  
- Stateless communication  
- Uses standard HTTP verbs  
- Can return JSON, XML, or any other format  
- Scalability and simplicity

Web API vs WebService:

- WebService uses SOAP protocol, Web API uses REST.  
- Web API can return JSON, XML; WebService is mostly XML.  
- Web API is lightweight, better for modern web apps.

Microservices Concept:

A microservice is a small, independently deployable service that performs a specific function and communicates with other microservices via lightweight protocols like HTTP or messaging queues.

HttpRequest & HttpResponse:

- HttpRequest: Contains info from client → server (headers, method, body, etc.)  
- HttpResponse: Contains info server → client (status code, body, etc.)

Action Verbs in WebAPI:

- [HttpGet]: Used to retrieve data.  
- [HttpPost]: Used to create new resource.  
- [HttpPut]: Used to update resource.  
- [HttpDelete]: Used to delete resource.  
These are declared as attributes above action methods in controllers.

HTTP Status Codes:

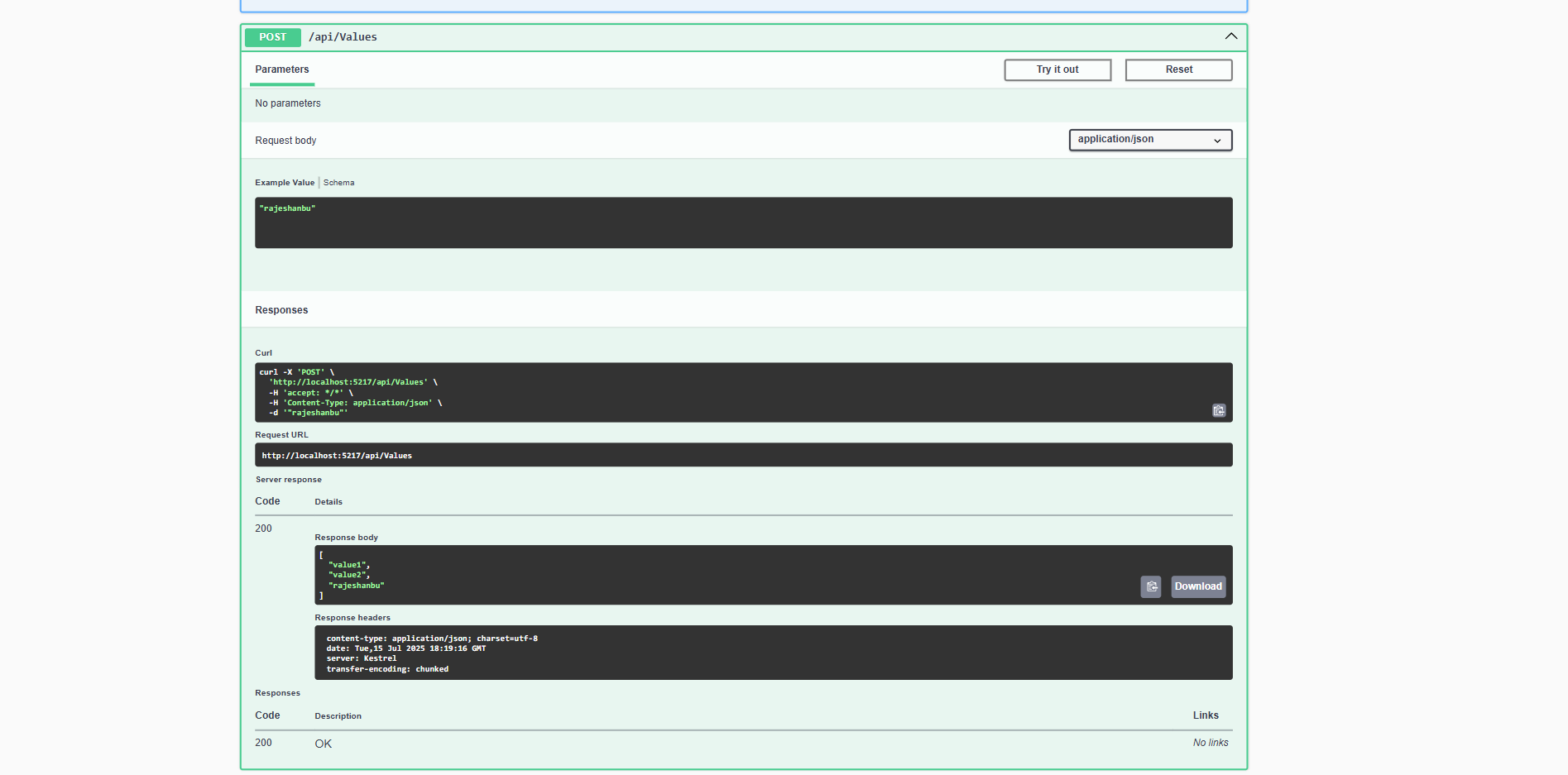
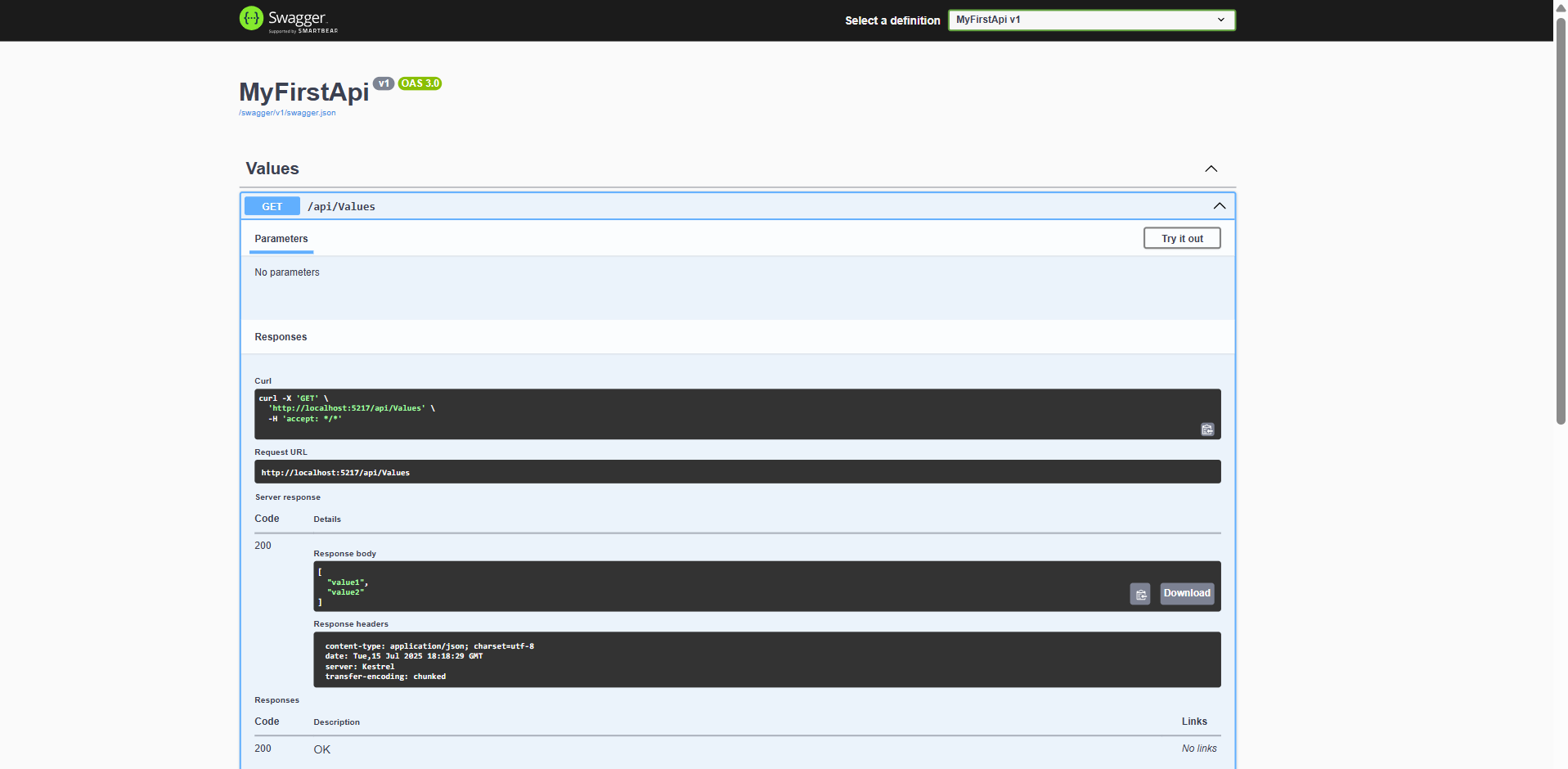
- 200 OK: Request successful.  
- 400 BadRequest: Client sent invalid data.  
- 401 Unauthorized: Not authenticated.  
- 500 InternalServerError: Server crashed.

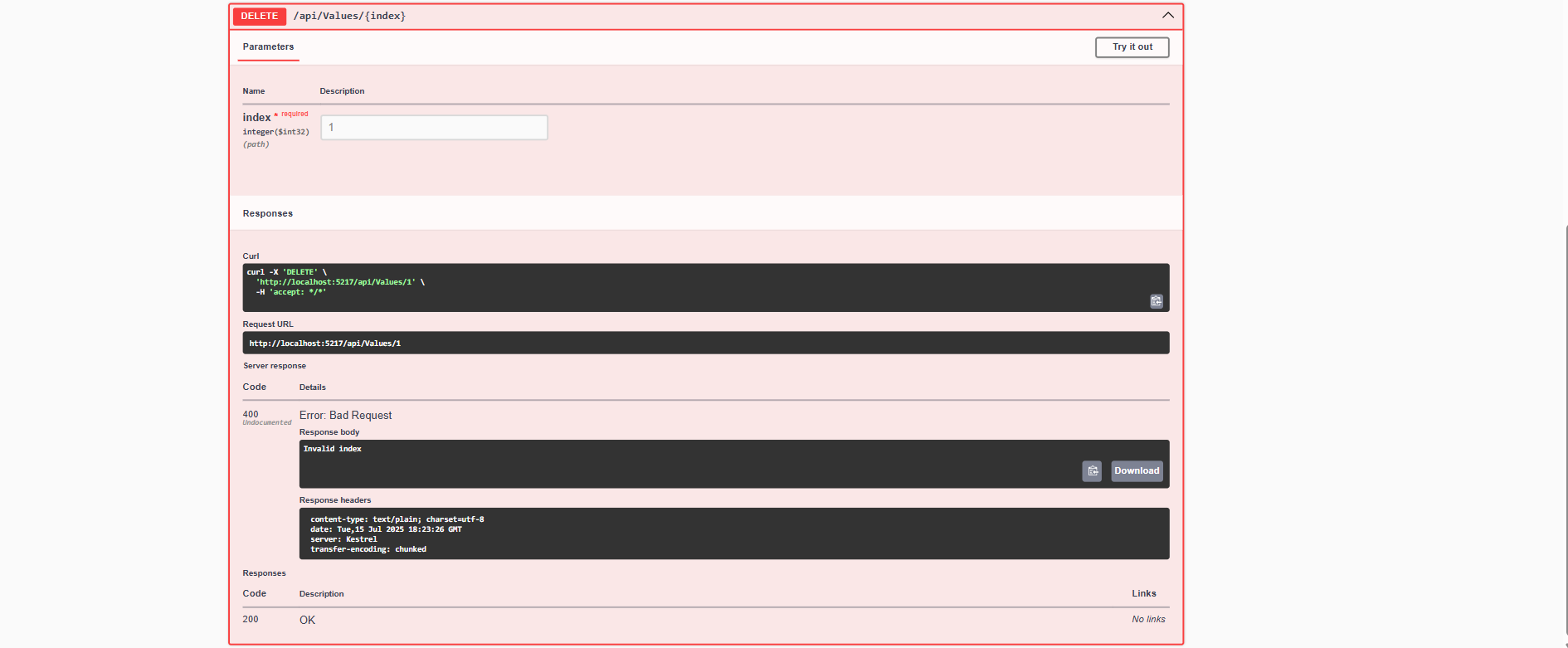
Configuration Files in WebAPI:

- appSettings.json: For custom settings, connection strings, etc.  
- launchSettings.json: Holds launch profile info for development.  
- Program.cs / Startup.cs: Registers services, routing, DI.  
- Web.config / Route.config: Used in .NET Framework (4.x) only.

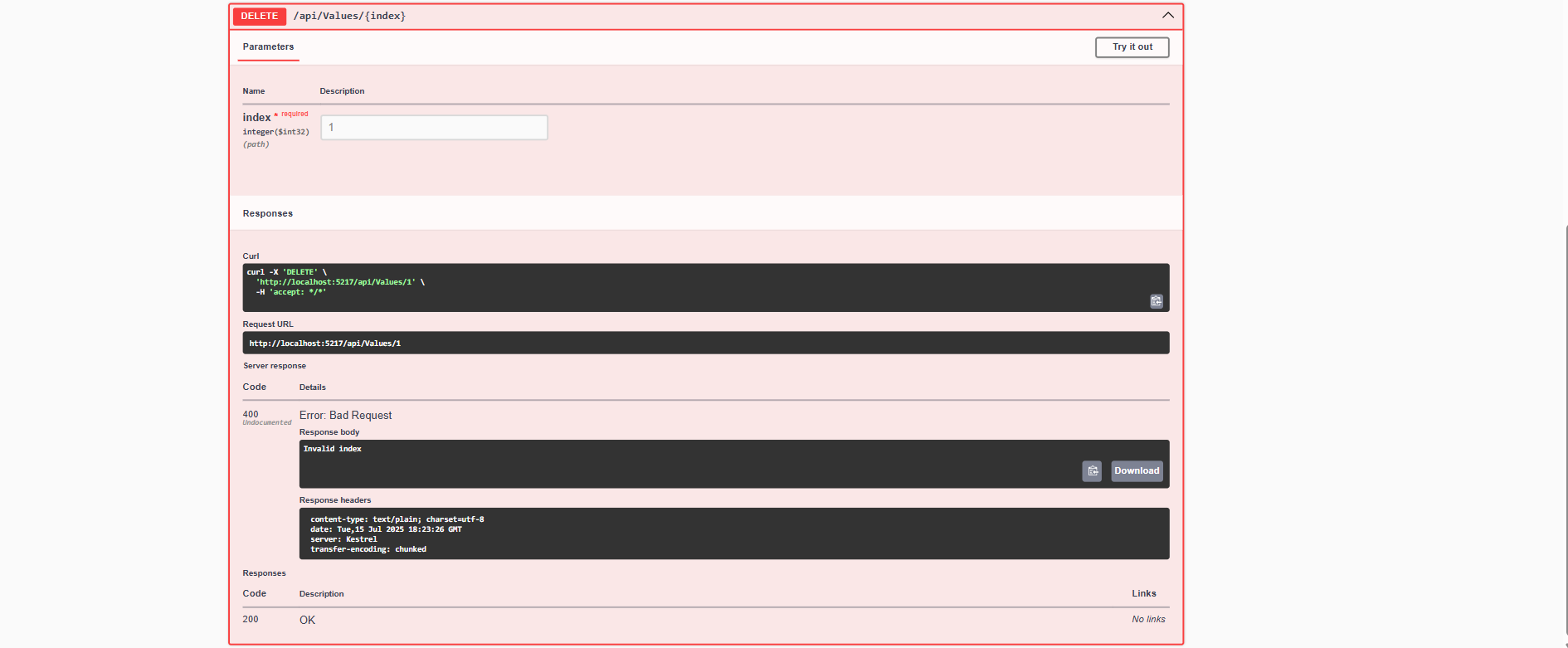
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Output: by swagger

1. Post:
2. Get
3. Delete



4.Put



Thank you