
Task 1: First Web API Using .NET Core

Objectives and Concepts

1. RESTful Web Service

A RESTful Web Service is based on the principles of REST (Representational State Transfer). It uses standard HTTP methods to perform CRUD operations (Create, Read, Update, Delete) on resources.

Features of REST Architecture:

- **Stateless Communication:** Each request from the client to the server must contain all necessary information.
 - **Uniform Interface:** Uses standard HTTP verbs (GET, POST, PUT, DELETE).
 - **Resource-Based:** Operates on resources (URI identifies resources).
 - **Representation:** Response can be in XML, JSON, or plain text, but JSON is widely used.
 - **Microservices Friendly:** Supports building modular services which can be independently deployed.
-

2. Web API vs Web Service

Feature	Web API	Web Service
Protocol	HTTP/HTTPS	SOAP/HTTP
Format	JSON, XML, plain text	Mostly XML
Lightweight	Yes	No
Platform Dependency	Platform Independent	Platform Dependent
Modern Use	Common in REST, Microservices	Used in legacy applications

3. HttpRequest & HttpResponseMessage

- **HttpRequest:** Represents the client's request. Contains headers, method, URI, and body.
 - **HttpResponse:** Represents the response sent to the client. Includes status code, headers, and response body.
-

4. Action Verbs in Web API

HTTP Verb	Description	Attribute in Web API
GET	Retrieve data	[HttpGet]
POST	Add new data	[HttpPost]
PUT	Update existing data	[HttpPut]
DELETE	Delete data	[HttpDelete]

5. HTTP Status Codes in Web API

Status Code	Meaning	Usage in ActionResult
200 OK	Success	<code>return Ok();</code>
400 BadRequest	Invalid client request	<code>return BadRequest();</code>
401 Unauthorized	Authorization header/token missing	Auto from [Authorize]
500 Server Error	Unhandled exception	Custom or global filter

6. Configuration Files in .NET Core Web API

File	Description
Program.cs	Contains app startup logic, dependency injection, routing.\nIn .NET 6+, replaces <code>Startup.cs</code> .
appSettings.json	Stores configuration values like connection strings, tokens, etc.
launchSettings.json	Defines profiles for running the project (IIS Express, ports, etc.)
WebApi.config / Route.config	Used in .NET Framework (pre-Core) to define routing and settings. Not used in .NET Core.

Implementation

1. Project Creation

- Create new project in Visual Studio:
- Template: ASP.NET Core Web API
- Framework: .NET 6 or .NET 7

2. Auto-Generated ValuesController

```
[ApiController]
[Route("[controller]")]
public class ValuesController : ControllerBase
{
    private static List<string> values = new() { "value1", "value2" };

    [HttpGet]
    public IEnumerable<string> Get() => values;

    [HttpPost]
    public IActionResult Post([FromBody] string value)
    {
        values.Add(value);
        return Ok(value);
    }
}
```

- Inherits from ControllerBase
 - Uses [HttpGet], [HttpPost] to map HTTP methods
 - Get() returns list, Post() adds a value
-

3. Run and Test Application

- Run project using `dotnet run` or Visual Studio
 - Open browser:
 - `http://localhost:<port>/swagger`
 - Execute GET method in Swagger UI.
-

Screenshot Placeholders

1. Swagger UI showing `ValuesController` with GET and POST
 2. Output of GET method execution in Swagger (200 OK)
-

Screenshots

Swagger UI

localhost:5192/swagger/index.html

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Swagger
OpenAPI 3.0

Select a definition: Swagger Demo v1

Swagger Demo ^{v1} ^{0AS.3.0}

/swagger/v1/swagger.json

Minimal + Controller API Swagger Integration with JWT

Adarsh - Website
Send email to Adarsh

Auth

GET /api/Auth

Employee

GET /api/Employee

POST /api/Employee

PUT /api/Employee/{id}

FruitsController1

GET /api/FruitsController1

GET /api/FruitsController1/{index}

Schemas

Department >

Employee >

Skill >

Swagger UI

localhost:5192/swagger/index.html

For quick access, place your bookmarks here on the bookmarks bar. [Import bookmarks now...](#)

Swagger Demo v1

Select a definition: Swagger Demo v1

Auth

GET /api/Auth

Employee

GET /api/Employee

POST /api/Employee

PUT /api/Employee/{id}

FruitsController1

GET /api/FruitsController1

Parameters

No parameters

Execute Clear

Responses

Curl

```
curl -X 'GET' \
  'http://localhost:5192/api/FruitsController1' \
  -H 'accept: */*'
```

Request URL

http://localhost:5192/api/FruitsController1

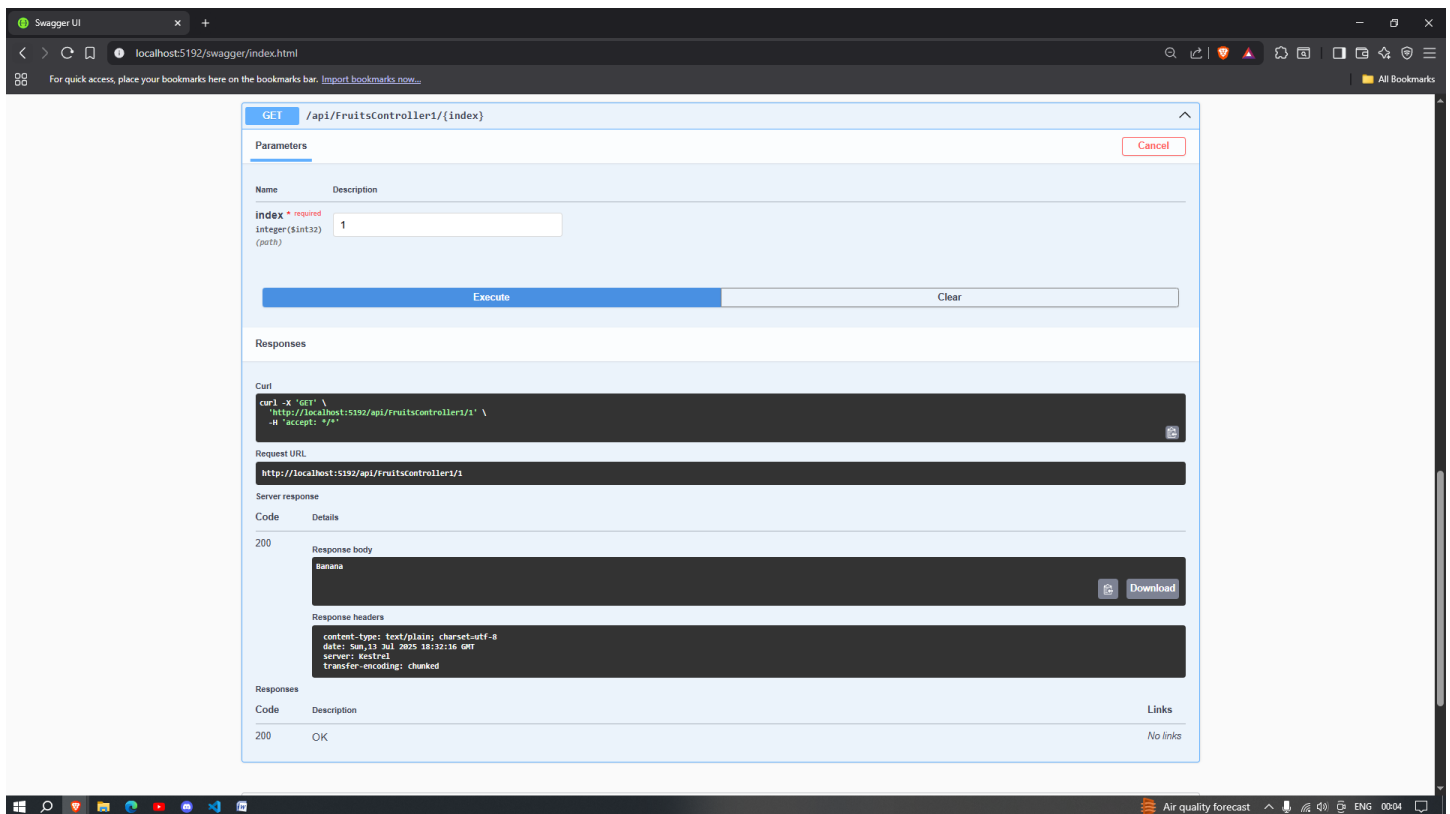
Server response

Code	Details
200	<p>Response body</p> <pre>{ "Apple", "Mango", "Cherry", "Date", "Elderberry" }</pre> <p>Response headers</p> <pre>content-type: application/json; charset=utf-8 date: Sun, 13 Jul 2025 18:33:40 GMT server: Kestrel transfer-encoding: chunked</pre>

Responses

Code	Description	Links
200	OK	No links

GET /api/FruitsController1/{index}



Conclusion

Task 1 introduces the fundamentals of Web API structure, action methods, HTTP verbs, status codes, and REST concepts. It also establishes the ability to test APIs using built-in tools like Swagger.

Let me know if you would like this added to the full report in Word format.