Understanding RESTful Web API in .NET Core

Introduction to RESTful Web Services

REST is an architectural style used for designing distributed systems. It uses standard HTTP methods and operates on resources, which are identified by URIs.

Key Features of REST:
- Stateless: Each request from client to server must contain all necessary information.
- Resource-Based: Everything is considered a resource (e.g., users, products) and is accessed using a URI.
- Uniform Interface: Standardized HTTP methods like GET, POST, PUT, DELETE.
- Multiple Representations: Responses can be in XML, JSON, plain text, etc.
- Client-Server Model: Clear separation of concerns between client and server.

Web Service vs Web API

Web Service vs Web API:
Web Service:
- Uses SOAP
- XML only
- Platform dependent
- Slower

Web API:
- RESTful over HTTP
- JSON/XML/plain text
- Platform independent
- Faster

Microservices Concept

Microservices is a modern architectural approach where an application is broken down into small, loosely coupled services. Each service is independently deployable, scalable, and responsible for a specific business functionality.

Benefits:
- Independent deployment
- Better scalability
- Technology agnostic
- Easier to maintain and test

HttpRequest & HttpResponse

HttpRequest:
- Sent from client to server.
- Contains method (GET, POST, etc.), headers, query parameters, body, and URI.
- Example: GET /api/products/5

HttpResponse:
- Sent from server to client.
- Contains status code, headers, and response body (data).
- Example:

```
{
  "id": 5,
  "name": "Laptop",
  "price": 45000
}
```

HTTP Action Verbs

HTTP verbs indicate the desired action to be performed on a resource.

GET – Retrieve data
POST – Create new data
PUT – Update existing data
DELETE – Remove data

Used as attributes like [HttpGet], [HttpPost], etc. in controller methods.

HTTP Status Codes in Web API

Common status codes:
200 OK – Successful request
400 Bad Request – Invalid input
401 Unauthorized – Authentication required
500 Internal Server Error – Server-side issue

Used in Web API as: return Ok(), return BadRequest(), return Unauthorized(), etc.

Structure of a Web API in .NET Core

A controller handles HTTP requests and returns responses. It inherits from ControllerBase or ApiController.

Example:

```
[ApiController]
```

```
[Route("api/[controller]")]
public class ProductsController : ControllerBase
{
   [HttpGet]
   public IActionResult GetAll() { ... }

   [HttpPost]
   public IActionResult Create(Product p) { ... }
}
```

Configuration Files in ASP.NET Core Web API

Startup.cs / Program.cs – Configures middleware, routing, and services via DI
appsettings.json – Stores configuration values like connection strings
launchSettings.json – Defines ports, profiles, and environments
WebApiConfig.cs / RouteConfig.cs – Used in .NET Framework (not Core)

Creating a Simple Web API in .NET Core

Steps:
1. Open Visual Studio > New Project > ASP.NET Core Web API
2. Use template to generate ValuesController.cs
3. Run the project to see Swagger UI
4. Test endpoints (GET, POST, PUT, DELETE)

Sample GET Response:

```
[
  "value1",
  "value2"
]
```