**Name: Deepanshu Rathore**

**Superset ID: 6358199**  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
  
Week 4  
**1. WebApi\_Handson**A RESTful web service is an architectural style for designing networked applications. It relies on a uniform set of stateless operations, most often mapped to the standard HTTP methods. In REST you work with resources identified by URLs and transfer their representations typically JSON or XML between client and server. The term “representational state transfer” refers to the idea that when a client holds a representation of a resource, it can manipulate that resource’s state via the messages it sends. Because each request contains all the information needed to process it, the server does not need to remember any client context between requests.

A Web API is a way to expose server functionality over HTTP so that other applications can consume it. It often follows REST principles, but it is not limited to XML responses; in modern frameworks JSON is the default. A microservice is a small, independently deployable service that implements a single business capability. You can build a microservice as a Web API, but you might also choose other styles of communication. In contrast, a traditional WebService (for example using SOAP) relies on a strict XML envelope and often requires a service description in WSDL. WebAPI tends to be more lightweight, more flexible in payload formats, and easier to consume from a variety of clients.

Every interaction in a Web API happens through an HttpRequest and an HttpResponse. An HttpRequest carries metadata such as the method, headers, URL, query parameters, and often a body. The HttpResponse carries a status code, headers, and possibly a body. The status code tells the client whether the request succeeded or failed, and the body contains the data or error details.

In a RESTful API you use action verbs that correspond to HTTP methods. HttpGet retrieves resources without side effects. HttpPost creates new resources or triggers operations with side effects. HttpPut replaces or updates resources. HttpDelete removes resources. In most frameworks you decorate your controller methods with attributes like [HttpGet], [HttpPost], [HttpPut], and [HttpDelete] so that the framework routes the correct method based on the incoming request.

When your controller action returns a result, you use standard HTTP status codes. Returning Ok (200) signals success and often includes data. BadRequest (400) indicates that the client’s input was invalid. Unauthorized (401) tells the client it must authenticate before proceeding. InternalServerError (500) shows that something went wrong on the server. In code you often return types like IActionResult or ActionResult<T> and then call methods such as Ok(), BadRequest(), Unauthorized(), or StatusCode(500) to produce the appropriate response.

A simple Web API consists of a controller class that inherits from ControllerBase (or ApiController in older versions). Each public method in the controller is an action method, annotated with its HTTP verb attribute. For example, a GetAll method might use [HttpGet] and return a list of items. A Create method might use [HttpPost] and accept an object in the request body. The framework handles routing based on conventions or explicit route attributes, matches incoming requests to these methods, and serializes the responses.ValueController.cs:  
using Microsoft.AspNetCore.Mvc;

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

[ApiController]

public class ValuesController : ControllerBase

{

private static readonly List<string> \_values = new() { "value1", "value2" };

[HttpGet]

public ActionResult<IEnumerable<string>> Get()

{

return Ok(\_values);

}

[HttpPost]

public ActionResult Post([FromBody] string value)

{

if (string.IsNullOrWhiteSpace(value))

return BadRequest("Value cannot be empty");

\_values.Add(value);

return Ok();

}

}

**Output:**

A screenshot of a computer

AI-generated content may be incorrect.