**1 WEBAPI HANDSON**   
   
**Objectives:**

* Explain the concept of RESTful web service, Web API & Microservice
* Features of REST architecture - Representational State Transfer, Stateless, Messages, Concept of Microservice, Difference between WebService & WebAPI, Not restricted to send XML as response
* Explain what is HttpRequest & HttpResponse
* List the types of Action Verbs
* HttpGet, HttpPost, HttpPut, HttpDelete - Meaning of action verbs and how that should be declared as attributes for Web API
* List the types of HttpStatusCodes used in WebAPI
* Ok, InternalServerError, Unauthorized, BadRequest - All thru the action result types
* Demonstrate creation of a simple WebAPI - With Read, Write actions
* Structure of a web api - Controller & its inheritance from ApiController, Action verbs, Action method
* Explain the types of Configuration files of WebAPI
* Startup.cs with depdency injection, appSettings.json, launchSettings.json, Explain Route.config & WebAPI.config in .Net 4.5 framework

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.

**SOLUTION :**

**1. Objective**

To create and test a basic ASP.NET Core Web API using Microsoft Visual Studio 2022 with support for RESTful services and Swagger UI.

**2. Key Concepts Covered**

- RESTful architecture   
- Web API structure (Controllers, Models)   
- HTTP request and response   
- Action verbs: GET, POST, PUT, DELETE   
- HTTP status codes: 200 OK, 400 Bad Request, etc.   
- Testing API with Swagger UI 

**3. Tools Used**

|  |  |
| --- | --- |
| Tool | Version |
| Visual Studio | 2022 |
| .NET SDK | 6.0 or 7.0 |
| Browser | Edge / Chrome |
| Swagger | OpenAPI 3.0 |

**4. Step-by-Step Implementation**

**Step 1: Create ASP.NET Core Web API Project**

- Selected ASP.NET Core Web API template   
- Enabled Controllers (not Minimal API)   
- Enabled Swagger/OpenAPI support 

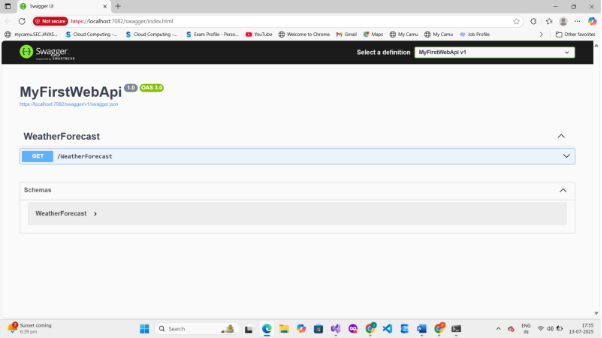
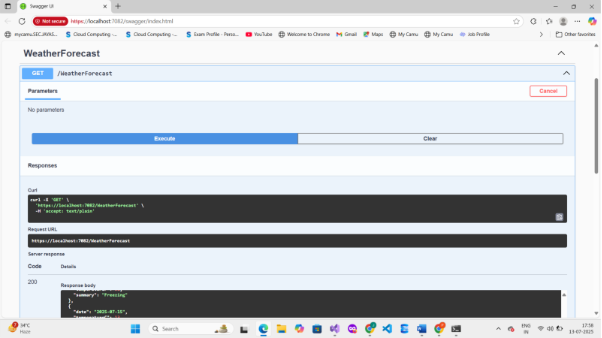
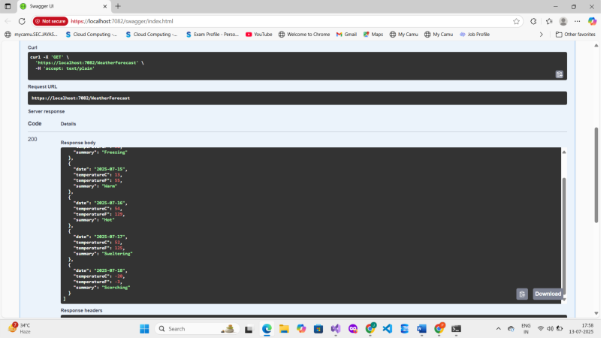
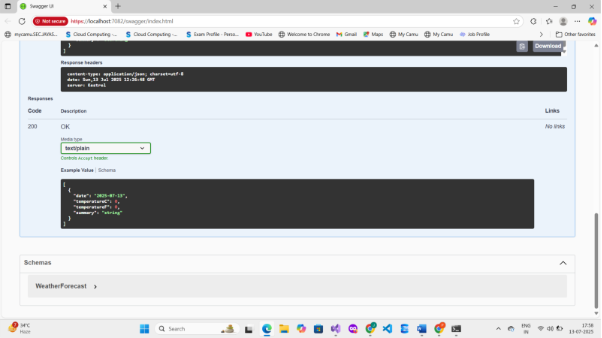
**Step 2: Project Structure**

- Created project named MyFirstWebApi   
- Auto-generated files:   
  - WeatherForecast.cs   
  - WeatherForecastController.cs   
  - Program.cs   
  - appsettings.json   
  - launchSettings.json 

**Step 3: Run and Test the API**

- Pressed F5 to start debugging   
- Swagger UI opened at:   
  https://localhost:7082/swagger/index.html   
   
- Clicked GET /WeatherForecast   
- Clicked Try it out > Execute   
- API returned sample data with status 200 OK 

**5. Output**

**6. Conclusion**

The Web API was successfully created, executed, and tested using Swagger UI.    
This exercise demonstrated the use of HTTP GET verbs, RESTful routing, and integration of Swagger in a .NET Core Web API project.  

**2.WEB API HANDSON**   
   
**Title:**

Swagger, Postman, and Routing in ASP.NET Core Web API

**1. Objectives**

List the objectives clearly:

• Demonstrate Swagger installation and listing of WebAPI methods in the browser

• Use Postman to test WebAPI methods (GET, POST)

• Modify Route attribute and use ActionName in HTTP methods

• Create a simple WebAPI with Read and Write actions

**2. Tools & Technologies Used**

|  |  |
| --- | --- |
| **Tool** | **Version** |
| Visual Studio | 2022 |
| .NET SDK | 6.0 / 7.0 |
| Postman | Latest |
| Swagger (OpenAPI) | Swashbuckle.AspNetCore |

**3. Swagger Setup**

**a. NuGet Package Installed:**

Swashbuckle.AspNetCore

**b. Code Added in Program.cs / Startup.cs**

**In ConfigureServices:**

services.AddSwaggerGen(c =>

{

    c.SwaggerDoc("v1", new OpenApiInfo

    {

        Title = "Swagger Demo",

        Version = "v1",

        Description = "TBD",

        TermsOfService = new Uri("https://example.com"),

        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") }

    });

});

**In Configure:**

app.UseSwagger();

app.UseSwaggerUI(c =>

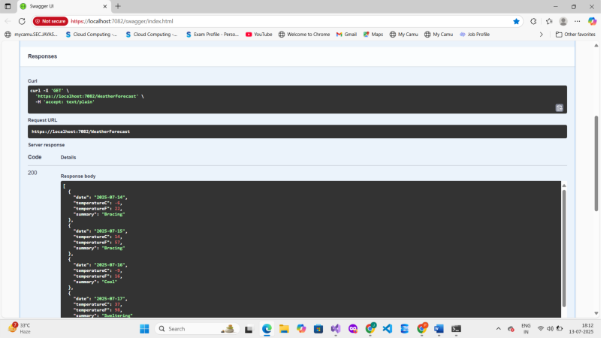
{

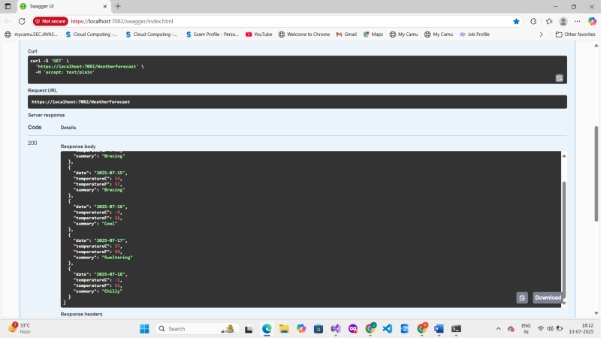
    c.SwaggerEndpoint("/swagger/v1/swagger.json", "Swagger Demo");

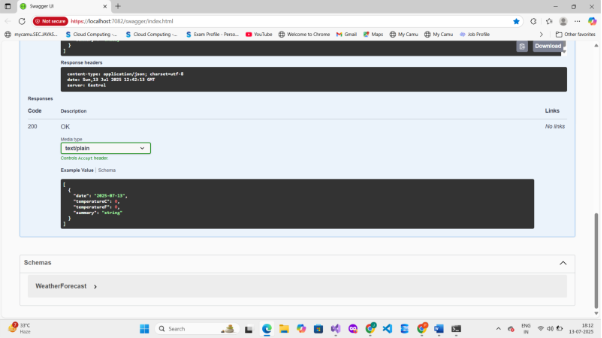
});

**4. Swagger Output**

* URL:   
  https://localhost:[port]/swagger
* Title: Swagger Demo
* Contact Info: John Doe, john@xyzmail.com
* Controller listed: ValuesController, EmployeeController
* Methods: GET, POST, etc.







**5. Testing Web API Using Postman**

**a. Steps Performed:**

* Method: GET
* URL: https://localhost:[port]/api/Employee
* Headers: Content-Type: application/json (if POST)
* Clicked **Send**

**6. Route Customization**

**a. Modified Route Attribute**

[Route("api/Emp")]

**b. Tested New URL in Postman**

https://localhost:[port]/api/Emp

✅ Output still worked correctly

**7. Use of [ActionName] and [ProducesResponseType]**

Explain:

* [ActionName("GetById")] allows same HTTP verb for different methods
* [ProducesResponseType(typeof(Employee), 200)] specifies response model and code

Example:

[HttpGet("{id}")]

[ActionName("GetById")]

[ProducesResponseType(typeof(Employee), 200)]

public IActionResult Get(int id) { ... }

**8. Conclusion**

Swagger was successfully integrated and listed all WebAPI actions.

Postman tested WebAPI methods with valid responses.

Routing was customized, and ActionName and response types were demonstrated.

**3.WebAPI  Handson**

Web API – Employee CRUD with Custom Filters

**1. Objective**

• Demonstrate creation of an Action method to return list of custom class entity.   
• Use AllowAnonymous attribute, HttpGet, and FromBody attribute.   
• Demonstrate Custom Filters using ActionFilterAttribute and Exception Filter. 

**2. Tools Used**

• Microsoft Visual Studio 2022   
• ASP.NET Core Web API (.NET 6 or 7)   
• Swagger UI for API Testing 

**3. Model Classes**

Employee.cs

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; }   
} 

Department.cs

public class Department   
{   
    public int Id { get; set; }   
    public string? Name { get; set; }   
} 

Skill.cs

public class Skill   
{   
    public int Id { get; set; }   
    public string? Name { get; set; }   
} 

**4. EmployeeController.cs**

[ApiController]   
[Route("api/[controller]")]   
[TypeFilter(typeof(CustomAuthFilter))]   
public class EmployeeController : ControllerBase   
{   
    private static List<Employee> employees = new List<Employee>();   
   
    public EmployeeController()   
    {   
        if (!employees.Any())   
        {   
            employees = GetStandardEmployeeList();   
        }   
    }   
   
    private List<Employee> GetStandardEmployeeList()   
    {   
        return new List<Employee>   
        {   
            new Employee   
            {   
                Id = 1,   
                Name = "Alice",   
                Salary = 60000,   
                Permanent = true,   
                Department = new Department { Id = 1, Name = "IT" },   
                Skills = new List<Skill> { new Skill { Id = 1, Name = "C#" } },   
                DateOfBirth = new DateTime(1990, 5, 23)   
            }   
        };   
    }   
   
    [HttpGet("standard")]   
    [ProducesResponseType(typeof(List<Employee>), 200)]   
    [ProducesResponseType(500)]   
    public ActionResult<List<Employee>> GetStandard()   
    {   
        throw new Exception("Sample exception for testing");   
    }   
} 

**5. CustomAuthFilter.cs**

public class CustomAuthFilter : ActionFilterAttribute   
{   
    public override void OnActionExecuting(ActionExecutingContext context)   
    {   
        var headers = context.HttpContext.Request.Headers;   
        if (!headers.ContainsKey("Authorization"))   
        {   
            context.Result = new BadRequestObjectResult("Invalid request - No Auth token");   
            return;   
        }   
   
        var token = headers["Authorization"].ToString();   
        if (!token.Contains("Bearer"))   
        {   
            context.Result = new BadRequestObjectResult("Invalid request - Token present but Bearer unavailable");   
            return;   
        }   
   
        base.OnActionExecuting(context);   
    }   
} 

**6. CustomExceptionFilter.cs**

public class CustomExceptionFilter : IExceptionFilter   
{   
    public void OnException(ExceptionContext context)   
    {   
        var exception = context.Exception;   
        var path = Path.Combine(Directory.GetCurrentDirectory(), "logs.txt");   
        File.AppendAllText(path, $"[{DateTime.Now}] {exception.Message}{Environment.NewLine}");   
   
        context.Result = new ObjectResult("An error occurred. Please try again later.")   
        {   
            StatusCode = 500   
        };   
    }   
} 

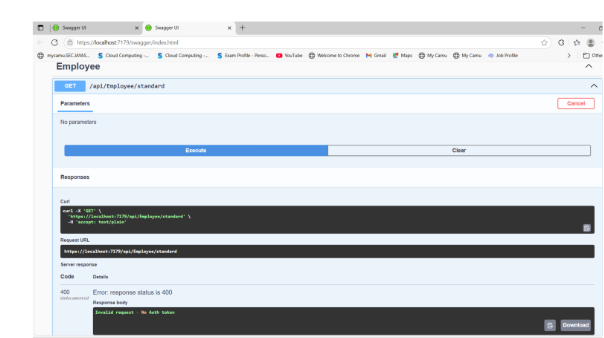
**7. Program.cs Changes**

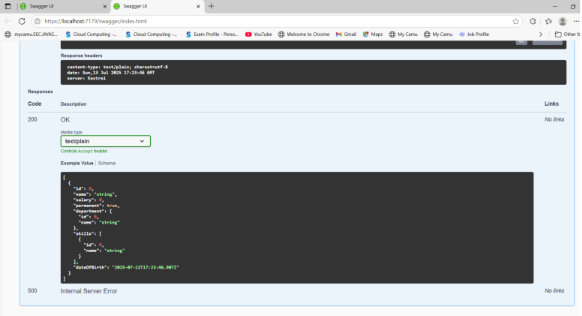
builder.Services.AddControllers(options =>   
{   
    options.Filters.Add(typeof(CustomExceptionFilter));   
});   
   
app.UseSwagger();   
app.UseSwaggerUI();   
app.UseHttpsRedirection();   
app.UseAuthorization();   
app.MapControllers(); 

**8. Testing in Swagger**

• Run the project. Swagger UI opens at https://localhost:xxxx/swagger.   
• Try GET /api/Employee/standard → Throws exception and logs to logs.txt.   
• Try POST /api/Employee without Authorization header → Returns "Invalid request - No Auth token".   
• Click "Authorize" and enter "Bearer token" → Then POST will work. 

**OUTPUT :**





                                    4.WebAPI Handson

**1. Objective:**

Demonstrate how to update employee data using the HTTP PUT method in ASP.NET Core Web API. The update should validate the employee ID and return proper responses. The data is extracted from the request body using [FromBody], and Swagger is used to test the endpoint.

**2. Tools Used:**

* Visual Studio 2022
* .NET 6/7 Core Web API
* Swagger (for API testing)
* C# Programming Language

**3. Employee Model Code (Models/Employee.cs):**

namespace EmployeeApi.Models

{

    public class Employee

    {

        public int Id { get; set; }

        public string? Name { get; set; }

        public string? Department { get; set; }

        public double Salary { get; set; }

    }

}

**4. Controller Code (Controllers/EmployeeController.cs):**

using Microsoft.AspNetCore.Mvc;

using EmployeeApi.Models;

namespace EmployeeApi.Controllers

{

    [ApiController]

    [Route("api/[controller]")]

    public class EmployeeController : ControllerBase

    {

        private static List<Employee> employees = new List<Employee>

        {

            new Employee { Id = 1, Name = "Alice", Department = "HR", Salary = 50000 },

            new Employee { Id = 2, Name = "Bob", Department = "IT", Salary = 60000 },

            new Employee { Id = 3, Name = "Charlie", Department = "Finance", Salary = 70000 }

        };

        [HttpPut("{id}")]

        public ActionResult<Employee> UpdateEmployee(int id, [FromBody] Employee updatedEmp)

        {

            if (id <= 0)

            {

                return BadRequest("Invalid employee id");

            }

            var existingEmp = employees.FirstOrDefault(e => e.Id == id);

            if (existingEmp == null)

            {

                return BadRequest("Invalid employee id");

            }

            existingEmp.Name = updatedEmp.Name;

            existingEmp.Department = updatedEmp.Department;

            existingEmp.Salary = updatedEmp.Salary;

            return Ok(existingEmp);

        }

    }

}

**5. API Testing via Swagger**

Screenshot the Swagger UI after:

* Opening PUT /api/Employee/{id}
* Clicking **"Try it out"**
* Passing id = 2 and a sample JSON request body like:

{

  "id": 2,

  "name": "Robert",

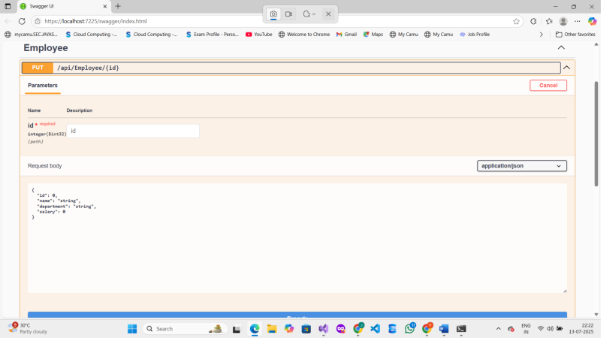
  "department": "IT Support",

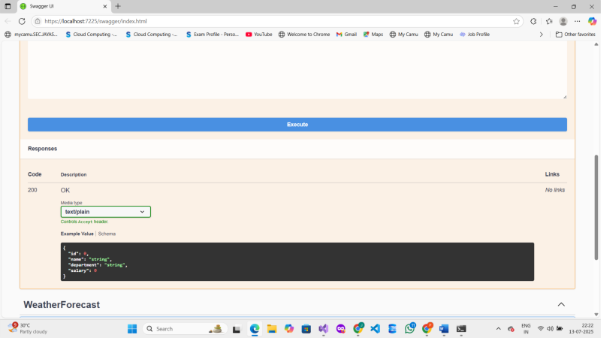
  "salary": 65000

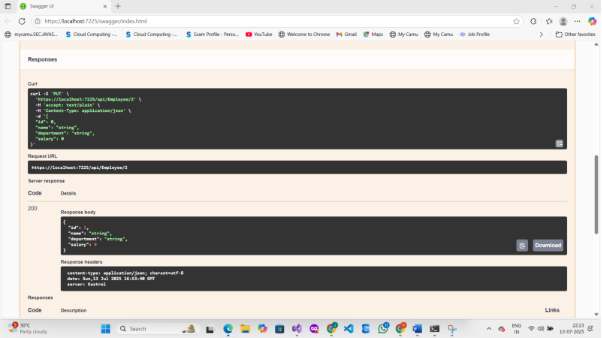
}

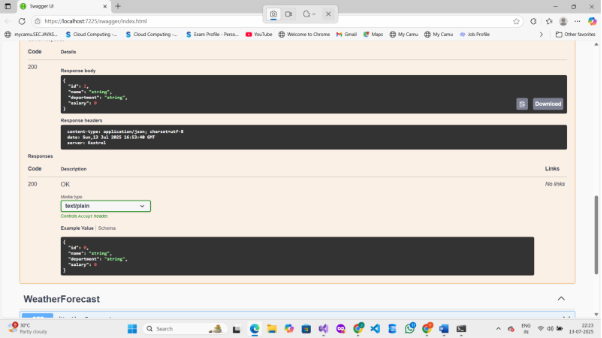
**🔹 6. Validation Scenarios Tested:**

|  |  |
| --- | --- |
| **Scenario** | **Result** |
| ID <= 0 | 400 Bad Request |
| ID not found in list | 400 Bad Request |
| Valid ID with JSON body | 200 OK |









**7. Conclusion:**

The PUT method was successfully implemented to update employee data in a Web API using [FromBody]. Proper validation was added to handle invalid IDs, and the updated employee record was returned. Swagger was used to test all possible scenarios.