**WebApi\_Hansdon**

**lab1.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

**Solution:-**

✅ **RESTful Web Service:**

REST (Representational State Transfer) is an architectural style for networked applications.

Uses **stateless communication** (each request is independent).

Works over HTTP using **methods**: GET, POST, PUT, DELETE.

Uses **JSON/XML for data transfer**, but not restricted to XML.

**WebService vs WebAPI:**

WebService (.asmx) supports only SOAP/XML.

WebAPI supports REST, JSON, XML, and lightweight HTTP calls.

**Microservice:**

Independently deployable, small, modular services communicating over HTTP.

✅ **HttpRequest & HttpResponse:**

**HttpRequest**: carries client data to the server (method, headers, body).

**HttpResponse**: server's reply to the client (status code, headers, body).

✅ **Action Verbs:**

**HttpGet**: Retrieve data.

**HttpPost**: Create data.

**HttpPut**: Update data.

**HttpDelete**: Delete data.

✅ **HttpStatusCodes:**

**Ok (200)**: Request succeeded.

**BadRequest (400)**: Client error, bad input.

**Unauthorized (401)**: Authentication required.

**InternalServerError (500)**: Server-side error.

✅ **WebAPI structure:**

Uses **Controller classes** inheriting from ControllerBase or ApiController.

Action methods decorated with HTTP verb attributes ([HttpGet], [HttpPost], etc).

Routes define URL structure.

✅ **Configuration files:**

**Startup.cs**: Middleware, services, routing configuration.

**appSettings.json**: Application configuration (connection strings, custom configs).

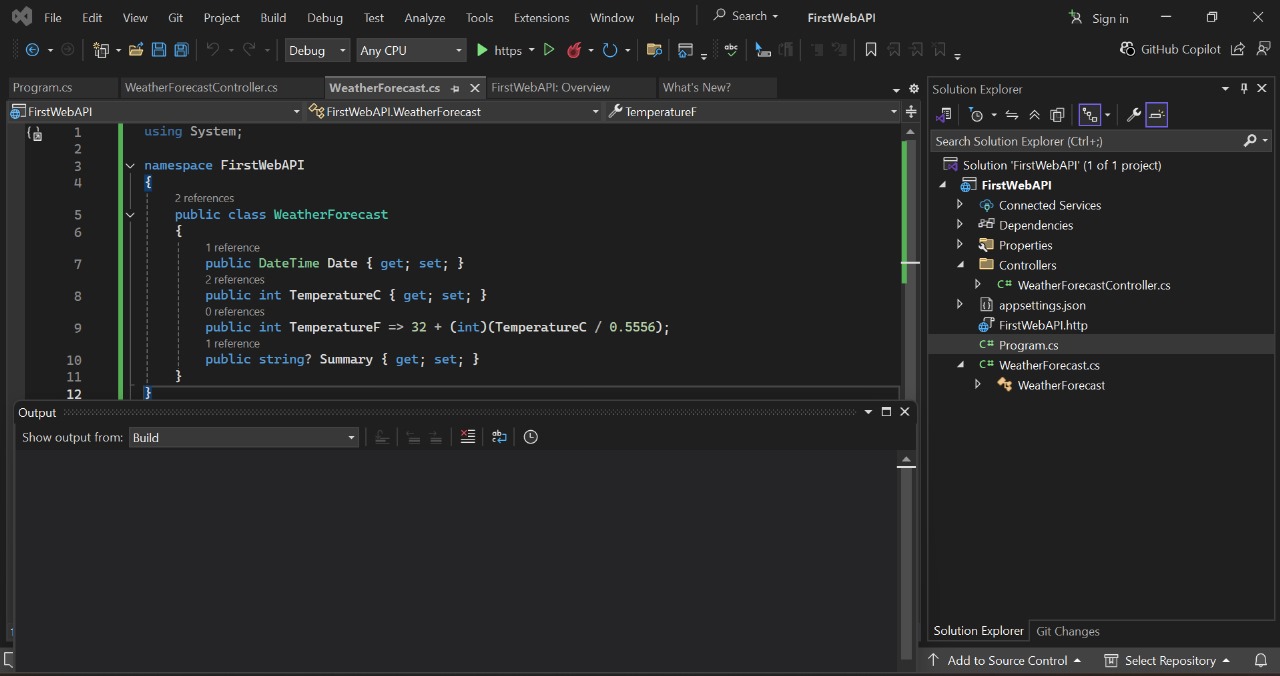
**launchSettings.json**: Debug launch profiles (URL, environment).

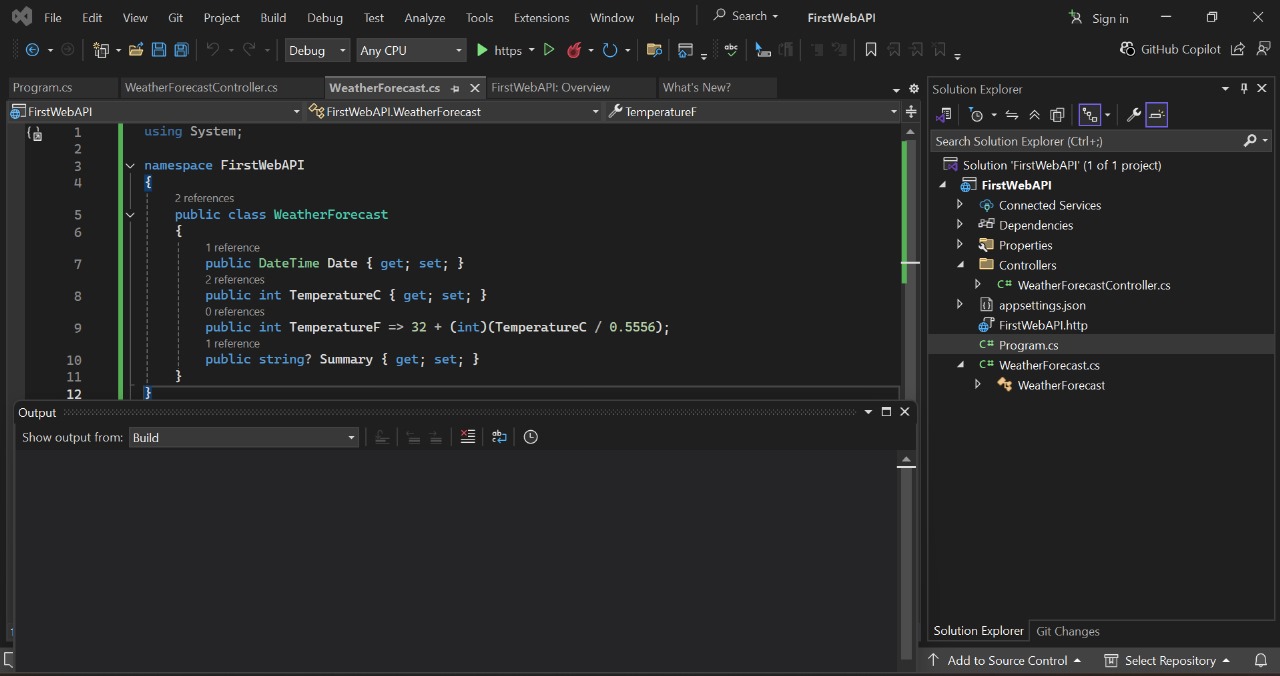
**Route.config, WebAPI.config**: Used in .NET Framework, **not in .NET Core**.

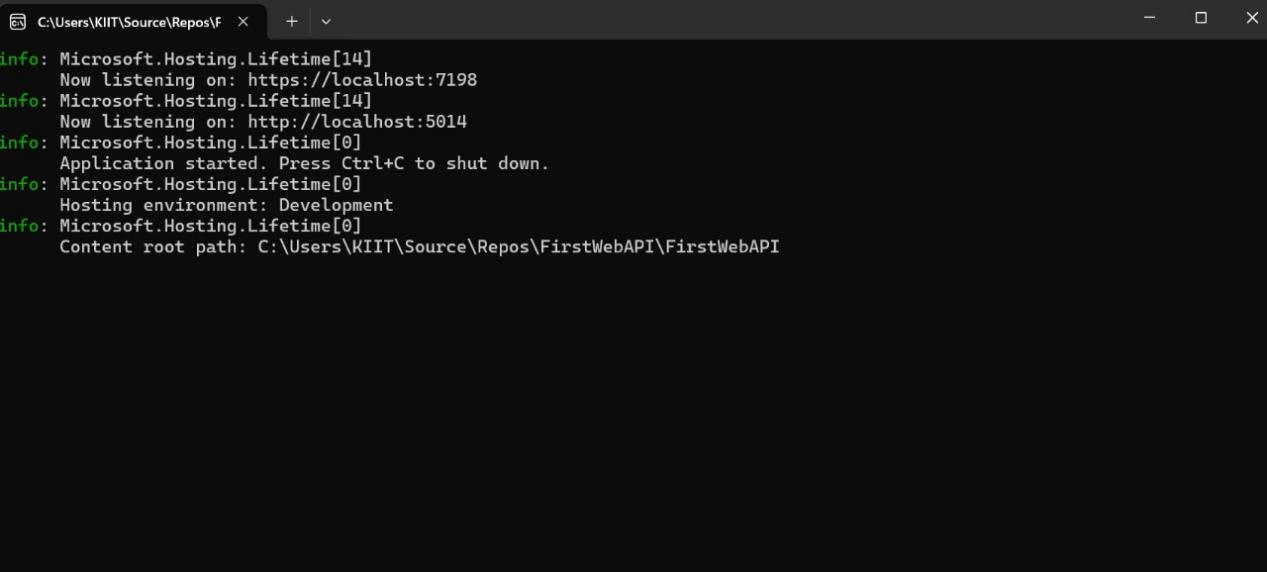
**First Web Api using .Net core**

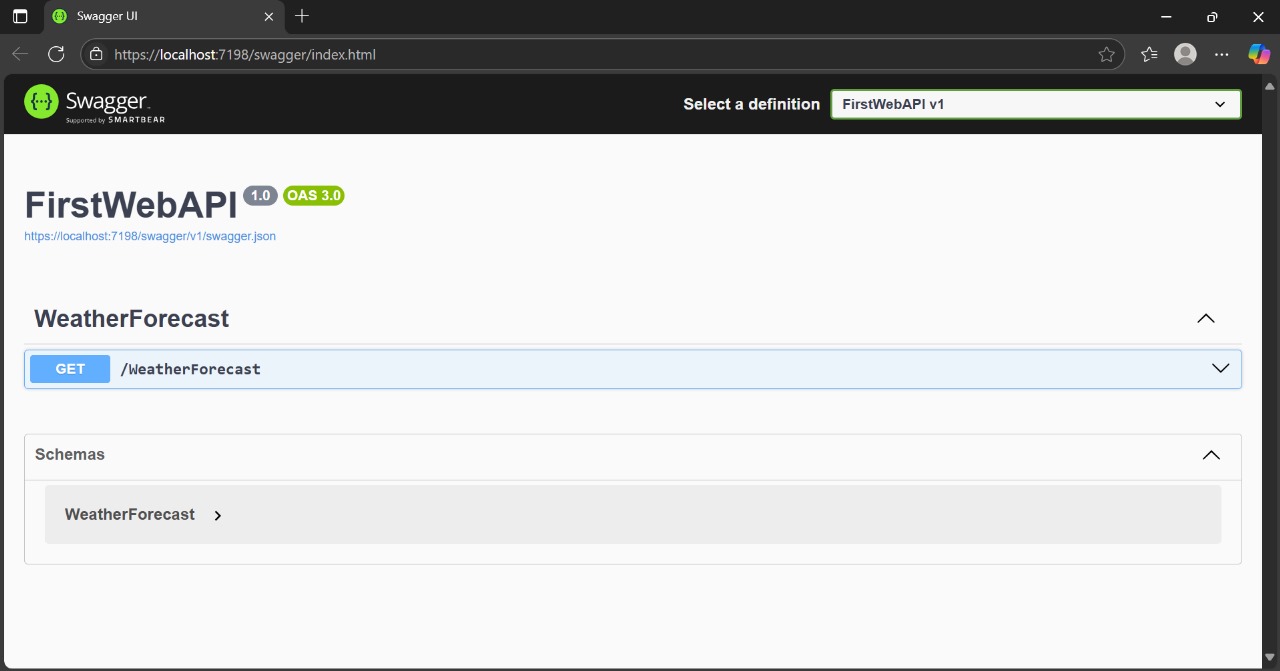
Create a .Net core web application with API template. Use the option to create controller with Read Write permissions. Notice the ValuesController creation with Action methods corresponding to the Action verbs.

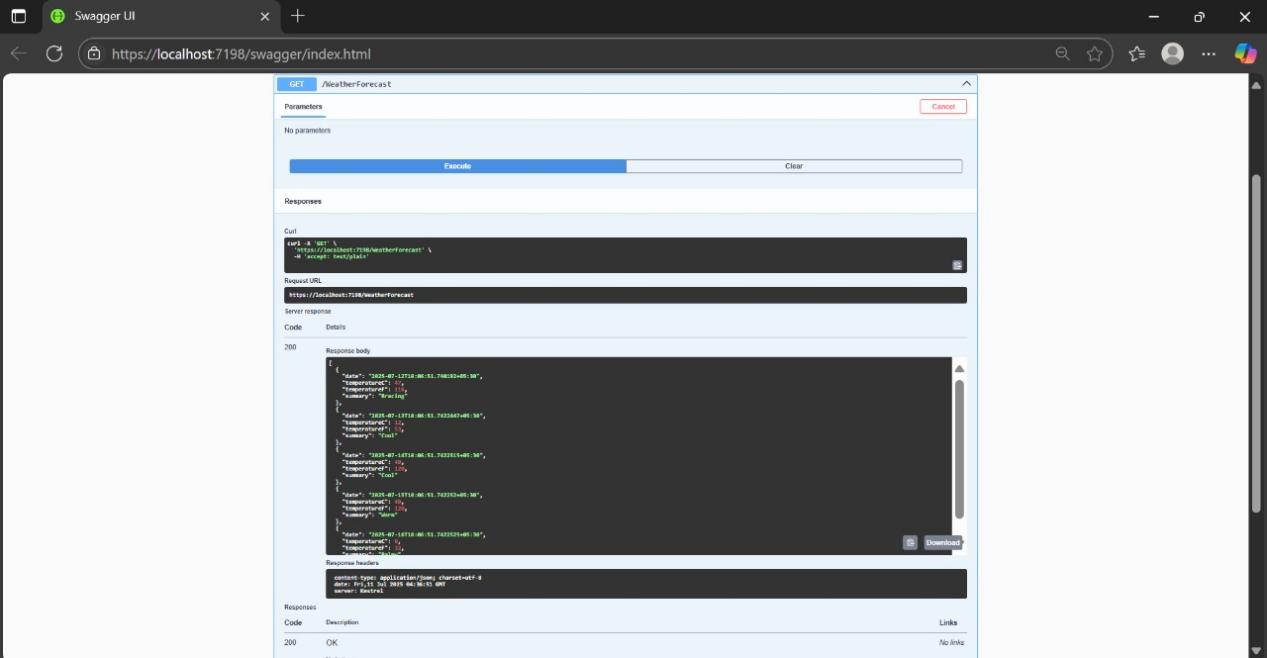
On creation of the Web API, execute the application and check if the GET action method result is returned as expected.

****

****

****

****

****

**lab2.Objectives:**

* Demonstrate Swagger installation to WebAPI and WebAPI listing on browser
  + Nuget package to download Swashbuckle.AspNetCore, Usage of ProducesResponseType to Web API method, AddSwaggerGen, UseSwaggerUI
* Demonstrate the usage of Postman tool to hit WebAPI methods
  + Structure in Postman tool, Headers with Authorization, Body as JSON, Option to choose the type of request, Request collection and how to add a new request in the collection, Tabs in the center pane that corresponds to the request
* Demonstrate the usage of Route and Explain Name attribute in Http requests
  + Importance of user friendly name to action method, Explain the usage of ActionName to have more than 1 method with the same Action verbDemonstrate creation of a simple WebAPI - With Read, Write actions

**Solutions:-**

✅ **Swagger in WebAPI:**

Swagger is an **API documentation & testing tool** for REST APIs.

Enables visualization and testing of API endpoints on the browser.

Uses Swashbuckle.AspNetCore NuGet package to integrate into ASP.NET Core.

Uses ProducesResponseType to specify expected HTTP status codes and responses in Swagger UI.

✅ **Postman Usage:**

Postman is used for testing APIs by sending HTTP requests and inspecting responses.

Supports adding **Authorization headers, request body (JSON),** and selecting **request type (GET, POST, etc).**

Supports **collections for organizing API requests**.

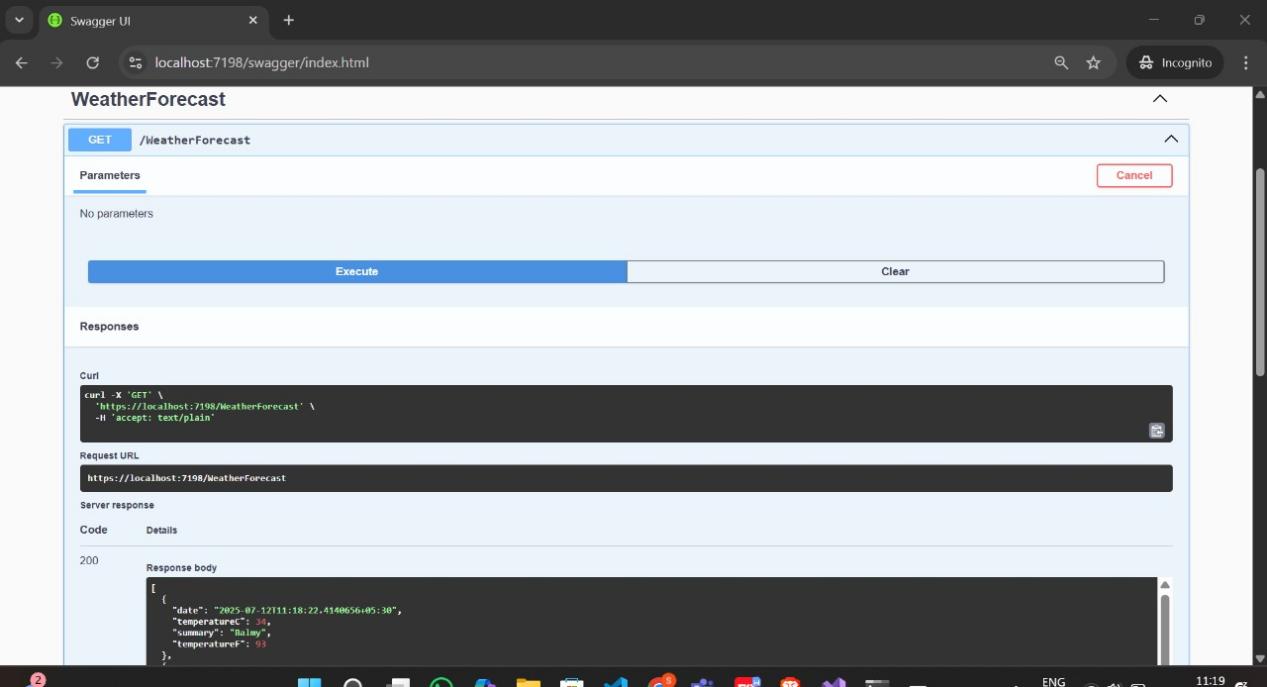
✅ **Route and Name attribute:**

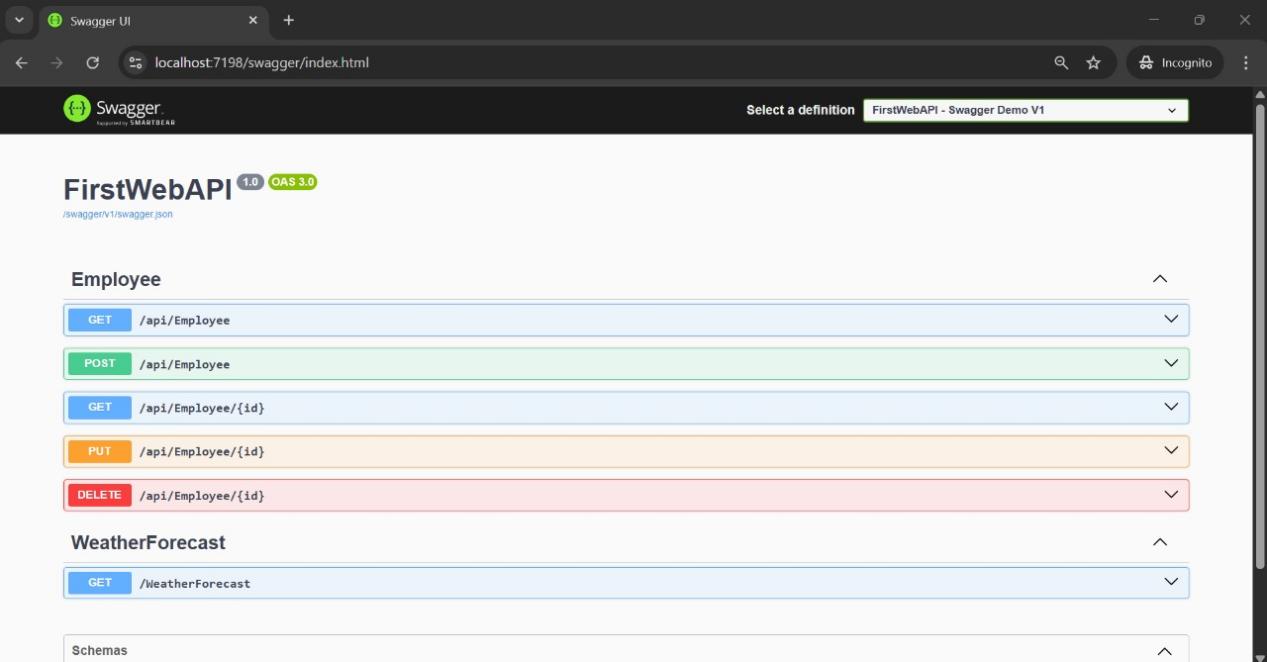
Route defines the URL structure for the API controller/action.

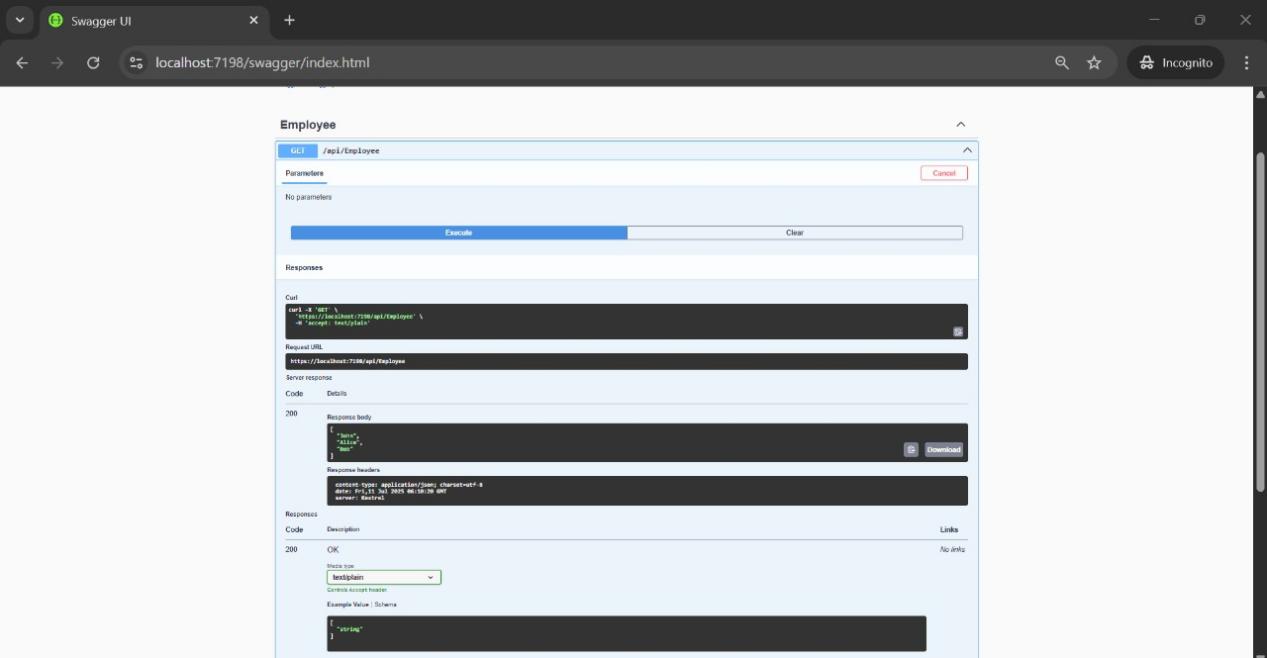
Using Name with [HttpGet(Name = "MyAction")] makes URLs user-friendly and supports identifying specific routes.

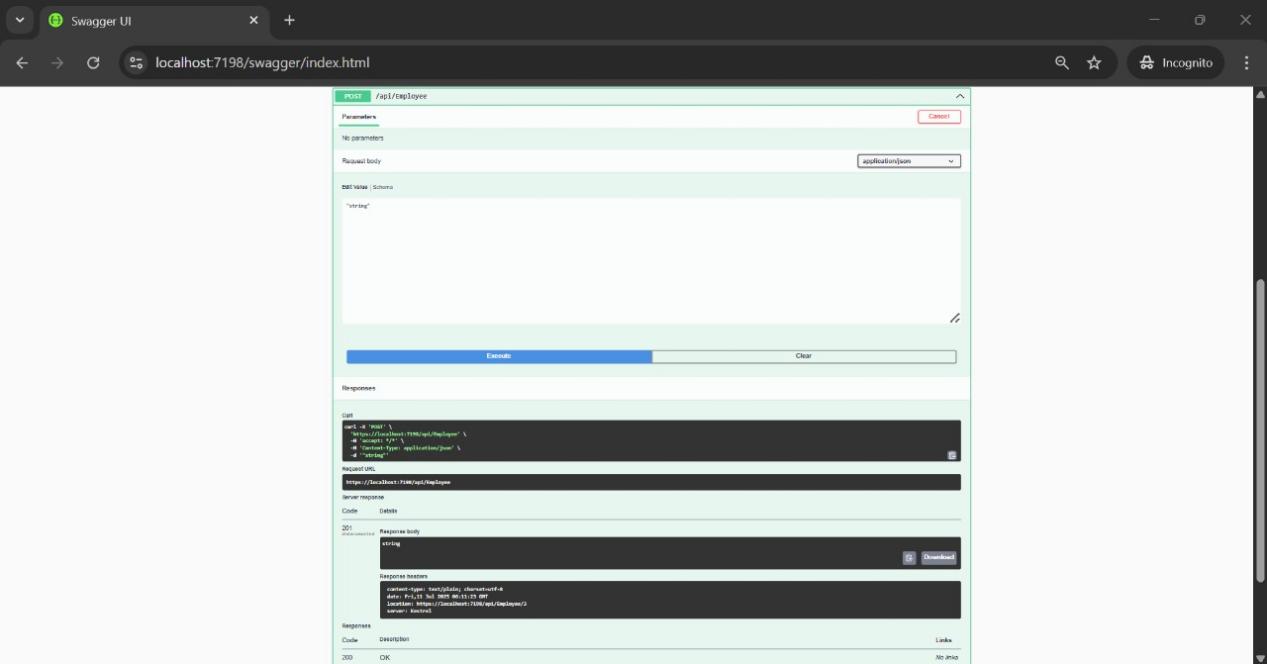
ActionName allows **overloading HTTP methods** with different names while using the same HTTP verb.

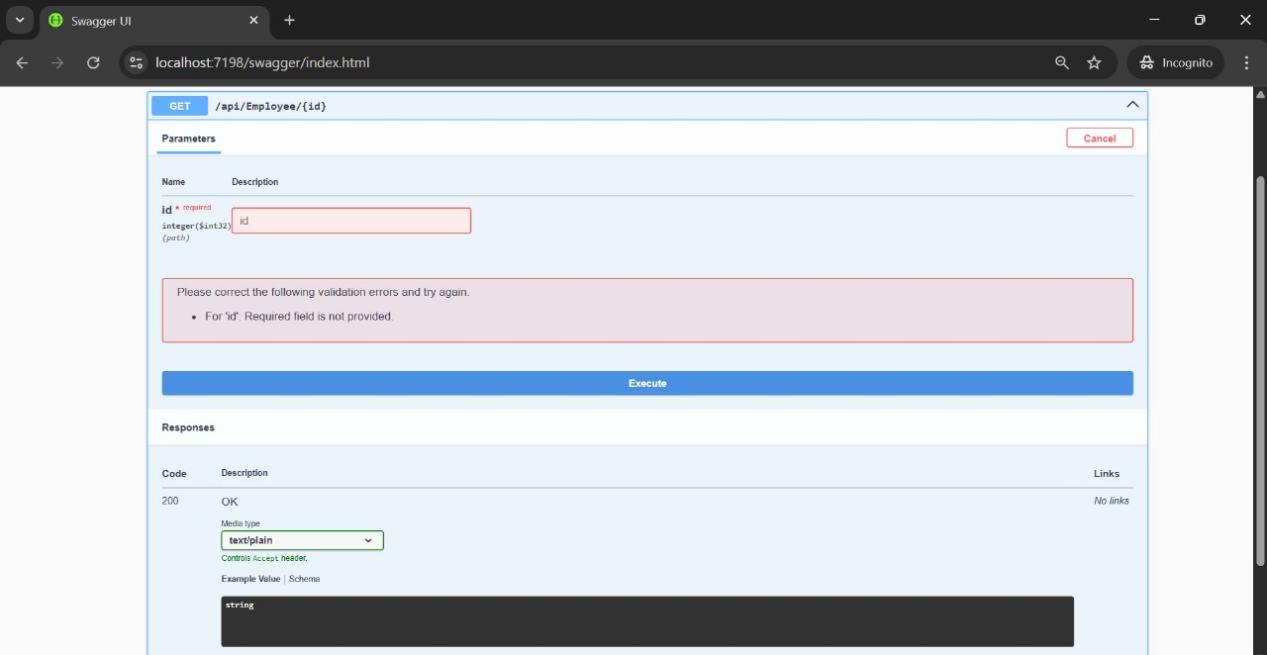
1.

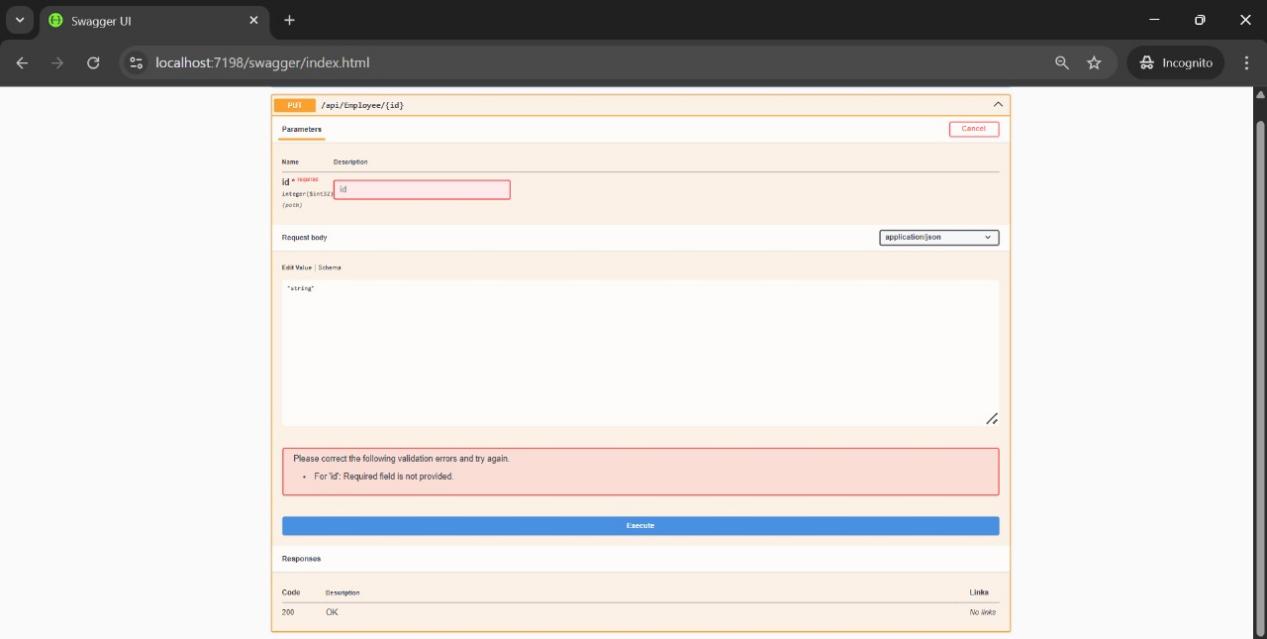


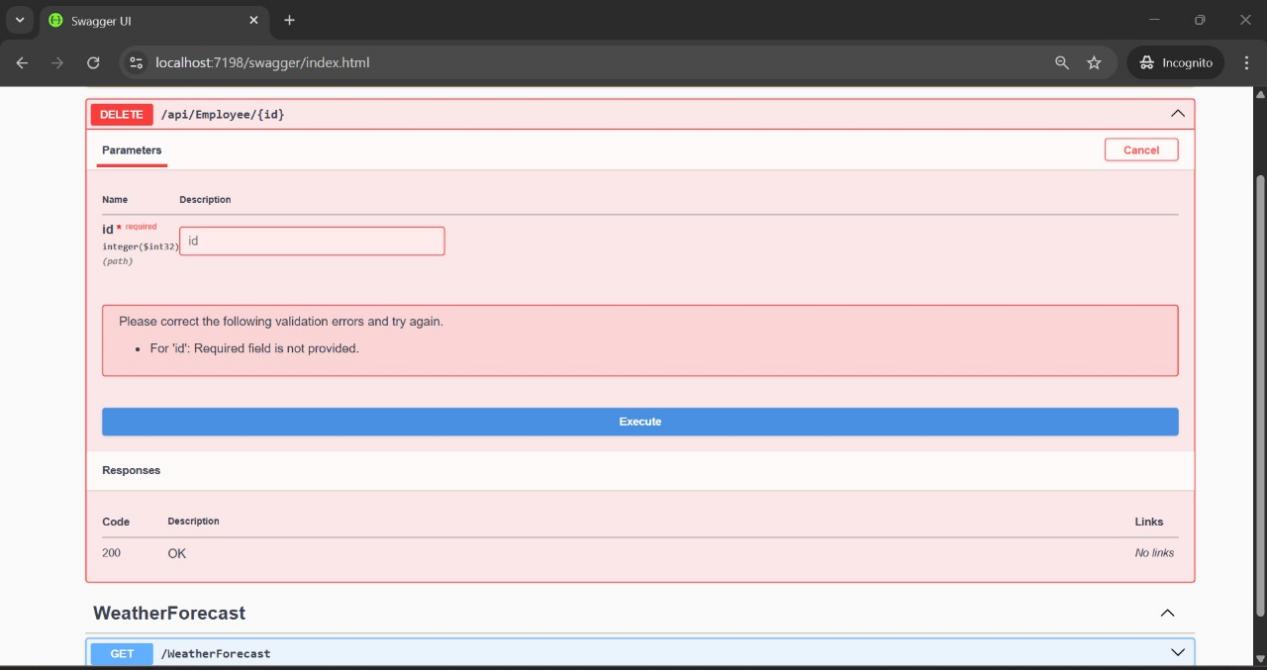




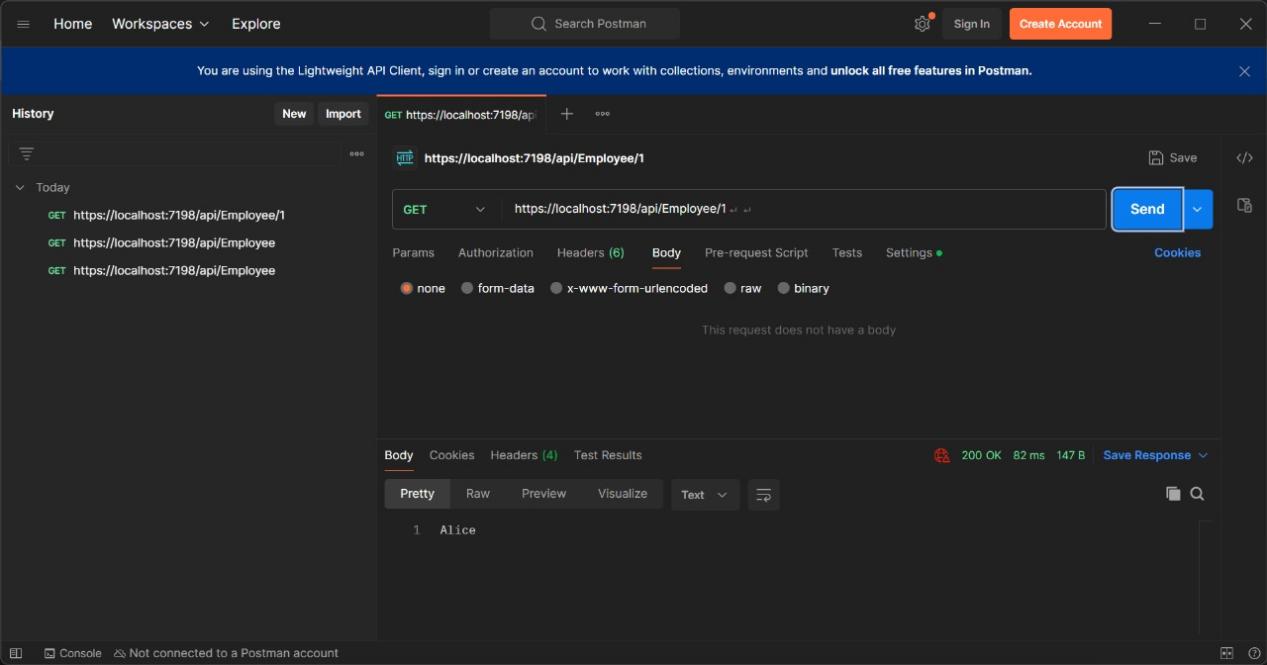




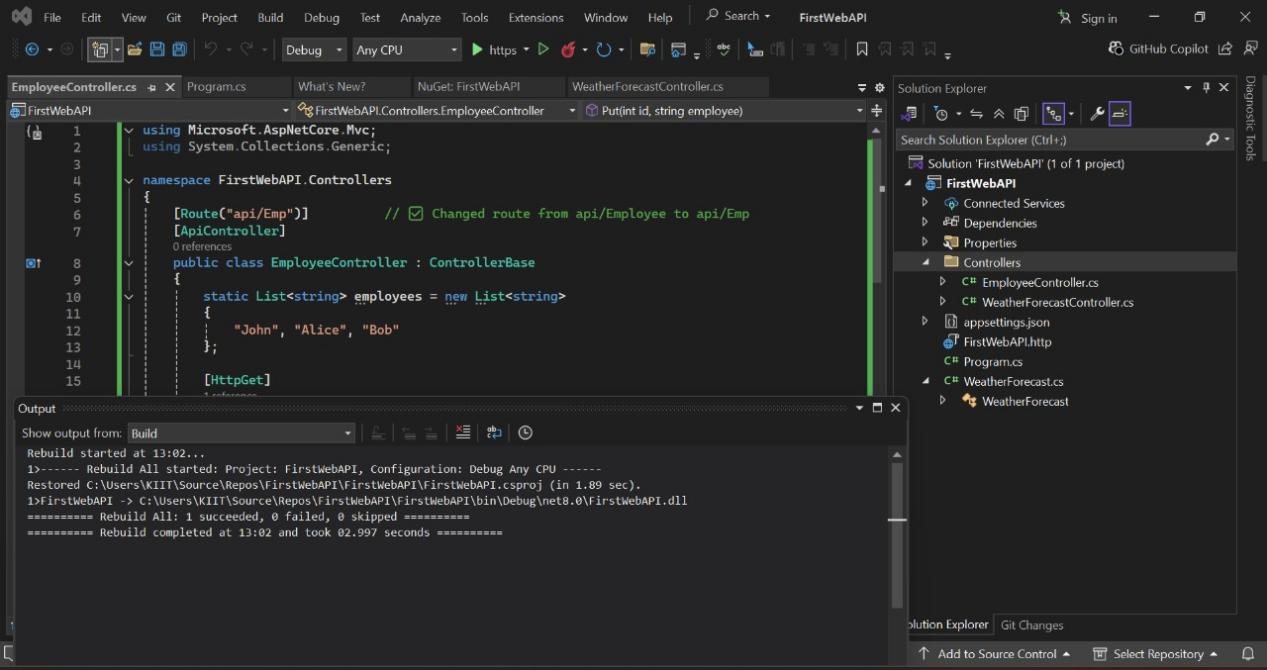


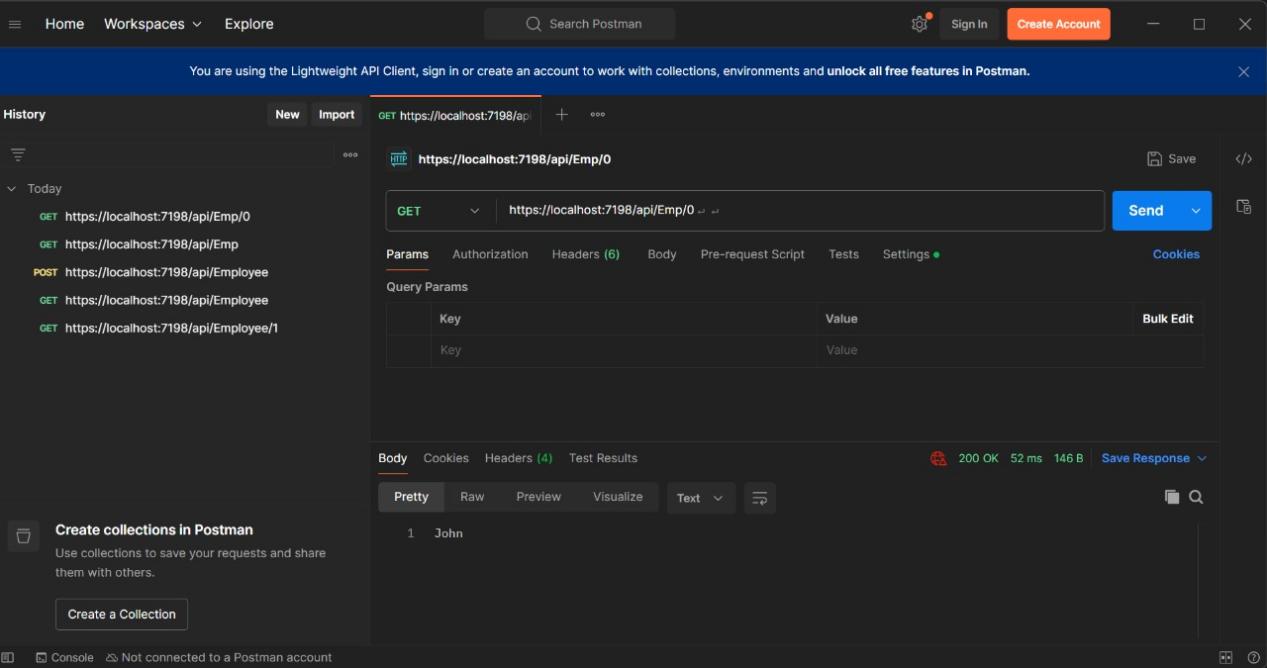


2.



3.





**Lab3.Objectives:**

* Demonstrate creation of an Action method to return list of custom class entity
  + Model class creation, Use AllowAnonymous attribute, Use HttpGet action method
* Explain the usage of FromBody attribute
  + Read the model object from request, other than the query string parameter
* Demonstrate Custom filter

Usage of ActionFilterAttribute, OnActionExecuting method to intercept the request, Create filter for Custom exception - Need to install Microsoft.AspNetCore.Mvc.WebApiCompatShim package

**Solution:-**

✅ **Action method returning a list of custom class entity:**

Create **model class** Employee with nested Department and Skill classes.

Use HttpGet**,** AllowAnonymous, and return **List of Employee** using ActionResult.

✅ **Using** FromBody**:**

Allows reading complex object data **from request body** rather than query parameters.

Used with HttpPost and HttpPut to receive **JSON data** and bind it to the model automatically.

✅ **Custom Action Filter:**

Intercepts requests to check for the Authorization header.

Validates if it contains Bearer.

Implements using ActionFilterAttribute.

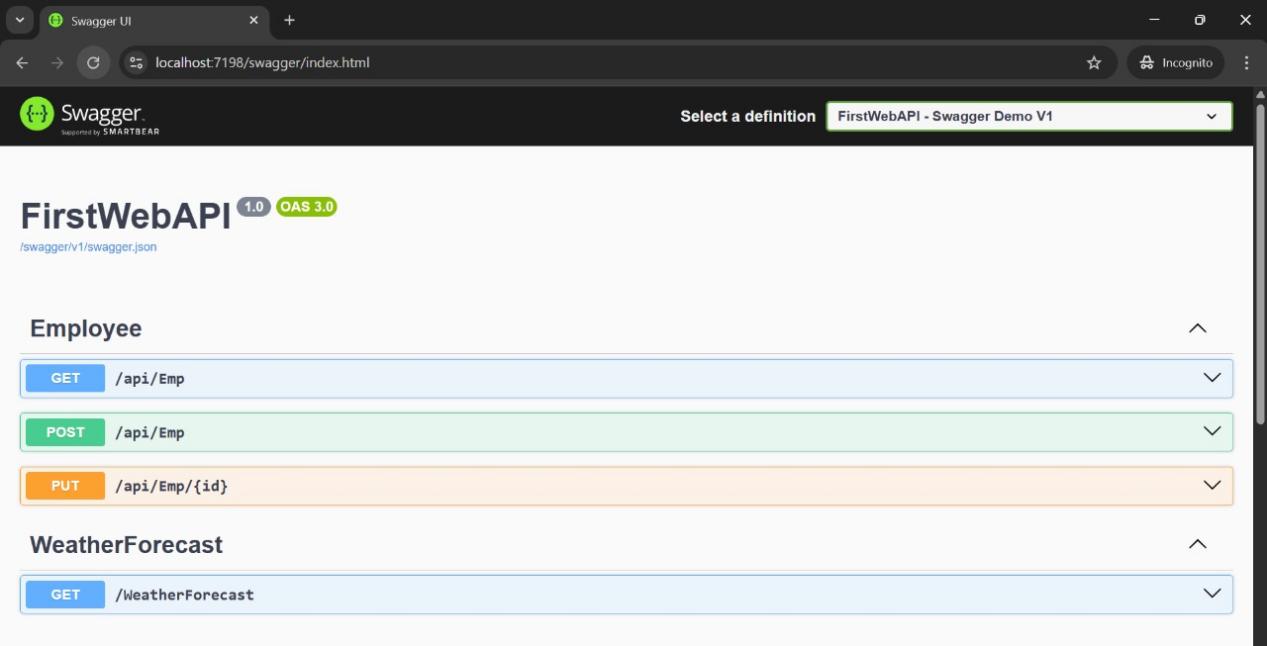
✅ **Custom Exception Filter:**

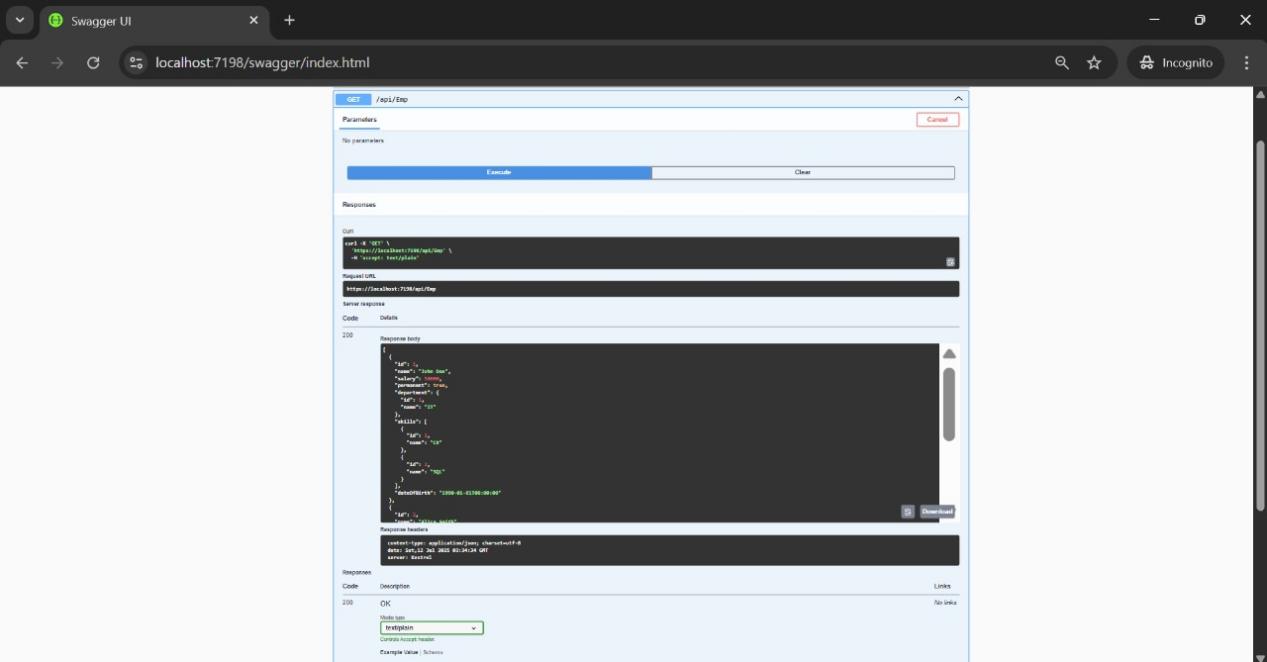
Catches unhandled exceptions globally.

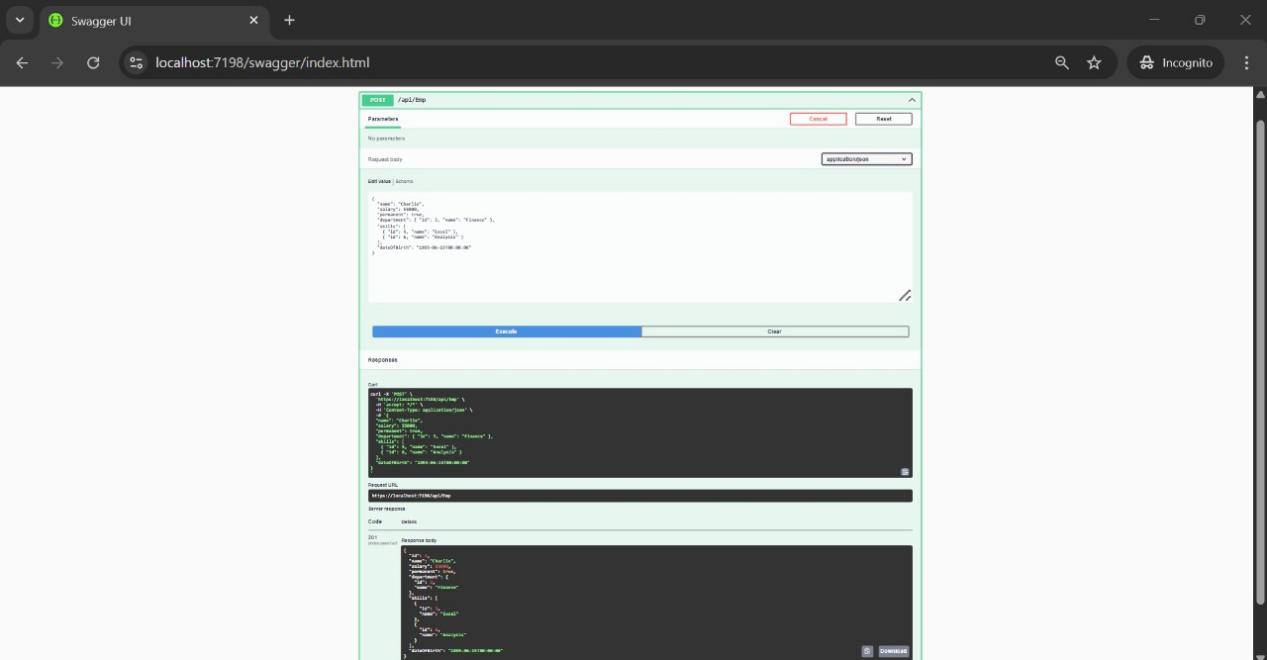
Writes exception details to a **file for logging**.

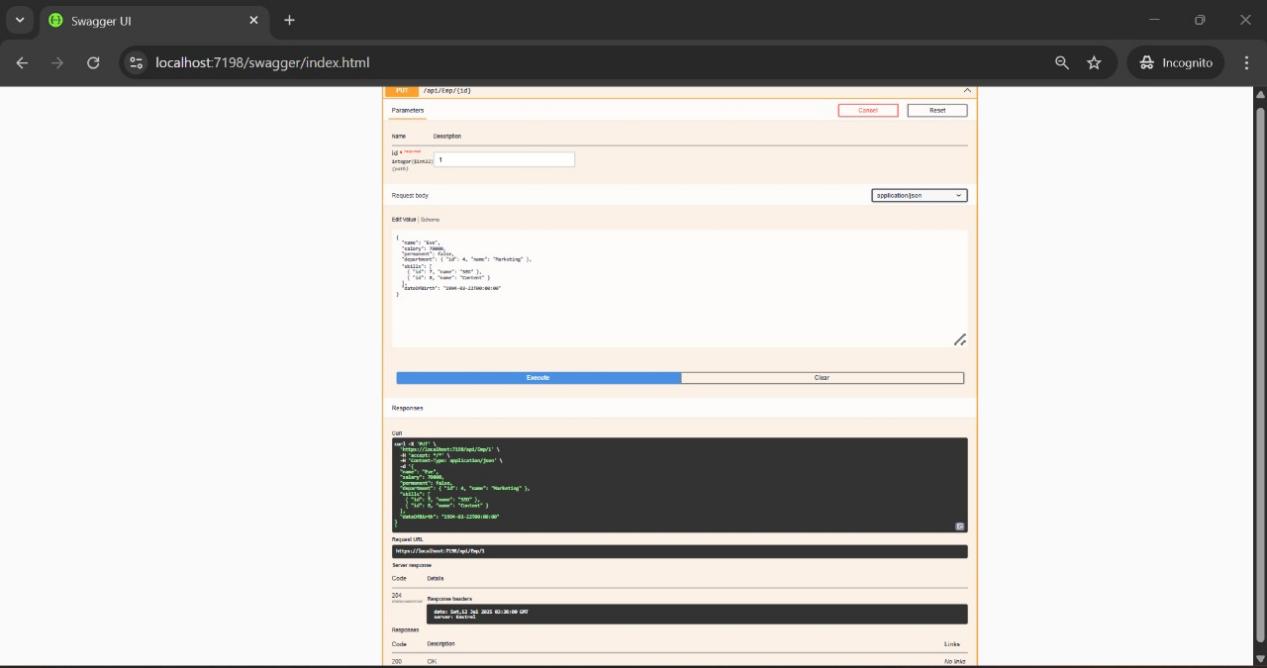
Sets the result to InternalServerError (500).

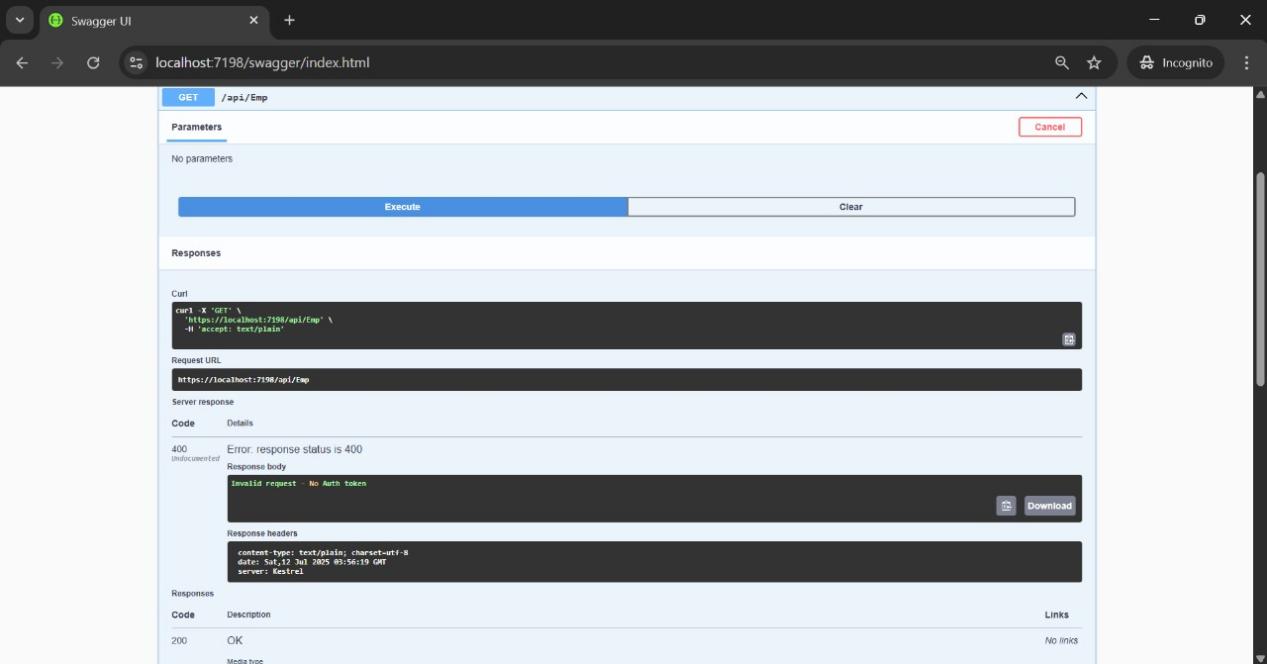
Uses IExceptionFilter.

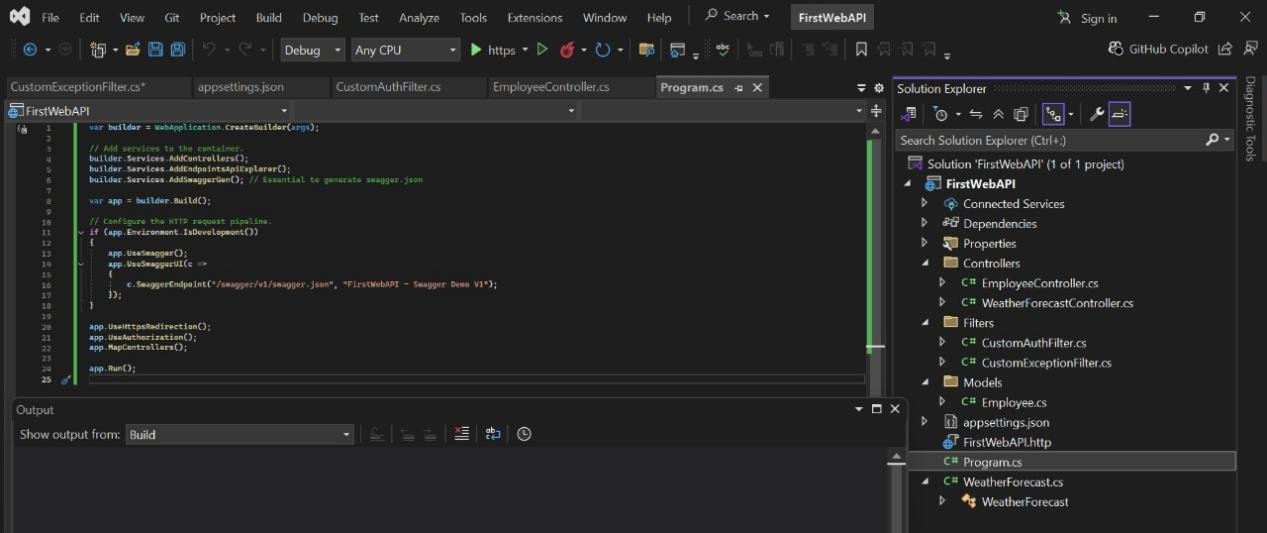


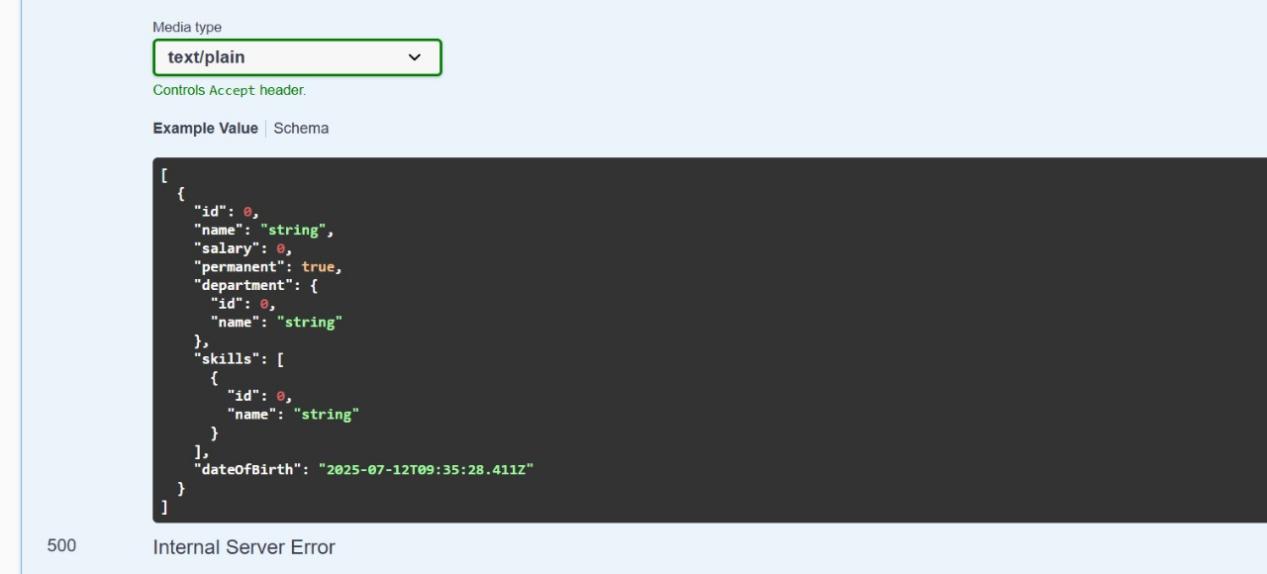


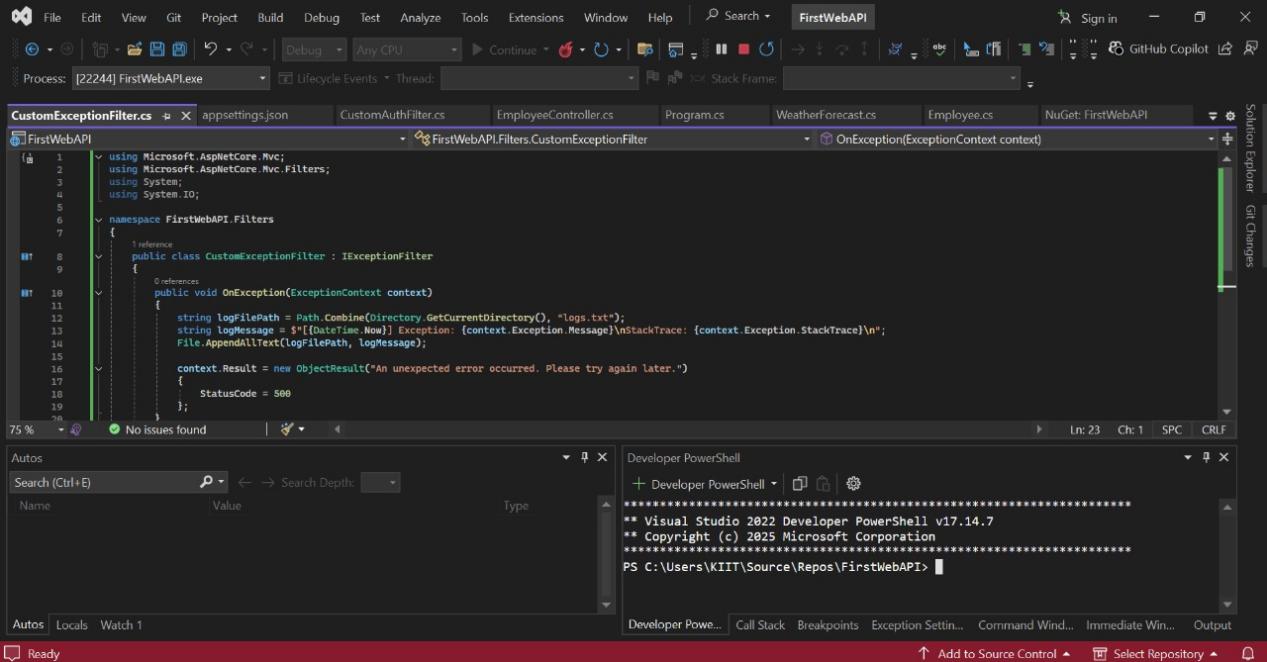


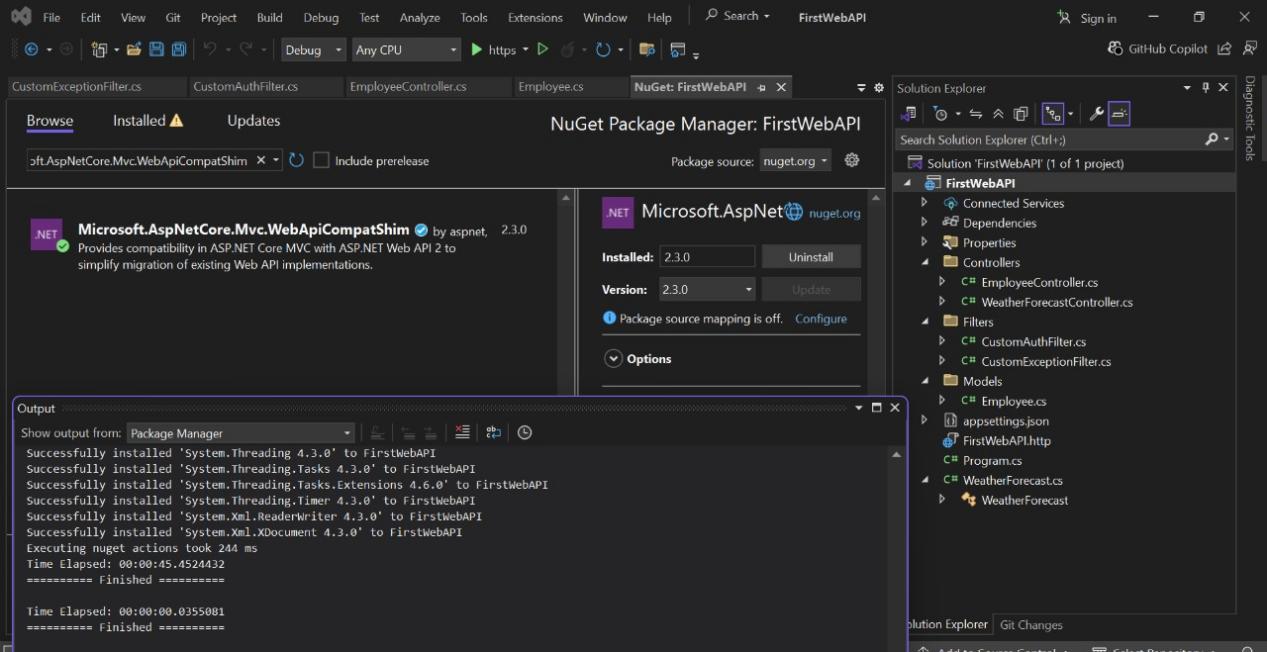












**Lab4.Objectives:**

**Solution:-**

**CRUD in Web API:**

**Create (POST):** Adds a resource.

**Read (GET):** Retrieves data.

**Update (PUT):** Modifies existing data.

**Delete (DELETE):** Removes data.

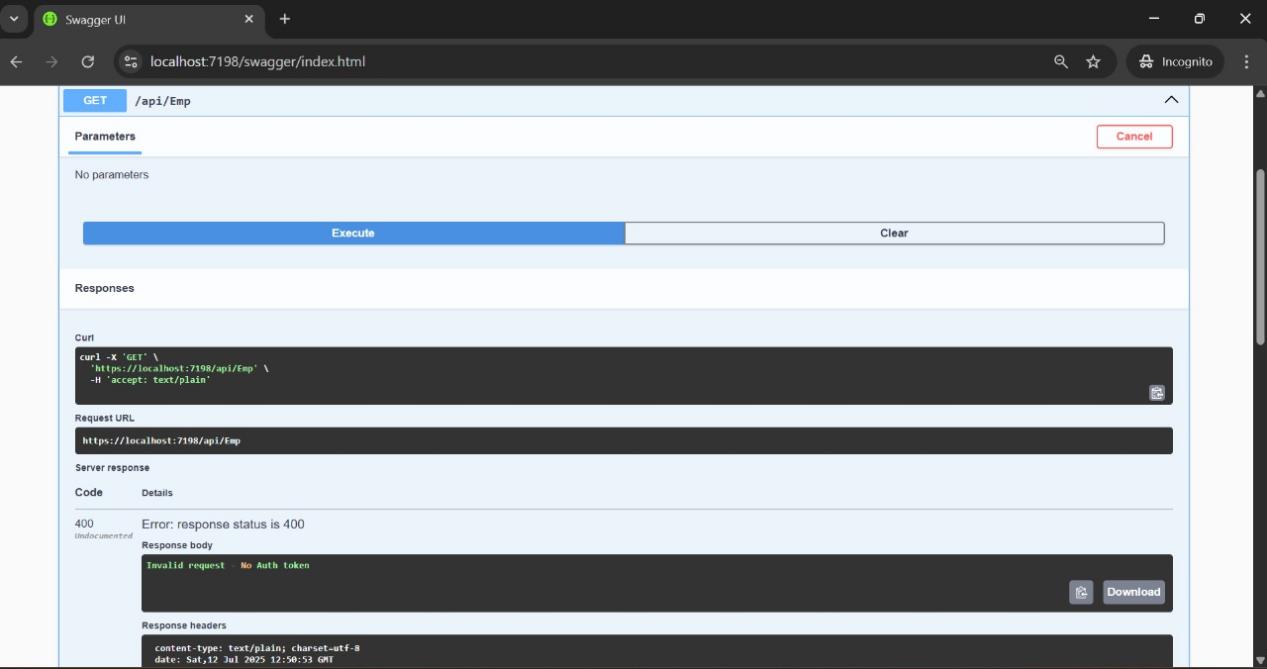
[FromBody]**:**

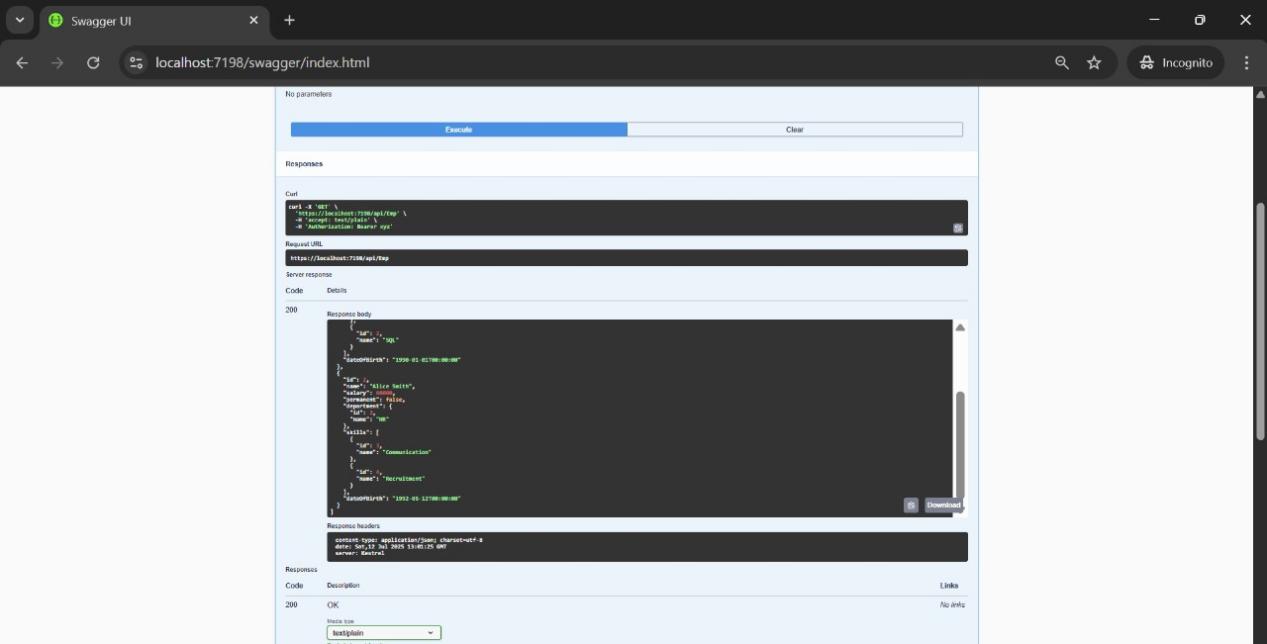
Binds complex objects sent as JSON in request body to action parameters.

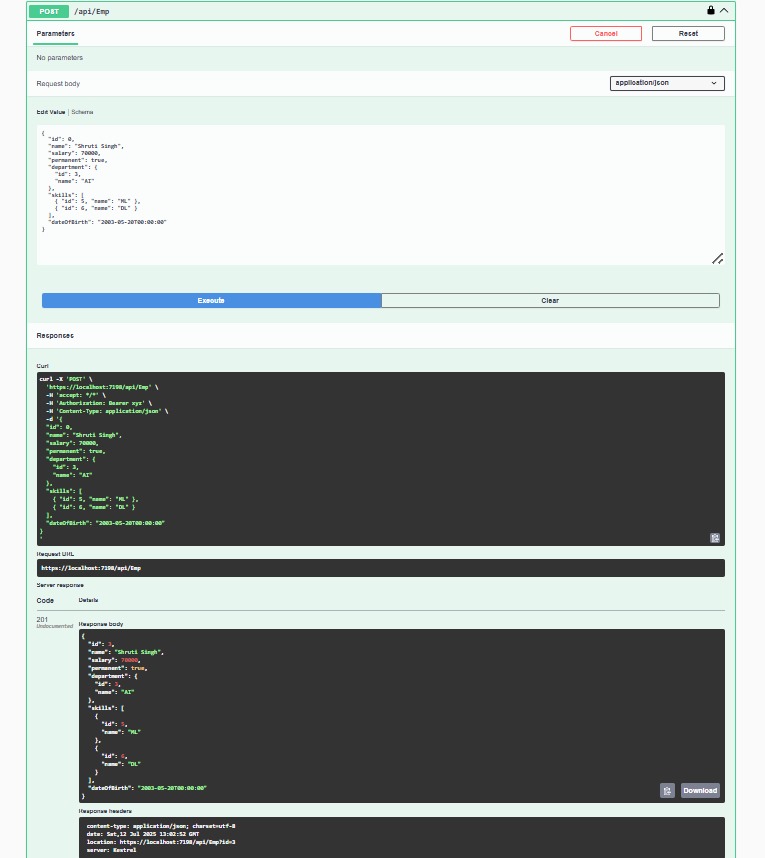
**Validation Handling:**

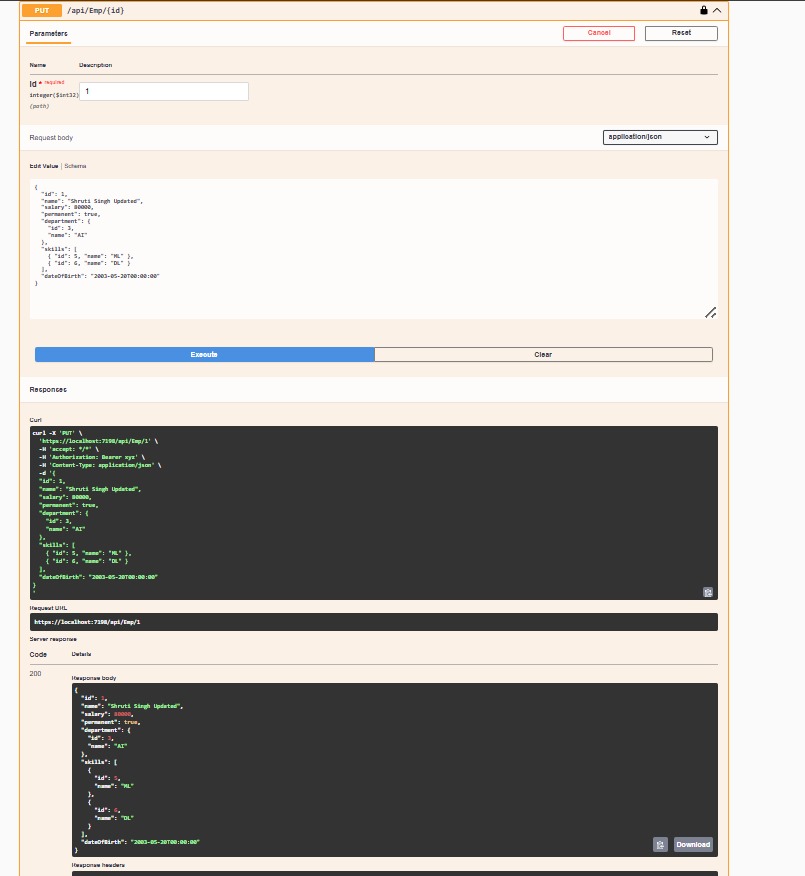
Return BadRequest if id <= 0 or if id not in records.

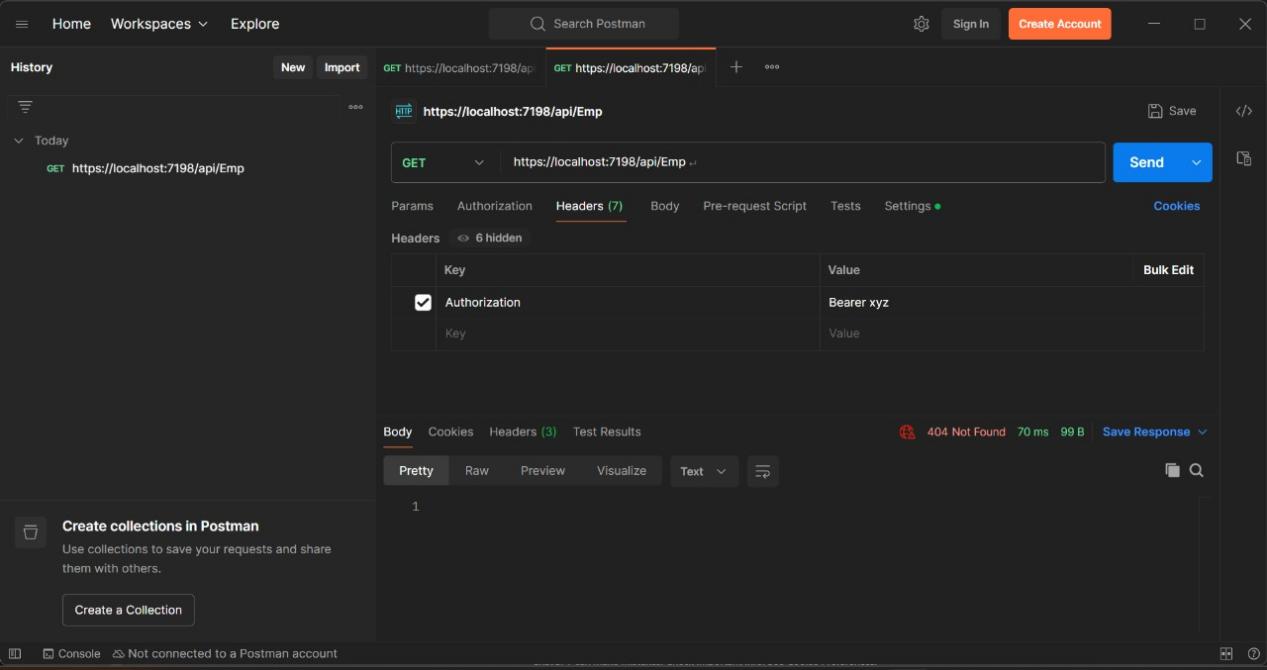
**Swagger and POSTMAN** help test endpoints interactively and visually.











**Lab5.Objectives:**

* Explain CORS enablement for Web API access for local application
  + What is CORS?, How to enable CORS thru Startup.cs, Install Cors nuget package to Web API application
* Demonstrate security in WebAPI
  + Bearer and Jwt token authentication, Use Authorize attribute & send roles in Jwt token, Setting in Startup.cs for AddAuthentication and AddJwtBearer with validation attributes, UseAuthentication, AllowAnonymous to AuthController to generate token, Claims

**Solutions:-**

✅ **Bearer and JWT Token Authentication:**

Bearer is the scheme used in the Authorization header.

JWT contains encoded claims about the user (e.g., Role, UserId) and is signed with a secret key for integrity.

Authorize attribute ensures only authenticated requests can access endpoints.

Roles can be used in Authorize(Roles="Admin") for role-based filtering.

✅ **Key Classes/Methods:**

JwtBearerDefaults.AuthenticationScheme: Used to configure JWT auth.

AddAuthentication, AddJwtBearer: Configure JWT token validation.

UseAuthentication(): Adds middleware for authentication in the pipeline.

AllowAnonymous: Used on endpoints where token is not required.

✅ **JWT Expiration:**

Set in:

expires: DateTime.Now.AddMinutes(10)

After expiry, API will return 401 Unauthorized.

✅ **Testing Tools:**

**Swagger:** Testing with Authorization tokens.

**POSTMAN:** Testing with headers, body, tokens, and verifying status codes.

