Week -4

Aditya Raj

SuperSet ID - 6363544

LAB - 1:

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.

Ans:

using Microsoft.AspNetCore.Mvc;

namespace FirstWebApi.Controllers

{

[ApiController]

[Route("[controller]")]

public class WeatherForecastController : ControllerBase

{

private static readonly string[] Summaries = new[]

{

"Freezing", "Bracing", "Chilly", "Cool", "Mild", "Warm", "Balmy", "Hot", "Sweltering", "Scorching"

};

private readonly ILogger<WeatherForecastController> \_logger;

public WeatherForecastController(ILogger<WeatherForecastController> logger)

{

\_logger = logger;

}

[HttpGet(Name = "GetWeatherForecast")]

public IEnumerable<WeatherForecast> Get()

{

return Enumerable.Range(1, 5).Select(index => new WeatherForecast

{

Date = DateOnly.FromDateTime(DateTime.Now.AddDays(index)),

TemperatureC = Random.Shared.Next(-20, 55),

Summary = Summaries[Random.Shared.Next(Summaries.Length)]

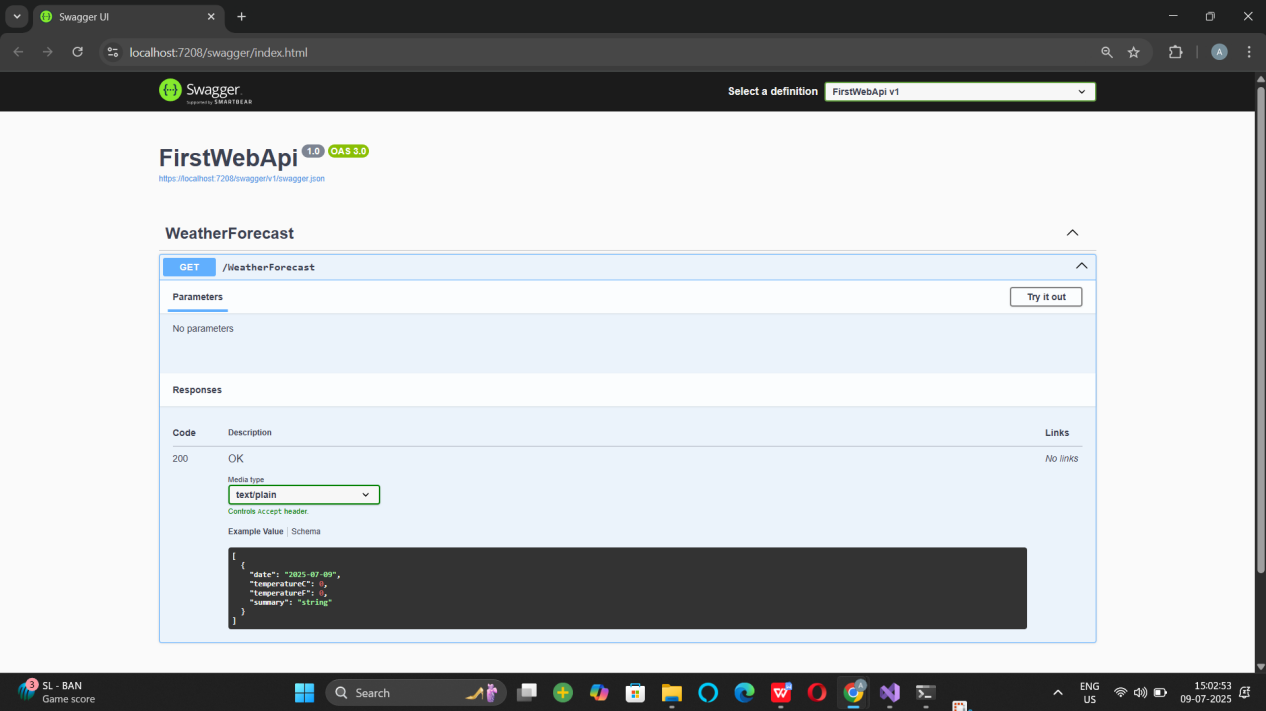
})

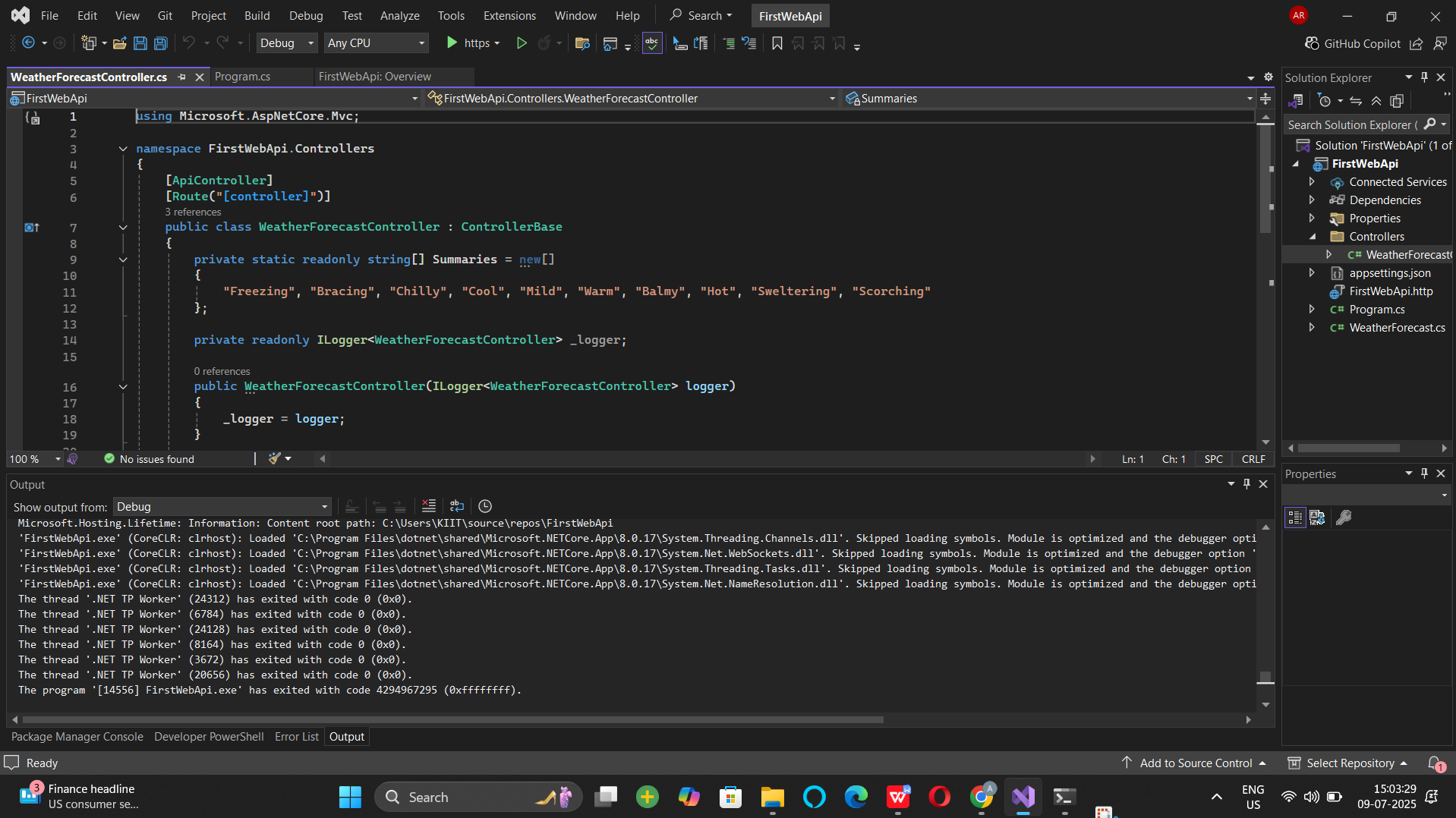
.ToArray();

}

}

}





LAB - 2

1. **Web Api using .Net core with Swagger**

Create a .Net core web application with API template. (Use existing application if created). Install Swashbuckle.AspNetCore Nuget package. Post this do the following steps in Startup.cs

* In ConfigureServices method, add the code provided below.

services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Info

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

TermsOfService = "None",

Contact = new Contact() { Name = "John Doe", Email = "john@xyzmail.com", Url = "www.example.com" },

License = new License() { Name = "License Terms", Url = "www.example.com" }

});

});

* In Configure method, add the code provided below.

app.UseSwagger();

app.UseSwaggerUI(c =>

{

// specifying the Swagger JSON endpoint.

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

});

Execute the application which will load the default ‘Values’ controller(Settings as per launchSettings.json) GET action method. Change the url to <https://localhost:[port> number]/swagger

Notice the Title, Version, Contact detail provided shown on the top of the page

Notice the Values controller HttpVerb action methods getting listed.

Click the ‘GET’ action verb method(Without the parameter).

it opens a panel which has ‘Try it out’ button. Click that and Click ‘Execute’ button.

ANS:

var builder = WebApplication.CreateBuilder(args);

.

builder.Services.AddControllers();

builder.Services.AddSwaggerGen(c =>

{

c.SwaggerDoc("v1", new Microsoft.OpenApi.Models.OpenApiInfo

{

Title = "Swagger Demo",

Version = "v1",

Description = "TBD",

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

Contact = new Microsoft.OpenApi.Models.OpenApiContact

{

Name = "John Doe",

Email = "john@xyzmail.com",

Url = new Uri("http://www.example.com")

},

License = new Microsoft.OpenApi.Models.OpenApiLicense

{

Name = "License Terms",

Url = new Uri("http://www.example.com")

}

});

});

var app = builder.Build();

if (app.Environment.IsDevelopment())

{

app.UseDeveloperExceptionPage();

app.UseSwagger();

app.UseSwaggerUI(c =>

{

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

});

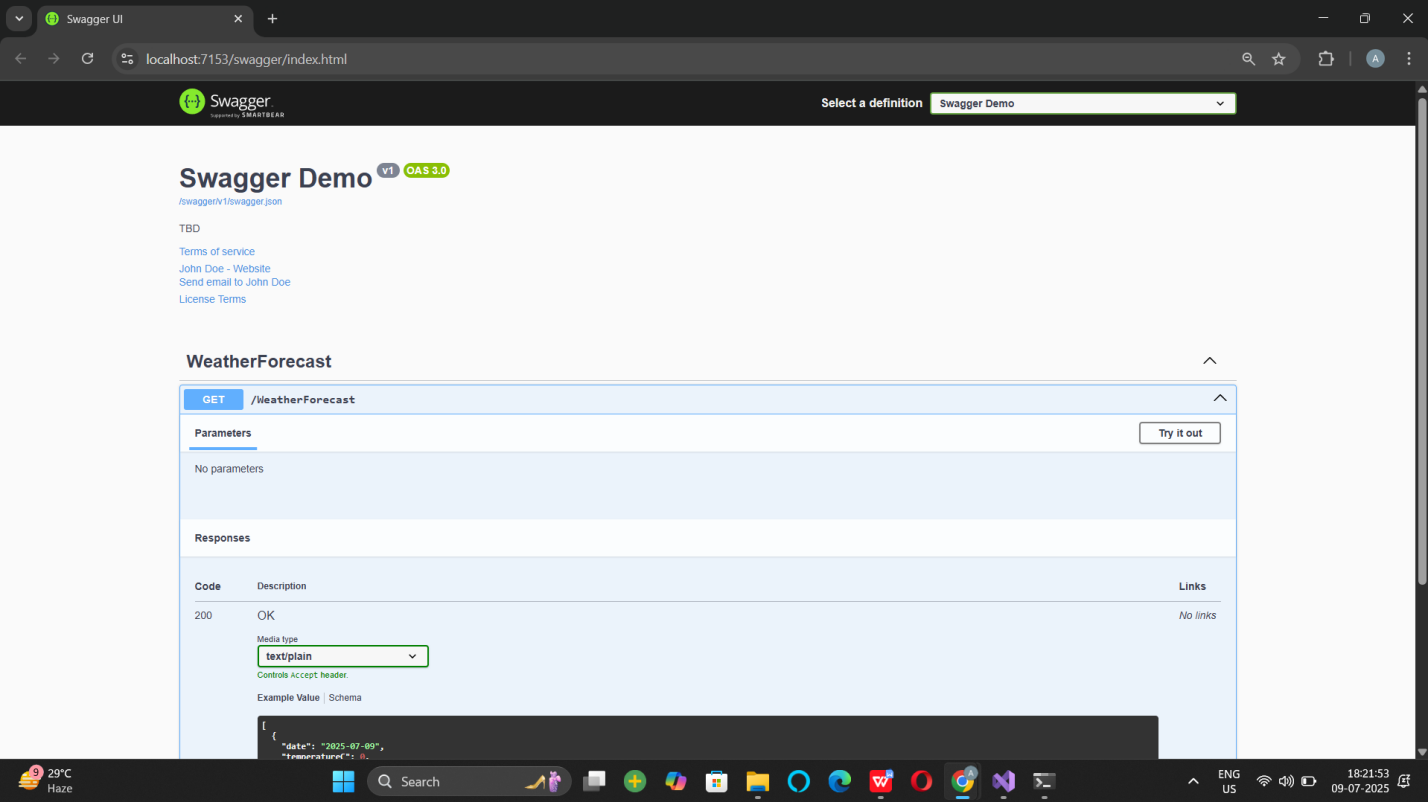
}

app.UseRouting();

app.UseAuthorization();

app.MapControllers();

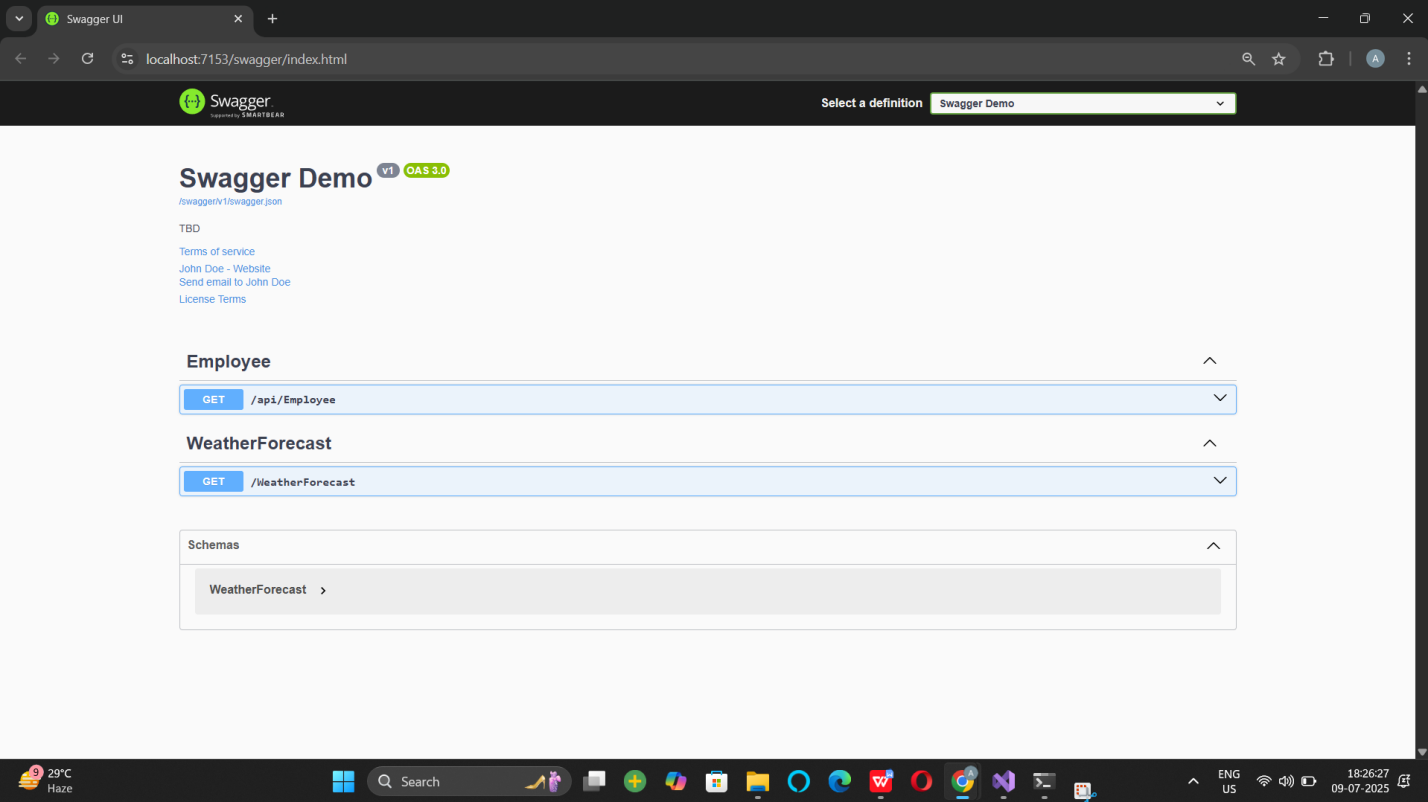
app.Run();

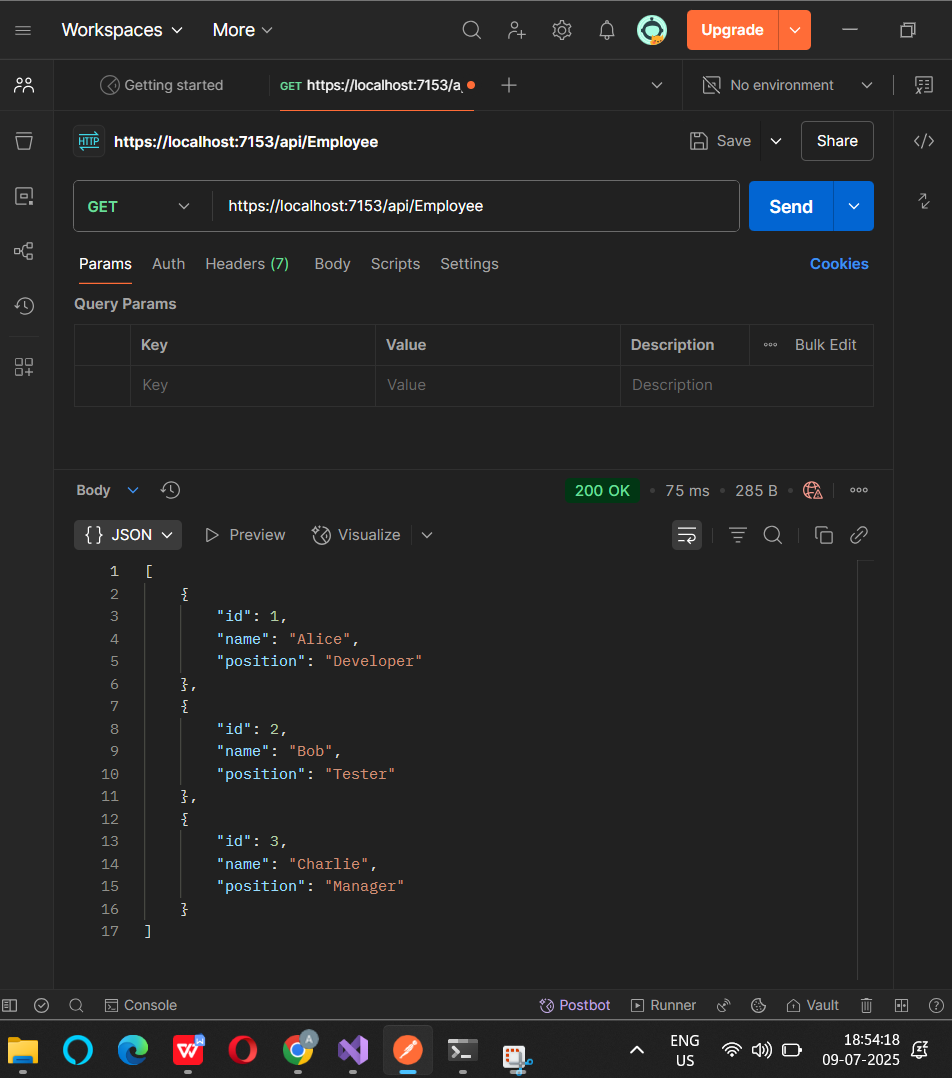


1. Use POSTMAN tool, to point to the local Web API that was created with Employee controller. Test the GET action method using POSTMAN.

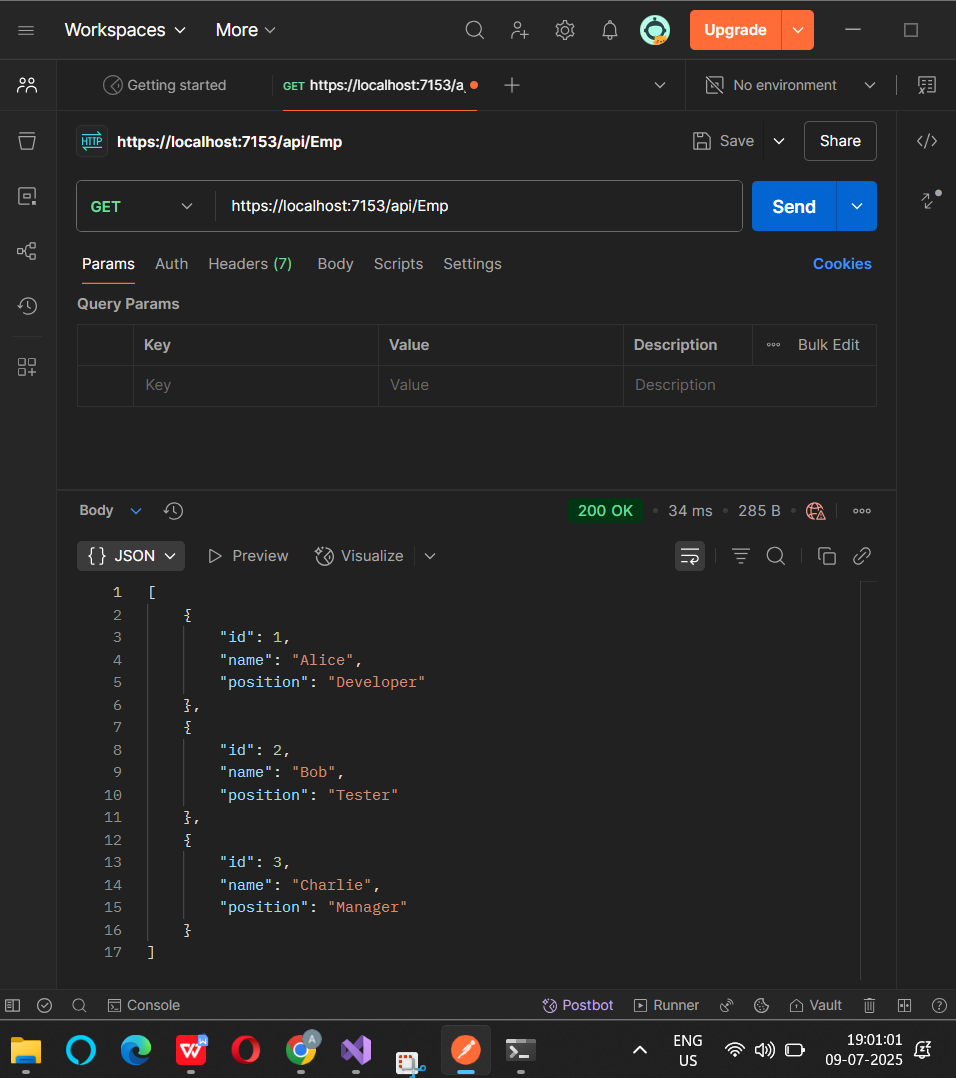
Verify the output if the List of employees are listed in the ‘Body’ part of the GET window on POSTMAN tool.

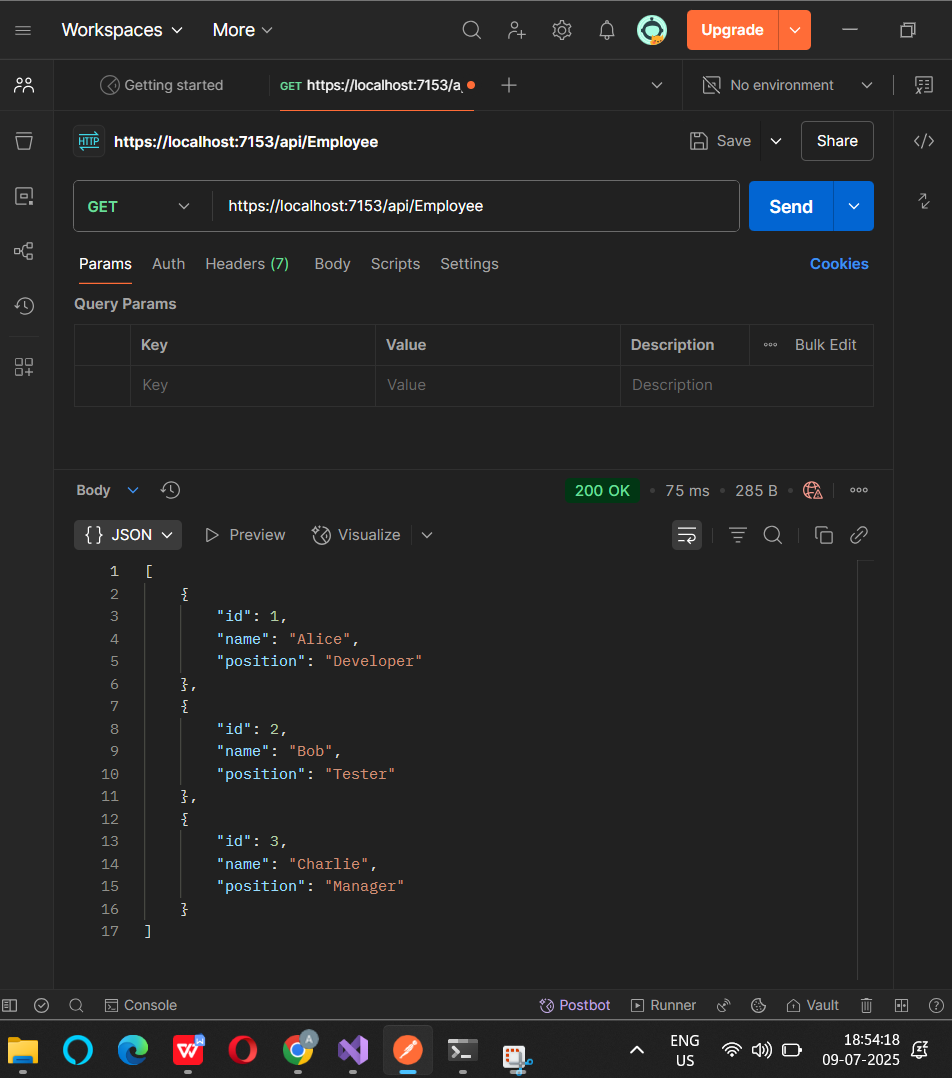
Verify the Status on the right side of the output pane on POSTMAN tool.





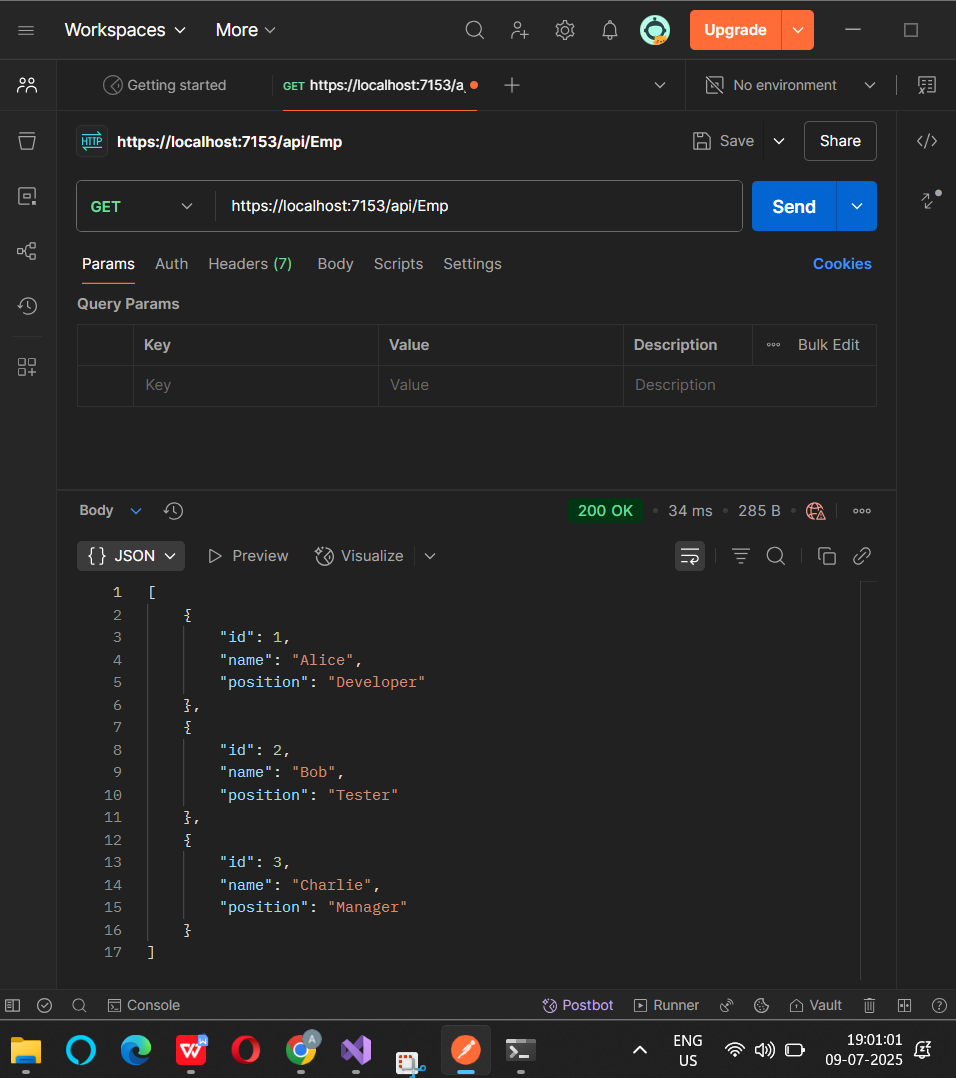
1. Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN





Verify the Status on the right side of the output pane on POSTMAN tool.

1. Modify the Controller name in the Route attribute of the Employee controller to ‘Emp’ and check its access thru POSTMAN



LAB - 3

**1 . Web Api using custom model class**

Create a Custom class ‘Employee’ of the below defined structure

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; }

}

Create a new controller - EmployeeController with Read Write actions

Constructor: Create few records, HTTPGet, HTTPPost/HTTPPut

Create a Private method GetStandardEmployeeList that returns a List of Employee class. Invoke this method in the Get action method of the EmployeeController that was created in the previous step.

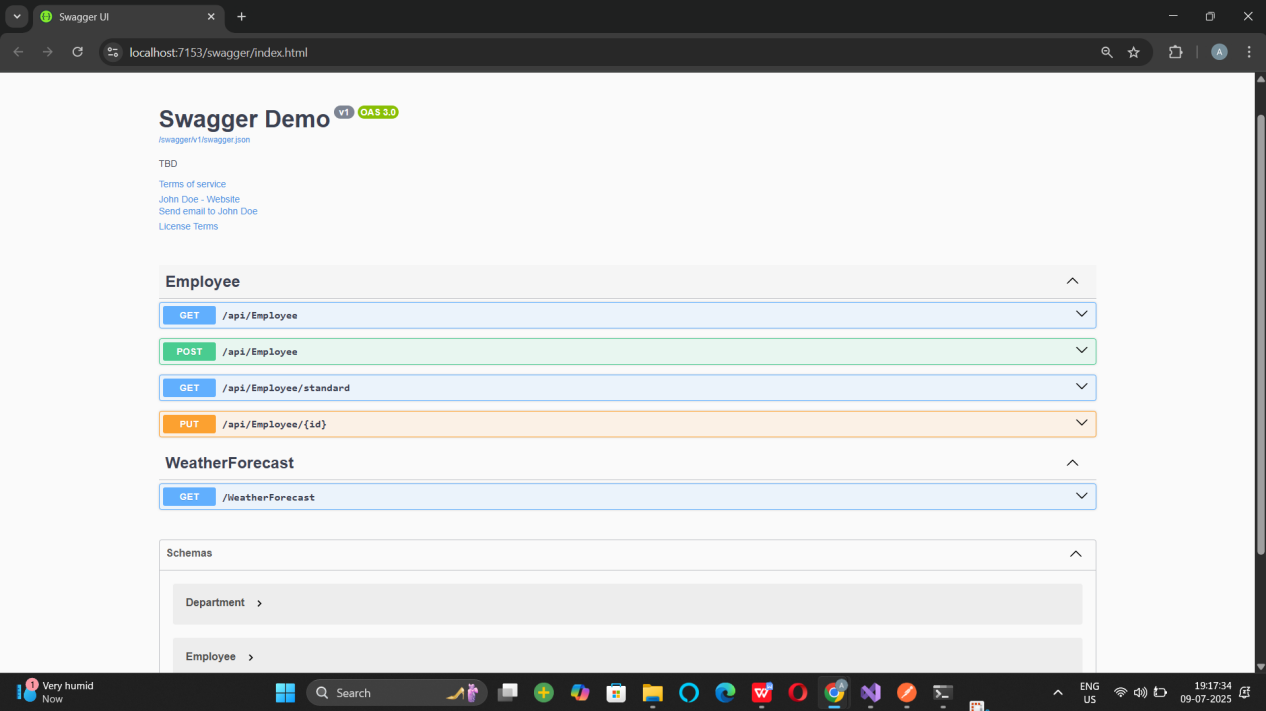
Public ActionResult<Employee> GetStandrad()

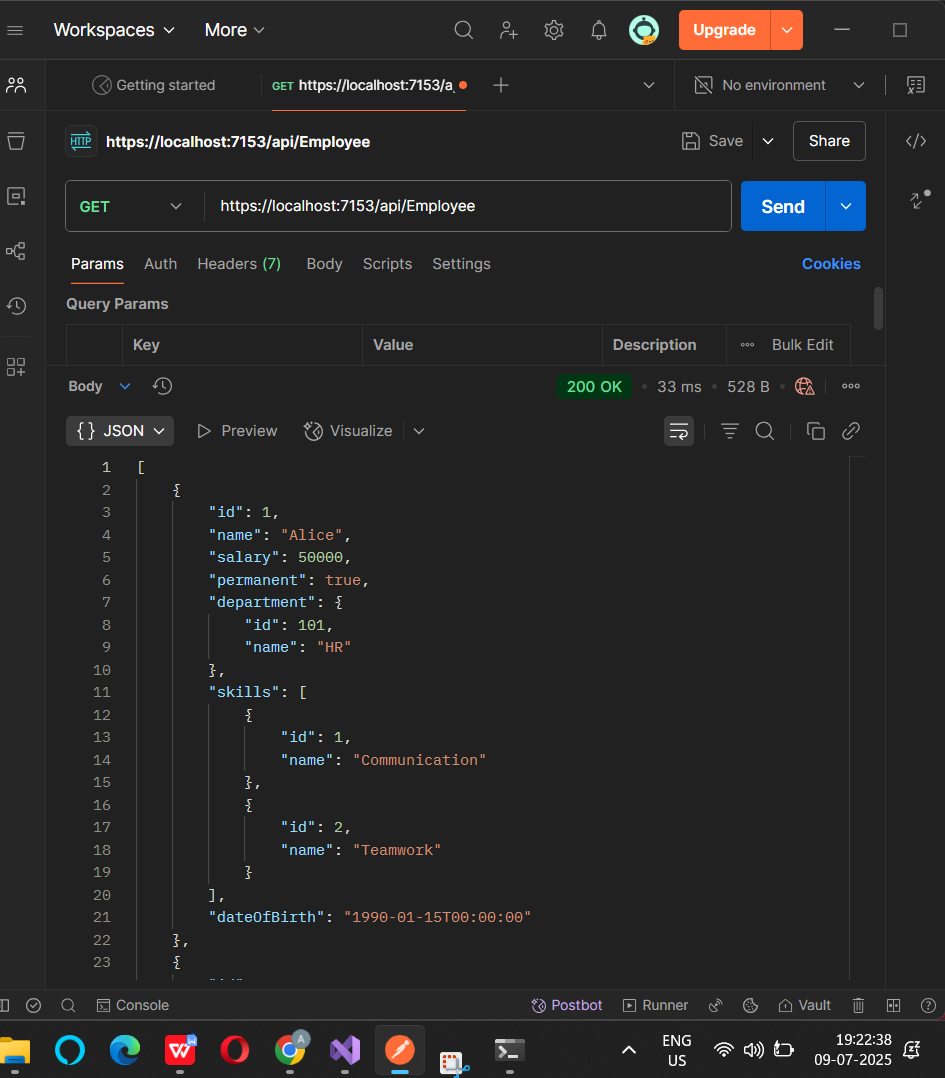
Modify the return type of the Get action method(without parameter) to return List of Employee class object

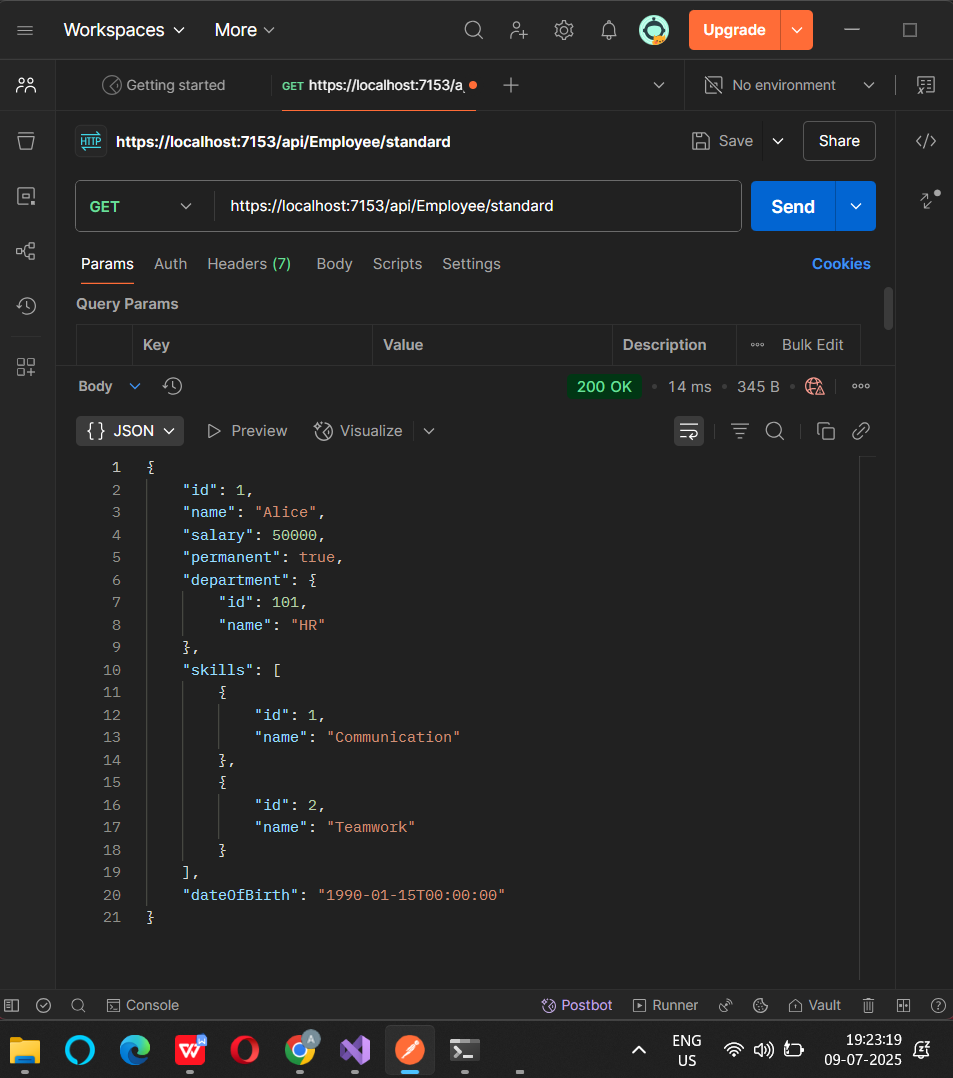
Add ProducesResponseType to the GET action method for Status code 200

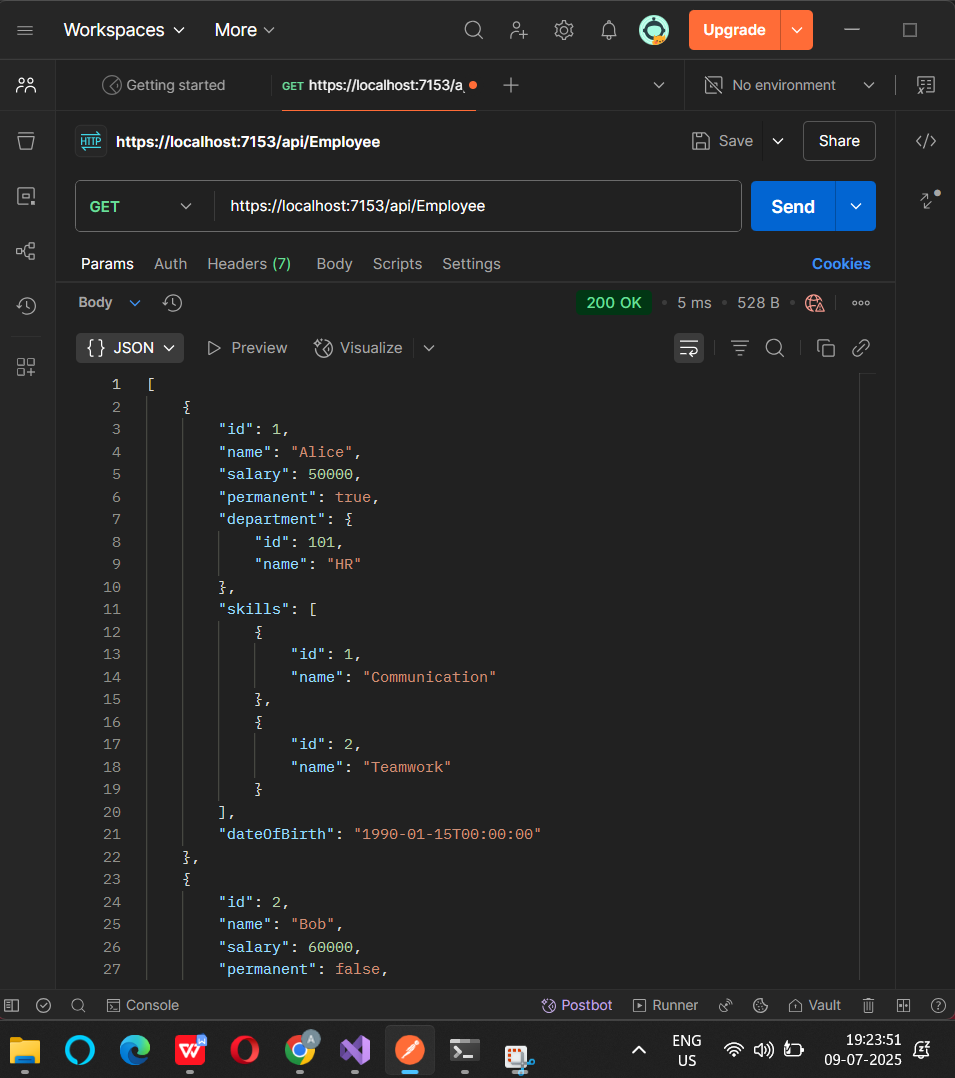
Check the Swagger description for the GET method for success status code

ANS:







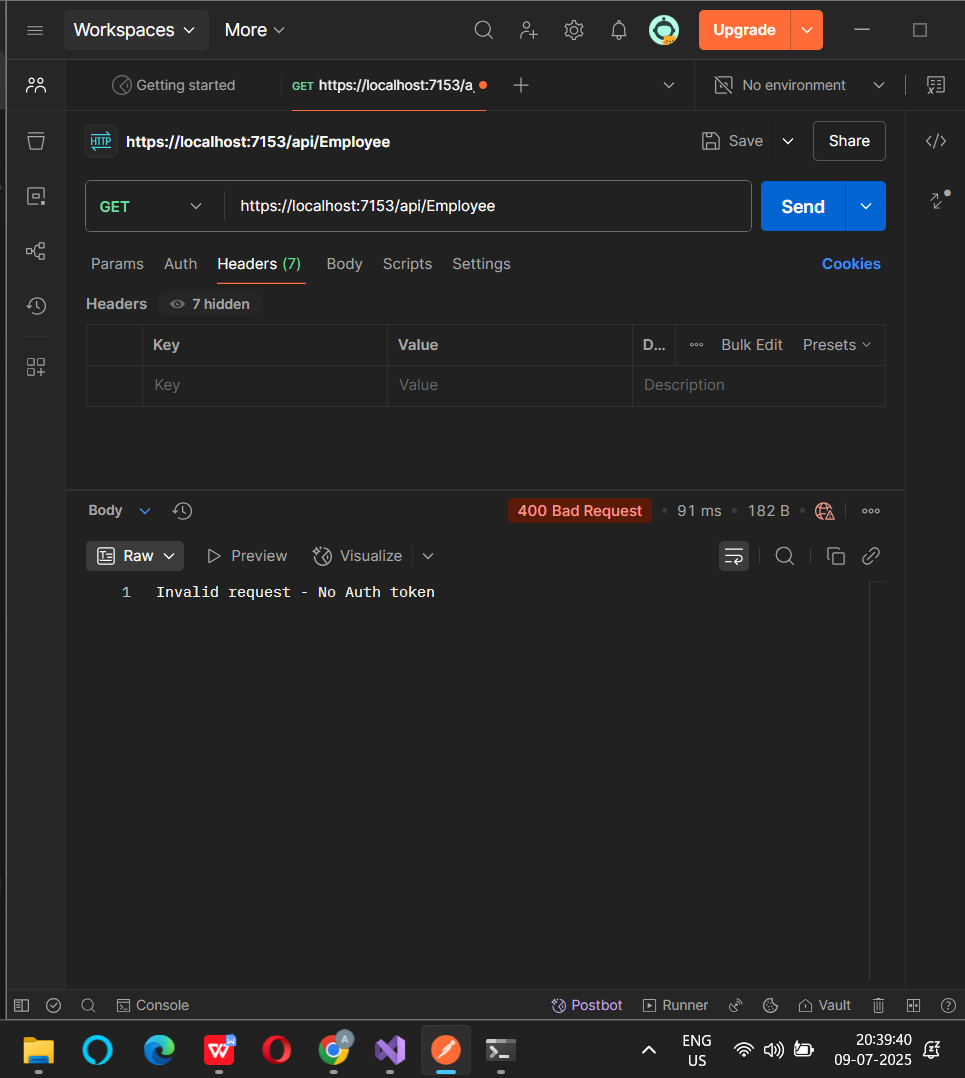


1. **Create a Custom action filter for Authorization.**

The requirement is to intercept incoming requests and check if there is a key ‘Authorization’ in the request header or not. If it is there, then to check if it contains a value ‘Bearer’ or not.

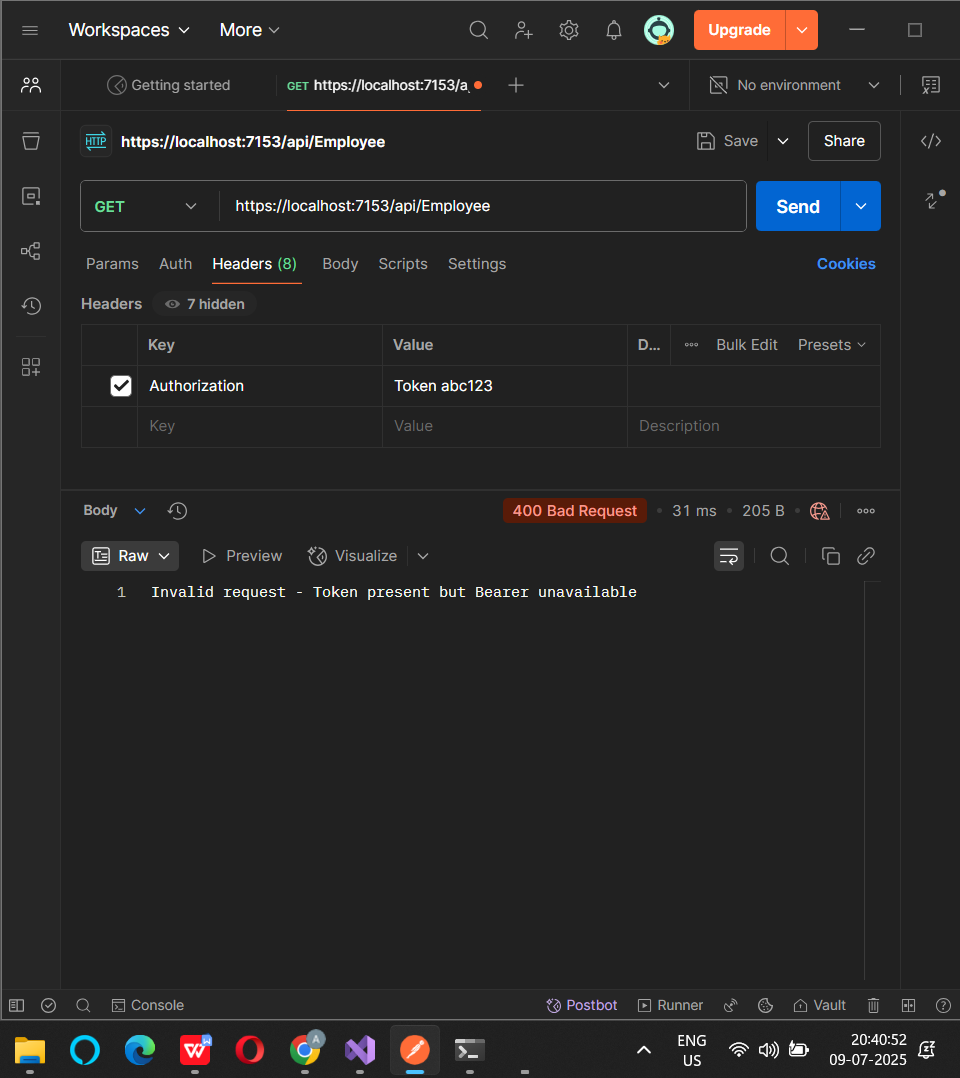
Create a folder ‘Filters’ in the application solution. Create a class ‘**CustomAuthFilter**’ to filter requests. Inherit ActionFilterAttribute. Override OnActionExecuting method to check if the request object has Header ‘Authorization’ or not. If not, throw BadRequestResult with the message

Invalid request - No Auth token

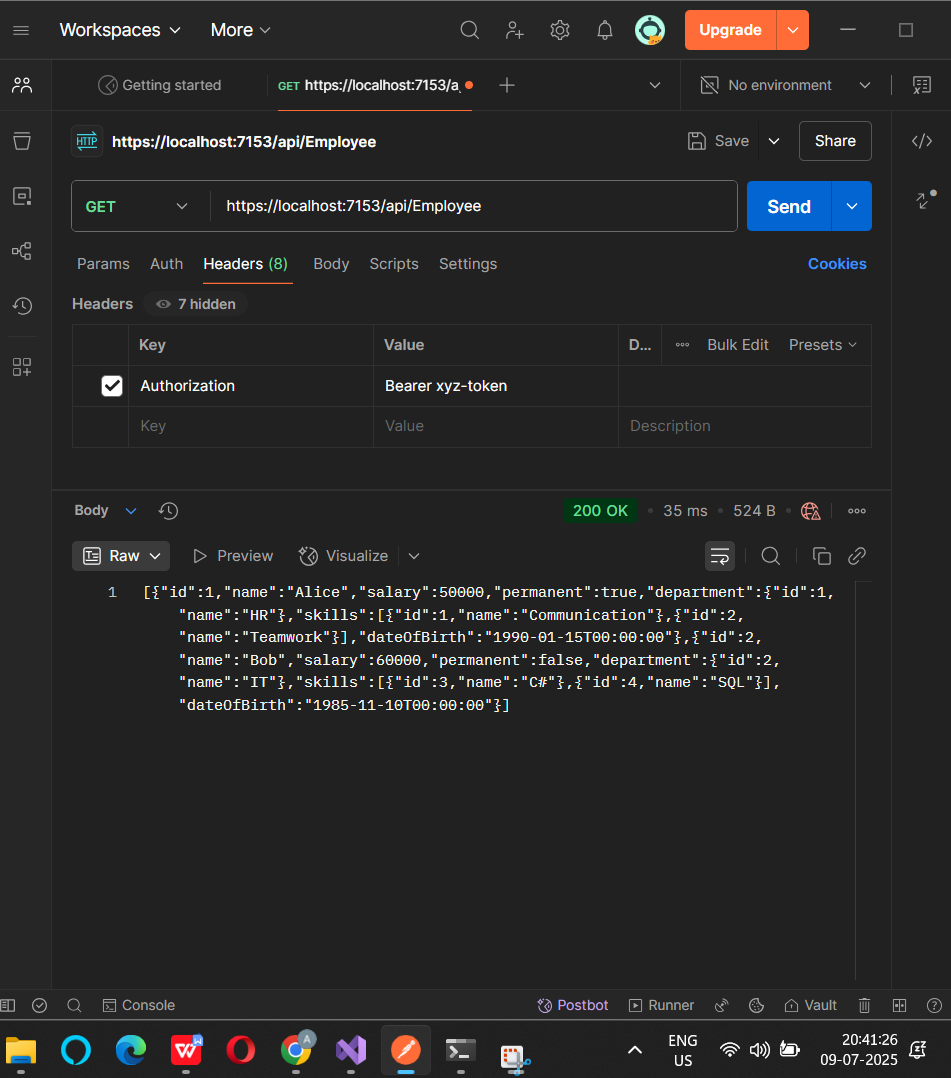


If the header is present, then check if the value contains the word ‘Bearer’. If not, throw BadRequestResult with the message

Invalid request - Token present but Bearer unavailable



Add an attribute **CustomAuthFilter** to the Employee controller to filter any request to check for the Authorization token in the request header.



**LAB 4:**

**Web Api CRUD operation**

Update Employee data as per the input thru Web API PUT action method call

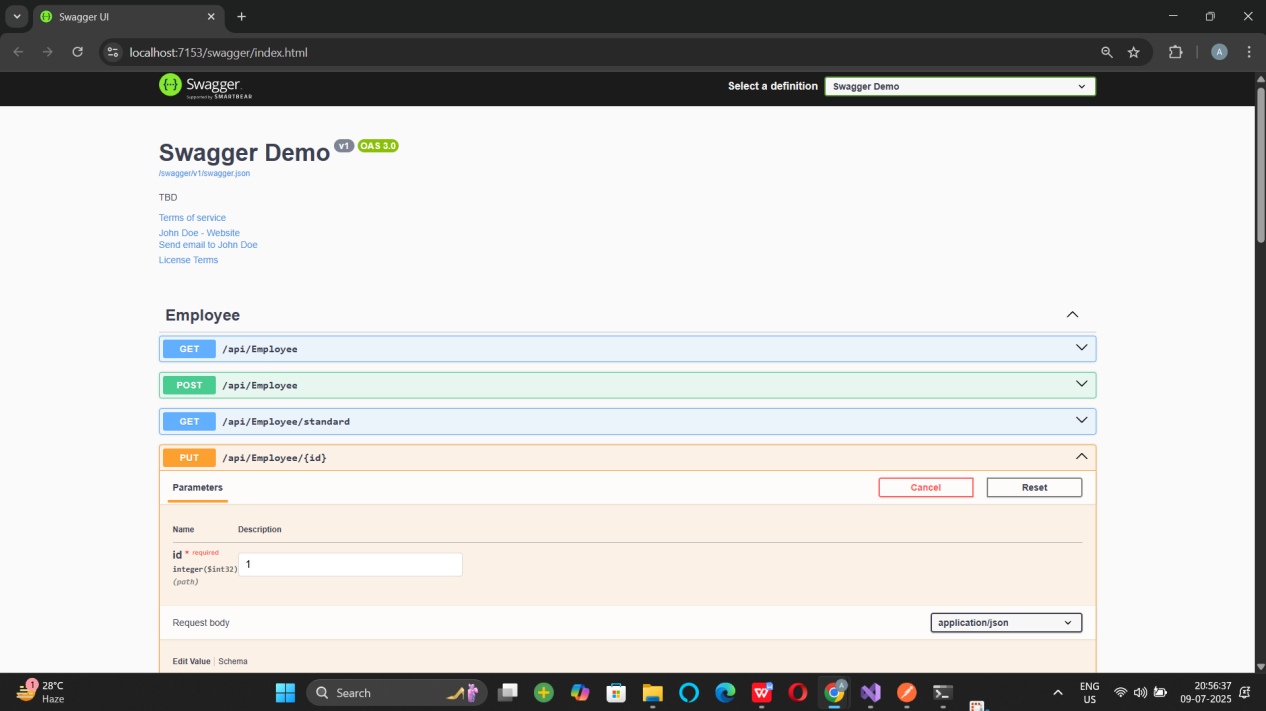
Employee information has to be updated based on the user input. Use Swagger tool to invoke the action method mapped with Http PUT action verb to update an employee data.

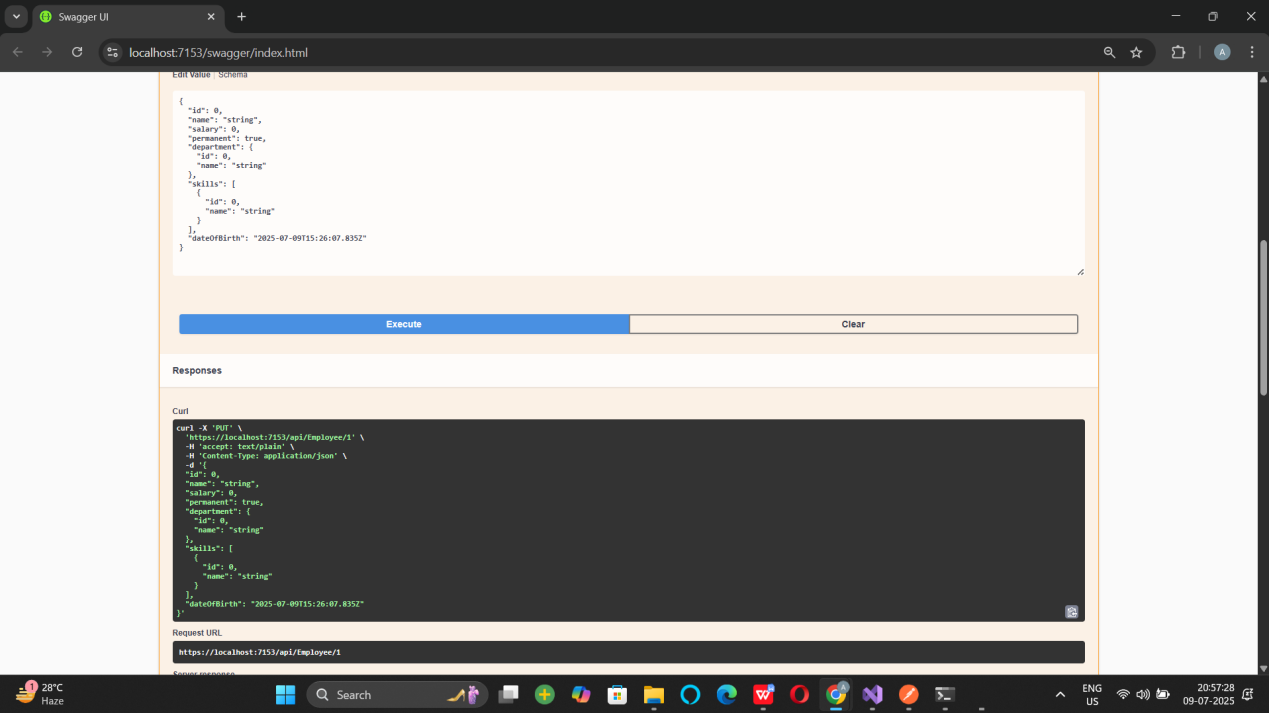
Modify the action method to return Employee data thru ActionResult.

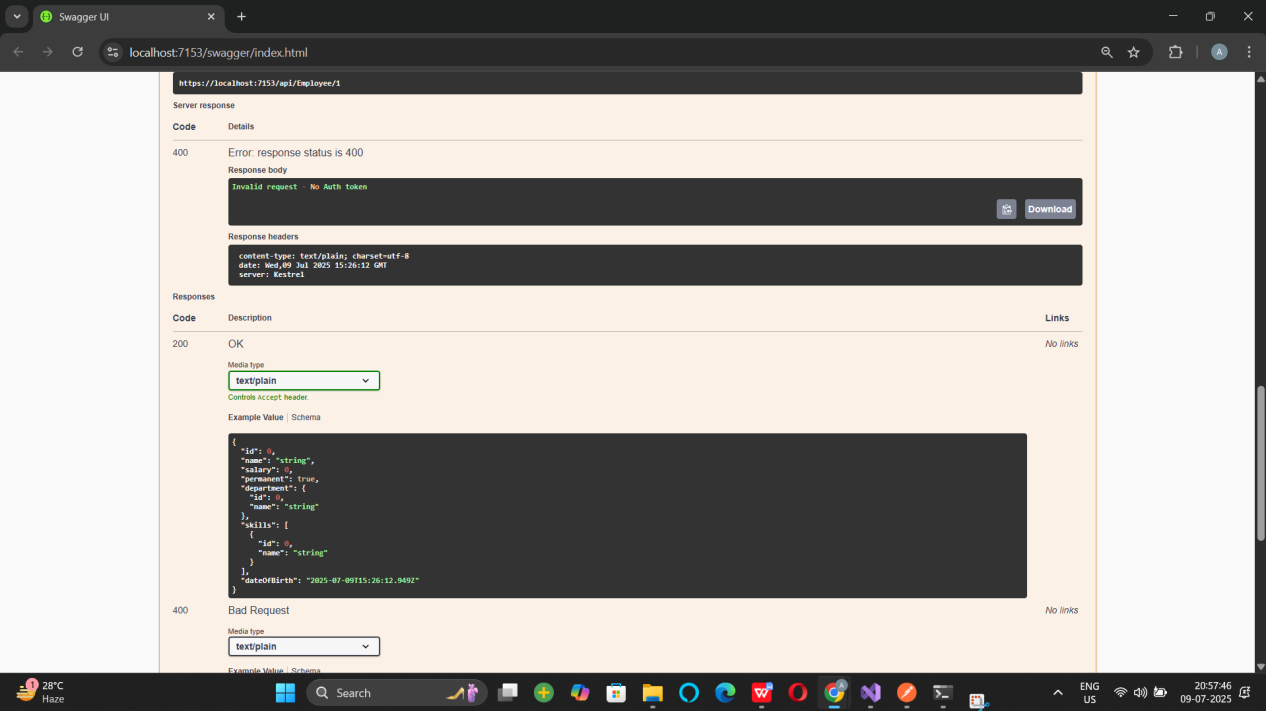
Check if the id value is lesser than or equal to 0. If true, throw BadRequest action result with the message ‘Invalid employee id’

If the value is greater than 0 but not available in the list of employee ids that is there in the hardcoded list of employees, throw BadRequest action result with the same message as stated above.

If the id value is valid, use the JSON data from the input body and update the hardcoded list. Filter the employee list data for the input id and return that as the output.







**LAB 5:**

**JsonWebToken**

There are various modes of authenticating a request. Json Web Token(JWT) authentication is one among them. It is a methodology of passing a token in the Authorization header value in the request so that it can be checked at the WebAPI and validated. If not there, then ‘**Unauthorized**’ status message with status code 401 should be thrown.

Use the below code in Startup.cs

* In ConfigureServices method

string securityKey = "mysuperdupersecret";

var symmetricSecurityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(securityKey));

services.AddAuthentication(x =>

{

x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;

x.DefaultSignInScheme = JwtBearerDefaults.AuthenticationScheme;

})

.AddJwtBearer(JwtBearerDefaults.AuthenticationScheme, x =>

{

x.TokenValidationParameters = new TokenValidationParameters

{

//what to validate

ValidateIssuer = true,

ValidateAudience = true,

ValidateLifetime = true,

ValidateIssuerSigningKey = true,

//setup validate data

ValidIssuer = "mySystem",

ValidAudience = "myUsers",

IssuerSigningKey = symmetricSecurityKey

};

});

* In Configure method

app.UseAuthentication();

This is to enable the JWT authentication in .Net core

Create a new controller ‘AuthController’ in the Web API application. Add **AllowAnonymous** attribute to the controller. Create a private method GenerateJSONWebToken as shown thru the code below.

private string GenerateJSONWebToken(int userId, string userRole)

{

var securityKey = new SymmetricSecurityKey(Encoding.UTF8.GetBytes("mysuperdupersecret"));

var credentials = new SigningCredentials(securityKey, SecurityAlgorithms.HmacSha256);

var claims = new List<Claim>

{

new Claim(ClaimTypes.Role, userRole),

new Claim("UserId", userId.ToString())

};

var token = new JwtSecurityToken(

issuer: "mySystem",

audience: "myUsers",

claims: claims,

expires: DateTime.Now.AddMinutes(10),

signingCredentials: credentials);

return new JwtSecurityTokenHandler().WriteToken(token);

}

Note that the issuer, audience and the securitykey defined in the Startup.cs code and method code shown above is the same and should match.

Invoke the GenerateJSONWebToken in the GET action method by sending some value for user id and ‘Admin’ for the user role. This is to set Claims information to check the user role

