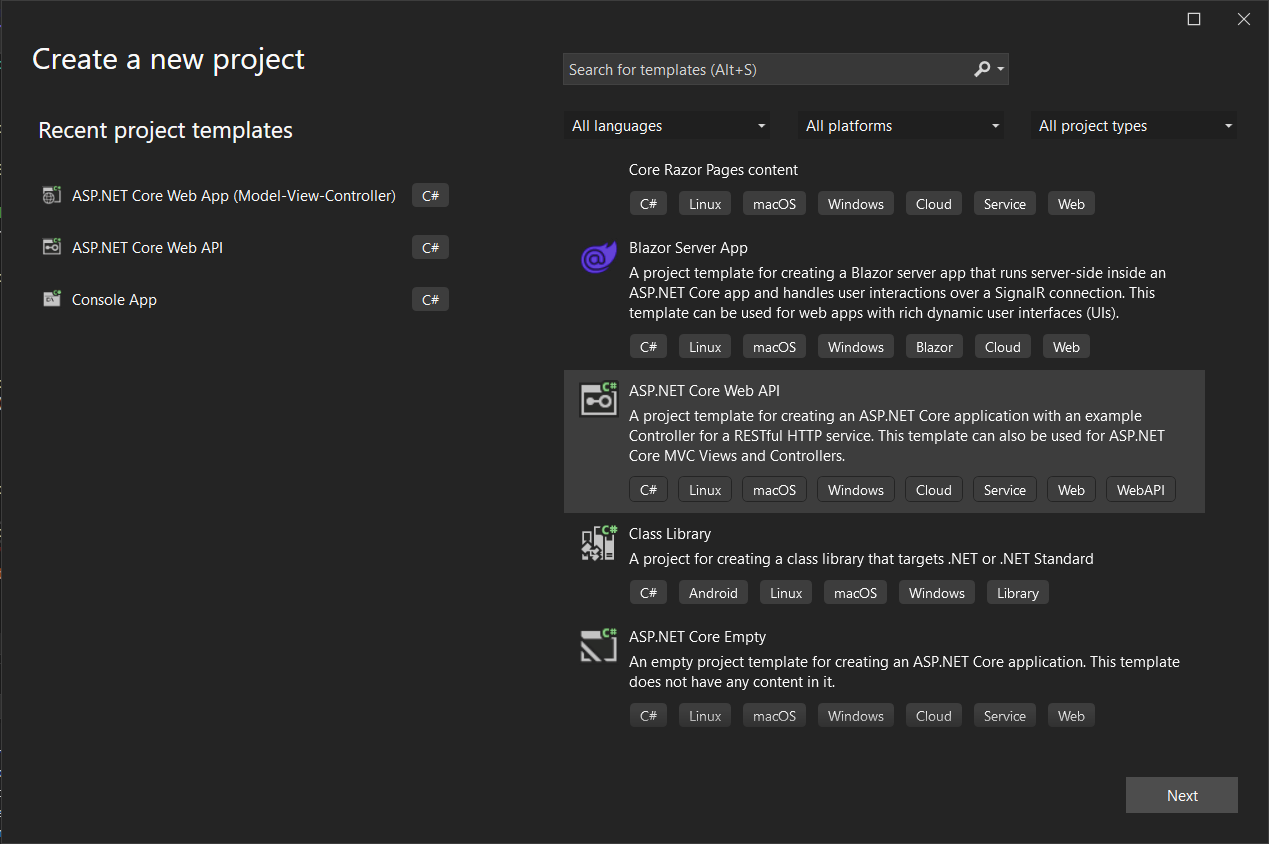
**Practical No. 02**

**Aim: Build a Web Application using API.**

Create a new project with **ASP.NET Core Web API**



**Code for *Program.cs*:**

var builder = WebApplication.CreateBuilder(args);

// Add services to the container.

builder.Services.AddControllers();

// Learn more about configuring Swagger/OpenAPI at https://aka.ms/aspnetcore/swashbuckle

builder.Services.AddEndpointsApiExplorer();

builder.Services.AddSwaggerGen();

var app = builder.Build();

// Configure the HTTP request pipeline.

if (app.Environment.IsDevelopment())

{

app.UseSwagger();

app.UseSwaggerUI();

}

app.UseHttpsRedirection();

app.UseAuthorization();

app.MapControllers();

app.Run();

**Code for *WeatherForecastController.cs* under the folder *Controllers*:**

using Microsoft.AspNetCore.Mvc;

namespace webappwithAPI.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();

}

}

}

**Code for *WeatherForecast.cs*:**

namespace webappwithAPI

{

public class WeatherForecast

{

public DateOnly Date { get; set; }

public int TemperatureC { get; set; }

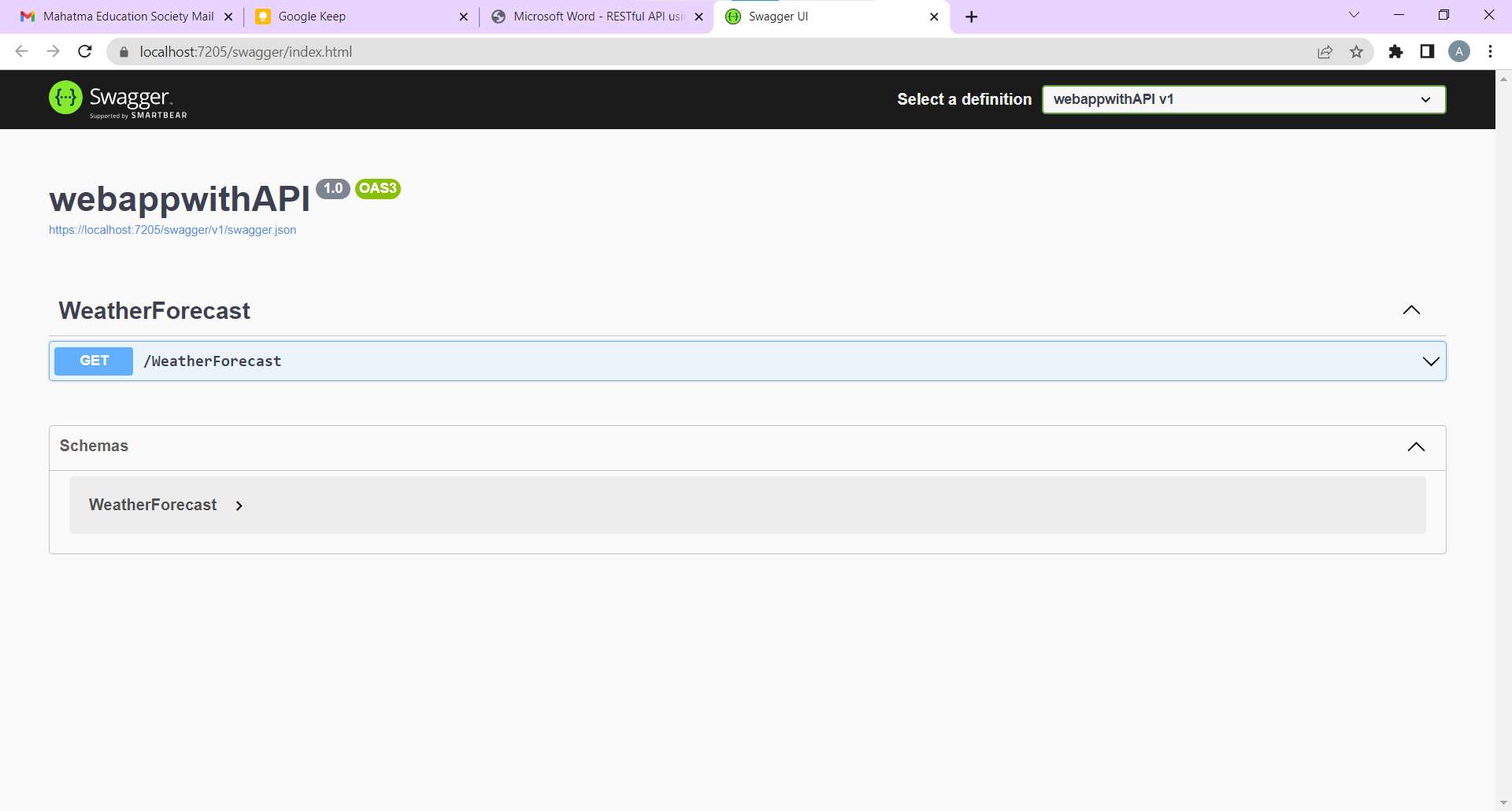
public int TemperatureF => 32 + (int)(TemperatureC / 0.5556);

public string? Summary { get; set; }

}

}

**Output:**

****

**Replace the *WeatherForecastController.cs* under the folder *Controllers* with *GlossaryController.cs*:**

//Controllers/GlossaryController.cs using System;

using System.Collections.Generic;

using Microsoft.AspNetCore.Mvc;

using System.IO;

namespace Glossary.Controllers

{

[ApiController]

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

public class GlossaryController : ControllerBase

{

private static List<GlossaryItem> Glossary = new List<GlossaryItem> { new GlossaryItem

{

Term= "HTML",

Definition = "Hypertext Markup Language"

},

new GlossaryItem

{

Term= "MVC",

Definition = "Model View Controller"

},

new GlossaryItem

{

Term= "OpenID",

Definition = "An open standard for authentication"

}

};

[HttpGet]

public ActionResult<List<GlossaryItem>> Get()

{

return Ok(Glossary);

}

[HttpGet]

[Route("{term}")]

public ActionResult<GlossaryItem> Get(string term)

{

var glossaryItem = Glossary.Find(item =>

item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{

return NotFound();

}

else

{

return Ok(glossaryItem);

}

}

[HttpPost]

public ActionResult Post(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item =>

item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));

if (existingGlossaryItem != null)

{

return Conflict("Cannot create the term because it already exists.");

}

else

{

Glossary.Add(glossaryItem);

var resourceUrl = Path.Combine(Request.Path.ToString(), Uri.EscapeUriString(glossaryItem.Term)); return Created(resourceUrl, glossaryItem);

}

}

[HttpPut]

public ActionResult Put(GlossaryItem glossaryItem)

{

var existingGlossaryItem = Glossary.Find(item => item.Term.Equals(glossaryItem.Term, StringComparison.InvariantCultureIgnoreCase));

if (existingGlossaryItem == null)

{

return BadRequest("Cannot update a nont existing term.");

}

else

{

existingGlossaryItem.Definition = glossaryItem.Definition; return Ok();

}

}

[HttpDelete]

[Route("{term}")]

public ActionResult Delete(string term)

{

var glossaryItem = Glossary.Find(item =>

item.Term.Equals(term, StringComparison.InvariantCultureIgnoreCase));

if (glossaryItem == null)

{

return NotFound();

}

else

{

Glossary.Remove(glossaryItem); return NoContent();

}

}

}

}

**Replace the *WeatherForecast.cs* with *GlossaryItem.cs*:**

//GlossaryItem.cs

namespace Glossary

{

public class GlossaryItem

{

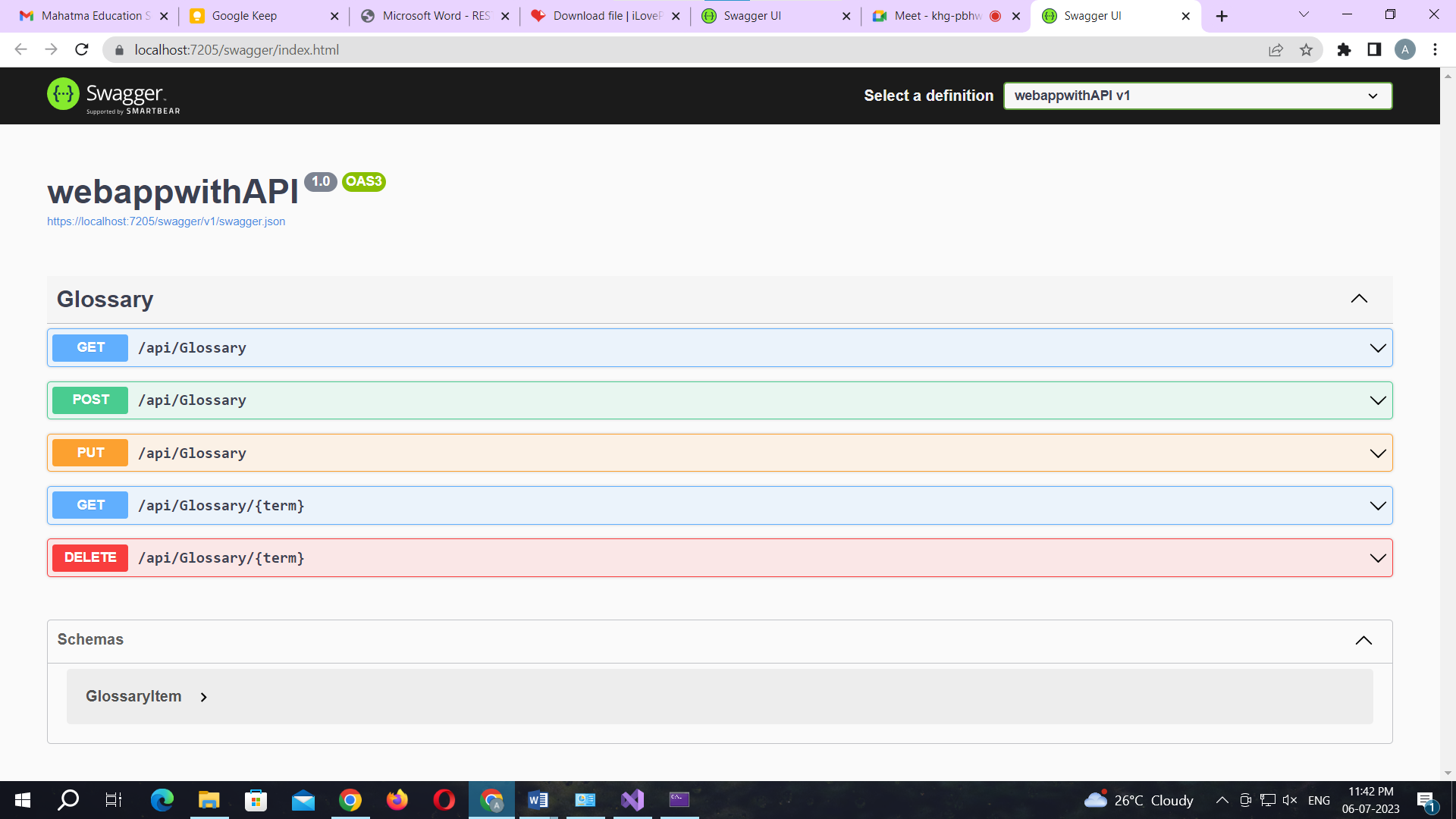
public string Term { get; set; }

public string Definition { get; set; }

}

}

**Output:**

****