

Necesito que actues como un experto en .net, En esta ocasion vas a realizar una api minimal en .net 8 con nombre DevSecOpsDemo.Api

Debes diseñar dos endpoint REST que son los siguientes:

1.- GET /api/health

ejemplo: Debe responder con un JSON indicando que el servicio está “ok”.

- Debe usar un código HTTP adecuado para éxito.

2.- POST /api/suma

- Debe recibir un body JSON con dos números enteros (por ejemplo A y B).

- Si el body es válido, debe devolver:

· Código de éxito.

· Un JSON con el resultado de la suma.

- Si el body es inválido o nulo, debe devolver:

· Un código HTTP de error del cliente.

· Un mensaje de error en el body.

DevSecOpsDemo.Api - Walkthrough

Resumen

Se ha creado exitosamente una API minimal en .NET 8 con el nombre DevSecOpsDemo.Api que incluye dos endpoints REST según las especificaciones solicitadas.

Cambios Implementados

Estructura del Proyecto

Se creó un nuevo proyecto de API minimal en .NET 8 en la ubicación:

```
# Start the application
cd C:\Users\ignac\gemin\antigravity\scratch\DevSecOpsDemo.Api
dotnet run
# In another terminal, test the health endpoint
curl -X GET http://localhost:5000/api/health
```

Endpoints Implementados

1. GET /api/health

Propósito: Health check endpoint para verificar que el servicio está operativo.

Características:

Retorna código HTTP 200 (OK)

Respuesta JSON: {"status":"ok"}

No requiere autenticación ni parámetros

Implementación:

Program.cs:19-24

2. POST /api/suma

Propósito: Endpoint para sumar dos números enteros.

Características:

Acepta un body JSON con dos propiedades: A y B (enteros)

Validación de entrada:

Body nulo o inválido → HTTP 400 (Bad Request) con mensaje de error

Body válido → HTTP 200 (OK) con el resultado de la suma

Respuesta exitosa: {"resultado": <suma>}

Respuesta de error: {"error": "El body de la solicitud no puede ser nulo"}

Implementación:
Program.cs:26-39

DTO Creado
Se definió un record
SumaRequest
para el endpoint de suma:

curl -X POST http://localhost:5000/api/suma -H "Content-Type: application/json" -d "{\"A\":5,\"B\":3}"
Pruebas Realizadas

Test 1: Health Check Endpoint ✓
Comando:

Invoke-WebRequest -Uri http://localhost:5000/api/health -Method GET
Resultado:

Status Code: 200 OK
Content: {"status": "ok"}
Content-Type: application/json; charset=utf-8
Test 2: Suma con Números Positivos ✓
Comando:

Invoke-WebRequest -Uri http://localhost:5000/api/suma -Method POST -ContentType "application/json" -Body
'{"A":5,"B":3}'
Resultado:

Status Code: 200 OK
Content: {"resultado": 8}
Validación: 5 + 3 = 8 ✓
Test 3: Suma con Números Negativos ✓
Comando:

Invoke-WebRequest -Uri http://localhost:5000/api/suma -Method POST -ContentType "application/json" -Body '{"A": -10, "B": 25}'
Resultado:

Status Code: 200 OK
Content: {"resultado": 15}
Validación: -10 + 25 = 15 ✓
Test 4: Body Inválido/Nulo ✓
Comando:

Invoke-WebRequest -Uri http://localhost:5000/api/suma -Method POST -ContentType "application/json" -Body ""
Resultado:

Status Code: 400 Bad Request
Validación: Error del cliente correctamente manejado ✓
Cómo Ejecutar la API
Compilar el Proyecto
cd C:\Users\ignac\gemini\antigravity\scratch\DevSecOpsDemo.Api
dotnet build
Ejecutar la API
dotnet run --urls "http://localhost:5000"
La API estará disponible en:

HTTP: http://localhost:5000
Swagger UI: http://localhost:5000/swagger (en modo Development)
Detener la API
Presionar Ctrl+C en la terminal donde se está ejecutando.

Características Adicionales
Swagger/OpenAPI: La API incluye documentación interactiva de Swagger disponible en modo Development
HTTPS Redirection: Configurado para redirigir automáticamente a HTTPS
Minimal API: Implementación moderna y ligera usando el patrón Minimal API de .NET
Conclusión

✓ Todos los requisitos han sido implementados y verificados:

GET /api/health retorna status "ok" con código HTTP 200
POST /api/suma suma dos números correctamente con código HTTP 200
POST /api/suma valida el body y retorna HTTP 400 para entradas inválidas
La API compila sin errores ni advertencias

Todos los tests manuales pasaron exitosamente

Gracias por realizar los test pero necesito que crees un proyecto de pruebas llamado DevSecOpsDemo.Tests, donde debes levantar la api en memoria hacer peticiones a sus endpoints (por ejemplo, usando WebApplicationFactory u otra estrategia equivalente de pruebas de integración) donde implementes por lo menos las siguientes tres pruebas:

1.- Prueba para GET /api/health:

- Verificar que devuelve el código HTTP correcto.
- Verificar que el body contiene la información de “status” esperada.

2.- Prueba para POST /api/suma – caso exitoso:

- Enviar dos números válidos.
- Verificar código HTTP correcto.
- Verificar que el resultado de la suma es correcto en el body.

3.- Prueba para POST /api/suma – caso inválido:

- Enviar body nulo o inválido.
- Verificar que se devuelve el código HTTP de error correcto.
- Verificar que se retorna un mensaje de error acorde.

Ademas realiza pruebas adicionales y arma una estructura del proyecto que consideres necesaria y util para la solucion

DevSecOpsDemo - Walkthrough Completo

Resumen

Se ha creado exitosamente una solución completa de .NET 8 que incluye:

DevSecOpsDemo.Api - API minimal con dos endpoints REST

DevSecOpsDemo.Tests - Proyecto de pruebas de integración con 12 tests

Estructura del Proyecto

DevSecOpsDemo/

```
|—— DevSecOpsDemo.Api/  
|   |—— Program.cs      # API minimal con endpoints  
|   |—— DevSecOpsDemo.Api.csproj  
|   |—— appsettings.json
```

```
└—— DevSecOpsDemo.Tests/  
    |—— HealthEndpointTests.cs  # 3 pruebas para /api/health  
    |—— SumaEndpointTests.cs   # 9 pruebas para /api/suma
```

```
|--- Usings.cs           # Global usings  
└-- DevSecOpsDemo.Tests.csproj
```

Parte 1: API Implementation

Endpoints Implementados

1. GET /api/health

Ubicación:

Program.cs:19-26

Características:

- Retorna HTTP 200 (OK)
- Respuesta JSON: {"status":"ok"}
- No requiere parámetros ni autenticación

Código:

```
app.MapGet("/api/health", () =>  
{  
    return Results.Ok(new { status = "ok" });  
})  
.WithName("GetHealth")  
.WithOpenApi();
```

2. POST /api/suma

Ubicación:

Program.cs:28-43

Características:

- Acepta JSON con propiedades A y B (enteros)
- Validación de entrada (retorna HTTP 400 si body es nulo)
- Retorna HTTP 200 con resultado de la suma
- Mensaje de error descriptivo para casos inválidos

Código:

```
app.MapPost("/api/suma", (SumaRequest? request) =>  
{  
    if (request == null)  
    {  
        return Results.BadRequest(new { error = "El body de la solicitud no puede ser nulo" });  
    }
```

```
    var resultado = request.A + request.B;
    return Results.Ok(new { resultado });
}

.WithName("PostSuma")
.WithOpenApi();
```

DTO:

```
record SumaRequest(int A, int B);
```

Modificación para Testing

Ubicación:

Program.cs:50-51

Se agregó una clase parcial

Program

para permitir que WebApplicationFactory acceda al punto de entrada:

```
public partial class Program {}
```

Parte 2: Integration Tests

Configuración del Proyecto de Pruebas

Paquetes NuGet instalados:

Microsoft.AspNetCore.Mvc.Testing v8.0.11 - Para WebApplicationFactory

FluentAssertions v8.8.0 - Para assertions legibles

xunit - Framework de testing

Referencia al proyecto DevSecOpsDemo.Api

Archivos de Prueba

Usings.cs

Usings.cs

Global usings para reducir código repetitivo:

```
global using Xunit;
global using FluentAssertions;
global using Microsoft.AspNetCore.Mvc.Testing;
global using System.Net;
global using System.Net.Http.Json;
HealthEndpointTests.cs
HealthEndpointTests.cs
```

3 pruebas implementadas:

- GetHealth_ReturnsOk_WithCorrectStatus (REQUERIDA)

Verifica HTTP 200

Verifica que el body contiene "status": "ok"

Deserializa y valida el objeto JSON completo

- GetHealth_ReturnsJsonContentType

Verifica Content-Type: application/json

- GetHealth_IsIdempotent

Verifica que múltiples llamadas retornan el mismo resultado

SumaEndpointTests.cs

SumaEndpointTests.cs

9 pruebas implementadas:

Pruebas Requeridas

- PostSuma_WithValidNumbers_ReturnsOkWithCorrectSum (REQUERIDA #2)

Envía {A: 5, B: 3}

Verifica HTTP 200

Verifica resultado: 8

- PostSuma_WithNullBody_ReturnsBadRequest (REQUERIDA #3)

Envía body vacío

Verifica HTTP 400

Verifica mensaje de error

- PostSuma_WithInvalidJson_ReturnsBadRequest (REQUERIDA #3 - adicional)

Envía JSON malformado

Verifica HTTP 400

Pruebas Adicionales - Casos Exitosos

- PostSuma_WithNegativeNumbers_ReturnsCorrectSum

Prueba: $-10 + 25 = 15$

- PostSuma_WithZeroValues_ReturnsZero

Prueba: $0 + 0 = 0$

PostSuma_WithLargeNumbers_ReturnsCorrectSum

Prueba: $1000000 + 2000000 = 3000000$

PostSuma_WithMixedPositiveNegative_ReturnsCorrectSum

Prueba: $100 + (-50) = 50$

PostSuma_ReturnsJsonContentType

Verifica Content-Type: application/json

Pruebas Adicionales - Casos de Error

PostSuma_WithoutContentType_ReturnsBadRequest

Verifica que falla sin Content-Type apropiado

Resultados de las Pruebas

Ejecución Completa

Comando ejecutado:

```
cd C:\Users\ignac\gemin\antigravity\scratch\DevSecOpsDemo.Tests
```

```
dotnet test --verbosity normal
```

Resultados

La serie de pruebas se ejecutó correctamente.

Pruebas totales: 12

Correcto: 12

Fallido: 0

Tiempo total: 1.5 Segundos

Desglose por Clase de Prueba

Clase de Prueba	Pruebas	Estado
-----------------	---------	--------

HealthEndpointTests	3	<input checked="" type="checkbox"/> Todas pasaron
---------------------	---	---

SumaEndpointTests	9	<input checked="" type="checkbox"/> Todas pasaron
-------------------	---	---

TOTAL	12	<input checked="" type="checkbox"/> 100% éxito
-------	----	--

Tiempos de Ejecución

Primera prueba (inicialización): ~500ms

Pruebas subsecuentes: <1ms - 9ms

Tiempo total: 1.5 segundos

Pruebas Requeridas vs Implementadas

Requisitos Cumplidos

#	Requisito	Estado	Prueba(s)	Implementada(s)
---	-----------	--------	-----------	-----------------

1	GET /api/health	- HTTP correcto y status	<input checked="" type="checkbox"/>
---	-----------------	--------------------------	-------------------------------------

GetHealth_ReturnsOk_WithCorrectStatus

2 POST /api/suma - Caso exitoso

PostSuma_WithValidNumbers_ReturnsOkWithCorrectSum

3 POST /api/suma - Caso inválido

PostSuma_WithNullBody_ReturnsBadRequest

PostSuma_WithInvalidJson_ReturnsBadRequest

Pruebas Adicionales (9 pruebas extra)

Health Endpoint (2 adicionales):

Validación de Content-Type

Verificación de idempotencia

Suma Endpoint - Casos Exitosos (4 adicionales):

Números negativos

Valores cero

Números grandes

Mezcla positivo/negativo

Validación de Content-Type

Suma Endpoint - Casos de Error (1 adicional):

Sin Content-Type header

Características de la Implementación

Patrón WebApplicationFactory

Todas las pruebas utilizan WebApplicationFactory<Program> para:

- Levantar la API en memoria
- No requerir servidor externo
- Ejecución rápida y aislada
- Configuración automática del entorno de pruebas

Ejemplo de uso:

```
public class HealthEndpointTests : IClassFixture<WebApplicationFactory<Program>>
{
    private readonly WebApplicationFactory<Program> _factory;
    public HealthEndpointTests(WebApplicationFactory<Program> factory)
    {
        _factory = factory;
    }
    [Fact]
```

```
public async Task GetHealth_ReturnsOk_WithCorrectStatus()
{
    var client = _factory.CreateClient();
    var response = await client.GetAsync("/api/health");
    // assertions...
}
```

Patrón Arrange-Act-Assert

Todas las pruebas siguen el patrón AAA:

```
// Arrange - Preparar datos y cliente
var client = _factory.CreateClient();
var request = new { A = 5, B = 3 };

// Act - Ejecutar la acción
var response = await client.PostAsJsonAsync("/api/suma", request);

// Assert - Verificar resultados
response.StatusCode.Should().Be(HttpStatusCode.OK);

var result = await response.Content.ReadFromJsonAsync<SumaResponse>();
result!.Resultado.Should().Be(8);
```

FluentAssertions

Uso de FluentAssertions para assertions legibles:

```
response.StatusCode.Should().Be(HttpStatusCode.OK, "el endpoint debe retornar HTTP 200");
```

```
result!.Resultado.Should().Be(8, "5 + 3 debe ser igual a 8");
```

Cómo Ejecutar

Compilar la Solución

```
# Compilar la API
```

```
cd C:\Users\ignac\gemini\antigravity\scratch\DevSecOpsDemo.Api
```

```
dotnet build
```

```
# Compilar las pruebas
```

```
cd C:\Users\ignac\gemini\antigravity\scratch\DevSecOpsDemo.Tests
```

```
dotnet build
```

Ejecutar las Pruebas

```
cd C:\Users\ignac\gemini\antigravity\scratch\DevSecOpsDemo.Tests
```

```
dotnet test --verbosity normal
```

Ejecutar la API Manualmente

```
cd C:\Users\ignac\gemini\antigravity\scratch\DevSecOpsDemo.Api
```

```
dotnet run --urls "http://localhost:5000"
```

La API estará disponible en:

HTTP: <http://localhost:5000>

Swagger UI: <http://localhost:5000/swagger>

Conclusión

Todos los requisitos cumplidos:

API minimal en .NET 8 con dos endpoints REST funcionales

Proyecto de pruebas de integración con WebApplicationFactory

3 pruebas requeridas implementadas y pasando

9 pruebas adicionales para casos edge y validaciones extras

12/12 pruebas pasando (100% éxito)

Estructura de proyecto profesional y escalable

Código limpio siguiendo mejores prácticas de .NET

Beneficios de la Implementación

Testing en Memoria: No requiere servidor externo, pruebas rápidas

Cobertura Completa: Casos exitosos, errores, y edge cases

Mantenibilidad: Código organizado y bien documentado

Extensibilidad: Fácil agregar nuevos endpoints y pruebas

CI/CD Ready: Las pruebas pueden ejecutarse en pipelines automáticamente