Ejercicio propuesto para practicar políticas de resiliencia con Polly:

La idea de este ejercicio es trabajar con 2 web apis locales (api A y api B), donde la api A será la api principal que implementará el patrón HttpClientFactory y tendrá configurada como cliente a la api B para consumir uno de sus endpoints. Entonces en la api A instalaremos la librería Polly, y configuraremos políticas de Retry y de Circuit Breaker, y haremos que el endpoint de la api B falle intencionalmente algunas veces para poder probar el funcionamiento de las políticas de Polly.

Vamos paso a paso:

1. Crea un nuevo proyecto de tipo web api y en la carpeta Controllers agrega un nuevo controlador que contenga el siguiente código:

```
[ApiController]
[Route("api/[controller]")]
public class TestController : ControllerBase
{
    private static int _requestCount = 0;
    private int _requestUmbral = 5;

    private readonly ILogger<TestController> _logger;

    public TestController(ILogger<TestController> logger)
    {
        _logger = logger;
    }

    [HttpGet("unstable")]
    public IActionResult GetUnstableResponse()
    {
        _requestCount++;

        _logger.LogWarning($"Recibimos request el: {DateTime.Now} - contador: {_requestCount}");

        if (_requestCount <= _requestUmbral) // Simula fallas las primeras 5 veces
        {
            return StatusCode(500, "Internal Server Error simulated");
        }

        return Ok(new { message = "Success after failures", attempts = _requestCount });
    }

    [HttpGet("reset")]
    public IActionResult Reset()
    {
        _requestCount = 0; // Reseteamos el contador de solicitudes entrantes
        _logger.LogInformation($"Se reseteó el contador el: {DateTime.Now}");
        return Ok("Counter reset");
    }
}
```

Esta api será nuestra api B (la api cliente).

2. Luego, en el proyecto del ejercicio anterior que consumía la api de chistes, deberías agregar algunos nuevos archivos para que ahora, además de poder consumir la api de chistes también pueda consumir el endpoint de nuestra nueva api B.
   Estos archivos deberían ser:
   - SolutionName.Application.Models.ApiBResponseDTO;
   - SolutionName.Application.Interfaces.IApiBService;
   - SolutionName.Infrastructure.Services.ApiBService;
   - SolutionName.Presentation.Controllers.CallController;

   Procura configurar correctamente todo para que la api A se pueda comunicar con la api B. Y por supuesto también agrega en Program.cs la configuración del nuevo cliente HttpClient "apiB" con su baseAddress.

3. A continuación instala los paquetes necesarios para poder utilizar Polly en la api A y configurar las 2 políticas. (puedes guiarte del apunte)

4. Crea en en la capa de infraestructura de la api A estas 2 nuevas clases:
   .PollyResiliencePolicies: aquí colocarás la configuración general de ambas políticas de resiliencia, la de reintentos y la de cortocircuito. (Puedes guiarte del apunte y del repo de ejemplo bookchampion).
   .ApiClientConfiguration: esta clase será necesaria para setear los parámetros que utilizaremos para configurar las políticas. (también puedes guiarte del apunte y del repo de ejemplo bookchampion).

5. Instancia dentro de la clase Program.cs una nueva instancia de ApiClientConfiguration justo antes de agregar los clientes HttpClient, y configura los valores que quieras. Recomiendo cambiar el intervalo exponencial de reintentos por uno fijo para comenzar.

6. Pon a correr ambas apis e intenta ejecutar el endpoint de la api A que consume al endpoint de la api B para probar el funcionamiento de las nuevas políticas. Puedes revisar los logs de ambas consolas para intentar entender el funcionamiento.
   (NOTA: juega cambiando los valores de la instancia de ApiClientConfiguration y del contador del edpoint de la api B para conseguir diferentes resultados)

EXTRA:

Por defecto, la consola de ASP.NET nos mostrará unos logs con nivel de info cada vez que se prepare y envíe una request, y cada vez que se recibe una response.

Pero a los fines de este ejercicio, también podríamos agregar algunas líneas de código en la clase donde están configuradas las políticas de Polly, para que se agreguen unos logs caseros por la consola cada vez que se hace un reintento y cada vez que se abre o cierra el circuito.

Podríamos hacerlo así:

```
public static class PollyResiliencePolicies
{
    public static IAsyncPolicy<HttpResponseMessage> GetRetryPolicy(ApiClientConfiguration config)
    {
        return HttpPolicyExtensions
            .HandleTransientHttpError()
            .OrResult(msg => msg.StatusCode == System.Net.HttpStatusCode.BadRequest)
            .WaitAndRetryAsync(
                config.RetryCount,
                retryAttempt => TimeSpan.FromSeconds(config.RetryAttemptInSeconds),
                //retryAttempt => TimeSpan.FromSeconds(Math.Pow(2, retryAttempt) * config.RetryAttemptInSeconds)
                onRetry: (outcome, timespan, retryAttempt, context) =>
                {
                    Console.ForegroundColor = ConsoleColor.Yellow;
                    Console.WriteLine($"[Polly Retry] Intento #{retryAttempt} en {timespan.TotalSeconds}s. Motivo: {GetFailureReason(outcome)}");
                    Console.ResetColor();
                }
            );
    }

    public static IAsyncPolicy<HttpResponseMessage> GetCircuitBreakerPolicy(ApiClientConfiguration config)
```

```
public static IAsyncPolicy<HttpResponseMessage> GetCircuitBreakerPolicy(ApiClientConfiguration config)
{
    return HttpPolicyExtensions
        .HandleTransientHttpError()
        .OrResult(msg => msg.StatusCode == System.Net.HttpStatusCode.BadRequest)
        .CircuitBreakerAsync(config.HandledEventsAllowedBeforeBreaking, TimeSpan.FromSeconds(config.DurationOfBreakInSeconds),
        onBreak: (outcome, breakDelay) =>
        {
            Console.ForegroundColor = ConsoleColor.Red;
            Console.WriteLine($"[Polly Circuit Breaker] Circuito ABIERTO por {breakDelay.TotalSeconds}s. Motivo: {GetFailureReason(outcome)}");
            Console.ResetColor();
        },
        onReset: () =>
        {
            Console.ForegroundColor = ConsoleColor.Green;
            Console.WriteLine("[Polly Circuit Breaker] Circuito CERRADO. Operación normal reanudada.");
            Console.ResetColor();
        },
        onHalfOpen: () =>
        {
            Console.ForegroundColor = ConsoleColor.Cyan;
            Console.WriteLine("[Polly Circuit Breaker] Circuito en estado SEMI-ABIERTO. Probando conexión...");
            Console.ResetColor();
        });
}

private static string GetFailureReason(DelegateResult<HttpResponseMessage> outcome)
{
    if (outcome.Exception != null)
        return outcome.Exception.Message;

    if (outcome.Result != null)
        return $"HTTP {(int)outcome.Result.StatusCode} ({outcome.Result.StatusCode})";

    return "Error desconocido";
}
```

Agregar estos nuevos logs a los que nos ofrece ASP.NET por defecto, ayuda a rastrear mejor el funcionamiento de los reintentos y la apertura y cierre del circuito.