DESARROLLO DE BACKEND III

GONZALEZ DIAZ ANSELMO ALEXIS

Aarón Hernández García

HttpClient

31/03/2025

¿Qué es HttpClient?

HttpClient es una clase proporcionada por .NET que permite enviar solicitudes HTTP y recibir respuestas de recursos identificados por una URI. Es una herramienta esencial para interactuar con APIs web y realizar operaciones como GET, POST, PUT y DELETE.

En C#, trabajar con *HttpClient* requiere comprender cómo crearlo correctamente, implementar middleware, garantizar la resiliencia, manejar reintentos, usar disyuntores y optimizar la ejecución de solicitudes.

Crear cliente HTTP

Necesitas saber cómo crear *HttpClient* correctamente antes de usarlo. Hay varias opciones, con sus pros y sus contras:

- Constructor
- Instancia estática
- IHttpClientFactory
- Clientes nombrados
- Clientes tipificados
- Clientes generados *

Constructor

La forma más sencilla de trabajar con el *HttpClient* simplemente crear una nueva instancia y *.Dispose()* terminarla.

```
aplicación.MapGet( "/" , () =>
{
    HttpClient  cliente  =  new  HttpClient ();
                . . .
    cliente.Dispose();
});
```

Sin embargo, *HttpClient* un poco especial y no se desecha de forma adecuada.

Con cada *HttpClient* instancia se crea una nueva conexión HTTP. Sin embargo, incluso cuando se elimina el cliente, el socket TCP no se libera inmediatamente. Si la aplicación crea nuevas conexiones constantemente, puede
agotar los puertos disponibles.

Uso básico de HttpClient

A continuación, se muestra un ejemplo de cómo utilizar *HttpClient* para realizar una solicitud GET y obtener el contenido de una página web:

```
using System;
using System.Net.Http;
using System.Threading.Tasks;

class Program
{
    private static readonly HttpClient client = new HttpClient();

    static async Task Main(string[] args)
    {
        try
        {
            HttpResponseMessage response = await client.GetAsync("https://www.example.com");
            response.EnsureSuccessStatusCode();
            string responseBody = await response.Content.ReadAsStringAsync();
            Console.WriteLine(responseBody);
        }
        catch (HttpRequestException e)
        {
            Console.WriteLine($"Mensaje de error: {e.Message}");
        }
    }
}
```

En este ejemplo:

- Se crea una instancia estática de *HttpClient* para reutilizarla en múltiples solicitudes, lo cual es una práctica recomendada para optimizar el uso de recursos.
- Se envía una solicitud GET a *"https://www.example.com"*.
- Se asegura que la respuesta sea exitosa con *EnsureSuccessStatusCode()*.
- Se lee y muestra el contenido de la respuesta.

Consideraciones al usar HttpClient

- **Reutilización de instancias**: Es recomendable reutilizar una única instancia de *HttpClient* durante la vida útil de la aplicación para evitar el agotamiento de los sockets y mejorar el rendimiento.
- **Manejo de excepciones**: Es importante manejar las excepciones que puedan surgir durante las solicitudes HTTP para evitar que la aplicación falle inesperadamente.
- **Configuración de cabeceras**: Puedes configurar cabeceras HTTP para enviar información adicional en las solicitudes, como se muestra a continuación:

```
client.DefaultRequestHeaders.Accept.Clear();
client.DefaultRequestHeaders.Accept.Add(new MediaTypeWithQualityHeaderValue("application/json"));
```

Este código configura la cabecera "Accept" para indicar que se espera una respuesta en formato JSON.

Uso avanzado de HttpClient

Para operaciones más complejas, como enviar datos en una solicitud POST, puedes utilizar el siguiente enfoque:

```
var values = new Dictionary<string, string>
{
    { "parametro1", "valor1" },
    { "parametro2", "valor2" }
};

var content = new FormUrlEncodedContent(values);

var response = await client.PostAsync("https://www.example.com/api/datos", content);

var responseString = await response.Content.ReadAsStringAsync();
```

En este ejemplo:

- Se crea un diccionario con los datos a enviar.
- Se encapsulan los datos en un FormUrlEncodedContent.
- Se envía una solicitud POST a la URL especificada con el contenido.
- Se lee la respuesta como una cadena.

Diferentes librerías disponibles

Además de HttpClient, existen otras bibliotecas en .NET que facilitan las peticiones HTTP y la gestión de APIs:

- **RestSharp**: Una biblioteca popular que simplifica las llamadas HTTP y la manipulación de respuestas.
- **Refit**: Biblioteca que genera clientes de API a partir de interfaces definidas en C#.
- **Flurl**: Proporciona una API fluida para realizar peticiones HTTP de manera sencilla y legible.

Ejemplo de uso de RestSharp:

```
var client = new RestClient("https://api.example.com");
var request = new RestRequest("/datos", Method.Get);
var response = await client.ExecuteAsync(request);
Console.WriteLine(response.Content);
```

Datos básicos para llamadas a APIs

Para realizar llamadas a una API, es importante conocer algunos conceptos clave:

- **Endpoints**: Son las URLs específicas donde se hacen las solicitudes.
- **Métodos HTTP**: Los más comunes son GET, POST, PUT y DELETE.
- **Cabeceras (Headers)**: Información adicional que se envía en las solicitudes, como Authorization para autenticación.
- **Códigos de estado HTTP**: Indican el resultado de la solicitud, por ejemplo:
  - 200 OK: Respuesta exitosa.
  - 400 Bad Request: Solicitud incorrecta.
  - 401 Unauthorized: Se requiere autenticación.
  - 500 Internal Server Error: Error del servidor.

Deserialización de datos

Cuando se recibe una respuesta en formato JSON, es necesario deserializarla para convertirla en objetos C# y manipularla de forma sencilla. Para ello, se pueden usar bibliotecas como System.Text.Json o Newtonsoft.Json.

Ejemplo con System.Text.Json:

```
using System.Text.Json;

public class Usuario
{
    public string Nombre { get; set; }
    public int Edad { get; set; }
}

string json = "{ \"Nombre\": \"Juan\", \"Edad\": 30 }";
Usuario usuario = JsonSerializer.Deserialize<Usuario>(json);
Console.WriteLine($"Nombre: {usuario.Nombre}, Edad: {usuario.Edad}");
```

Referencias:

- Microsoft. (s.f.). *Realización de solicitudes HTTP con HttpClient - .NET*. Microsoft Learn. Recuperado el 1 de abril de 2025, de *https://learn.microsoft.com/es-es/dotnet/fundamentals/networking/http/httpclient*
- Iamprovidence, W. by. (2025, enero 17). HTTP Client in C#: Best Practices for Experts - iamprovidence. Medium. *https://medium.com/@iamprovidence/http-client-in-c-best-practices-for-experts-840b36d8f8c4*