Documentación Solución Stress Testing FortuneProgrammers

Por Sergio Monfort e Iker Ramírez

Indice

Background del proyecto:

Para este proyecto, se nos proporcionó un programa en .NET con distintos endpoints los cuales había que someter a pruebas de rendimiento. Las funcionalidades que debíamos probar eran las siguientes:

-GET/GameData (Devuelve toda la información)
-GET/GameData/id (Devuelve la información de una sola entrada)
-POST/GameData (Inserta un nuevo objeto en la BBDD)
-PUT/GameData/id (Actualiza la información de un juego)
-DELETE/GameData (Elimina la información de un juego)

La idea era hacer un SmokeTest, un Average-load y un SpikeTest, además de comprobar los límites de la aplicación.

Usage:

El programa es un ejecutable StressTestingAppV2.exe que, al ejecutarlo, abrirá una aplicación de consola que mostrará los distintos tests y sus resultados. Durante la ejecución, se pedirá el input del usuario para pasar de un test al siguiente.

Instalación:

La solución no requiere ninguna instalación extra además de las necesarias para poder ejecutar el programa original con el que comenzamos.

Stack Utilizado:

Para hacer la solución, las librerías utilizadas han sido:

- Microsoft.Asp.Net.Core.Mvc.Testing (7.0.14)
  (Para acceder a la WebAplicationFactory)

- Microsoft.EntityFrameworkCore.Relational (7.0.14)
  (Para las peticiones)

Para el control de versiones, se ha utilizado GitLab y, para crear y repartir el trabajo, Trello.

Toma de decisiones:

En un principio, hicimos una versión V1 para poder comprobar de manera manual los límites y problemas que nos podrían suceder durante el desarrollo de la versión final (El código de la V1 se ha mantenido en la entrega por si se desea revisar). En la primera iteración, se probarían los distintos endpoints en diversas cantidades y a la vez, para ver cómo se comportaría el sistema. Durante el desarrollo de la primera versión, cada uno de nosotros intentó distintos métodos para generar los tests: usando librerías como NBench, añadiendo Javascript, o creando proyectos de msTesting.

Cuando conseguimos los resultados deseados, decidimos hacer un modelo más real para hacer las verdaderas pruebas. Finalmente, nos decantamos por una aplicación de escritorio. En este modelo, simulamos usuarios que entrarían con distintos tiempos y que harían distintas acciones con pequeños retrasos entre estas para reproducir las acciones de un cliente. Durante este proceso, ambos nos centramos en el desarrollo de un mismo código. También se añadió la línea “public partial class Program { }” al final del código en Program.cs de GameStatistics para poder hacer las peticiones.

Finalmente, Sergio creó los casos de prueba (“opciones” de los clientes) e Iker elaboró la documentación.

Resultados:

Con la primera versión, se encontró que el límite de conexiones simultáneas que aceptaba la base de datos era de 100 usuarios, también se vieron los potenciales problemas de que múltiples llamadas intentasen borrar a la vez una misma entrada de la base de datos. Por ello, en la segunda versión se hizo que cada thread se ejecutase con un pequeño desfase de tiempo y que, entre cada acción, el cliente esperase entre 10ms y 50ms para simular un comportamiento más parecido a lo que nos encontraríamos en el mundo real.

En la respuesta entregada, se crean clientes que pueden hacer n acciones secuencialmente de una lista de acciones controlada y, después, se crean distintas cantidades de estos para hacer los tests, simulando así un entorno con condiciones normales. Durante las pruebas de esta versión, se pudo ver que, a diferencia de la primera (que quedaba bloqueada al probar con menos de 200 threads) el número no daba problemas hasta los 2000. Este número se alcanzó tras modificar los tiempos de espera entre acciones y de entrada del thread. Además, al controlar las acciones semialeatorias que hacían los clientes, el problema de varios intentos de borrar una misma entrada de la BBDD se solucionó.