**PRUEBA RENDIMIENTO HU-04**

**Pasos que sigue:**

Caso positivo: va a home, se loguea, va al buscador de libros donde hace la llamada a la API, obtenemos lista de libros buscamos un libro no verificado y lo verificamos.

Caso negativo: sigue los mismos pasos, pero cuando obtiene la lista de libros se mete en uno que ya esté verificado y no hace nada más porque el botón para verificar en ese caso sería falso.

En esta prueba se observa un rendimiento mucho peor que en el resto, que como se podrá comprobar tiene que ver con entrar en la pestaña de búsqueda de libros de nuestra aplicación.

En concreto, creemos que tiene que ver con el uso que se hace de una API externa, que al parecer no permite se sobrepase un cierto número de peticiones en un intervalo de tiempo.

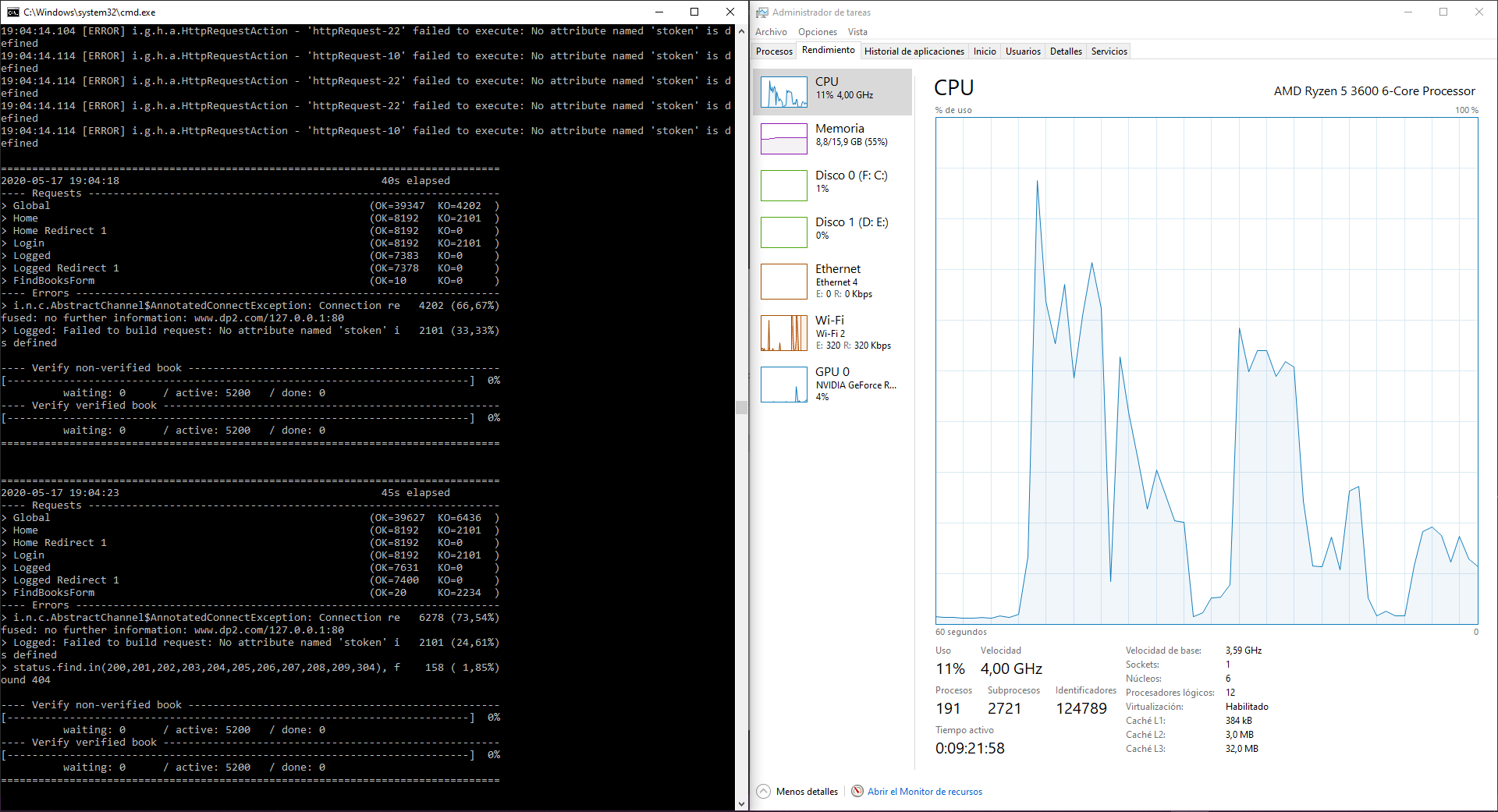
**Stress Test**: aquí obtenemos que con 5200 usuarios concurrentes en cada uno de los 2 escenarios repartidos en 10 segundos se produce un ligero cuello de botella en la CPU, pero sobre todo podemos comprobar el gran uso que se hace de la interfaz de red como consecuencia de las llamadas a la API externa.

Ilustración 1: stress test HU-04

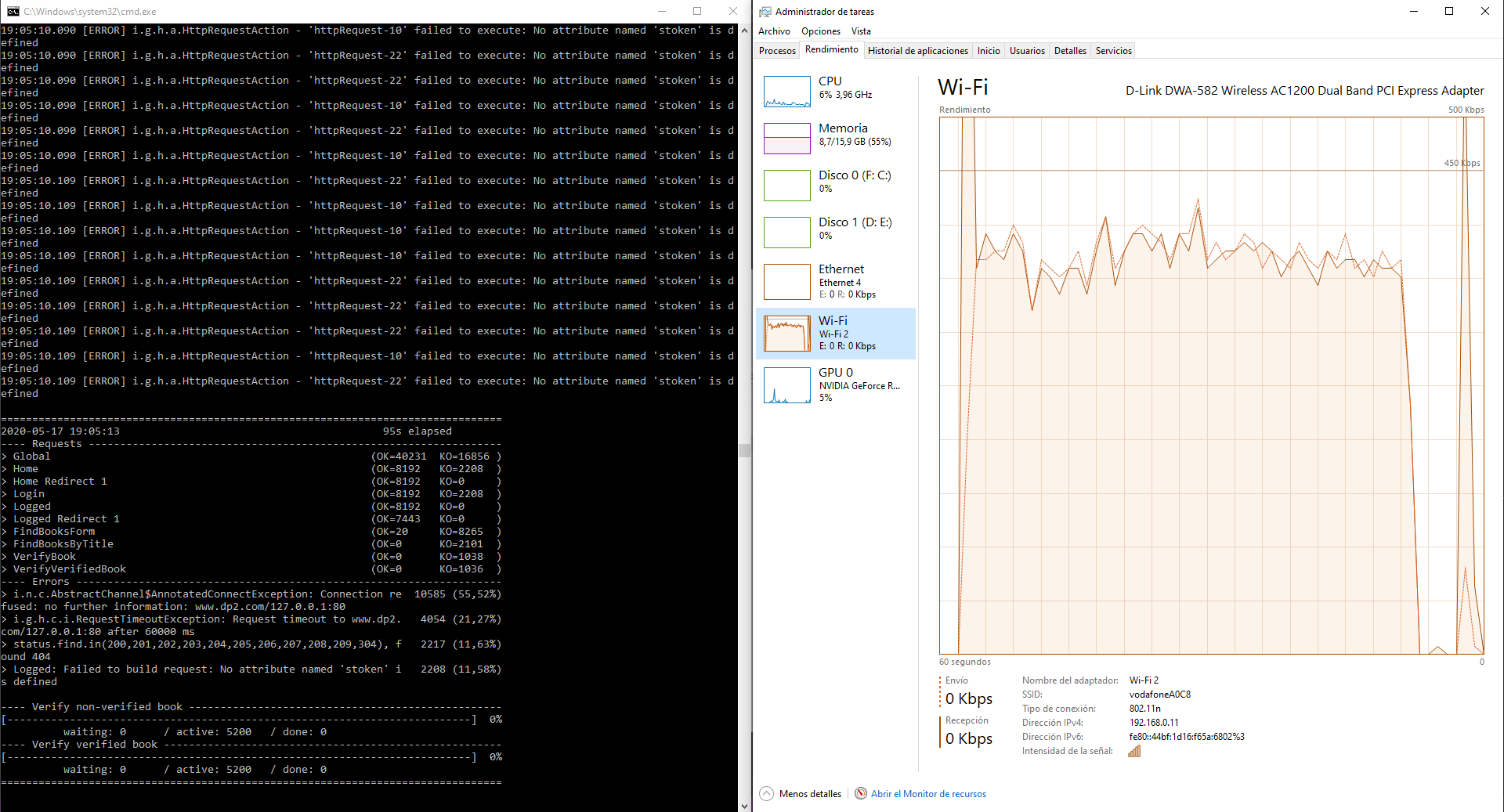


Ilustración 2: stress test HU-04

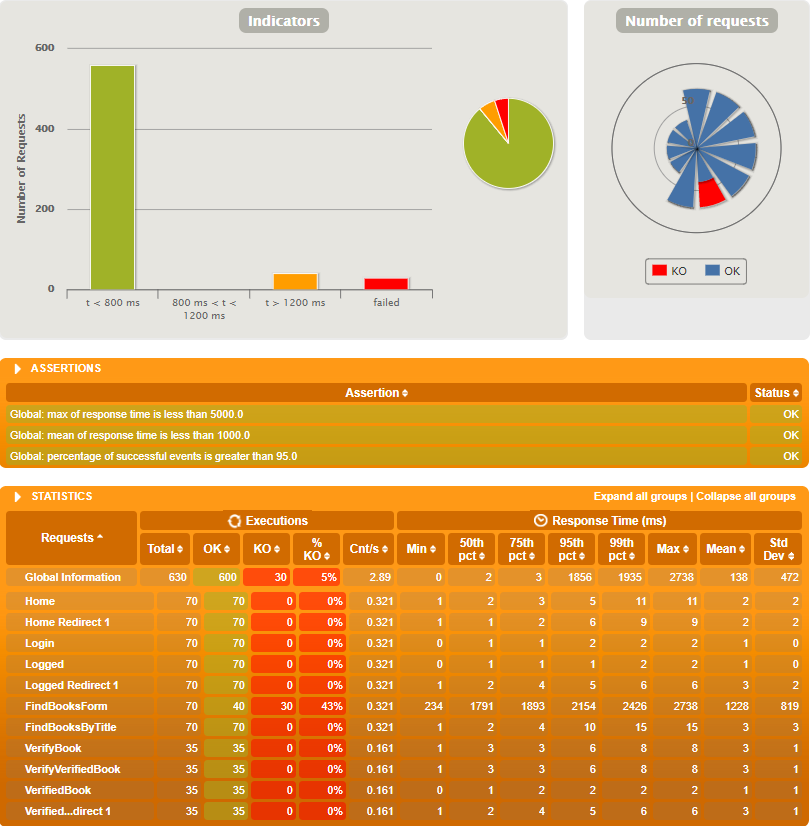
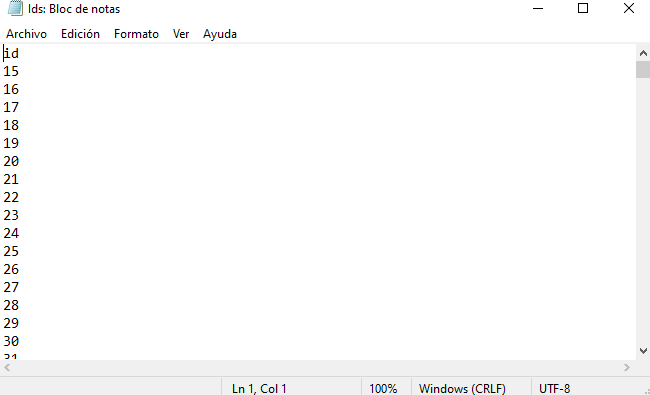
**Load Test**: debido a lo comentado sobre el rendimiento de la API, nuestra aplicación solo puede proporcionar un rendimiento óptimo para 35 usuarios durante 100 segundos, que como podemos comprobar en otros escenarios, es un rendimiento bastante pobre para lo que podría ser. 

Ilustración 3: load test HU-04

**PRUEBA RENDIMIENTO HU-14**

**Csv:** Se usa el mismo csv que para HU-01 pero adicionalmente 3 csv con Ids para ir añadiendo el libro que se va creando a leídos con la siguiente forma.



**Pasos que sigue:**

Caso positivo: en este caso he tenido que utilizar el mismo código que en la HU-01, pero añadiendo la parte de añadir libro a deseados después de crear el libro, ya que si no siempre añadiría el mismo libro y daría error.

Caso negativo: va a home, se loguea, va al buscador de libros donde hace la llamada a la API, obtenemos lista de libros buscamos un libro que esté leído o añadido a nuestra lista de deseados y no hace nada más porque el botón para verificar en ese caso sería falso.

En esta prueba también se aprecie un rendimiento bastante pobre, en parte por el uso de la API externa y en parte por la forma de probar la adición de un libro a la lista de libros más deseados.

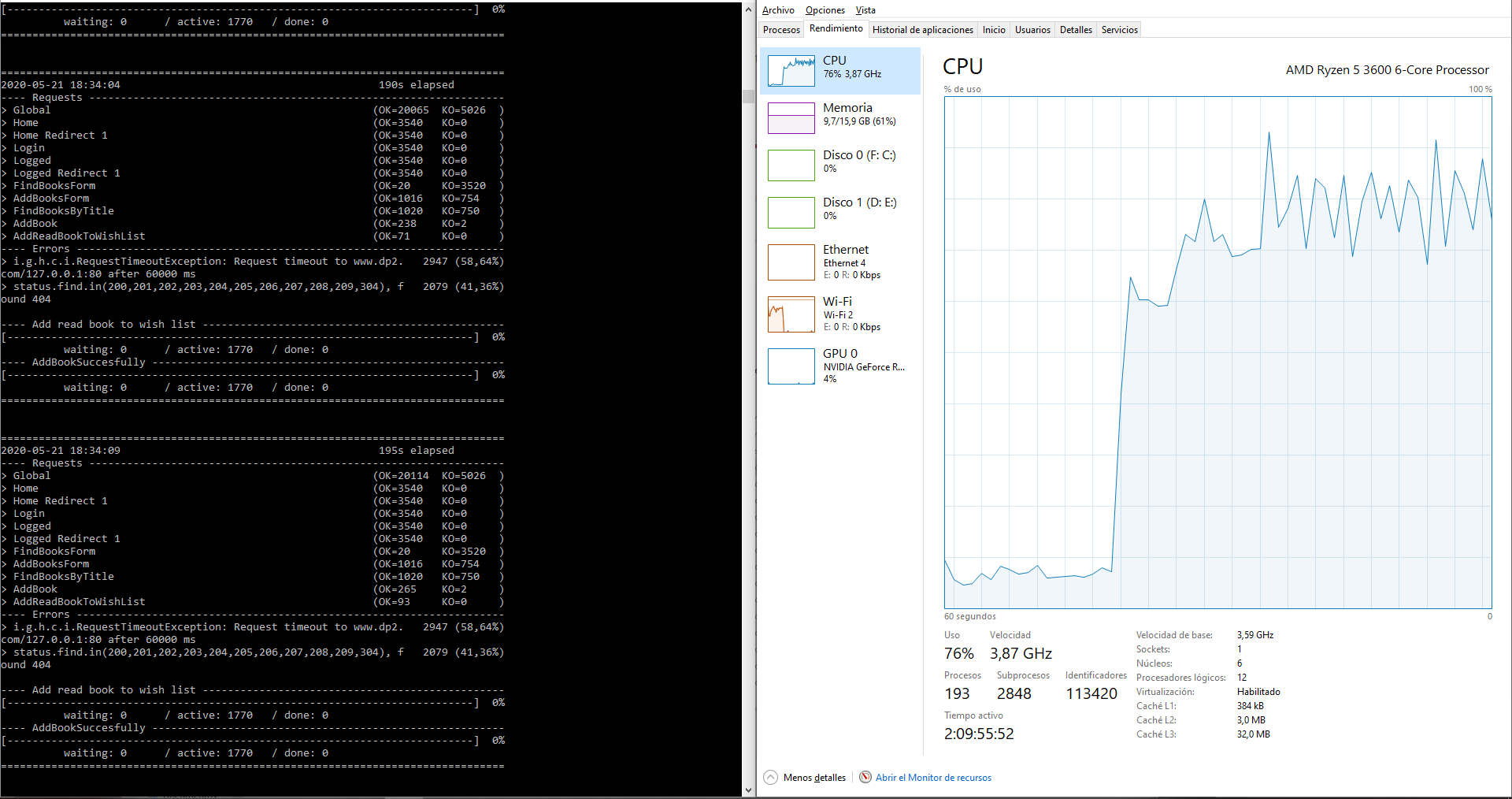
**Stress Test**: aquí obtenemos que con 1770 usuarios concurrentes en cada uno de los 2 escenarios repartidos en 10 segundos se produce un ligero cuello de botella en la CPU, manteniéndose en un alto nivel de uso durante un largo período de tiempo cuando se empieza a añadir libros. Se aprecia también un alto uso de la interfaz de red.

Ilustración : stress test HU-14

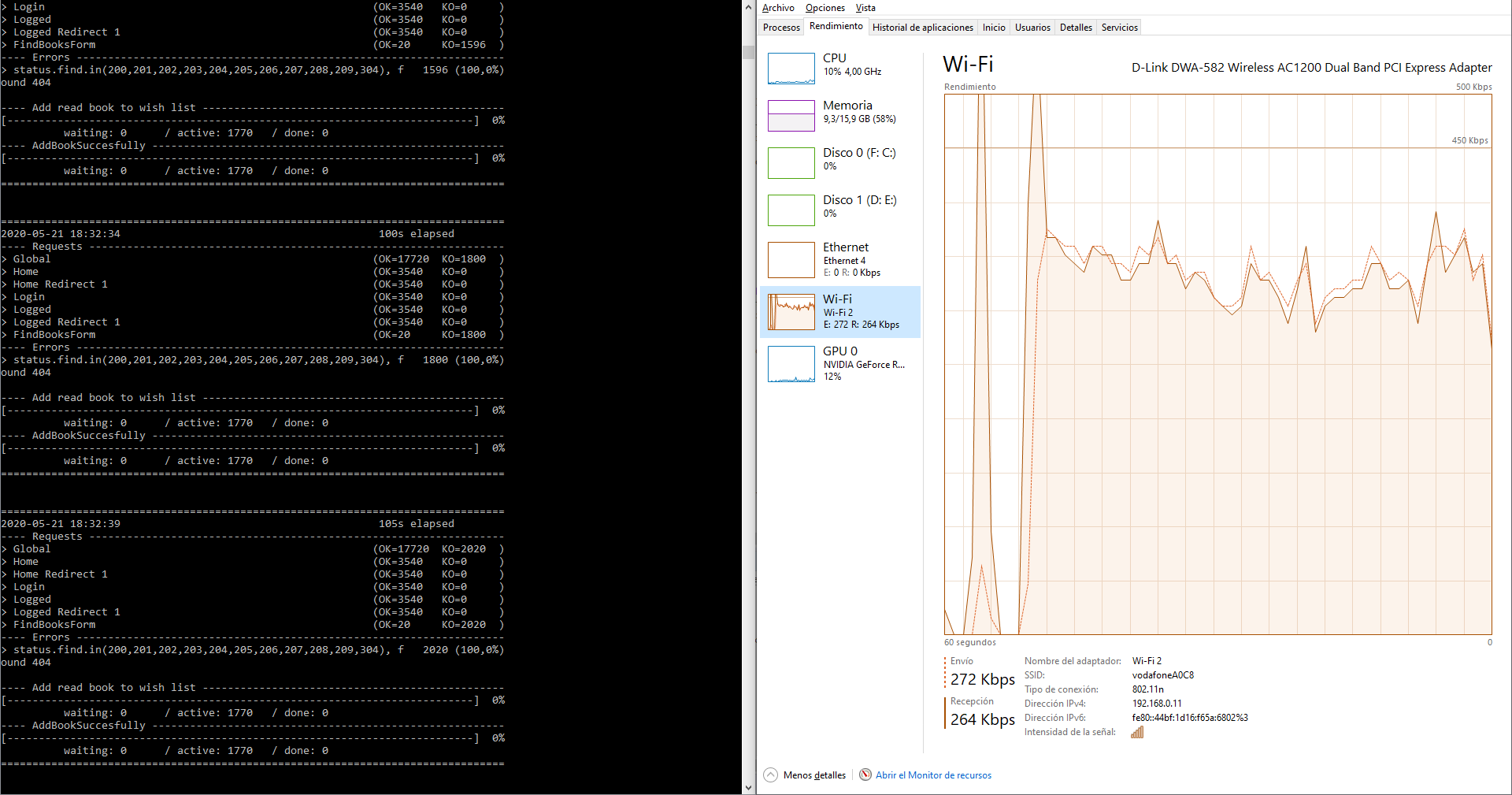


Ilustración : stress test HU-14

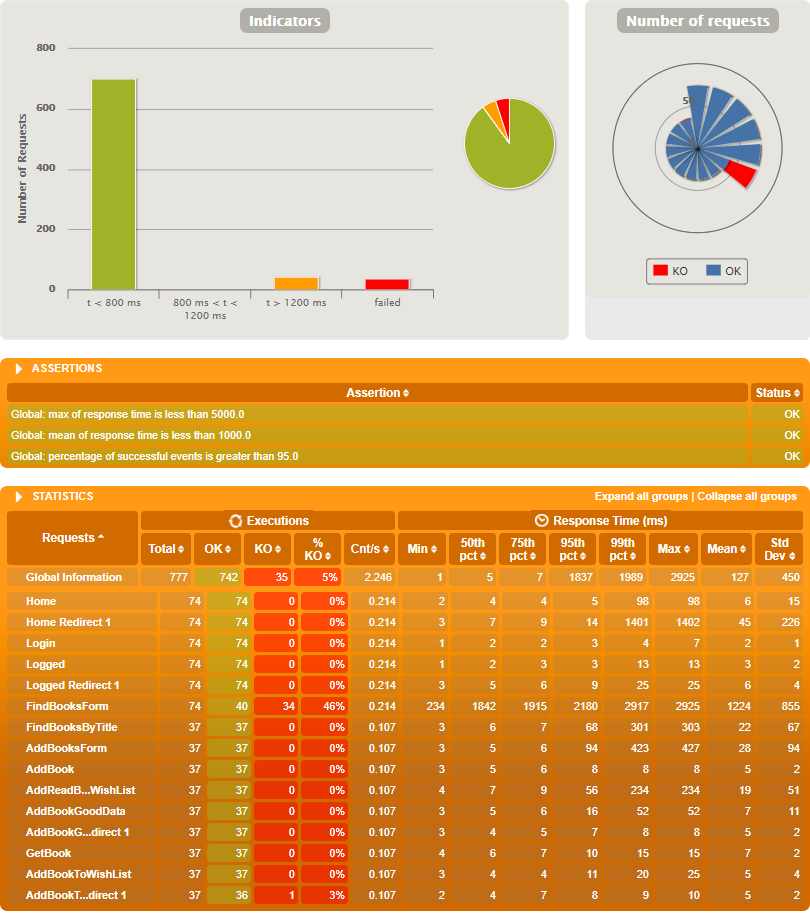
**Load Test**: debido a lo comentado sobre el rendimiento de la API, nuestra aplicación solo puede proporcionar un rendimiento óptimo para 35 usuarios durante 100 segundos, que como podemos comprobar en otros escenarios, es un rendimiento bastante pobre para lo que podría ser. 

Ilustración : load test HU-14

**PRUEBA RENDIMIENTO HU-17**

**Pasos que sigue:**

Caso positivo: va a home, se loguea y va al apartado de top libros más leidos.

Caso negativo: hace lo mismo que en el caso positivo, con una diferencia explicada a continuación.

En esta prueba debido a la naturaleza de los escenarios positivo y negativo de la historia de usuario, hemos utilizado el mismo script pero variando el número de usuarios dependiendo del caso, primero con la base de datos totalmente poblada y después con la base de datos pero sin los datos de libros más leídos.

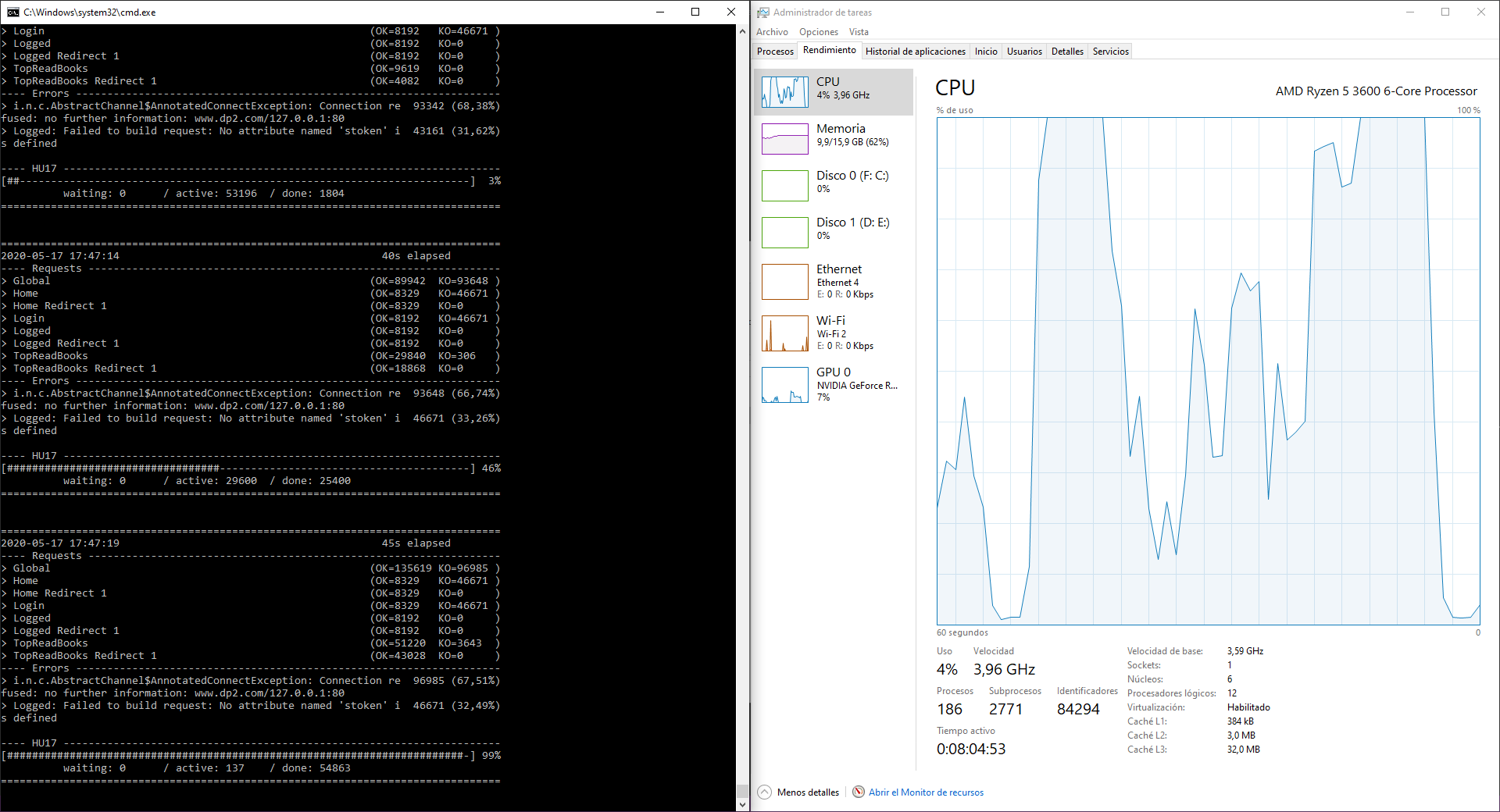
**Stress Test**: aquí obtenemos que con 55000 usuarios concurrentes en 10 segundos y con la base de datos totalmente poblada se producía un cuello de botella importante en la CPU, mientras que en el caso de la base de datos sin los datos de libros más leídos esta cifra de usuarios concurrentes llega hasta los 85000 en 10 segundos.

Ilustración 7: stress test HU-17 con BD totalmente poblada

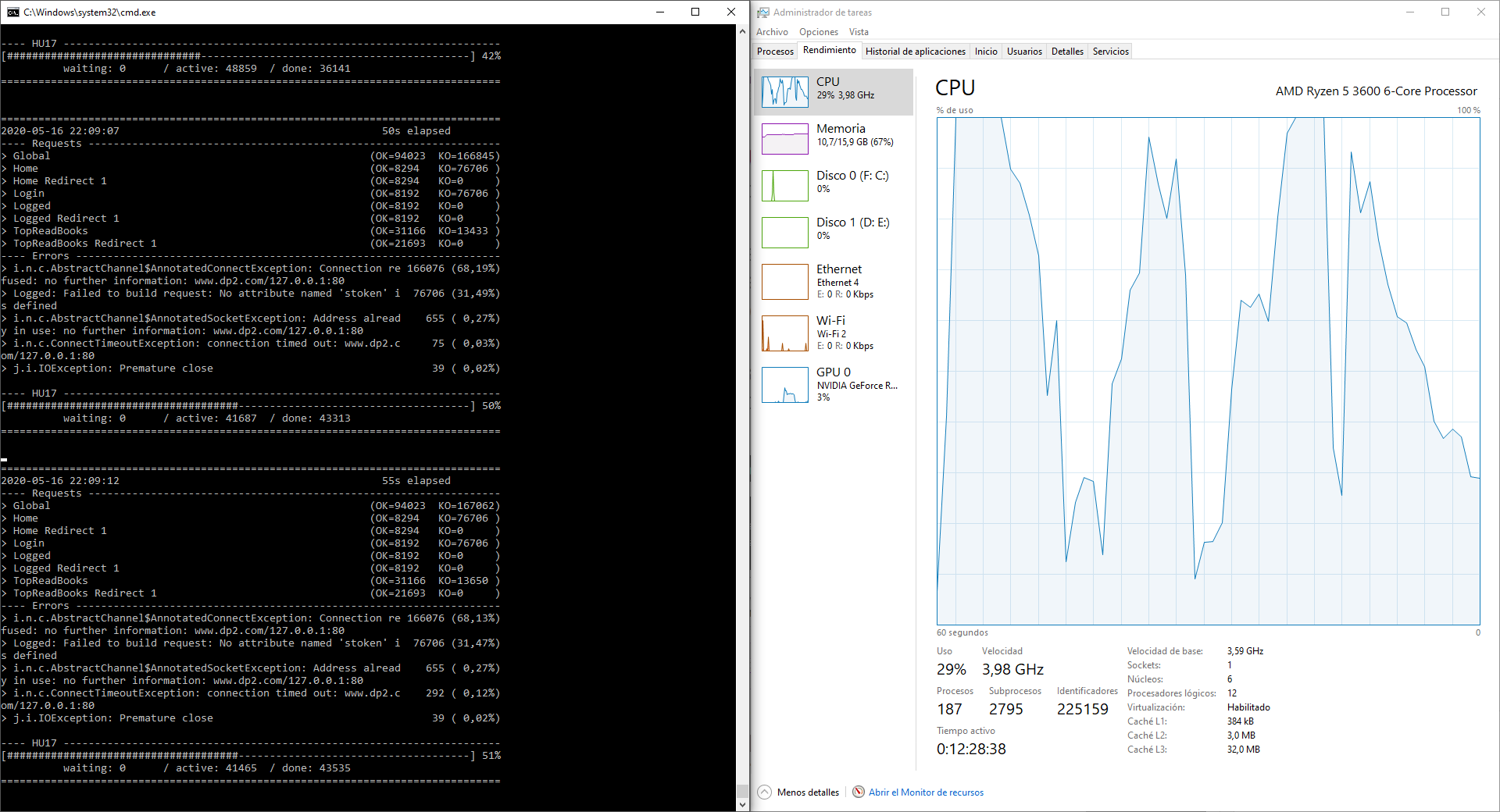


Ilustración 8: stress test HU-17 sin datos de libros más leídos

**Load Test**: como se puede observar, nuestra aplicación puede proporcionar un rendimiento óptimo para 35000 usuarios en 100 segundos en el caso de que no haya datos de libros más leídos y para 32500 usuarios en 100 segundos en caso contrario.

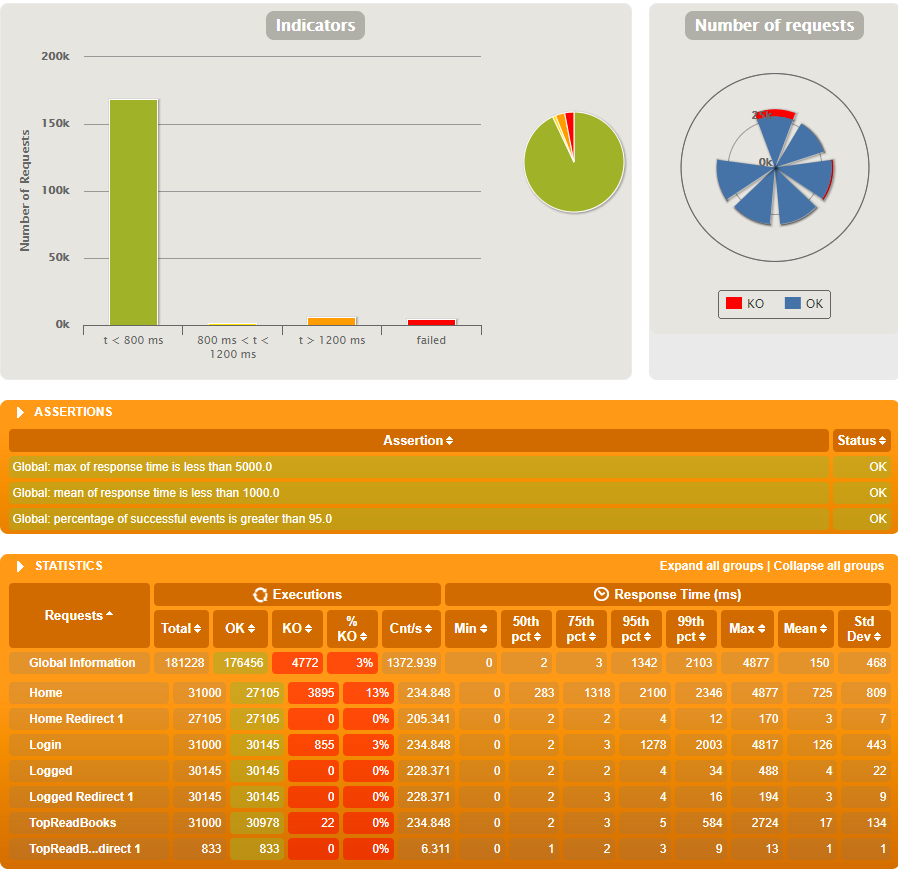


Ilustración 9: load test HU-17 sin datos de libros más leídos

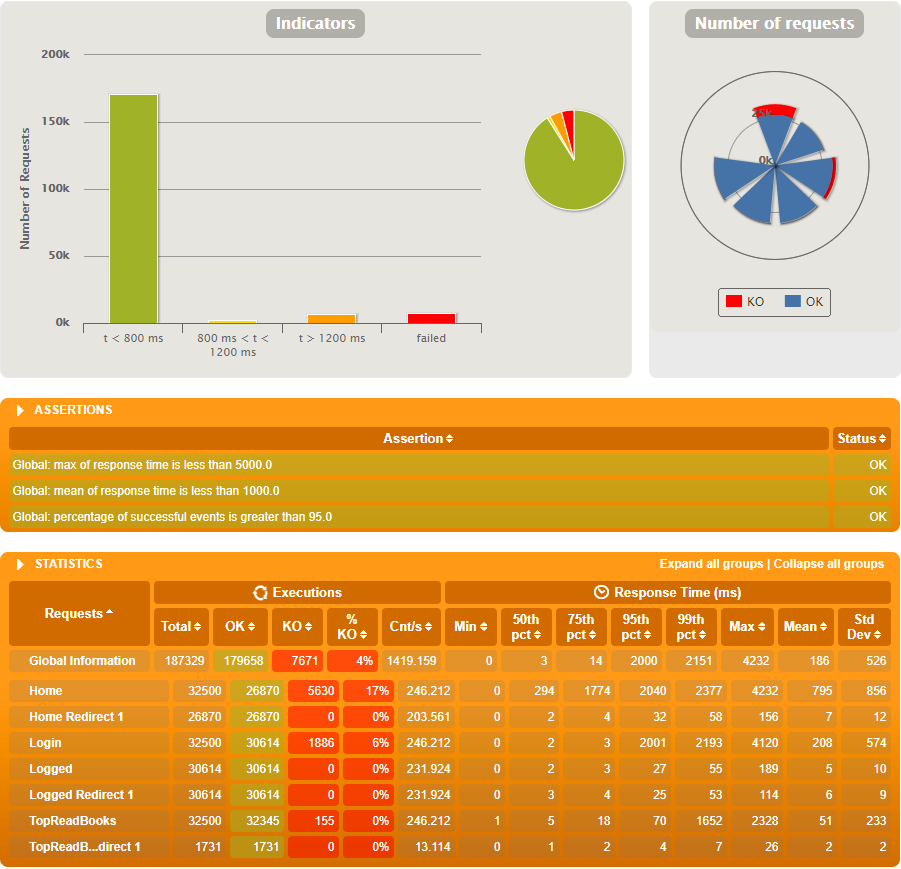


Ilustración 10: load test HU-17 con BD totalmente poblada

**PRUEBA RENDIMIENTO HU-20**

**Pasos que sigue:**

Caso positivo: va a home, se loguea, va a la lista de usuarios del sistema y verifica un usuario del sistema.

Caso negativo: sigue los mismos pasos, pero cuando obtiene la lista de no hace más ya que el botón de verificar a un usuario verificado sería falso.

**Stress Test**: aquí obtenemos que con 15000 usuarios concurrentes repartidos en 10 segundos se produce un cuello de botella importante en la CPU, aunque con el paso del tiempo se va recuperando.

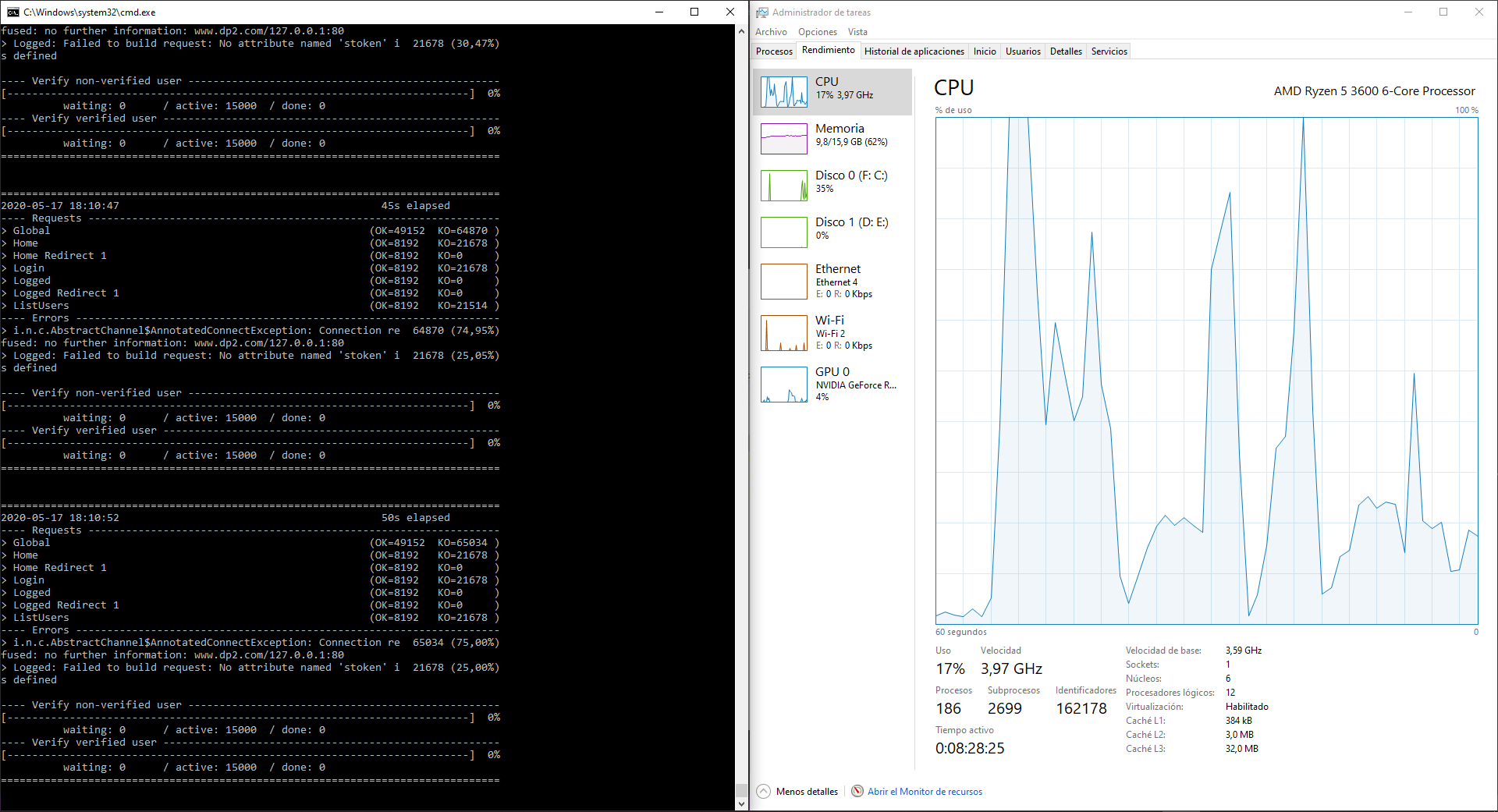


Ilustración 11: stress test HU-20

**Load Test**: en este caso, comprobamos que la aplicación puede ofrecer un rendimiento óptimo a 5500 usuarios durante 100 segundos.

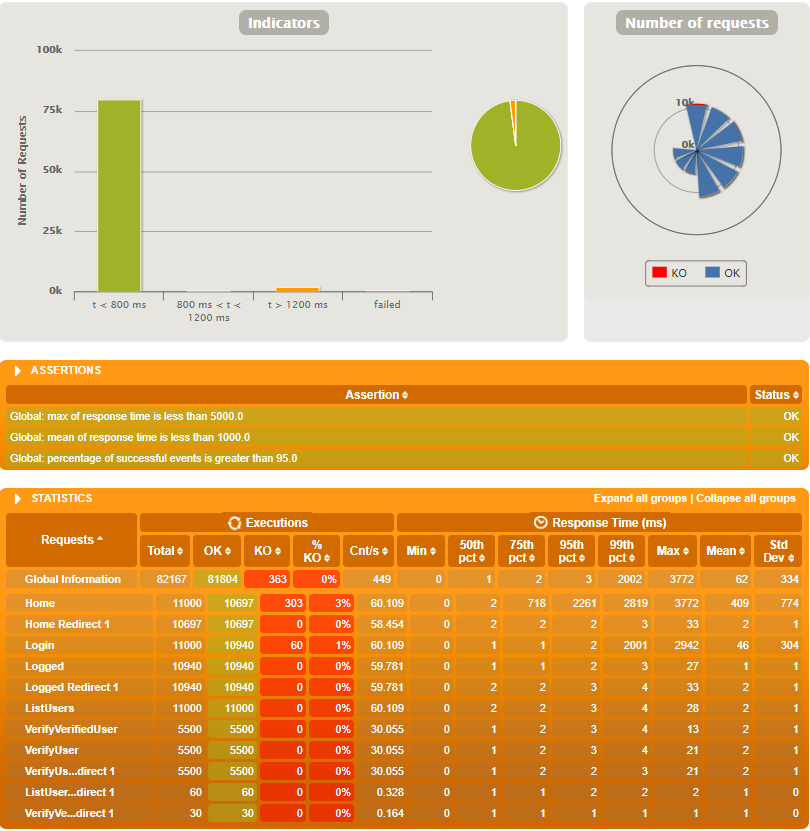


Ilustración 12: load test HU-20