El objetivo es realizar una refactorización para evitar el cuello de botella que provoca la API <https://poemist.github.io/poemist-apidoc>.

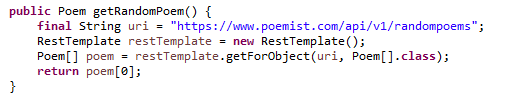
Para ello, de los procedimientos explicados en la asignatura, nos hemos decantado por hacer uso de una caché. Sin embargo, no es un uso convencional de la misma ya que queremos obtener poemas aleatorios, lo cuál va en contra del concepto de caché.

Para ello, vamos a hacer lo siguiente: en lugar de pedir un poema a la API, vamos a pedir varios (en torno a 10) y guardarlo en caché con un *time out* de 5 minutos. Seremos nosotros los que nos encargaremos de aleatorizar qué poema representar de entre los devueltos en la lista.

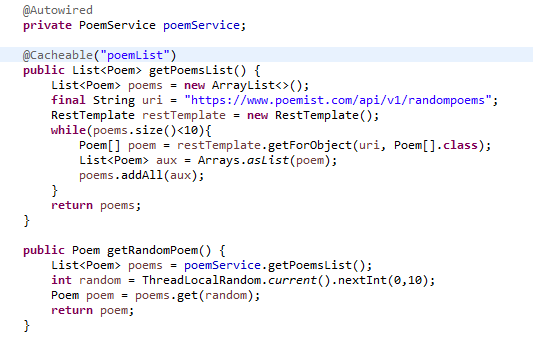
Es cierto que el rango de variación de poemas será menor, pero ganaremos mucha eficiencia ya que la API saturaba la aplicación rápidamente.

## Implementación

Hasta ahora se hacía una llamada a la API, que devuelve por defecto 5 poemas siempre, y nos quedábamos únicamente con uno, motivo de más para hacer uso de una caché:

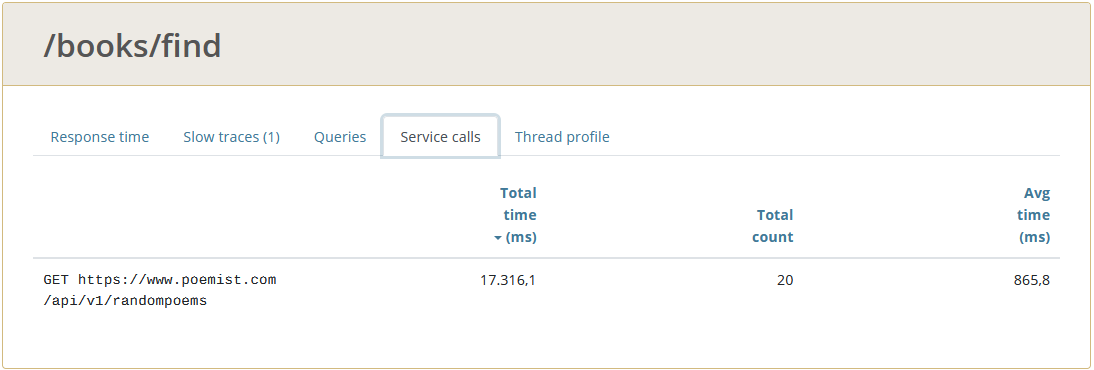


Ahora, hacemos el método que devuelve la lista cacheable y obtenemos uno aleatorio. Nótese que es necesario crear un atributo *@Autowired* del propio servicio porque de otra forma las llamadas internas no son tenidas en cuenta por la caché:

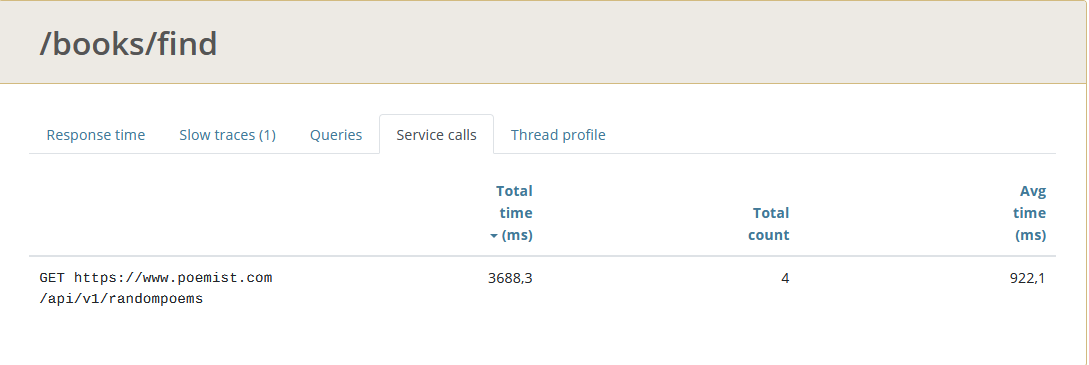


## Resultados

Antes de realizar la refactorización, si entrábamos 10 veces a la página donde se muestra el poema aleatorio, teníamos los siguientes resultados en Glowroot:



Sin embargo, tras implementar la caché, con la primera petición se guardan los resultados (podemos verlo en la consola gracias al logger) y vemos en Glowroot:

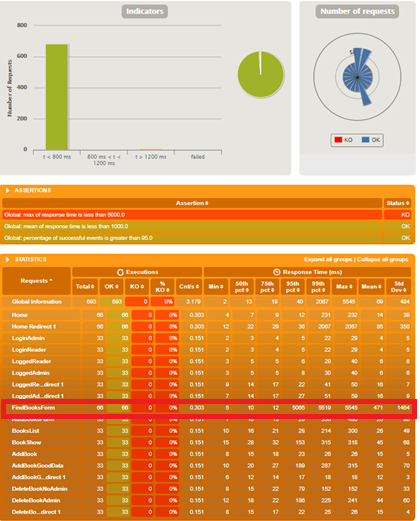
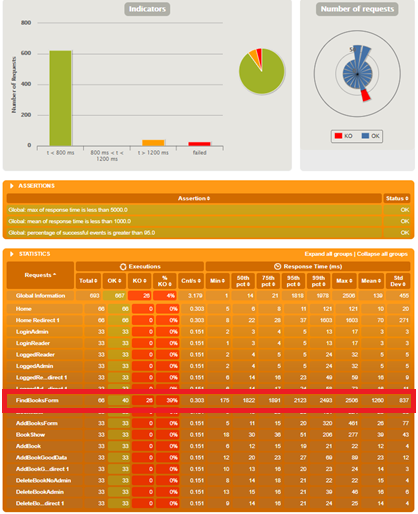


Es cierto que la primera petición es bastante lenta y tadará más de 3 segundos en se respondida, pero a partir de ahí y durante 5 minutos obtendremos un beneficio que compensa esta desventaja ya que no se harán llamadas a la API.

Finalmente, vamos a probar a ejecutar los test de rendimientos de gatling de la HU-03 que llama a la API de nuevo (resultados comparados en la siguiente página).

Podemos apreciar que e¡la primera llamada es muy lenta, tomando 5 segundos y medio, sim embargo el tiempo medio de respesta se reduce en más de la mitad.

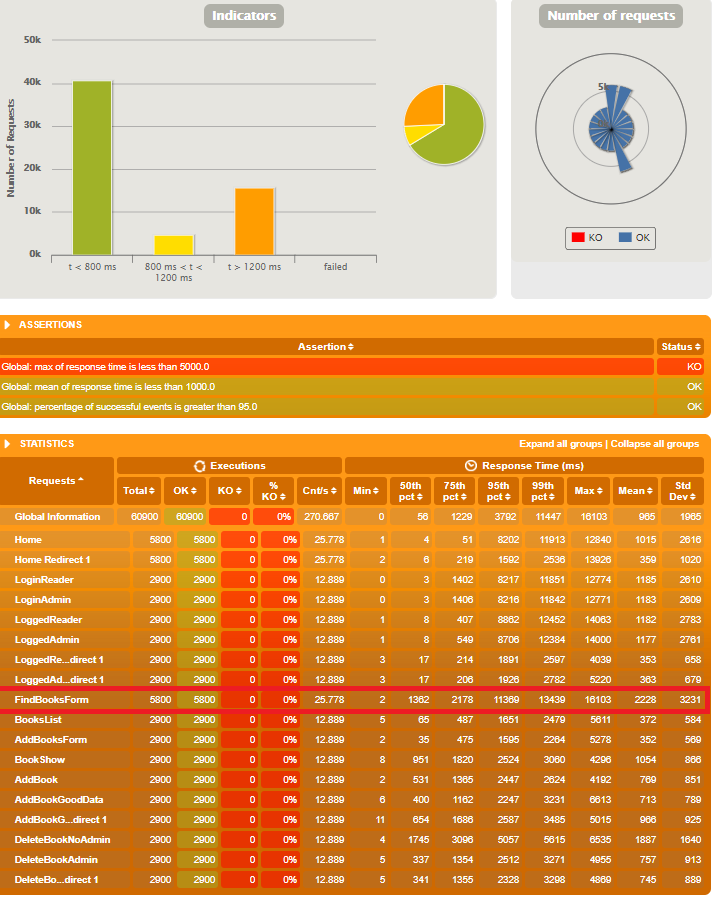
Además, la API a partire de las 20 llamadas dejaba de responder por un período de tiempo y así lo evitamos.



Después de la caché

Antes de la caché

Pero eso era con solo 33 usuarios repartidos en 100 segundos. Con la modificación pasamos a soportar 2900 usuarios en 100 segundos (obviando el KO en el tiempo máximo por la primera petición realizada a la API, que es la que se cachea):



Nuevo test de carga