**Interview Questions**

# 1. Why Would You Use POST Instead of GET for a Read Operation?

# Security reasons

When a get GET request is received, many servers log information about the incoming request. Most of them will log the whole requested URL including query parameters, which might include sensitive information. In our case, we would be potentially logging the phone number of our users.

# URL length

Browsers and HTTP servers can have a maximum URL length. For example Microsoft Internet Explorer is limited to 2,048 characters, and Apache HTTP Server can handle up to 4,000 characters in a URL. In our case, given that a telephone number might have a maximum length of 9 characters, there would be no reason to use POST instead of GET.

# Accountability

There is a very interesting point in the W3C’s paper [URIs, Addressability, and the use of HTTP GET and POST](https://www.w3.org/2001/tag/doc/whenToUseGet.html):

*Use POST if […] (the) user****be held accountable for the results of the interaction****.*

Requesting a user’s profile exposes an individual’s personally identifiable information. One could argue that requesting this data should not be done lightly.

## **Custom Thread Pool to manage parallel streams**

**We can actually pass a custom *ThreadPool*when processing the *stream*.**

The following example lets have a parallel *Stream* use a custom *ThreadPool* to calculate the sum of long values from 1 to 1,000,000, inclusive:

@Test

**public** **void** **giveRangeOfLongs\_whenSummedInParallel\_shouldBeEqualToExpectedTotal**()

**throws** InterruptedException, ExecutionException {

**long** firstNum = 1;

**long** lastNum = 1\_000\_000;

List<Long> aList = LongStream.rangeClosed(firstNum, lastNum).boxed()

.collect(Collectors.toList());

**ForkJoinPool** customThreadPool = **new** **ForkJoinPool**(4);

**long** actualTotal = customThreadPool.submit(

() -> aList.parallelStream().reduce(0L, Long::sum)).get();

assertEquals((lastNum + firstNum) \* lastNum / 2, actualTotal);

}Copy

We used the *ForkJoinPool*constructor with a parallelism level of 4. Some experimentation is required to determine the optimal value for different environments, but a good rule of thumb is simply choosing the number based on how many cores your CPU has.

Next, we processed the content of the parallel *Stream*, summing them up in the *reduce*call.

This simple example may not demonstrate the full usefulness of using a custom thread pool, but the benefits become obvious in situations where we do not want to tie-up the common thread pool with long-running tasks – such as processing data from a network source – or the common thread pool is being used by other components within the application.

ResponseEntity

The ResponseEntity object is Spring’s wrapper around the request response. It inherits from the HttpEntity object and contains the **Http** response code (httpstatus), the response header (header), and the response body (body). A Spring MVC interface to get user information usually we return the entity directly (with @RestController).

# How Does the Spring Singleton Bean Serve Concurrent Requests?

**When the Spring container creates a bean with the singleton scope, the bean is stored in the heap.** This way, all the concurrent threads are able to point to the same bean instance.

It's possible for Spring to use the same bean instance in multiple threads, firstly because for each thread, Java creates a private [stack memory](https://www.baeldung.com/java-stack-heap#stack-memory-in-java).

**The stack memory is responsible for storing the states of the local variables used inside methods during thread execution.** This way, Java makes sure that threads executing in parallel do not overwrite each other's variables.

Secondly, because the bean sets no restrictions or locks at the heap level, **the**[**program counter**](https://www.baeldung.com/cs/process-control-block#2-program-counter)**of each thread is able to point to the same reference of the bean instance in the heap memory.** Therefore, both threads can execute the  method of the bean simultaneously.