**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.

## **The benefits of using Kafka vs. AMQP or JMS**

Kafka was designed to [deliver these distinct advantages](https://www.cloudera.com/products/open-source/apache-hadoop/apache-kafka.html) over AMQP, JMS, etc.

1. **Kafka is highly scalable.** Kafka is a distributed system, which is able to be scaled quickly and easily without incurring any downtime. Apache Kafka is able to handle many terabytes of data without incurring much at all in the way of overhead.
2. **Kafka is highly durable.** Kafka persists the messages on the disks, which provides intra-cluster replication. This makes for a highly durable messaging system.
3. **Kafka is Highly Reliable.** Kafka replicates data and is able to support multiple subscribers. Additionally, it automatically balances consumers in the event of failure. That means that it’s more reliable than similar messaging services available.
4. **Kafka Offers High Performance.**Kafka delivers high throughput for both publishing and subscribing, utilizing disk structures that are capable of offering constant levels of performance, even when dealing with many terabytes of stored messages.

**JAX-WS**

[Java API for XML Web Services (JAX-WS)](http://jax-ws.java.net/) is a standardized API for creating and consuming SOAP (Simple Object Access Protocol) web services.

**JAX-RS**

JAX-RS is nothing more than a specification, a set of interfaces and annotations offered by Java EE. And then, of course, we have the implementations; some of the more well known are [RESTEasy](http://resteasy.jboss.org/) and [Jersey](https://jersey.java.net/).

Also, if you ever decide to build a JEE-compliant application server, the guys from Oracle will tell you that, among many other things, your server should provide a JAX-RS implementation for the deployed apps to use. That's why it's called Java Enterprise Edition **Platform**.

**Marshalling and Unmarshalling**

Marshalling is the process of transforming Java objects into XML documents. Unmarshalling is the process of reading XML documents into Java objects. The JAXBContext class provides the client's entry point to the JAXB API. It provides API for marshalling, unmarshalling and validating.

## **JAXB**

Java Architecture for XML Binding (JAXB) is a software framework that allows Java developers to map Java classes to XML representations. JAXB enables to marshal Java objects into XML and unmarshal XML back into Java objects.

In Java 9, JAXB has moved into a separate module java.xml. In Java 9 and Java 10 we need to use the --add-modules=java.xml.bind option. In Java 11, JAXB has been removed from JDK and we need to add it to the project as a separate library via Maven or Gradle.

**Why Spring Batch used over Spring MVC?**

Spring Batch is a lightweight, comprehensive batch framework designed to enable the development of robust batch applications that are vital for the daily operations of enterprise systems.

Many applications within the enterprise domain require bulk processing to perform business operations in mission-critical environments. These business operations include:

* Automated, complex processing of large volumes of information that is most efficiently processed without user interaction. These operations typically include time-based events (such as month-end calculations, notices, or correspondence).
* Periodic application of complex business rules processed repetitively across very large data sets (for example, insurance benefit determination or rate adjustments).
* Integration of information that is received from internal and external systems that typically requires formatting, validation, and processing in a transactional manner into the system of record. Batch processing is used to process billions of transactions every day for enterprises.

Spring Batch provides reusable functions that are essential in processing large volumes of records, including logging and tracing, transaction management, job processing statistics, job restart, skip, and resource management.

Steps:

1. Add @EnableBatchProcessing
2. Autowire JobBuilderFactory, stepBuilderFactory and repository
3. Create a reader of ItemReader, processor and ItemWriter
4. Create step
5. Create Job