**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

## ArgumentCaptor

ArgumentCaptor allows us to capture an argument passed to a method to inspect it. This **is especially useful when we can't access the argument outside of the method we'd like to test**

@Test **void** **whenDoesSupportHtml\_expectHTMLEmailFormat**() {

**String** to = "info@baeldung.com";

**String** subject = "Using ArgumentCaptor";

**String** body = "Hey, let'use ArgumentCaptor";

emailService.send(to, subject, body, true); verify(platform).deliver(emailCaptor.capture());

**Email** value = emailCaptor.getValue(); assertThat(value.getFormat()).isEqualTo(Format.HTML);

}

**Entity Manager**

The EntityManager interface is an API that manages the lifecycle of an entity instance.

Entities cannot persist themselves on the relational database; annotations are used only to declare a POJO as an entity or to define its mapping and relationships with the corresponding tables on the relational database.

In JPA, the EntityManager interface is used to allow applications to manage and search for entities in the relational database.

The EntityManager is an API that manages the lifecycle of entity instances. An EntityManager object manages a set of entities that are defined by a persistence unit. Each EntityManager instance is associated with a *persistence context*. A persistence context defines the scope under which particular entity instances are created, persisted, and removed through the APIs made available by EntityManager. In some ways, a persistence context is conceptually similar to a transaction context.

**AWS**

How to create public subnet vs private subnet ?  
-> Create two different route tables for private and public  
    and in public subnet route table - to point all external bound traffic ( i.e.; 0.0.0.0/0) to Internet Gateway  
    and in private subnet route table – don’t add this rule for external traffic ( i.e.; 0.0.0.0/0) to Internet Gateway  
  
  
How will a private EC2 connect with other services in different VPC or Google cloud via internet ?  
- add rule in private subnet route table - to point all external bound traffic ( i.e.; 0.0.0.0/0) to NAT Gateway  
  
  
SSH cmd ?  
ssh -i file.pem ec2-user@<ip\_address\_of\_EC2>

I am trying to SSH into EC2 but getting timeout ? What could be the reason ?  
1) EC2 secruity group might not SSH port open

| **NACL** | **Security Group** |
| --- | --- |
| Network Access Control List that helps provide a layer of security to the amazon web services. There are two kinds of NACL- Customized and default. | A security group has to be explicitly assigned to an instance; it doesn’t associate itself to a subnet. |
| Multiple subnets can be bound with a single NACL, but one subnet can be bound with a single NACL only, at a time | Security groups are associated with an instance of a service. It can be associated with one or more security groups which has been created by the user. |
| NACL can be understood as the firewall or protection for the subnet. | Security group can be understood as a firewall to protect EC2 instances. |
| These are stateless, meaning any change applied to an incoming rule isn’t automatically applied to an outgoing rule. | These are stateful, which means any changes which are applied to an incoming rule is automatically applied to a rule which is outgoing. |
| Example: If a request comes through port 80, it should be explicitly indicated that its outgoing response would be the same port 80. | Example: If the incoming port of a request is 80, the outgoing response of that request is also 80 (it is opened automatically) by default. |
| NACL can be used to support as well as deny rules. Denial of rules can be explicitly mentioned, so that when the layer sees a specific IP address, it blocks connecting to it. | They support rules only, and the default behaviour is denial of all rules.    Every VPC can belong to different security groups. |
| It is considered to be the second layer of defence, which helps protect AWS stack. It is an optional layer for VPC, which adds another security layer to the amazon service. | It is considered to be the first defence layer that helps protect the Amazon Web Services infrastructure. |
| In case of NACL, the rules are applied in the order of their priority, wherein priority is indicated by the number the rule is assigned. | In case of a security group, all the rules are applied to an instance. |
| This means every rule is evaluated based on the priority it has. | This means all rules are evaluated before they allow a traffic. |

**Bastion host**

A bastion host is a server whose purpose is to provide access to a private network from an external network, such as the Internet. Because of its exposure to potential attack, a bastion host must minimize the chances of penetration. For example, you can use a bastion host to mitigate the risk of allowing SSH connections from an external network to the Linux instances launched in a private subnet of your Amazon Virtual Private Cloud (VPC).

Amazon VPC enables you to launch AWS resources on a virtual private network that you have defined. The bastion host runs on an Amazon EC2 instance that is typically in a public subnet of your Amazon VPC. Linux instances are in a subnet that is not publicly accessible, and they are set up with a security group that allows SSH access from the security group attached to the underlying EC2 instance running the bastion host.

**NAT gateway**

A NAT gateway is a Network Address Translation (NAT) service. You can use a NAT gateway so that instances in a private subnet can connect to services outside your VPC but external services cannot initiate a connection with those instances.

# Internet gateway

An internet gateway is a horizontally scaled, redundant, and highly available VPC component that allows communication between your VPC and the internet. It supports IPv4 and IPv6 traffic. It does not cause availability risks or bandwidth constraints on your network traffic.

An internet gateway enables resources in your public subnets (such as EC2 instances) to connect to the internet if the resource has a public IPv4 address or an IPv6 address. Similarly, resources on the internet can initiate a connection to resources in your subnet using the public IPv4 address or IPv6 address. For example, an internet gateway enables you to connect to an EC2 instance in AWS using your local computer.

**ECS**

Amazon Elastic Container Service (Amazon ECS) is a fully managed container orchestration service that helps you easily deploy, manage, and scale containerized applications. As a fully managed service, Amazon ECS comes with AWS configuration and operational best practices built-in. This also means that you don't need to manage control plane, nodes, or add-ons. It's integrated with both AWS and third-party tools, such as Amazon Elastic Container Registry and Docker. This integration makes it easier for teams to focus on building the applications, not the environment. You can run and scale your container workloads across AWS Regions in the cloud, and on-premises, without the complexity of managing a control plane or nodes.

The following are key features of Amazon ECS:

* A serverless option with AWS Fargate. With AWS Fargate, you don't need to manage servers, handle capacity planning, or isolate container workloads for security. Fargate handles the infrastructure management aspects of your workload for you. You can schedule the placement of your containers across your cluster based on your resource needs, isolation policies, and availability requirements.
* An external instance option with ECS Anywhere. With ECS Anywhere, you can use the Amazon ECS console and AWS CLI to manage your on-premises container workloads.
* An Amazon EC2 option. With EC2, you can use the Amazon ECS console and AWS CLI to manage your EC2 instances.
* Integration with AWS Identity and Access Management (IAM). You can assign granular permissions for each of your containers. This allows for a high level of isolation when building your applications. In other words, you can launch your containers with the security and compliance levels that you've come to expect from AWS.
* AWS managed container orchestration.
* Continuous integration and continuous deployment (CI/CD). This is a common process for microservice architectures that are based on Docker containers. You can create a CI/CD pipeline that takes the following actions:
  + Monitors changes to a source code repository
  + Builds a new Docker image from that source
  + Pushes the image to an image repository such as Amazon ECR or Docker Hub
  + Updates your Amazon ECS services to use the new image in your application
* Support for service discovery. This is a key component of most distributed systems and service-oriented architectures. With service discovery, your microservice components are automatically discovered as they're created and terminated on a given infrastructure.
* Support for sending your container instance log information to CloudWatch Logs. After you send this information to Amazon CloudWatch, you can view the logs from your container instances in one convenient location. This prevents your container logs from taking up disk space on your container instances.

## Launch types

There are two models that you can use to run your containers:

* Fargate launch type - This is a serverless pay-as-you-go option. You can run containers without needing to manage your infrastructure.
* EC2 launch type - Configure and deploy EC2 instances in your cluster to run your containers.