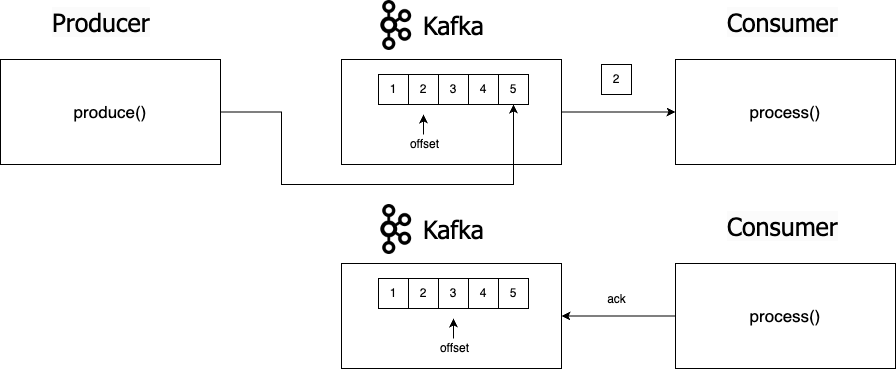
# **Preventing Data Loss With Kafka Listeners in Spring Boot**

Data loss is one of the biggest problems developers face when building distributed systems. Whether due to network issues or code bugs, data loss can have serious consequences for enterprises. In this article, we'll look at how to build Kafka listeners with Spring Boot and how to use Kafka's acknowledgment mechanisms to prevent data loss and ensure the reliability of our systems.

### **Apache Kafka**

is a distributed message platform used to store and deliver messages. Once a message is written to Kafka, it will be kept there according to a retention policy. The consumer groups mechanism is used to read out messages.

The offset for each consumer group is used to understand the stage of message processing and to keep track of the progress of each consumer group in reading messages from a partition. It allows each consumer group to independently read messages from a topic and resume reading from where it left off in case of failures or restarts. In a simplified way, this can be represented as follows:



After successfully processing a message, a consumer sends an acknowledgment to Kafka, and the offset pointer for that consumer group is shifted. As mentioned earlier, other consumer groups store their offset values in the [message broker](https://dzone.com/articles/introduction-to-message-brokers-part-1-apache-kafk), allowing messages to be read independently.

When we talk about high-reliability systems that must guarantee no data loss, we must consider all possible scenarios. Apache Kafka, by design, already has the features to ensure reliability. We, as consumers of messages, must also provide proper reliability. But what can go wrong?

* The consumer receives the message and crashes before he can process it
* The consumer receives the message, processes it, and then crashes
* Any network problems

This can happen for reasons beyond our control — temporary network unavailability, an incident on the instance, pod eviction in a K8s cluster, etc.

Kafka allows guaranteeing message delivery using the acknowledgment mechanism — at least once delivery. It means that the message will be delivered at least once, but under certain circumstances, it can be delivered several times. All we need to do is to configure Apache Kafka correctly and be able to react to duplicate messages if needed. Let's try to implement this in practice.

But what if an error happens during message processing? In that case, we need to handle it correctly. If this error is related to an invalid message, we can write to the log or place this message in a separate topic — DLT (dead letter topic) for further parsing of this message. And what if processing implies calling another microservice, but that microservice doesn't answer? In this case, we may need the retry mechanism.

Here we have specified that in case of an error, we will do retries (maximum three times) at intervals of five seconds. But if we have an NPE, we won't do iterations in that case but just write a message to the log and skip the message.

Here we set enable.auto.commit=false (if true, the consumer's offset will be periodically committed in the background. In that case property auto.commit.interval.ms (default 5000ms will be used) and spring.kafka.listener.ack-mode=MANUAL, which means we want to control this mechanism ourselves.

The Acknowledgment object allows you to explicitly acknowledge or reject (nack) the message. By calling acknowledge(), you are telling Kafka that the message has been successfully processed and can be committed. By calling nack(), you are telling Kafka that the message should be re-queued for processing after a specified delay (i.e., in a case when another microservice isn't responding).

# **10 Ways To Keep Your Java Application Safe and Secure**

With cyber-attacks becoming more sophisticated and prevalent every day, ensuring the safety and security of your Java application is more important than ever.

The last thing any developer wants is to be the victim of a security breach that could compromise their users' sensitive data and jeopardize the trust they have worked so hard to build.

Fear not; we've got you covered! In this blog, we'll explore ten proven ways to keep your Java application safe and secure.

From basic security protocols to advanced encryption techniques, you'll be armed with everything you need to ensure your application stays protected.

So, sit back, relax, and get ready to learn how to take your Java application's security to the next level!

## **1. Secure Your Network Connection**

In our highly connected world, network links have become a critical aspect of our everyday lives. With the expanding use of technology, though, also comes a heightened hazard of cyberattacks. Thus, as a Java programmer, ensuring the security of your [network connection](https://dzone.com/articles/trust-models-for-secure-network-connections) must be of utmost importance to guard your program and its consumers.

One of the most noteworthy things you can do to fortify your network connection is to make use of secure protocols such as HTTPS. By encrypting the network traffic, you can keep malicious attackers from deciphering and manipulating the data sent from the server to the customer.

It is analogous to closing the door of your home before you go to sleep at night. You would not want anyone to trespass without permission, and you do not want anyone to intercept your data without authorization. By guaranteeing your network connection, you are creating a virtual barrier on your application's data, ensuring that only those with proper authorization can access it.

An alternate way to defend your network connection is to limit the number of available ports on your server. It is comparable to having one doorway for your house, which limits the points of entry for possible trespassers. This straightforward step can substantially decrease the danger of an attacker accessing your application unlawfully.

By performing these basic measures, you can safeguard your application and its consumers from probable threats and cyberattacks. Keep your virtual entrance locked, and you can have peace of mind in knowing that your network connection is safe.

## **2. Use a Security Manager**

In regards to protecting your Java software, utilizing a [Security Manager](https://dzone.com/articles/the-java-security-manager-why-and-how) is a necessity. This beneficial utility functions like a doorman, moderating access to various resources in your program.

Envision the Security Manager like an exclusive list of invitees. It allows admission only to those present on the roster and denies passage to all else. Similarly, a Security Manager confines entry to certain assets in your software, such as documents or system parameters.

Employing a Security Manager assists in deterring malicious code from attaining access to the delicate resources of your application. That's really important! Envisage an individual stealing all your consumer data or spoiling the efficacy of your software. Not only would that be devastating for your company, but it could also jeopardize the data of your customers.

So, how do you execute a Security Manager? Well, it's quite straightforward. You just have to create a security policy that delineates which liberties are allowed for each resource. Then, you have to set the security policy as the system default.

It might appear to be bothersome to construct a Security Manager; however, trust us, it is worth it. After all, the security of your program and its customers must always be of utmost priority.

## **3. Validate User Input**

It can be difficult to abstain from allowing users to enter whatever they want into our applications. It's easy to believe in their honesty and presumes they will not input malicious data; however, we know that people cannot always be trusted. This is why validating all user input is so essential.

Incorrect data can result in decreased performance of an application, creating a door for a range of attacks like injection and cross-site scripting. Nevertheless, it's not just about the protection from damaging information; it is also important to make sure the user's input is compliant with the format and rules of the application.

Think of this situation, if you operate an online store, and a user submits a zip code that does not exist, their order will not process. This could leave them disappointed and frustrated. To avoid such issues, adding measures that guarantee all user input is validated is fundamental. It's beneficial to utilize a validation framework such as Apache Commons Validator or Spring Validation to designate what is considered legitimate input for each field. This way, the code is better organized, and validation becomes easier to incorporate.

Validation is not only about checking that the data is in the proper format. It is also about verifying that the data applies to the application. If a user is required to input their date of birth, it must be made certain that they are of legal age to buy your products.

Therefore, to safeguard your application and create a good user experience, validating user input is paramount. This will demonstrate to your users that you value them while also providing an additional security layer.

## **4. Use Encryption**

To safeguard your Java application and avert breaches, [encryption](https://dzone.com/articles/what-is-encryption-and-how-does-it-work) is a critical measure. With encryption, confidential information is disguised in a form that can't be deciphered by people who have nefarious intentions. Suppose you became aware that an intruder had filched your business' private information—the dismay and shock you would feel would be tremendous! For that reason, encrypting your data is of utmost importance.

Encryption thwarts cybercriminals from penetrating and examining the information you're keeping, granting you a greater possibility of evading cyberattacks. Without it, the personal details of your clients, like credit card numbers and login particulars, are like an open book anyone can read. Therefore, you must encode your data to confirm that you care about their security.

If you apply encryption, your customers can trust that you're protecting their details and be at ease. This way, your company can make a name for itself as a dependable and reliable firm.

So, don't leave anything to chance with your Java application's security. Employ encryption to shield your essential information and the privacy of your clients. We can guarantee that both your customers and the bottom line will be appreciative.

## **5. Implement Authentication and Authorization**

When it comes to protecting your Java system, it is absolutely essential that you use [authentication and authorization](https://dzone.com/articles/authentication-cluster). After all, you don't want your sensitive information being accessed by just anyone. So, what does this require? Authentication confirms the identity of the user or app, and authorization decides what those users or apps are able to access.

We understand that this can sound like a lot of work, but it is worth the peace of mind you get knowing that your data is protected. In addition, it's not as difficult as it appears. A great way to start is by implementing multi-factor authentication (MFA). This means that those who try to access the system have to show more than one form of identification. This could range from a password to a biometric scan, which substantially reduces the likelihood of any unauthorized access.

Furthermore, it is important to have [role-based access control](https://dzone.com/articles/what-is-roles-based-access-control-rbac) (RBAC) in place, which limits the view and actions of each user, depending on their job. This prevents employees from seeing data they do not need, despite the fact that they possess the right authentication.

All in all, authentication and authorization are vital in order to make sure your Java program is secure. By utilizing these two strategies, you are protecting sensitive details and allowing only those who have been granted permission to view them. So, make sure to add this to your security plan!

## **6. Keep Your Libraries and Frameworks Up to Date**

We recognize how taxing it can be to keep your applications current with the most recent library and framework updates. It can often feel like an unending chore that is forever being relegated to the bottom of the list of priorities. Nonetheless, updates must not be disregarded as this could lead to disastrous results, including exposing your confidential data to malevolent parties.

Furthermore, updating your libraries and frameworks can provide more than just security - it can enhance your application's operation, ridding it of any previously existing bugs that might compromise user experience.

Even though we recognize the difficulty in staying abreast of updates, we strongly suggest dedicating the effort necessary to guarantee the security and trustworthiness of your applications. We promise you that the payoff for your diligence is worth it.

## **7. Sanitize User Output** It's essential to consider the thorough task of sanitizing user output. But why is it necessary? Consider this: an individual submits a hazardous code, and your application unconsciously distributes it to all other customers, developing quickly. The outcomes could be catastrophic, from security breaches to the detriment of reputation. That's why it's crucial to ensure your output is unsoiled.

So how can you do that? First and foremost, utilize a secure encoding approach, such as HTML escaping or URL encoding, to encode your output. By doing so, any special characters that may cause damage will be rendered inoperative. Second, steer clear of using user input in dynamic SQL queries or command shells. It's like unlocking the gateways to your application's vulnerabilities, making it a desirable target for hackers.

Additionally, apply a Content Security Policy (CSP) to restrict which resources, like JavaScript, pictures, and fonts, your application is able to load. This obstructs criminals from adding malevolent scripts and tampering with your content. Last but not least, scrub all user-generated files prior to displaying or downloading them. Attackers are capable of integrating malware, viruses, or other noxious elements in seemingly safe files.

To summarize, sanitizing user output is a prerequisite, not an alternative, for a secure Java application. Don't permit a basic error to make your application and customers vulnerable to danger. Take the essential precautions and let your users appreciate a safe and problem-free experience.

## **8. Limit Access to Sensitive Information**

We all keep our own individual secrets hidden in our minds, and this also applies to the data your application contains. It is absolutely essential to protect any confidential details your users give to you, such as social security numbers or banking details.

Nowadays, with so many digital dangers lurking in the shadows, restricting access to delicate data should be your primary focus. Carefully monitor who has access to your program and its content, as well as the credentials of users interacting with it.

Limiting access can successfully stop unauthorized individuals from gaining entry into your application and the details within. You must be cognizant of which workers or users are granted authorization to see confidential info and implement a method that can instantly take away permission if needed.

Just think of what could happen if your app was hacked and secret data became public knowledge. The financial repercussions could be ruinous, not to mention the broken trust your users will have in you. By strictly controlling access to sensitive facts, you can avert illegal access, lower the threat of breaches, and defend your reputation.

It's also important to mention that human beings can be responsible for data breaches as well. At times, users can unintentionally give away private details, or employees may not be cautious with their qualifications. As such, you must teach your workers and customers about the safest ways to preserve their confidential data, such as not writing down passwords on Post-it notes or sharing login information with outsiders.

At the end of the day, the protection of your users' sensitive details is an obligation that must not be taken lightly. Take all necessary precautions to ensure the safety of this information, manage access to it, and make certain your application is as secure as possible.

## **9. Utilize Secure Coding Standards**

Making software that respects safety standards is imperative to keep application and user data protected from malicious activities. Following a standard will enable one to secure their product from the very beginning instead of trying to repair flaws at a later time. It is not only about obeying a code but about being mindful of users' safety and privacy. Hence, as software creators, we are held to the obligation of safeguarding our clients from any harm. For that reason, it is fundamental to make secure coding standards a priority in your software development routine.

Such techniques are, for example, validating inputs, securely handling mistakes, and implementing cryptography in a proper way. Although it may necessitate some more effort, making use of these standards could prevent countless complications in the future. By reducing vulnerability and including several layers of security, you will be able to build a product that is potent as well as secure. To conclude, these coding standards are not only beneficial but also a moral duty of developers who understand the weight of their responsibility. By embracing these methods, we can contribute to keeping users protected and making the digital realm a safer place.

## **10. Monitor for Security Threats and Respond Quickly**

In today's competitive programming industry, safeguarding your Java application from probable hazards is vital. Unfortunately, even the most foolproof systems are vulnerable to intruders. This is why it's so important to vigilantly keep track of the security of your software.

It is totally acceptable to feel uneasy about the security of your software and data. But there's no need to worry! You can stay one step ahead of potential issues by constantly monitoring your Java application for any potential flaws.

You can investigate your system's logs and analyze the behavior of its users to proactively detect any peculiar activity. Doing so will provide you with the opportunity to tackle potential security breaches at the onset and mitigate their effects on your system.

However, just detecting security threats is not enough. It is equally essential to promptly and effectively react to these threats. When facing a cyber attack, it is quite common to be overcome with panic or confusion. Yet, having an adequate response plan in place will enable you to take effective action and restrict the damage done.

For instance, you can isolate affected systems, reset login details, or revoke any recent updates that may have triggered a security issue. These measures can help limit the damage done and provide you with enough time to make up for any weak security.

Therefore, the security of your Java application must be continuously maintained. Through proper monitoring of your system for any security threats and immediate response, you can ensure your software is safe, and your peace of mind is kept intact.