Security report

This security report is going to go over the top 10 OWASP risks and analyze how affected this project is.

	Likelihood that app is affected	Impact once occurred	Risk of appearance in new code	Actions possible	Planned to fix
A1: broken access control	Low	High	High	Quality standards assure it does not happen	N/A
A2: Cryptographic failure	Very unlikely	Severe	Low	No passwords saved in clear text. Could hash username too.	No, risk accepted
A3: Injection	Very unlikely	Severe	Low	Escape parameters when using native queries	N/A
A4: Insecure design	Medium	High	Low	Redesign the entire application	N/A
A5: Security misconfiguration	Medium	Severe	Moderate	Spend time studying Spring security	No, risk accepted
A6: Vulnerable and outdated components	Very high	Moderate	High	Update all components and dependencies of the application	No, risk accepted. It would take way too long for such a last-minute change
A7: Identification and Authentication Failures	Medium	Moderate	Moderate	Add MFA, password recovery and weak password check	No, risk accepted
A8: Software and Data Integrity Failures	Low	High	Low	N/A	N/A
A9: Security Logging and Monitoring Failures	High	Medium	Low	Log actions such as failed logins and high-value transactions	N/A, fixed
A10: Server side request forgery	Medium	Moderate	Moderate	Improve aspects such as deny by default, disabling HTTP redirections, whitelist of accepted URLs	No, risk accepted

Reasoning for the OWASP top 10

A1: This is a pretty nasty one, because it happens when code is not written up to quality. So far, the application is safe for this thanks to extensive testing. Impact would be severe, because assuming someone found a broken access gate in the system, they could abuse admin privileged to delete auctions and potentially wipe the whole system.

A2 Application is not affected by A2, because sensitive information like user passwords are kept hashed in the system. Although usernames are saved in clear text so far, that risk is accepted, as someone would have to leak the whole database for someone to brute force the hashed passwords, which is very unlikely. JWT secrets are also hashed and are hard to break, a good idea in production would be to move them away from the application.properties file.

A3 Is very unlikely to appear in this application because it uses an ORM and JDBC, which automatically escape parameters in queries, so that attackers cannot execute queries on the database. Such frameworks are tested and staying up to date ensures safety here.

A4 is a bit of an abstract vulnerability, but basically involves exploits in a way that the system is intended to function, not how it is implemented. Such exploits are basically a pitfall in the core design of the application. In the case of this project it is entirely possible for someone to bid on an auction and back out of it, refusing to buy it. This is fixable with placing a hold on the bidder's credit card, which is unfortunately out of the scope of this project.

A5 Is a combination of small pitfalls that can lead to a big problem, such as error handling revealing stack traces. Even though this is not the case for this project, spring security is a huge part of spring and most of it remains to be explored, so it is entirely possible that some aspects of it are currently configured wrong.

A6: The application is definitely currently using some outdated parts of dependencies, such as the STOMP websockets and a bunch of date-time pickers on the front-end side in react. As for the back-end, minor versions are available for upgrade. Since this is not going to be an application with a lot of traffic, the worthwhile of abusing these outdated dependencies is low.

A7 Basically has to do with improper treating of user credentials. While this application does not feature MFA, password recovery and a password strength checker, these things are way out of scope for this application and would definitely be fixed if it was a bigger project.

A8: Software and data integrity failures relate to code and infrastructure that does not protect against integrity violations. An example of this is where an application relies upon plugins, libraries, or modules from untrusted sources, repositories, and content delivery networks. This application only uses very popular and trusted libraries, to which it does not provide any credentials.

A9: Logging is pretty important for applications in production, because it monitors things that are going on. Logging for some security aspects has been implemented in the application, however it can be improved multi-fold, if it was in scope.

A10: This application is mostly safe from SSRF, because all user input is sanitized and DTOs are used to send information to the client. More things can be done, however for such an application it is out of scope.

Conclusion

This application is safe from from half of the OWASP top 10, and partially safe from the other half. Security can definitely be improved to cover all of the top 10, however, for this project it is not the main focus and is out of scope. It is not nearly big enough and desirable to hack for security to be that big of a concern. If it ever grow bigger and acquired more traffic, vulnerabilities would definitely be taken more seriously.