****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **12/8/2023** | **Gavin Lyons** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

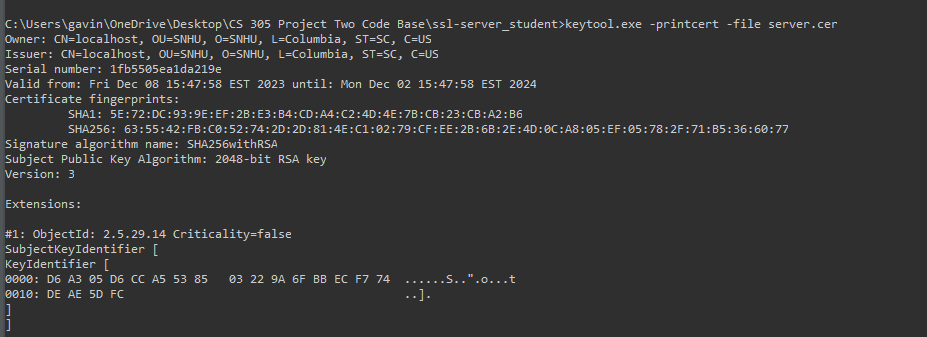
## Developer

Gavin Lyons

## Algorithm Cipher

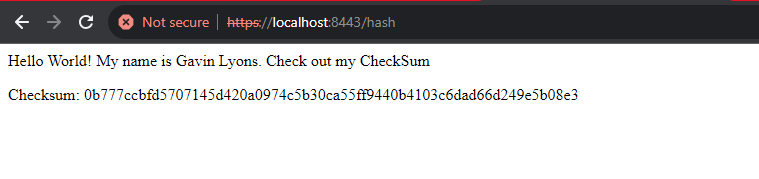
I recommend the use of SHA-256, a secure hash function that provides a higher level of cryptographic strength. While AES-256 is known for its security, it can be comparatively slower. SHA-256, part of the SHA-2 family, is widely adopted for its robustness and efficiency in generating a 256-bit hash value. The algorithm processes data in blocks, employing a series of logical operations and transformations, resulting in a fixed-size hash. Its strength lies in its resistance to collision attacks and the unpredictability of its output. SHA-256 is commonly utilized in securing digital signatures, password storage, and data integrity verification. Symmetrically, encryption relies on random numbers for generating secure keys and initialization vectors. In the realm of secure communication and data storage, SHA-256 serves as a cornerstone in modern cryptographic protocols, offering a balanced approach between security and computational efficiency.

## Certificate Generation



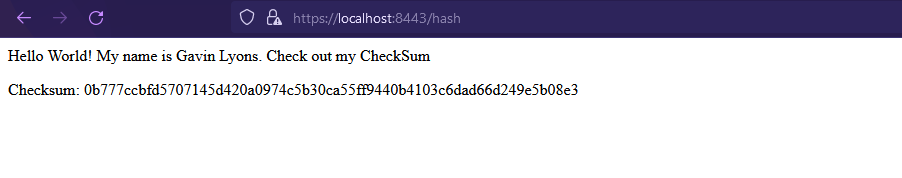
## Deploy Cipher

Insert a screenshot below of the checksum verification.



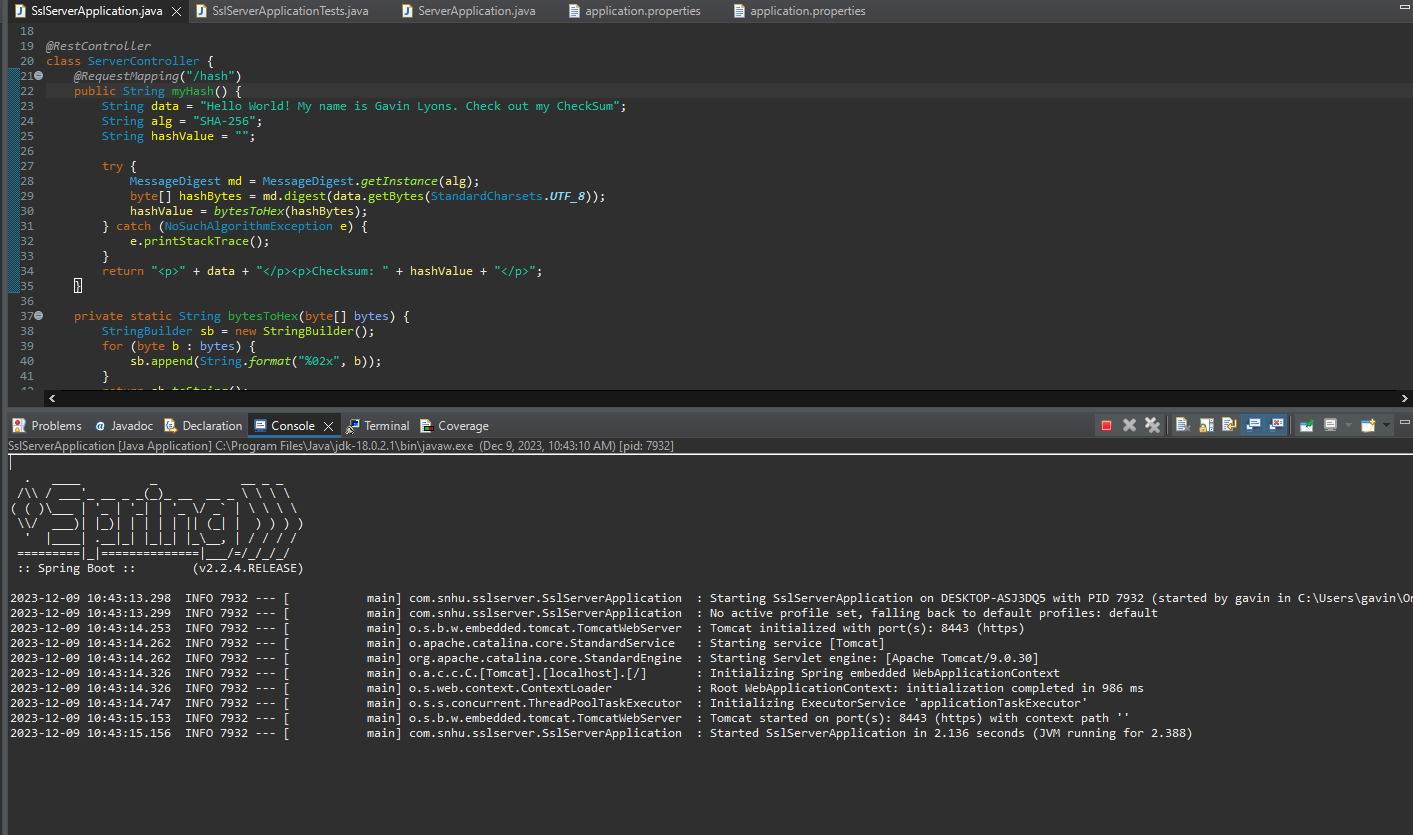
## Secure Communications

Insert a screenshot below of the web browser that shows a secure webpage.

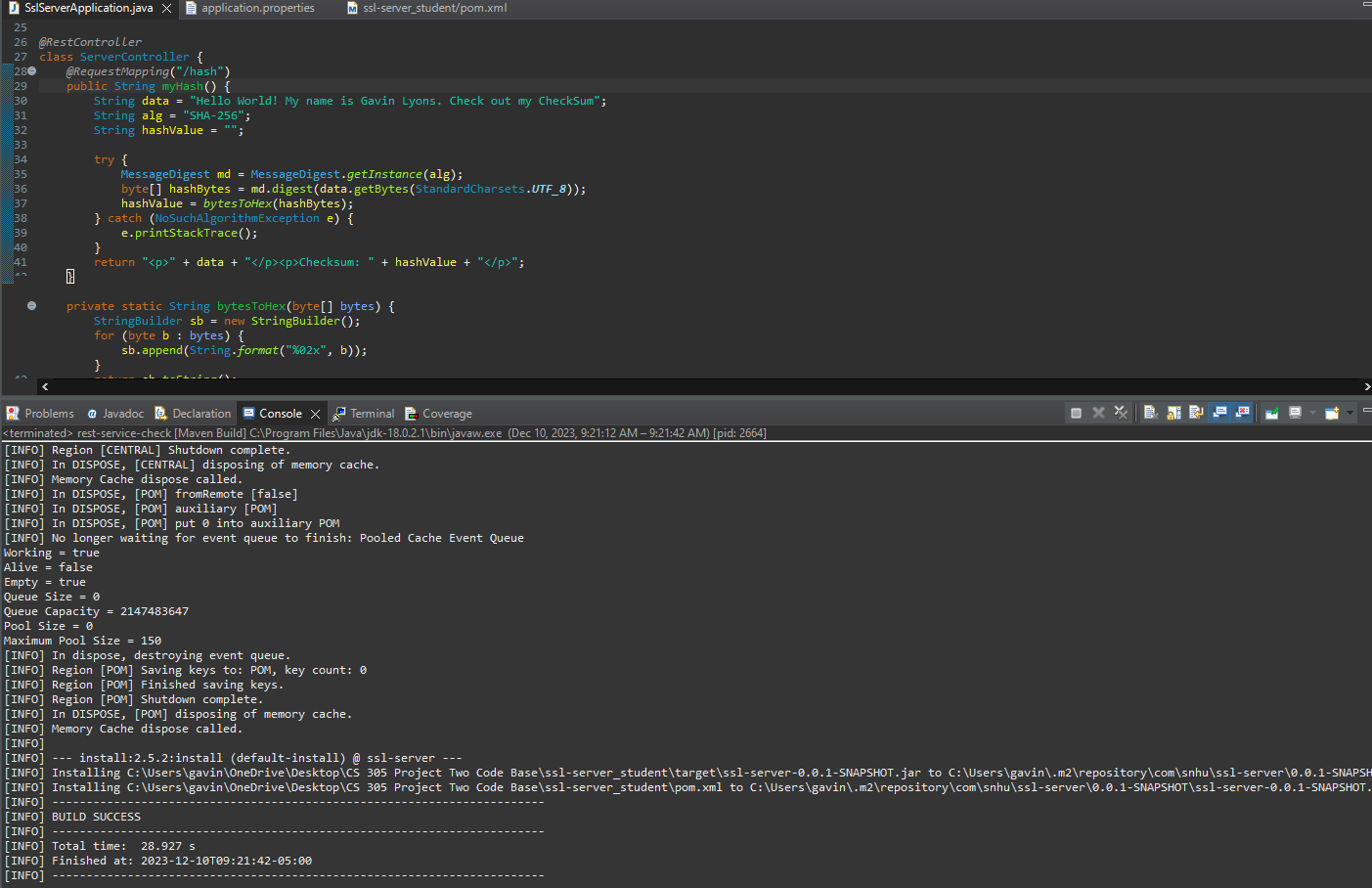


## Secondary Testing

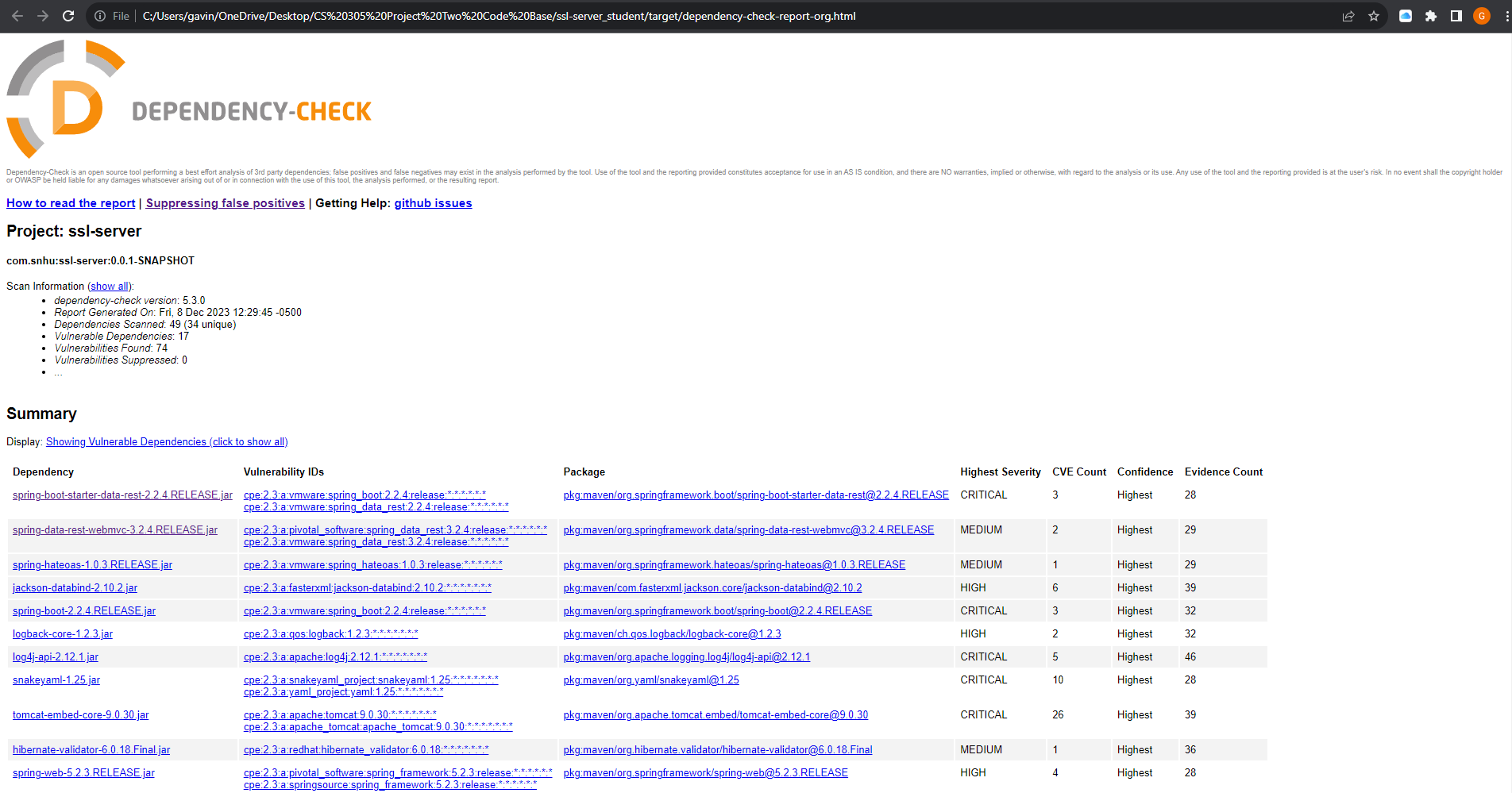
Insert screenshots below of the refactored code executed without errors and the dependency-check report.



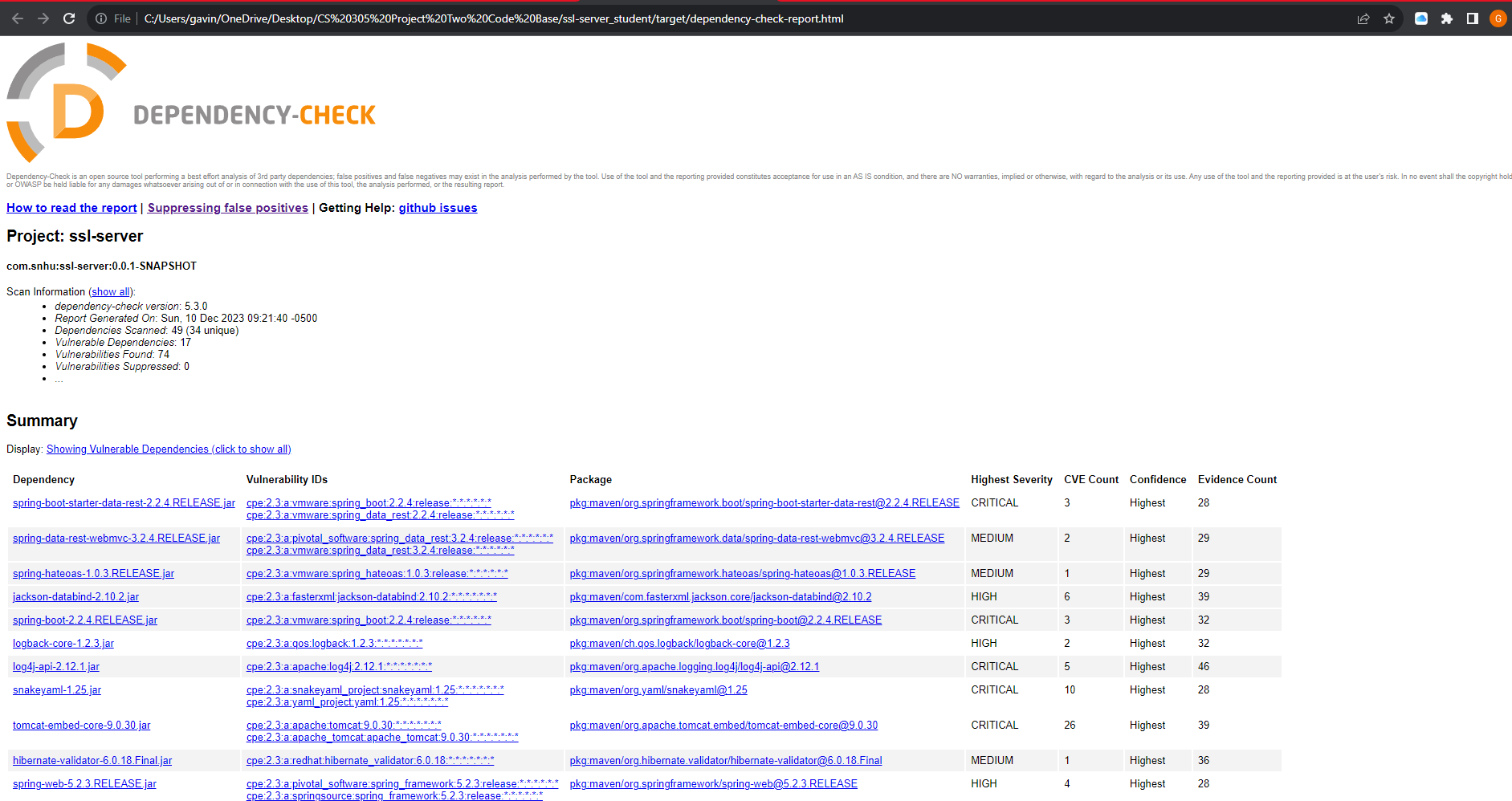
Above is the successful run of the deploying the tomcat server to localhost.



Above is the successful run of the Dependency Checker.



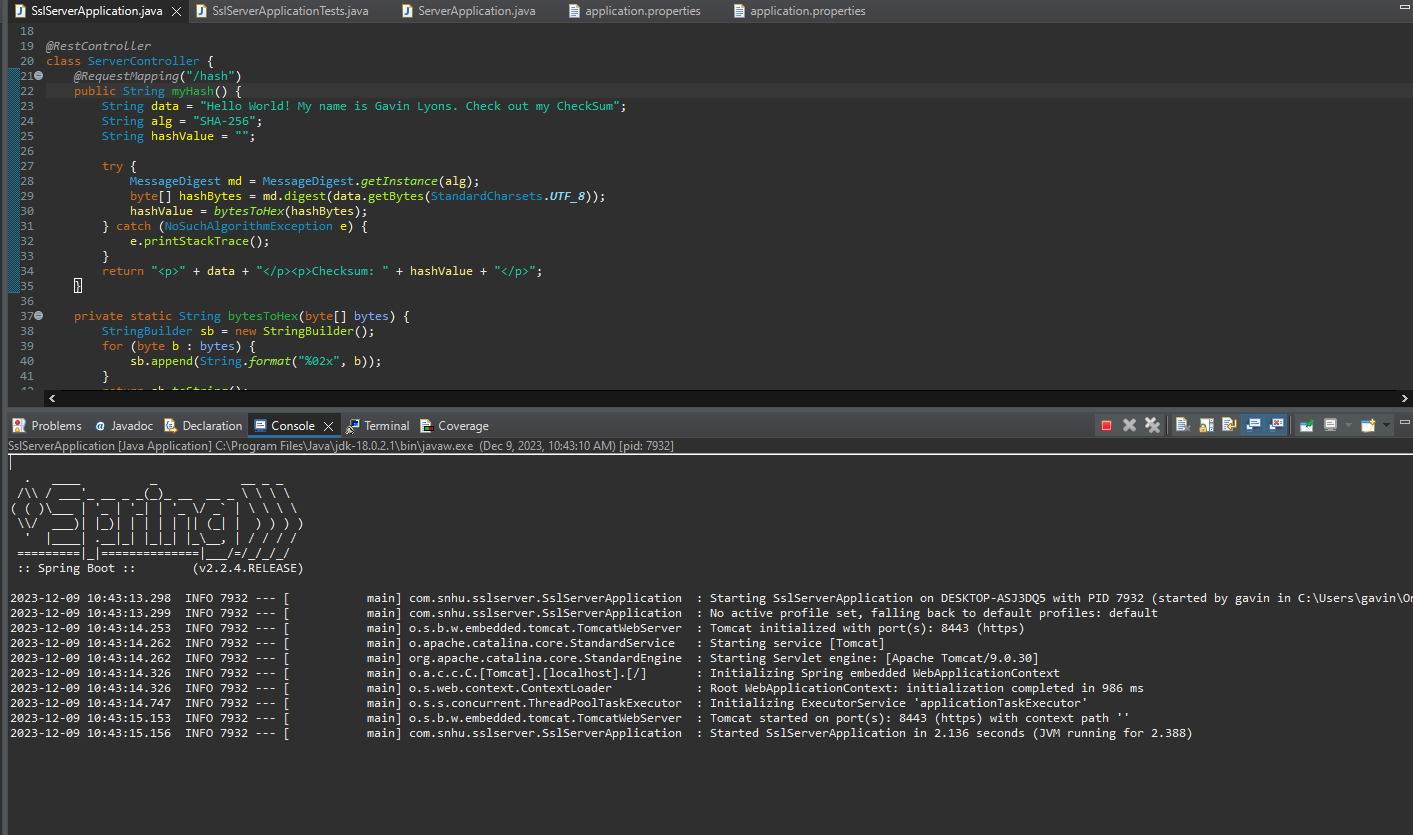
Above is the original run of the Dependency Check.



Above is the current run of the Dependency Check. There are no new errors.

## Functional Testing

Insert a screenshot below of the refactored code executed without errors.



Above is a screen shot of the refactored code running with no errors and no vulnerabilities.

## Summary

Securing this app is pretty straightforward compared to some. We've got a good handle on things because there's no external communication—it's all happening within the app. The Spring Boot code got a nice touch-up for readability and follows security testing rules. We're rolling with HTTPS by default, so our client-server talks are secure. Using SHA-256 for hashing keeps our data integrity strong. We've thrown in some security layers like HTTPS and a tough hashing algorithm to help combat outside threats. We regularly do thorough code checks and static analysis which adds an extra security boost. Based on the Vulnerability Assessment Process Flow Diagram we focused heavily on the Cryptography, Client/Server, Code error, Code Quality, and Encapsulation in this project.

## Industry Standard Best Practices

In securing the provided Spring Boot application, I applied industry-standard best practices for secure coding to mitigate known security vulnerabilities. The use of the HTTPS protocol by default ensures encrypted communication between the client and server, aligning with a widely recognized and recommended practice for safeguarding data in transit. By employing the SHA-256 algorithm for hashing, the code adheres to industry standards for secure cryptographic hash functions, addressing concerns related to data integrity. Additionally, the refactoring process focused on readability and followed established coding conventions, contributing to the maintenance of the software application's current security. The value of applying these industry-standard best practices is significant for the company's overall wellbeing. It establishes a robust defense against common security threats, reducing the risk of data tampering and unauthorized access. This, in turn, enhances the company's reputation for providing secure and trustworthy software, fostering customer trust and compliance with regulatory requirements. Moreover, adhering to industry standards promotes a proactive approach to security, helping the company stay ahead of emerging threats and potential vulnerabilities, ultimately safeguarding its assets and maintaining a resilient security posture.