****

# Practices for Secure Software Report

Table of Contents

[Practices for Secure Software Report 1](#_Toc406430730)

[Table of Contents 1](#_Toc887479603)

[Document Revision History 2](#_Toc811010882)

[Client 3](#_Toc473929213)

[Instructions 3](#_Toc1122098876)

[Developer 3](#_Toc465863667)

[1. Algorithm Cipher 4](#_Toc1398134644)

[2. Certificate Generation 4](#_Toc878479917)

[3. Deploy Cipher 5](#_Toc319594344)

[4. Secure Communications 6](#_Toc1571354737)

[5. Secondary Testing 6](#_Toc1688649513)

[6. Functional Testing 7](#_Toc678207458)

[7. Summary 9](#_Toc780407970)

[8. Industry Standard Best Practices 10](#_Toc925996708)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/21/2025** | **Roger Fisher** |  |

## 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

Roger Fisher

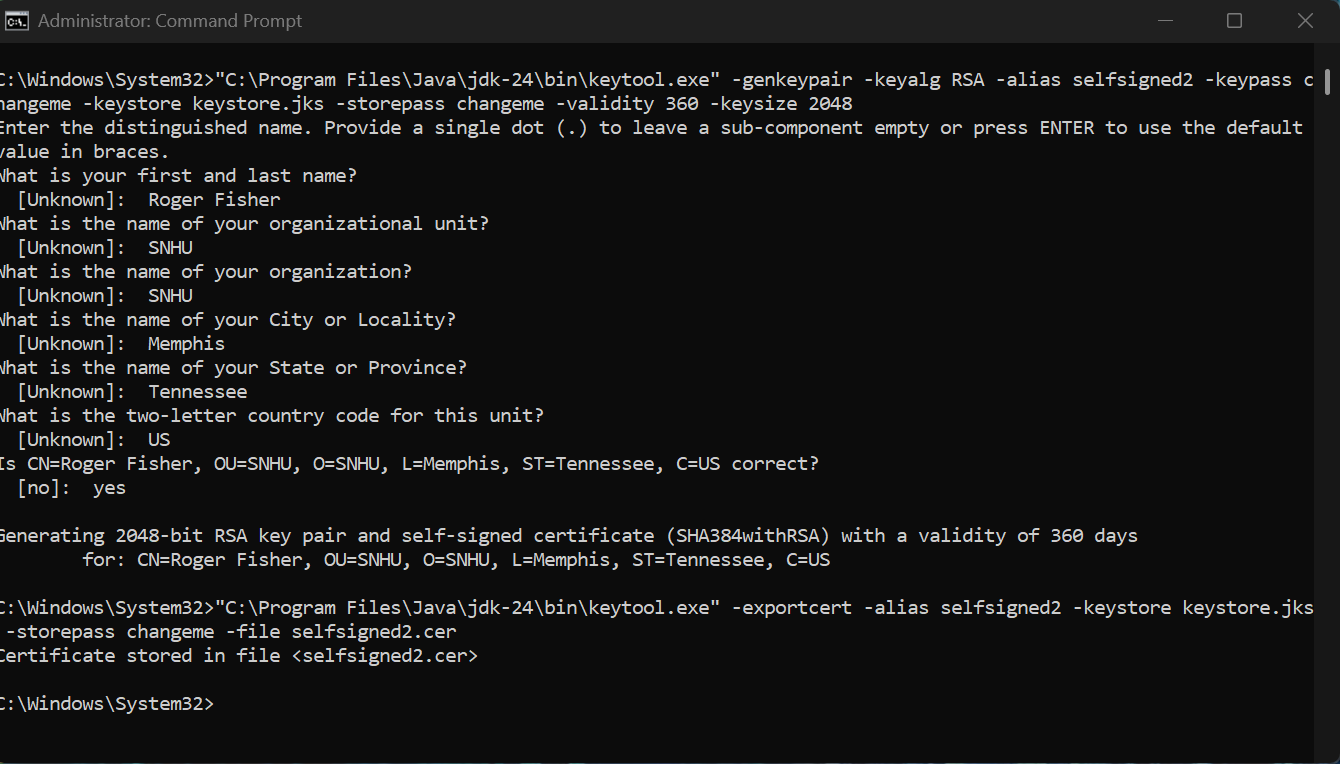
## Algorithm Cipher

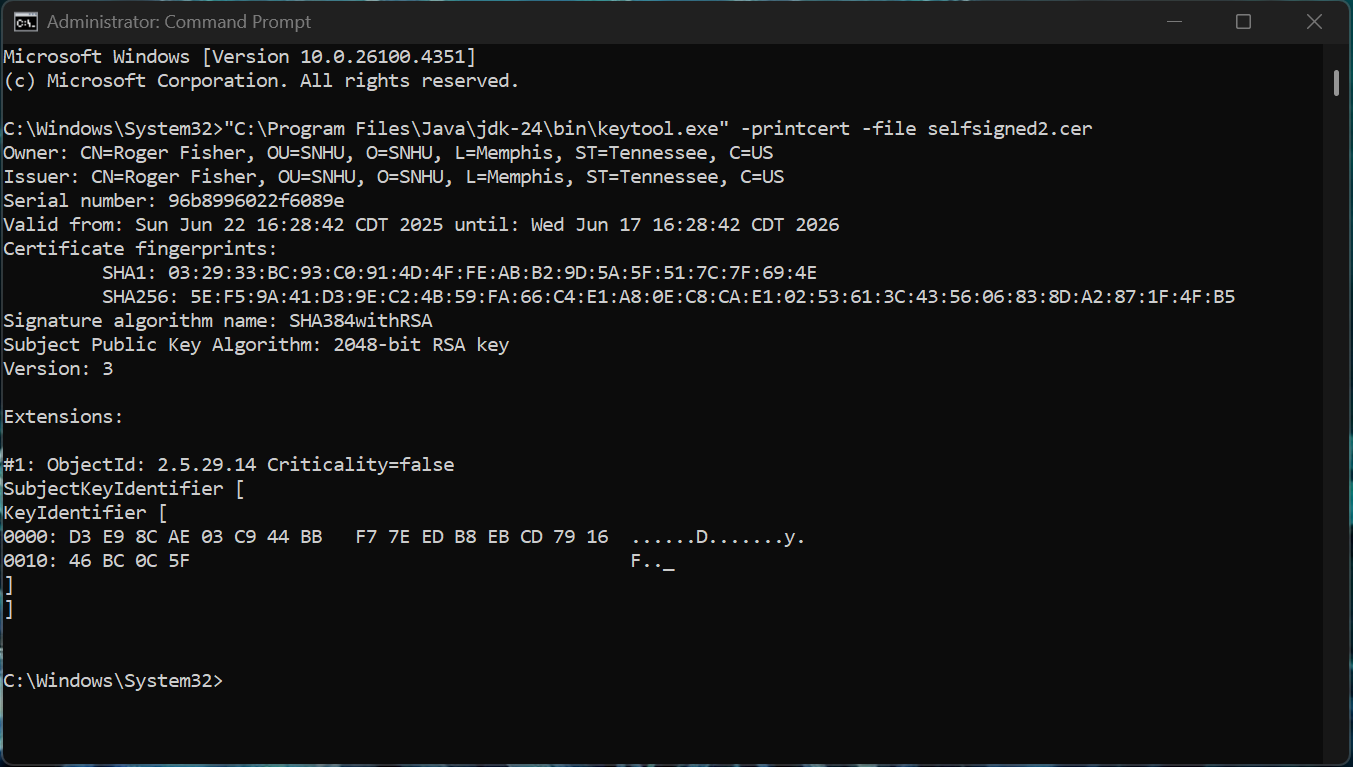
To heighten the effectiveness of the security of Artemis Financial software, the hashing algorithm I would suggest is the SHA-256. SHA-256 belongs to the family of SHA-2, which was invented by the National Security Agency (NSA) and it is currently classified as one of the safest and most dependable cryptographic hash functions to date. Its good collision and pre-image resistance makes it widely used in industries like banking, blockchain based technology and cyber security industries.

1. The SHA-256 is a cryptographic hash algorithm that accepts input of any length and creates a result that is a fixed-length 256-bit (32-byte) hash. In contrast with encryption, a hash is irreversible, that is, it is not possible to determine the original data based on the hash. It is well suited to checking file integrity, digital signatures, and safe password storage.
2. SHA-256 employs bitwise operations, modular additions, as well as compression functions with data arranged in 512-bit block groups. It gives 256-bit output. Its bit length renders it brute force and collision resilient, with 2^256 possible combinations available, way beyond the capacity of any modern computer.
3. SHA-256 is a non-symmetric (one way) algorithm and unlike encryption, it does not require keys but can be paired with randomly generated salts to enhance password hashing. SHA-256 is also used as part of a larger security system (e.g., to create cryptographic keys) or (in the case of symmetric use-cases, such as encryption) as part of a larger security protocol.
4. In 2001, the SHA-2 family was created to overcome SHA-1 problems, including SHA-256. Since then, it has become the gold standard in hashing; where it is recommended by the National Institute of Standards and Technology (NIST). Compared to the older algorithms; notably MD5 and SHA-1, which are presently defeated or rendered obsolete, SHA-256 has no known weaknesses and is a trusted algorithm in contemporary cryptographic frameworks.

## Certificate Generation

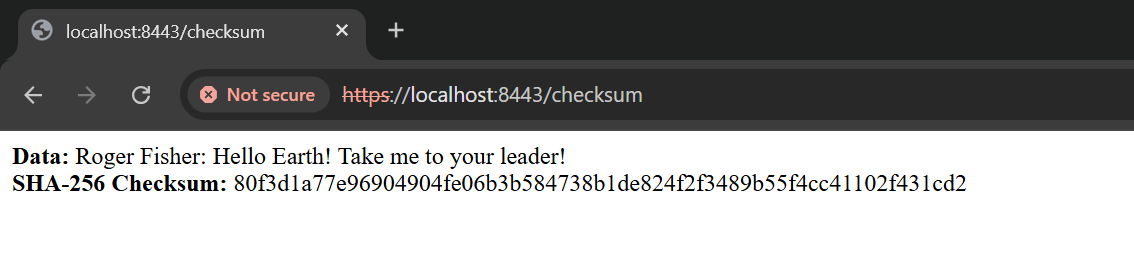
Insert a screenshot below of the CER file.





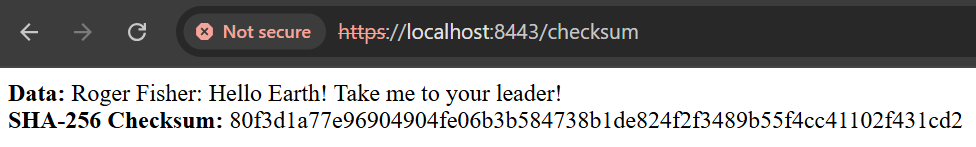
## Deploy Cipher

Insert a screenshot below of the checksum verification.



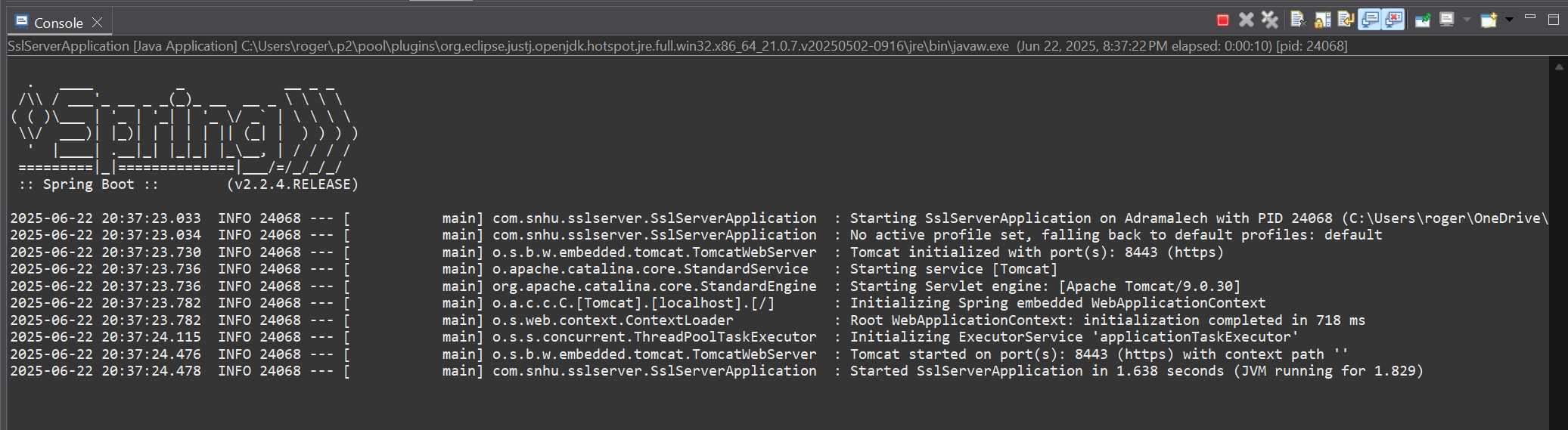
## Secure Communications

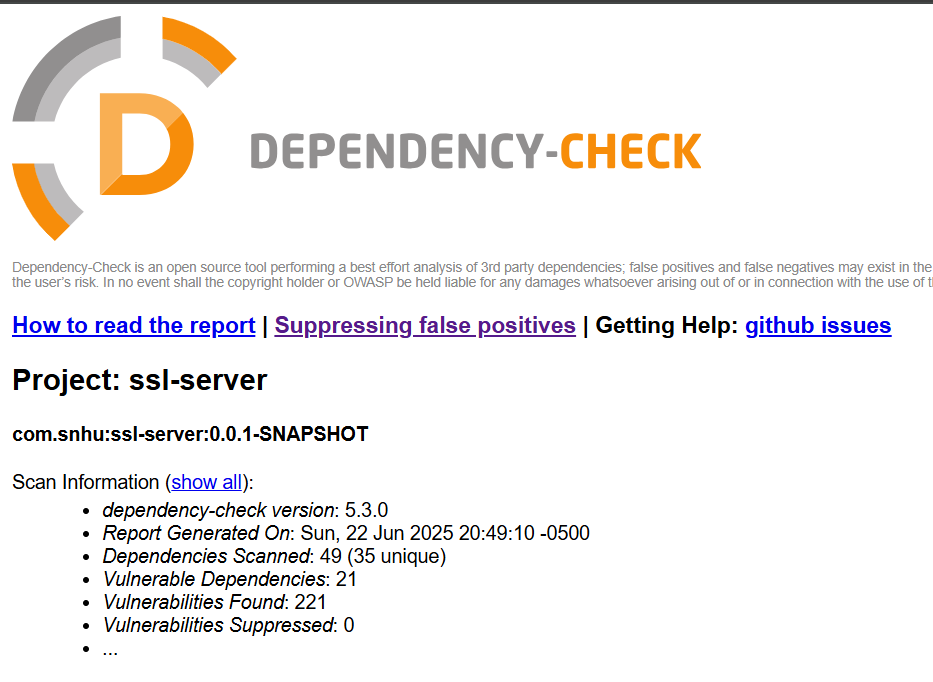
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

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

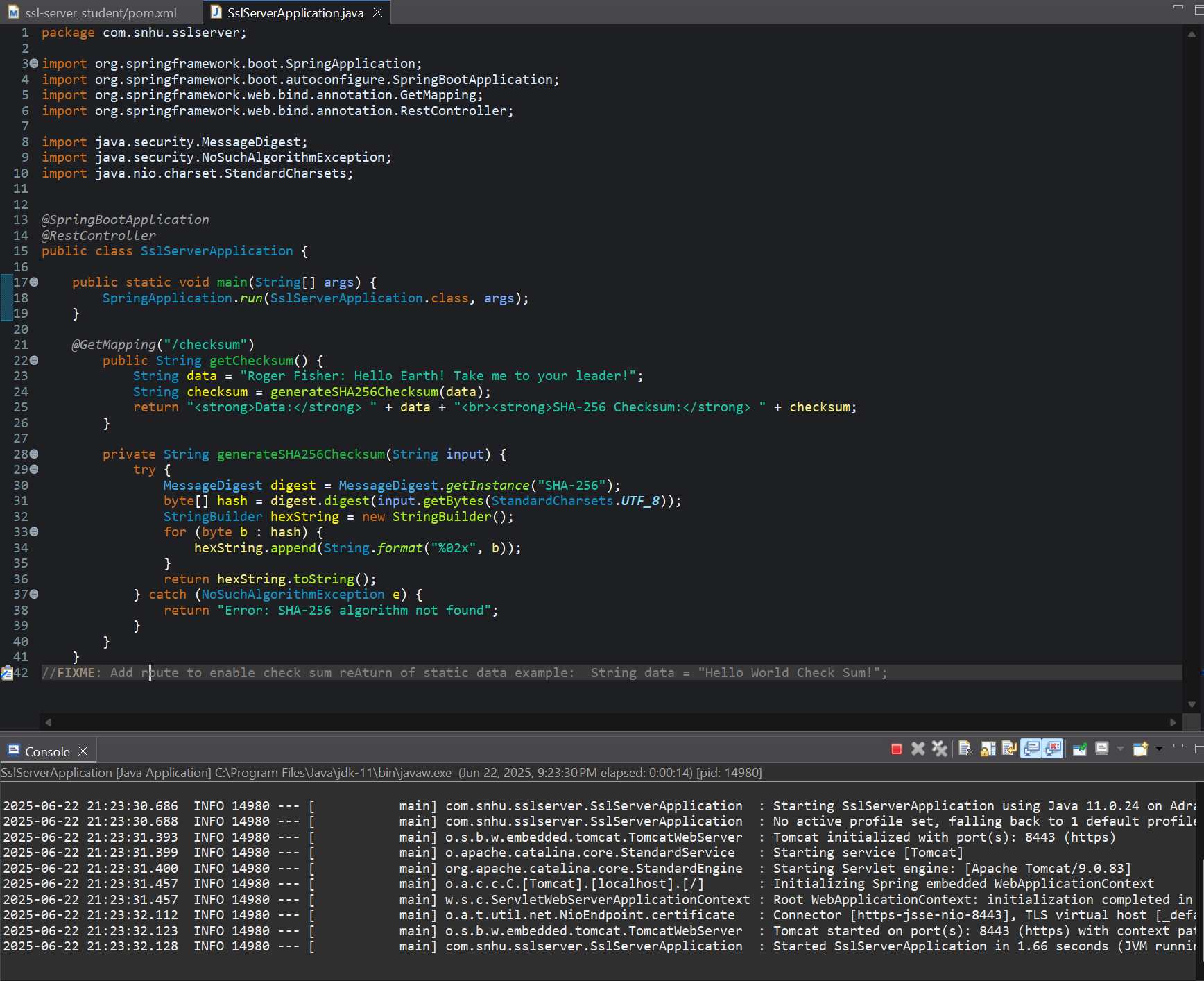




## Functional Testing

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





## Summary

1. In an attempt to enhance the security of the application, I used the methodical vulnerability assessment process provided in the supporting materials. It started with the initial static scan analysis with OWASP Dependency-Check that showed 221 known vulnerabilities in different dependencies. I then carried out a manual code inspection to determine syntactical and logical problems. Given these outcomes, I made refactoring of the pom.xml file to upgrade important dependencies, especially upgrading the Spring Boot parent to 2.7.18, upgrading the Dependency Check plugin, and the removal of outdated logging libraries. Such fixes concerned many severe vulnerabilities. As part of a follow up scan, it revealed the vulnerabilities were decreased to 53, which represented significant improvement of the security of the project. Such changes specifically indicate the remediate and retest phases of the vulnerability assessment process and reveal how direct code modification can have a direct impact on counter threats.
2. In order to improve the security of the application further, I inserted some secure programming techniques into the source code. The main functionality, the generation of a SHA-256 checksum was coded, to employ an established cryptographic algorithm by means of the java.security.Message.Digest class. I implemented UTF-8 encoding to protect the character encoding challenges and probable injection flaws. I have also considered using proper exception handling to make sure that the application does not crash or start spewing stack traces when fed with invalid or unexpected data. By eliminating redundent or obsolete dependencies and following best coding practices, I was able to minimize the application attack surface and make it more stable. These improvements provide additional security to the functionality of the application as well as the structure.

## Industry Standard Best Practices

1. In order to secure and enhance the security of the application, I used a few industries standard best practices implemented during the development and refactoring process. The most important one was the upgrading and deletion of old dependencies that were marked by OWASP Dependency-Check. I also updated Spring Boot and key libraries to current versions supporting Java 11 and thus minimized the vulnerabilities related to existing vulnerabilities. I also added secure logging with the recommended installation of SLF4J and Logback so that sensitive information is not revealed but there is still an ability to do proper auditing. The Checksum method was well implemented with exceptional handling to prevent system level leakage. Such practices follow a secure software development lifecycle which makes sure that every level of the application; including libraries, as well as code, is secured with trusted guidelines.
2. The use of industry standard best practices to secure code will provide the company with an inextricable value when it comes to its overall health. Early remedying vulnerabilities means that the organization does not run that risk later and does not incur technical debt, as well as saving on the cost to patch in an emergency. Also, a visible effort in secure development increases customer confidence and aids in adherence to the industry guidelines. The long-term advantage is a more stable, resilient application that may scale and adjust without sacrificing security. Such enhancements prevent not only the leakage of sensitive data and operations but also build a respectable reputation for the high quality and secure software products offered by the company.