****

# 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** | **8/11/2025** | **Steven Nave** |  |

## Client



## Developer

Steven Nave

## Algorithm Cipher

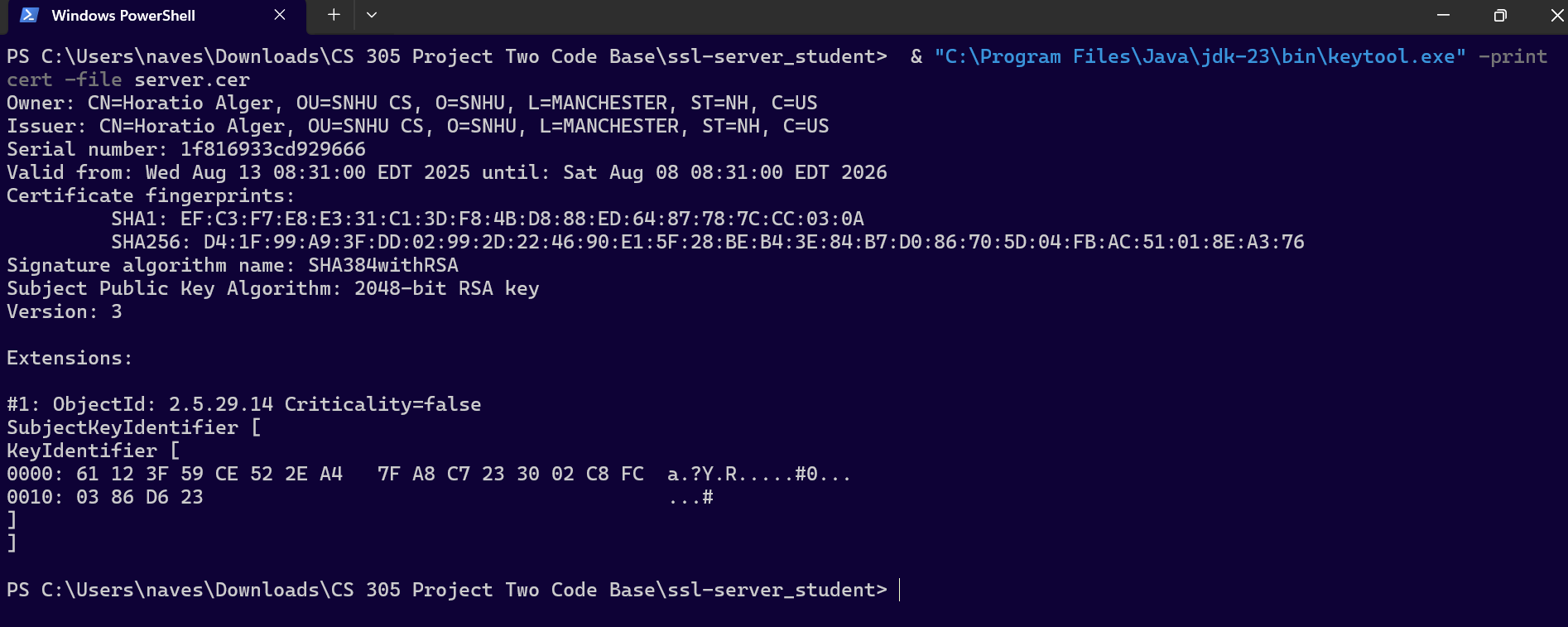
Secure Hashing Algorithm – 256

The 256-bit instance of the SHA hashing algorithm, which is a one-way, highly collision resistant algorithm that is listed on the Java Security Standard Algorithm Names list for use with the MessageDigest class.

SHA-256 is a well-known collision resistant hash function included in the Java Security Standard Algorithm Names list. It is a deterministic one-way, irreversible algorithm that produces a fixed-length 256-big digest. SHA-256 is commonly used for the purpose of data authenticity verification and checksum generation in industry, as it is so reliable, so far uncracked, and so highly unlikely to produce collisions.

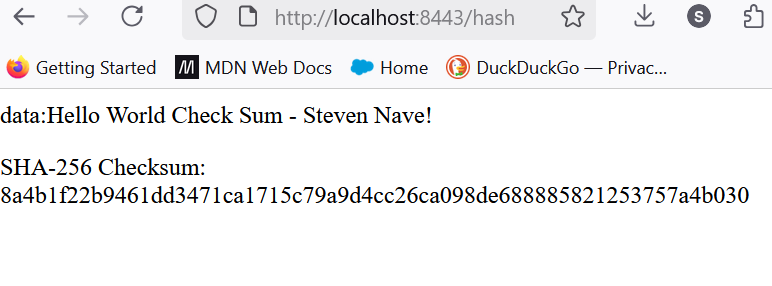
## Certificate Generation

Insert a screenshot below of the CER file.



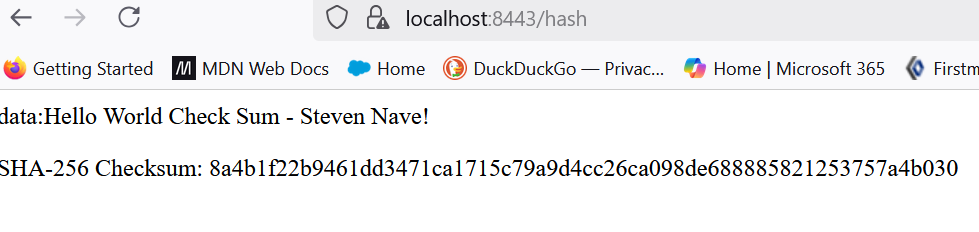
## Deploy Cipher

Insert a screenshot below of the checksum verification.



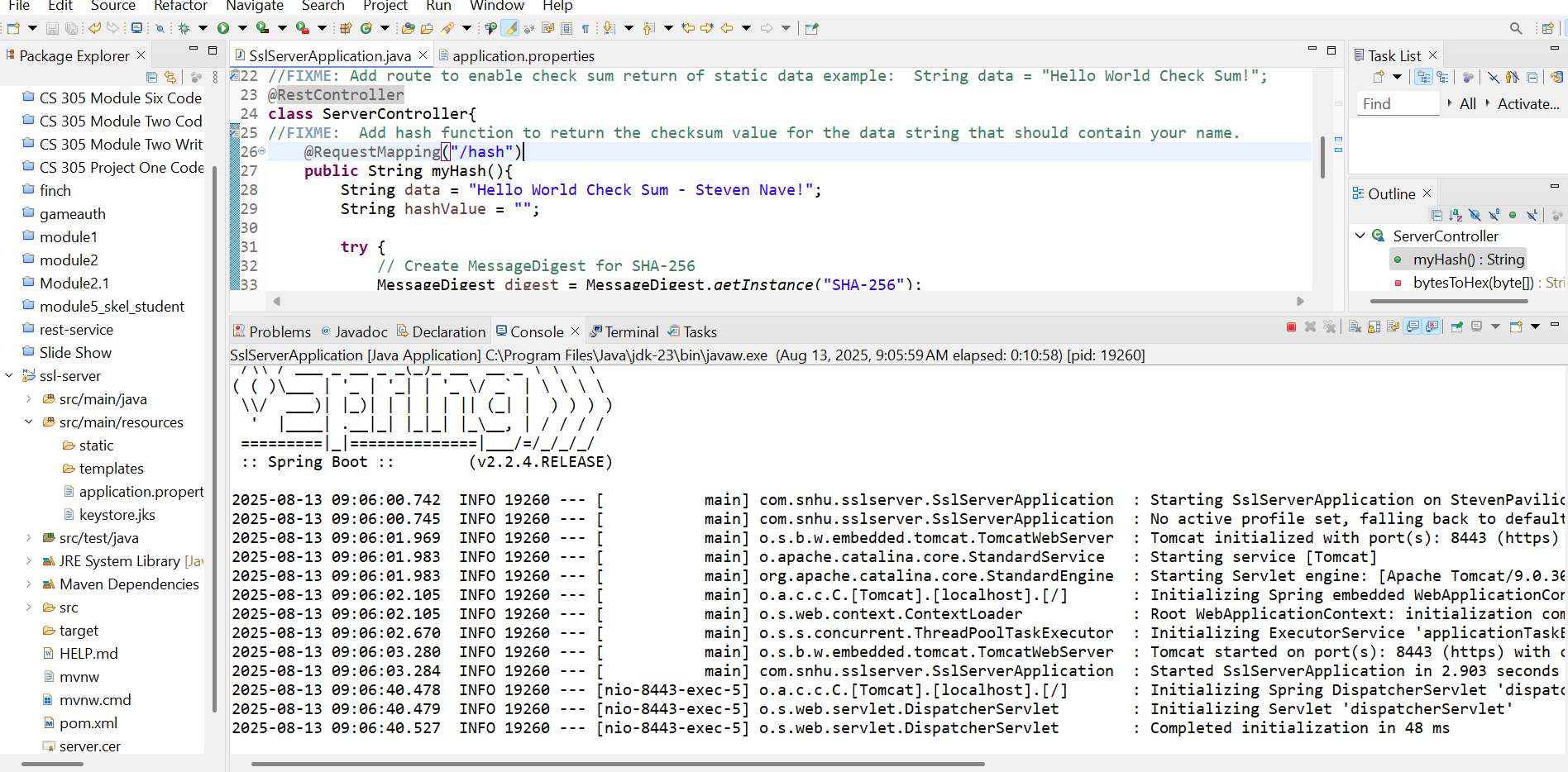
## Secure Communications

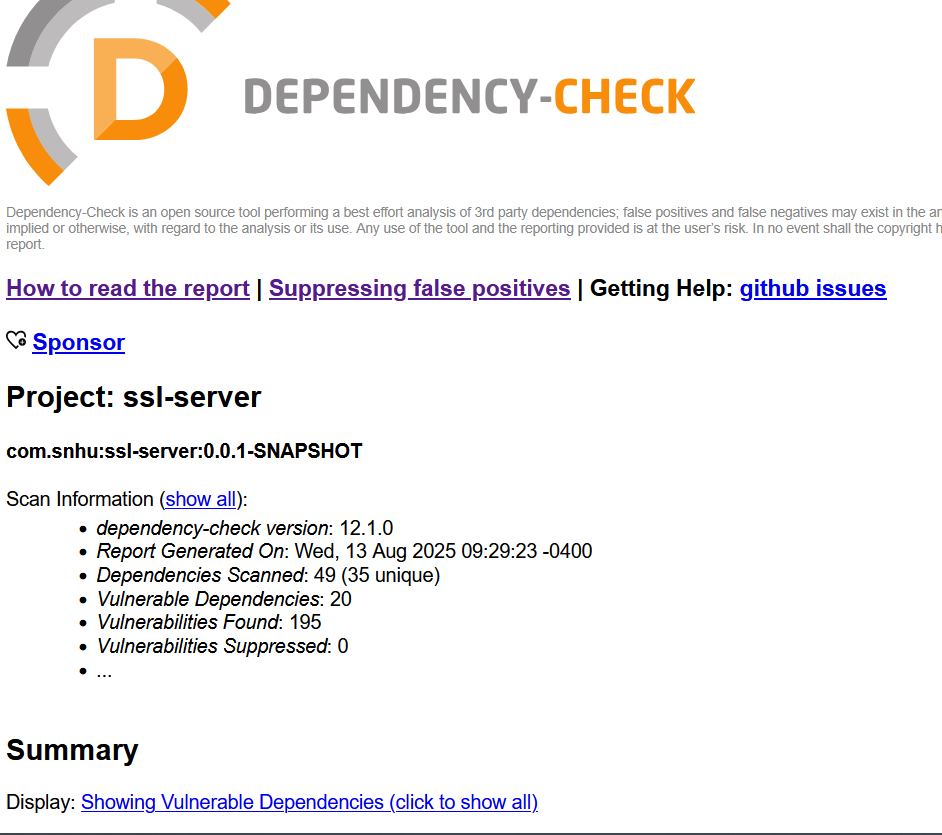
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

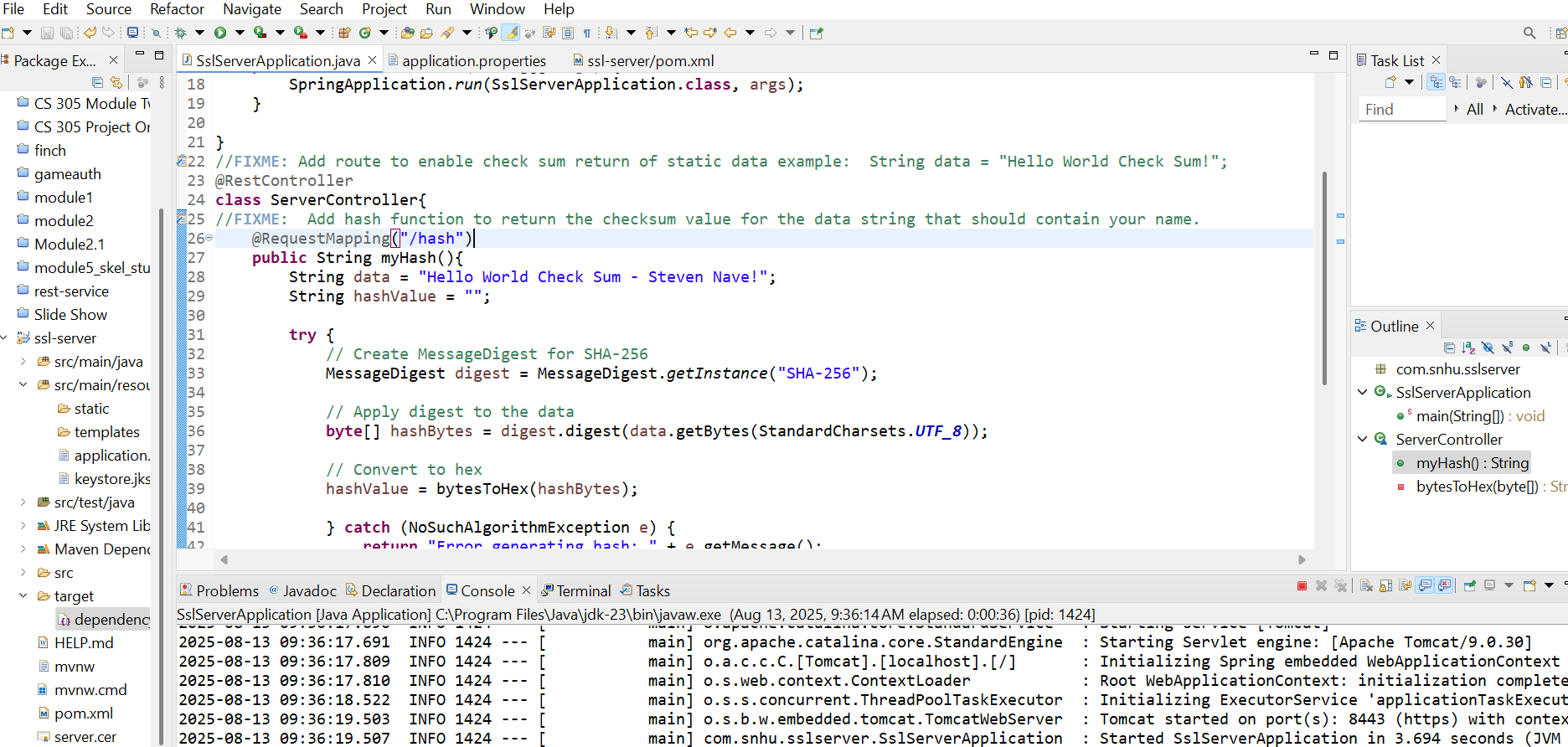
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



The Java code within this Artemis Financial project is syntactically correct, as there are no syntax errors, and the hashing algorithm has been constructed to properly handle algorithm errors using some of the Java built in exception handling classes, namely the NoSuchAlgorithmException. This falls under the code error and code quality areas of software security. The API endpoint /hash is handled carefully within the encapsulation of the ServerController class, and the SHA-256 algorithm is a powerful cryptographic method for protecting data in transit and at rest. There are no input validation errors within this code, and everything looks secure for Artemis Financial.

## Summary

As stated above the code employs these areas of the software security assessment flow diagram: Input validation, APIs, cryptography, code error, and encapsulation. The code has been refactored to have no input validation vulnerabilities, encapsulate the /hash API endpoint and secure it with an SSL certificate, handle Java errors thrown when hashing, hash the moving data with the 256-bit Secure Hashing Algorithm (SHA-256), and address all dependency vulnerabilities.

## Industry Standard Best Practices

The best practices of the Industry Standard for software security includes all of the concerns addressed above, and maintain the principles of least privilege and access, so that everyone may access and see only the least amount of data they should be privy too in order to utilize the service. Secure coding will not only keep the integrity of the data for the customers and company well into the future, and safeguard against loss of business and revenue, but most importantly hedge against the violation of trust and privacy that will damage people immediately and besmirch the good name of the company for a very long time – if they survive the loss of trust at all.