****

# 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** | **06/16/2025** | **Sharif Ayesh** |  |

## 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

Sharif Ayesh

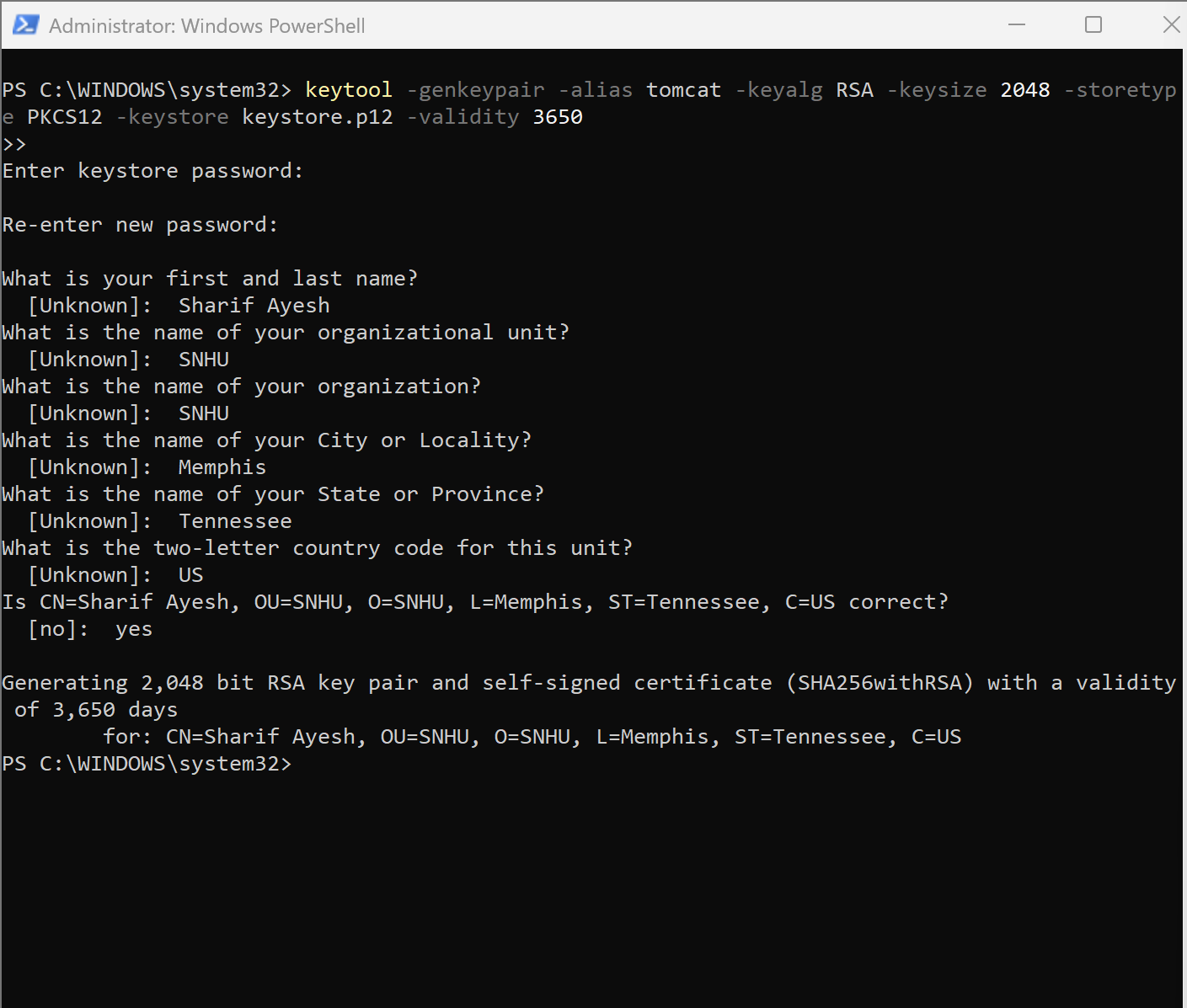
## Algorithm Cipher

In this project, I implemented the SHA-256 cryptographic hash function to generate a checksum for validating data integrity. SHA-256 is part of the SHA-2 family developed by the National Security Agency (NSA) and is widely regarded as secure against modern collision and preimage attacks. With a 256-bit output, it produces a fixed-length digest regardless of input size, making it ideal for verifying that files haven’t been tampered with. While SHA-256 doesn’t encrypt data, it plays a critical role in secure software development by ensuring that archived or transmitted data remains unaltered.

Although my code uses hashing rather than encryption, it’s essential to note that encryption ciphers, such as AES (Advanced Encryption Standard), are the standard for protecting data confidentiality. AES is a symmetric key cipher that supports 128, 192, and 256-bit key sizes and relies heavily on random number generation for secure key creation. Unlike hashing, encryption is a two-way process, allowing data to be decrypted when needed. Historically, encryption algorithms have evolved from the now-broken DES to the widely adopted AES, which is still recommended today for encrypting sensitive archives, especially when paired with secure hashing to verify integrity.

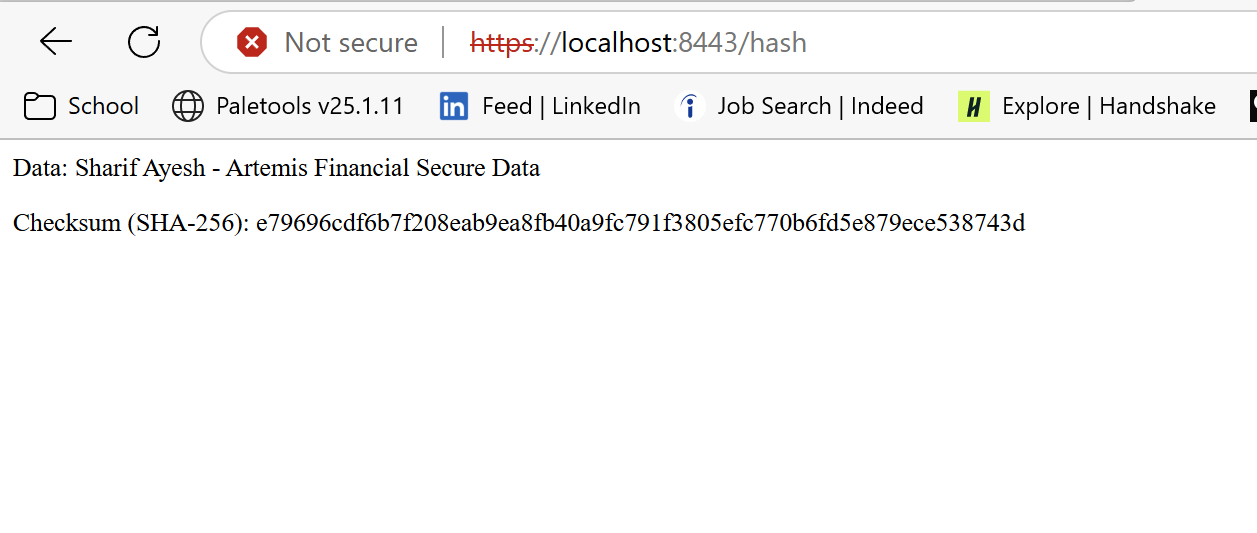
## Certificate Generation

Insert a screenshot below of the CER file.



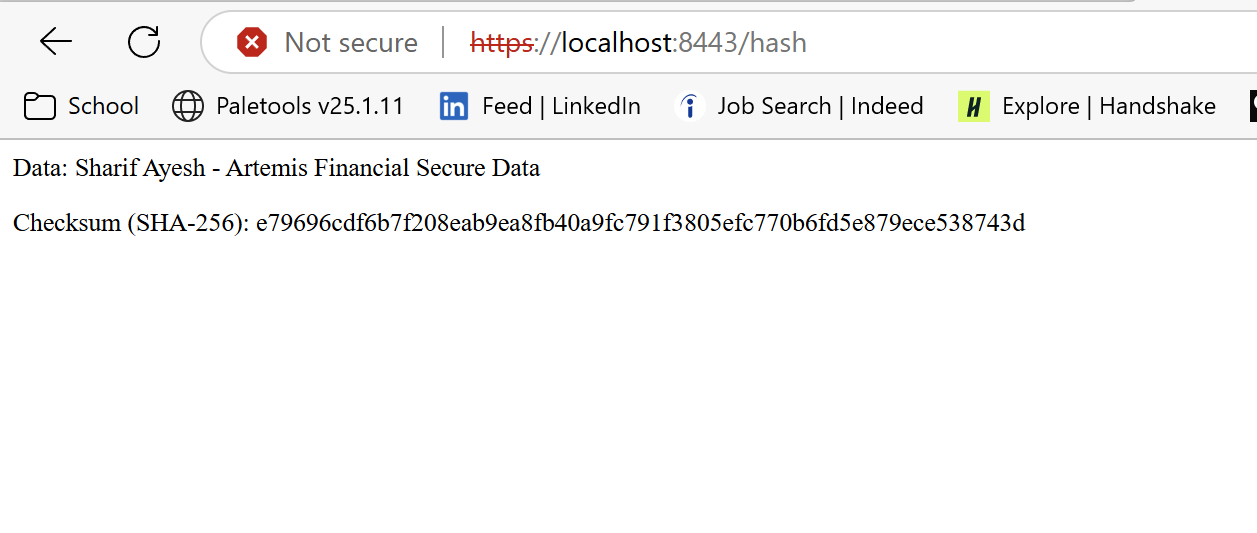
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

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



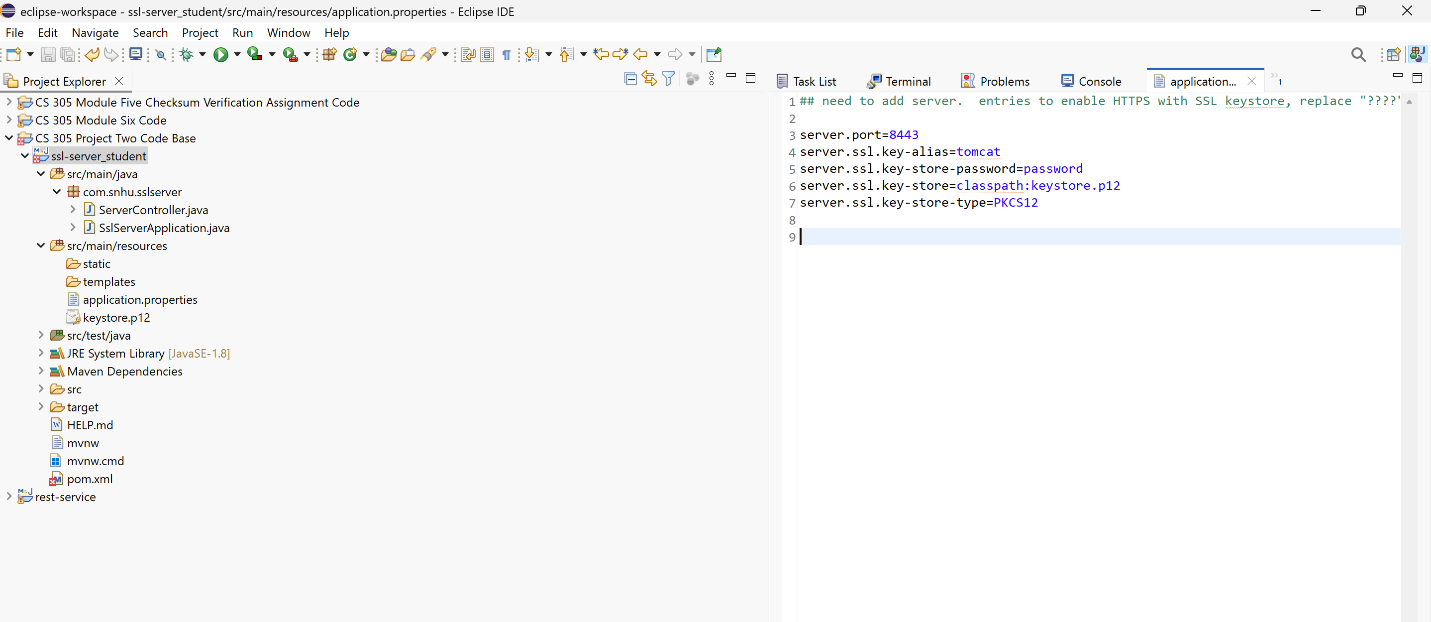
## Secondary Testing

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

A close-up of a white background

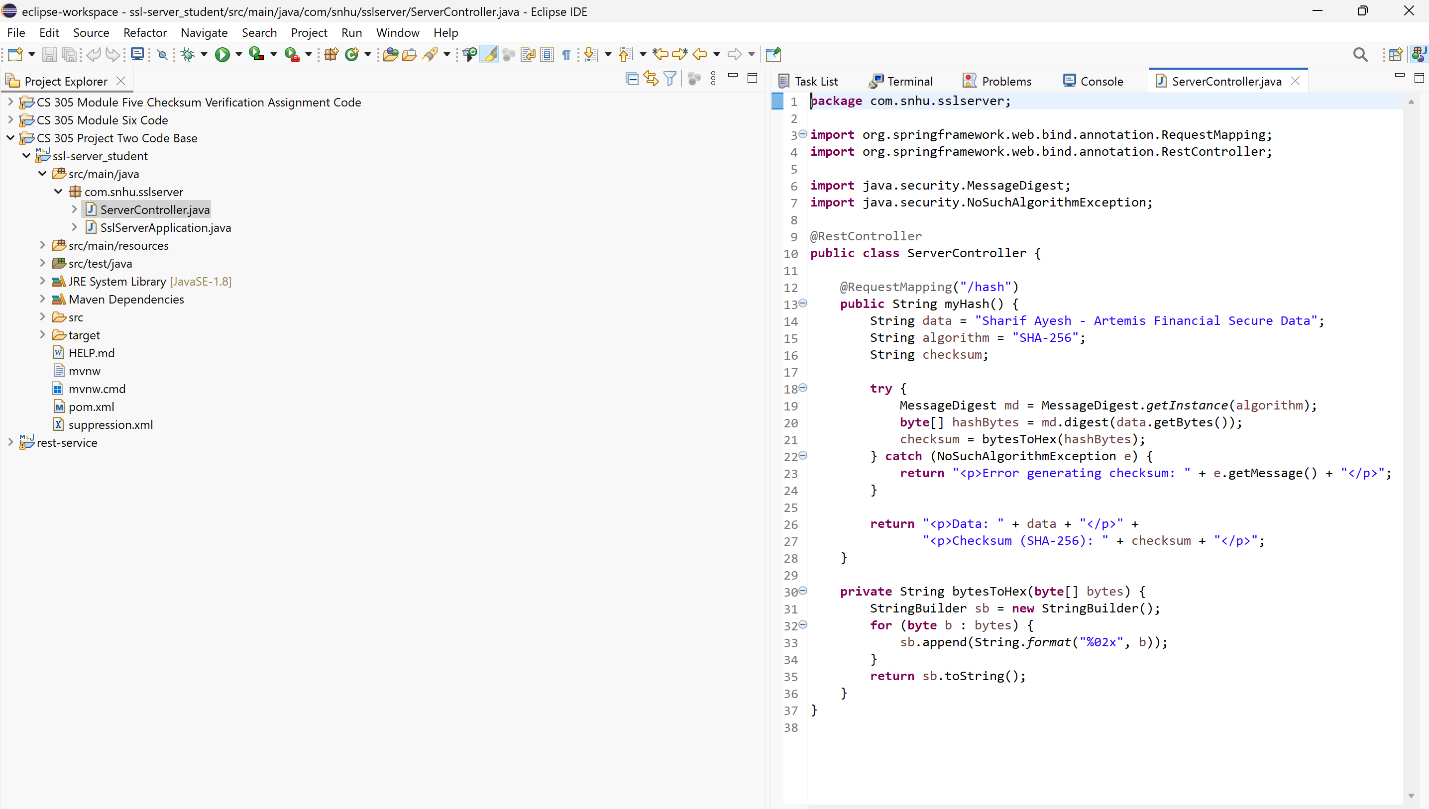
AI-generated content may be incorrect.





## Functional Testing

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



## Summary

The code was refactored to align with secure software development practices, primarily by implementing SHA-256 hashing for data validation and integrating OWASP Dependency-Check to identify and manage vulnerabilities. Based on the vulnerability assessment process flow, key areas addressed include dependency analysis*,* vulnerability suppression*, and* checksum validation. Suppression rules were added to eliminate false positives for known, non-exploitable CVEs. This ensured that the focus remained on real threats. Additionally, HTTPS was configured for secure data transmission, further strengthening the transport layer.

To add layers of security, I adopted a layered defense approach. First, I ensured that any data tampering could be detected by using cryptographic hashing (SHA-256). Then, the project’s Maven dependencies were scanned and monitored through automated vulnerability assessments, with suppression files configured for trusted components. Finally, I ensured that the application ran over SSL, securing both external access and internal communications. These steps together represent a proactive effort to assure the software at multiple levels: data integrity, dependency management, and network transport.

## Industry Standard Best Practices

To maintain the software application’s existing security, I applied several industry-standard best practices for secure coding throughout the refactoring process. First, I used the @RestController annotation in the ServerController class to ensure that web requests are properly routed and that application data is exposed only through secure and defined endpoints. This follows RESTful design principles, helping to prevent unauthorized access. I also implemented a SHA-256 cryptographic hash function, recognized for its strength and resistance to collision attacks, to verify the integrity of data transmitted through the application. This function ensures that the checksum output cannot be easily forged or reversed, which is crucial for protecting sensitive information. Furthermore, I integrated the OWASP Dependency-Check plugin into the project’s Maven pom.xml file to automatically scan for known vulnerabilities in third-party libraries. I addressed several of these vulnerabilities through suppression rules for safe, legacy components and documented their presence to support secure deployment.

Following these secure coding standards directly enhances the organization’s ability to deliver safe and reliable software. Industry best practices such as input validation, encryption, and dependency analysis protect applications against common threats like injection attacks, data breaches, and insecure dependencies. They also promote long-term maintainability by enforcing code consistency and compliance with regulatory frameworks, such as NIST and OWASP. Applying these practices reduces the risk of system compromise, fosters customer trust, and demonstrates the company’s commitment to cybersecurity. In a business environment where software vulnerabilities can lead to financial loss and brand damage, integrating these secure development techniques is crucial for achieving sustainable success and maintaining legal accountability.