****

# 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** | **June 28 2025** | **Edgar I Pena** |  |

## Client



## Developer

Edgar I. Pena

## Algorithm Cipher

For this project, we recommend using AES (Advanced Encryption Standard) as the encryption cipher. AES is a widely trusted, government-approved algorithm used to protect sensitive data. It uses the same key for both encryption and decryption, which makes it a symmetric encryption algorithm. This method is efficient and well-suited for financial applications that require the protection of large amounts of data quickly and securely.

AES supports multiple key sizes: 128-bit, 192-bit, and 256-bit. The higher the bit level, the stronger the encryption. AES-128 is commonly used and secure for most cases, while AES-256 offers extra strength for more sensitive data.

Random numbers are used during the encryption process to make sure each session is unique and unpredictable. This helps prevent attackers from guessing or reusing encrypted data. AES is symmetric, meaning it uses one shared key, unlike asymmetric encryption (like RSA), which uses two keys.

Earlier algorithms, such as MD5 and DES, were once standard but are no longer considered secure due to advances in computing power and known weaknesses. AES replaced these older methods and is now considered the modern standard for secure encryption in both government and industry.

By using AES, we ensure that the application adheres to current encryption standards and provides robust protection for Artemis Financial’s client data.

## Certificate Generation

Insert a screenshot below of the CER file.

server.cer


## Deploy Cipher

Insert a screenshot below of the checksum verification.

A screenshot of a computer

Description automatically generated

## Secure Communications

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

A screenshot of a computer

Description automatically generated

## Secondary Testing

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

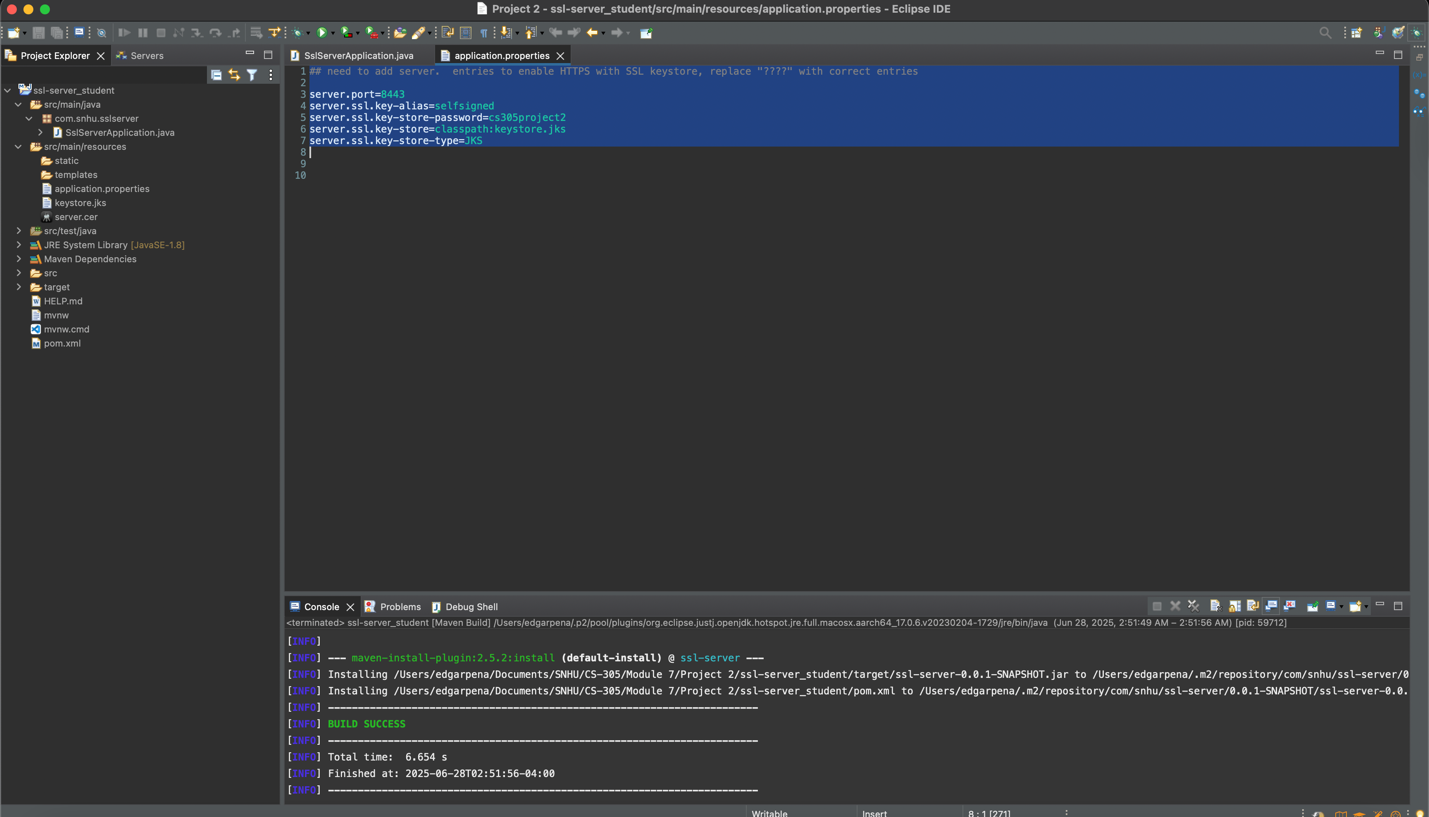
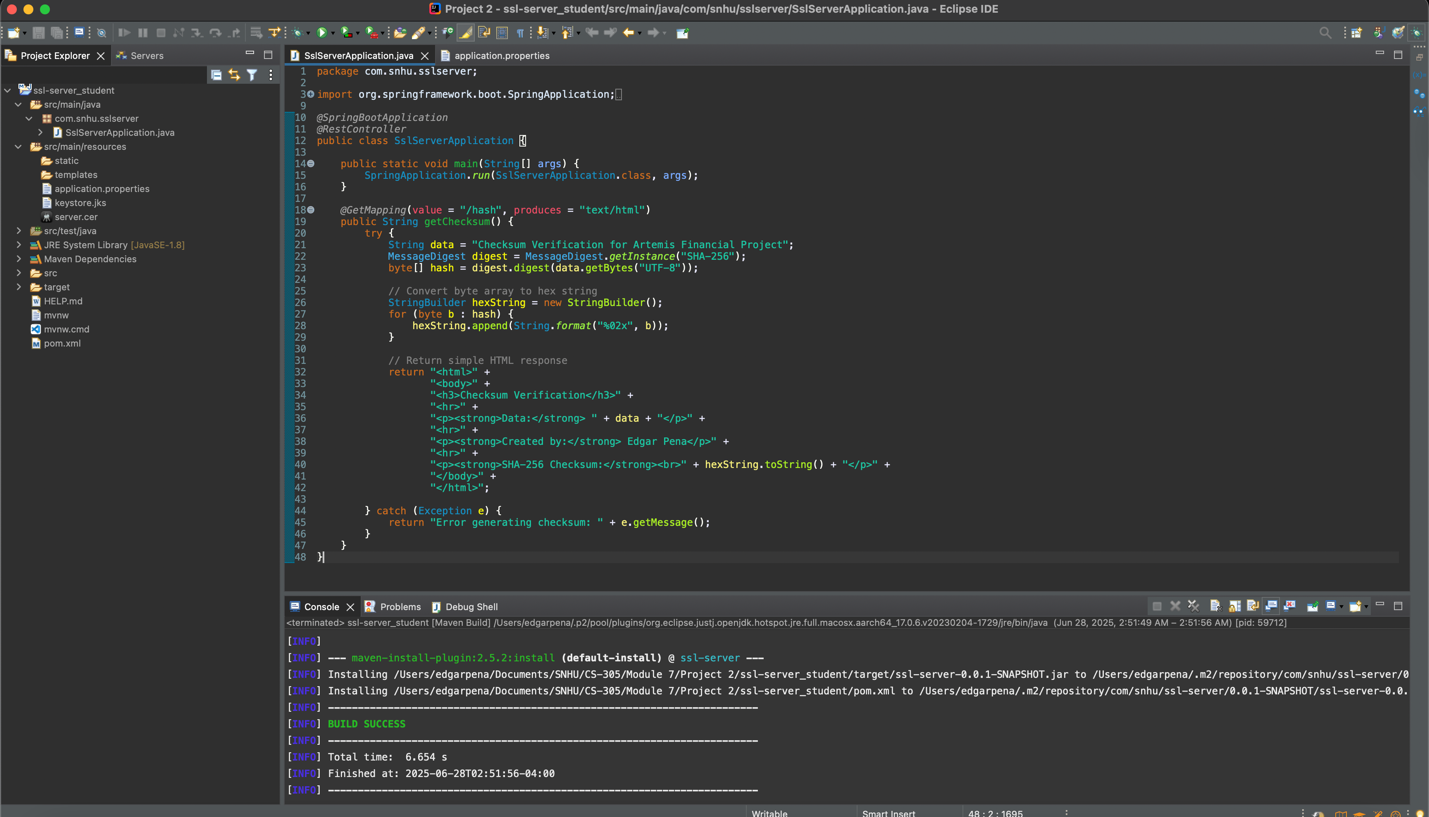
A screenshot of a computer program

Description automatically generatedA screenshot of a computer

Description automatically generated

## Functional Testing

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



## Summary

To enhance the security of Artemis Financial’s web application, we implemented two key changes: we added checksum verification to safeguard data integrity, and we enabled HTTPS to secure communications between the server and the user’s browser. These updates follow a structured process based on the vulnerability assessment flow diagram, helping ensure that your software aligns with current security standards.

The process began by manually reviewing the existing code and identifying areas where modern security practices could be added. We started with checksum verification, which ensures that the data has not been altered in any way. We created a new /hash route in the application that uses the SHA-256 hashing algorithm to generate a unique fingerprint from a known data string. This makes it easy to detect any unauthorized changes to that data.

Next, we focused on securing the communication channel. We generated a self-signed SSL certificate using Java Keytool and created a keystore to hold it. Then, we configured the application to use HTTPS on port 8443 by updating the application.properties file. Once this was done, the application was able to securely encrypt all data sent between the browser and the server.

To make sure these changes did not create any new issues, we ran a static security scan using the OWASP Dependency-Check tool. This tool checks all the project’s dependencies for known vulnerabilities. The scan confirmed that the updates did not introduce any new security concerns.

## Industry Standard Best Practices

Throughout this project, we adhered to industry-standard best practices for secure software development to help protect Artemis Financial’s application from known vulnerabilities and potential future security risks.

To maintain the application’s existing security, we avoided making unnecessary changes to stable code and focused only on areas that required updates. This helped reduce the chance of introducing bugs or creating new vulnerabilities. All sensitive information, such as the keystore password, was kept outside the source code in the application.properties file, which is a common best practice. This keeps confidential data separate from the core application logic, making it easier to manage securely.

We also utilized trusted and modern tools, including the SHA-256 hashing algorithm for checksum verification and HTTPS encryption with a secure SSL certificate. Both are widely recognized methods for securing data and ensuring its integrity. Additionally, we conducted a static analysis using the OWASP Dependency-Check tool to scan all project dependencies for known security vulnerabilities. This helped confirm that none of the changes introduced new risks.

Applying these best practices supports Artemis Financial’s long-term goals. Secure coding protects client data, improves trust, and lowers the risk of cyberattacks or data breaches. It also aligns the company with compliance standards and demonstrates a proactive approach to developing safe and reliable financial software.