****

# Practices for Secure Software Report

**Table of Contents**

[**Document Revision History 3**](#_heading=h.30j0zll)

[**Client 3**](#_heading=h.1fob9te)

[**Instructions 3**](#_heading=h.3znysh7)

[**Developer 4**](#_heading=h.2et92p0)

[**1. Algorithm Cipher 4**](#_heading=h.tyjcwt)

[**2. Certificate Generation 4**](#_heading=h.3dy6vkm)

[**3. Deploy Cipher 4**](#_heading=h.1t3h5sf)

[**4. Secure Communications 4**](#_heading=h.4d34og8)

[**5. Secondary Testing 4**](#_heading=h.2s8eyo1)

[**6. Functional Testing 4**](#_heading=h.17dp8vu)

[**7. Summary 4**](#_heading=h.3rdcrjn)

[**8. Industry Standard Best Practices 4**](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **4/20/2024** | **Josh Hall** |  |

## Client



## Developer

Josh Hall

## Algorithm Cipher

Given that Artemis Financial aims to modernize its operations and enhance the security of its web application for transferring sensitive client data, the SHA-256 algorithm would be a suitable choice for implementing file verification through checksums.

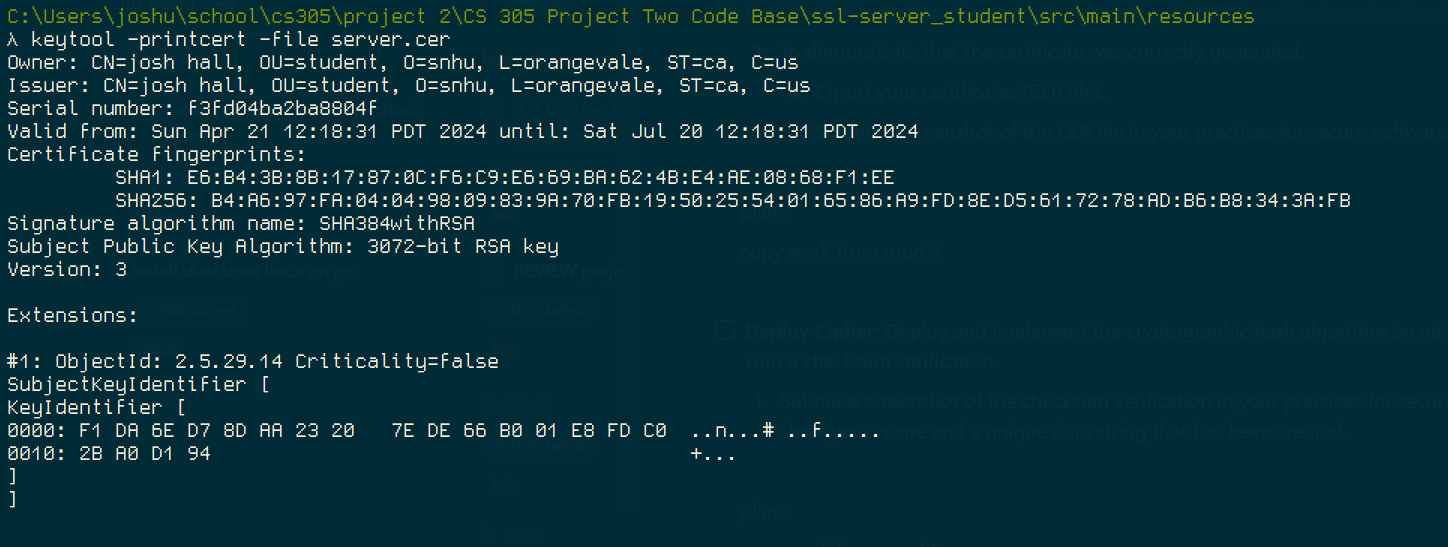
SHA-256, a member of the Secure Hash Algorithm family, is a cryptographic hash function designed by the NSA. It generates a fixed-size 256-bit hash value from input data, making it highly resistant to collision attacks and ensuring data integrity. The algorithm operates by iterating through a series of modular additions and bitwise rotations, resulting in a hash value that is unique to the input data.

In the context of secure communications for Artemis Financial, the 256-bit hash output provides a high level of security against malicious tampering and data corruption. It can efficiently verify the integrity of transferred files by generating a checksum for each file at the sender's end and comparing it with the checksum calculated at the receiver's end. This checksum comparison ensures that the data has not been altered during transmission, providing an additional layer of security to protect client data and financial information.

Furthermore, SHA-256 supports the use of random numbers for salting, enhancing its resistance to pre-image attacks. While symmetric keys are primarily used for encryption and decryption processes, hash functions like SHA-256 are designed for data integrity verification and do not require keys for their operation. Therefore, implementing SHA-256 for file verification in Artemis Financial's web application would align with modern security practices and provide great protection against potential security vulnerabilities.

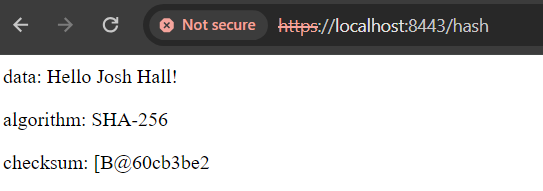
## Certificate Generation

Insert a screenshot below of the CER file.



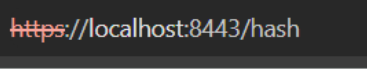
## Deploy Cipher

Insert a screenshot below of the checksum verification.



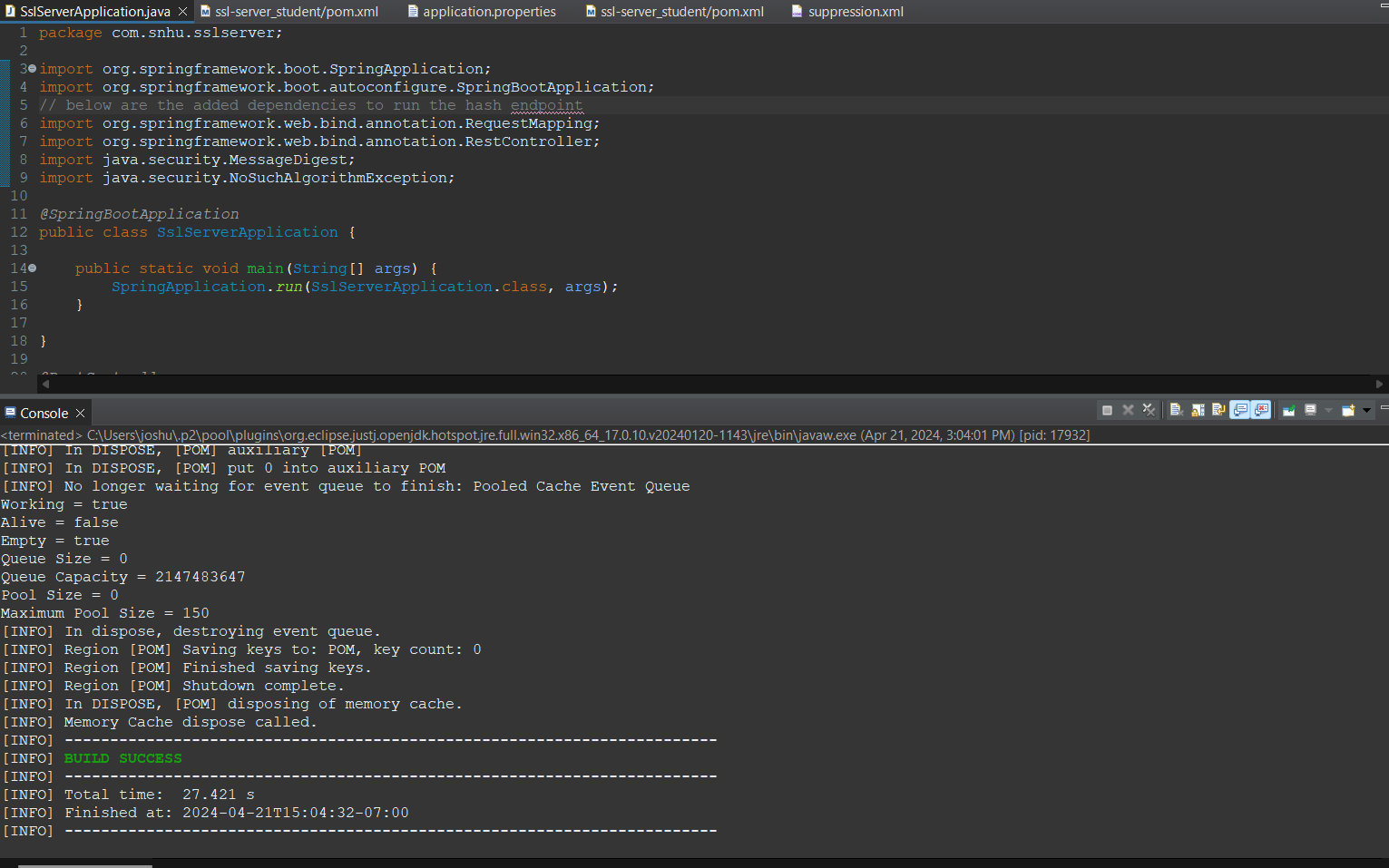
## Secure Communications

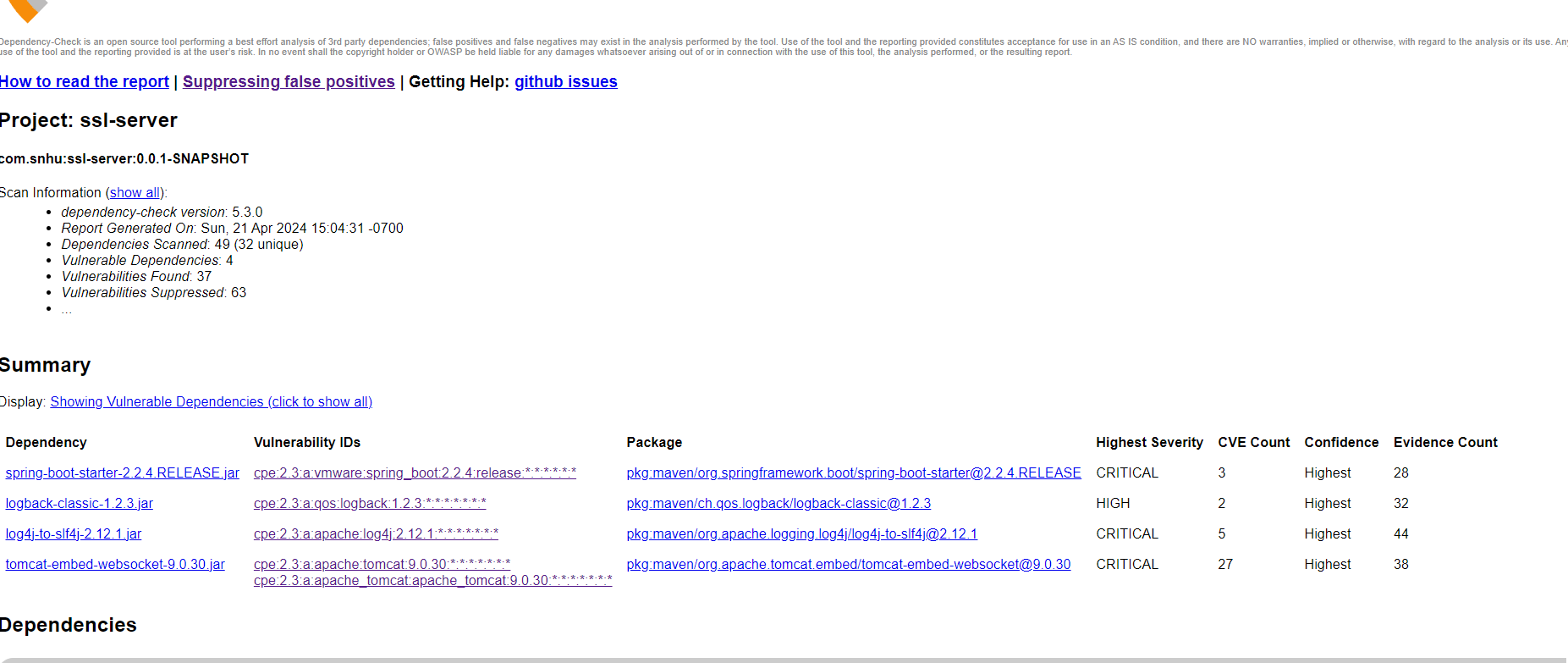
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

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



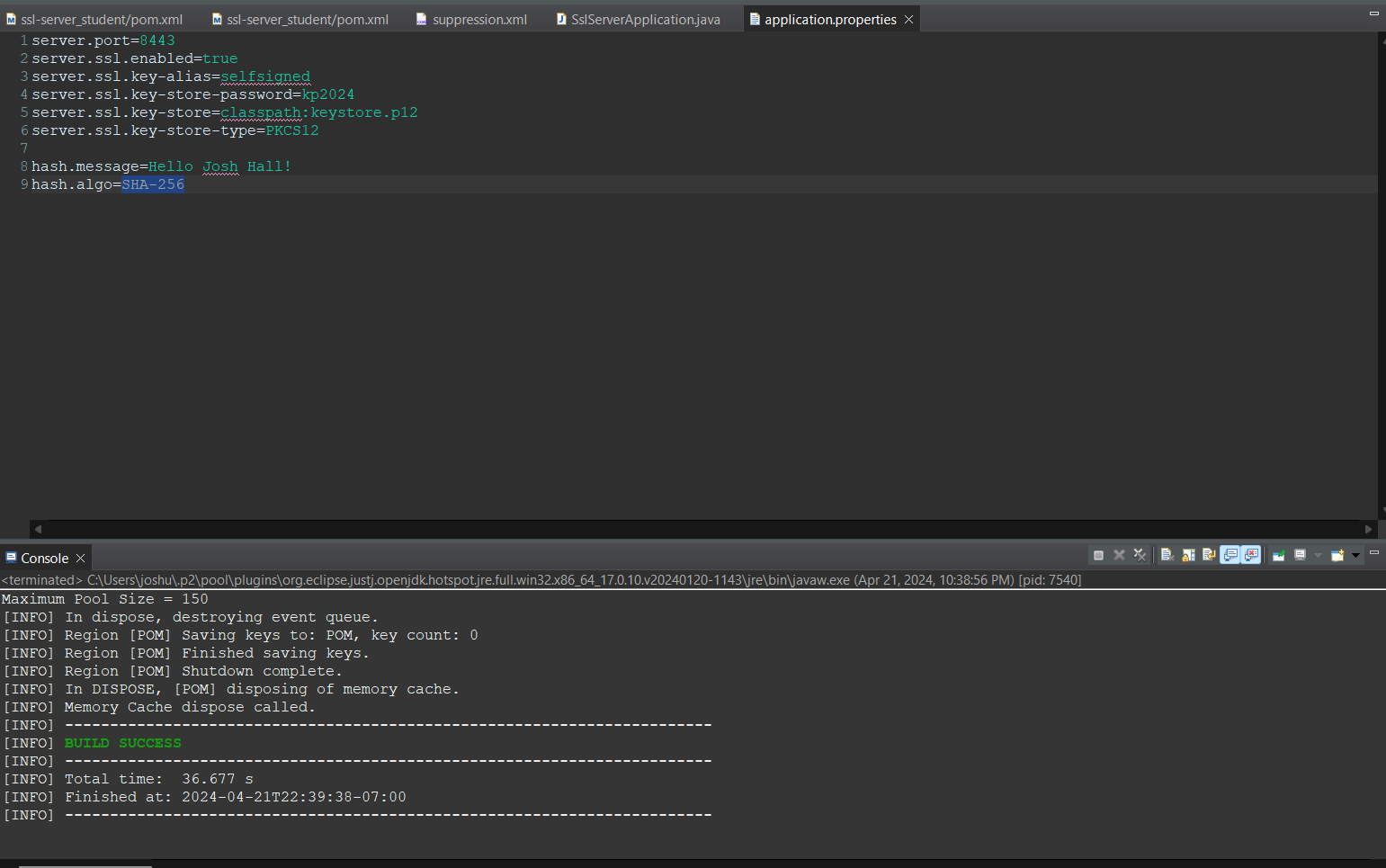


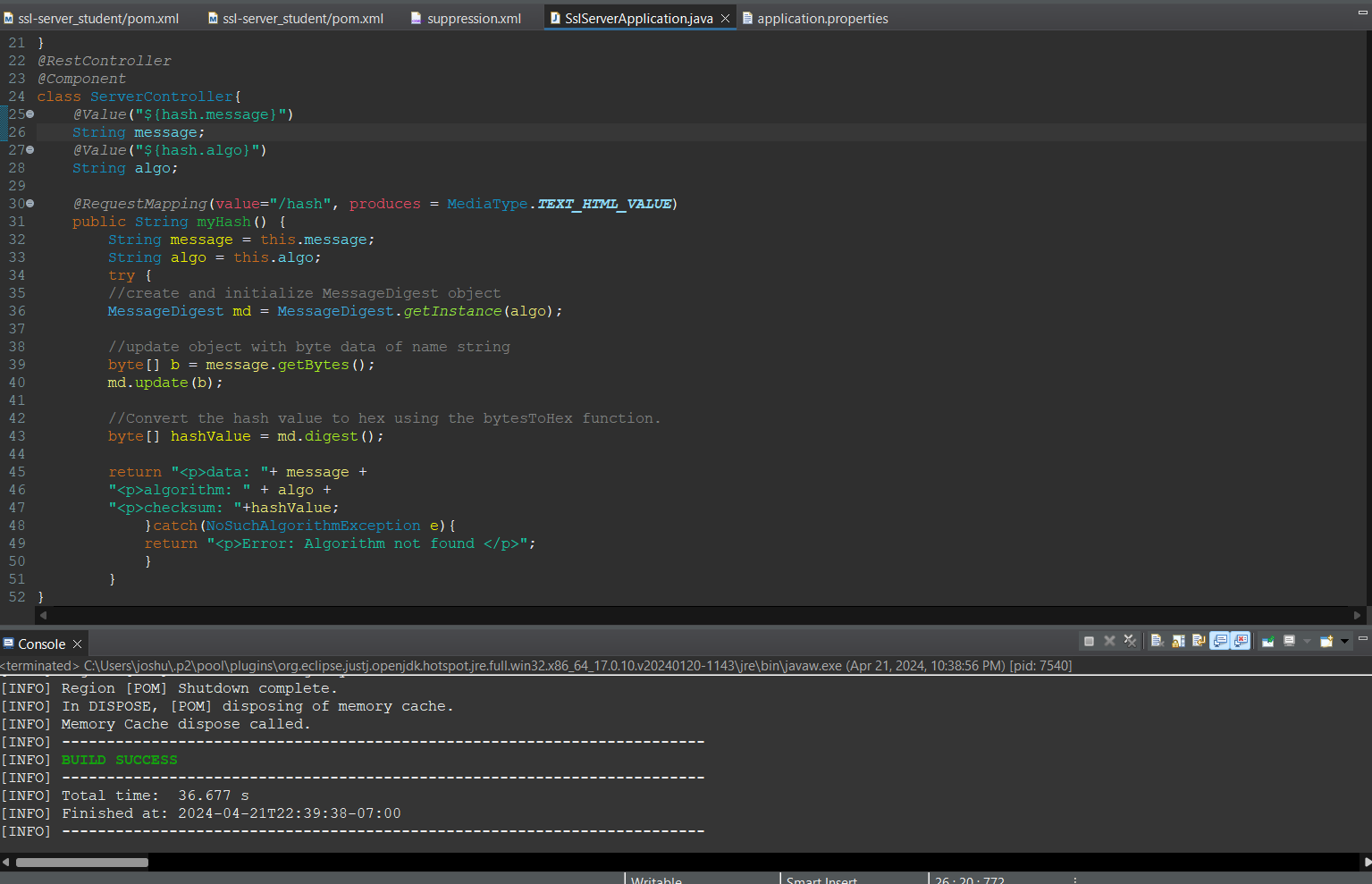
## Functional Testing

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

After analyzing the code, I found three flaws worth addressing. These were:

1. Verifying and protecting any sensitive data, in our case the message and the algorithm type. In a live project with user input, we would also want to implement validation in order to ensure there are no injection vulnerabilities. In this case, I implemented the message and algo variables as @Values that were imported from the [application.properties](http://application.properties) file.
2. The myHash() function did not initially specify a type for the API mapping, so we should specify that it produces text HTML in order to keep this clear for browsers and any other consumers of the end point.
3. We did not initially implement a handling of the potential error produced by a faulty algorithm name. So we wrapped the MessageDigest logic in a try/catch block in order to handle the error.





## Summary

In the Vulnerability Assessment Process Flow Diagram, several key areas of security were addressed through code refactoring to enhance the overall security of the Artemis Financial web application.

– Input Validation: Secure Input and Representations

Input validation is crucial to prevent malicious input that could compromise the application's security. To ensure the data's integrity and protection, I implemented obfuscation techniques for the inputs passed to the myHash function. This approach ensures that the data is properly sanitized and protected against potential injection attacks, safeguarding the application from unauthorized access and data corruption.

–APIs and Cryptograhy

The application's API endpoint is fortified with Transport Layer Security (TLS) to encrypt data transmission between the client and server. Additionally, the communication is further secured using a SHA-256 certificate, ensuring that the data exchanged remains confidential and tamper-proof, mitigating the risks associated with data interception and man-in-the-middle attacks.

–Code Error Handling

The application's resilience is enhanced by ensuring that any errors occurring during runtime are appropriately handled, allowing the program to recover and resume its normal execution flow without crashing. This proactive approach minimizes potential vulnerabilities and enhances the application's stability and reliability.

–Code Quality

Data protection measures, such as type checking and validation, were implemented to ensure the integrity and consistency of the data processed by the application. Best practices of utilizing the Spring Framework also help keep the code well organized and secure.

## Industry Standard Best Practices

In addressing the security vulnerabilities of the Artemis Financial web application, secure coding guidelines, such as input validation and output encoding, were followed to prevent common vulnerabilities like SQL injection. By implementing these practices, the application's codebase was fortified against known security weaknesses. In addition to this hardening of the code base, we performed static analysis using the Maven dependency checking tool which helped us ensure our dependent code didn't prevent any further vulnerabilities.

The value of adhering to industry-standard best practices for secure coding extends beyond mere compliance, it plays a pivotal role in safeguarding the company's overall wellbeing. A secure application not only protects sensitive client data and financial information but also preserves the company's reputation and trustworthiness in the market. By prioritizing security throughout the development lifecycle, Artemis Financial demonstrates its commitment to providing a secure and reliable platform for its clients.