****

# CS 305 Project Two

**Practices for Secure Software Report**

**Table of Contents**

[Document Revision History 3](#_heading=h.1fob9te)

[Client 3](#_heading=h.3znysh7)

[Instructions 3](#_heading=h.2et92p0)

[Developer 4](#_heading=h.tyjcwt)

[1. Algorithm Cipher 4](#_heading=h.3dy6vkm)

[2. Certificate Generation 4](#_heading=h.1t3h5sf)

[3. Deploy Cipher 4](#_heading=h.4d34og8)

[4. Secure Communications 4](#_heading=h.2s8eyo1)

[5. Secondary Testing 4](#_heading=h.17dp8vu)

[6. Functional Testing 5](#_heading=h.3rdcrjn)

[7. Summary 5](#_heading=h.26in1rg)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6/26/22** | **Matthew Muller** | **Full Document Completed** |

## Client



## Instructions

Deliver this completed Practices for Secure Software Report documenting your process for writing secure communications and refactoring code that complies with software security testing protocols.

Respond to the steps outlined below and replace the bracketed text with your findings in your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Matthew Muller

## 1. Algorithm Cipher

Determine an appropriate encryption algorithm cipher to deploy given the security vulnerabilities, justifying your reasoning. Be sure to address the following:

* Provide a brief, high-level overview of the encryption algorithm cipher.
* Discuss the hash functions and bit levels of the cipher.
* Explain the use of random numbers, symmetric vs non-symmetric keys, and so on.
* Describe the history and current state of encryption algorithms.

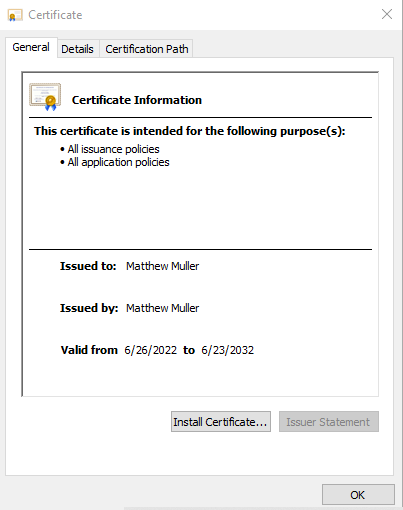
Due to the sensitive nature of the information in Artemis Financial’s long-term archive files, it is extremely important that they be properly encrypted in order to prevent data breaches. Because of this, I believe that the company should deploy the Advanced Encryption Standard (AES) encryption algorithm cipher. AES is a symmetric block cipher developed by the U.S. government that can be implemented using cryptographic keys of 128, 192, or 256 bits. All three of these options will provide the system with security against brute force attacks, as the time required to try all possible key combinations would be too great for even extremely powerful computers. Therefore, it may not be necessary to choose the most secure cipher (the 256-bit) as it will require much more processing power and take longer to execute. Since this is a symmetric cipher, the same key is used for both encrypting and decrypting the data. This is one of the risks of my recommendation because it means that both the sender and the receiver must know the key, which could lead to vulnerabilities if the key is not properly protected. However, this can be guarded against by taking precautions such as implementing multi-factor authentication and deploying firewalls and antimalware software (Bernstein).

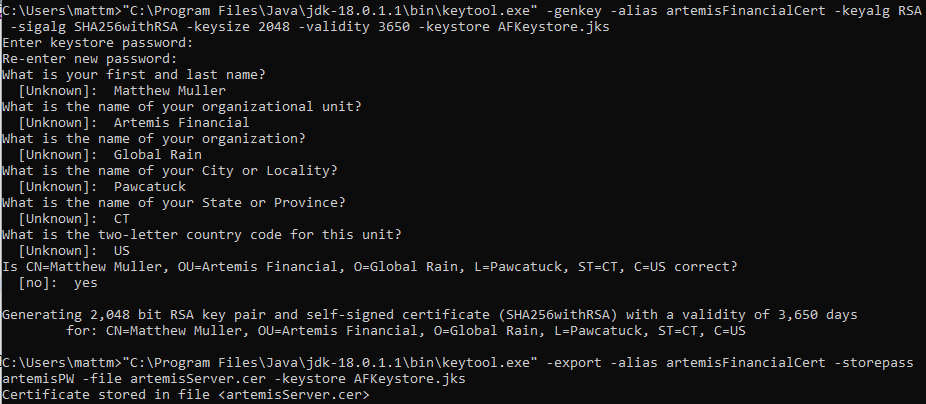
AES uses AES-hash, a secure hash function that takes a string input of an arbitrary amount of bits and returns a fixed bit length string as output. This process ensures that each input will provide a unique output as finding two files which hash to the same value would require on average approximately 2^128 operations (Cohen). AES was developed by the National Institute of Standards and Technology (NIST) starting in 1997. This came after it announced that there was a need for an alternative to the currently used Data Encryption Standard (DES) because it was starting to become vulnerable to brute-force attacks. AES was designed to provide effective encryption well into the 21st century in a manner that is easy to implement in hardware and software. It was created with voluntary free use programs that provide encryption services but any nongovernmental organizations that use AES are subject to limitations created by U.S. export control (Bernstein).

## 2. Certificate Generation

Generate appropriate self-signed certificates using the Java Keytool, which is used through the command line.

* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

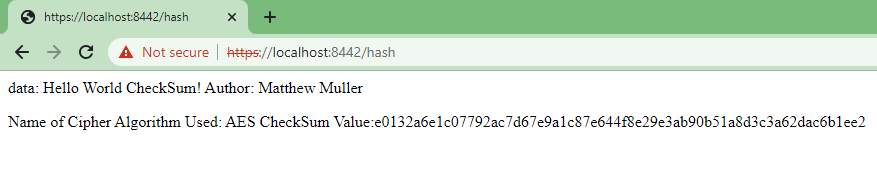




## 3. Deploy Cipher

Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.

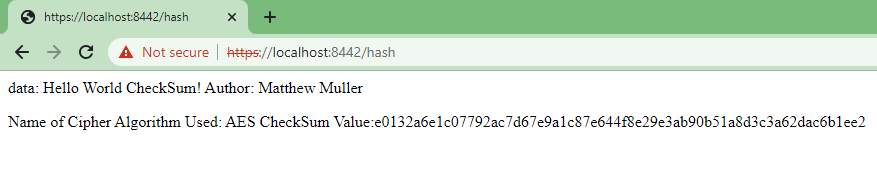
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 4. Secure Communications

Refactor the code to convert HTTP to the HTTPS protocol. Compile and run the refactored code to verify secure communication by typing **https://localhost:8443/hash** in a new browser window to demonstrate that the secure communication works successfully.

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

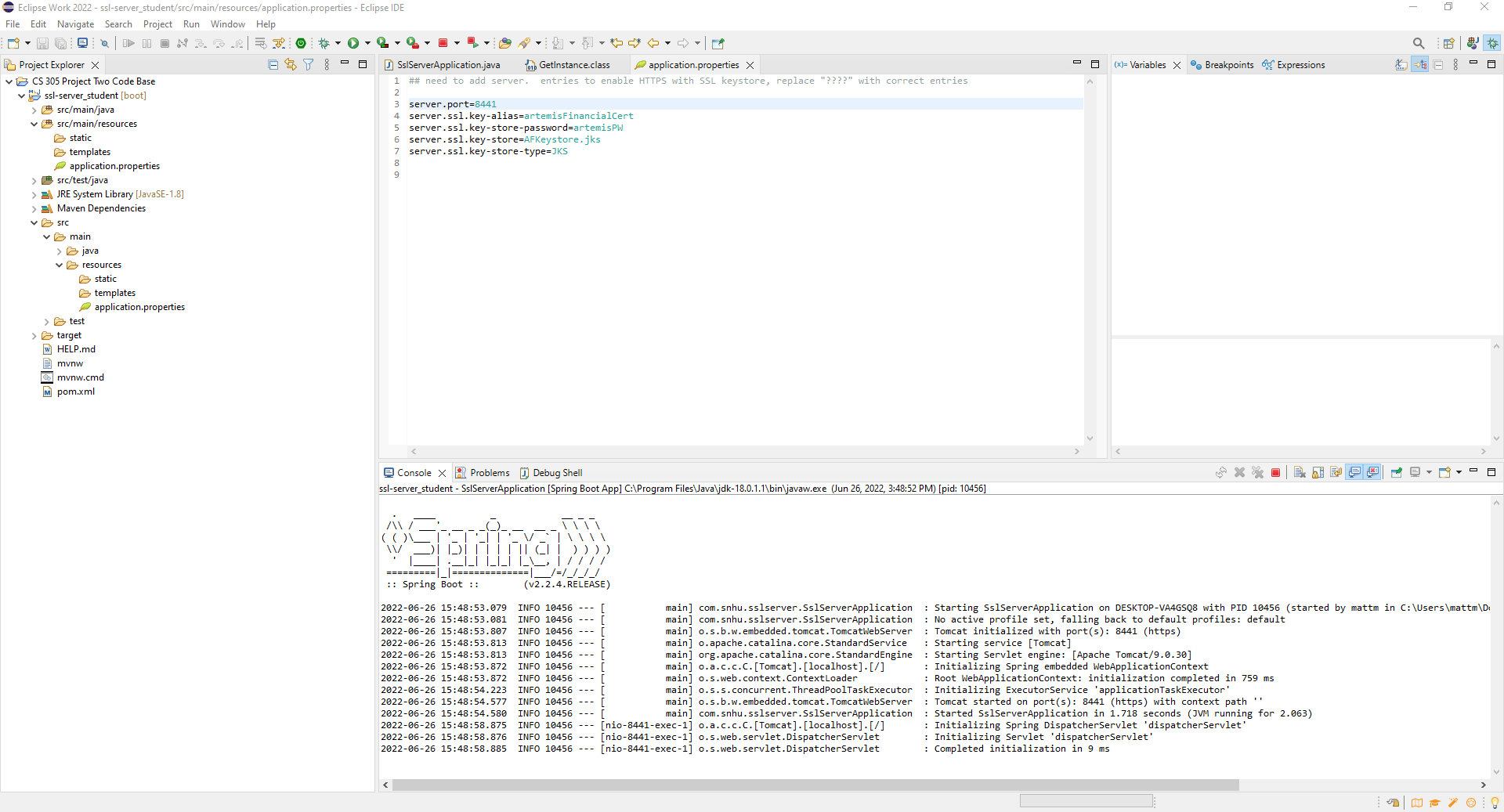


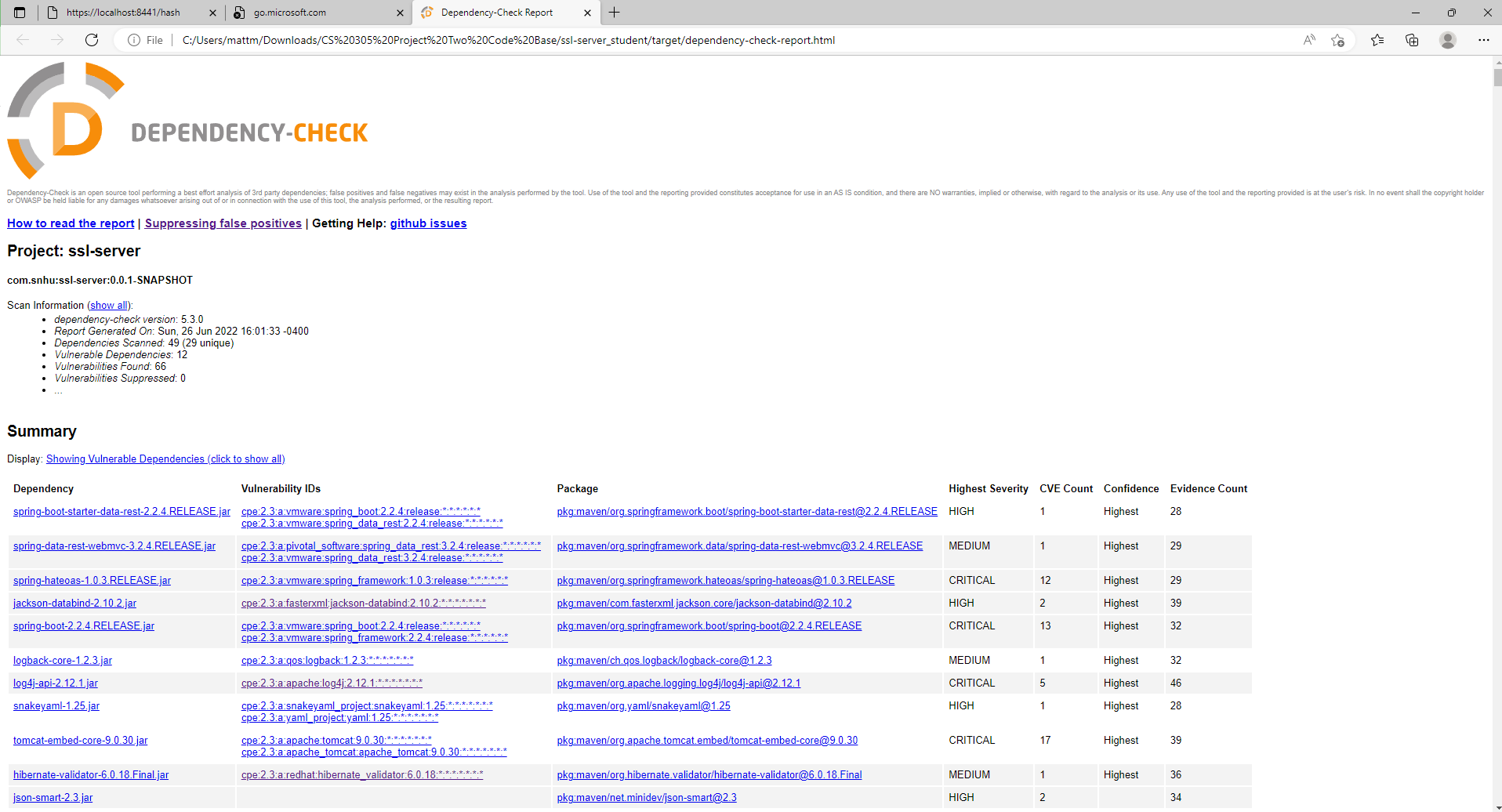
The browser gives a Not secure warning due to the certificate being self-signed.

## 5. Secondary Testing

Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

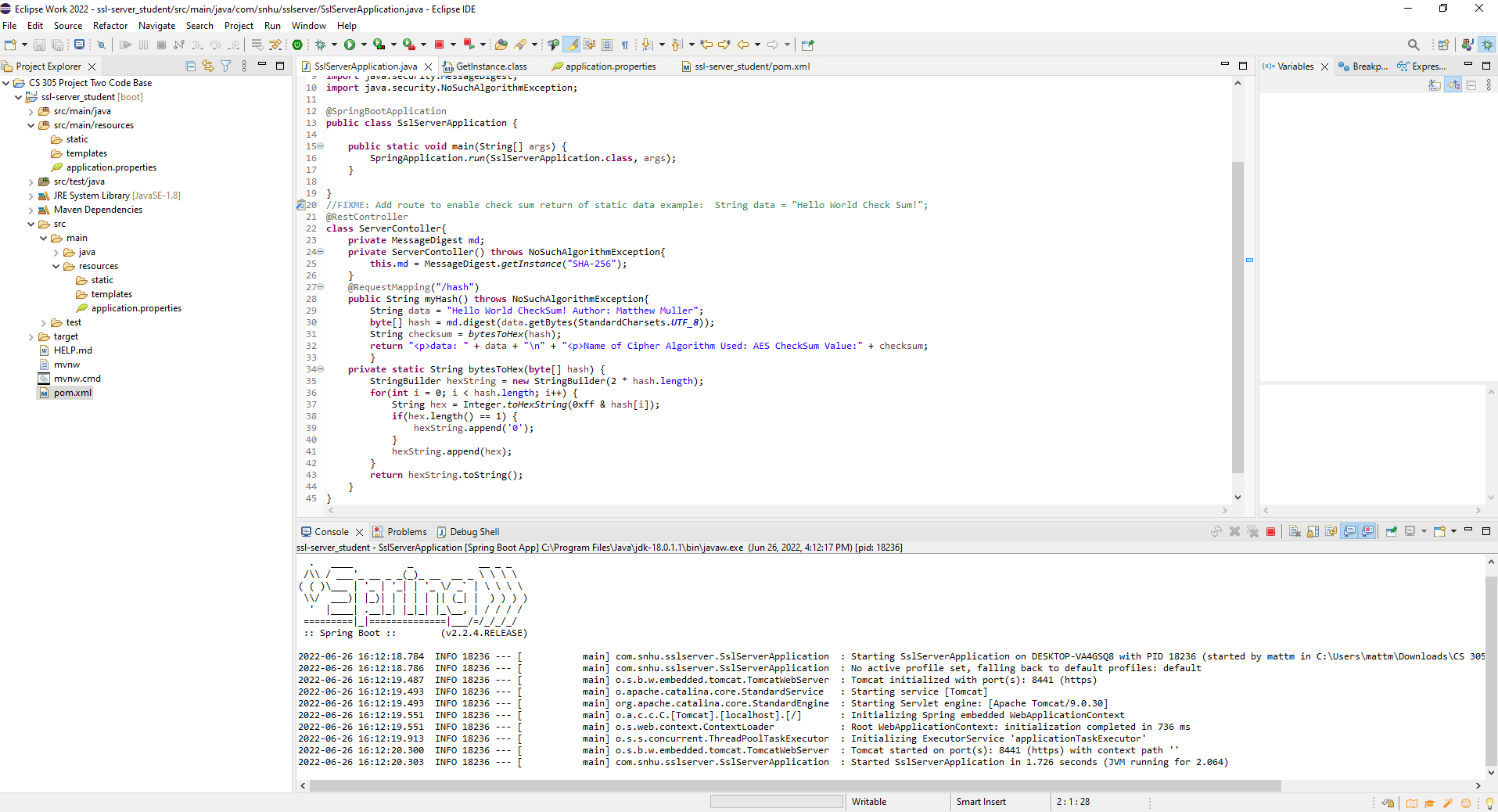




## 6. Functional Testing

Identify syntactical, logical, and security vulnerabilities for the software application by manually reviewing code.

* Complete this functional testing and include a screenshot below of the refactored code executed without errors.



## 7. Summary

Discuss how the code has been refactored and how it complies with security testing protocols. Be sure to address the following:

* Refer to the Vulnerability Assessment Process Flow Diagram and highlight the areas of security that you addressed by refactoring the code.
* Discuss your process for adding layers of security to the software application and the value that security adds to the company’s overall wellbeing.
* Point out best practices for maintaining the current security of the software application to your customer.

One area of security that I addressed by refactoring the code was Cryptography. I did this through the implementation of the encryption algorithm cipher. Incoming data will now be hashed to a unique 256-bit string and will only be able to be accessed by using a private key. Another area of security that I addressed by refactoring the code was APIs. I did this through the use of the Spring Boot API and the implementation of HTTPS. I also addressed code error and code quality by utilizing the NoSuchAlgorithm exception and carefully reviewing my code for functionality and readability.

The first big step in adding layers of security to this software application was generating a self-signed certificate that allowed the program to utilize HTTPS. The next step was to implement the hash function and verify that it was functioning properly by generating a checksum. Finally, I checked to make sure that the changes that I made to the system did not introduce any new vulnerabilities through a dependency check. All of these things add to the company’s overall well being as they greatly improve their level of security and greatly reduce their risk for financial loss through cyber attacks.

The most important thing for Artemis financial to keep in mind to ensure the continued security of this system is to keep all of the software up to date. Eventually new vulnerabilities will be found in the software that is currently being utilized by the system so updating consistently is essential.

**Resources**

Bernstein, C., &amp; Cobb, M. (2021, September 24). Advanced Encryption Standard (AES).

TechTarget. Retrieved June 5, 2022, from https://www.techtarget.com/searchsecurity/definition/Advanced-Encryption-Standard

Cohen, B. (n.d.). AES-hash - csrc.nist.rip. Retrieved June 6, 2022, from

https://csrc.nist.rip/groups/ST/toolkit/BCM/documents/proposedmodes/aes-hash/aeshash.pdf