****

# CS 305 Project Two

**Practices for Secure Software Report**

Table of Contents

[Document Revision History 3](#_Toc33111302)

[Client 3](#_Toc33111303)

[Instructions 3](#_Toc33111304)

[Developer 4](#_Toc33111305)

[1. Algorithm Cipher 4](#_Toc33111306)

[2. Certificate Generation 4](#_Toc33111307)

[3. Deploy Cipher 4](#_Toc33111308)

[4. Secure Communications 4](#_Toc33111309)

[5. Secondary Testing 4](#_Toc33111310)

[6. Functional Testing 5](#_Toc33111311)

[7. Summary 5](#_Toc33111312)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **03/06/2022** | **Amy E Vazquez** |  |

## 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

Amy E Vazquezz

## 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.

[**This what I can conclude with what I was given., I have confidence in the most appropriate encryption algorithm cipher is the Advanced Encryption Standard (AES), precisely the AES-256. AES is the present encryption standard used by the United States Government since 2001, after it was replaced the Data Encryption Standard (DES). AES is frequently used within both the public and private sectors and is considered the “gold standard” of encrypting data (Crawford).**

**The Secure Hashing Algorithms (SHA) are common hash functions. SHA can be used in conjunction with AES to guarantee security, as SHA takes data input as a plaintext and creates a cipher text (Crane) that is doubtful to be deciphered. The bit level in security notifying us that an attacker would essentially need to perform 2n operations (In being the bit level) to crack a cipher of that bit level. AES utilizes either 128, 192 or 256 bits, with AES-256 being the most secure of them.**

**AES is a symmetric key encryption cipher (Crawford), that means that it uses the same key for both encryption and decryption. Symmetric encryption is seen efficient and easier to use and implement than asymmetric, which uses different keys for encryption and decryption. While AES encrypts every data block the same way, it utilizes longer key lengths, ensuring security. Keys need to be random (Martin), as a key that is easy for one user to remember is also easier to crack for an attacker.**

**Computer-based encryption has been around since at least the 1970’s, with the creation and adoption of the DES. Encryption has been around for periods, as Julius Caesar applied a substitution cipher in about 60 B.C.E., but encryption algorithms have progressed over time, from the Playfair Cipher in 1854 by Charles Wheatstone and the Enigma machine in 1918 by Arthur Scherbius to the very standard of encryption algorithms we have today. We have spent years and progressing security**

## 2. Certificate Generation

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

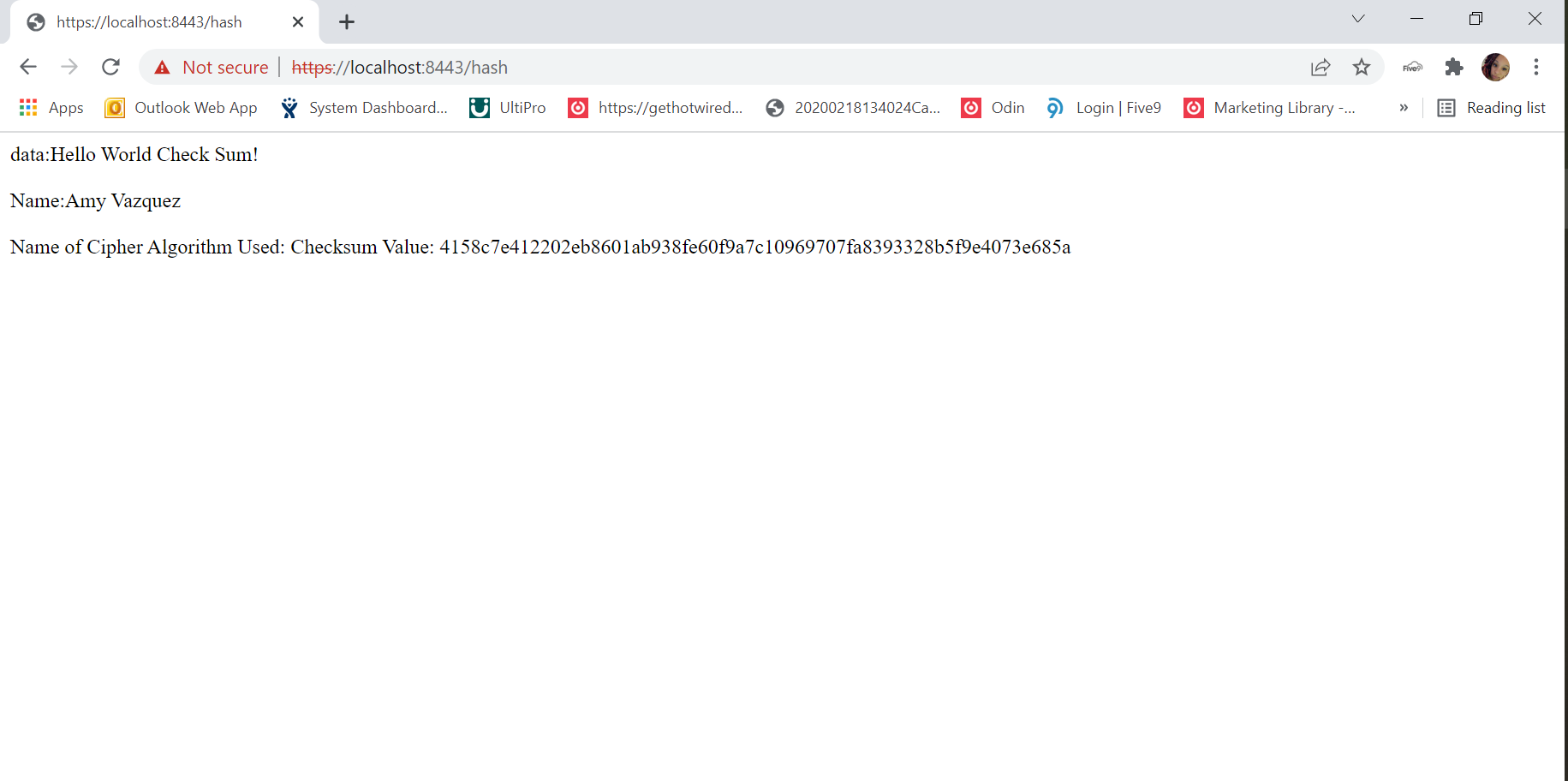
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.

[Insert screenshot(s) here.]

## 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.

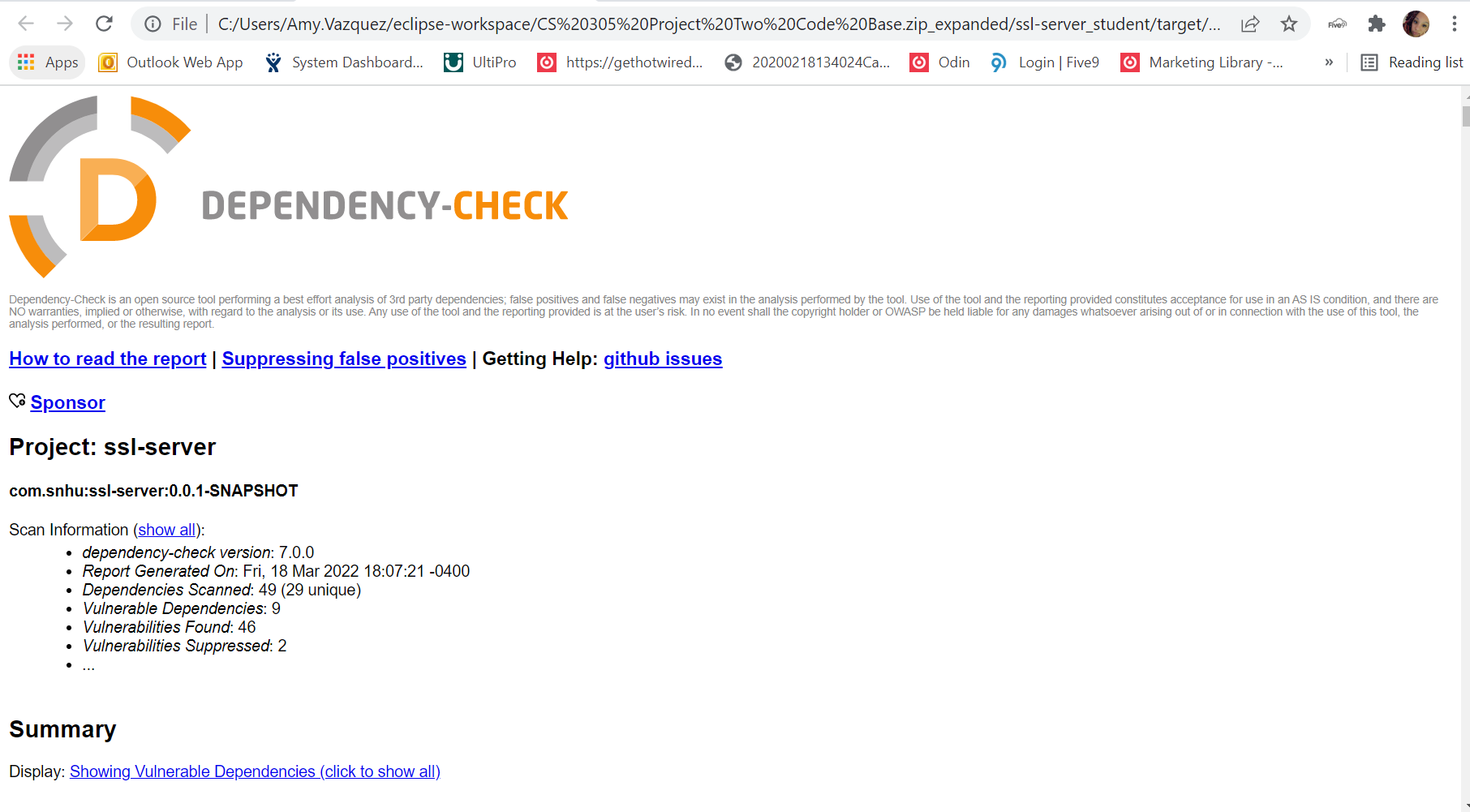
* Insert a screenshot below of the web browser that shows a secure webpage.

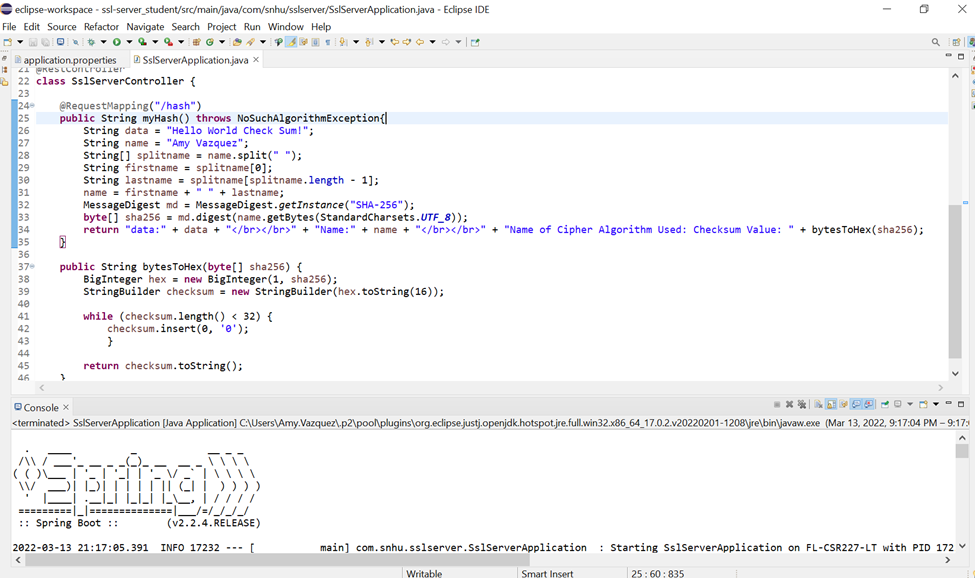
[Insert screenshot(s) here.]

## 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

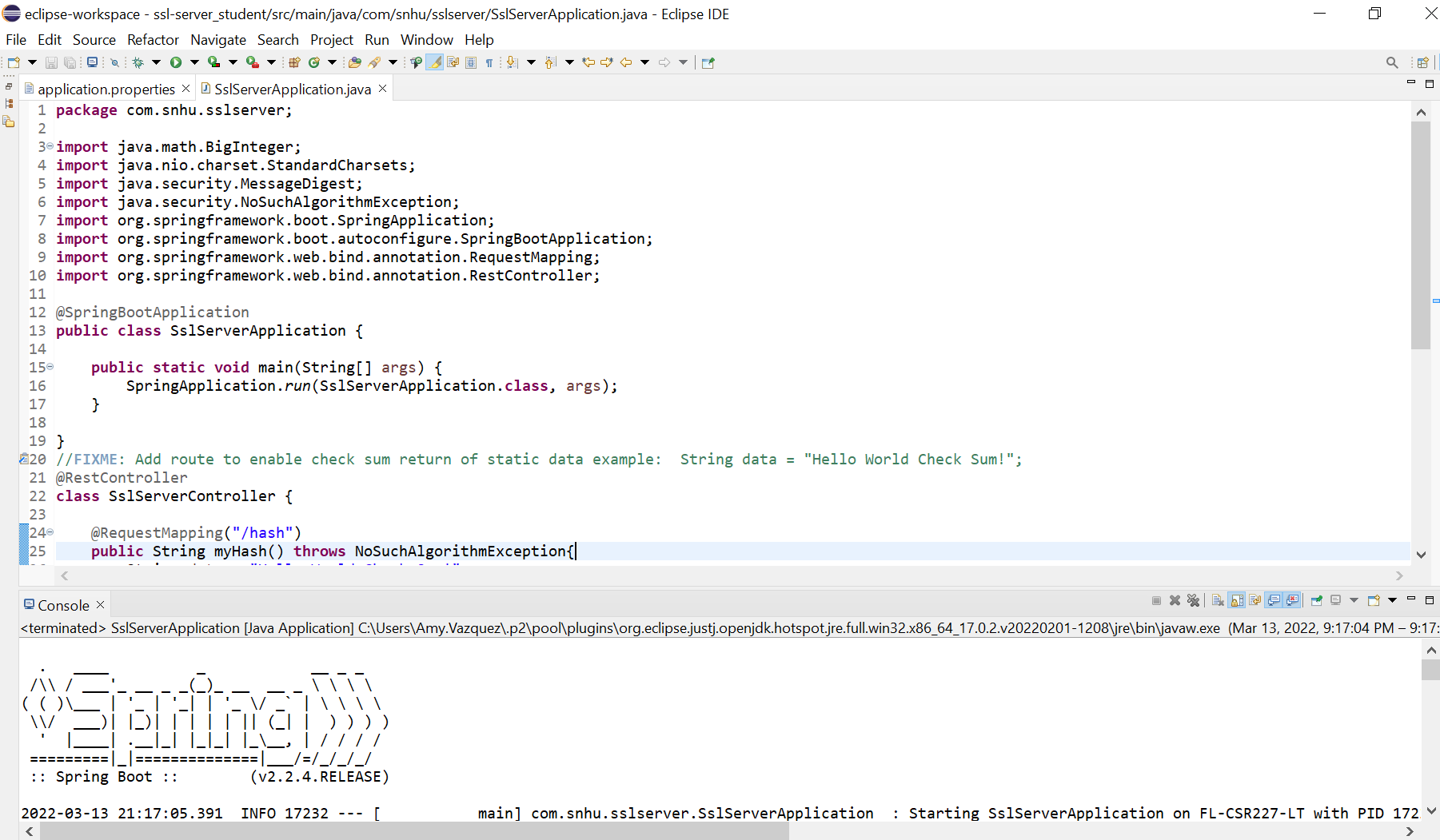


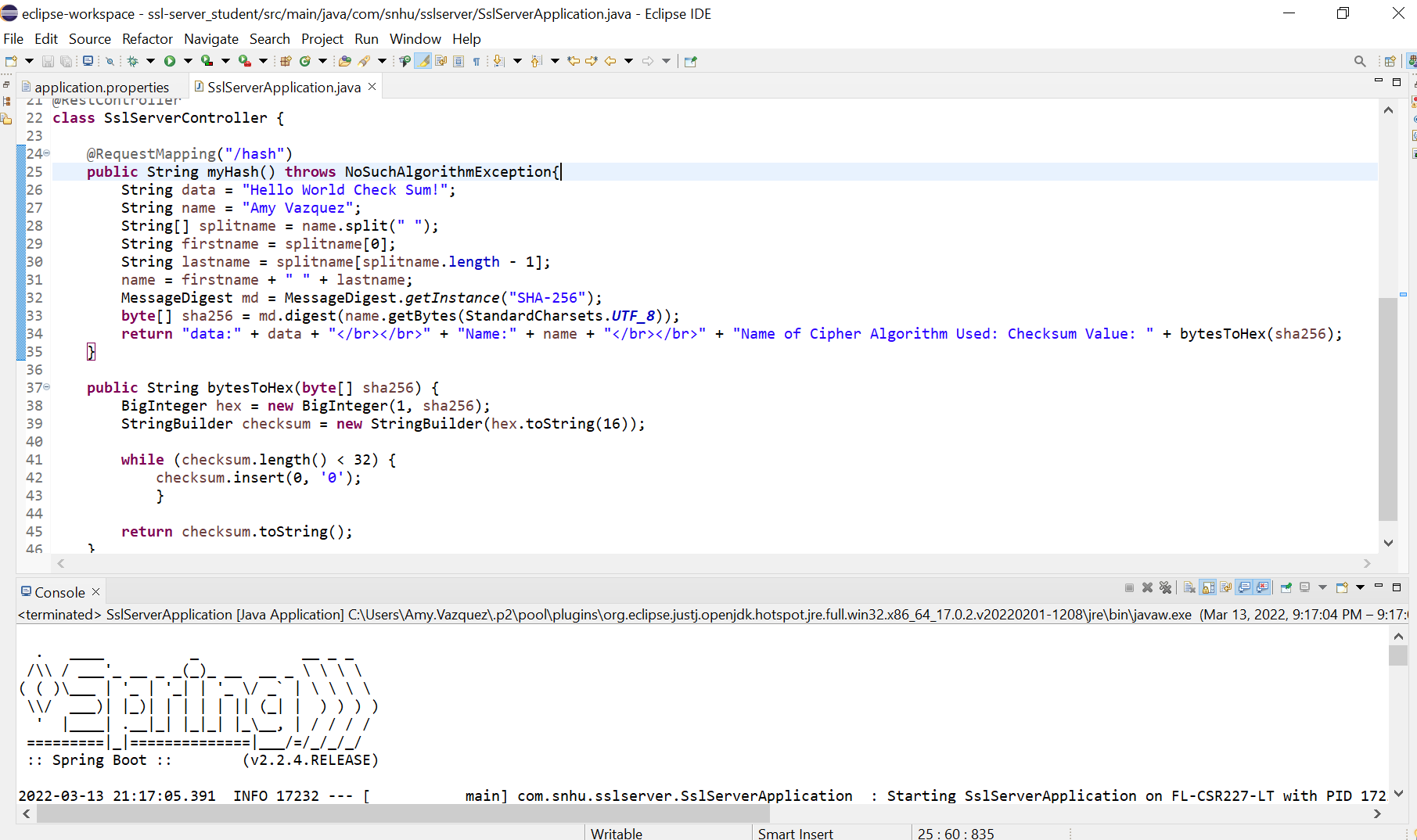


## 6. Functional Testing

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.

**The main security that was added to our application was self-signed certificates that allowed for HTTPS to be used. We also refactored the pom.xml file to ensure that all vulnerabilities that were discovered within the dependency check were resolved.**

**The first step in my process was ensuring that the certificates were made correctly so that we would be able to utilize HTTPS once our application was up and running. This security adds to our company’s well-being by ensuring that our webpage is secure, and users can be assured that they are dealing with us and not an imposter.**

**The next step was making sure that our hashing function worked properly and verifying this with the checksum. This security helps our company’s well-being by letting us be well assured that our users’ datais being hashed properly and not easily retrievable.**

**The final step was ensuring that all vulnerabilities were patched up. Having that security ensures that we as a company have our bases covered and we can be certain that all our application’s inner workings are up to date and working as intended.**

**One best practice for keeping our application’s security is strengthening our software and systems to guarantee everything is updated. This confirms that attackers cannot exploit outdated systems. Enforcing minimum privilege is also critical. While it is not in place with the current state of our application, guaranteeing users only have the access that they need rather than giving everybody access to everything shields the organization from attacks within the group.**