****

# 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** | **12/11/2021** | **Ashley Santo** | **Updated the necessary portions of the document** |

## 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

Ashley Santo

## 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.

Artemis Financial is a company that handles sensitive client data that must be secured throughout all communications or data transfer. It’s crucial that the company use encryption to protect that data from possible third-party attackers who would benefit from it.

Encrypting information is nothing new and some form of encryption has been around for centuries. That being said, electronic cryptography is relatively new and was really only used for military purposes to protect information. It wasn’t until the 1970’s where the United States government put out a request for a block cipher to be the national standard. From this, the Data Encryption Standard (DES) was accepted and used widely. The issue was it had a small size encryption key. With computer power increasing, it was becoming easy to perform brute force attacks to obtain the key and this system was no longer viable for protecting data.

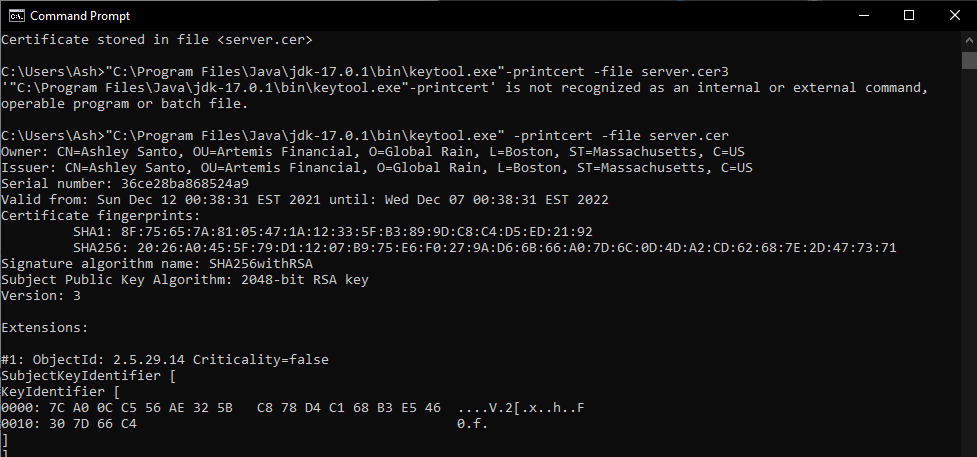
This leads to my recommendation of the Advanced Encryption System, or AES, as an encryption algorithm cipher. This is one of the most trusted algorithms that was implemented as a way to replace DES by the United States government for the past 20 years. It’s a symmetric type of encryption that uses the same key for both encrypting and decrypting data. It includes three block ciphers which are 128, 192 and 256 bits. It’s extremely efficient as a 128-bit cipher but can be used for 192- and 256-bit ciphers when needed for more heavy-duty protection. It also uses a substitution permutation network (SPN) algorithm which applies multiple rounds to encrypt data. The encryption process includes converting data into blocks of bits where columns and rows are shifted around and then a key gets applied. Then it repeats this process multiple times. For most places that would use the 128-bit key, it goes through 10 rounds but the 256-bit key goes through 14 rounds. This is what makes the system so hard to break is that attackers need to get through all these rounds in order to get access.

This can be combined with the SHA-256 hashing algorithm which is a256-bit key that has proven to be much more secure than some other hashing algorithms offered. What makes it so secure is that there are 2256 possible hash values. This makes it near impossible to figure out the initial data from the hash value itself. This also lowers the chances of a collision happening where two messages would share the same hash value. Lastly, is the avalanche effect where even the slightest change to the data completely changes the hash value adding more protection.

## 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.

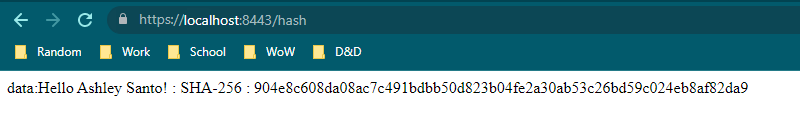
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.

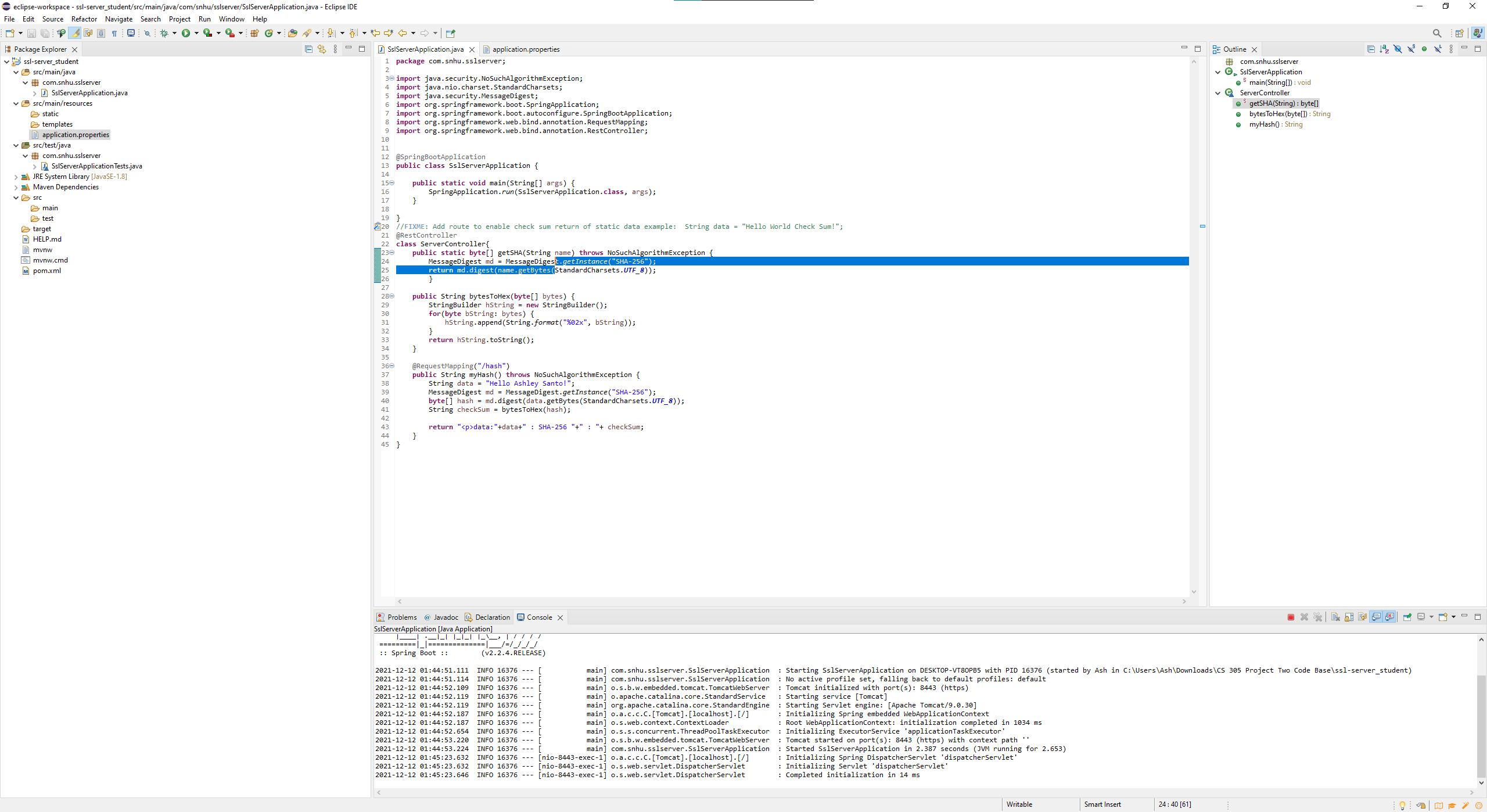
* Insert a screenshot below of the web browser that shows a secure webpage.

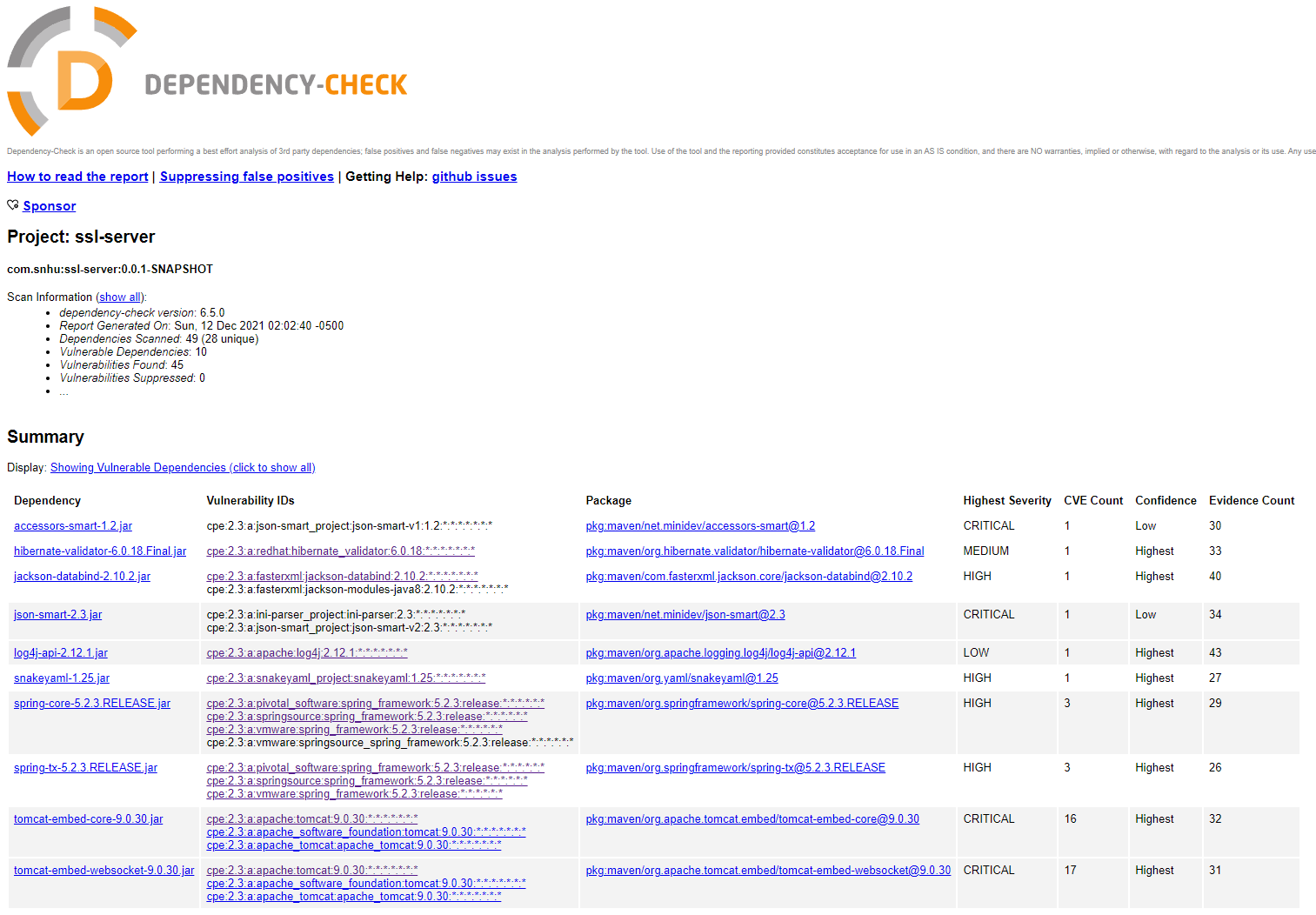


## 5. Secondary Testing

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



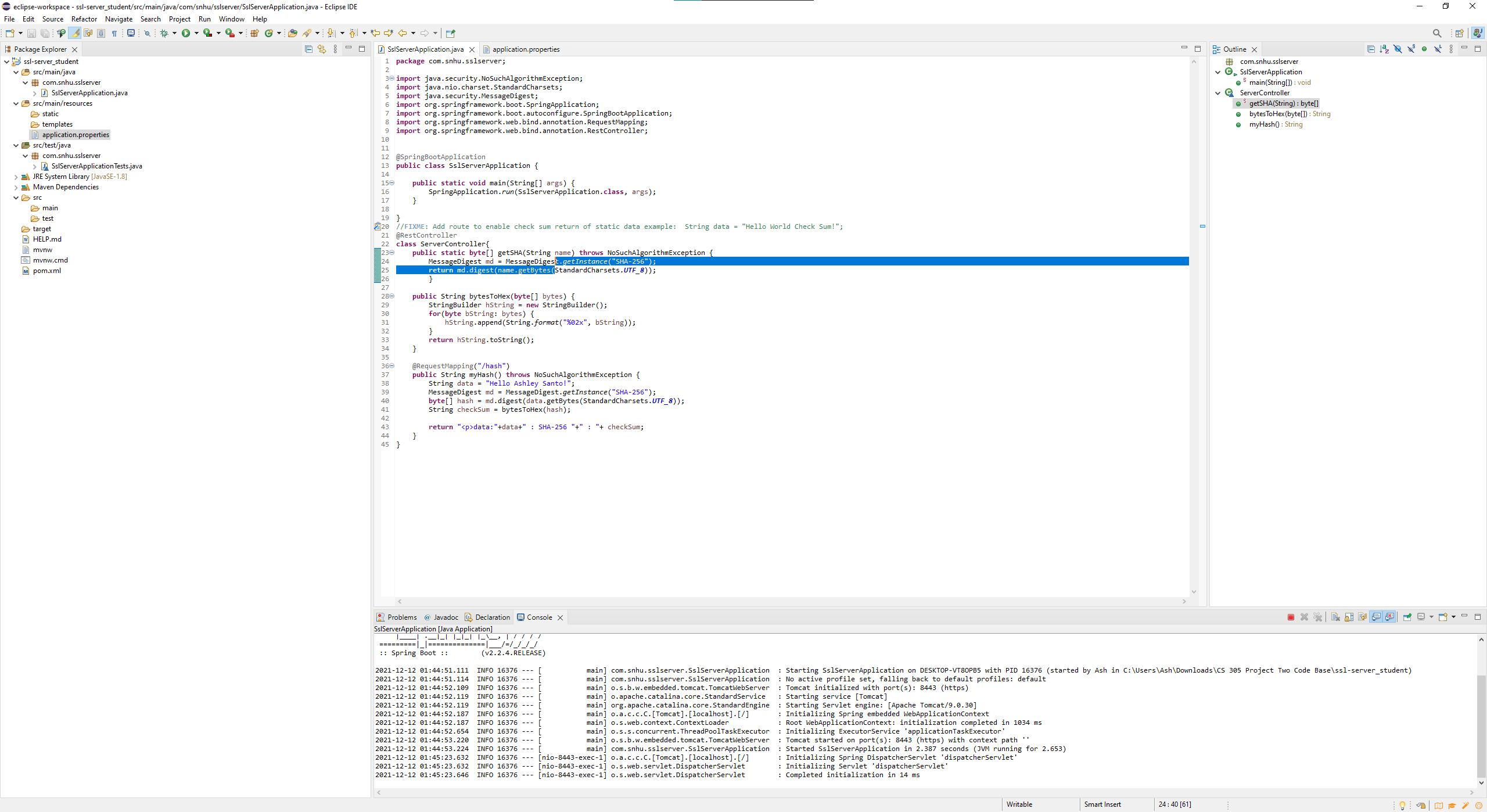


## 6. Functional Testing

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.

Vulnerabilities could include that the application.properties file contains a clear password.



## 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.

There are multiple areas of security that I addressed in my refactored code. The first one was APIs where I created a certificate that allowed the program to connect to a web browser through a secure HTTPS connection. This means that a user could use the web browser interface to interact with the program. Cryptography was also implemented with the hash function that was refactored into the program. This could be seen with the Checksum Verification. I think client/server was addressed as well where the client in this case would be the web browser connecting to the server via the URL link. Code quality was verified to make sure there weren’t any vulnerabilities created in the refactored code.

My process for adding layers of security to the application started with creating a self-signed certificate that allowed for the program to connect securely to the web browser through HTTPS. This benefits both the client and the company as it will provide protections to the company by verifying secure connections and the client can be sure that it is the correct, trusted site and not a fake site. The next step was adding the hashing function to the code base. This adds to the company’s security in that all customer data is being hashed correctly, protecting it from being easily deciphered by third party attackers. Lastly, running the dependency check verified that there were no new vulnerabilities created within the refactored code and that if there were any, it would be handled to make sure they were removed.

The best practices to follow to maintain the current security of the software application is to make sure that all the systems and software is up to date. This is crucial because any issues or vulnerabilities that are found in previous versions would be addressed in current releases. By not updating, the company would be leaving known holes in security for attackers to take advantage of. Another would be to make sure all employees are prepared and know what to do when suspicious emails or messages come through. Phishing emails are always going to be attempted and it’s best that employees know what to look for before clicking anything that they shouldn’t. Lastly, I would recommend only giving permissions out where they are needed. So, depending on the level of access an employee needs, limit what they have permission to have access to. This will ensure systems are staying secure.