****

# 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** | **4-14-2021** | **Noah Pohl** |  |

## 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

Noah Pohl

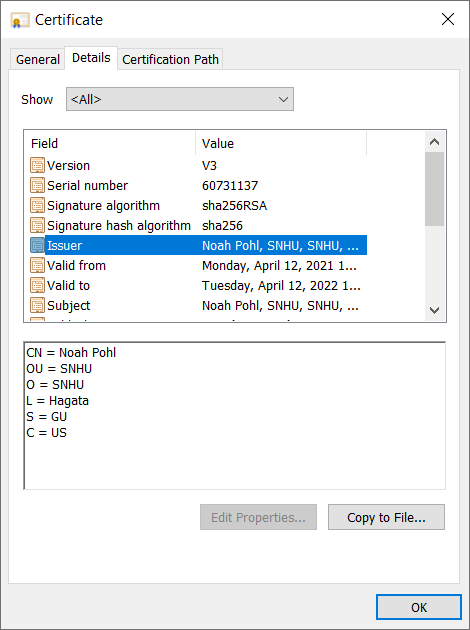
## 1. Algorithm Cipher

* After reviewing the security requirements desired by Artemis Financial, I have determined the best encryption cipher for this situation is the SHA-256 algorithm. This secure hash algorithm is ideal as it is providing a high level of security and will allow us to add file verification through the generated checksum. The SHA-256 algorithm is also resistant to collision attacks and is the new standard for web applications that are using some form of SSL or transport layer security (TLS) certificate.
* The SHA algorithm cipher uses a symmetric hash function which will typically take a piece of data or text and then convert it to a unique hashed string of text which can be checked. This means that SSL certificates can use the SHA hashing algorithm for signing these digital signatures as the hashed value will always be unique. If the SHA-256 algorithm did not produce these unique hashed values, then it would be possible for two different text values to have the same hash which would result in a collision. By using SHA-256, we are able to prevent these collisions from happening and significantly increase the level of security for any application that uses this secure algorithm.
* The SHA algorithm was developed and released by the National Security Agency (NSA) in 2001 as a cryptographic hash function and was a significant upgrade compared to older forms of hashing algorithms such as MD5 which is only 128-bits and has become increasingly less secure over time as large-scale vulnerabilities have been exposed. SHA-256 has become the cryptographic standard in recent years and even in the crypto sphere where the algorithm is used in various blockchain technologies including the generation of Bitcoin addresses.

## 2. Certificate Generation

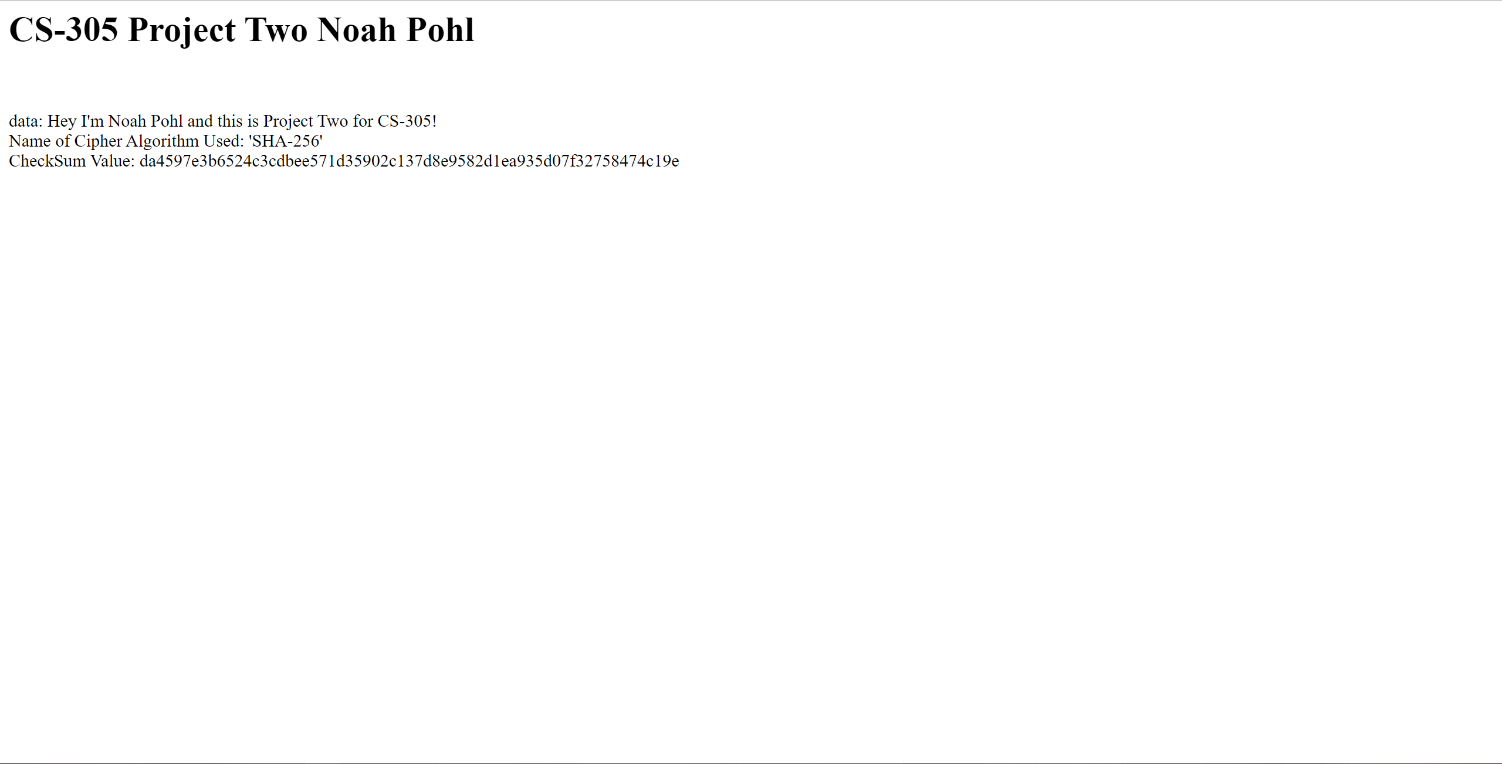
Generate appropriate self-signed certificates using the Java Key tool, which is used through the command line.

* After generating the keys using the Java Key tool and exporting the certificate file, we are able to view the issuer details which are self-signed and the sha256 signature hash algorithm used.



## 3. Deploy Cipher

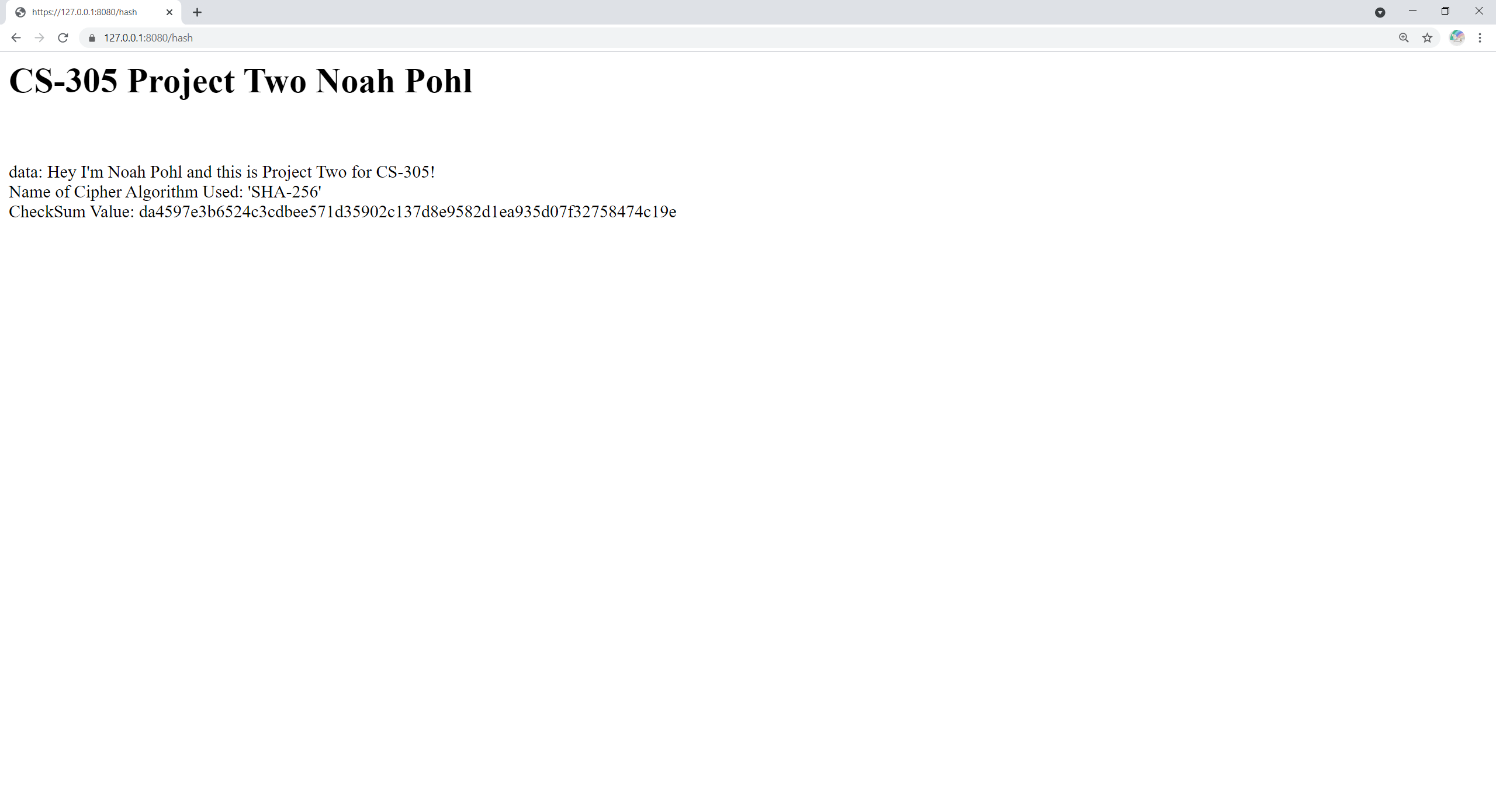
Refactor the code and use security libraries to deploy and implement the encryption algorithm cipher to the software application. Verify this additional functionality with a checksum.



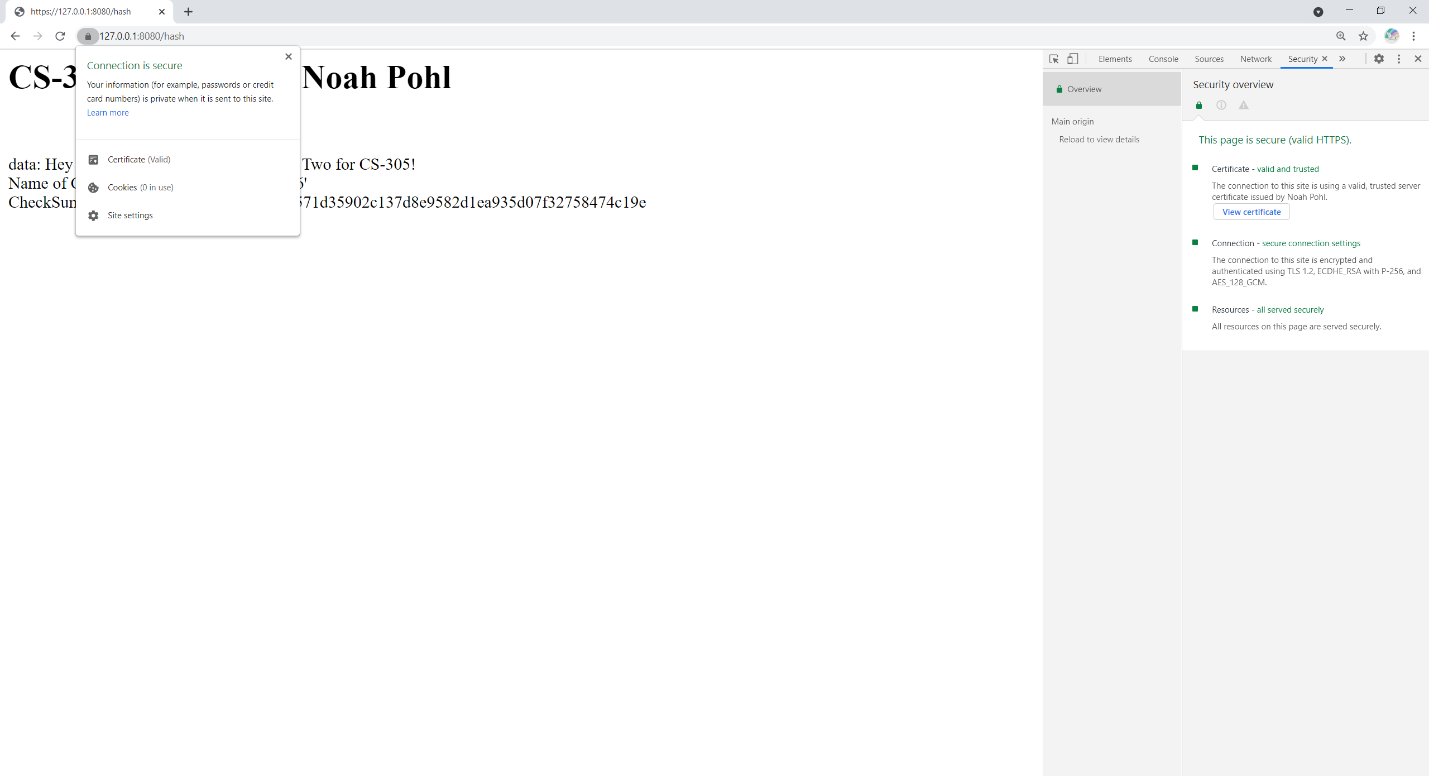
* Screenshot of checksum verification running with deployed encryption cipher algorithm.

## 4. Secure Communications

* Now accessing our local web server by typing **https://127.0.0.1:8080/hash** we can see the HTTPS secure communication protocol is working properly.



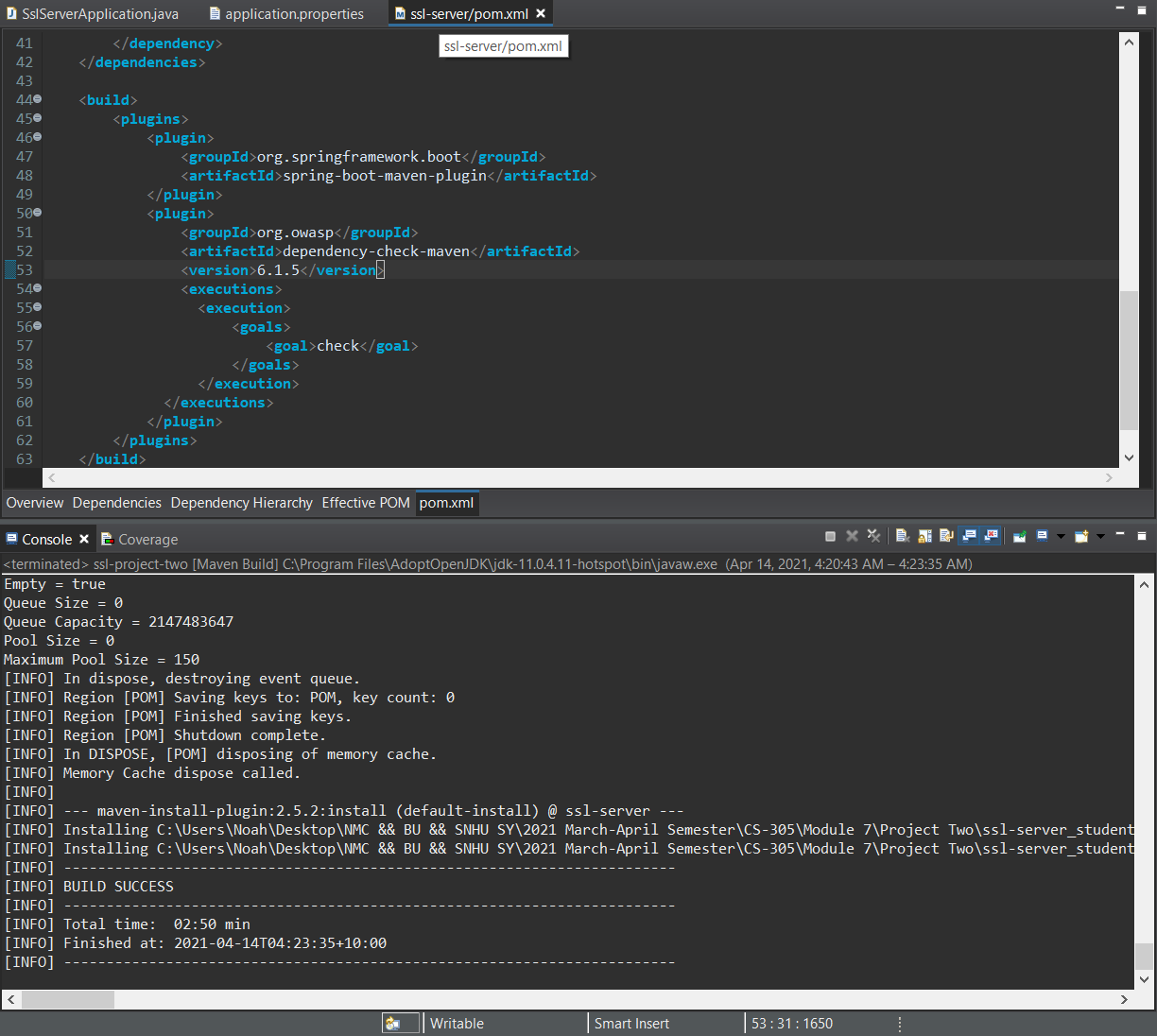
* Now opening up the Chrome developer tools window and then selecting the security tab reveals this page is using a valid and trusted certificate with HTTPS encryption.



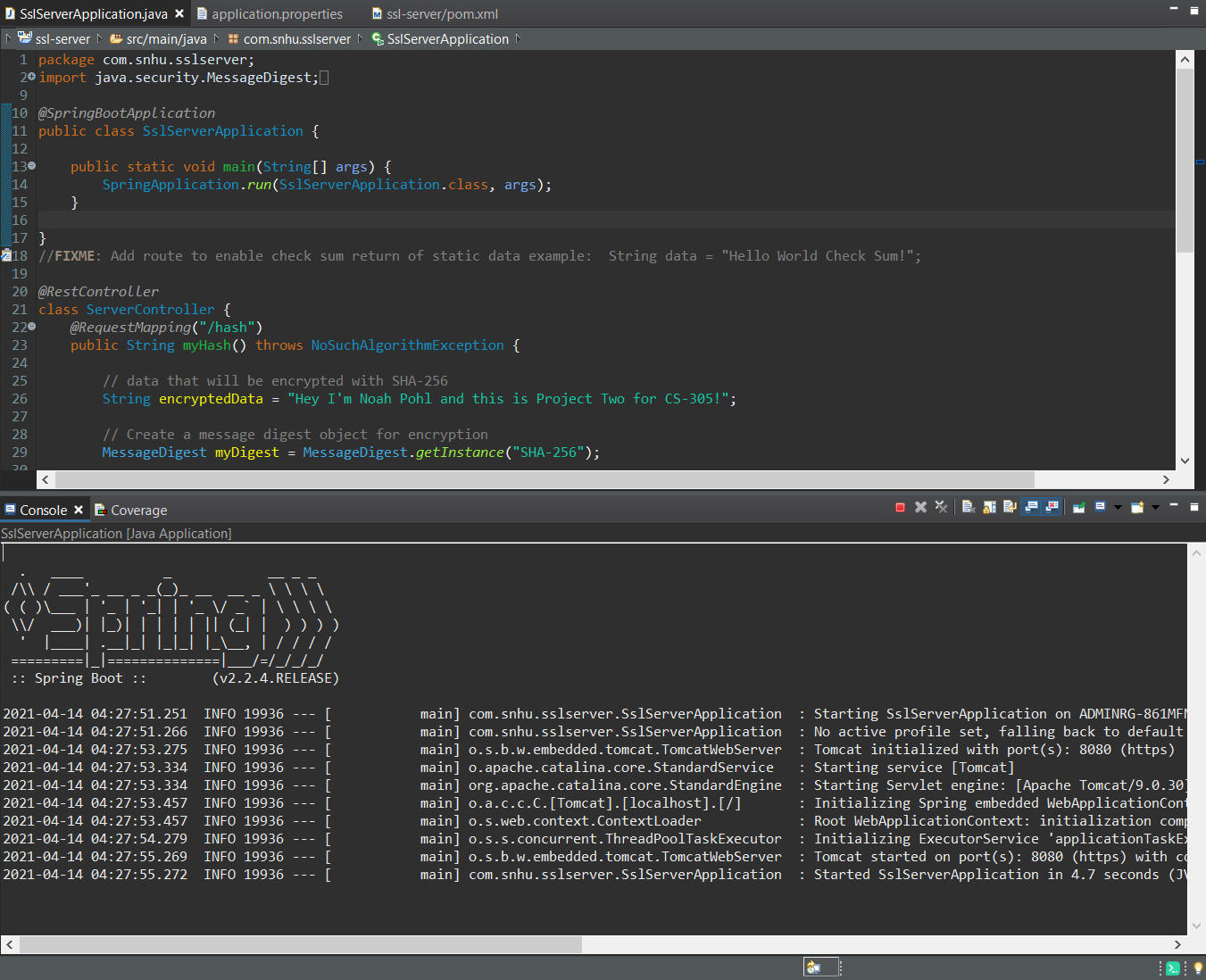
## 5. Secondary Testing

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.

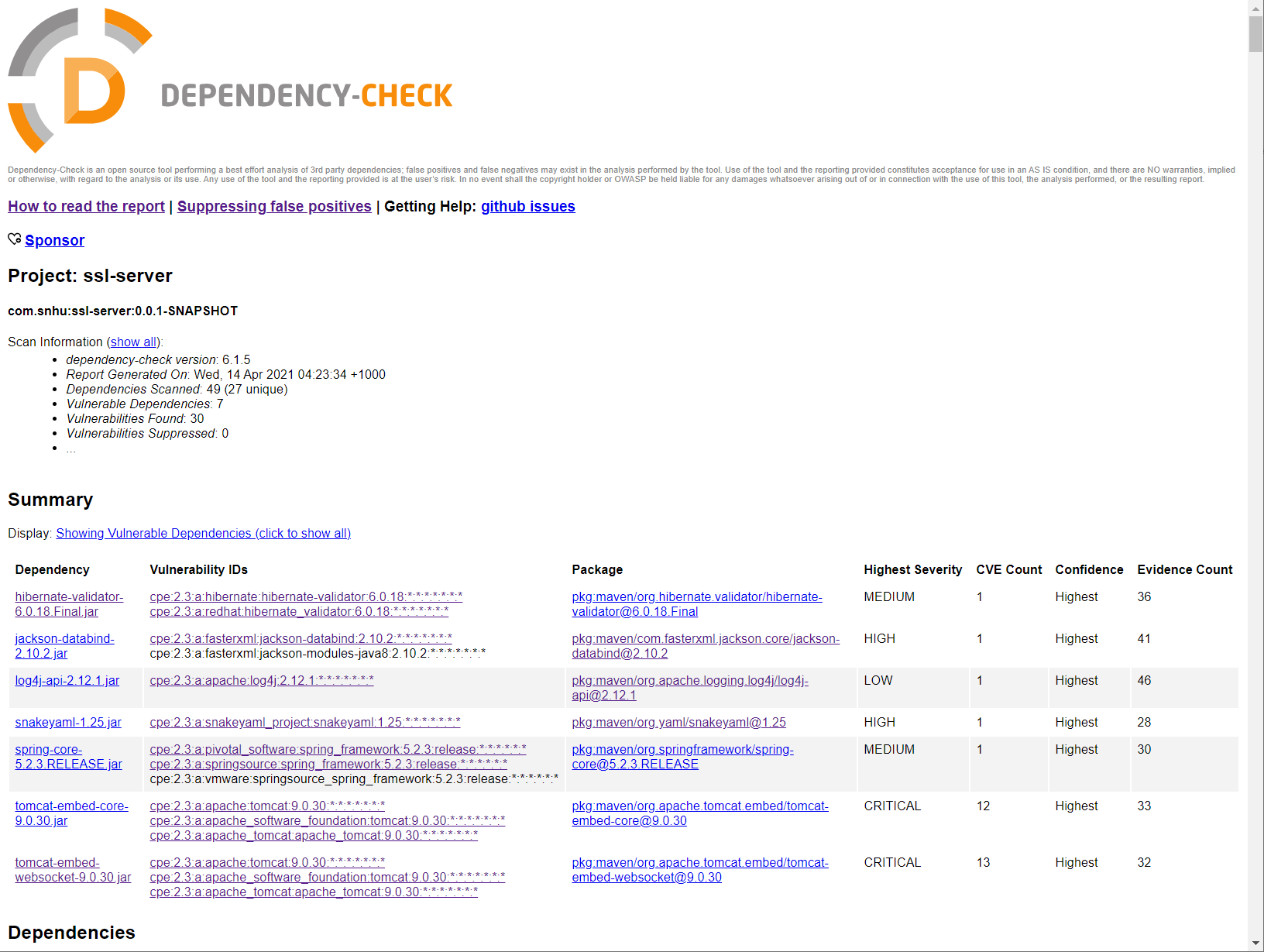
• Screenshot of the Maven dependency check completed successfully without errors



* Screenshot of the SSL server running with the refactored code as a Spring boot web server.



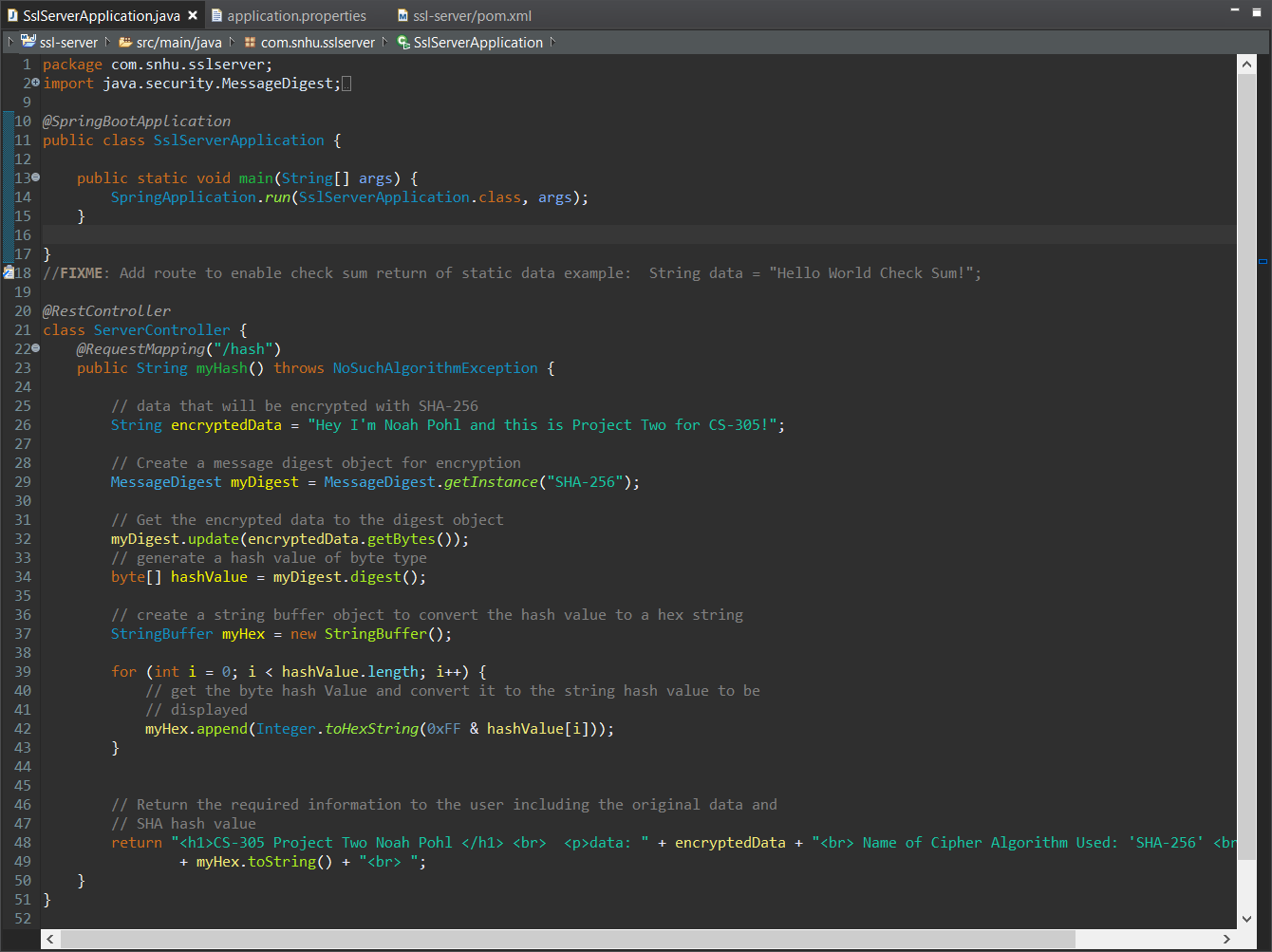
* Running the updated dependency check report with maven verifying no additional security vulnerabilities have been added to the codebase.



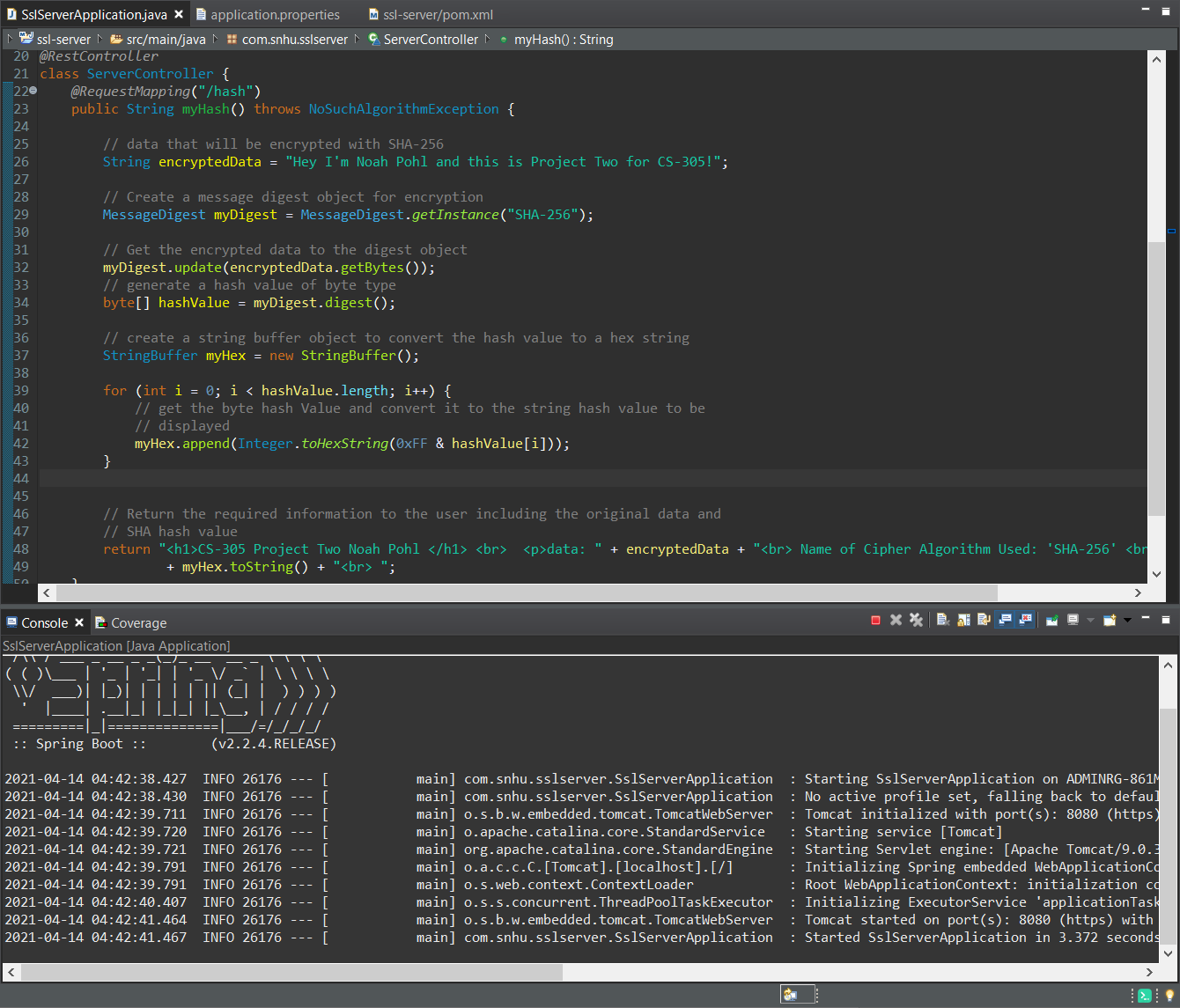
## 6. Functional Testing

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

* My refactored code includes the addition of the Java security and Message Digest libraires to implement the SHA-256 hash function for data encryption. I also am using the Spring Rest Controller to enable the check sum route accessible in the web application and display the provided data along with the SHA hash value. The full details of the refactored code can be found in the attachments to this project.



* I was then able to run the application as a Spring Boot web server without any errors during execution.



## 7. Summary

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

* Looking over the Vulnerability Assessment Process Flow Diagram there were a couple of areas I had to consider while I made progress towards refactoring the code. The first of these areas includes cryptography and the use of encryption in the codebase. I decided to implement the SHA-256 hashing algorithm for the refactor to properly encrypt the data and add a high level of security to the web application. Another area of security I needed to address was the client-server security, specifically generating a self-signed certificate with the Java key tool and then adding the key store for encrypted HTTPS security. I was able to verify my code was properly using my generated certificate across the client and server. The last area of security I needed to address was the use of various APIs within my application. I needed to run a maven dependency check to validate all of the dependencies my web application was using are not introducing any potential security vulnerabilities.
* I found the process of adding increased levels of security to my application was highly rewarding and beneficial to the Artemis Financial client. The SHA algorithm cipher I included within the codebase provides a secure file verification for their web application and can be verified using a checksum. The process of migrating the web application over from the unsecure HTTP protocol to HTTPS was also highly beneficial for the company. The HTTPS protocol creates a lock in the top left corner of a web browser indicating to their users that the application is secure and trustworthy. If the web application were still using the unsecure HTTP protocol, a user would be prompted with a message by the browser indicating the application might not be safe or trustworthy. In order to maintain this high level of security, the company should continue using HTTPS along with encrypting data with a proper algorithm such as SHA-256. By following these guidelines Artemis Financial will be able to deliver an application that is highly secure and dependable to all of their existing and potential customers.