****

# 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](#_heading=h.1t3h5sf) 4-5

[3. Deploy Cipher](#_heading=h.4d34og8) 6

[4. Secure Communications](#_heading=h.2s8eyo1) 6

[5. Secondary Testing](#_heading=h.17dp8vu) 7-8

[6. Functional Testing](#_heading=h.3rdcrjn) 9

[7. Summary](#_heading=h.26in1rg) 10

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **6-17-2022** | **Cody Gregory** |  |

## 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

Cody Gregory

## 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.

TLS\_ECDHE\_RSA\_WITH\_AES\_256\_GCM\_SHA384

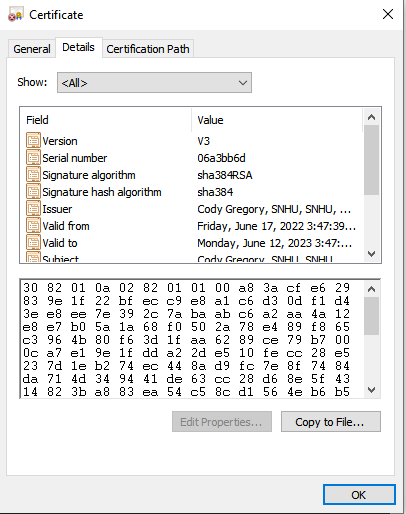
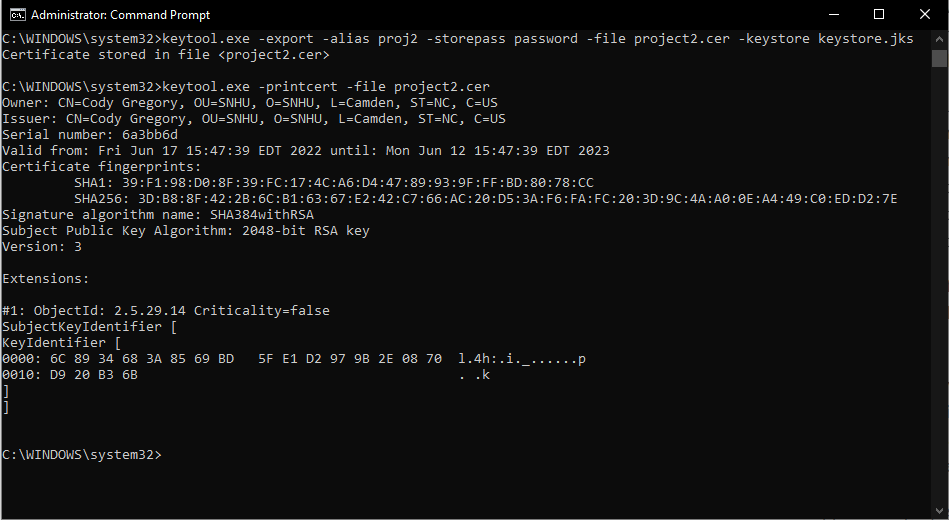
* Transport Layer Security (TLS) is the modern standard cryptographic protocol, having taken over for Secure Socket Layer (SSL). Employed by HTTPS, and the newest version 1.3 has no known vulnerabilities.
* Elliptic Curve Diffie-Hellman Ephemeral (ECDHE) is a key exchange algorithm that provides forward secrecy through the employment of “ephemeral” keys that are generated as needed.
* The RSA Cryptosystem which gets its name from its inventors, is a system based on prime numbers, and while one of the oldest algorithms has stood the test of time.
* AES (Advanced Encryption Standard) as the authentication algorithm is the adopted standard of the U.S. Government, used by the NSA for top secret communications as well as approved by the Secretary of Commerce, all of which is to say it is a secure standard that Artemis Financial will need to adopt when handling the sensitive data of their customers.
* GCM (Galois/Counter Mode) is the mode of authentication, it provides enhanced security while also offering good performance to improve the user experience.
* SHA384 (Secure Hash Algorithm) is a hashing algorithm that provides a 384-bit long hash.

## 

## 2. Certificate Generation

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

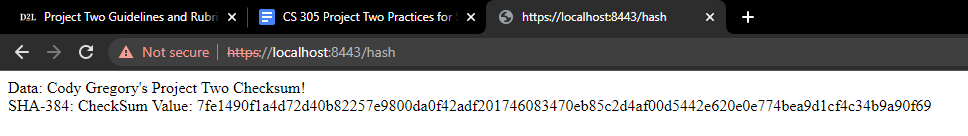
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.

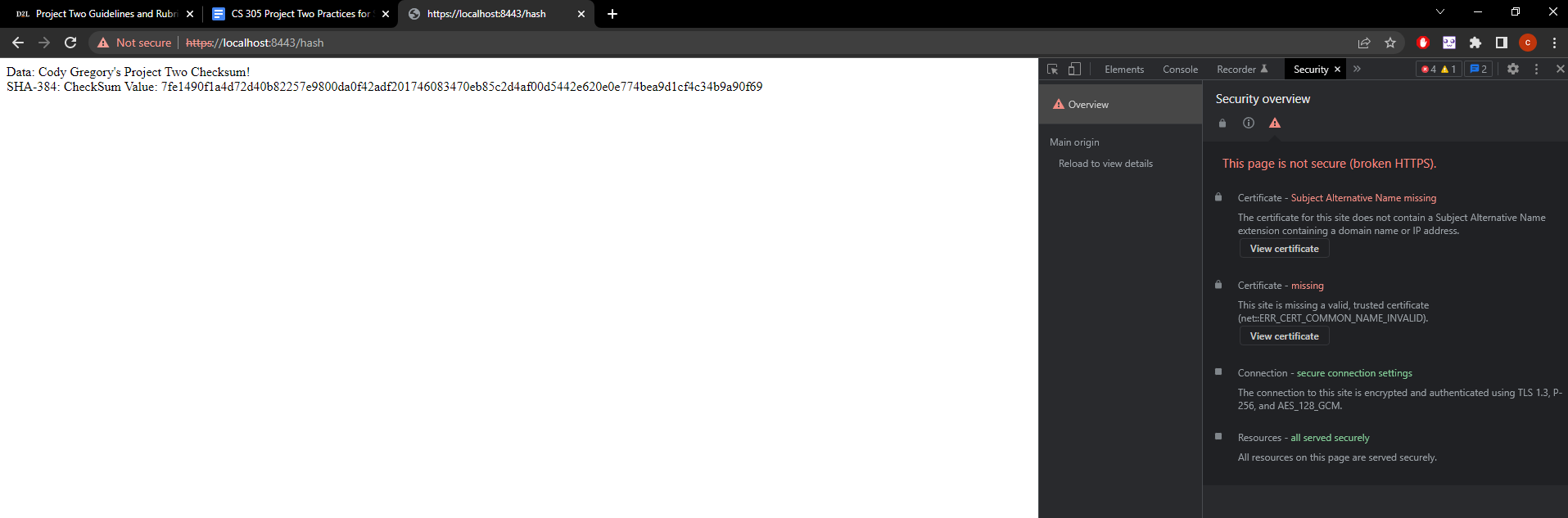
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



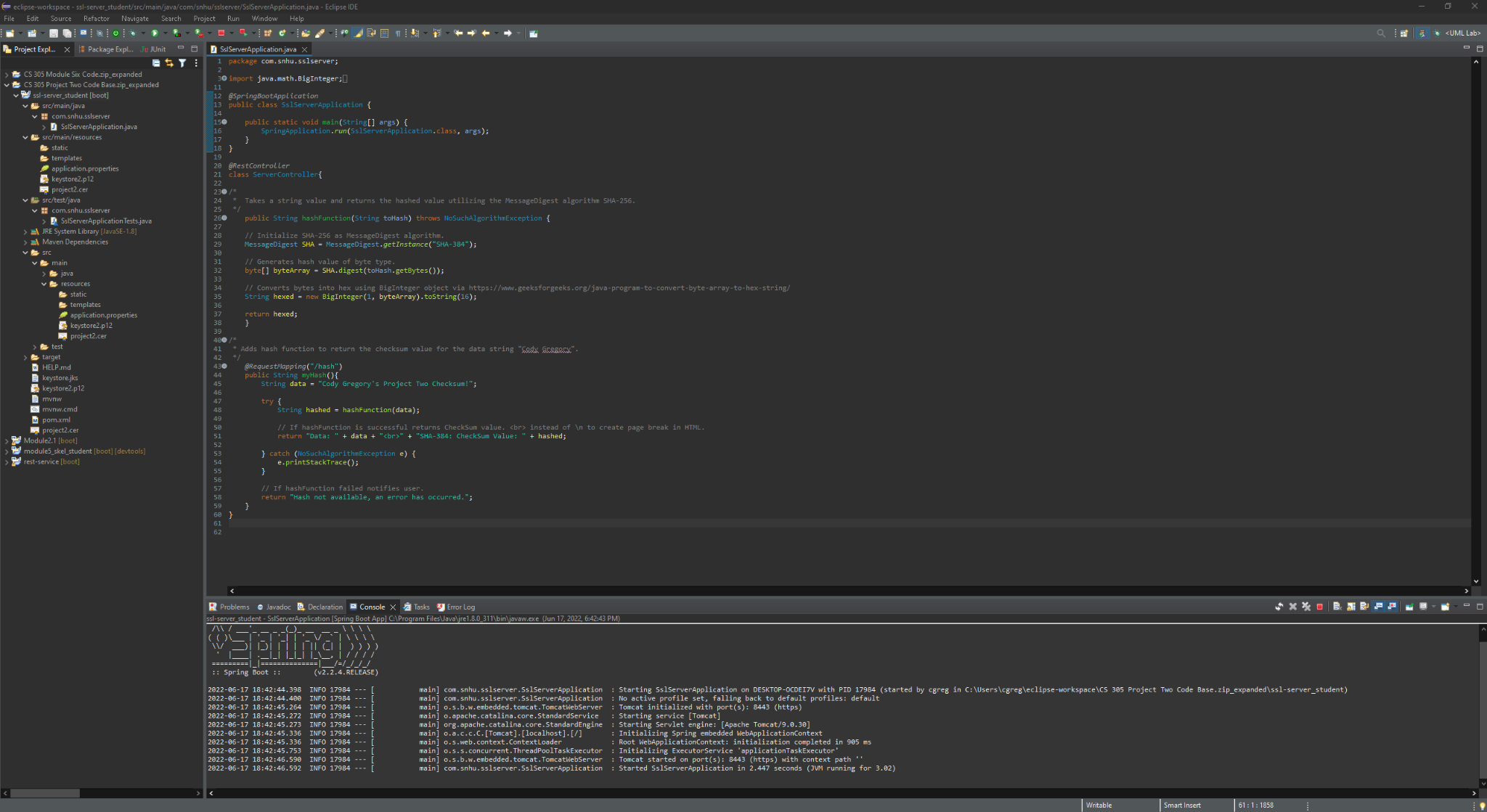
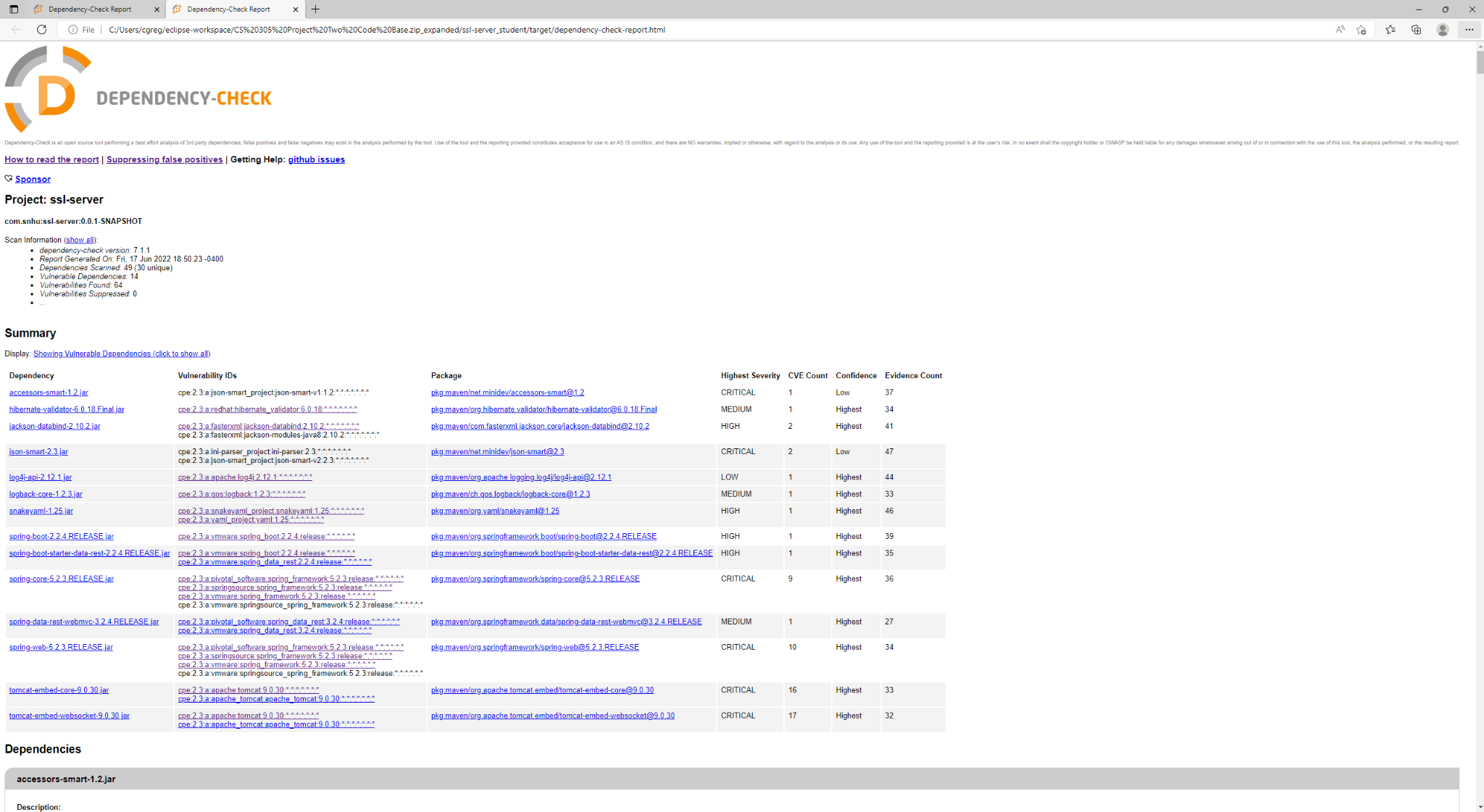
As seen above the connection is secured using TLS 1.3, however due to browser operations https cannot be validated without further credentials.

## 

## 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.

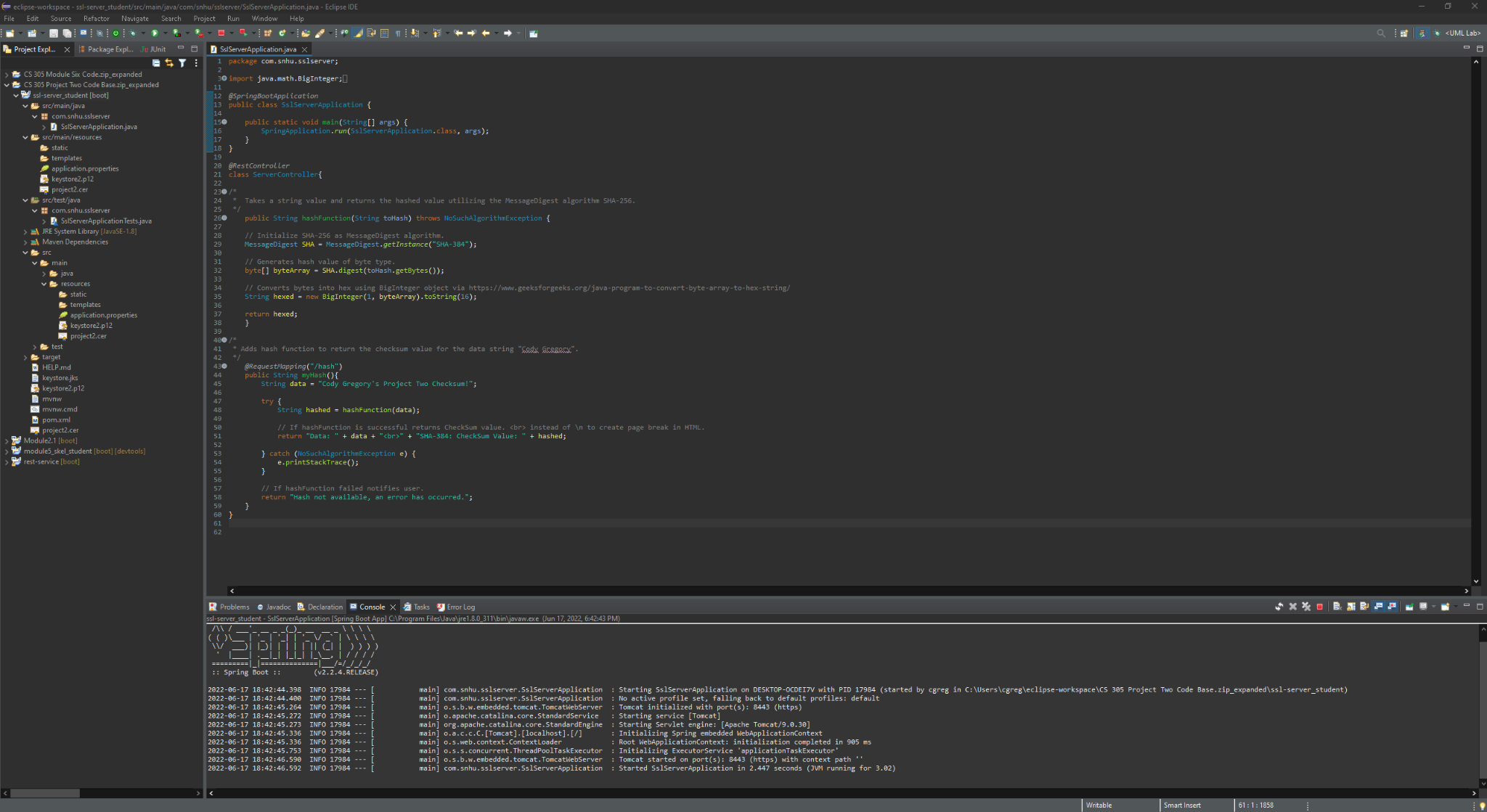
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report



## 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.



As the code stands now at its most updated state (creation) it is a sophisticated starting point to develop Artemis Financial’s web application. Potential vulnerabilities could be isolated to the possible change of syntax involving the Secure Hash Algorithm (SHA) wherein the currently employed “SHA-384” could cause a security threat.

## 

## 

## 

## 

## 

## 

## 

## 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.

Security is never a problem until it is a problem. When assessing the security of Artemis Financial’s web application a combination of testing and logical evaluation needs to be applied to ensure their business is safe from malicious entities.

Static testing and dependency checks are a great way to ensure that “outsourced” code in the form of libraries such as Spring or Tomcat are up to date and do not suffer from any vulnerabilities that could harm the system outside of the developed code. As it stands currently Tomcat nor the Spring Core have any known vulnerabilities, but that is always subject to change and thus needs to be reviewed regularly.

For in-house developed code best practices need to be applied to prevent any errors that could cause a security breach. These errors can be in the form of a logical error that open the system to an attack, to even things such as poor syntax and javadoc that result in confusing code which is difficult to keep track of resulting in an uncertain environment where issues are more likely to arise.

To best maintain security of the application it is recommended that regular dependency checks are reviewed to ensure no dependencies have ground-breaking vulnerabilities. Any additions to the code base need to be clearly defined with javadoc as necessary so that it is clear what every line of code’s function is, making for an easier system to work with and also spot any potential malicious code as well.