****

# Practices for Secure Software Report

Table of Contents

[Document Revision History 3](#_Toc102040754)

[Client 3](#_Toc102040755)

[Instructions 3](#_Toc102040756)

[Developer 4](#_Toc102040757)

[1. Algorithm Cipher 4](#_Toc102040758)

[2. Certificate Generation 4](#_Toc102040759)

[3. Deploy Cipher 4](#_Toc102040760)

[4. Secure Communications 4](#_Toc102040761)

[5. Secondary Testing 4](#_Toc102040762)

[6. Functional Testing 4](#_Toc102040763)

[7. Summary 4](#_Toc102040764)

[8. Industry Standard Best Practices 4](#_Toc102040765)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **10/18/2024** | **Brandon Cook** |  |

## Client



## Instructions

Submit this completed practices for secure software report. Replace the bracketed text with the relevant information. You must document your process for writing secure communications and refactoring code that complies with software security testing protocols.

* Respond to the steps outlined below and include your findings.
* Respond using your own words. You may also choose to include images or supporting materials. If you include them, make certain to insert them in all the relevant locations in the document.
* Refer to the Project Two Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Brandon Cook

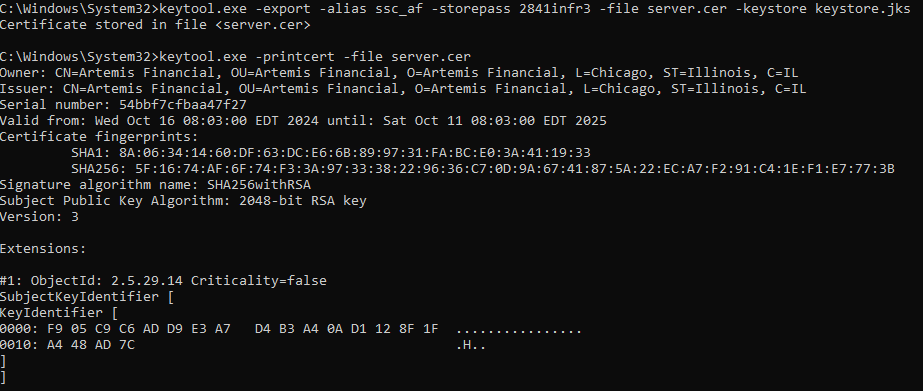
## Algorithm Cipher

An organization that handles sensitive information, such as credit cards, social security numbers, passwords, or any other information, will need to select an appropriate encryption cipher to secure their data at rest. When it comes to securing data at rest, the best method is to wrap the data in several layers of encryption. Not securing data could lead to many issues, such as legal ramifications and a loss of reputation. Security needs to be in place to comply with government regulations, such as HIPAA and the GDPR, if information from the European Union is being handled.

A symmetric algorithm, AES-128, should be used to fit Artemis Financials needs. The AES algorithm supports three different key lengths: 128, 192, and 258-bits. The AES-128 cipher works by segmenting the data into blocks that are 128-bits in length, and then will encrypt each block separately. Using a symmetric key for this process works best for Artemis Financial; they are encrypting their archives, and are not dealing with outside organizations. A symmetrical algorithm works by using a key to encrypt data; that same key is then used to decrypt the data later. The organization will only need to keep the key they used for encryption safe. Occasionally, for best practice, it would be best to occasionally change the key. An asymmetrical algorithm is different in that there is a private key that is used for decryption, and a public key that is shared with the outside world for other organizations to send encrypted data. Only the private key can be used to decrypt that data, the public key will not work. An asymmetric cipher would be better if Artemis Financial was looking to handle data in transit.

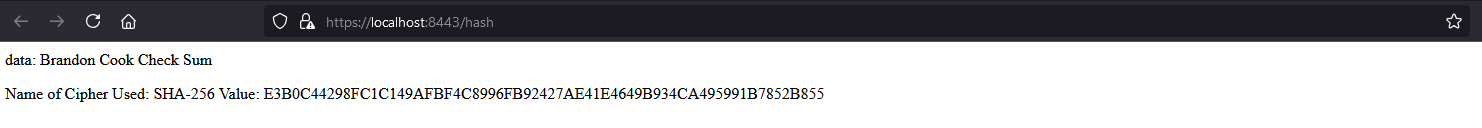
## Certificate Generation

Insert a screenshot below of the CER file.



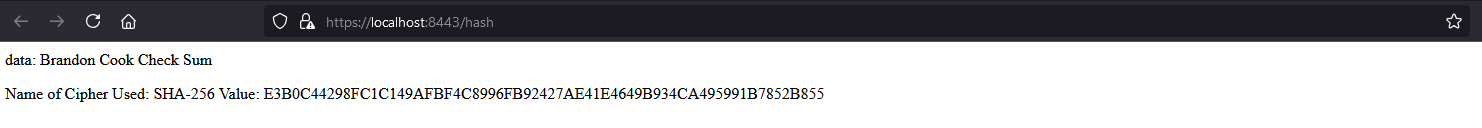
## Deploy Cipher

Insert a screenshot below of the checksum verification.



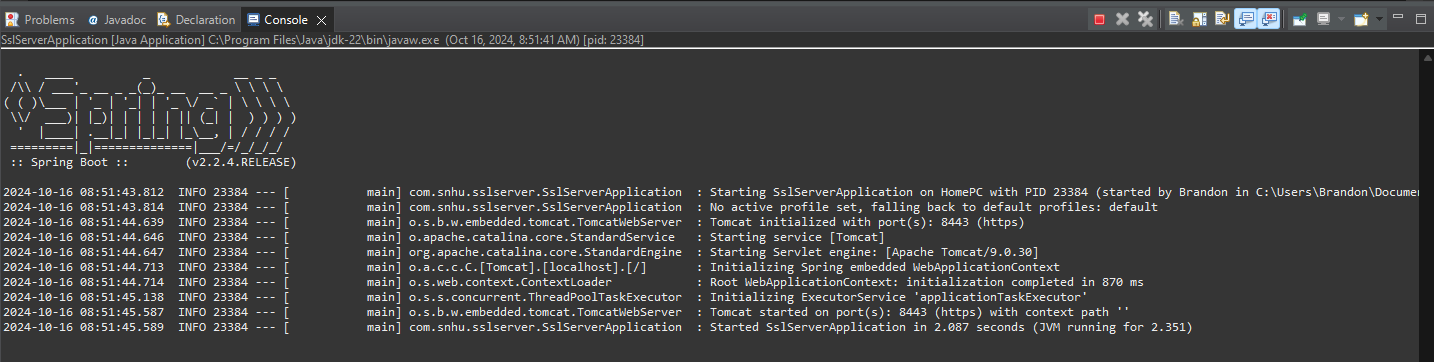
## Secure Communications

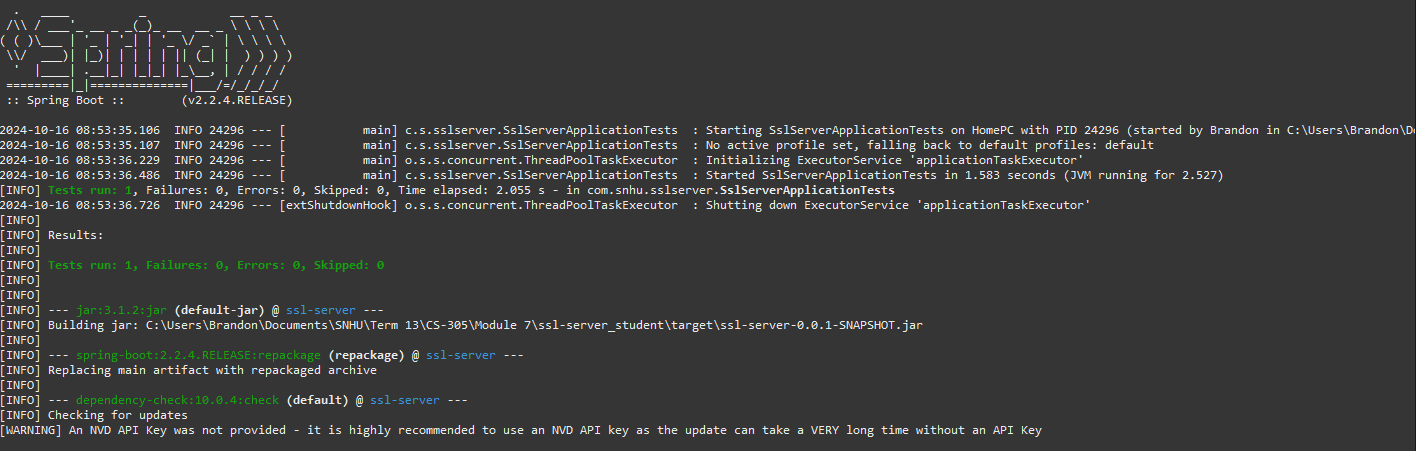
Insert a screenshot below of the web browser that shows a secure webpage.

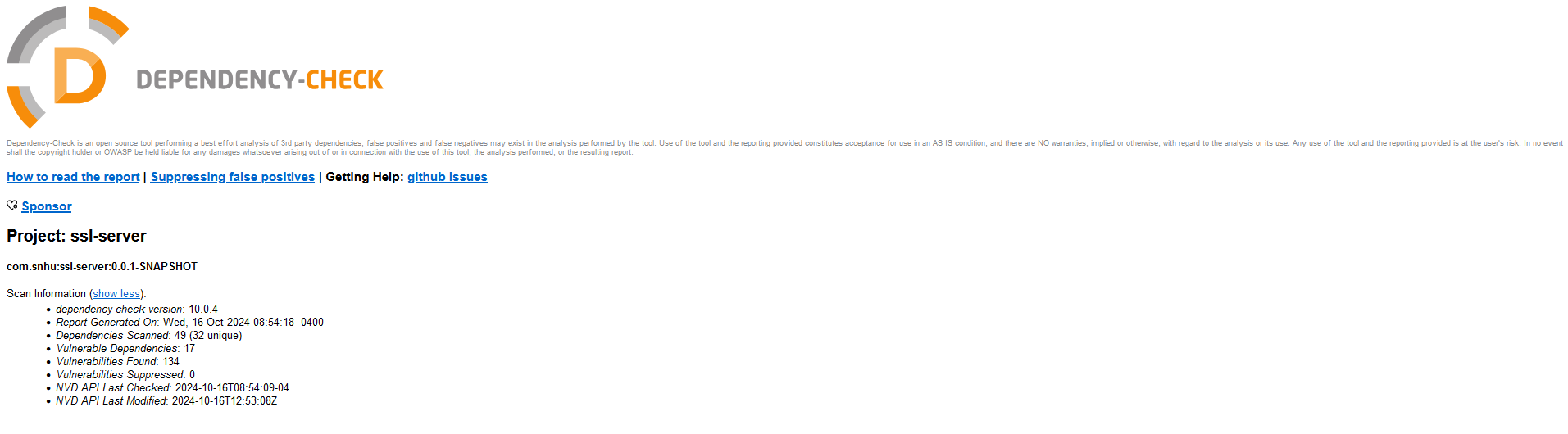


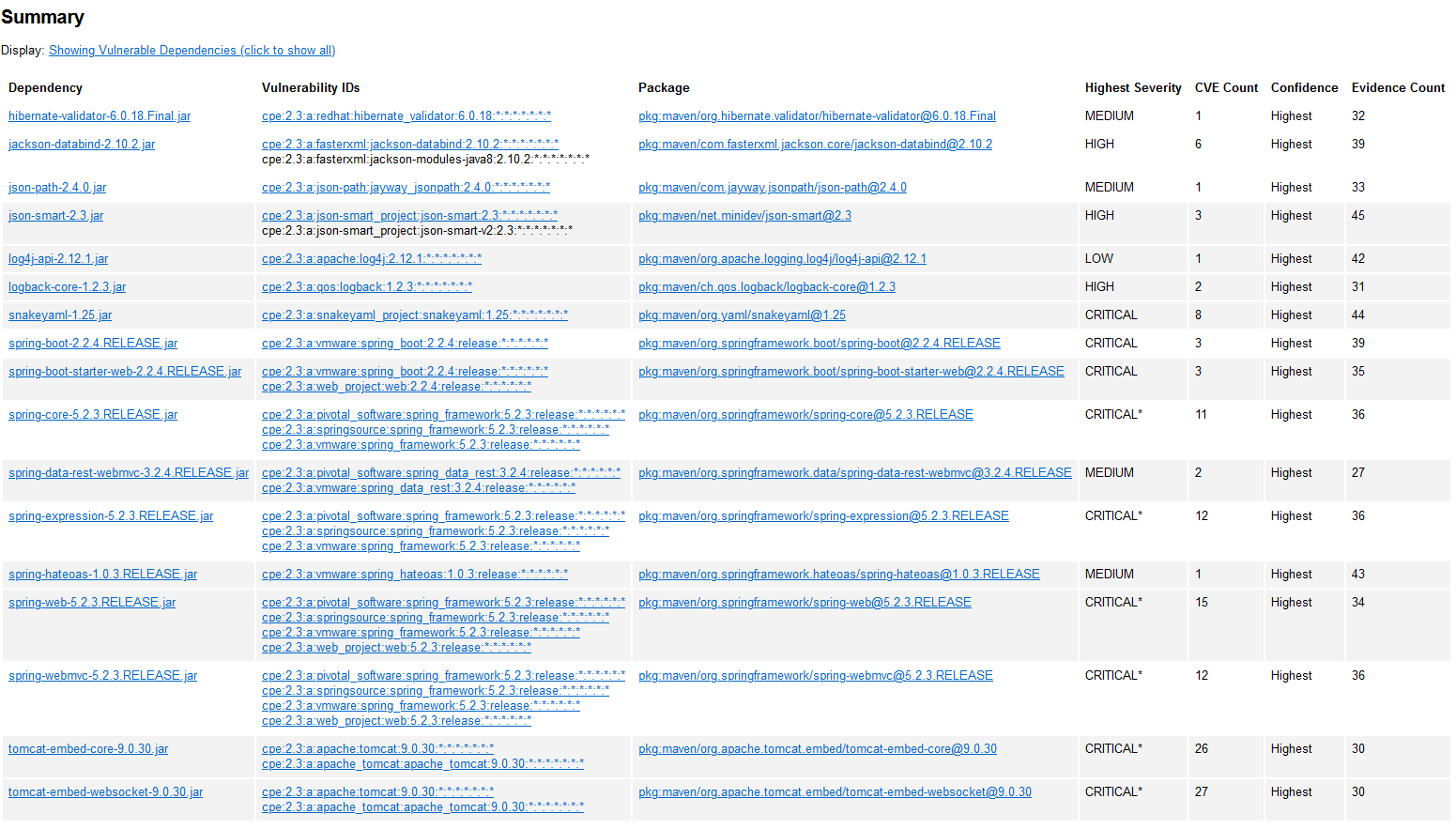
## Secondary Testing

Insert screenshots below of the refactored code executed without errors and the dependency-check report.



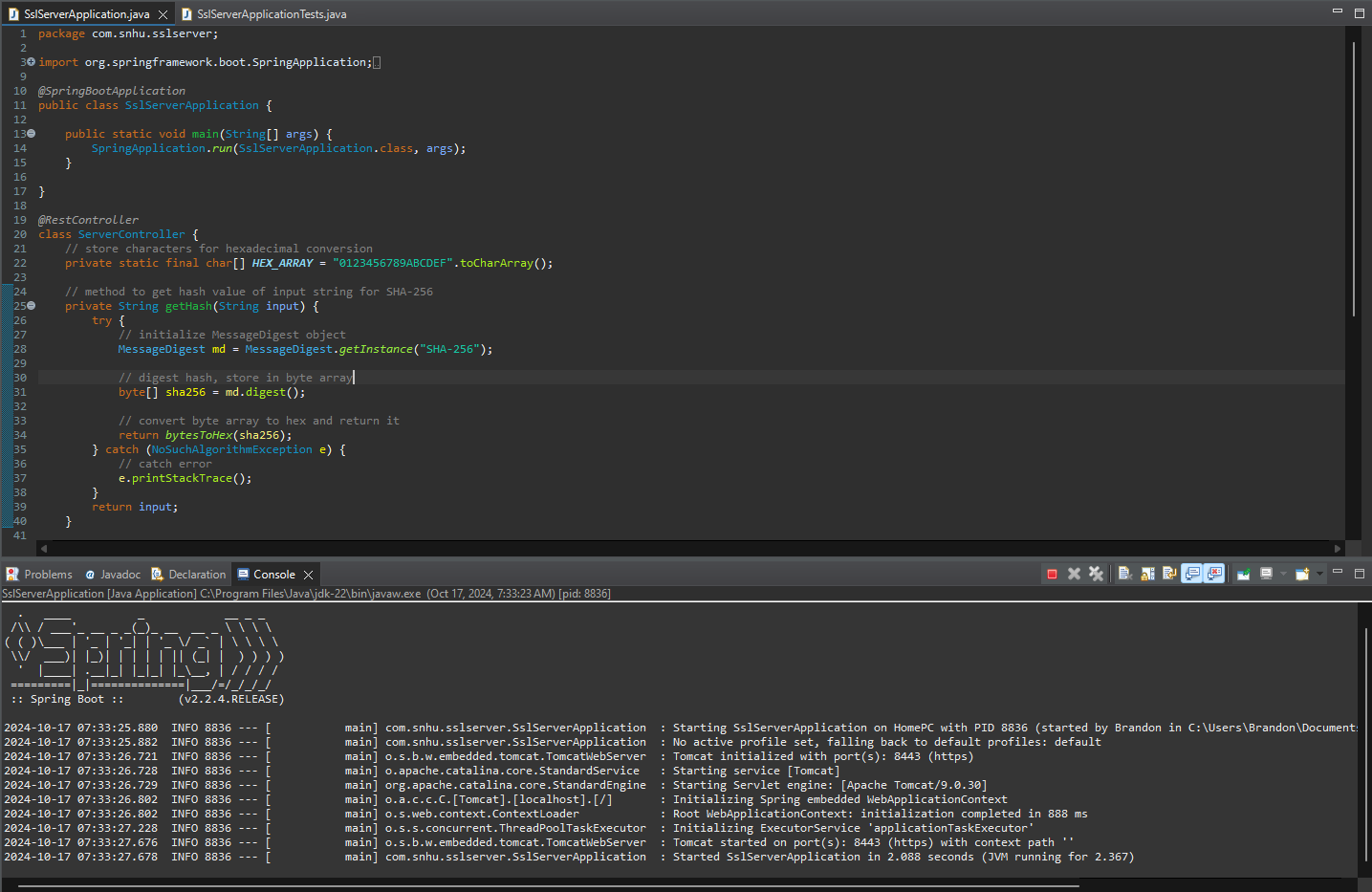


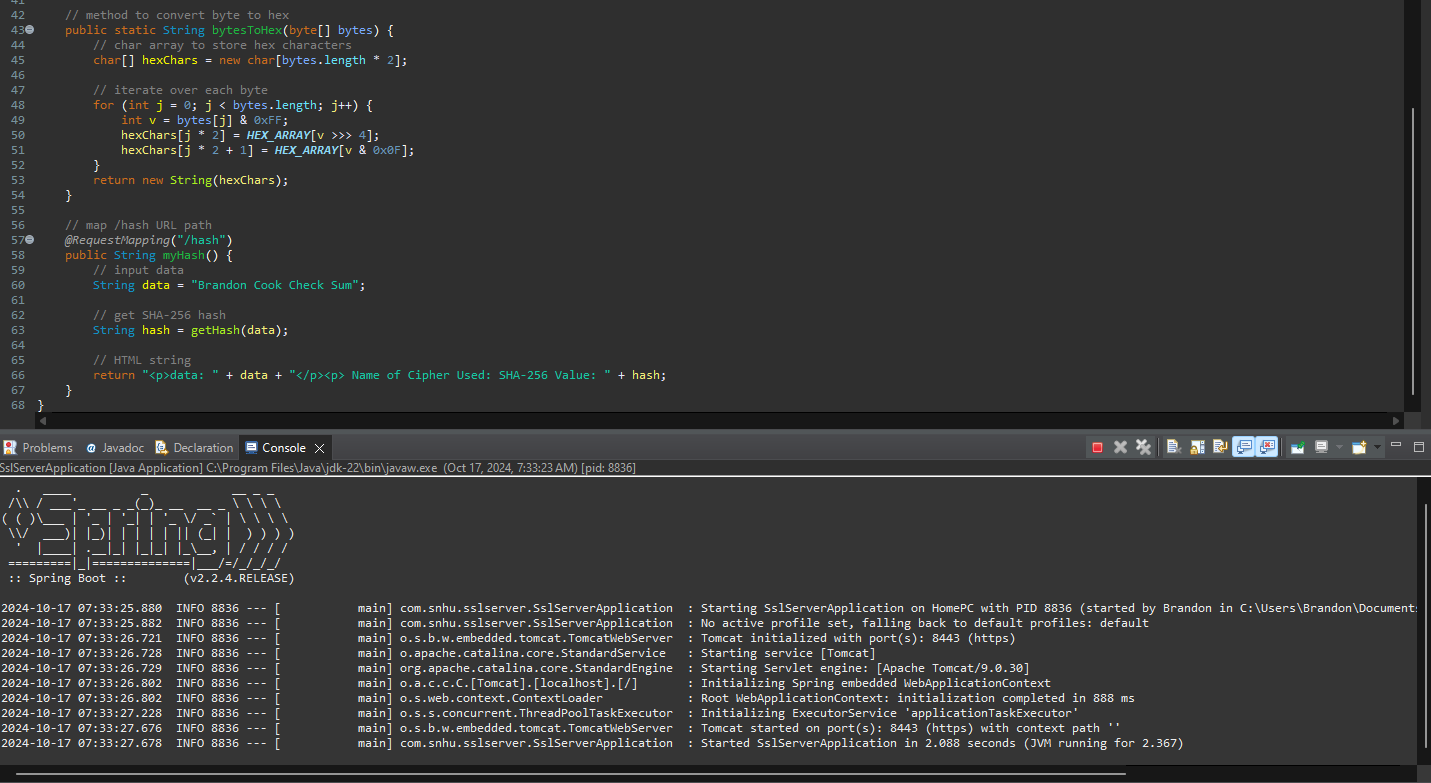


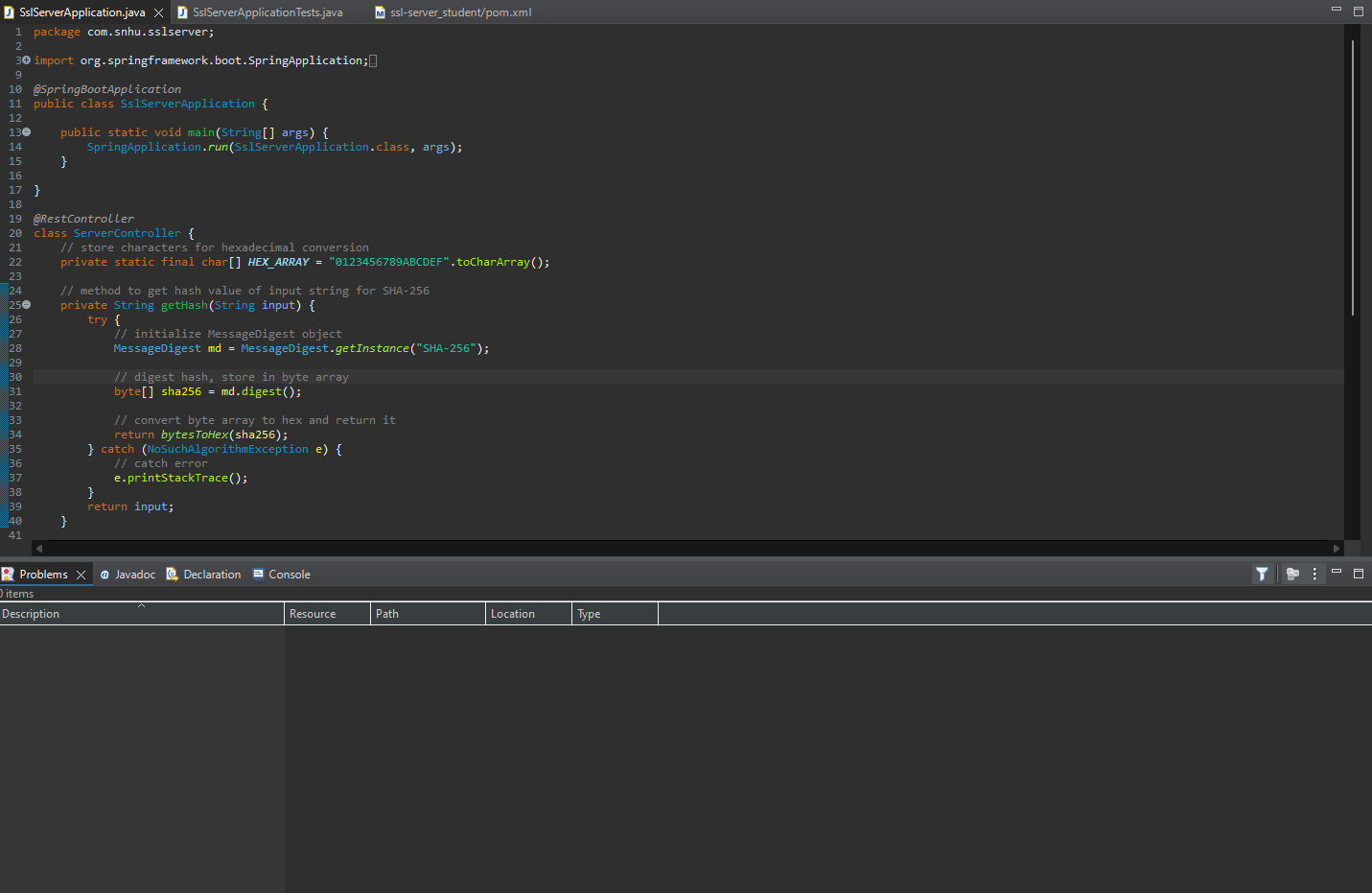


## Functional Testing

Insert a screenshot below of the refactored code executed without errors.







## Summary

Cryptography is used to ensure the integrity of data that is sent. SHA-256, a secure hashing algorithm, is used to check and verify the input string “Brandon Cook Check Sum.” Doing this will make the software less susceptible to attacks, as it will be easier to determine if the data has been tampered with.

The software was refactored with proper error handling procedures in place to catch a potential exception. The getHash method catches NoAlgorithmException errors, which can be thrown if SHA-256, or any other requested algorithm, is unavailable. This will minimize crashes and vulnerabilities that may lead to data leaks.

In the future, any other input that may be handled by the getHash method, passed into it by the argument, should be sanitized to ensure that it is not attempting to exploit the code. Additionally, introducing better error logging and monitoring capabilities can further help with locating issues that arise.

## Industry Standard Best Practices

The program uses the SHA-256 algorithm, widely considered to be one of the best options available for cryptographic hashing. Using secure algorithms like this rather than more outdated algorithms, such as from the SHA-1 family, leads to less vulnerabilities and collisions. The use of try-catch blocks helps with handling potential exceptions that may be thrown, whether from input manipulation or oversight. The value in this is that it prevents attackers from gaining knowledge about the system’s inner workings, which they can later use as an attack vector. Detailed error messages and logs should not be exposed to a user, but should be logged for later review by the developers. Using industry standard best practices prevents attack vectors from surfacing, minimizes vulnerabilities, and helps maintain compliance with government regulations.