****

# 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** | **12/11/21** | **Alex Grimes** |  |

## 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

Alex Grimes

## 1. Algorithm Cipher

I would come to Artemis Financial with AES 256 bit this form of encryption is a very strong option with its only real weakness being a brute force attack. However, there are a ton of other ways to negate brute force attacks that are easy to implement. This form of encryption is endorsed and used by the Government and is believed to be a standard of the future. There are currently very minimal encryption laws in the US here is a short description of what there is currently.

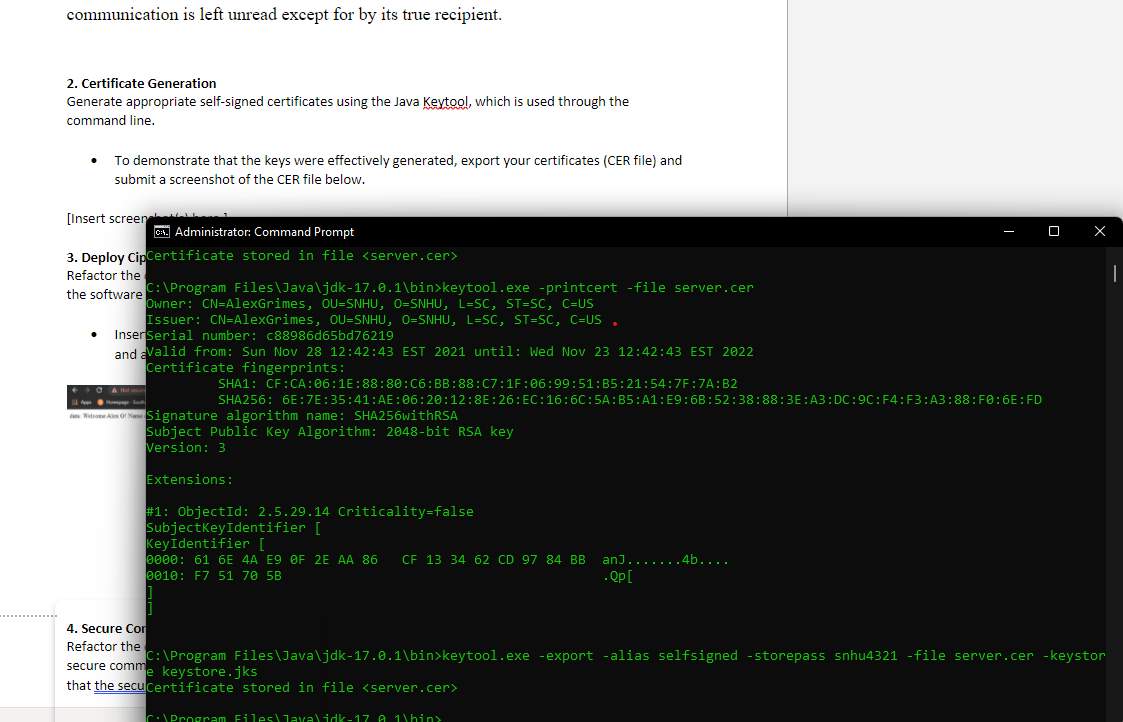
**“In the United States, the law imposes controls on the export of certain forms of encryption. There is no legislative power which can be used to require telecommunication or online service providers to facilitate the decryption of encrypted communications. However, all telecommunications carriers are required to ensure that their equipment, facilities, or services that provide a customer or subscriber with the ability to originate, terminate or direct communications have certain capabilities which includes interception of communications and delivering intercepted communications to the government, where the government obtains a court order or there is some other lawful authorization. Telecommunications carriers however cannot be required to decrypt, or to ensure the government’s ability to decrypt, any communications which are encrypted by the subscriber or customer unless the encryption was provided by the carrier and they are able to decrypt it.”**

In summary, there is no rule for or against encryption, but your provider is required to provide the government with communications if requested. This is even more of a reason to encrypt strongly with the best software available and that is AES. The only real problem with the method currently is that if the key is somehow lost to the encrypted data you can consider it lost for the near future as with current computers even left to their own devices it would take decades to crack. This is However less of a concern with it being an asymmetric 192- or 256-bit key meaning that there is only one key for encrypting and decryption. As for its history, it has been around since 1997 however I think it just speaks for how solid this method of encryption is. To this day the 256-bit has not been cracked. This is extremely impressive. There certainly are other options specifically asymmetric ones with different keys for sending and receiving but I truly believe this AES method will be the most certain tool to use to ensure that our communication is left unread except for by its true recipient.

## 2. Certificate Generation

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

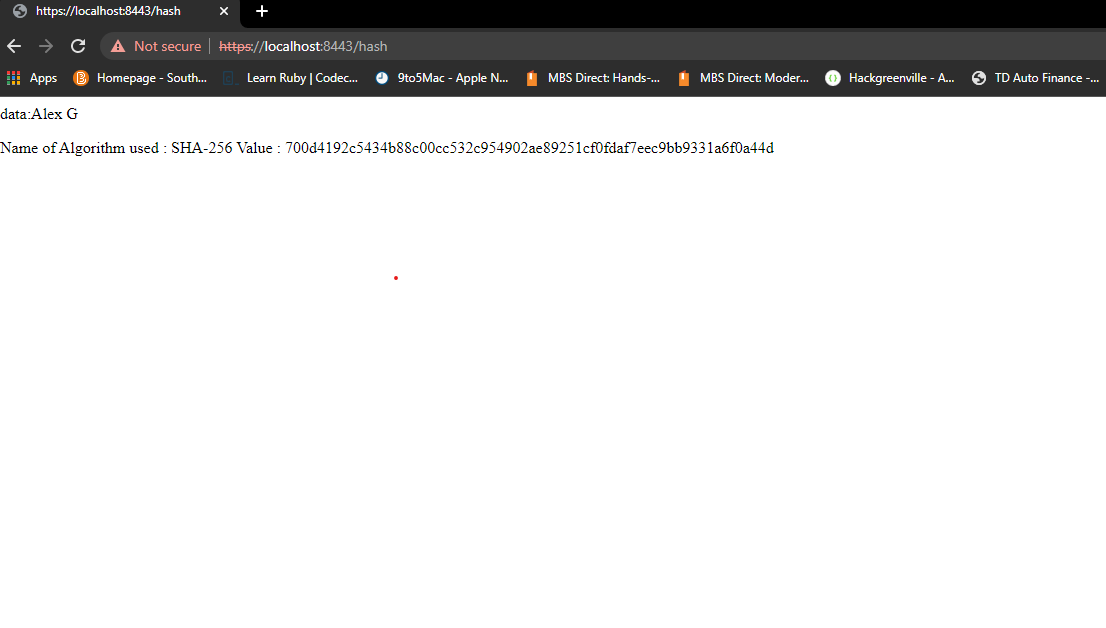
* To demonstrate that the keys were effectively generated, export your certificates (CER file), and submit a screenshot of the CER file below.



## 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.

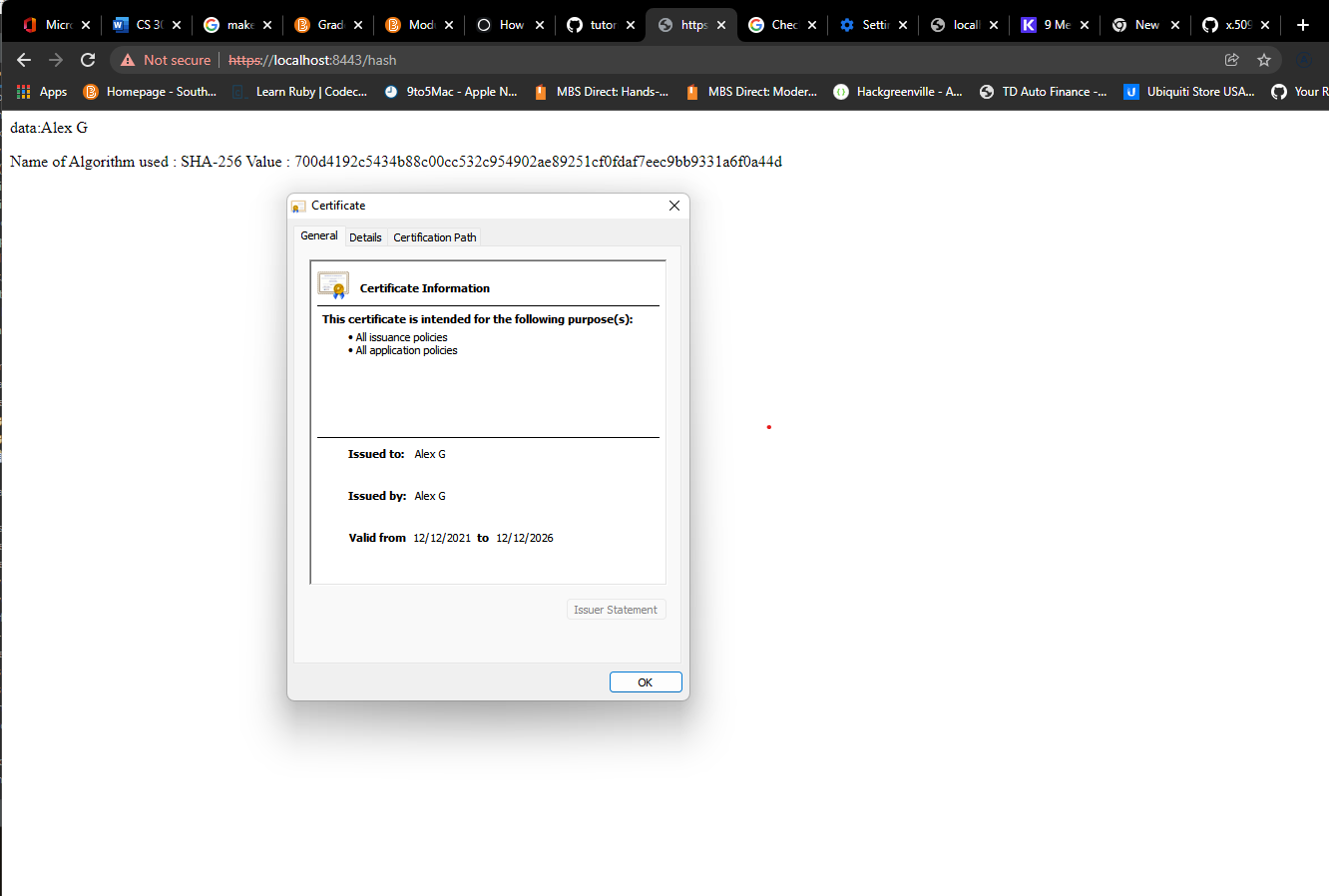
* Insert a screenshot below of the checksum verification. The screenshot must show your name and a unique data string that has been created.



## 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.



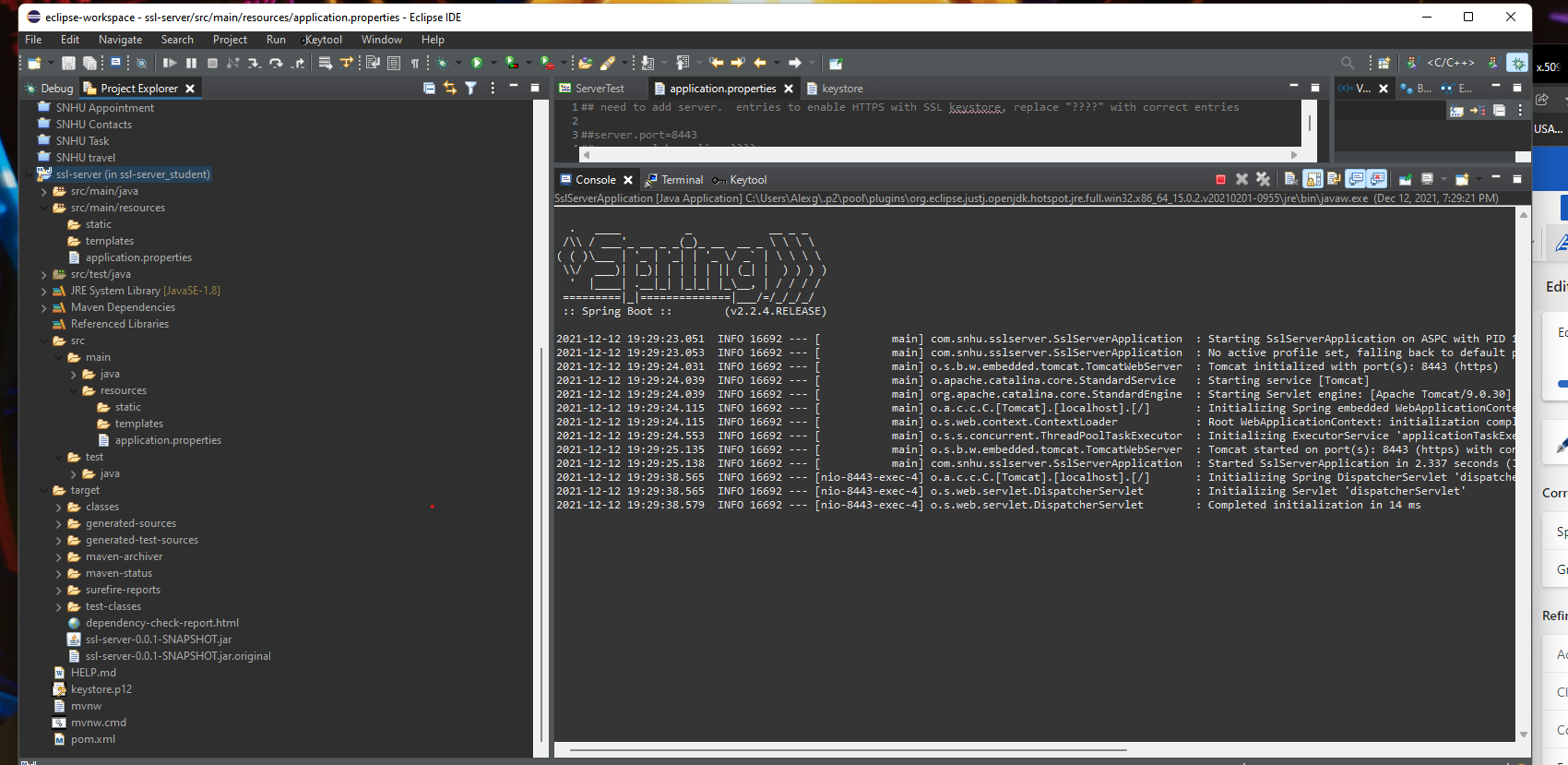
HTTPS not working exactly correctly but I believe this is a problem with my device as you can see correctly using my certificate in the image screenshot.

[Insert screenshot(s) here.]

## 5. Secondary Testing

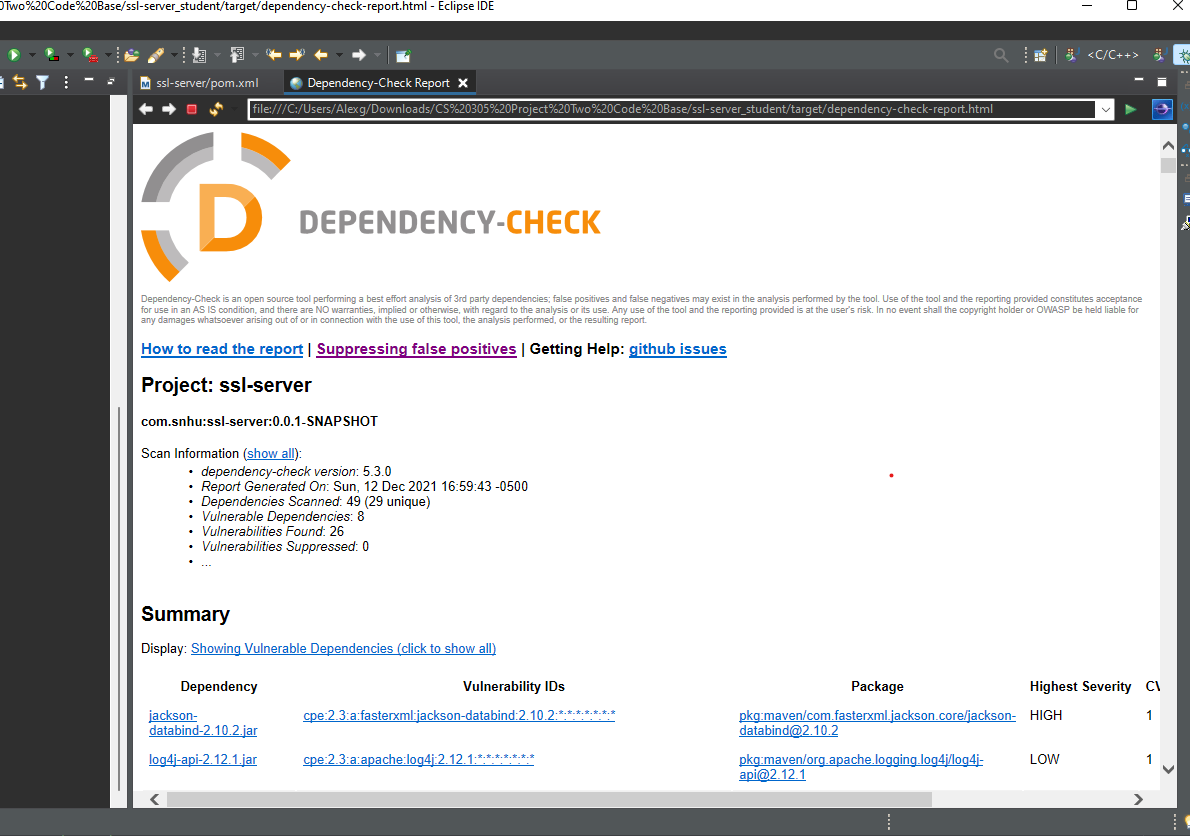
Complete a secondary static testing of the refactored code using the dependency check tool to ensure code complies with software security enhancements. You only need to focus on the code you have added as part of the refactoring. Complete the dependency check and review the output to ensure you did not introduce additional security vulnerabilities.

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



Refactored code running with no issues.

* + A screenshot of the dependency check report



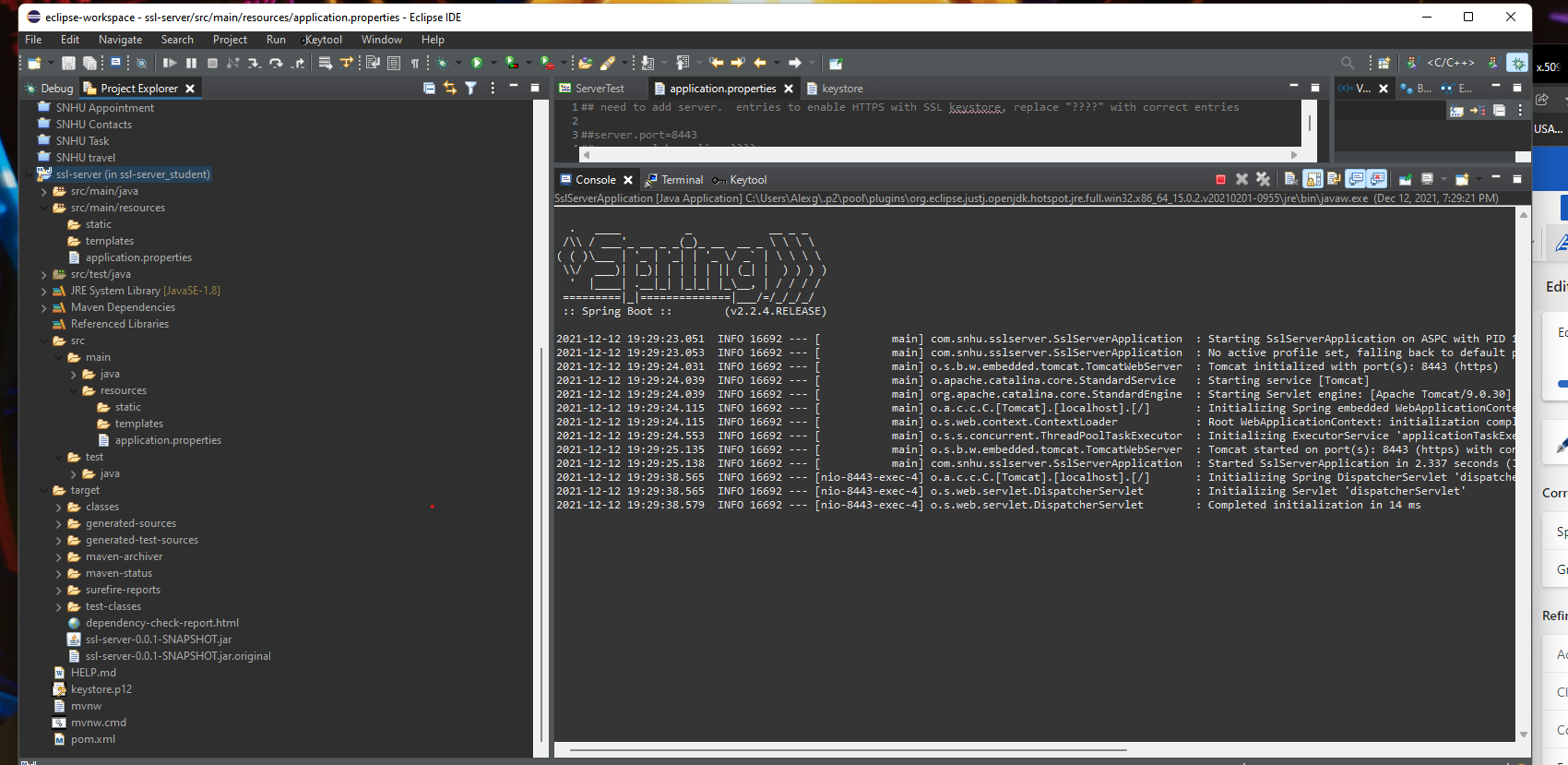
No additional vulnerabilities added to the system.

## 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.

Manually Reviewed code for any errors corrected issues and code ran without error.



## 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.

My Process for adding security to this application was as follows. Initially, I wrote the chunks of code that remained and did a once over static check of the code to confirm that everything appeared correct to my eyes and without bug or error. After the static check and configuration, I ran a maven vulnerability scan to catch any issues I did not see as well as get a list of the possible vulnerabilities. Using this as a template I went back and checked the code manually for a second time to fix any errors that could be repaired as well as assess the vulnerabilities. After this, I launched the java application to do some dynamic testing and confirm that the software was running as expected. After a bit of back and forth with slight configuration adjustments, the application was running smoothly and as expected. I believe that this repeat testing of small chunks and reinforcing as I go is the best way to secure the application during the development phase. During these checks, I was addressing multiple points on the vulnerability assessment diagram. These included cryptography, client/server, code error, code quality, and dependencies checks. In the Future the customer is going to want to update and patch regularly as well as keep up with the latest information on the vulnerabilities that do still exist these may become more severe or known fixes for them may appear at a later date and can be implemented.