****

# 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** | **December 12, 2020** | **Jesse Quijano** |  |

## 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

Jesse Quijano

## 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.

For the purposes of this application, I believe Artemis Financial should use RSA encryption for their certificate. This is mostly because encrypted data will be sent across the web. In cases where encrypted data is being transmitted to remote systems, it makes more sense to use an asymmetric algorithm such as RSA. The reason for this is that asymmetric encryption has two keys: a private one only known to the system that will be encrypting the data and a public one that will be used by any system that wishes to decrypt the data.

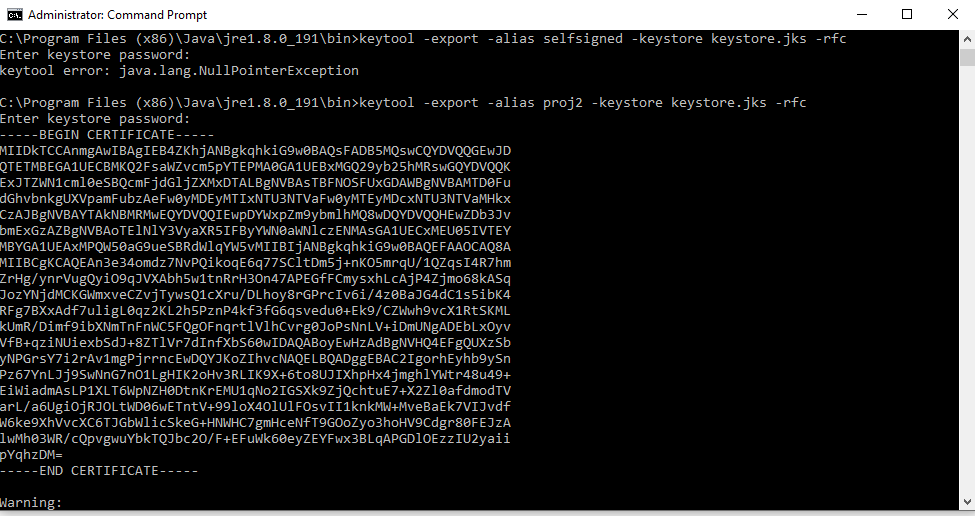
RSA public keys are handled in 1024-, 2048-, and 4096-bit sizes. The obvious tradeoffs for these is performance over security. It is typically safe to use RSA with 2048-bit encryption unless you are working with sensitive information. This is to ensure your app is still secure while remaining performant.

In modern times, RSA and AES should always be used over DES due to DES’s deprecation. In fact, AES was initially created as an upgraded DES and to address DES’s issue of short (56-bit) keys. Triple DES, the predecessor to DES, might have been an option here, but AES is often recommended over Triple DES for symmetric algorithms for performance reasons.

## 2. Certificate Generation

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

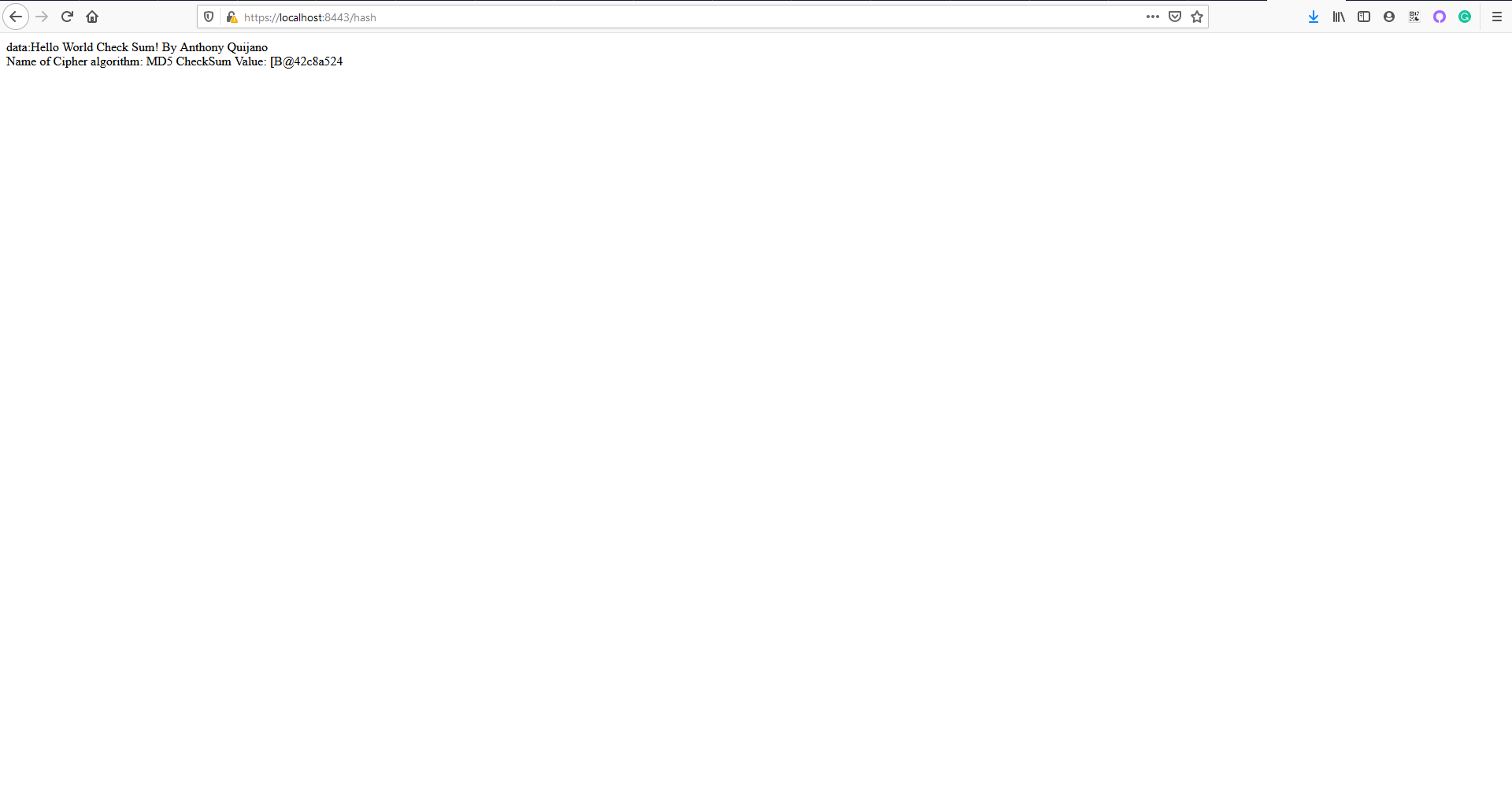
* To demonstrate that the keys were effectively generated, export your certificates (CER file) and submit a screenshot of the CER file below.



## 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.



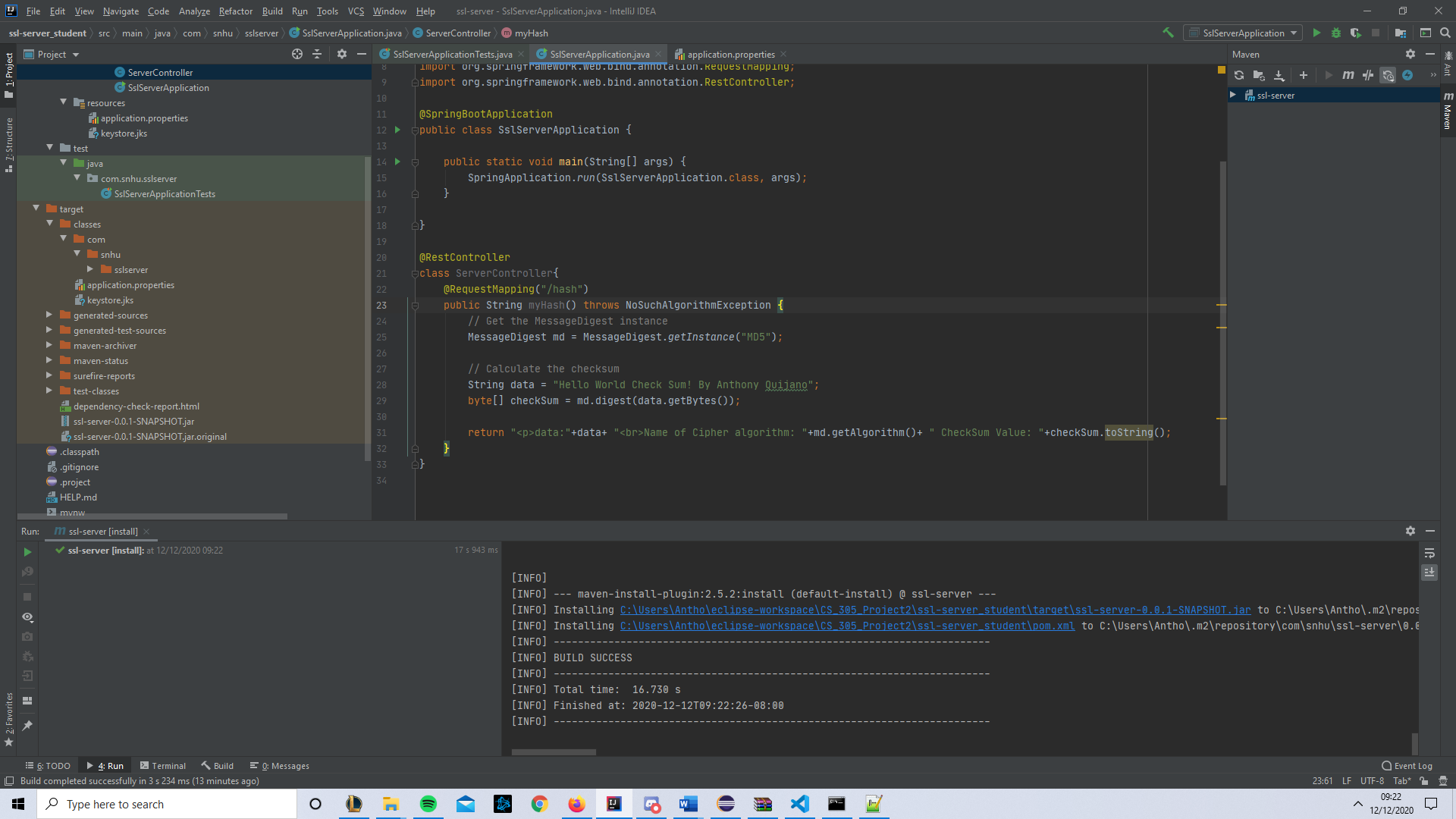
## 5. Secondary Testing

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.

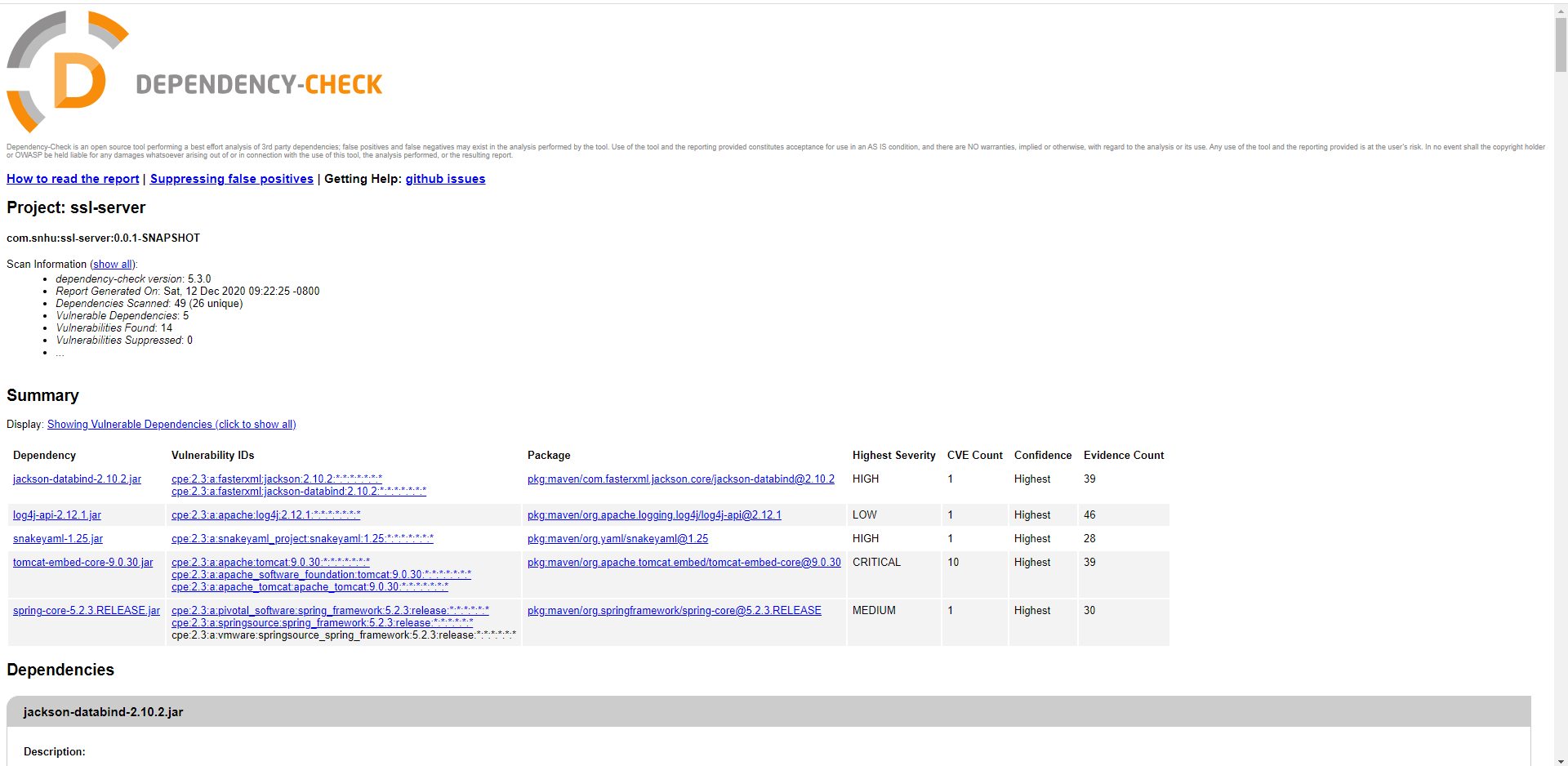
* Include the following below:
  + A screenshot of the refactored code executed without errors
  + A screenshot of the dependency check report

Refactored code:

(The “BUILD SUCCESS” message can be seen in the IntelleJ terminal)



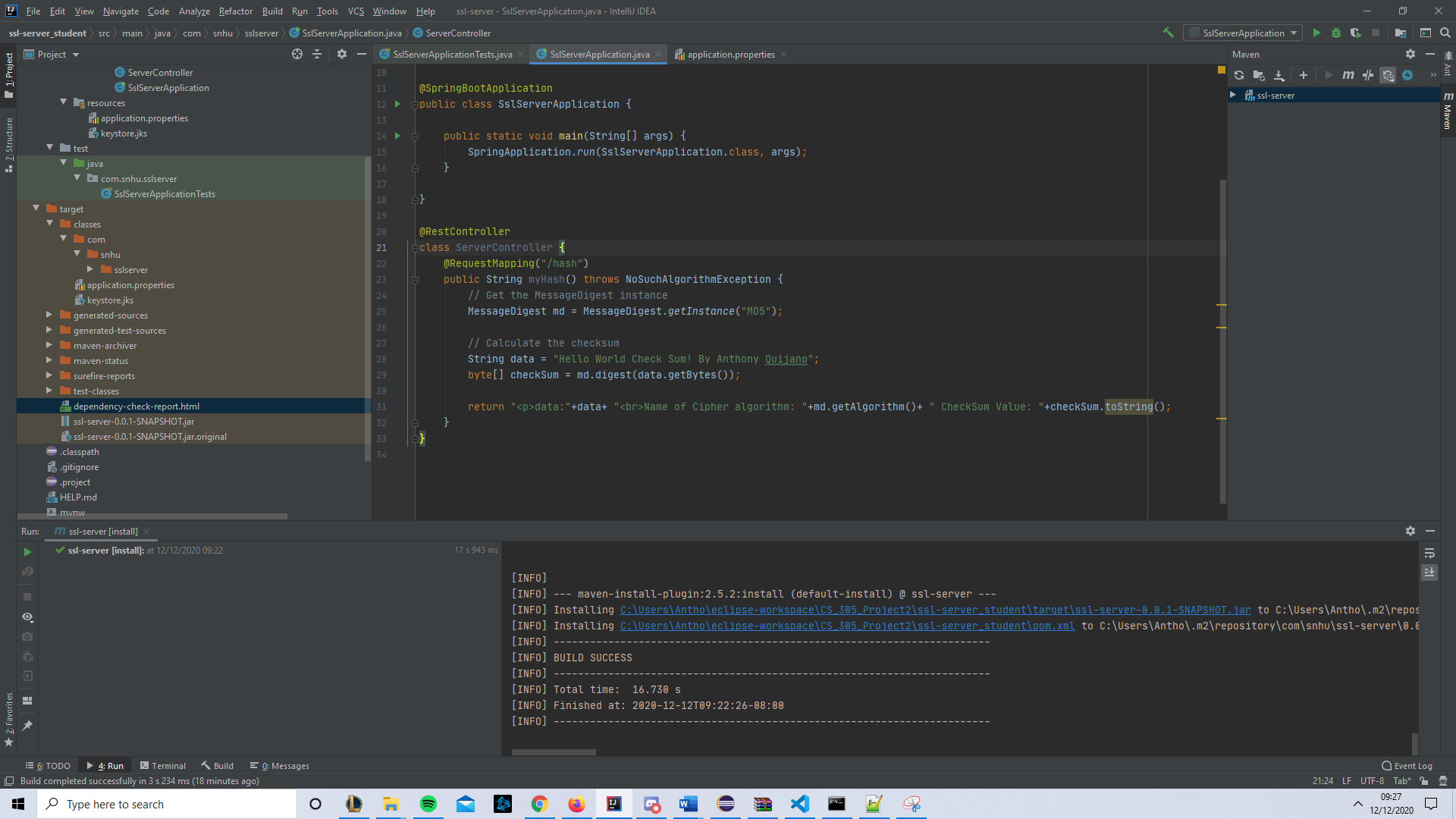
Dependency check:



## 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.



## 

## Lacking Authentication – Although there is no immediate need for there to be authentication within this application, Artemis Financial will likely need to expand their REST API to serve more sensitive data. In its current state, the application is vulnerable to anyone who is aware of the mapped routes.

## Currently, certs are being saved directly as part of the project. Developers working on this application in the future will need to be sure to avoid saving the entire project on multiple machines, i.e. remote git repositories.

## Currently, no errors are explicitly handled within the /hash route. This could cause undesired behaviors for users.

## 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.
* **APIs –** The purpose of this application is to serve as a REST API for the client. In updating the code base, had to address this area of Vulnerability assessment. The API route was developed using the Spring framework and HTTPS, so security measures were taken into account.
* **Cryptography –** In updating Artemis Financial’s code base, different cryptographic algorithms and ciphers were considered. After some review, the application was updated to use RSA encryption with SSL certs.
* **Client/Server** – The application we are updating is a server with a REST API. In our changes, we added a new route to the API that will be served by our application. Security best practices were taken into account here.

To add security to the application, I first considered the application context. Being that it is a REST API server, I knew that a symmetric encryption algorithm would not be idea. The app will be serving data over the web, so I opted for the asymmetric algorithm, RSA. Once I chose my algorithm cipher, I made use of Java’s keytool to generate a self-signed cert. In practice, Artemis Financial should have their certificate generated by a CA.

With the cert generated, we deployed it to our application by adding it to updating the application.properties file meant to be used with the Spring framework. Doing this established SSL/TLS to Artemis Financial’s REST API server. This measure can protect their organization from various attacks like phishing or man in the middle attacks. It makes their brand more trustworthy for users accessing their content.

To maintain the security of the application, Artemis Financial will need to ensure their certs do not expire. Typically, CAs will issue certificates with expiry dates to prevent algorithmically generated keys from remaining out in the wild forever.

Another step they must take is to implement new routes using the existing framework. Spring is a well-established and trusted architecture used by Java engineers to build REST APIs. It has support for SSL/TLS certificate deployment that makes this process a breeze. Artemis Financial should continue building their API using Spring moving forward to ensure security.