****

# 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** | **08/13/2023** | **Alexander Creznic** | **First Version** |

## 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

Alexander Creznic

## Algorithm Cipher

***Brief Overview of Encryption Algorithm Cipher***

The Secure Hash Algorithm SHA-2 family is sufficient to meet the current security needs of Artemis Financial. After review of the requirements and research into the transactions that require this algorithm cipher, an upgrade to SHA-512 from SHA-256 is recommended for Artemis Financial.

***Discuss the Hash Functions and bit Levels of the Cipher***

Developed by the National Institute of Standards and Technology (NIST), this algorithm has a length of 512 bits, making it twice as long as the SHA-256 algorithm cipher (Dobraung, Eichlseder, & Mendel, 2015, p. 614). While SHA-256 is quite secure, its length being half of SHA-512 means that, at some point in the future, there will be two inputs whose has *will* be equal. Upgrading to SHA-512 cuts this risk in half, assuring Artemis Financial that they have incredible security while maintaining enough speed to facilitate the desired user transactions.

***Explain the use of random numbers, symmetric versus non-symmetric keys***

According to Sidhpurwala (2019), random numbers are used in encryption during key generation to guard against predictability-driven attacks. High-quality random number generation techniques increase system entropy and make it difficult for decryption techniques to be deployed.

In regard to symmetric and non-symmetric keys, they each have different uses and different applications (Okeke, 2022). Symmetric keys involve the use of a single key in encryption and decryption operations. This key is private and is shared between both parties. While faster than its non-symmetric counterpart, the secure key must be kept from outsiders, and this proves difficult. Conversely, non-symmetric keys involve a pair of keys – one public and the other one private. The public key is freely shared, and is used by everyone. The private key is never shared, and is used to decrypt the public key’s encrypted information. While this method is more secure than the symmetric mode of encryption, it requires much larger file sizes when the encryption process if finished, resulting in longer data transfer times.

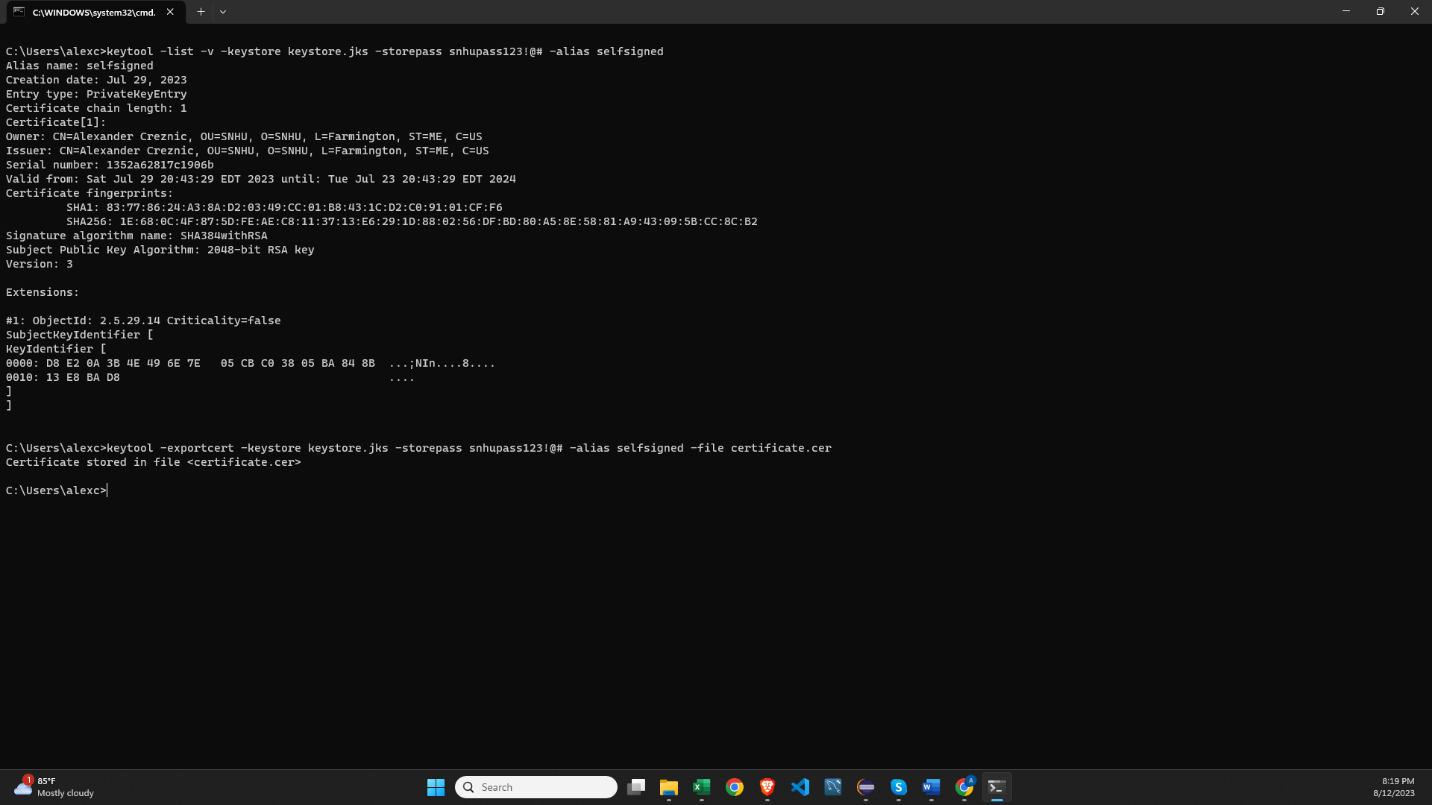
**Describe the History and Current State of Encryption Algorithms**

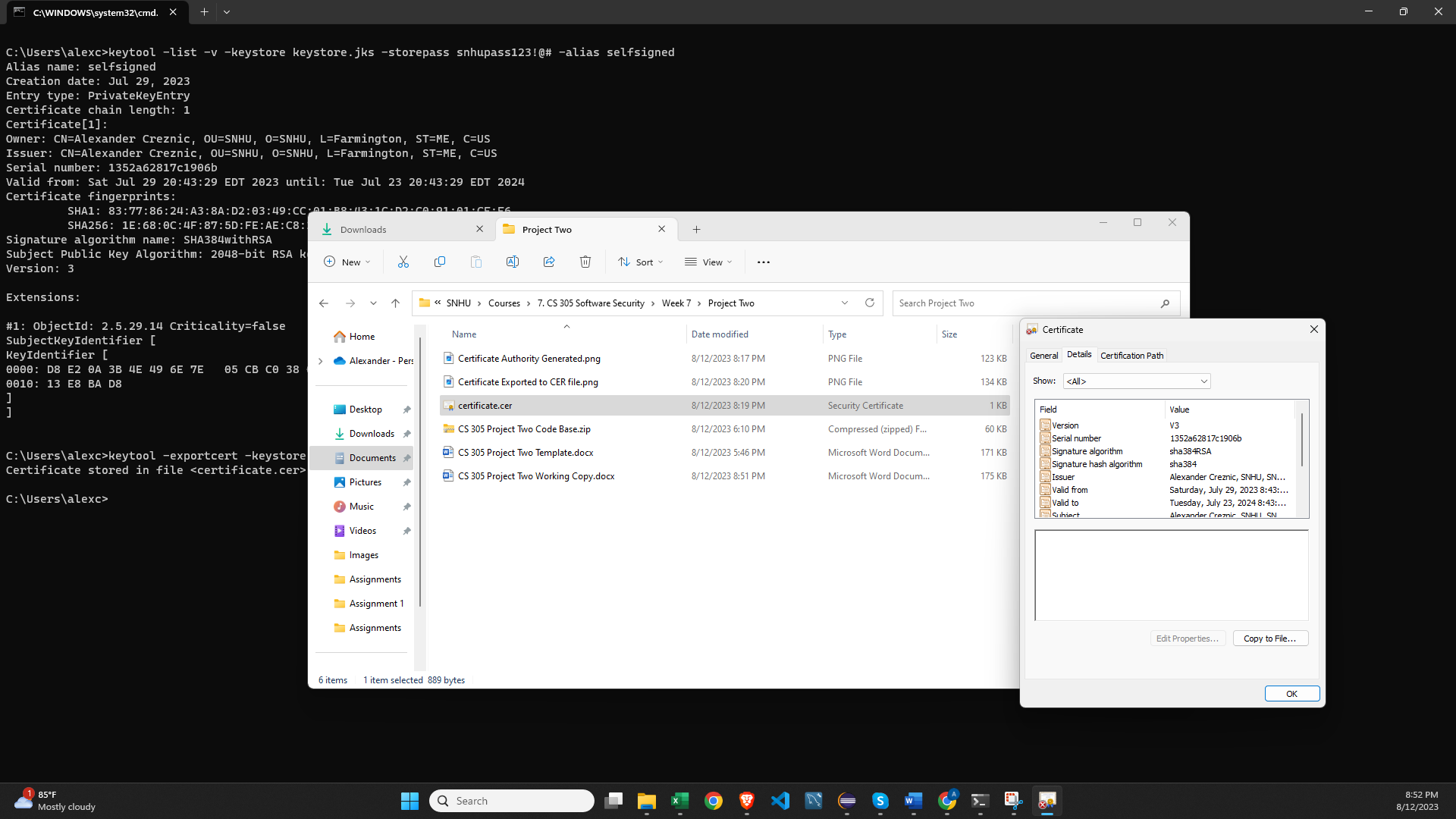
According to Thales (2023), the history of algorithms dates back to around 600 BC, to the ancient Spartan societies. This progressed to the Roman Caesar cipher, that shifts letters in the alphabet and generates novel ways of security letters. Recorded progress takes a further leap into the future in the mid-1500s, with the first recorded key exchange system – an idea we still use today!

The world of encryption continued to progress through the work of Germany’s Arthur Scherbius (ibid), who invented the Enigma machine for encrypting data, which was used during WWII by the Nazi party. In 1945, Claude E Shannon published his seminal paper *A Mathematical Theory of Cryptography* which allowed for secure online communications and further developed cipher capabilities. In the 1970s, the DES algorithm was formed by IBM, and remained as the standard until 1997 (ibid). While this symmetric algorithm was developed, a pair of researchers – Whitfield Diffie and Martin Hellman – developed what would later become the Diffie Hellman key exchange, with a private and public key, opening the world to non-symmetric key pair encryption. Since then, people have been creating and cracking newer versions of these encryption methods, but the core methodology as remained the same.

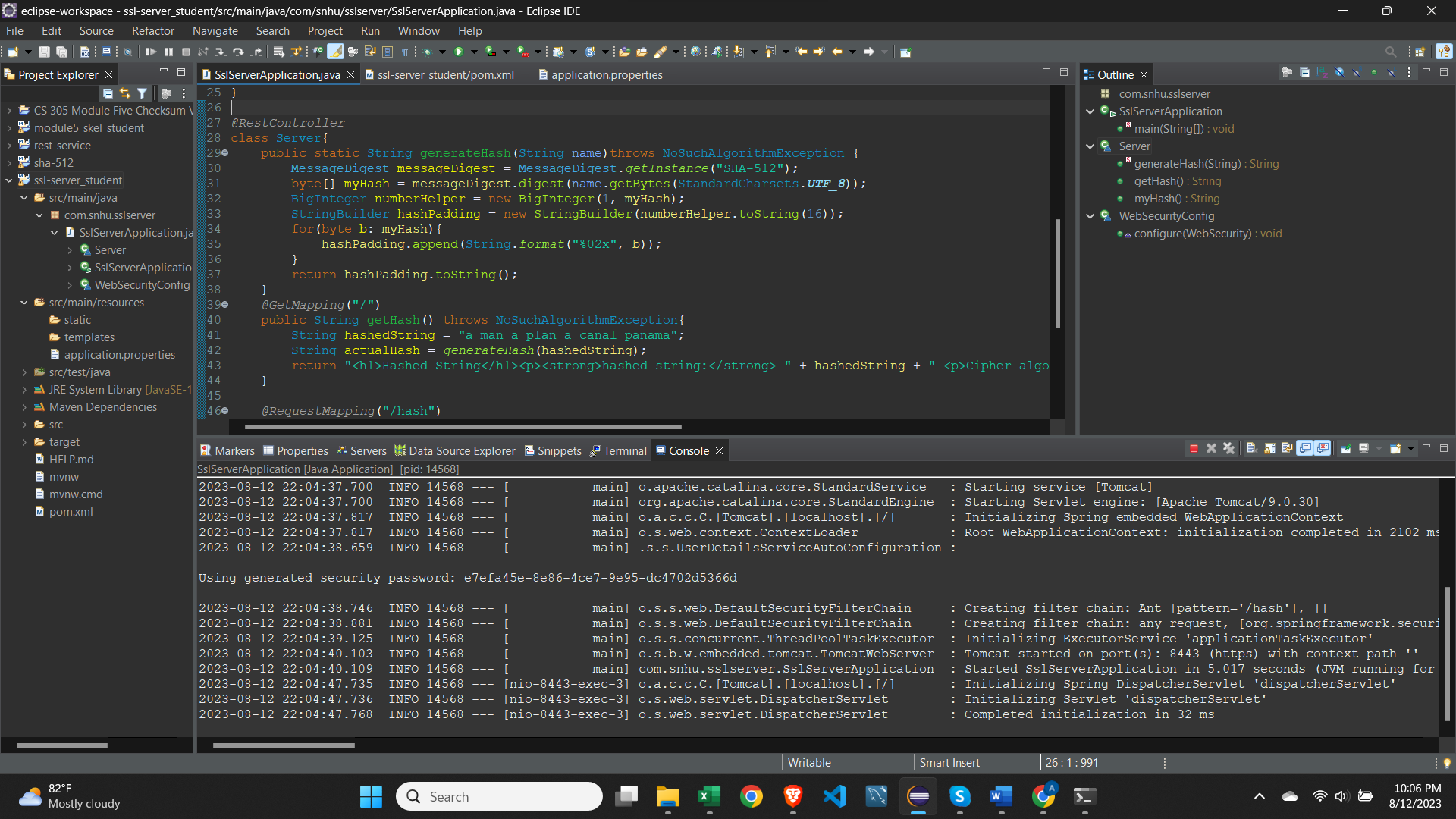
## Certificate Generation

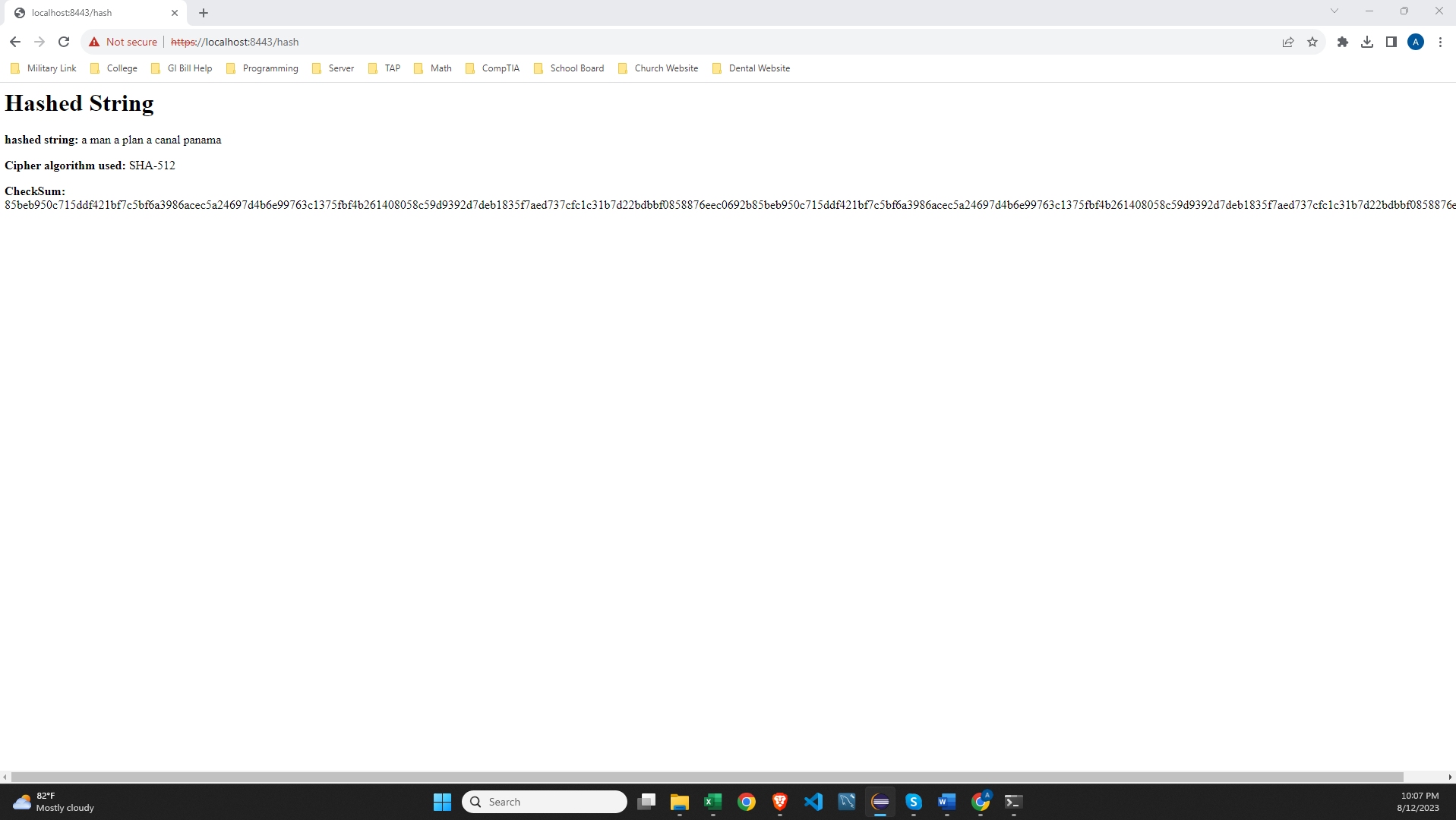
Below is the screenshot of the generated and exported certificate file:



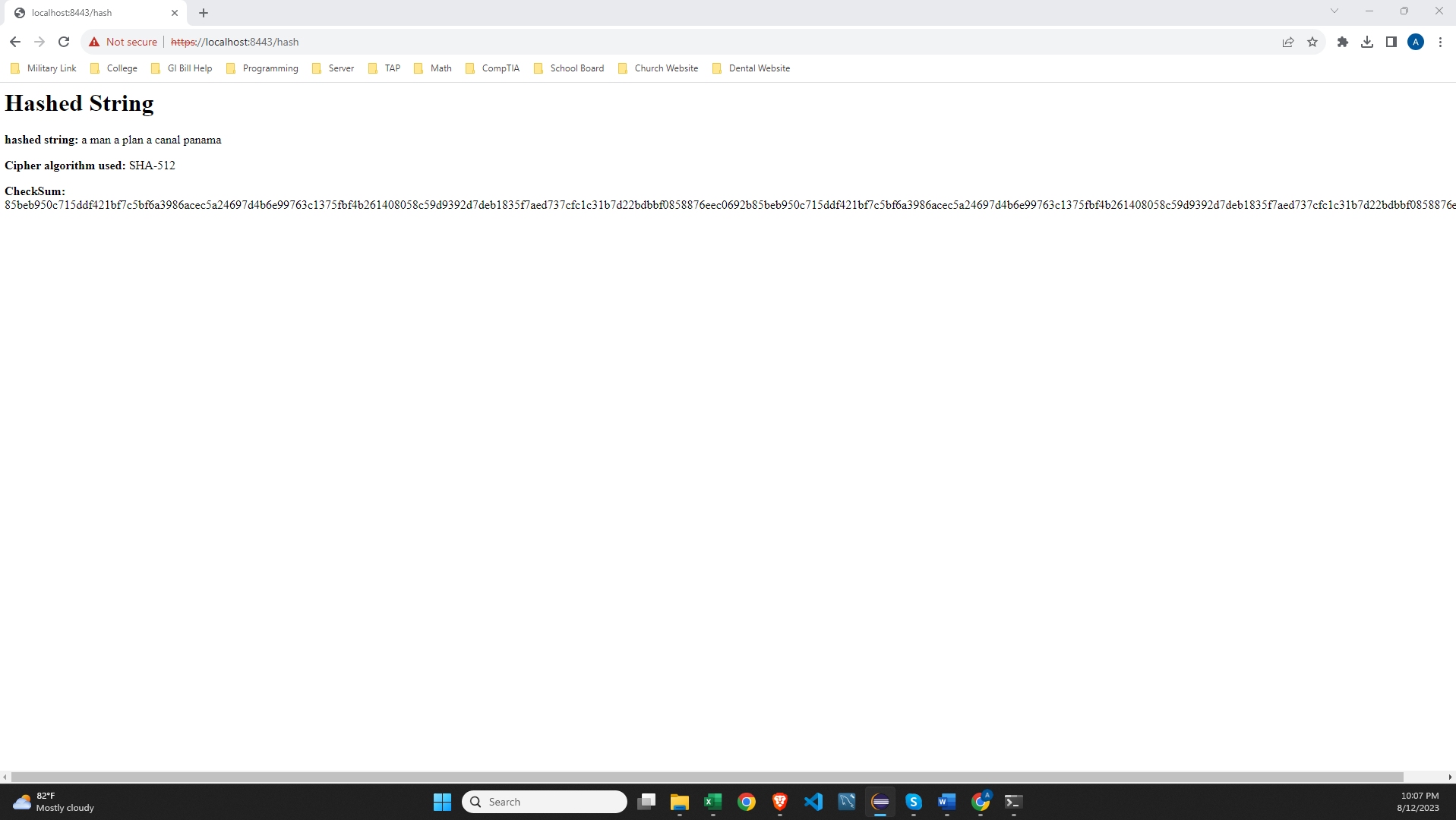


## Deploy Cipher



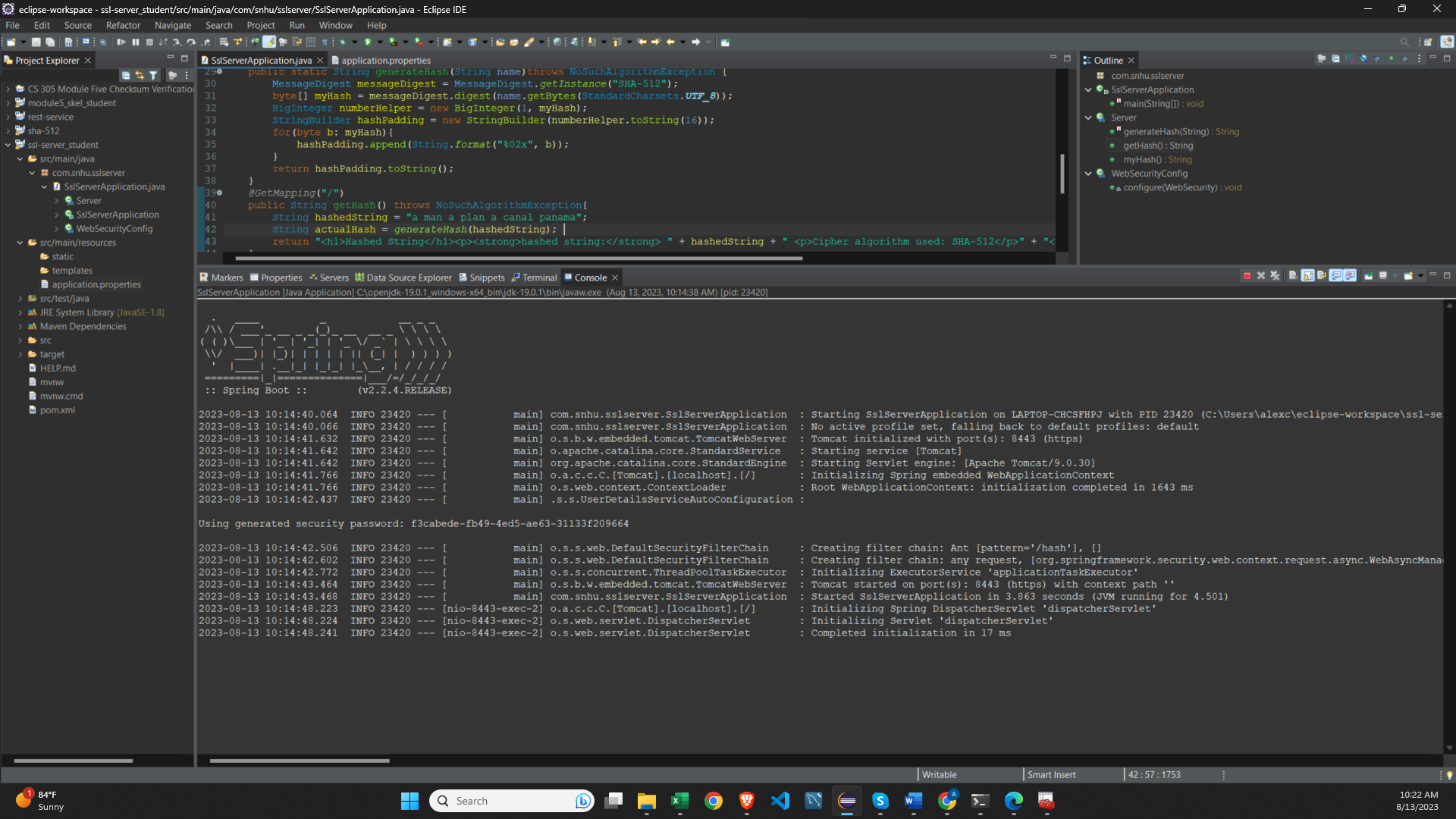


## Secure Communications

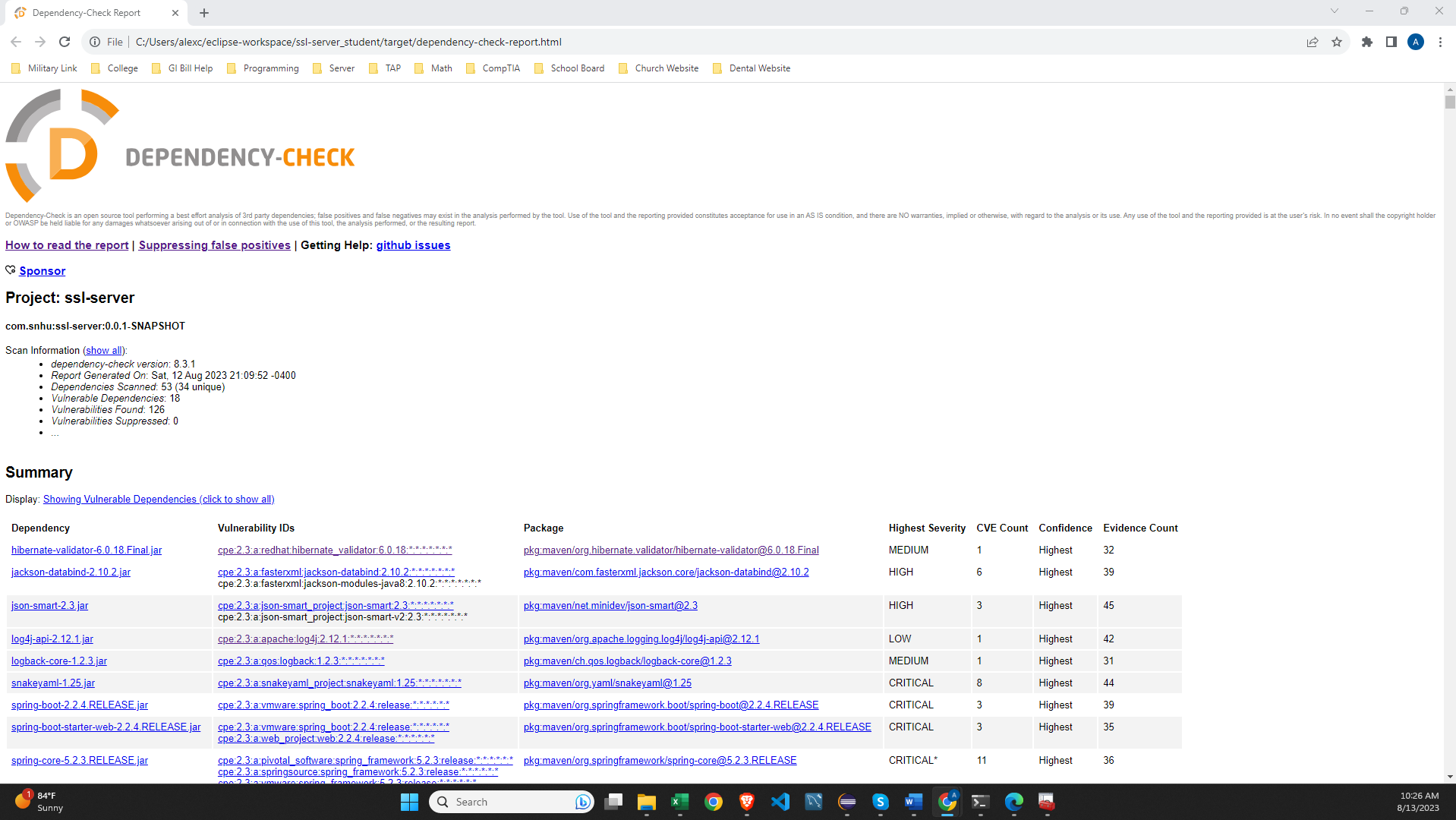
In the image below, the URL displays that the connection occurs over port 8443, and that it used https protocol for connection.

## Secondary Testing

The code below had substantial errors, most of which pertained to the required dependencies that the project needed to import. These were mitigated by importing Springboot security and by running maven update goal.



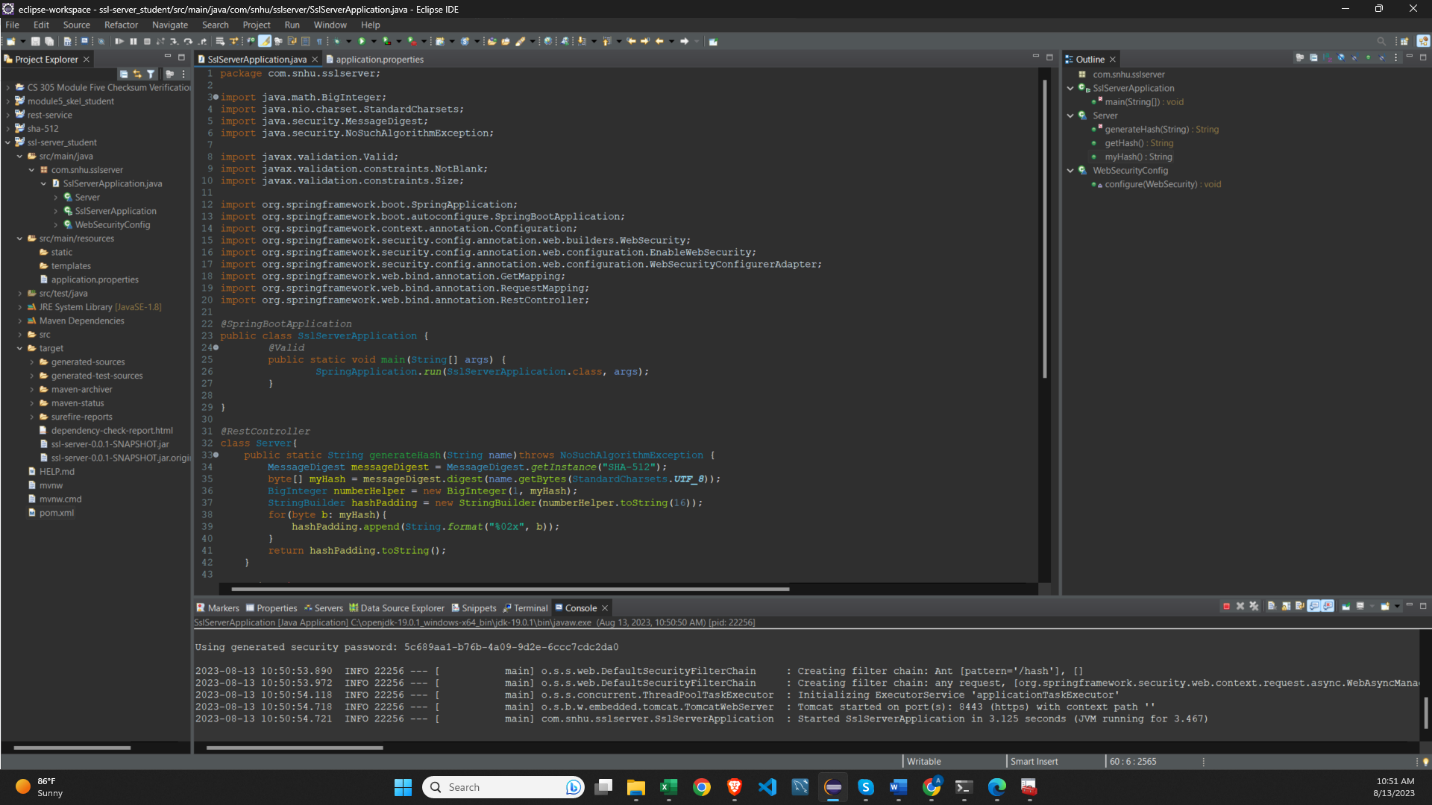
Here is the initial dependency check report.

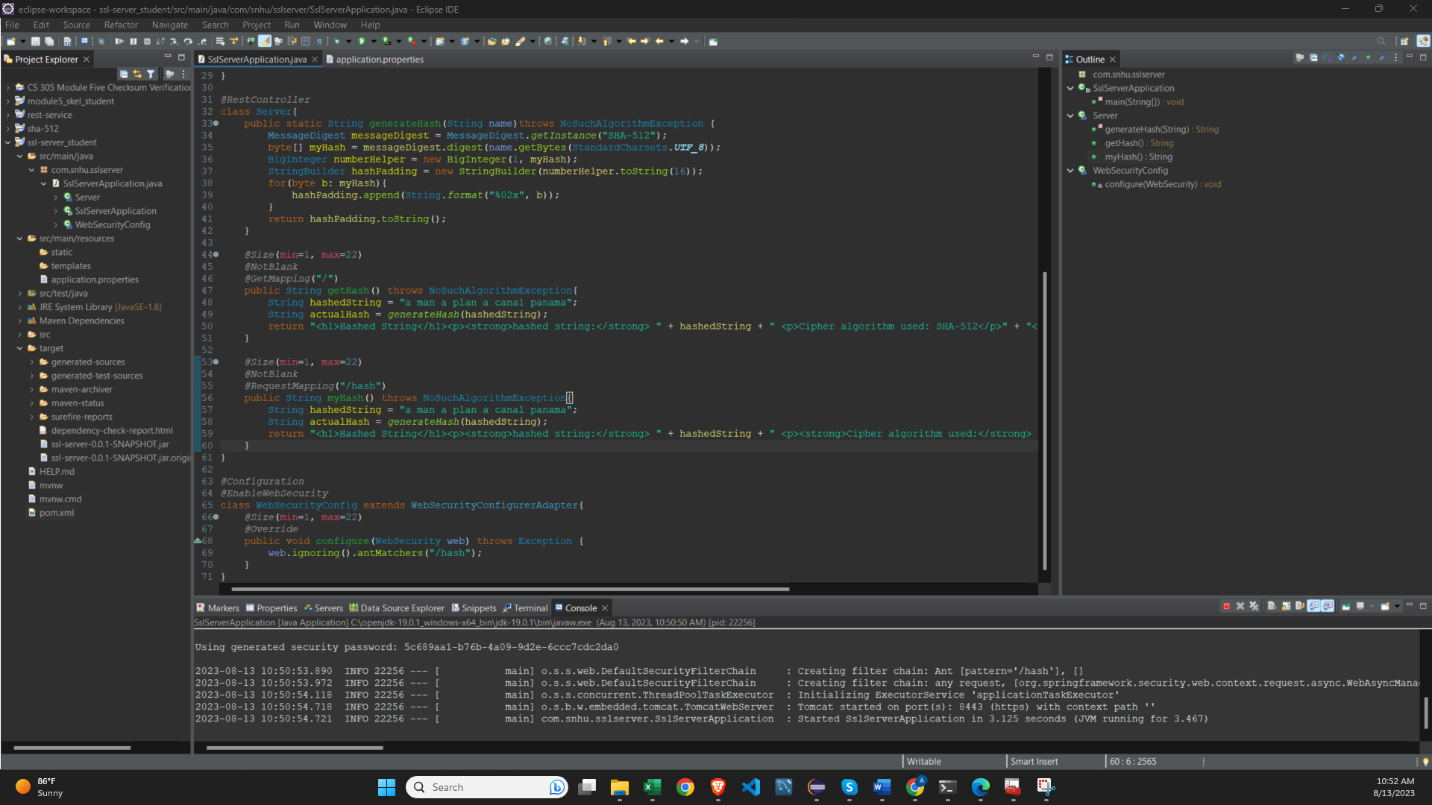


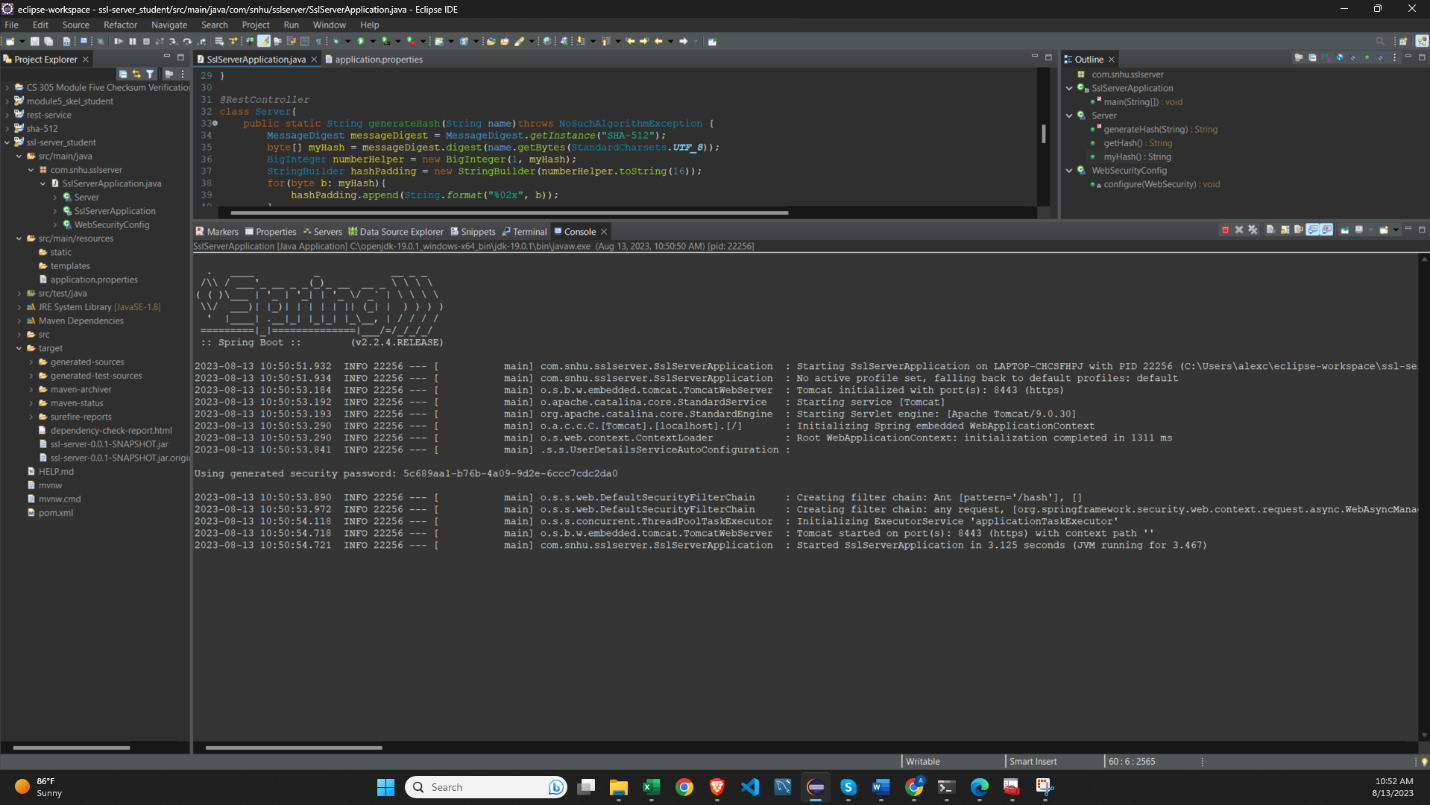
## Functional Testing

Testing of the code revealed that the application runs as needed. It generates a message digest of SHA-512 bit size, and does not introduce any additional vulnerabilities. I added @NotBlank annotations to the methods that accept parameterized values, to ensure that no empty spaces, null values, or blanks insertions are run against the application. Additionally, @Size() annotations were added to control for input above or below desired ranges. Finally, @Valid annotation was added to main() as a safety check.

The first two images below show the annotations as written into the source code. The final image shows the Spring server’s output log data, and that it is running without producing any errors.

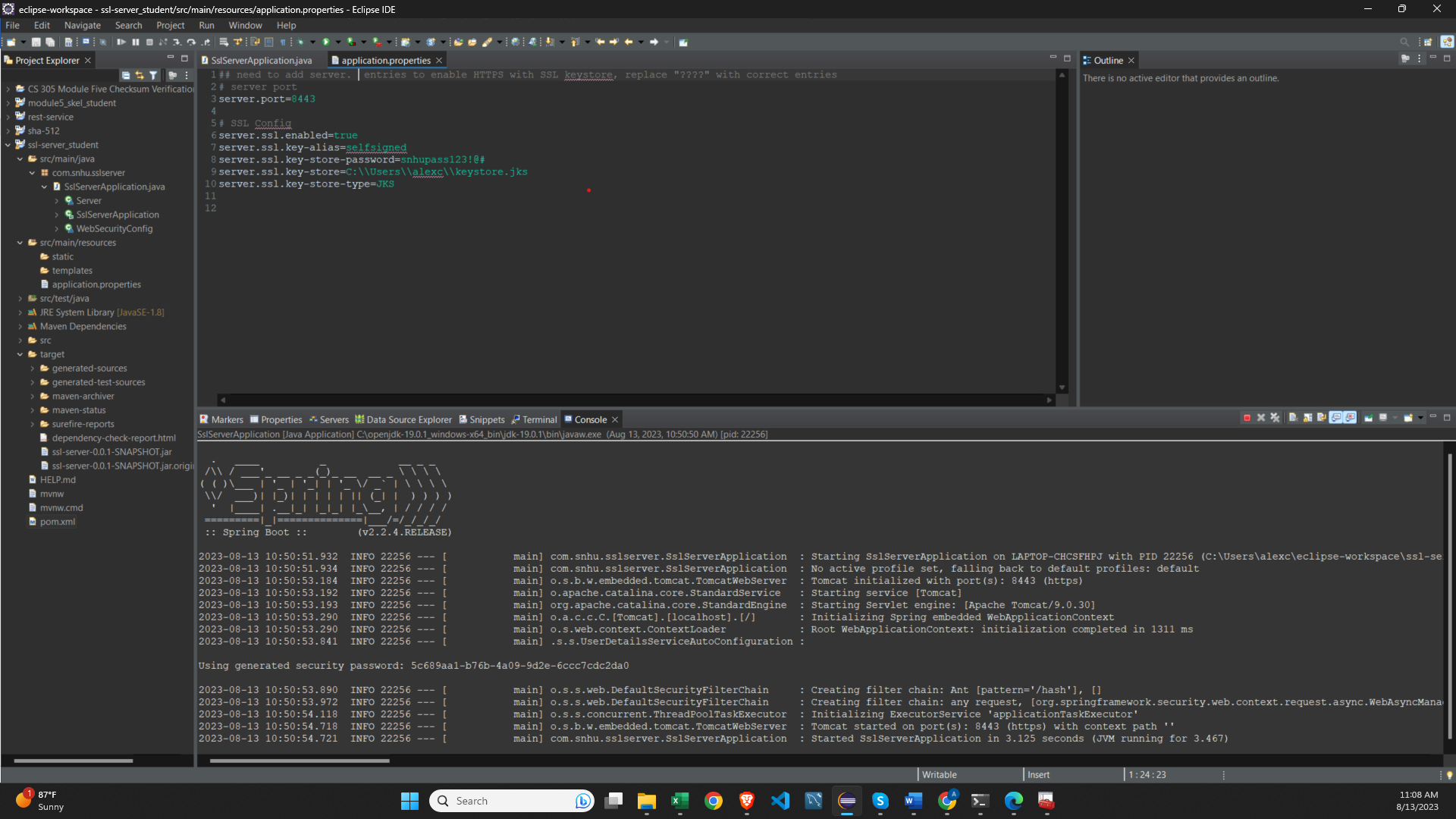






## Summary

The refactored code has been enhanced to provide the required functionality to the client – Artemis Financial. The code focused mostly on security wishes by incorporating industry best practices and utilizing Spring Boot’s built-in security features. The refactored code dovetails nicely with the Vulnerability Assessment Process Flow Diagram. The systemic approach to adding security into the application is as follows:

1. Input validation:
   1. The input validation annotation of @Size(min, max) was added protect the code from SQL injections and cross site scripting attacks. Other annotations like @NotBlank and @Valid were added to validate and sanitize user inputs into the application. Together, these validations guard against bad actors and ensure data is received in valid format.
2. HTTPS with SSL/TLS
   1. The application has been configured to run over HTTPS using SSL/TLS encryption. This makes sure that communication is protected from tampering by an outside party. The application.properties file was upgraded to take a certificate from the java key store and to point the application’s connection through port 8443, for https connection. Please see below:
   2. 
3. Hashing Algorithm Cipher:
   1. The code was upgraded from SHA-256 to SHA-512-bit length, increasing the resistance of the application’s hashing algorithm shields it from collision attacks and from the birthday problem, where any two different inputs result in an identical hash output.
4. Web Security Configuration:
   1. I coded a WebSecurityConfigurerAdapter to configure web security. The specific URL of ‘/hash’ was marked as non-secure for testing purposes, and conforms to the least privilege principle, which grants access only when necessary.
5. Code Sanitation:
   1. The code was tested to follow industry best practices. The testing involved:
      1. Manual code review
      2. Input validation
      3. HTTPS implementation

## Industry Standard Best Practices

Industry standard best practices for inputs and outputs were followed thoroughly and encoded into the application so that common vulnerabilities such as SQL injection and cross site scripting were mitigated. No application can entirely guard against such attacks, but a concerted effort should include baseline defenses against common threats in the digital landscape. Strong authentication and an authorization are required to keep those with a need to conduct legitimate business as the only ones who interact with Artemis Financial’s web products.

The implemented security standards, code formation, and manual testing are crucial for Artemis Financial’s well-being. Securing applications during development builds and maintains trust between Global Rain as a consulting company, and Artemis Financial as a customer. If Artemis Financial is compromised, the damage results in reputation degradation for them and for Global Rain, and lost revenue for both parties. A secure application lowers the risk of data breaches, financial losses, and reputation damage. By following the best practices that were written into this application, Artemis Financial demonstrates their commitment to cybersecurity, data security compliance, customer data privacy, and the ultimately, a safer web experience for the company and its users.

## References

Dobraunig, C., Eichlseder, M., & Mendel, F. (2015). *Analysis of SHA-512/224 and SHA-512/256.* Graz University of Technology, Austria. https://link.springer.com/content/pdf/10.1007/978-3-662-48800-3\_25.pdf?pdf=inline%20link.

Okeke, F. (August 9, 2022). *Asymmetric vs symmetric encryption: What’s the difference?* TechRepublic. https://www.techrepublic.com/article/asymmetric-vs-symmetric-encryption/.

Sidhpurwala, H. (June 5, 2019). *Understanding random number generators, and their limitations, in Linux.* RedHat. https://www.redhat.com/en/blog/understanding-random-number-generators-and-their-limitations-linux.

Thales Staff. (June 10, 2023). *A brief history of encryption (and cryptography).* Thales. https://www.thalesgroup.com/en/markets/digital-identity-and-security/magazine/brief-history-encryption.