****

# 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** | **04/21/2024** | **Bobby Rust** |  |

## 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

Bobby Rust

## Algorithm Cipher

One of the most secure encryption algorithms available today is Advanced Encryption Standard (AES). The AES is supported by the U.S. National Institute of Standards and Technology (NIST) as being approved for protecting sensitive data. It originally came out in 2001 and the mathematical formulas behind the algorithms have shown to have no practical weaknesses thus far. AES-128, AES-192, and AES-256 are all fine choices. As the bit levels increase, both the strength of the encryption against brute force attacks and the computational overhead increase. It can be speculated that some of these lower bit lengths are stronger than the higher ones in non-brute force attacks, though it is mostly just speculation, as it has proven to be extremely difficult to create a mathematical proof.

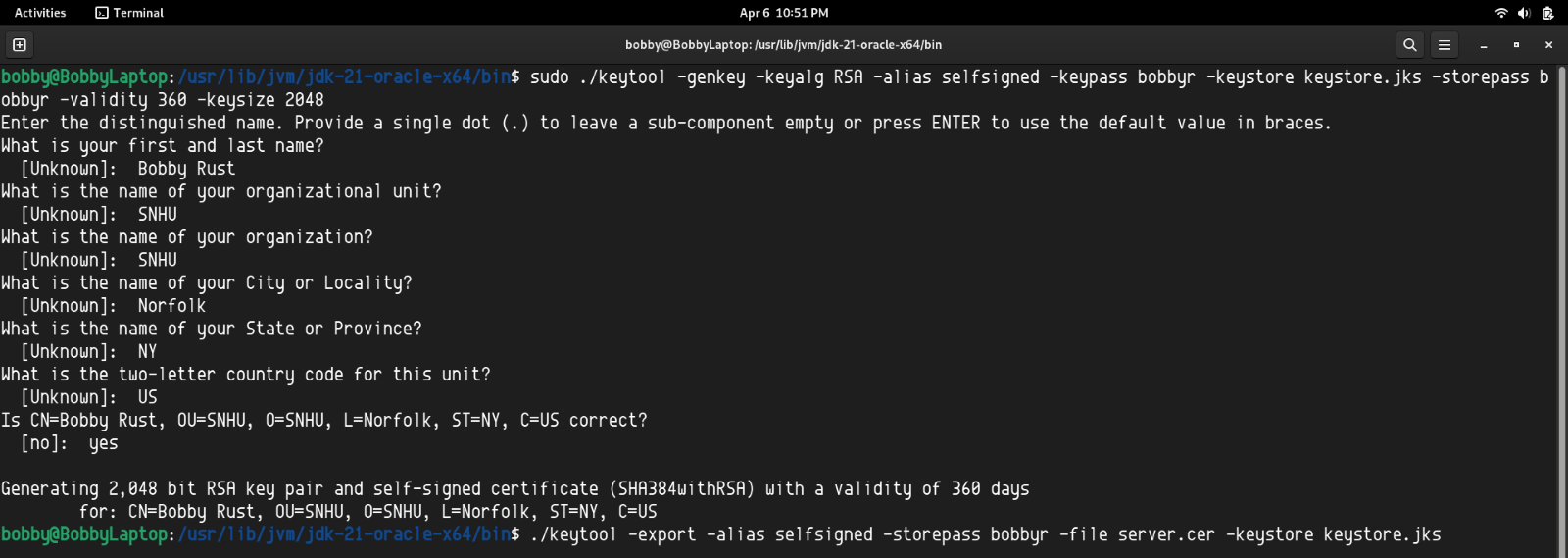
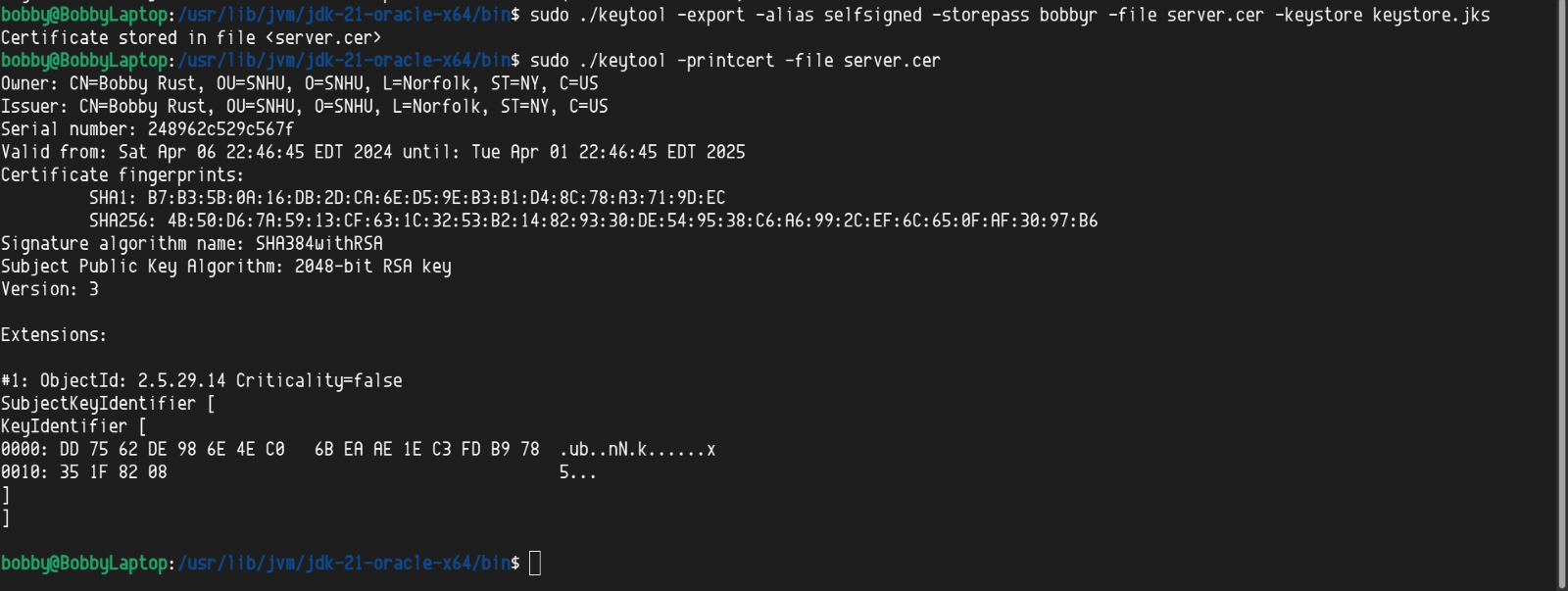
Given Artemis Financials' working with large amounts of customer money, the most robust cipher algorithms that mathematics has discovered must be deployed. AES-256 is approved by the U.S. government to protect TOP SECRET level information. According to the National Institute of Standards and Technology (2003), AES-128 is approved to protect SECRET level information, which is a fair level of protection for Artemis Financials’ needs. However, with any encryption technique, the data is only protected if the key is unknown. AES is very secure against brute-force attacks, but hackers can still gain access to the key which can be used to decipher all the data the key was used to encrypt. Due to this, it is considered best practice to rotate keys annually, which requires re-encrypting all the stored data. Another best practice is to isolate the encryption and decryption process from the application server and the storage server (Manico & Detlefsen, 2014). This way, an attacker with access to the application or storage server will not have access to the keys. In some cases, it is unnecessary to choose the most secure cipher algorithm. One reason for not choosing the most secure cipher available is performance. The more secure the encryption algorithm, the more computational overhead that is introduced. These less-secure algorithms can be appropriate for web application systems that receive a high amount of traffic but aren’t protecting any incredibly sensitive information and are not under any regulation to do so.

Cipher hash functions are an important part of data security. Hash functions and encryption techniques perform different tasks and have different goals. A hash function is an irreversible process which maps an arbitrarily sized input to a fixed-length output. These functions are appropriate for data that does not need to be stored and used in its un-hashed format. For example, a user signing up for a web service must enter a password to keep their account secure. This password should not be stored in the web service’s database. Instead, a salt is generated, then the password along with the salt is hashed. The hashed password resulting from this along with the salt are stored in the database. This causes identical passwords to result in different hashes (Popa, 2018). This way, if an attacker gains access to a database, they cannot derive the user’s password from the hashed password. This is because hash functions are one-way functions. They are deterministic, efficient, and public (MIT, 2023). As such, common passwords can be hashed by anyone using the popular SHA-256 hashing algorithm. These common passwords in hashed form can then be compared to the hashes stored in the database to derive the plaintext passwords. The bit levels, which are appended at the end of cryptographic security functions, denote the fixed length of the output. In other words, given any input of arbitrary but bounded length, SHA-256 or AES-256 will produce 256-bit outputs. The AES algorithm is a symmetric key algorithm, meaning that the same key is used for both the encryption and decryption of data. This is opposed to an asymmetric key algorithm, where there are different keys for encrypting and decrypting data. In the case of AES, inputs are to be hashed prior to encryption. This randomizes the bits in a statistically uniform distribution, while also changing the size of the input data to be the correct size for the encryption function (Ryan, 2018).

One of the earliest known cryptographic methods used for encrypting and decrypting data was created by Greek historian Polybios (Roberts, 1998). Cryptography has grown increasingly complex since the earliest methods, much of that from the effects of modern computing. Modern cryptography in computer science is held secure by the P versus NP problem. This problem argues whether there exists an algorithm that solves any non-polynomial problem in polynomial time. Cryptographic functions are made to be problems that cannot currently be solved in polynomial time, thus requiring what are currently unreasonably large amounts of computing power and/or time to crack. The future of computing power cannot be predicted. Breakthroughs in mathematics and fields like quantum computing have the potential to make current cryptographic algorithms useless (Lotsvin, 2017). As a result, the cryptography field will surely go through massive changes in the future and can change the digital world permanently (Rust, 2024).

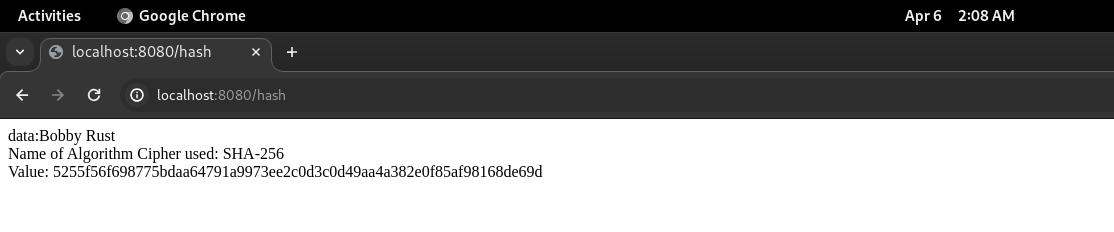
## Certificate Generation

Insert a screenshot below of the CER file.

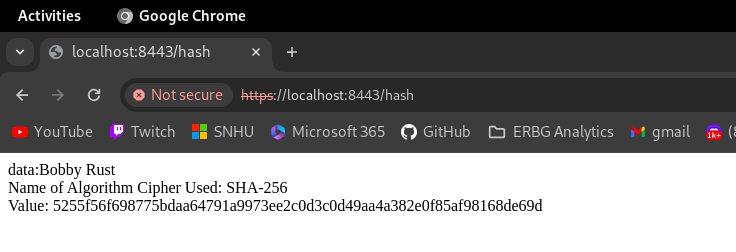
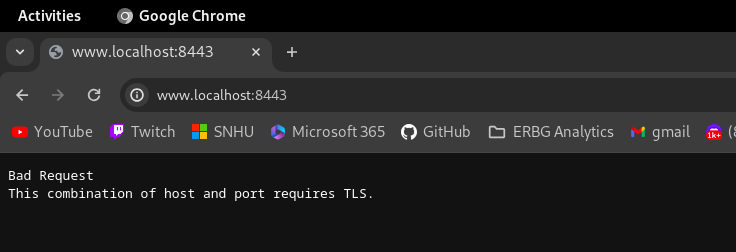
## Deploy Cipher

Insert a screenshot below of the checksum verification.



## Secure Communications

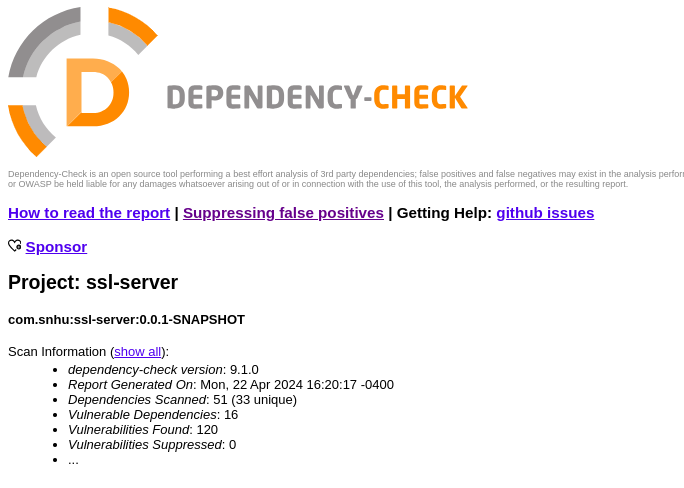
Insert a screenshot below of the web browser that shows a secure webpage.

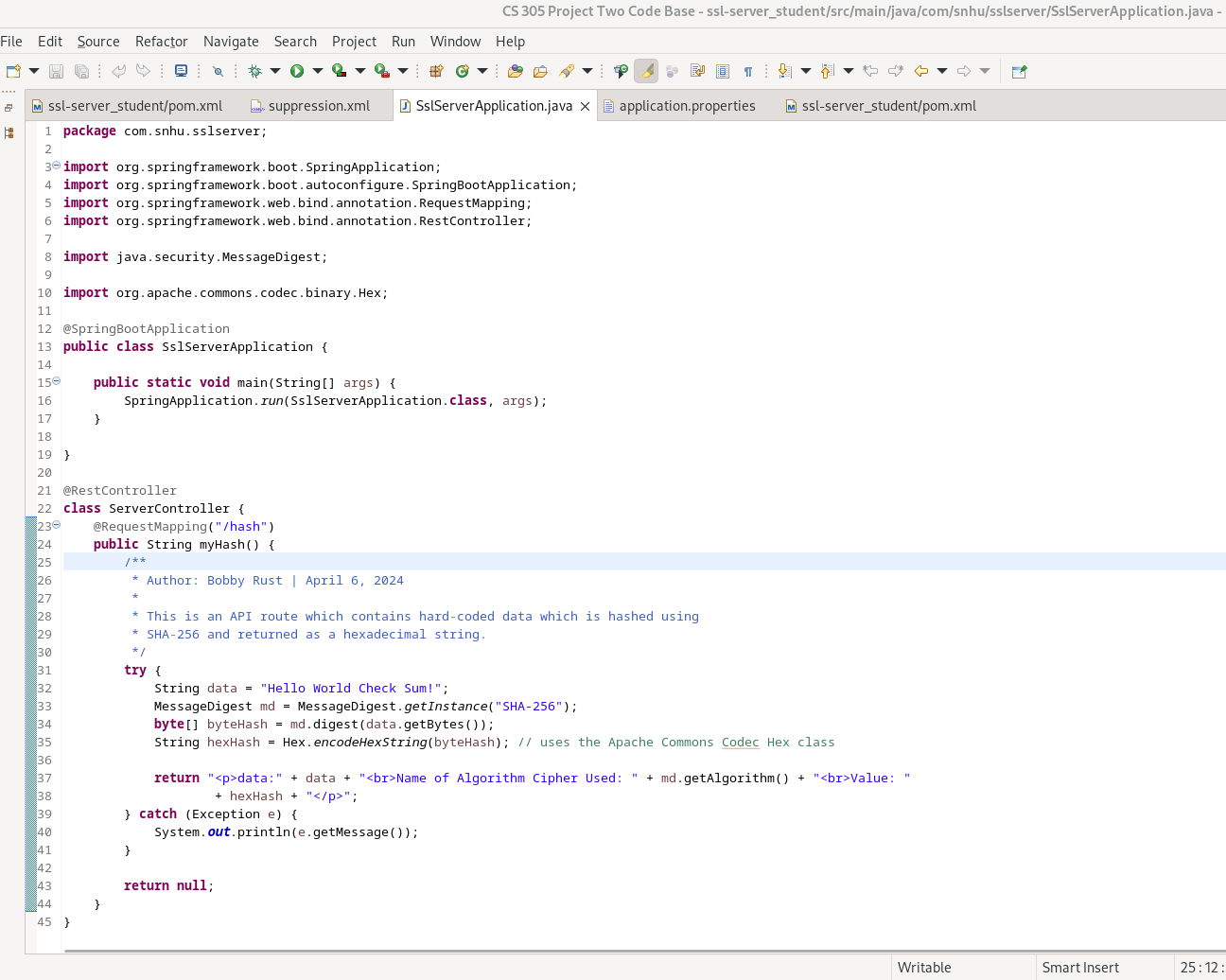
When HTTPS is not used, the server does not accept the request. Even with using HTTPS, the web browser notifies us that the server is not secure as it does not have a CA certificate.

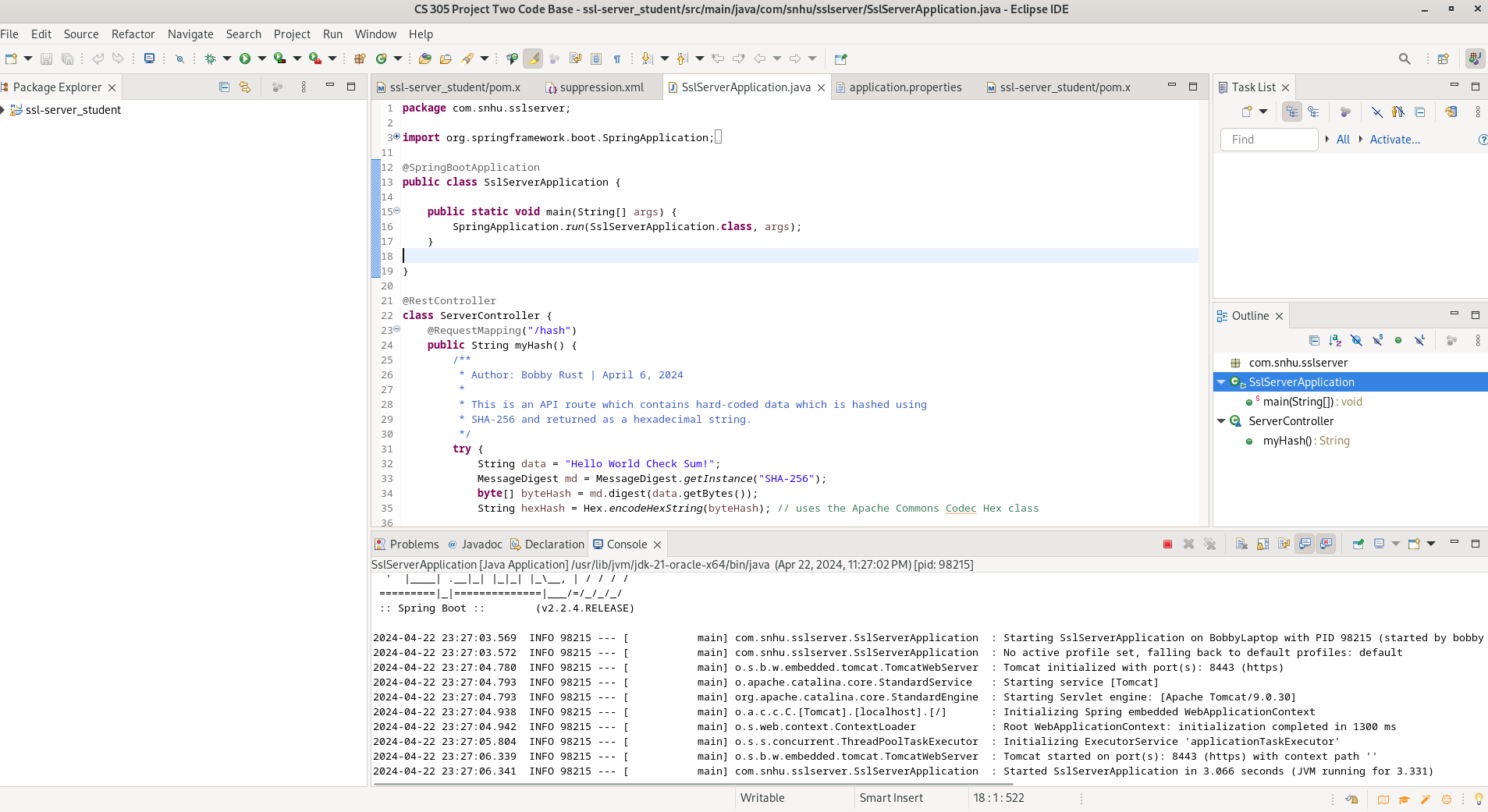
## Secondary Testing

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

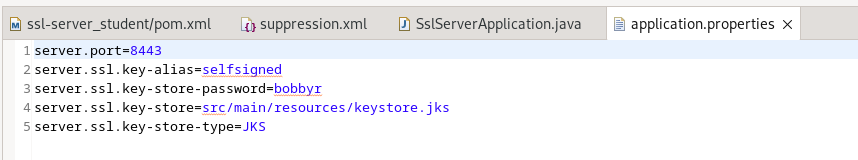
**Dependency Check Before Refactoring**

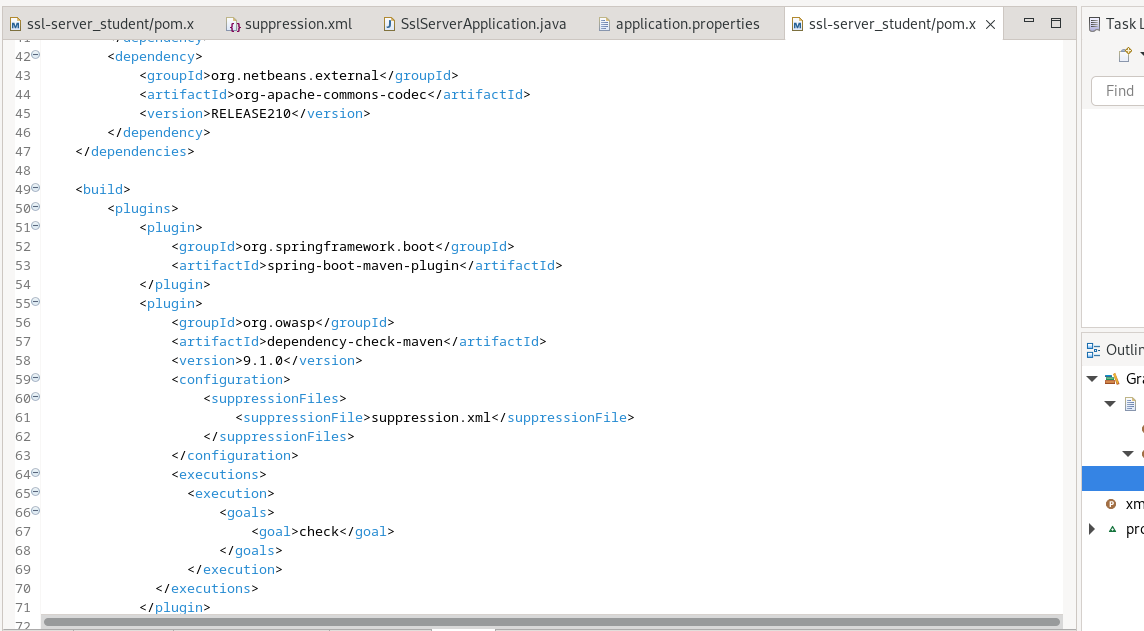


**SslServerApplication.java**

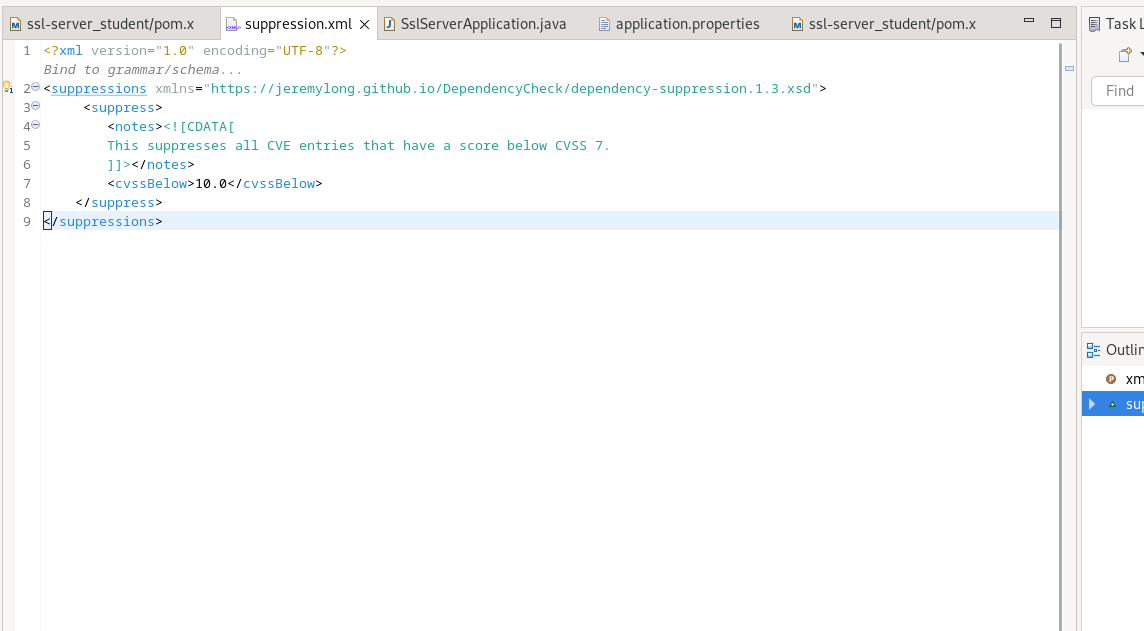
**Running Without Errors**

**Application.properties**

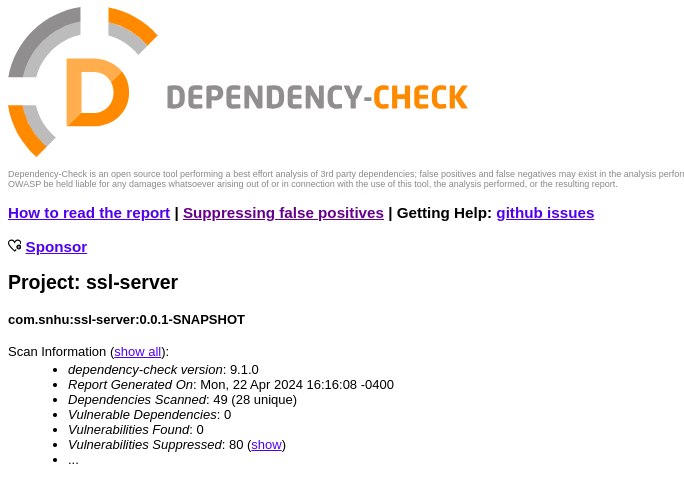


**Pom.xml**

**Suppression.xml**



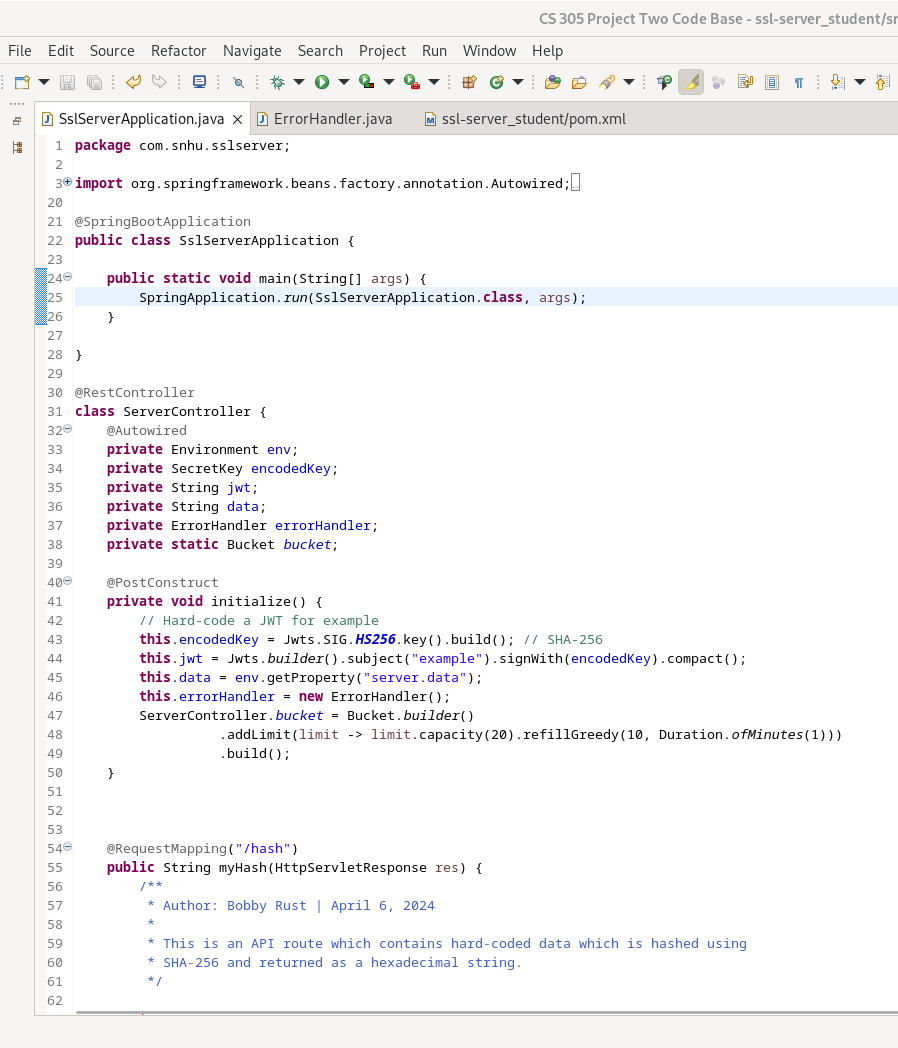
**Dependency Check After Refactoring**

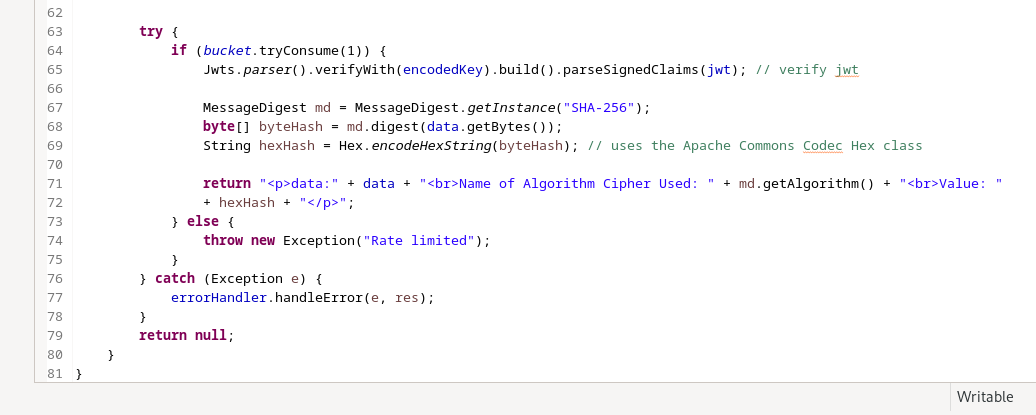


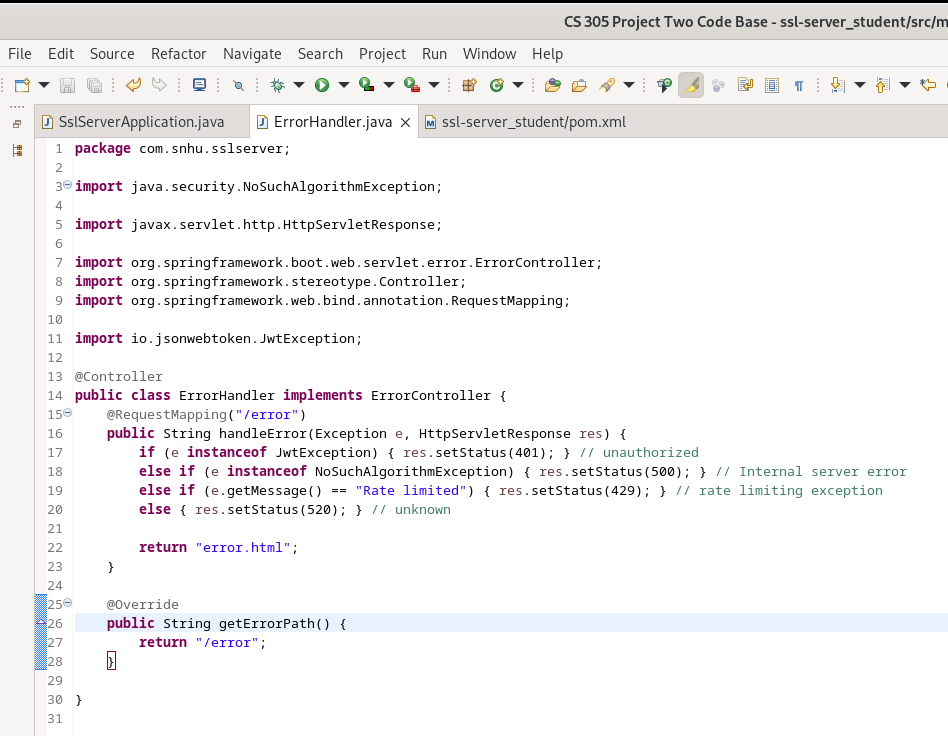
## Functional Testing

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

**SslServerApplication**



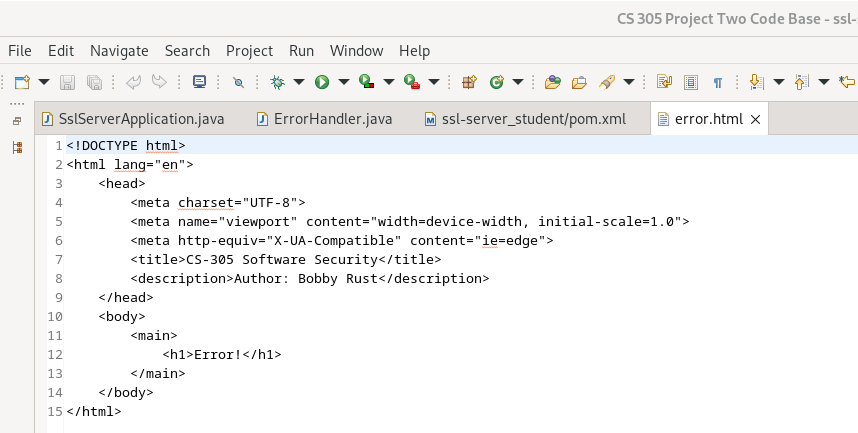
**ErrorHandler**



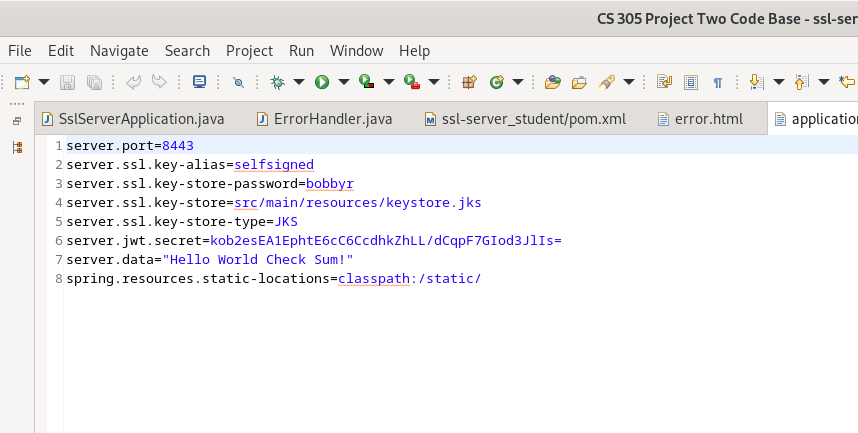
**Pom.xml Changes**

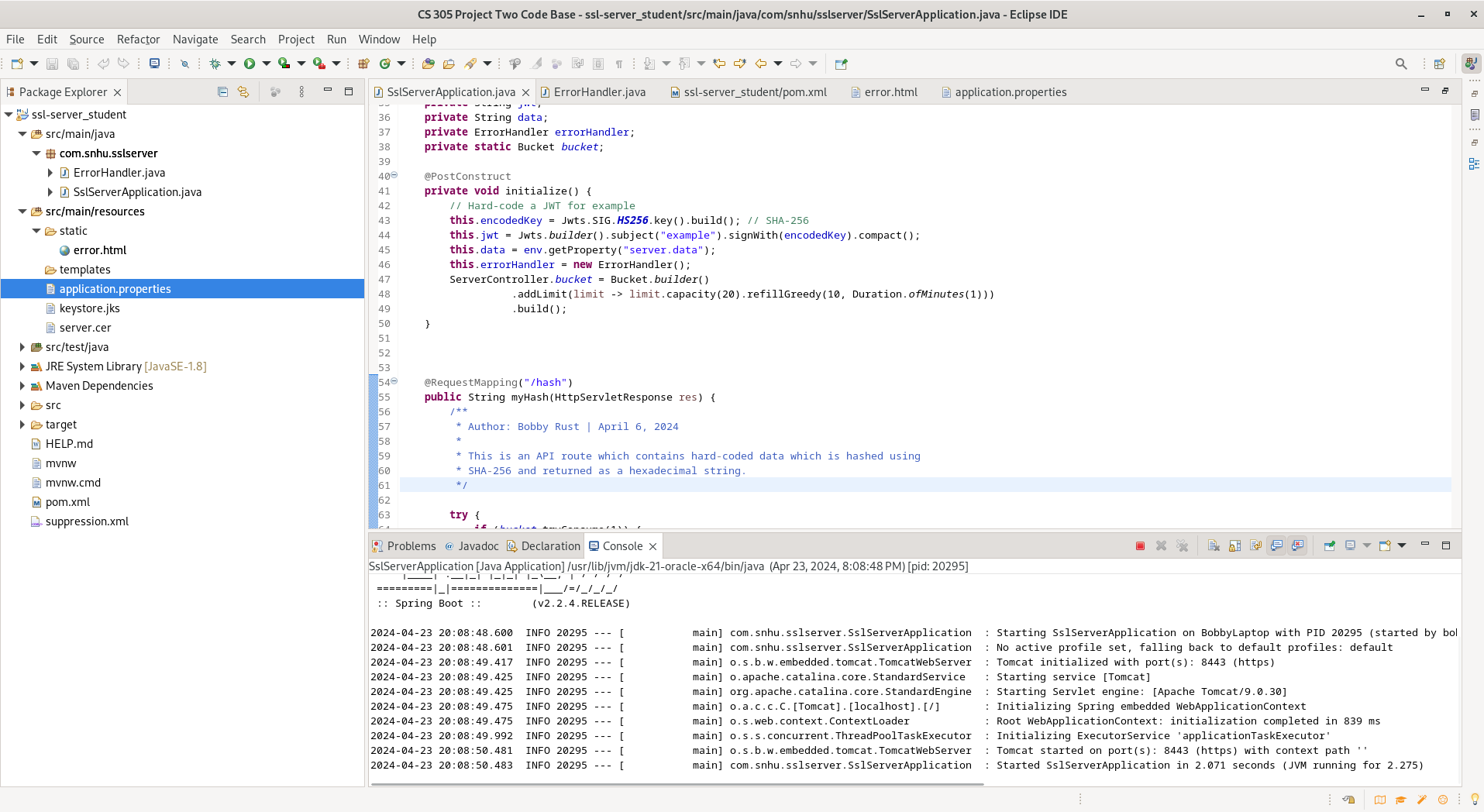


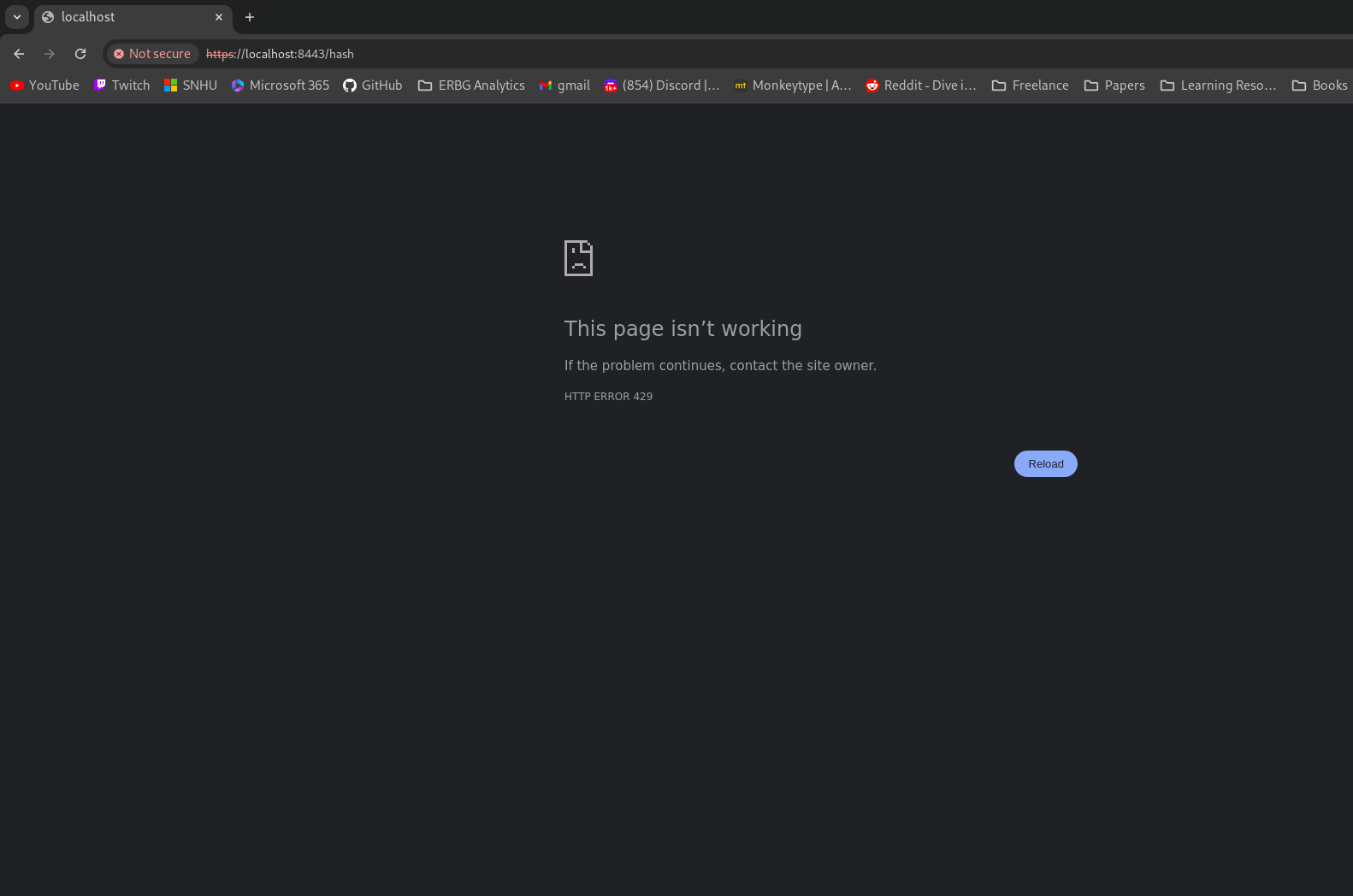
**Error.html File**

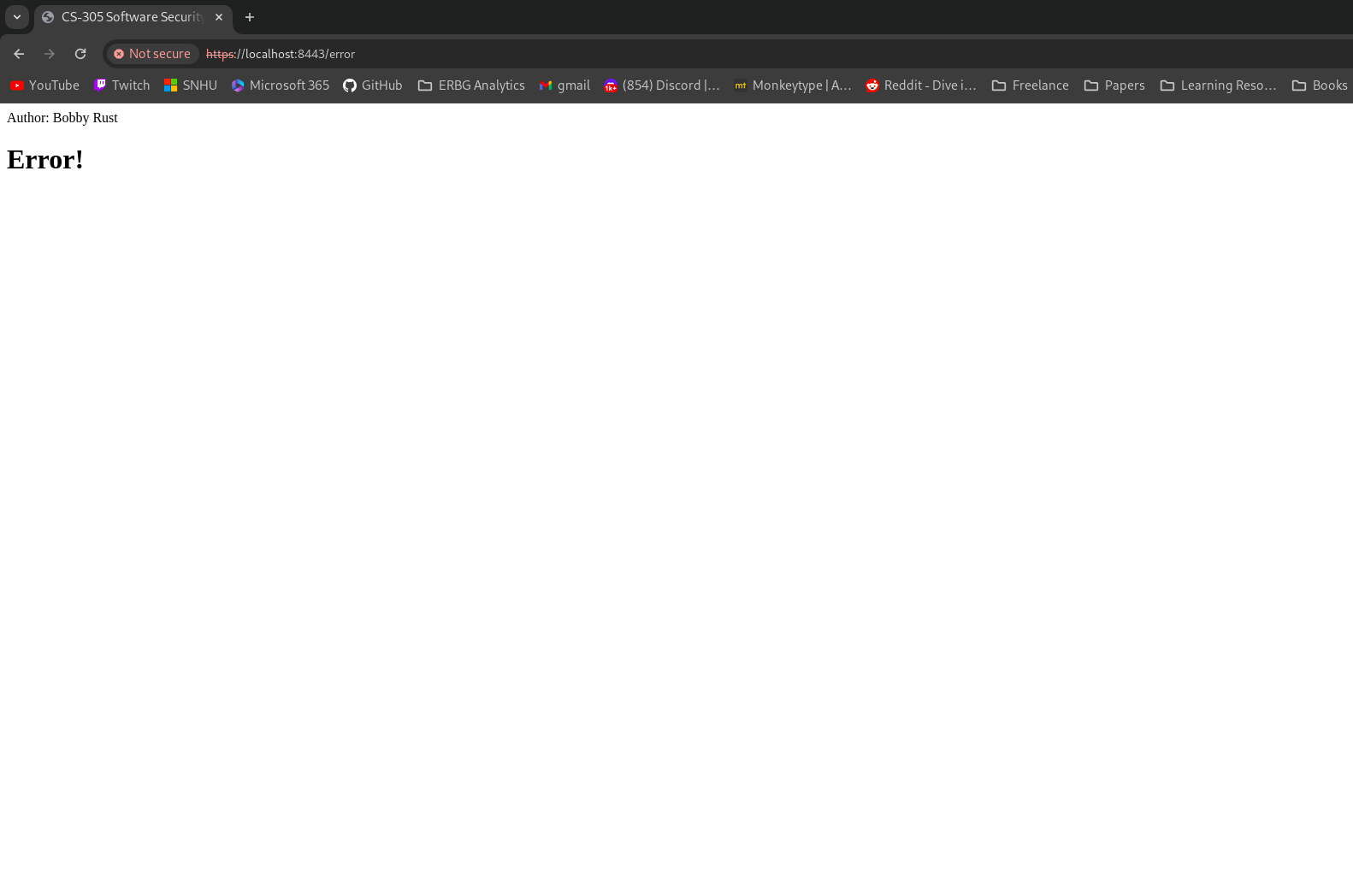


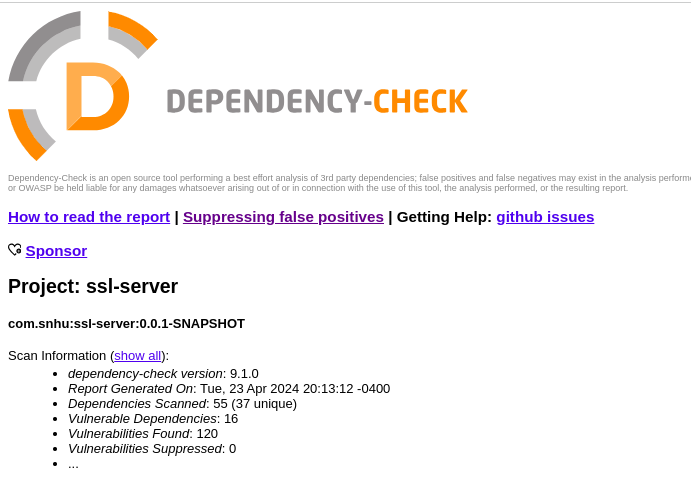
**Application.properties**

**Code Running**



**Rate Limiting in Effect**

**Error Page**

**Final Dependency Check**

## Summary

## Due to Artemis Financials' strict security requirements, a robust security protocol was implemented. The checksum generation is used to validate the integrity of the message and confirm that it is authentic. Another way to secure a web application efficiently is the implementation of JSON Web Tokens (JWTs). These tokens are used for authentication and authorization without the need to make additional API calls with each request. JWTs are securely signed with a private token and stored on the client side to be automatically sent with each request. A valid JWT authenticates and/or authorizes the user to perform the requested function (Auth0 by Okta, 2024).

## Next, the data stored in the server was removed from the code file and added to the application.properties file. This stores the data as an environment variable, meaning the data can only be accessed inside the environment in which the server is running, which adds an additional layer of protection for attackers looking to gain access to the data. Using environment variables instead of hard coding the data into the application prevents it from being uploaded to version control which may be accessed by a malicious party.

## Handing errors was the next step in securing the application. When errors are thrown, it may crash the server, which can be leveraged for a Denial-of-Service attack. Proper error handling also prevents information disclosure. If a development debug message is shown to users, it may give away critical information about the application that can be exploited. Implementing an ErrorController class in Java is an easy way to handle errors securely. Only the necessary and non-sensitive information about the error is shown to the user when they occur, and they are handled gracefully to prevent crashes.

## Another vector for a Denial-of-Service attack is flooding the server with requests. To prevent this, rate limiting was implemented which slows down these attacks. The token bucket algorithm was used for the rate limiting via the Bucket4j library (bucket4j, 2024).

## Industry Standard Best Practices

Following industry standard best practices is extremely important to securing web applications. There are many known security vulnerabilities and many security experts working on fixes. With new security vulnerabilities being discovered daily, security professionals must constantly look to fix these problems. One of the easiest ways for a hacker to gain access to a system is by exploiting previously found flaws, as it is much more difficult to find new ones. The OWASP dependency check tool for Java was used to automatically check for known security flaws in all of the Java application’s dependencies. This prevents having to go through and manually search for vulnerabilities in each library. Moreover, vulnerabilities change per dependency based on its version. It becomes extremely difficult to track all of these changes. Using a process flow diagram and manually reviewing code is another way to ensure the security of an application. Going through the diagram, it is helpful to think about the application’s functionality and whether that category of security applies. In this application, many of these categories applied such as cryptography, error handling, client / server management, and encapsulation. Allotting time to review the security of an application is necessary as its importance demands its very own step in the process of development. Security vulnerabilities can cause tremendous damage. Firstly, each moment of downtime in the application is lost capital. This downtime can also jeopardize a company’s reputation, especially when dealing with sensitive information such as financial data.

**References**

Rust, R. (2024). *Algorithm Ciphers.* Southern New Hampshire University.

Auth0 by Okta. (2024). *Introduction to JSON Web Tokens.* <https://jwt.io/introduction>

Bucket4j. (2024). *Java rate limiting library based on token-bucket algorithm.* https://github.com/bucket4j/bucket4j