****

# 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** | **10/20/24** | **Sarah Wagner** | **Final Draft with screenshots** |

## Client



## Instructions

Submit these 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

Sarah Wagner

## Algorithm Cipher

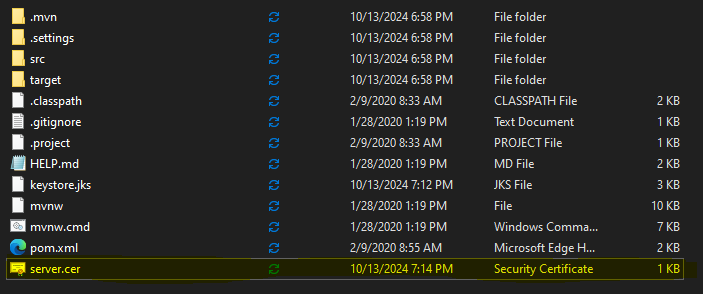
For Artemis Financial’s project, I recommend using the **AES (Advanced Encryption Standard)** cipher. AES is one of the most trusted encryption methods out there and is commonly used to protect sensitive information like financial data. It works by breaking data into blocks of 128 bits and uses a key size of either 128, 192, or 256 bits, which gives it a high level of security. Since AES is approved by the National Institute of Standards and Technology (NIST), it’s a solid choice for ensuring that data is safe when it's being transferred or stored.

AES uses what's called symmetric encryption, meaning the same key is used to both encrypt and decrypt the data. This makes it a fast and efficient option, especially when you’re dealing with a lot of information, like financial records. To make it even more secure, AES includes random numbers and an initialization vector (IV) to prevent attackers from exploiting any patterns. For added protection, we can pair AES with a hash function like SHA-256, which will allow Artemis Financial to verify data integrity through checksum verification, making sure the data hasn’t been tampered with during transfer.

AES was developed to replace the older and weaker DES (Data Encryption Standard), which became vulnerable to attacks over time. Today, AES is widely used in everything from online banking to government security and is constantly reviewed to ensure it remains secure, making it a great choice for Artemis Financial's modernization efforts.

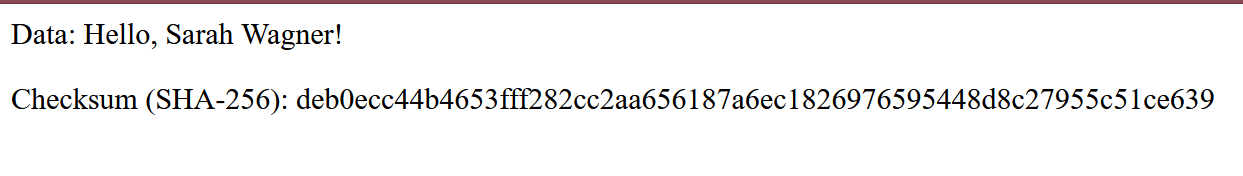
## Certificate Generation

Insert a screenshot below of the CER file.



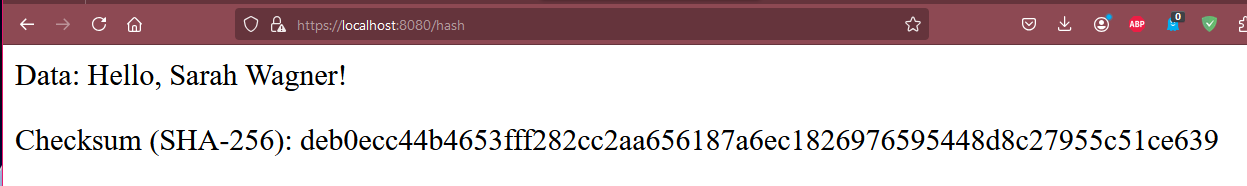
## Deploy Cipher

Insert a screenshot below of the checksum verification.



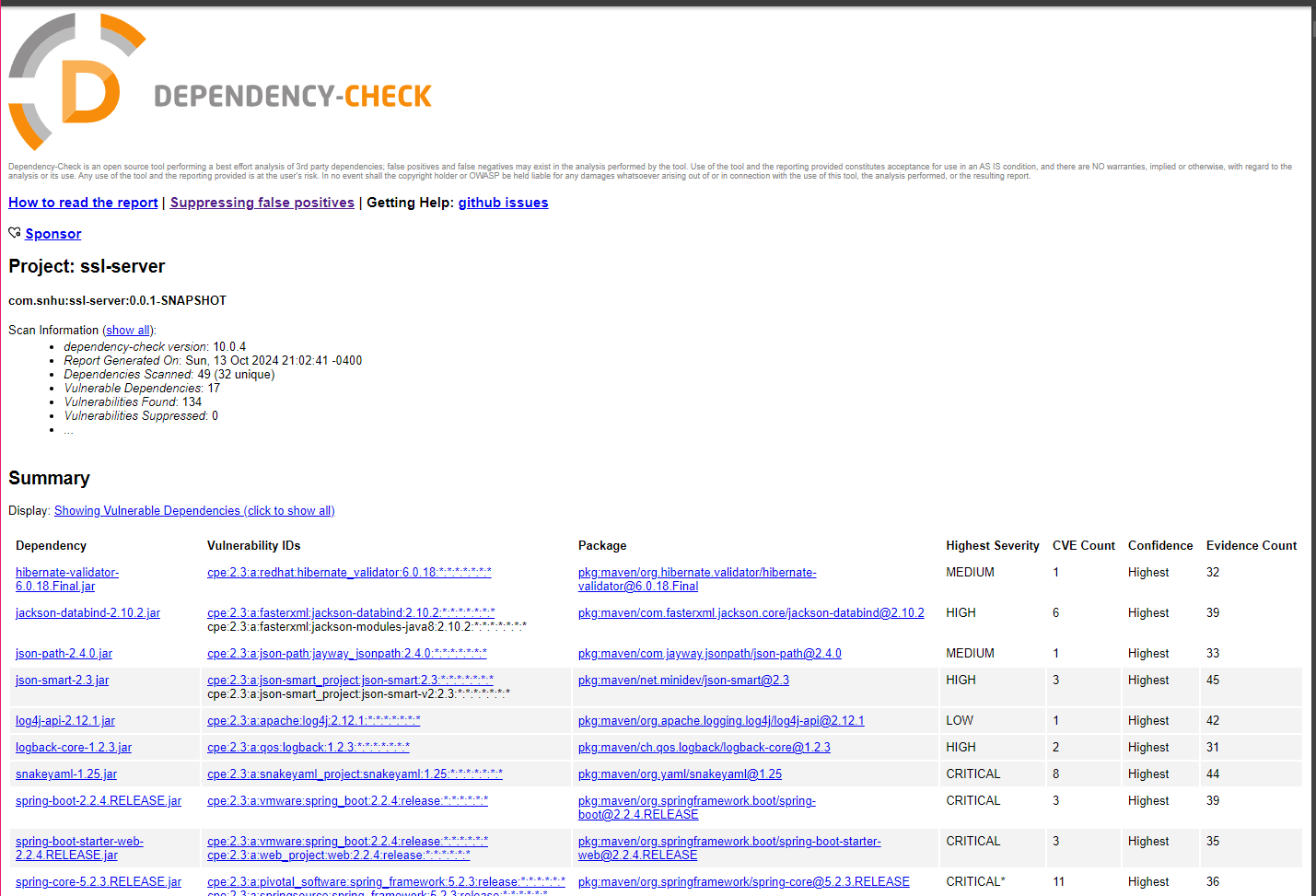
## Secure Communications

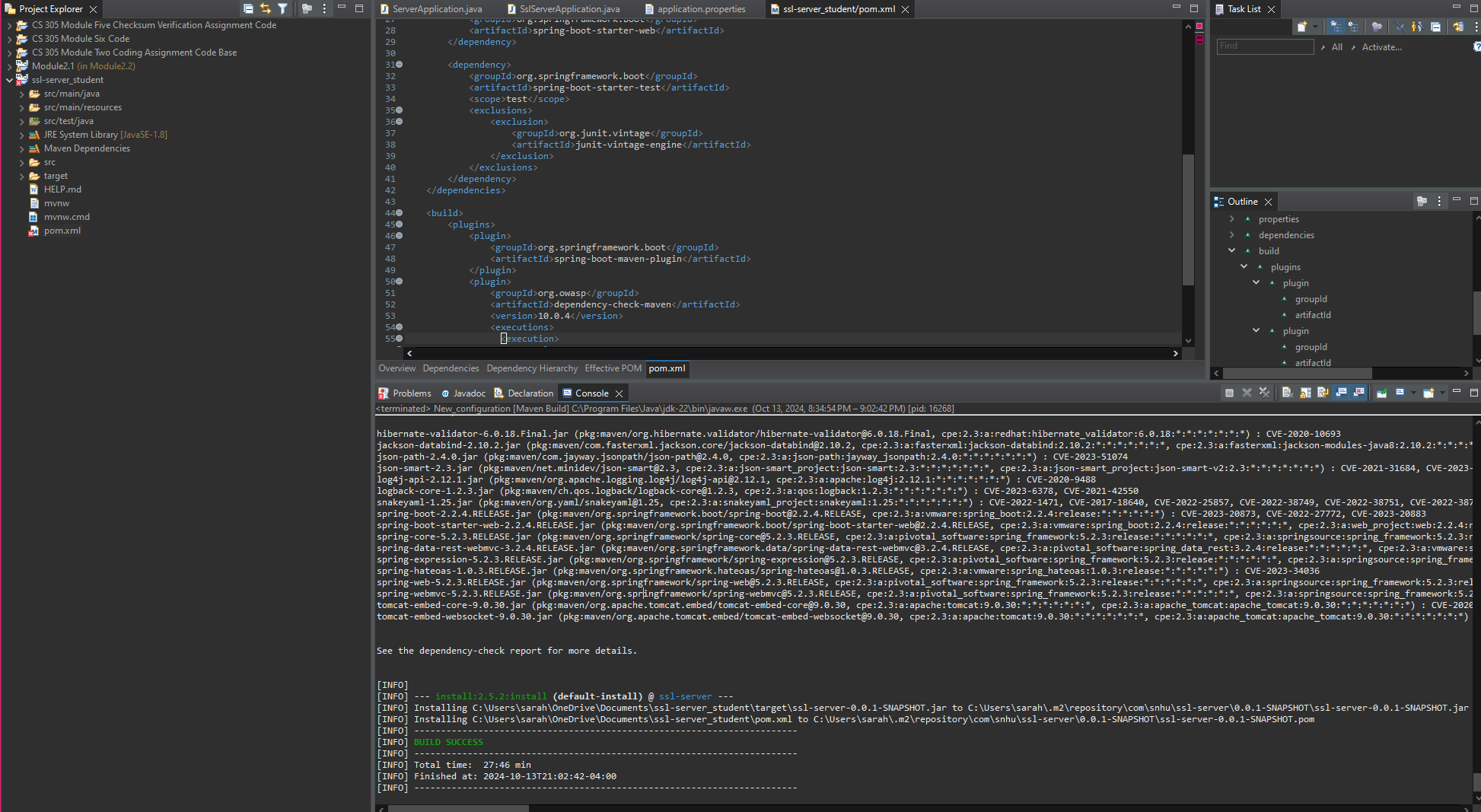
Insert a screenshot below of the web browser that shows a secure webpage.



## Secondary Testing

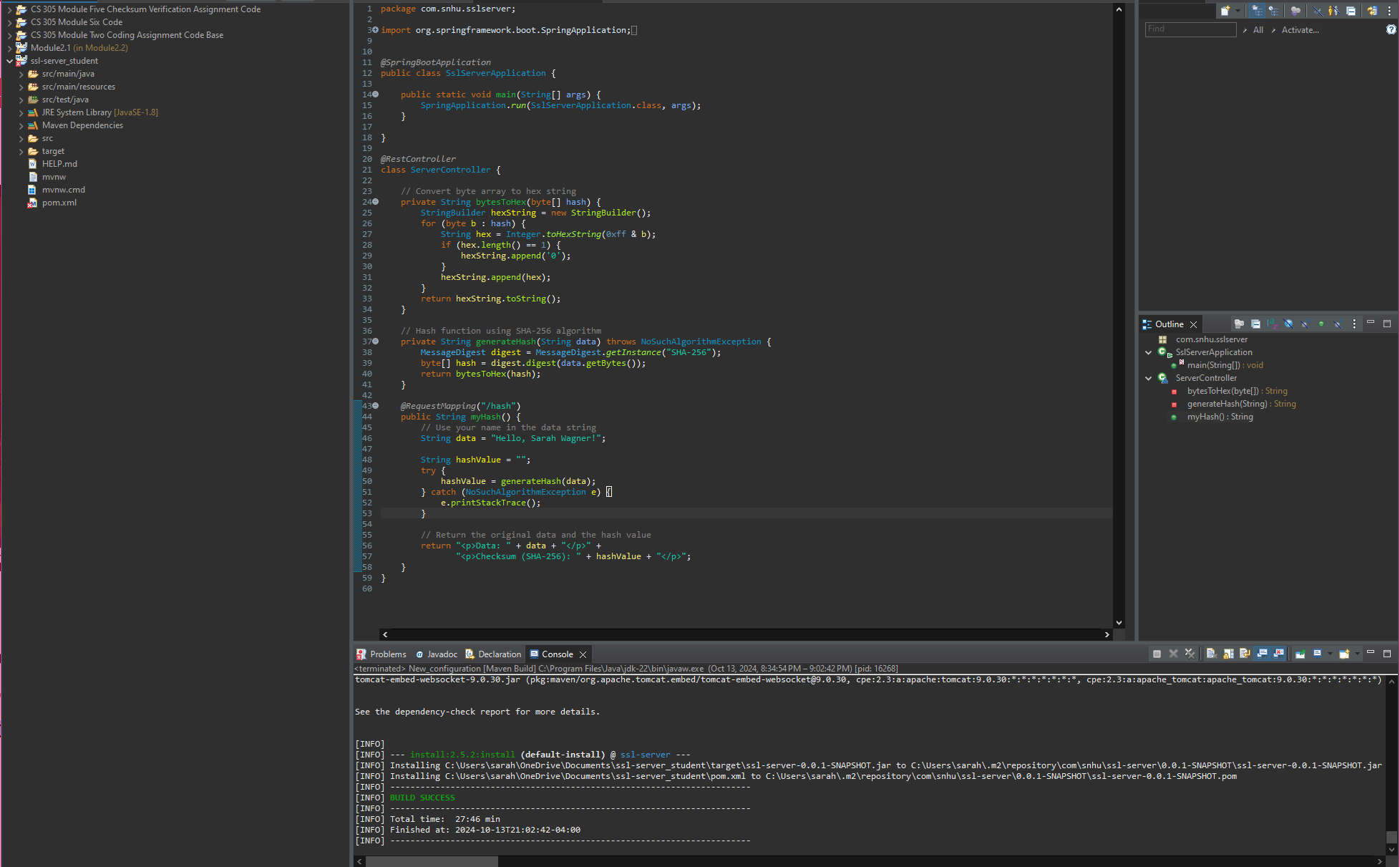
Insert screenshots below of the refactored code executed without errors and the dependency-check report.





## Functional Testing

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



## Summary

In this project, I refactored the code, so it complied with security testing protocols by following a structured vulnerability assessment process. Using the OWASP Dependency-Check tool, I analyzed the dependencies to identify any known vulnerabilities and prioritized them based on severity. During the refactoring, I focused on addressing potential weaknesses, such as upgrading vulnerable libraries flagged by the tool. After making necessary changes, I re-ran the Dependency-Check to make sure that no new vulnerabilities were introduced, iterating on this process until the application was free of high-priority security issues.

The areas of security I addressed during refactoring include dependency management, cryptographic hashing, input validation, and secure communication. Specifically, I updated or replaced insecure dependencies, implemented a strong cryptographic hash function (SHA-256) for checksum verification, and ensured that user inputs were validated to prevent injection attacks. I also configured the application to use SSL/TLS, guaranteeing secure communication over HTTPS.

Throughout the process, I applied a layered security approach by adding multiple layers of protection to the application. I integrated the OWASP Dependency-Check tool into the Maven build to automatically scan for vulnerabilities in dependencies, ensuring that no outdated or insecure libraries are used. Next, I implemented secure cryptographic practices by utilizing SHA-256 instead of weaker algorithms like MD5. Additionally, I enforced SSL/TLS to protect client-server communication and applied secure error-handling techniques to prevent the leakage of sensitive information.

## Industry Standard Best Practices

Adhering to industry standard best practices was important to mitigating known security vulnerabilities. I followed secure coding guidelines from OWASP and NIST, implementing input validation and secure cryptographic techniques. Furthermore, I ensured that static analysis was integrated into the build process to catch potential vulnerabilities early. This continuous verification process not only ensures security compliance but also helps maintain the application’s resilience against evolving security threats. By applying these best practices, I upheld the company’s existing security standards and ensured the software remained robust and secure.

The value of implementing industry standard best practices for secure coding goes beyond immediate security benefits. Preventing vulnerabilities early in the development process minimizes the risk of security breaches, protecting sensitive data and maintaining the company's reputation. Additionally, these practices reduce the long-term costs associated with patching vulnerabilities after deployment, resulting in more efficient and secure software. Following secure coding standards also builds customer trust and demonstrates the company’s commitment to security, while ensuring compliance with regulations like GDPR and HIPAA. Overall, applying these practices strengthens the software’s security and contributes to the company’s overall well-being by ensuring data protection, legal compliance, and maintaining a strong reputation in the market.