****

# 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** | **2/25/2024** | **Rachel Aldava** | **Initial Brief** |
| **1.0.1** | **2/25/2024** | **Rachel Aldava** | **New Vulnerability detected** |

## Client



## Developer

Rachel Aldava

## Algorithm Cipher

We have been asked to identify an appropriate encryption algorithm cipher for the requested program. Specifically, we have been asked to create a hash based off of a unique string of text. Creating such a hash would be considered a one-way operation; we are essentially converting the unique text through a mathematical process into another fixed-length text called a hash. Because the character length of the hash is fixed, whereas there can be any number of characters as the input, there will always be some chance that two potential inputs will have the same hash. When two inputs share the same hash, it is called a collision. The number of bits of a cipher correspond to how long the hash will be. Thus, more bits means that there is a lower chance that there will be a collision. A cipher with 1 bit would have a very high chance of collision at 50% (21 potential hashes) for any two given inputs. A 2 bit cipher would have an exponentially lower chance of collision at 25% (22 potential hashes) for any two given inputs. At 2128 potential hashes, SHA-128 would have a much lower chance of collision.

Some hashing algorithms have been so compromised that a threat actor can guess the inputs based off of the hash output, and they will be able to calculate which inputs can produce that hash. Whenever confidential information such as passwords are stored in the form of a hash, it is extremely important that the information remains a secret. Whenever a checksum is used as a verification of a file’s contents, this verification is less useful whenever a threat actor can spoof a hash. Because of this, there is an evolving need to select hashing algorithms which are secure.

In previous consultations regarding this project, we have spoken at length about the various algorithm ciphers available, which ones are viable, and what reasons we might have for choosing between several options (Aldava, 2024). To summarize, we discussed why SHA2 and SHA3 are the only families of ciphers which we should use. Other mainstream ciphers have been heavily compromised and less mainstream ciphers have not been as thoroughly verified as SHA2 and SHA3. We also discussed which industries used each cipher, speed versus security tradeoffs of the ciphers, and we decided to use SHA2-512 based off of the scenario we were given.

We are still recommending SHA-2 because it is still considered to be a general-purpose cipher family whereas SHA-3 was released as an improvement to SHA-2 in very specific use-cases. Our initial decision to recommend 512 bits was predicated on the assumption that communications would be very infrequent and thus there would be no significant expense in investing a few additional moments to ensure maximal security. However, as this project seems to be involving an internet facing application, there is an increased chance that communications will be more frequent than previously anticipated. Because SHA2-256 is still very secure, and because it has widespread use in secure banking websites in the financial sector, we recommend the improved speed of SHA2-256 at this time (Hash algorithm comparison, 2022).

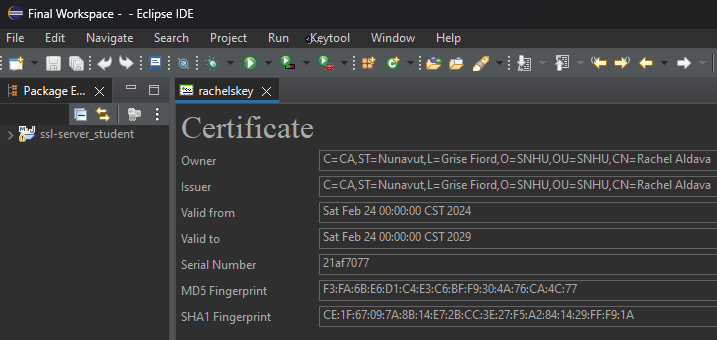
As with any algorithm, a threat actor may be able to infer the input data if the SHA2-256 hash is known; they could also guess at the data until they got a match. Consider the SHA2-256 hash value:

b94d27b9934d3e08a52e52d7da7dabfac484efe37a5380ee9088f7ace2efcde9

If one were to use a search engine on the above hash, one could easily discover that the input was “hello world”. Random numbers are one method to help ensure that the input is unique. For very technical reasons, computers are unable to create truly random numbers and there is ongoing research in improving the randomness of inputs. An alternative strategy against a threat actor guessing one’s input is to supply an extremely lengthy input which is less random. There was no guidance about which unique text we should use, so in our scenario we added one sentence to the full text of William Shakespeare’s *King Lear*.

Finally, we were to discuss symmetric and non-symmetric keys. We went into some detail in previous communications, but to reiterate, this topic is slightly different from the one-way operations we have been discussing. Instead of creating a hash which obfuscates the input, a two-way operation explicitly allows for an encrypted message to be decrypted. These types of algorithms use “keys” in their mathematical operations. Keys are bits of data which the algorithm uses to encode and decode a message. A symmetrical key is one which can be used to encrypt and decrypt the same message. An asymmetrical key-pair is one which can decrypt a message which the other key encrypted, but the key cannot decrypt its own message. Symmetrical keys can be useful in certain limited instances, but asymmetrical keys allow for a server and client to communicate without a third party being able to intercept the communications. These client:server communications could be explained over several pages, but we fear that further detail on the subject may exceed the scope of this brief; the focus of the this section seems to be intended to be on one-way hashing operations. We encourage the reader to reference previous communications or to reach out to use directly if they wish for more clarifications on the subject.

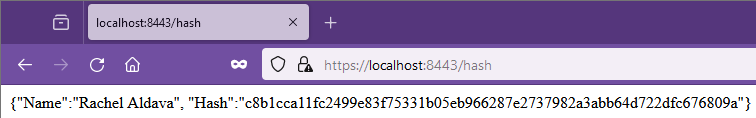
## Certificate Generation

Last time, we discussed generating a self-signed certificate via the command line using java’s built-in keytool. The eclipse IDE, with the appropriate addon, is also able to generate certificates as shown below:

## Deploy Cipher

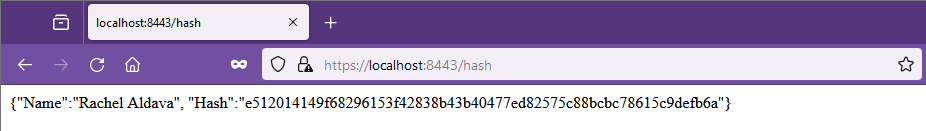
We used standard java functions to read bytes from an arbitrary file. We then ran those bytes through the Message Digest class using SHA2-256, then output the resultant hash along with the name of our lead developer. For testing purposes, we used the script of *King Lear* by William Shakespeare (Shakespeare, n.d.).

Output using kingLear.txt (Original version):



Next, we added a line to the text file to demonstrate that the hash will change whenever the file is changed:

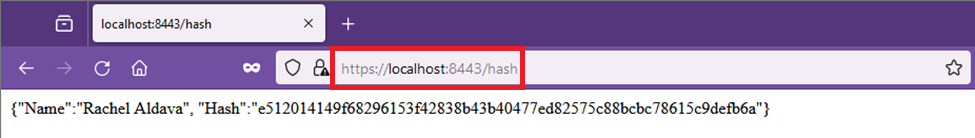
Output using kingLear.txt (edited version)



The reader is encouraged to modify the /target/classes/kingLear.txt file to see this functionality in action.

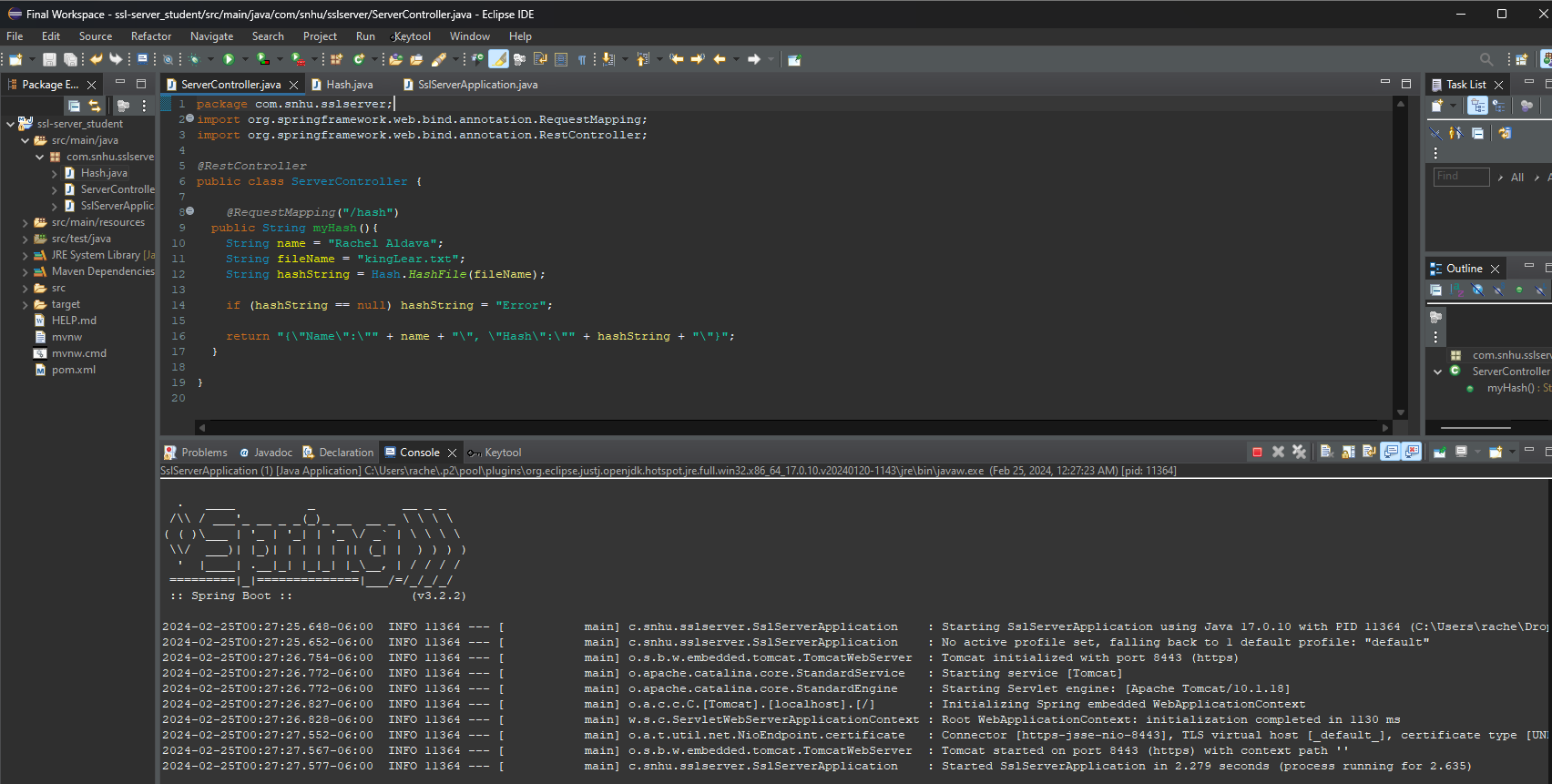
## Secure Communications

The program has been configured to use HTTPS protocols and port 8443.

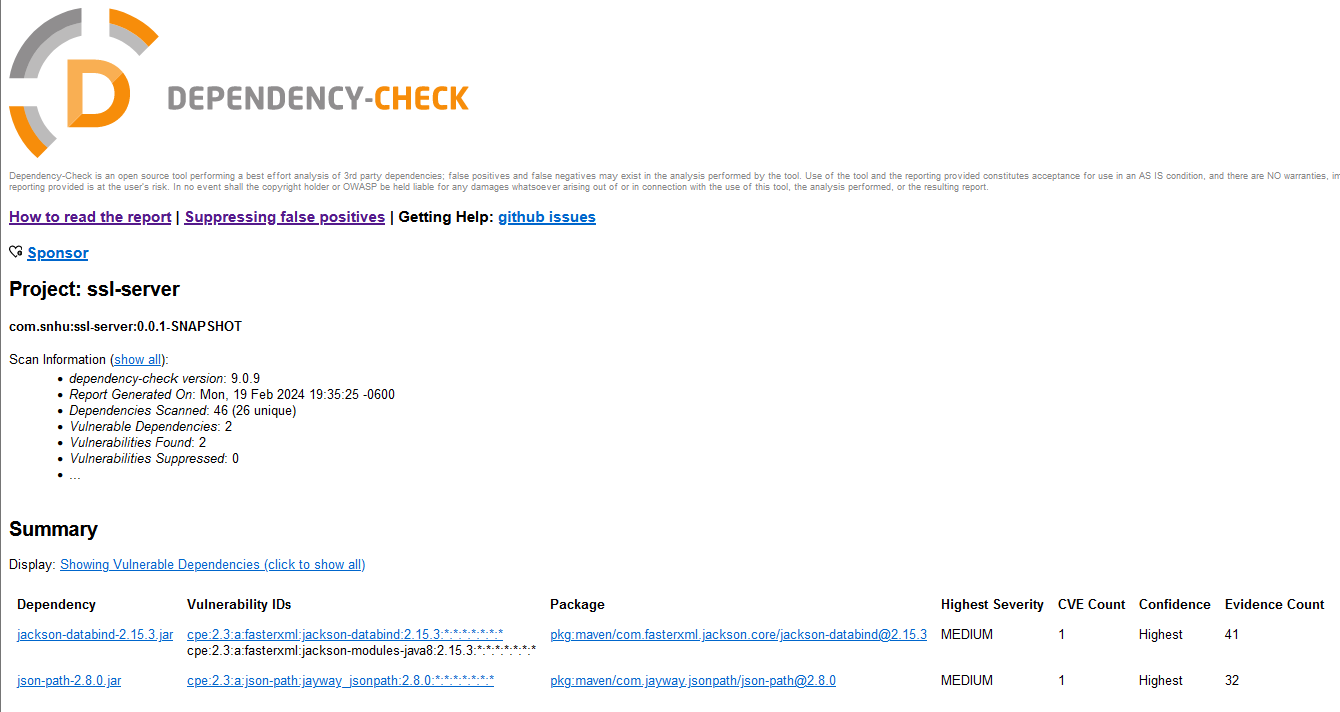


## Secondary Testing

At this point, we can confidently declare that it works on our machine.



And when we run a vulnerability scan, we see there are two detected vulnerabilities:



**Vulnerability: CVE-2023-35116:**

This issue exists with Jackson-databind-2.15.3.jar and occurs whenever the class ObjectMapper is used with a recursive datatype. While this issue may occur with one of our dependency files, our code does not use the ObjectMapper class nor does it use any recursive data structures. It seems to be the opinion of several github users that this issue does not constitute a security vulnerability (Github issue #3972, n.d.). For these two reasons, we consider this to be a false-positive.

**Vulnerability: CVE-2023-51074:**

This issue exists with json-path-2.8.0.jar and occurs whenever the Criteria class is used to parse a maliciously crafted input(Github issue #973, n.d.). We do not use this class in our code nor does this code allow for users to input data. As such, this vulnerability does not seem to be an issue at this time.

**<BRIEF 1.0.1 ADDENDUM>**

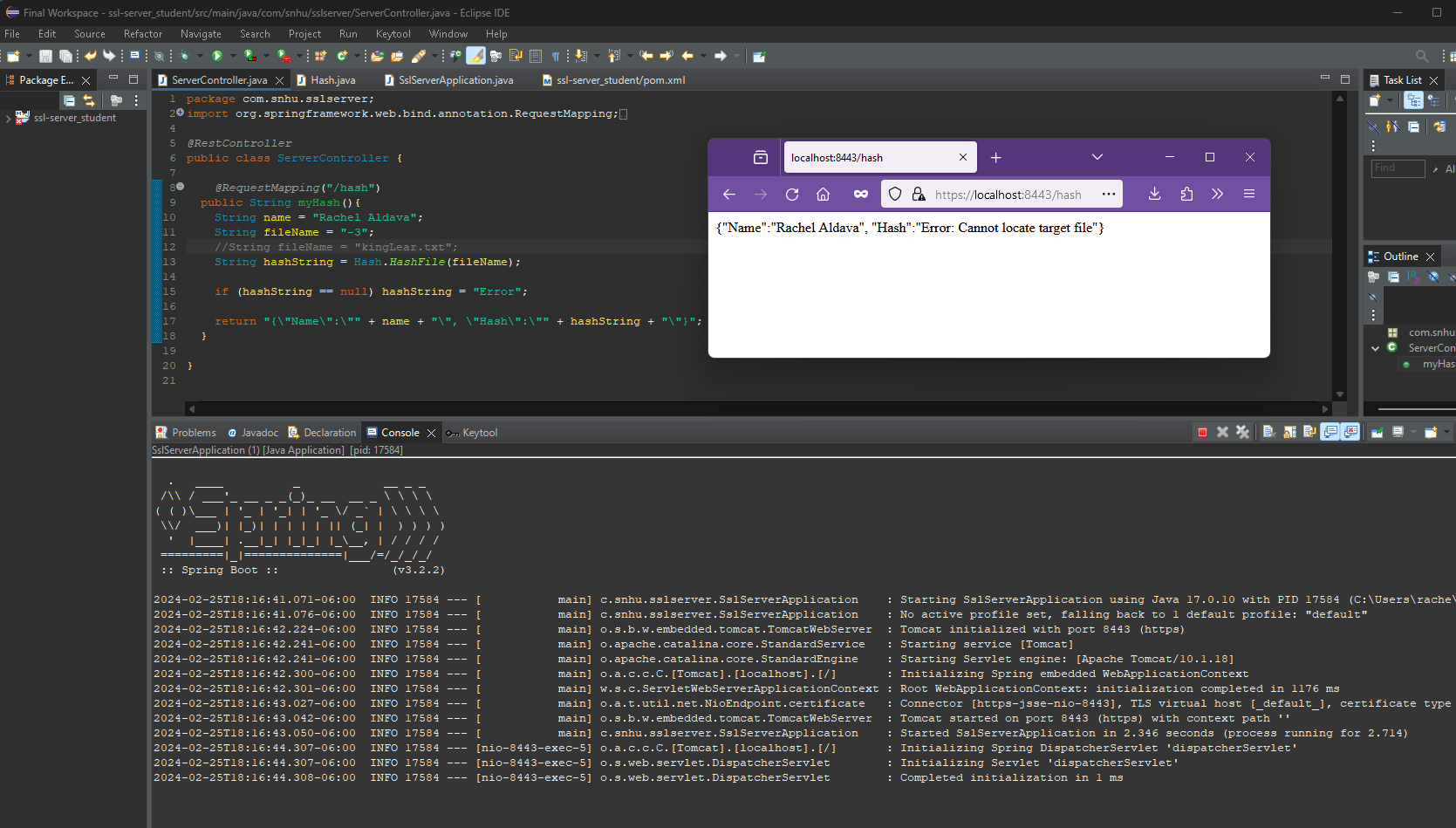
In a neurotic compulsion to check everything before submitting this document, we ran the vulnerability check again and discovered a third vulnerability: **CVE-2024-22243** is a vulnerability with the class UriComponentsBuilder whenever an external URL is supplied and parsed. In our code we call

Paths.*get*(ClassLoader.*getSystemResource*(path).toURI());

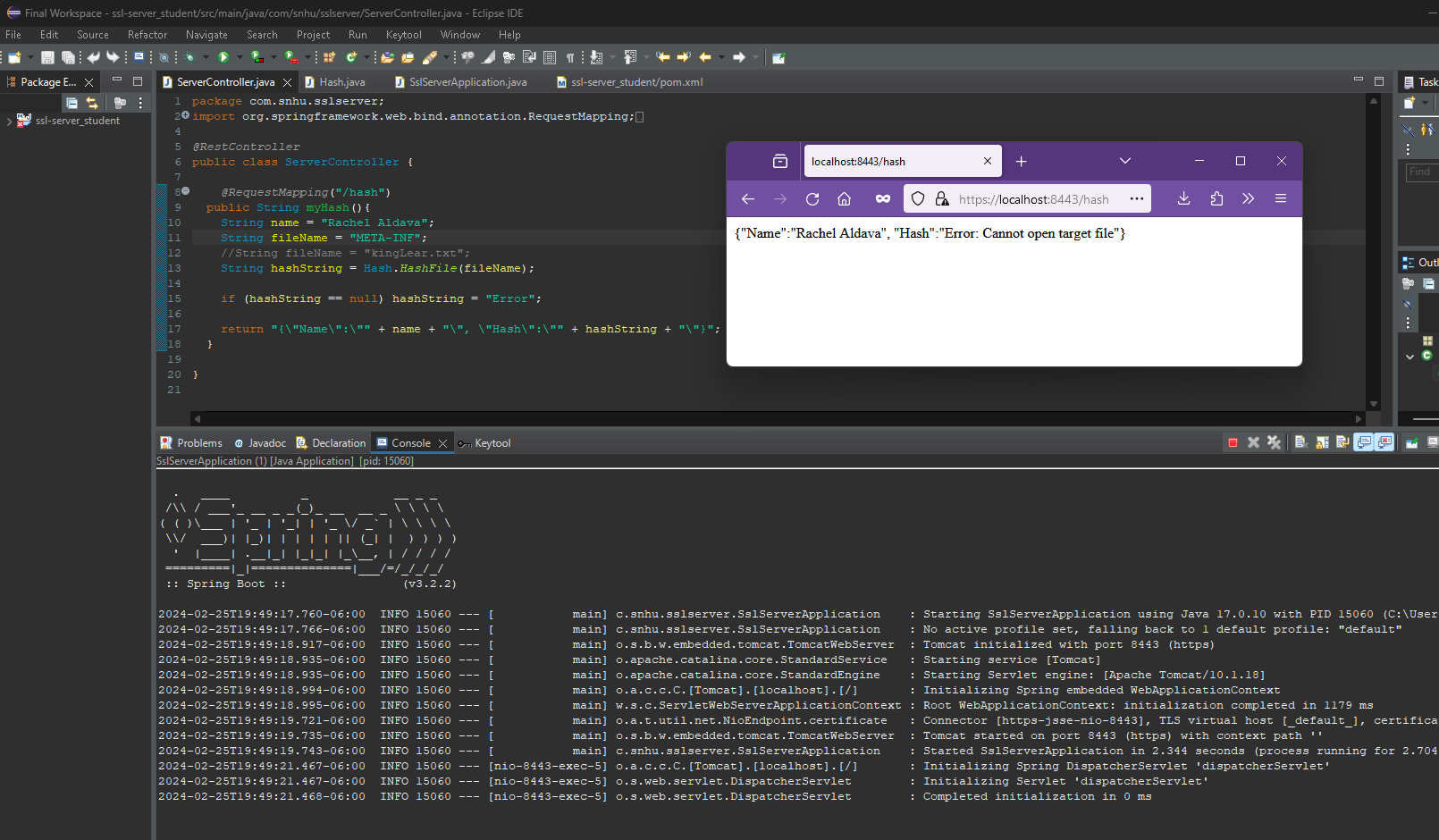
Where path is a string which is hard-coded in ServerController.java. It seems that the .toURI() method is not part of SpringFramwork’s UriComponentsBuilder (which uses toURIString()), but instead is part of java.net.URI. Thus, we are not affected by this issue (docs.spring.io, n.d.). **</BRIEF 1.0.1 ADDENDUM>**

## Functional Testing

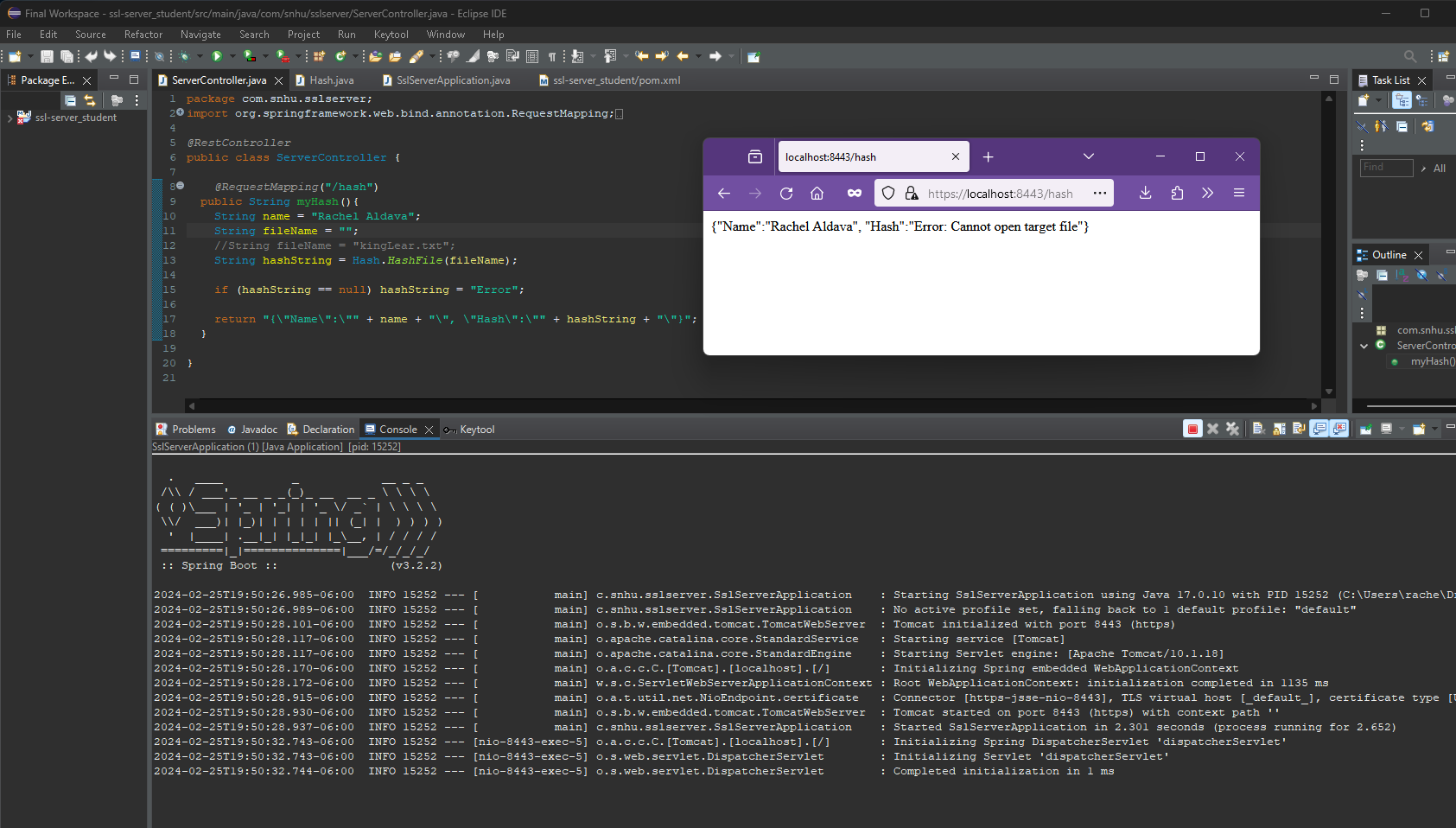
Initial functional testing revealed one error involving an uncaught exception whenever an invalid filepath was supplied, which was easily corrected. There appears to be no obvious errors in logic or syntax. There are very few “moving parts” in this code, so provided that the dependencies do not change the only logical errors that could be conceivably produced involve inputs. In this code, the entire file of kingLear.txt could be considered an input, but since the program only reads the bytes of the file it seems that any file in the directory could be read. If, instead, we were to consider the “fileName” string in ServerController.java as an input (even if it is currently hard-coded) then we could run some tests with different values:

First, we test basic sanity: testing “-3” correctly produces an error because there is no -3 file in the class directory:

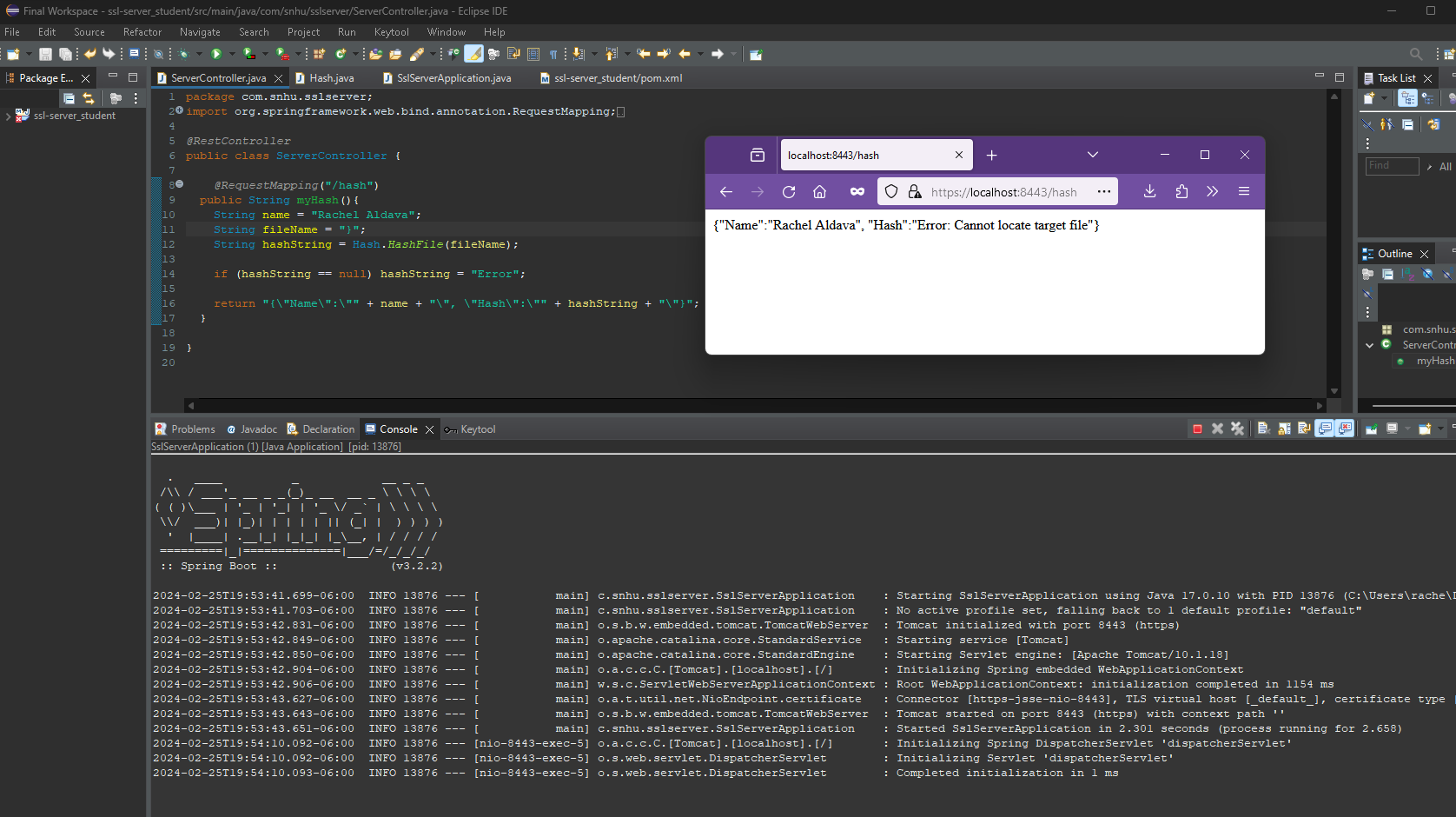
Next, we test for unexpected structures: Testing “META-INF” correctly produces an error because META-INF is a folder in the directory:



Next, we test a blank string as an edge case: Testing “” correctly produces an error because there is no “” file in the class directory:

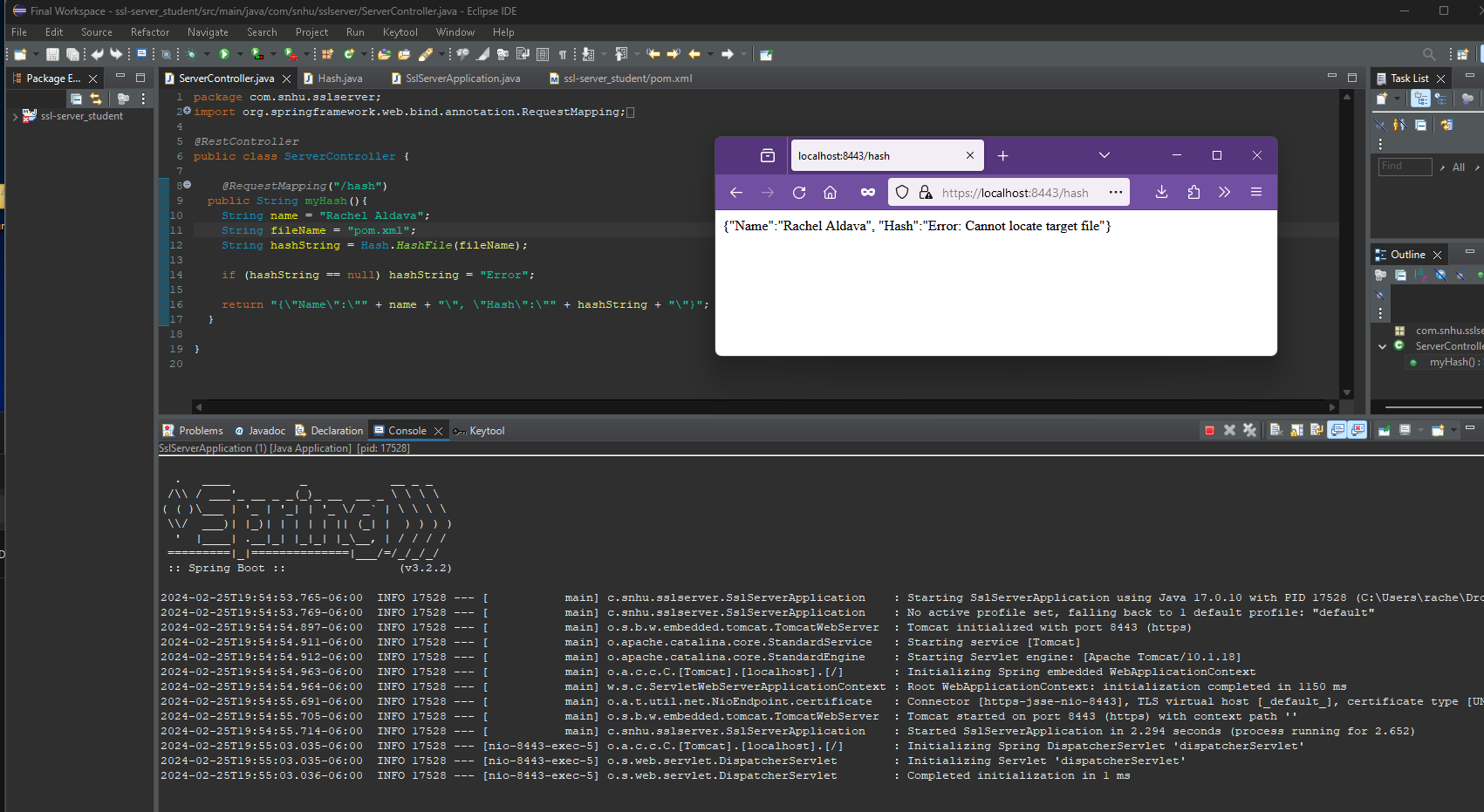


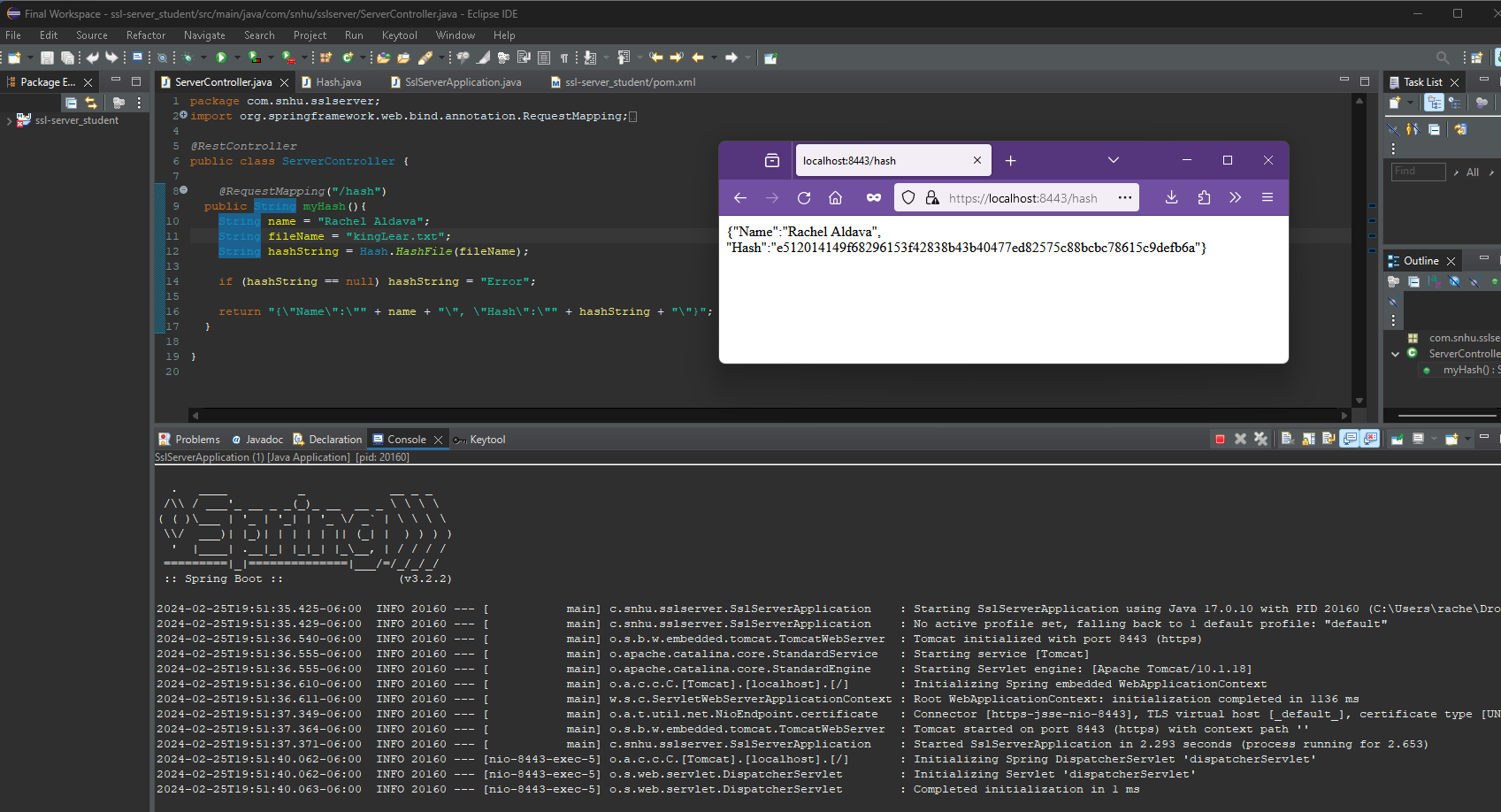
Next we make sure there isn’t some strange parsing: Testing “}” correctly produces an error because there is no } file in the class directory:



Note: This is not a substitute for proper QA. More testing should occur.

Next, we test if existing files outside of the class directory can be accessed. Testing “pom.xml” correctly produces an error because, even though it exists in the project there is no - pom.xml file in the class directory:



Finally, we demonstrate that the normal inputs still produces the expected output:

## Summary

Much of the refactoring of the original codebase involved properly configuring the config files. While we did not mention this above, we started by running an initial vulnerability check which revealed dozens of vulnerabilities. We updated the dependencies to the highest level which still maintained compatibility and we ran a second vulnerability check, which showed the two vulnerabilities we mentioned above. We then created a certificate and configured the server to run on the specified port using HTTPS protocols. Once the server was operational, we added the mapping and implemented the hashing logic. Once done, we ran the vulnerability test again and found no new vulnerabilities. We then researched the vulnerabilities to verify that our program was not affected. We ran QA tests, found and fixed one uncaught exception, and quadruple checked the vulnerability report.

Regarding the Vulnerability Assessment Flow diagram, we spent the most time reviewing input validation because the rest of the program is extremely simple. We did review that API interactions, the encryption use (and we ensured that we do not store the input bits in the program longer than needed), error handling (as noted), code quality (to the exacting standards of sub-junior level), and attempted to implement Encapsulation as much as possible.

## Industry Standard Best Practices

The primary step we took to mitigate every detected vulnerability was upgrading the java version and dependencies. We ensured that there was appropriate levels of encapsulation, we tested various inputs, and ensured that exceptions were appropriately caught. We decided not to print stack traces and implement our own error messages because some organizations believe that showing the output of printstacktrace to the enduser reveals too much information about the internal workings of the system; This program is currently very simple, so finding where the error occurred will not be very difficult.

Applying industry standard best practices to a project is valuable in several ways. Every organization ought to be motivated to deliver a timely and reasonably high quality product to its customers and these best practices help ensure a timely, high quality product. If a product is built around standards, it will be easier for other people to work with the product, whereas an idiosyncratic product may require additional time or skill. Another way the best practices are valuable is because they have been formulated over many years by a plurality of experts, all of whom are more knowledgably and skilled than the author of this brief; It is a font of knowledge and going against expert guidance seems like a bad idea. Another reason these practices are valuable is that it can streamline the design and implementation process because there is a reduced planning and implementing overhead whenever one implements something which one has already implemented many times before.

**Works Cited**

Aldava, R. (2024). CS 305 Module Five Coding Assignment Checksum Verification [Paper]. SNHU.

Hash algorithm comparison: MD5, SHA-1, SHA-2 & sha-3. Code Signing Store. (2022, February 15). https://codesigningstore.com/hash-algorithm-comparison

Shakespeare, W. (n.d.). King Lear. King lear: Entire play. http://shakespeare.mit.edu/lear/full.html

Stack overflow error caused by serialization of `Map` with cyclic dependency -- NOT CVE · Issue #3972 · FasterXML/jackson-databind. (n.d.). GitHub. https://github.com/FasterXML/jackson-databind/issues/3972

Stack Overflow (Criteria.parse) · Issue #973 · json-path/JsonPath. (n.d.). GitHub. https://github.com/json-path/JsonPath/issues/973

UriComponentsBuilder (Spring Framework 6.1.4 API). (n.d.). https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/web/util/UriComponentsBuilder.html