****

# CS 305 Project One

**Artemis Financial Vulnerability Assessment Report**

Table of Contents

[Document Revision History 3](#_Toc32574607)

[Client 3](#_Toc32574608)

[Instructions 3](#_Toc32574609)

[Developer 4](#_Toc32574610)

[1. Interpreting Client Needs 4](#_Toc32574611)

[2. Areas of Security 4](#_Toc32574612)

[3. Manual Review 4](#_Toc32574613)

[4. Static Testing 4](#_Toc32574614)

[5. Mitigation Plan 4](#_Toc32574615)

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **3/18/21** | **Tom Czubat** | **Initial Vulnerability Assessment was completed.** |

## Client



## Instructions

Deliver this completed vulnerability assessment report, identifying your findings of security vulnerabilities and articulating recommendations for next steps to remedy the issues you have found.

Respond to the five steps outlined below and include your findings. Replace the bracketed text on all pages with your own words. If you choose to include images or supporting materials, be sure to insert them throughout.

## Developer

Tom Czubat

## 1. Interpreting Client Needs

Determine your client’s needs and potential threats and attacks associated with their application and software security requirements. Consider the following regarding how companies protect against external threats based on the scenario information

* What is the value of secure communications to the company?
* Are there any international transactions that the company produces?
* Are there governmental restrictions about secure communications to consider?
* What external threats might be present now and in the immediate future?
* What are the “modernization” requirements that must be considered, such as the role of open source libraries and evolving web application technologies?

The value of secure communications for Artemis Financial is vital. All of its business communications must be protected to ensure their information does not fall into the wrong hands. This application provides the functionality of executing financial transactions and exchanges.This information is sensitive because it could include client names and payment information. A breach of this application could cause Artemis Financial a great deal of money and clients.

Artemis Financial is involved in multinational transactions. These transfers occur over the internet from across the world. This allows the company’s customers to be able to use their application from all around the world.

There are several government restrictions to consider for this application. For example, the General Data Protection Regulation (GDPR) needs to be kept in mind. This proposal will mandate that data handlers and service providers ensure that their client's information is secure from attacks. Data encryption and secure storage will be an important part of the security protocol.

There are several possible external threats that need to be accounted for. First, we need to consider a Man in the Middle attack, which is when a hacker intercepts communication between a host and client and then manipulates data by listening in on the packets of flowing information. Another external attack could be a Distributed Denial of Service (DDOS) attack, which an attack that over-exercises specific functions or features of a website with the intention to disable those functions or features.

The modernization requirements that must be considered are sandboxing and virtualizing everything. This ensures hackers have a much harder time of accessing the main portion of the system and even if they get into a VM, they can’t really do anything. Using VPN’s is another possible way of modernizing this company’s approach.

## 2. Areas of Security

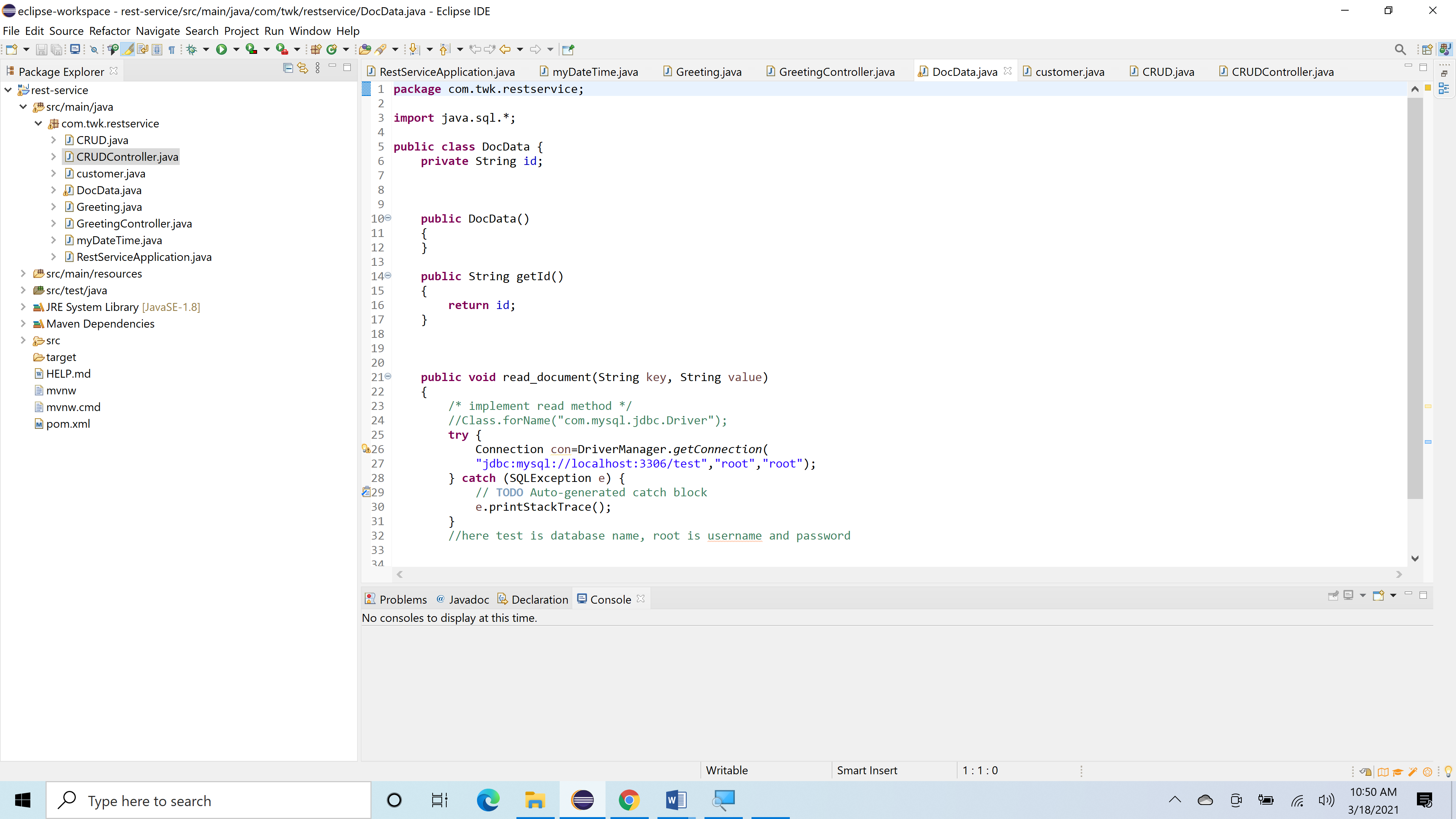
Referring to the Vulnerability Assessment Process Flow Diagram, identify which areas of security are applicable to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

When looking at the Vulnerability Assessment Process Flow Diagram, I can identify several areas of security that are relevant to this application for Artemis Financial. The sections of the vulnerability diagram that I will discuss will be API, Cryptography, Client/Server, and Secure Coding. First we have the API section and this area of security applies to this application because the web application uses APIs calls. It is vital that there is a secured configuration for the APIs in order to ensure that there is no vulnerabilities between the system and the API connection. Next we have cryptography and this security area applies because there is the access and transmission of sensitive private information over the internet. Without proper encryption in the data transfer can result to a compromise of the integrity and confidentiality of the information when it is transmitted over the internet by the consumers of the Artemis Financials application. If there is no encryption, this vulnerability could allow hackers to see the flow of data coming in and out of your network with a Man in the Middle Attack. Another factor to consider to make sure that there is proper certificate validation being performed. Next we have to consider the Client/Server section and this area of security applies because the web application involves the communication between the customer, which is the client, and the backend of the web application, which is the server, to retrieve and send financial transactions. The security of this connection is a very important and consideration for the Artemis Financial’s application. Last but not least, secure coding will be an important area of security and applies to the Artemis Financial’s application because there are several instances that would require secure code updates to maintain a consistency in the business logic of the application. One example of this could be the lack of exception handling and error checks in the source code of this application.

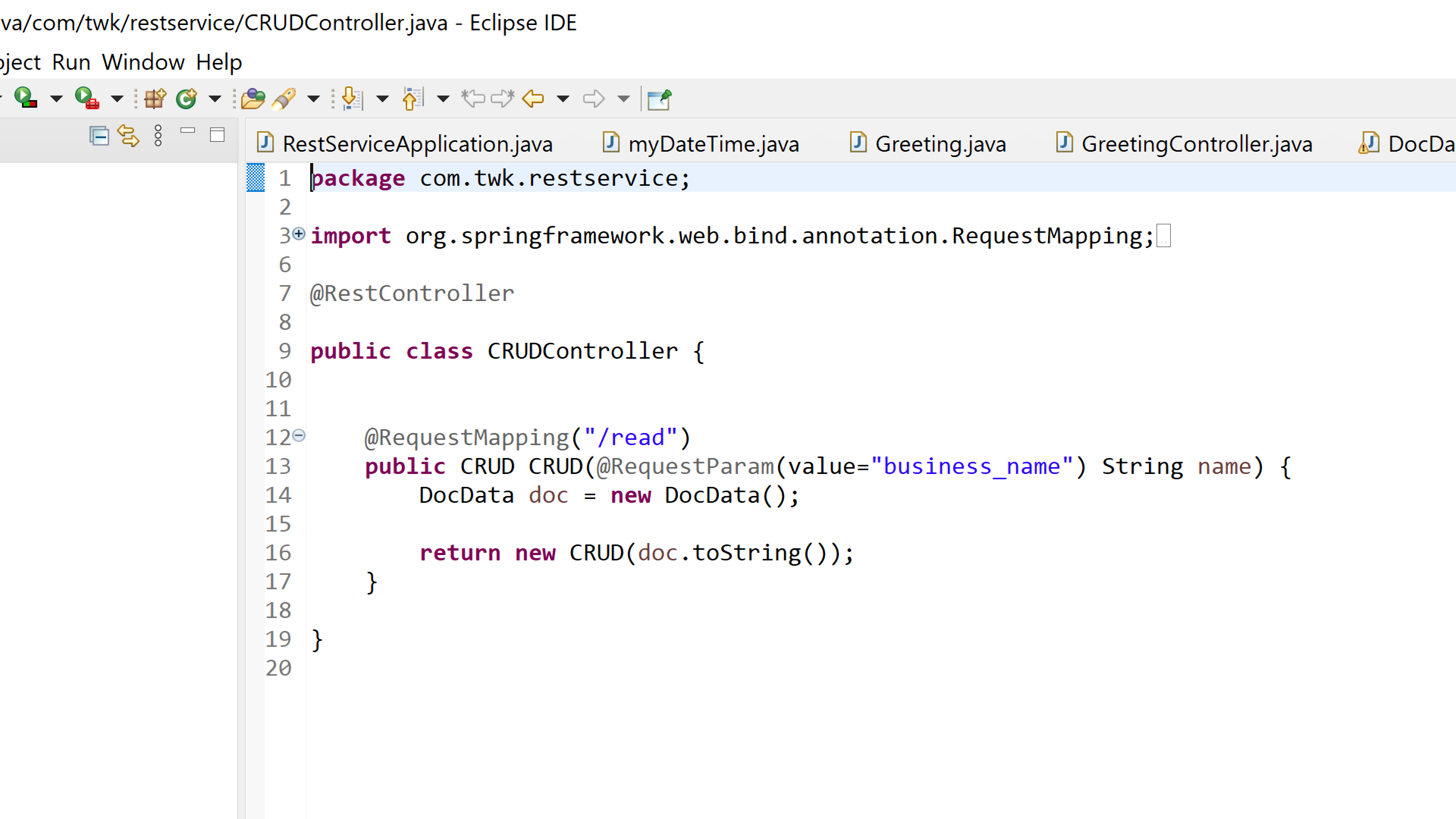
## 3. Manual Review

Continue working through the Vulnerability Assessment Process Flow Diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

The first vulnerability has to do with data access. For example, in the DocData.java, the data access method that involves the definition of the location of the database, the username and password has some vulnerabilities. There is the root users and the password which is not recommendable to use the root user. The root password can be easily guessed and compromised. The root is used as the username and also as the password. This makes it easy to crack and unauthorized users can access the system by brute force attacks. The member methods in the DocData.java class should be set to private so that they can only be accessed by an instance of a DocData object.



The next possible vulnerability I found was with direct object reference. There is a vulnerability in the CRUDController.java class where there is a chance of the application exposing the internal objects. They can be passed and accessed through injections of code. The value=”business\_name” is passed in the CRUD method which may expose the DocData object database access vulnerability. This is shown in the figure below.



The next vulnerability I have found has to do with using proper coding conventions and setting the proper access modifiers on class member variables. First, when it comes to the customer.java class, proper coding practices state that the first letter in the class name must be capitalized. All of the other class in this project start with a capitol letter so the first letter of the customer.java class should be capitalized to maintain consistency with naming conventions. In the myDataTime.java class, the member variables and functions should be set to private to provide more security.

## 4. Static Testing

Run a dependency check on Artemis Financial’s software application to identify all security vulnerabilities in the code. Record the output from dependency check report. Include the following:

1. The names or vulnerability codes of the known vulnerabilities
2. A brief description and recommended solutions provided by the dependency check report
3. Attribution (if any) that documents how this vulnerability has been identified or documented previously

After running a dependency check on the application, there were 8 vulnerabilities that were discovered.

The first vulnerability was **bcprov-jdk15on-1.46.jar** and this has to do with the Bouncy Castle Crypto package is a Java implementation of cryptographic algorithms. This jar contains JCE provider and lightweight API for the Bouncy Castle Cryptography APIs for JDK 1.5 to JDK 1.7. This is vulnerable because The TLS implementation in the Bouncy Castle Java library before 1.48 and C# library before 1.8 does not properly consider timing side-channel attacks on a noncompliant MAC check operation during the processing of malformed CBC padding, which allows remote attackers to conduct distinguishing attacks and plaintext-recovery attacks via statistical analysis of timing data for crafted packets, a related issue to CVE-2013-0169. The vulnerability code is [**CVE-2013-1624**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2013-1624)**.** To fix this issue, you need to make sure to have the latest version of the Bouncy Castle Java Library.

The second vulnerability is **hibernate-validator-6.0.18.Final.jar** and this has to do with Hibernate's Bean Validation (JSR-380) reference implementation. This in includes the [**CVE-2020-10693**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-10693)  vulnerability which is a flaw that was found in Hibernate Validator version 6.1.2.Final. A bug in the message interpolation processor enables invalid EL expressions to be evaluated as if they were valid. This flaw allows attackers to bypass input sanitation (escaping, stripping) controls that developers may have put in place when handling user-controlled data in error messages. This can be addressed by improving input validation.

The third vulnerability **is** **jackson-databind-2.10.2.jar** and this has to do with general data-binding functionality for Jackson: works on core streaming API. The vulnerability code is [**CVE-2020-25649**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-25649)and its description is that a flaw was found in FasterXML Jackson Databind, where it did not have entity expansion secured properly. This flaw allows vulnerability to XML external entity (XXE) attacks. The highest threat from this vulnerability is data integrity. To fix this, the Restriction of XML External Entity Reference must be addressed.

The fourth vulnerability is in **log4j-api-2.12.1.jar** and this has to do with the Apache Log4j API. The vulnerability code is [**CVE-2020-9488**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9488)and this entails Improper validation of certificate with host mismatch in Apache Log4j SMTP appender. This could allow an SMTPS connection to be intercepted by a man-in-the-middle attack which could leak any log messages sent through that appender. This can be fixed with proper certificate validation.

The fifth vulnerability is **snakeyaml-1.25.jar** and this is the YAML 1.1 parser and emitter for Java. The vulnerability code is [**CVE-2017-18640**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2017-18640)and it entails The Alias feature in SnakeYAML 1.18 allows entity expansion during a load operation, a related issue to CVE-2003-1564. This can be fixed with proper restriction of recursive entity references in DTDs.

The sixth vulnerability is **spring-core-5.2.3.RELEASE.jar** which has to do the Spring Core. The vulnerability code is [**CVE-2020-5421**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-5421)and this has to do with the Spring Framework versions 5.2.0 - 5.2.8, 5.1.0 - 5.1.17, 5.0.0 - 5.0.18, 4.3.0 - 4.3.28, and older unsupported versions, the protections against RFD attacks from CVE-2015-5211 may be bypassed depending on the browser used through the use of a jsessionid path parameter. I think this can be addressed by updated to the latest supported Spring Framework.

The seventh vulnerability is **tomcat-embed-core-9.0.30.jar** which has to do with the Core Tomcat implementation. The vulnerability code is [**CVE-2019-17569**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2019-17569)and this has to do with the refactoring present in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50 and 7.0.98 to 7.0.99 introduced a regression. The result of the regression was that invalid Transfer-Encoding headers were incorrectly processed leading to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner. Such a reverse proxy is considered unlikely. This can be fixed by addressing the Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling').

The eight vulnerability is **tomcat-embed-websocket-9.0.30.jar** which has to do with Core Tomcat implementation. The vulnerability code is [**CVE-2019-17569**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2019-17569)which has to do with the refactoring present in Apache Tomcat 9.0.28 to 9.0.30, 8.5.48 to 8.5.50 and 7.0.98 to 7.0.99 introduced a regression. The result of the regression was that invalid Transfer-Encoding headers were incorrectly processed leading to a possibility of HTTP Request Smuggling if Tomcat was located behind a reverse proxy that incorrectly handled the invalid Transfer-Encoding header in a particular manner. This vulnerability can fixed by addressing the Inconsistent Interpretation of HTTP Requests ('HTTP Request Smuggling').

## 5. Mitigation Plan

After interpreting your results from the manual review and static testing, identify the steps to remedy the identified security vulnerabilities for Artemis Financial’s software application.

After interpreting my results from the manual review and static testing, I have identified a mitigation plan to protect Artemis Financial’s software application. To address the data access of the username and password, the database access of the username and password need to be securely implemented. Another recommendation is for a mandated strong password that should contain a combination of letters, numbers, and symbols. When this information is stored in the database, strong encryption must be used for maximum security.

Next, there are recommendations based on the manual code review that was conducted. Secure code practices must be implemented to create a secure and consistent software product. Implementing strong user authentication will strongly recommended to protect user’s information. Creating consistency across code in all class of the application is vital. An example of this is making sure that all classes are defined where the first letter is capitalized.

Based on the vulnerability scan that was performed, it is imperative that the latest version of all technology, including frameworks and libraries, to mitigate and vulnerabilities from unsupported versions of frameworks. This includes the Apache Server, the Bouncy Castle Java Library, and the Spring Core.

Certificate Validation is another area of concern that was found when the vulnerability scan was being performed. The various cryptographic certificates should be well validated. The codes should be sanitized to allow proper validation and verification of the digital certificates in the application and in the web server. This will help address the vulnerability found in the [**CVE-2020-9488**](http://web.nvd.nist.gov/view/vuln/detail?vulnId=CVE-2020-9488)vulnerability.