# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| **1.0** | **[Date]** | **[Your name]** |  |

## Client



## Instructions

Submit this completed vulnerability assessment report. Replace the bracketed text with the relevant information. In this report, identify your security vulnerability findings and recommend the next steps to remedy the issues you have found.

* Respond to the five steps outlined below and include your findings.
* Respond using your own words. You may also include images or supporting materials. If you include them, make certain to insert them in the relevant locations in the document.
* Refer to the Project One Guidelines and Rubric for more detailed instructions about each section of the template.

## Developer

Adryan Knight

**1. Interpreting Client Needs**

Determine your client’s needs and potential threats and attacks associated with the company’s application and software security requirements. Consider the following questions 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 on secure communications to consider?
* What external threats might be present now and in the immediate future?
* What modernization requirements must be considered, such as the role of open-source libraries and evolving web application technologies?

• Value of secure communications to the company: Prevents disclosure or tampering of sensitive financial data; preserves client trust and regulatory compliance.

• International transactions: Potential cross‑border data transfers require attention to differing privacy laws (e.g., GDPR) and financial standards.

• Governmental restrictions: Use approved cryptographic libraries/algorithms (FIPS‑compliant where required) and follow applicable regulations for financial data retention and protection.

• External threats (now and near future): Injection, broken access control, sensitive data exposure, security misconfiguration, vulnerable/outdated components, and insufficient logging/monitoring.

• Modernization requirements to consider:

– Manage open‑source components to avoid known CVEs (software composition analysis).

– Enforce HTTPS/TLS everywhere with strong ciphers and HSTS.

– Centralize secret management (no credentials in source code).

– Add authentication/authorization on all endpoints and validate/sanitize inputs.

**2. Areas of Security**

Refer to the vulnerability assessment process flow diagram. Identify which areas of security apply to Artemis Financial’s software application. Justify your reasoning for why each area is relevant to the software application.

• Input Validation – Controllers accept request parameters without format/length validation.

• APIs – Public endpoints lack authentication/authorization and rate limiting.

• Cryptography – No explicit TLS enforcement settings; credentials appear in code; DB connection is not configured with TLS.

• Client/Server – App listens on port 8081 with no HTTPS termination or security headers configured.

• Code Error – Limited error handling; potential to leak stack traces in default configuration.

• Code Quality – Incomplete classes and weak encapsulation reduce maintainability and may hide defects.

• Encapsulation – Package‑private fields and missing validation expose internal state.

**3. Manual Review**

Continue working through the vulnerability assessment process flow diagram. Identify all vulnerabilities in the code base by manually inspecting the code.

1. Outdated platform: pom.xml uses spring‑boot‑starter‑parent 2.2.4.RELEASE, which is end‑of‑life and likely to inherit multiple CVEs through transitive dependencies.

2. Hard‑coded database credentials in DocData.java (username 'root' / password 'root') and a local MySQL JDBC URL without TLS or certificate verification.

3. Unencrypted DB connection string (jdbc:mysql://...), no use of TLS parameters (e.g., useSSL=true; verifyServerCertificate=true) and no least‑privilege DB account.

4. Controllers accept unvalidated inputs (e.g., /greeting and /read). No length, pattern, or type checks; susceptible to abuse and injection risks.

5. File uploads enabled with large limits (200MB) and relative upload path './uploads' in application.properties; no file type allow‑list or scanning and risk of path traversal.

6. No authentication/authorization (no Spring Security) – all endpoints are public (OWASP A01:2021 Broken Access Control).

7. Weak encapsulation: class 'customer' exposes package‑private fields and no validation; business logic lacks guardrails.

8. Incomplete placeholder code in myDateTime.java (unused/uninitialized fields and methods), which can lead to logic errors when referenced.

9. Potential information leakage: default error pages and stack traces may be exposed in non‑production profiles.

10. Missing standard security headers by default (HSTS, X‑Content‑Type‑Options, X‑Frame‑Options, Content‑Security‑Policy).

**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 the dependency-check report. Include the following items:

* The names or vulnerability codes of the known vulnerabilities
* A brief description and recommended solutions provided by the dependency-check report
* Any attribution that documents how this vulnerability has been identified or documented previously

I could not execute the OWASP Dependency‑Check plug‑in in my environment due to network restrictions. However, by reviewing the project's pom.xml and Spring Boot baseline (2.2.4.RELEASE), I identified components that are widely known to have historical CVEs. The items below are provided as \*\*representative findings\*\* one would expect Dependency‑Check to report for this code base. They are clearly marked as inferences and should be verified by running the tool on a machine with internet access.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component | Version (from baseline) | Representative CVE(s) | Risk Summary | Recommended Fix |
| Spring Framework (via Spring Boot 2.2.4) | 5.2.x | CVE‑2022‑22965 ("Spring4Shell") | RCE risk under specific conditions for data binding on JDK 9+ with certain app configurations. | Upgrade platform to Spring Boot 3.x (Spring 6.x) or, if staying on 2.x, at least Boot 2.5.12+/2.6.6+ where patched; retest. |
| Jackson‑databind | 2.10.x (transitive) | CVE‑2020‑36518; multiple gadget‑based deserialization CVEs | Potential DoS and deserialization issues when polymorphic typing is misused. | Upgrade with Spring Boot 3.x BOM (Jackson 2.15+); avoid enabling default typing; audit input. |
| Logback‑classic/logback‑core | 1.2.x (transitive) | CVE‑2021‑42550 | JNDI‑related vulnerability enabling config manipulation in some scenarios. | Use logback ≥ 1.2.9 or versions aligned with current Spring Boot 3.x; restrict writable config paths. |
| Apache Tomcat (embedded) | 9.0.30± (transitive) | CVE‑2020‑1938 (AJP "Ghostcat") and others in older 9.0.x | Information disclosure / file read via AJP if enabled and exposed. | Upgrade via newer Spring Boot baseline (Tomcat ≥ 9.0.56+); disable AJP unless required; bind interfaces safely. |
| OWASP Dependency‑Check Maven Plug‑in | 5.3.0 (configured) | Outdated scanning engine (no CVE—tooling gap) | Old engine may miss recent CVEs or fail updates. | Use latest 8.x CLI or plugin when network access is available to produce an authoritative report. |

\*\*Attribution/References to include when verified:\*\* NVD entries for the listed CVEs, Spring advisories, Jackson and Logback release notes, and Tomcat security pages. Once Dependency‑Check is run successfully, paste the actual CVE list and versions resolved from your build into this section to replace or confirm these entries.

### Expanded CVE References (Representative)

CVE-2022-22965 (Spring4Shell)  
 NVD: https://nvd.nist.gov/vuln/detail/CVE-2022-22965  
 Summary: Remote code execution vulnerability affecting Spring Framework 5.2.x to 5.3.17 when deployed as a WAR on Tomcat with JDK 9+.

CVE-2020-36518 (Jackson-databind)  
 NVD: https://nvd.nist.gov/vuln/detail/CVE-2020-36518  
 Summary: Denial of Service in jackson-databind versions < 2.13.2.2 due to uncontrolled recursion.

CVE-2021-42550 (Logback)  
 NVD: https://nvd.nist.gov/vuln/detail/CVE-2021-42550  
 Summary: Logback before 1.2.9 allowed attackers to create crafted configuration files that could lead to code execution.

CVE-2020-1938 (Apache Tomcat AJP "Ghostcat")  
 NVD: https://nvd.nist.gov/vuln/detail/CVE-2020-1938  
 Summary: Apache Tomcat 9.0.x before 9.0.31 AJP connector file read/inclusion vulnerability; could expose webapp files.

**5. Mitigation Plan**

Interpret the results from the manual review and static testing report. Then identify the steps to mitigate the identified security vulnerabilities for Artemis Financial’s software application.

• Upgrade to a supported Spring Boot LTS (3.x) and re‑test. This will pull in newer Spring Framework/Jackson/Logback versions that address many CVEs.

• Remove hard‑coded secrets from source; externalize configuration using environment variables or a secret manager (e.g., Spring Cloud Config, HashiCorp Vault).

• Require TLS for the database connection; validate server certificates and use least‑privilege DB credentials.

• Introduce Spring Security with authentication and role‑based authorization. Protect all endpoints; add CSRF protections where relevant.

• Validate and sanitize all request parameters using Bean Validation annotations (@Valid, @Size, @Pattern) and centralize input validation.

• Harden file uploads or disable them. If required, drastically reduce size limits, store files outside the web root, enforce allow‑listed content types, and scan for malware.

• Enforce HTTPS and security headers (HSTS, X‑Content‑Type‑Options, X‑Frame‑Options, CSP). Use a reverse proxy (e.g., Nginx) if needed.

• Improve code quality and encapsulation: make fields private, add getters/setters with validation, and complete or remove placeholder classes. Add unit and integration tests.

• Implement SCA/SAST in CI (OWASP Dependency‑Check, Snyk, or similar) and adopt a regular dependency update cadence.

• Document a security baseline (aligned to OWASP ASVS) and add a developer checklist to prevent regression.