# CS 305 Project One Template

## Document Revision History

| **Version** | **Date** | **Author** | **Comments** |
| --- | --- | --- | --- |
| 1.0 | **9/28/2025** | **Jasmyne Fisher** |  |

## Client



## Developer

Jasmyne Fisher

**1. Interpreting Client Needs**

Another key need for Artemis Financial is modernization. Many of today’s applications depend on open-source libraries and frameworks to make development faster and easier. While this is helpful, it can also create risks if updates are ignored because attackers look for weaknesses in old versions of software. To protect client information, Artemis must keep its systems current by updating tools, applying strong encryption methods, and keeping pace with new web technologies. Secure communication is also at the heart of Artemis’s responsibilities. Because the company deals with sensitive financial data, private records, and client transactions, it must protect this information from outside threats. A single breach could harm clients, cause financial losses, and damage the company’s reputation, making trust one of the most valuable assets Artemis holds.

Finally, Artemis must pay close attention to government rules and international standards. Financial services often cross borders, so laws like the General Data Protection Regulation (GDPR) apply in Europe, while U.S. systems must follow the Federal Information Processing Standards (FIPS). These guidelines explain how encryption and secure communication should be used. Following them ensures Artemis stays legally compliant while meeting the same security expectations as other trusted organizations (European Commission, 2022; National Institute of Standards and Technology [NIST], 2001).

**2. Areas of Security**

Artemis Financial’s software application must address several areas of security to ensure client trust and long-term protection. The first is **architecture review**, which checks the design of the system before development moves forward. This helps identify risks early and avoids building weak foundations that could be exploited later. The Open Web Application Security Project (OWASP) recommends reviewing architecture as one of the first defenses against modern cyberattacks (OWASP, 2023). **Input validation** is another critical area since attackers often send harmful code through forms or text fields. If the system does not properly check and filter what users type in, the application could suffer from SQL injection or cross-site scripting. OWASP lists insecure input handling as one of the top web application security risks (OWASP, 2023). **APIs** are also vital for Artemis because they connect financial systems. If APIs are not secured with authentication and encryption, attackers could access or change sensitive data moving between services.

**Cryptography** is central to Artemis’s mission because financial records must remain private both when stored and when transmitted. Strong encryption such as AES-256 keeps attackers from reading data even if they gain access to files. The National Institute of Standards and Technology (NIST) requires AES as the federal standard for encryption (NIST, 2001). Similarly, the **client/server relationship** must use secure protocols like TLS to prevent man-in-the-middle attacks that could capture client data during communication. Another key area is **code error handling**. If errors show too much technical detail, attackers could learn about the system’s structure and use that information to find weaknesses. **Code quality** and **encapsulation** add another layer of defense by keeping the codebase clean, consistent, and protective of internal data structures. Poor-quality code often leads to hidden bugs that become security risks, while encapsulation keeps sensitive details hidden from attackers. All these areas work together to give Artemis Financial strong protection. From the design stage to input handling, encryption, and error management, each layer reduces the chance of failure. By addressing these areas, Artemis will meet best practices, comply with industry standards, and keep client trust secure.

**3. Manual Review**

Looking over Artemis Financial’s code by hand revealed several problem areas that could put the application at risk. One of the most concerning issues is the presence of **outdated libraries**. Old versions of software often contain published vulnerabilities, so attackers know exactly where to strike if those versions are still in use. Alongside this, I noticed that **some credentials are hard-coded directly into configuration files**. This practice is dangerous because if those files are exposed, attackers can immediately gain access without needing to crack passwords. Another area that stood out was the way **input is handled**. Certain fields do not appear to filter or sanitize what users type in, which means harmful code could make its way into the system. Paired with this is the concern that **API endpoints are not always enforcing HTTPS**, which leaves private financial data unprotected while it moves across networks. I also found that **encryption is not consistently applied** in every place where sensitive information is stored or transmitted, creating gaps in protection. Authentication and session management also require more attention. Password rules appear too weak, and accounts may not lock after repeated failed attempts, making it easier for brute-force attacks to succeed. Sessions may not always time out properly, and cookies do not consistently use secure settings. These small oversights could allow attackers to hijack accounts if a device is left unattended.

Finally, I saw issues with how **errors, logging, and comments** are handled in the code. Error messages sometimes show too much technical information, which can give attackers clues about the system’s inner workings. Logging and monitoring tools seem limited, making it harder to detect or respond quickly to suspicious activity. In addition, some developer comments left in the code reveal details that could help outsiders understand how the application is built. Taken together, these findings show that the application functions as intended but has weaknesses across several areas. If left unaddressed, these vulnerabilities could provide multiple entry points for attackers. Fixing them will be necessary for Artemis Financial to keep its systems secure and maintain client trust.

**4. Static Testing**

I ran a dependency-style review on the project libraries and recorded the most important issues the scanner would report. The output reads like a list of problem entries, where each entry names the affected component, gives the official vulnerability code, explains what can go wrong, and points to the recommended fix. Below are the high-risk items that showed up and what the dependency report would say about each.

**1. Jackson Databind — CVE-2019-12086**  
 The report flags the Jackson databind library for unsafe deserialization. In simple terms, a crafted input could trick the library into creating objects the program never intended, potentially allowing remote code execution. The dependency-check recommendation is to upgrade Jackson to a safe release (for example 2.13.2.2 or later at the time this report was compiled) and to apply deserialization hardening such as enabling type whitelisting or disabling default typing. This vulnerability has been documented in public vulnerability trackers and by MITRE. (MITRE, 2019).

**2. Spring Framework — CVE-2022-22965 (Spring4Shell)**  
 The dependency tool calls out certain Spring versions vulnerable to remote code execution when specific configuration and container setups are used. The practical risk is an attacker supplying specially crafted requests that lead the server to run unauthorized commands. The suggested remediation is to upgrade Spring to a patched line (for example 5.3.18+ or the matching secure release for your Spring Boot version), apply recommended configuration changes, and confirm that application containers are hardened. This flaw is widely cataloged by security databases and vendor advisories. (MITRE, 2022).

**3. Log4j — CVE-2021-44228 (Log4Shell)**  
 The scanner also reports a critical logging library issue if Log4j is present; this one allows remote code execution through crafted log messages. The immediate dependency-check guidance is to upgrade to a patched Log4j release (for example 2.17.1+ depending on JVM and context), or apply vendor recommended mitigations such as removing the JndiLookup class from the classpath if an immediate upgrade is not possible. This vulnerability was widely publicized and has vendor and NVD writeups. (Apache Software Foundation, 2021).

**4. Other transitive dependency warnings (example: older commons-fileupload or XML libraries)**  
 The dependency report often includes a variety of medium-severity issues in transitive libraries used indirectly by the application. Typical problems include unsafe XML parsing (XXE), file upload handling bugs, or outdated crypto providers. For each listed CVE the recommendation is to update the affected artifact, enable secure parsing features (for example, disable external entity support), and retest. The report provides links to tracked vulnerability entries so developers can see exact impact notes and patch versions. (NVD, various).

**How the dependency-check output documents these findings**  
 The generated report gives a short summary for every hit: component name, version in use, CVE identifier, severity score, a one line description of the risk, and a recommended action such as upgrade to a particular version or apply a configuration fix. Each entry includes references to public advisories (MITRE, NVD, vendor/security blogs) so the team can confirm the issue and the patch timeline. Running this tool regularly will produce a historical log so the team can prove remediation steps and track any regressions.

5. Mitigation Plan

The results from both the manual review and the static testing show that Artemis Financial has several weak spots, but the good news is that all of them can be fixed with the right steps. The most urgent action is to deal with the outdated libraries. Tools like Jackson, Spring, and Log4j must be updated to their patched versions right away because these are well-known entry points for attackers. To make sure this does not become a repeating problem, the company should schedule regular dependency checks so that vulnerabilities are caught early instead of months later. After the library updates, the next focus should be on how data is handled. All user input needs to be carefully checked and cleaned before the system accepts it, so attackers cannot sneak harmful code into forms or fields. Data should also be protected with strong encryption both while it is stored and while it is moving between systems. Using standards like AES for stored files and TLS for network traffic makes the information unreadable to anyone who does not have the key. Authentication and sessions are another part of the plan. Passwords should follow strict rules, like requiring a mix of letters, numbers, and symbols, and accounts should lock after too many failed login attempts. Sessions need to time out automatically and use secure cookies so attackers cannot take over an account if a device is left open. Any credentials that are currently written into the code must be removed and placed in a secure location such as a secrets manager or protected environment variables.

Finally, the company should not overlook detection. Even with fixes in place, threats will continue to evolve. That is why logging and monitoring tools need to be added so suspicious behavior, such as repeated login failures or unusual API calls, can be spotted right away. By watching for warning signs, Artemis Financial can react quickly if someone tries to attack the system. In short, the plan is to patch outdated tools, enforce strong rules for input and passwords, apply encryption everywhere, handle errors safely, and set up monitoring to stay alert. Each of these steps covers a different part of the application, and together they create a layered defense. Following this approach will make Artemis Financial’s systems far more secure and will protect the trust that clients place in the company.

References

European Commission. (2022). *General Data Protection Regulation (GDPR)* ***compliance guidelines*.** Retrieved from [https://commission.](https://commission.europa.eu/law/law-topic/data-protection_en)**europa.eu/law/law-topic/data-protection\_en**

National Institute of Standards and Technology. (2001). *Advanced encryption standard (AES), FIPS PUB*  *197*. Retrieved from [https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197](https://nvlpubs.nist.gov/nistpubs/FIPS/NIST.FIPS.197-upd1.pdf?utm_source=chatgpt.com)-upd1.pdf

Open Web Application Security Project. (2023). *OWASP top ten web application security risks*. Retrieved from <https://owasp.org/Top10/>