

Security Considerations

Author: Karan Mehta

Date: 2026-02-25

Table of Contents

Security Considerations	1
Authentication & Authorization	1
Data Integrity.....	1
API Security.....	2
Infrastructure Security	2
Future Production Hardening.....	2

Security Considerations

This document details the security measures implemented within the Minimum Viable Product (MVP) of our Banking Service and outlines the strategic hardening steps planned for a true production environment.

Authentication & Authorization

- **Stateless Authentication:** We utilize stateless JSON Web Tokens (JWT) passed as Bearer tokens to securely authenticate user requests.
- **Password Hashing:** Passwords are cryptographically hashed using `bcrypt` (via the `passlib` library) prior to storage, safeguarding against brute-force and dictionary attacks.
- **IDOR Prevention:** Insecure Direct Object Reference (IDOR) prevention is strictly enforced at both the router and service levels. Requests for sensitive resources (such as accounts and transactions) meticulously validate that the requested resource belongs to the authenticated user.

Data Integrity

- **Atomic Transactions:** All critical database operations, especially fund transfers, are wrapped in atomic database transactions using `session.commit()` and `session.rollback()`. This ensures operations either fully succeed or completely revert, avoiding partial states.
- **Integer-Based Currency:** To prevent floating-point manipulation and precision loss errors, all monetary amounts are represented and processed as integers (i.e., in cents).

- **SQL Injection Protection:** By utilizing the SQLAlchemy ORM for database interactions, queries are inherently parameterized, offering deep native protection against SQL Injection vulnerabilities.

API Security

- **CORS Configuration:** Our Cross-Origin Resource Sharing (CORS) setup is explicitly configured to restrict allowed origins, mitigating unauthorized cross-site requests.
- **Global Exception Handlers:** A global exception handling mechanism is configured to intercept internal errors and prevent sensitive internal system information or stack traces from being leaked to the client.
- **Frontend Session Timeout:** The platform includes a frontend session timeout mechanism that invalidates idle sessions, reducing the risk of unauthorized access on unattended devices.

Infrastructure Security

- **Environment Variable Management:** We strictly use `.env` files to configure secrets and application settings, successfully preventing the leakage of sensitive credentials into version control.
- **Container Security:** Our containerization strategy utilizes multi-stage Docker builds. This limits the deployed container to only essential runtime dependencies, keeping the application's attack surface minimal.

Future Production Hardening

As the application transitions to a real-world launch, the following enterprise-grade security enhancements will be implemented:

- **HTTPS/TLS Termination:** Utilizing a robust reverse proxy (e.g., NGINX, HAProxy, or a cloud Load Balancer) to terminate TLS and guarantee all data in transit is securely encrypted.
- **Rate Limiting:** Integrating Redis-backed rate limiting to protect critical endpoints (such as authentication and transaction routes) against automated brute-force attempts and DoS attacks.
- **Web Application Firewall (WAF):** Deploying a WAF to actively filter and monitor HTTP traffic, identifying and blocking common OWASP vulnerabilities and malicious payloads.
- **Database Encryption at Rest:** Enabling volume-level encryption (e.g., via AWS KMS or similar cloud-native tools) to ensure underlying persistent data remains encrypted and inaccessible even if the physical or virtual storage is compromised.