Blox

Abhishek Jamhoriya

Question) Banking works by transferring money from account A to account B. Most of the time account A is in one bank while account B is another bank. Implement the code to transfer money. Remember, payee's code runs on a different computer than that of the receiver's code.

1. **What are the issues in such a system?**
    - **Lack of Security:**
        1. **Sensitive Data Exposure:** The transfer system doesn't use encryption (such as HTTPS). If this system were used in a real-world scenario, sensitive data (like account numbers and amounts) would be exposed to eavesdropping or man-in-the-middle attacks.
        2. **Authentication and Authorization:** There is no authentication or authorization mechanism. Without verifying the identity of the sender or the receiver, anyone can send a transfer request.
    - **No Transaction Consistency (Atomicity):**
        1. **Risk of Partial Transfers:** If there's a failure between the sender and receiver (e.g., network issues, server crashes), the transfer might not complete successfully, leading to inconsistencies. For example, money may be deducted from the sender's account, but not credited to the receiver's account.
    - **Error Handling and Resilience:**
        1. **Limited Error Handling:** The system does basic error handling (e.g., invalid methods, failed encoding/decoding), but it lacks advanced mechanisms for retrying, dead-letter queues, or fallbacks in case of failure.
    - **Scalability:**
        1. **Single-Server Architecture:** The system assumes that both the sender's and receiver's services run on a single machine. This is not scalable for handling millions of transactions simultaneously.
    - **Data Integrity:**
        1. **Lack of Database Transactions:** There is no mechanism for ensuring the integrity of the sender's and receiver's account balances. If the data is stored in a database, any partial or failed transaction could leave account balances inconsistent.
    - **Logging and Monitoring:**

   i. **Limited Logging:** Although basic logging is present, it doesn't provide detailed information for tracking transaction states, failures, or performance metrics.

- **Lack of Input Validation:**
   i. **No Input Validation:** The system does not check for invalid amounts or other malformed inputs that might be malicious or just erroneous (e.g., negative amounts, excessively large transfers).

- **Concurrency Issues:**
   i. **Race Conditions:** If multiple transfers happen concurrently, there is a risk that two requests could try to modify the same account balance at the same time, leading to inconsistent data.

- **Lack of Auditing and Traceability:**
   i. **No Transaction Logging:** In a financial system, it’s crucial to maintain an audit trail for every transaction. There is no mechanism for logging the details of the transaction for later review or investigation in case of disputes.

- **No Handling of Time Zones or Delays:**
   i. **No Awareness of Time Zones:** The system doesn’t handle time zones or time-based requirements like scheduling transfers for a later time or handling currency exchange rates.

2. **What can we do to mitigate some of the issues ?**
   - **Lack of Security:**
      i. **Solution:** Implement HTTPS for encrypted communication. Use OAuth, API tokens, or other authentication mechanisms to ensure that only authorized parties can initiate and receive transfers.

   - **No Transaction Consistency (Atomicity):**
      i. **Solution:** Implement transactional consistency using two-phase commit (2PC) or eventual consistency mechanisms to ensure that both sides of the transaction either succeed or fail together.

   - **Error Handling and Resilience:**
      i. **Solution:** Implement robust error handling and retry mechanisms to recover from network or service failures. Introduce circuit breakers to prevent cascading failures in distributed systems.

   - **Scalability:**

    i. **Solution:** Deploy the services in a cloud environment with auto-scaling and load balancing. Use microservices architecture to allow each service to scale independently.

- **Data Integrity:**
    i. **Solution:** Use database transactions with appropriate isolation levels (e.g., ACID transactions in relational databases) to ensure that account balances are correctly updated.

- **Logging and Monitoring:**
    i. **Solution:** Implement structured logging, along with a centralized logging system like ELK (Elasticsearch, Logstash, Kibana) or an observability stack with Prometheus and Grafana to monitor the system's health, track failures, and analyze performance.

- **Lack of Input Validation:**
    i. **Solution:** Implement thorough input validation, including: Check if the transfer amount is valid (non-negative, reasonable range). Ensure the accounts exist and belong to the correct entities. Prevent malicious input like SQL injection (if using a database).

- **Concurrency Issues:**
    i. **Solution:** Use locks or other synchronization mechanisms to ensure that concurrent transfers are properly handled. For databases, this would involve row-level locking or optimistic concurrency control.

- **Lack of Auditing and Traceability:**
    i. **Solution:** Implement an audit trail to record every action performed by the system (e.g., sender, receiver, amount, timestamps). This helps with debugging and auditing transactions for fraud detection.

- **No Handling of Time Zones or Delays:**
    i. **Solution:** Make sure to handle time zone conversions if necessary and include mechanisms for scheduling transfers or handling deferred payments.

3. **Write the fixing yourself to demonstrate the mitigations.**

Sender's Bank Code

Enhancements:

- **Secure communication via HTTPS :** Ensures that all data exchanged between the sender and receiver is encrypted, protecting sensitive information (e.g., account details and transfer amounts) from interception by attackers.

- **Authentication using an API token :** Adds an API token in the Authorization header of the request, verifying the sender's identity.
- **Input validation for the transfer details :** Checks the validity of incoming data before processing:
    - Ensures amount is positive.
    - Ensures FromAccount and ToAccount are not empty.
- **Error retries for transient failures :** Ensures reliability by making multiple attempts to complete the transfer if a transient error occurs (e.g., server timeout or temporary disconnection).

Receiver's Bank Code

Enhancements:

- **Authentication verification :** The receiver validates the API token included in the sender’s request to ensure that only trusted parties can initiate transfers.
- **Input validation :** Confirms that the incoming transfer request contains valid and logical data, such as a valid FromAccount, ToAccount, and a positive Amount.
- **Atomicity for transaction processing :** Ensures that all steps in the transaction (e.g., deducting from one account and adding to another) are completed as a single, indivisible operation.
- **Logging for traceability :** Logs successful transactions and key events, creating an audit trail for troubleshooting and accountability.