Architecture Decision Records (ADRs)

ADR 1: Transaction Manager

Status: Accepted
Decision: Opt for a Transaction Manager as a central orchestration service to coordinate the different steps and interactions between microservices involved in transaction processing.
Context: A transaction process in a banking system can involve multiple steps and services. Effective coordination is necessary to ensure transaction integrity and a consistent user experience.
Considerations: For - Centralizes transaction management, reduces complexity for clients, and facilitates maintenance. Against - Creates a single point of failure and can become a bottleneck if not properly scaled.
Consequences: The Transaction Manager must be designed to handle high load and be highly available to minimize the risks of failure.

ADR 2: Transaction Validator

Status: Accepted
Decision: Use a Transaction Validator to verify the legitimacy and feasibility of transactions before processing.
Context: Validation is a crucial step to prevent fraud and ensure solvency of transactions.
Considerations: For - Ensures an additional layer of security and prevents inappropriate transactions. Against - Can introduce delays if the validation rules are too complex or the service is overloaded.
Consequences: The Transaction Validator must be optimized for high performance and integrate dynamic validation rules to adapt to different transaction requirements.

ADR 3: Currency Exchange

Status: Rejected
Decision: Develop a Currency Exchange service capable of calculating the necessary currency conversions for simulations and the validation of transactions.
Context: Multi-currency transactions are common in global banking operations and require precise conversions based on current exchange rates.
Considerations: For - Allows accurate simulations for clients and ensures correct conversions during validations. Against - Depends on real-time exchange rates that can vary rapidly.
Consequences: The Currency Exchange ADR has been rejected because we can directly ask the Stock exchange for simulation. It was only an intermediate, and it could crash while Stock Exchange is an external service always up (if it is not, we could use New York Stock Exchange instead of Paris Stock exchange ect).

ADR 4: Transaction Processing

Status: Accepted
Decision: Establish a Transaction Processing service to execute account updates following transaction validation and to initiate transactions with the stock exchange through the Trader.
Context: After validation, transactions need to be processed in a way that reflects changes in user accounts.
Considerations: For - Centralizes the processing of transactions after validation to ensure data integrity. Against - Requires close synchronization with other services and effective error management.
Consequences: The Transaction Processing must be highly reliable and capable of handling concurrent transactions consistently.

ADR 5: Trader

Status: Accepted
Decision: Implement a dedicated Trader service to perform transactions on the stock exchange on behalf of the bank.
Context: Some transactions require direct interaction with the stock exchange for the buying and selling of currencies.
Considerations: For - Allows the bank to respond agilely to market conditions and execute currency transactions in a timely manner. Against - Increases system complexity and introduces market risks.
Consequences: The Trader must be designed to operate reliably during stock exchange hours and appropriately manage transactions when the exchange is closed.

ADR 6: Update User Total

Status: Under Review
Decision: Update user.total_sold at every write or read of the user
Context: We use a “total_sold” variable to estimate if an user can pay without having to check all accounts each time
Considerations: We obviously need to update this number with each new transaction made by this user. But what if the user doesn’t use any account for a few months or/and the exchange rates variate a lot ? Problem : If we update before each transaction, it will take time (and it is useless to stock it in User), but if we don’t, we may not have the correct amount. Idea from the professor : we can update this number at each connection, when the user needs to see his accounts, so we don’t need to do it again before the transaction.
Consequences: We would need to update this total at each connection (before the transactions start) and after each transaction (after updating the account for the user).

ADR 7: Shared MongoDB Cluster

Status: Accepted

Decision: Implement a sharded cluster for the `transaction_manager`'s database.

Context: The transaction manager has its own database that experiences high demand, characterized by a very large number of requests. This heavy load has been impacting performance and scalability.

Considerations:

Scalability: Sharding the database will allow the system to handle more transactions by distributing the load across multiple database instances. This is crucial for maintaining performance as the number of transactions continues to grow.

Availability and Reliability: Implementing a sharded architecture can improve availability and fault tolerance. If one shard encounters an issue, the other shards can continue to operate, thereby reducing the risk of complete system downtime.

Complexity: Introducing sharding adds complexity to the system. This includes the complexity in managing multiple database instances, handling data distribution, and ensuring consistent transactions across shards.

Consequences:

Improved Performance and Scalability: The system will be able to scale more effectively to handle the increasing load, improving overall performance.

Increased System Complexity: The development and operations teams will need to manage the complexities introduced by a sharded database architecture. This includes ensuring the integrity and consistency of data across shards.

Enhanced Fault Tolerance: The risk of a complete system failure is reduced as the database load is distributed across multiple shards