# Simplified Stripe Implementation Report

## 1. System Components

- **Bank Servers:**
  - The bank server implementation manages user accounts and processes transactions.
  - Each bank server maintains a local database (JSON file) to store account information, including usernames, account numbers, and balances.
  - The bank server exposes gRPC services to allow the payment gateway to register clients, update client details, view balances, lock transactions, initiate transactions, and abort transactions.
  - The bank server uses etcd for service discovery, allowing the payment gateway to discover available bank servers.
- **Clients:**
  - The client implementation allows users to interact with the payment gateway.
  - Clients can register with the payment gateway, update their bank account details, view balances, initiate transactions, and view their transaction history.
  - Clients use gRPC to communicate with the payment gateway and provide authentication information via metadata.
  - Clients support offline payments by queuing requests locally and retrying them when the connection is restored.
- **Payment Gateway:**
  - The payment gateway acts as the central hub for processing transactions between clients and bank servers.
  - It provides gRPC services to clients for registering, updating details, viewing balances, initiating transactions, and viewing transaction history.
  - The payment gateway authenticates and authorizes client requests using gRPC interceptors and SSL/TLS mutual authentication.
  - It implements the 2PC protocol to ensure atomicity of transactions across multiple bank servers.
  - The payment gateway uses etcd to discover available bank servers.
  - It also maintains a database in `server.json`

## 2. Secure Authentication, Authorization, and Logging

- **Authentication:**
  - Mutual TLS (mTLS) is used for authentication.
  - The payment gateway and clients exchange certificates to verify each other's identity.
  - The payment gateway uses its server certificate and verifies client certificates against a Certificate Authority (CA).
  - Clients load their client certificate and key and trust the CA certificate to establish a secure connection.
  - gRPC is configured with TLS credentials to enforce secure communication.
  - I have not used JWT tokens, or any other form of authentication apart from mTLS, as this is a secure and robust method for authentication, and sufficient for the scope of this assignment.

- **Authorization:**
    - Authorization is implemented using a gRPC unary interceptor.
    - The interceptor extracts authentication information from the context (client certificate and metadata).
    - It verifies the client's credentials (username and password) against stored data in `server.json`.
    - Role-based access control is implemented to restrict access to certain methods. For example, only clients with the role "StrifeAdmin" can register new clients.
- **Logging:**
    - Logging is implemented using a gRPC unary interceptor.
    - The interceptor logs the incoming requests, the server's response, the status of the response, and any errors that occur.
    - The logs include the transaction amount, client identification (from certificate subject), method name, and any errors or exceptions.

## 3. Idempotent Payments

- Idempotency is ensured by assigning a unique transaction ID to each transaction.
- The payment gateway and bank servers check for existing transaction IDs before processing a transaction.
- If a transaction ID already exists, the transaction is considered a duplicate, and no further processing is performed.
- This approach prevents multiple deductions or unintended side effects in case of retries or network issues.

## 4. Offline Payments

- Clients maintain a queue of payment requests when they are offline.
- When the client regains connectivity, it automatically retries sending the pending payments from the queue.
- The client processes the queue in a separate goroutine, periodically checking for connectivity and attempting to send queued requests.
- Clients are notified about the success/failure of payments through the responses received from the payment gateway after retrying the requests.

## 5. 2PC with Timeout

- The 2PC protocol is implemented to ensure atomicity of transactions.
- The payment gateway acts as the coordinator, and the bank servers involved in the transaction act as voters.
- **Phase 1 (Prepare):**
    - The payment gateway sends a `LockTransaction` request to both sending and receiving bank servers to lock the transaction.
    - Bank servers check if the transaction is possible (e.g., sufficient balance) and respond with a `TransactionCheckResponse` indicating whether they are prepared to commit.
- **Phase 2 (Commit/Abort):**
    - If both bank servers respond positively, the payment gateway sends an `InitiateTransaction` request to both banks to commit the transaction.

    - If any bank server responds negatively or a timeout occurs, the payment gateway sends an `AbortTransaction` request to all involved banks to abort the transaction.
- Timeouts are implemented in the payment gateway when waiting for responses from bank servers. If a bank server does not respond within a configured timeout, the payment gateway aborts the transaction.

## 6. Implementation Details

- gRPC is used for communication between all components (clients, payment gateway, and bank servers).
- gRPC service definitions are defined in the provided proto files (`cl-gw.proto` and `gw-bank.proto`).
- etcd is used for service discovery, allowing the payment gateway to dynamically discover available bank servers.
- The payment gateway uses SSL/TLS mutual authentication for secure communication with clients.
- The system is designed to be fault-tolerant. Bank servers register themselves with etcd using a lease, which helps in case of server crashes.

## 7. Design Choices

- **Authentication:** Mutual TLS was chosen for strong authentication, ensuring that both the client and the server are verified.
- **Authorization:** gRPC interceptors provide a clean and efficient way to implement authorization, allowing for centralized control over access to different RPC methods.
- **Idempotency:** Using transaction IDs is a scalable and robust approach to ensure idempotency, as it does not rely on timestamps or other potentially unreliable factors.
- **Offline Payments:** Queuing payments at the client simplifies the handling of offline scenarios and ensures that payments are not lost.
- **2PC with Timeout:** 2PC guarantees atomicity, which is crucial for financial transactions. Timeouts are essential to prevent transactions from being blocked indefinitely in case of failures.
- **Service Discovery:** etcd is used for dynamic service discovery.

## 8. Failure Handling

- **Offline Payments:** * Clients handle offline scenarios by queuing payment requests and retrying them when connectivity is restored. * If a payment fails after retries, the client logs the failure and notifies the user.
- **2PC Timeouts:** * The payment gateway handles 2PC timeouts by aborting the transaction. * If a bank server does not respond within the timeout period, the payment gateway sends `AbortTransaction` requests to all involved bank servers to ensure that no partial transactions are committed.

## 9. Setup and Configuration

- Though not a requirement, the system needs to be set up with the necessary certificates and configurations to run successfully.
- The system requires the following components to be set up:
    - **Bank Servers:** Each bank server needs a unique port and ID. Bank server details are stored in `cmd/bank/databases/<bank_id>.json` files.

    - **Payment Gateway:** The payment gateway uses `cmd/gateway/server.json` to store user credentials and transaction history. The payment gateway connects to etcd to discover available bank servers. mTLS certificates are located in `cmd/gateway/`.
    - **Clients:** Each client needs a unique ID. Client data is stored in `cmd/client/databases/<client_id>.json`. mTLS certificates are located in `cmd/client/certs/`.
    - **Certificates:** Ensure that the necessary certificates (`ca.crt`, `server.crt`, `server.key`, `client1.crt`, `client1.key`, etc.) are generated and placed in the appropriate directories (`cmd/gateway/`, `cmd/client/certs/`). You can use a tool like `openssl` to generate these.
- The `setup.py` script can be used to generate all the necessary certificates and configuration files for the system.