Question3

(2023201038)

Strife - A Miniature Payment Gateway

Here's an in-depth explanation of the **design choices** for each requirement, along with **idempotency proof** and **failure handling mechanisms** for **offline payments** and **2PC timeouts**.

---

1. Design Choices for Each Requirement

1.1 Secure Authentication and Authorization

Design Choice:

- **Mutual TLS (mTLS)** is used to secure communication between **clients, the payment gateway, and banks**.
- **JWT-based authentication** is implemented to issue session-based tokens after a user logs in.
- **Role-based authorization** ensures that only authenticated users can:
    - View their balance.
    - Initiate payments.
    - View transaction history (optional).

Justification:

- **mTLS** prevents man-in-the-middle (MITM) attacks.
- **JWT** ensures stateless authentication, avoiding the need to maintain sessions.
- **gRPC Interceptors** are used to enforce authentication and authorization at the gateway.

1.2 Idempotent Payments

Design Choice:

- **Unique Transaction IDs** are assigned using a **distributed, timestamp-free, scalable ID generator** (inspired by **Snowflake IDs**).
- **Persistent transaction logs** ensure that duplicate transactions are not processed.
- **Replay attack detection** is implemented by checking if a transaction ID already exists in the logs before processing.

Justification:

- **Preventing Double Deductions:**
  If a transaction is retried due to a timeout, the same transaction ID ensures it isn't reprocessed.

Correctness Proof:

Assumptions:

1. Each transaction has a **unique transaction ID** assigned at creation.
2. Transactions are **only processed once**, even if retried due to a timeout.
3. Transactions are **logged after successful processing**.

Proof:

- Let **T** be a transaction.
- If **T1** and **T2** are two instances of the same transaction with **ID=X**, then:
  - Before processing **T2**, we check if **ID=X** exists in the transaction logs.
  - If **T1** was successful, **T2** is rejected as a duplicate.
  - If **T1** failed and was rolled back, **T2** is processed as a new attempt.
- **Ensures:** A transaction is applied exactly **once**.

1.3 Offline Payments

Design Choice:

- **Client maintains a queue** of transactions in a `pending_payments.json` file.
- **Automatic retry mechanism** periodically resends pending payments when the connection is restored.
- **Each transaction has a unique ID** to prevent double processing.

Failure Handling:

1. If the **client is offline**, payments are **queued** locally.
2. When back online, **only unprocessed transactions** are retried.

3. If the **payment gateway is reachable but the bank is down**, the transaction is **held at the gateway** for retry.

1.4 Two-Phase Commit (2PC)

Design Choice:

- The **Payment Gateway** acts as the **coordinator**.
- **Banks (sender & receiver)** act as **participants**.
- **Prepare Phase:**
  - The **sender bank** verifies available funds.
  - The **receiver bank** ensures the account exists.
  - Both banks **vote to commit or abort**.
- **Commit Phase:**
  - If both banks vote **commit**, the transaction is finalized.
  - If **any bank aborts**, the transaction is rolled back.

Failure Handling:

1. Handling 2PC Timeouts

- If **any bank does not respond** in the **prepare phase**, the transaction is **aborted**.
- If **one bank commits but the other does not**, the gateway **initiates a rollback**.

2. Ensuring Atomicity

- **Transactions are logged** so that in case of a crash, the state can be **restored**.
- If the **gateway crashes before commit**, banks detect the incomplete transaction and **rollback**.

How to run:

Quick Start Guide: Running Mini Stripe Payment System

Install Dependencies

Ensure **Python 3** is installed, then run:

pip install grpcio grpcio-tools protobuf

Run proto file

python3 -m grpc_tools.protoc -I. --python_out=. --grpc_python_out=. payment.proto

Start Bank Servers

Run each bank server in separate terminals:

python3 bank_server.py BankA 50052

python3 bank_server.py BankB 50053

python3 bank_server.py BankC 50054

Start Payment Gateway

python3 payment_gateway.py

Start Client

python3 client.py

Use the Client Menu

- **1️⃣ Check Balance**
- **2️⃣ Make a Payment**
- **3️⃣ Retry Pending Payments**
- **4️⃣ Logout & Exit**

Handle Offline Payments

If the gateway is down, payments are queued and retried automatically.