Technical Design and Architecture Document (TDAD): Personal P2P Payment Service (PPPS)

1. System Architecture: Transactional Layered Monolith

The PPPS will use a **Layered Spring Boot Monolith** designed around the **Service Layer's Transaction Manager**. The primary architectural focus is ensuring the **atomicity** of the P2P money movement operation.

Layer Focus: The TransferService

The core business logic resides in the TransferService. This service handles:

1. Input validation and PIN verification.
2. Concurrency control (Wallet locking).
3. Execution of the atomic database operation (debit, credit, logging).
4. Spring's @Transactional annotation will encapsulate the entire debit/credit/logging sequence.

2. Data Modeling (PostgreSQL)

The financial integrity of the system relies on two critical entities: **Wallet** (the mutable balance) and **LedgerEntry** (the immutable audit log).

2.1. Wallet Entity (The Mutable Balance)

Field Name	**Type**	**Constraints**	**Description**
id	UUID	PK, Generated	Internal wallet ID.
userId	UUID	FK to User	Link to the owner.
balance	BigDecimal	Required, >= 0	The current, mutable balance.
currency	String (3)	Default: NGN	Currency of the wallet.
version	Long	Optimistic Lock	Used by JPA for concurrency control (preventing lost updates).

2.2. Transaction Entity (The Master Log)

Field Name	**Type**	**Constraints**	**Description**

id	UUID	PK, Generated	Master reference ID for the entire operation.
senderWalletId	UUID	FK to Wallet	The source of the funds.
receiverWalletId	UUID	FK to Wallet	The destination of the funds.
amount	BigDecimal	Required	Total amount of the transfer.
status	Enum	PENDING, SUCCESS, FAILED	Final outcome of the transaction.
initiatedAt	Instant		Timestamp of the request.

2.3. LedgerEntry Entity (Double-Entry Audit)

Every successful transaction generates exactly two LedgerEntries (one DR, one CR).

Field Name	**Type**	**Constraints**	**Description**
id	UUID	PK, Generated	Unique ledger entry ID.
transactionId	UUID	FK to Transaction	Links to the master transaction.
walletId	UUID	FK to Wallet	The wallet affected by this entry.
entryType	Enum	DEBIT / CREDIT	**Crucial for accounting.**
amount	BigDecimal	Required	Amount of the specific entry.
createdAt	Instant		Timestamp of the ledger entry.

3. Key Technical Implementations

3.1 ACID Compliance and Concurrency Control

The primary P2P method must enforce atomicity:

1. **Spring @Transactional:** Applied to the TransferService.executeP2PTransfer() method to ensure database operations are treated as a single unit.
2. **Row-Level Locking:** When fetching the Sender's Wallet (Wallet A), the system must use SELECT FOR UPDATE (or its JPA equivalent) to lock the row. This prevents a second, simultaneous request from Wallet A from reading a stale balance and causing an overdraft.
3. **Overdraft Check:** After locking, the service validates Wallet A.balance >= requestedAmount.

3.2 Double-Entry Accounting Flow

For a successful transfer of N10,000:

1. **Update Wallet Balances:**
   - Wallet A balance is reduced by N10,000.
   - Wallet B balance is increased by N10,000.
2. **Log Ledger Entries:**
   - Create **LedgerEntry 1 (Debit)**: walletId=A, entryType=DEBIT, amount=10000.
   - Create **LedgerEntry 2 (Credit)**: walletId=B, entryType=CREDIT, amount=10000.
3. **Update Master Transaction:** Set Transaction.status = SUCCESS.
4. **Transaction Commit:** The entire database transaction commits.

3.3 Security Implementation

- **PIN Hashing:** User PINs must be salted and hashed using **Bcrypt** before storage.
- **Authentication:** Spring Security with JWT or API Key for user authentication (session management).
- **Authorization:** Ensure a user can only perform transfers from their own wallet.

4. System Design Diagram (P2P Atomic Transfer Flow)

The visualization below focuses specifically on the steps taken within the TransferService to ensure transactional integrity.
View PPPS Architecture Diagram (ppps_architecture_diagram.html)