1. RESTful API Design

We'll design an API with the following endpoints:

POST /transaction – Process a transaction (debit/credit).

GET /balance/{account_id} – Retrieve the balance for an account.

Key Considerations:

Atomicity: Each transaction should either be fully completed or fully rolled back.

Concurrency Handling: Use database transactions with proper isolation levels.

Error Handling: Return meaningful error messages when a transaction fails.

Performance Optimization: Indexing, caching, and batching of transactions.

2. Database Schema

A relational database like PostgreSQL or MySQL is ideal due to its strong consistency and ACID compliance.
Tables:

```
CREATE TABLE accounts (
    account_id SERIAL PRIMARY KEY,
    balance DECIMAL(20, 2) NOT NULL DEFAULT 0.00
);

CREATE TABLE transactions (
    transaction_id SERIAL PRIMARY KEY,
    account_id INT REFERENCES accounts(account_id),
    amount DECIMAL(20, 2) NOT NULL,
    transaction_type VARCHAR(10) CHECK (transaction_type IN ('debit', 'credit')),
    timestamp TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

3. Ensuring Consistency in Case of a Crash

Used Transactions: Wrap operations inside a transaction block to ensure atomicity.

Write-Ahead Logging (WAL): PostgreSQL uses WAL to recover in case of failure.

Optimistic Locking or Row-Level Locks: Prevent inconsistent updates.

4. Optimizations for High Performance

Indexing: Index frequently queried fields (account_id in transactions).

Batch Processing: Instead of processing one transaction at a time, batch multiple transactions.

Read Replicas: Use database replicas for balance inquiries to reduce load on the primary DB.

Caching: Use Redis to cache account balances for frequent requests.

Eventual Consistency with Queues: Offload non-critical operations to a queue like RabbitMQ.

Main Features of the Implementation

Ensuring Atomicity:

Uses SQLAlchemy's session.execute(text("BEGIN")) for database transactions.

Commits only if all operations succeed.

Rolls back in case of failure.

Concurrency Safety:

Uses with_for_update() to lock the account row, preventing race conditions.

Error Handling:

Proper HTTP status codes and messages for errors.