ZETA ASSIGNMENT

3. Scalable Banking API - Transaction Processing & Consistency

Zeta's backend must process millions of banking transactions daily while ensuring high consistency, low latency, and resilience to failures

Let's summarize what's built:
I've designed and implemented a scalable banking API for processing financial transactions with high consistency, low latency, and resilience. Here's what the implementation includes:

- RESTful API with FastAPI: Endpoints for account management Endpoints for debit, credit, and balance inquiry operations Clear error handling and validation
- Database Schema: Account table with optimistic locking via a version field Transaction table for recording all financial operations Proper constraints and indexes for data integrity and performance
- Concurrency Control Mechanisms: Database transactions for atomicity Row-level locking during balance updates Optimistic concurrency control with version numbers Proper isolation level (REPEATABLE READ)
- Performance Optimizations: Connection pooling for database connections Strategic indexes on frequently queried columns Async request handling for high throughput Configurable workers for horizontal scaling
- Reliability Features: Retry mechanism with exponential backoff Comprehensive error handling and logging Transaction status tracking Health check endpoints
- Development & Testing Tools: Docker and Docker Compose for easy setup Database migrations with Alembic Unit tests for critical functionality Load testing script for performance validation

The implementation follows best practices for financial systems:

- Ensures atomic operations (a transaction cannot be partially completed)
- Provides consistency guarantees even in case of crashes
- Handles concurrent requests efficiently with proper locking strategies
- Returns clear error messages in failure cases
- Is designed to scale horizontally for high load