Golang : Banking Ledger Application

You are developing a banking ledger service designed to reliably manage account operations, even under highload. The service must

- Support the creation of accounts with specified initial balances
- Facilitate deposits and withdrawals of funds¥Maintain a detailed transaction log (ledger) for each account
- Ensure ACID-like consistency for core operations to prevent double spending or inconsistent balances
- Scale horizontally to handle high spikes in transaction volume
- Integrate an asynchronous queue or broker to manage transaction requests efficiently
- Include a comprehensive testing strategy, covering feature tests and mocking for robust validation

Approach

Golang banking for core balance mutations, and uses an async queue to process transactions under high load.

Key Components:

- API Gateway (Go, chi or Gorilla Mux) for account creation and queuing transactions.
  - Proposed packages.
  - https://github.com/go-chi/chi
  - https://github.com/gorilla/mux
- Transaction Processor Worker (Go) consuming from **RabbitMQ**, applying balance updates in **PostgreSQL** with row-level locks and idempotency dedupe, then writing ledger entries to **MongoDB**.
  - PostgreSQL: github.com/jackc/pgx
  - Rabbit MQ: github.com/rabbitmq/amqp091-go
  - MongoDB: go.mongodb.org/
- **Docker Compose** to spin up containers for Postgres, MongoDB, RabbitMQ, API, and Worker. Initial configuration to be included.
- Scripts/Makefile for initialization compilation and misc. activities.
- Swagger File compliant to OpenAPI definition.
- Readme file for documentation.

- Testing: unit (with mocks go tests), integration (against ephemeral containers), and feature tests.
  - Will try to implement in given timelines, but considering it as optional.

High Level Architecture.