SPEC-1-Автоторговля-какао

Background

Цель — построить полностью автоматическую систему торговли фьючерсами на какао с возможностью быстрого переключения биржи (FORTS/CME/ICE/бумажный режим), инструмента (любой тикер фьючерса), и стратегии, при строгом разделении доменной логики, исполнения и инфраструктуры. Система должна быть расширяемой (портфель, несколько стратегий), тестируемой офлайн и безопасной за счёт Risk Engine.

Уточнённые вводные по MVP: - Первая целевая площадка/инструмент: **MOEX FORTS, фьючерс на какао CCH6**. Бумажный режим и реплей CSV используются для тестов/регрессии, но торговое исполнение — через адаптер FORTS. - Режим торговли: **внутридневный (без переноса позиций через ночь)**. Возможность переноса должна определяться **бизнес-логикой стратегии/ правилами риска**, а не архитектурой — архитектура остаётся нейтральной. - Базовые стратегии: VWAP Mean Reversion и Open Range Breakout; подключение новых стратегий — через плагинный интерфейс без изменения ядра. - Нефункциональные цели: безопасность капитала (дневной стоп, ограничение размера позиции, kill-switch), наблюдаемость (логи/метрики/алерты), конфигурация в YAML, детерминируемые бэктесты/реплей. Будущая расширяемость до портфеля и мульти-стратегий.

Requirements

Предпосылки и рамки

- Архитектура: Clean/Hexagonal с чётким разделением **стратегия ↔ домен ↔ порты ↔ адаптеры ↔ инфраструктура**.
- Ядро не знает о конкретной бирже/транспорте; взаимодействует только через порты.

MoSCoW

Must have 1. Полный цикл автоторговли на **FORTS CCH6** через **адаптер MOEX/Alfa Investments WebSocket**: приём маркет-данных, генерация сигналов, риск-проверки, исполнение, обновление позиций. 2. **Paper/Replay режим** для офлайн-тестов и регрессии (CSV реплей + симуляция исполнения). 3. Плагинообразные **стратегии** (VWAP Mean Reversion, ORB) через общий интерфейс `Strategy`. 4. **Risk Engine**: дневной стоп, ограничение размера позиции, запрет переворота, защита от double-send, kill-switch. 5. **Конфигурация YAML**: выбор биржи, инструмента, стратегии, параметров риска и контрактов без изменения кода. 6. Логирование, базовые метрики, алерты по критическим событиям (стоп-аут, дисконнект, ошибки исполнения). 7. **PersistencePort**: хранение сделок/позиций/PNL/параметров запуска для аудита и восстановления. 8. Строгое разделение **MarketDataPort / ExecutionPort / PositionPort / ClockPort / PersistencePort**. 9. **Веб-дашборд (Python Dash)**: статус системы, текущая позиция, PnL (реал./нереал.), активные заказы/сделки, лог событий, **JWT-авторизация**. 10. **Авторизация** в дашборде: **JWT (access+refresh)**, хранение пользователей в MariaDB. 11. **Управление стратегиями из UI**: добавить/редактировать/удалить стратегии и их параметры, запуск/ остановка.

Should have 1. Возможность переключения на **CME/ICE** через замену адаптера без изменения ядра/стратегий. 2. Пакет юнит-тестов домена и стратегий; тесты адаптеров с мок-портами. 3.

Простая панель наблюдаемости (CLI/минимальный веб-дашборд) для статуса, позиций, PnL. 4. Реплей из **PCAP/CSV** с контролем времени (ClockPort) для детерминированности.

Could have 1. Портфель из нескольких инструментов и запуск нескольких стратегий одновременно. 2. Расширенные метрики (latency, slippage, fill ratio), экспорт в Prometheus/ Timeseries БД. 3. Автоперезапуск/супервизор процессов.

Won't have (в MVP) 1. Высокочастотная арбитражная торговля и колокация. 2. Маржинальные/ кросс-биржевые спрэды и сложные оркестрации.

Примечание по коммуникациям

- По умолчанию **никаких межпроцессных брокеров** не требуется: адаптеры работают **в одном процессе** через интерфейсы портов.
- Если конкретный адаптер технологически вынесен во внешний процесс (например, QUIK LUA в терминале брокера), транспорт (gRPC/HTTP/ZeroMQ и т. п.) — это **деталь адаптера**, скрытая за портом; архитектура остаётся неизменной.

Инфраструктурные решения (MVP)

- **Runtime/оркестрация:** `pyjobkit` — для управления жизненным циклом задач, перезапусков, расписаний, graceful shutdown.
- **Хранилище (PersistencePort): MariaDB** (уже есть в вашем стеке) — журнал заявок/сделок, позиции, PnL, конфигурации запусков.
- **Формат конфигурации:** YAML.
- **Логи/метрики:** стандартный logging + экспорт метрик (например, в текст/JSON; интеграция с Prometheus — в Could).

- По умолчанию **никаких межпроцессных брокеров** не требуется: адаптеры работают **в одном процессе** через интерфейсы портов.
- Если конкретный адаптер технологически вынесен во внешний процесс (например, QUIK LUA в терминале брокера), транспорт (gRPC/HTTP/ZeroMQ и т. п.) — это **деталь адаптера**, скрытая за портом; архитектура остаётся неизменной.

Инфраструктурные решения (MVP)

- **Runtime/оркестрация:** `pyjobkit` — для управления жизненным циклом задач, перезапусков, расписаний, graceful shutdown.
- **Хранилище (PersistencePort): MariaDB** (уже есть в вашем стеке) — журнал заявок/сделок, позиции, PnL, конфигурации запусков.
- **Формат конфигурации:** YAML.
- **Логи/метрики:** стандартный logging + экспорт метрик (например, в текст/JSON; интеграция с Prometheus — в Could).

Method

Архитектурный обзор (Hexagonal)

```
@startuml
skinparam componentStyle rectangle
rectangle "STRATEGY LAYER" as SL
package "CORE DOMAIN" as CORE {
  [StrategyEngine]
  [RiskManager]
  [PositionService]
  [SignalRouter]
}
package "PORT INTERFACES" as PORTS {
  interface MarketDataPort
  interface ExecutionPort
  interface PositionPort
  interface ClockPort
  interface PersistencePort
}
package "ADAPTERS" as ADAPTERS {
  [MOEX FORTS Adapter (Alfa Investments WS)]
  [QUIK/LUA Adapter (optional)]
  [Paper Adapter]
  [CSV/Replay Adapter]
}
package "INFRA (pyjobkit / Config / Logging / Metrics)" as INFRA

SL --> CORE
CORE ..> PORTS
PORTS <.. ADAPTERS
CORE <.. INFRA : config/metrics/logging
@enduml
```

Ключевые интерфейсы (порты)

```
from typing import Protocol, Optional, Dict, Any
from dataclasses import dataclass
from enum import Enum
from datetime import datetime

class Signal(Enum):
    LONG = 1
    SHORT = -1
    FLAT = 0

@dataclass
class MarketState:
    price: float
```

```
    bid: float
    ask: float
    volume: float
    delta: float
    vwap: float
    time: datetime
    atr: Optional[float] = None

class Strategy(Protocol):
    def on_tick(self, state: MarketState) -> Optional[Signal]: ...

@dataclass
class OrderIntent:
    side: str  # "BUY" | "SELL"
    qty: int
    price: Optional[float]
    order_type: str  # "MARKET" | "LIMIT"
    reason: str
    client_order_id: Optional[str] = None  # для идемпотентности на уровне
адаптера

class MarketDataPort(Protocol):
    def get_state(self) -> MarketState: ...

class ExecutionPort(Protocol):
    def send(self, intent: OrderIntent) -> None: ...

class PositionPort(Protocol):
    def current(self) -> Dict[str, Any]: ...  # {qty, avg_price,
unrealized_pnl}

class ClockPort(Protocol):
    def now(self) -> datetime: ...

class PersistencePort(Protocol):
    def save_order(self, intent: OrderIntent, ts: datetime) -> int: ...
    def save_fill(self, order_id: int, fill_qty: int, fill_price: float, ts:
datetime) -> None: ...
    def save_position(self, qty: int, avg_price: float, ts: datetime) ->
None: ...
    def log_event(self, kind: str, payload: dict, ts: datetime) -> None: ...
    # для дашборда и CRUD стратегий
    def list_strategies(self) -> list: ...
    def upsert_strategy(self, name: str, params: dict) -> None: ...
    def delete_strategy(self, name: str) -> None: ...
```python
from typing import Protocol, Optional
from dataclasses import dataclass
from enum import Enum
from datetime import datetime
```
```

```python
class Signal(Enum):
    LONG = 1
    SHORT = -1
    FLAT = 0

@dataclass
class MarketState:
    price: float
    bid: float
    ask: float
    volume: float
    delta: float
    vwap: float
    time: datetime
    atr: Optional[float] = None

class Strategy(Protocol):
    def on_tick(self, state: MarketState) -> Optional[Signal]: ...

@dataclass
class OrderIntent:
    side: str  # "BUY" | "SELL"
    qty: int
    price: Optional[float]
    order_type: str  # "MARKET" | "LIMIT"
    reason: str
    client_order_id: Optional[str] = None  # для идемпотентности на уровне
адаптера

class MarketDataPort(Protocol):
    def get_state(self) -> MarketState: ...

class ExecutionPort(Protocol):
    def send(self, intent: OrderIntent) -> None: ...

class PositionPort(Protocol):
    def current(self) -> dict: ...  # {qty, avg_price, unrealized_pnl}

class ClockPort(Protocol):
    def now(self) -> datetime: ...

class PersistencePort(Protocol):
    def save_order(self, intent: OrderIntent, ts: datetime) -> int: ...
    def save_fill(self, order_id: int, fill_qty: int, fill_price: float, ts:
datetime) -> None: ...
    def save_position(self, qty: int, avg_price: float, ts: datetime) ->
None: ...
    def log_event(self, kind: str, payload: dict, ts: datetime) -> None: ...
```

Ядро (упрощённо)

```
class RiskManager:
    def __init__(self, max_loss_day: float, max_pos_size: int, forbid_flip:
bool):
        self.max_loss_day = max_loss_day
        self.max_pos_size = max_pos_size
        self.forbid_flip = forbid_flip
    def allow(self, intent: OrderIntent, position) -> bool:
        # проверки лимитов; запрет переворота; дневной стоп
        return True

class StrategyEngine:
    def __init__(self, strategy: Strategy, md: MarketDataPort, ex:
ExecutionPort,
                 pos: PositionPort, clock: ClockPort, store: PersistencePort,
risk: RiskManager):
        ...
    def tick(self):
        state = self.md.get_state()
        signal = self.strategy.on_tick(state)
        if not signal or signal == Signal.FLAT:
            return
        intent = self._map_signal(signal, state)
        if self.risk.allow(intent, self.pos.current()):
            oid = self.store.save_order(intent, self.clock.now())
            self.ex.send(intent)
            self.store.log_event("order_sent", {"id": oid, "intent":
intent.__dict__}, self.clock.now())
    def _map_signal(self, signal: Signal, state: MarketState) -> OrderIntent:
        # пример: лимит по цене VWAP или маркет при разрыве
        ...
```

Последовательность (on_tick → исполнение)

```
@startuml
actor Trader as T
participant StrategyEngine as SE
participant MarketDataPort as MD
participant RiskManager as RM
participant ExecutionPort as EX
participant PersistencePort as DB

T -> SE: tick()
SE -> MD: get_state()
MD --> SE: MarketState
SE -> SE: strategy.on_tick(state)
SE -> SE: map Signal -> OrderIntent
SE -> RM: allow(intent, position)
```

```
RM --> SE: ok/deny
SE -> DB: save_order(intent)
SE -> EX: send(intent)
EX --> SE: ack
SE -> DB: log_event(order_sent)
@enduml
```

Стратегии (плагины)

```python
class CocoaVWAPReversion(Strategy):
    def __init__(self, k: float, atr_window: int): ...
    def on_tick(self, s: MarketState):
        if s.atr and abs(s.price - s.vwap) > 1.2 * s.atr:
            return Signal.SHORT if s.price > s.vwap else Signal.LONG

class ORB(Strategy):
    def __init__(self, open_window: str, breakout_mult: float): ...
    def on_tick(self, s: MarketState):
        # при пробое диапазона открытия генерируем LONG/SHORT
        ...
```

Адаптеры

- **MOEX FORTS Adapter (Alfa Investments WebSocket, внутр.процесс)**
- MarketData: подписка на потоки через WebSocket (текущие цены/сделки/стакан по возможности API). Адаптер нормализует данные в `MarketState`.
- Execution: отправка/отмена заявок через предоставленный API брокера (прозрачно для ядра — WS/HTTP/др. неважно). Маппинг `OrderIntent → order` (LIMIT/MARKET/IOC/FOK при поддержке API). Подтверждения/сделки в `PersistencePort`.
- Надёжность: авто-reconnect, heartbeat/ping, **идемпотентность по** `client_order_id`, экспоненциальный backoff.
- Метаданные инструмента: тянуть из биржи при старте (tick size, lot, multiplier, сессии) с возможностью override в YAML; кэш в памяти и БД.
- **QUIK/LUA Adapter (опционально)**: альтернативный адаптер FORTS через LUA-скрипт в QUIK; транспорт (локальный TCP/IPC) скрыт внутри адаптера, порты неизменны.
- **Paper Adapter**: детерминированная симуляция заявок (latency, slip, partial fills).
- **CSV/Replay Adapter**: проигрывание исторических тиков/баров, синхронизация времени через `ClockPort`.

Runtime/Оркестрация (pyjobkit)

- **jobs**: `market_data_worker`, `execution_worker`, `strategy_worker`, `persistence_worker`, `replay_worker`, `dashboard_worker` **(Dash)**.
- Политики: авто-рестарт при фатальных ошибках, backoff, graceful shutdown; dep: `strategy_worker` зависит от `market_data` и `execution`.
- Конфиги задач: JSON/YAML, передаются в конструкторы адаптеров/ядра.

Dashboard (Python Dash)

- **Назначение**: наблюдаемость и управление стратегиями (CRUD), запуск/остановка.

- **Компоненты**:
- Статус узлов (pyjobkit jobs), состояние подключения к бирже.
- Текущая позиция, PnL (реал/нереал), таблицы orders/fills, поток логов.
- CRUD стратегий: форма параметров, список стратегий с `enabled` флагом; горячее применение параметров через graceful-перезапуск воркера.
- **Авторизация**: **JWT (access 60 мин, refresh 30 дней)**. `refresh` — HttpOnly cookie; **rotation + blacklist**. Роли: `admin` / `viewer`.
- **Постоянное наблюдение (24/7 wallboard)** — варианты: 1) **Silent refresh** через cookie: браузер зрителя держит долгую сессию (refresh 30д, auto-refresh access). Idle-logout отключаем для роли `viewer`. 2) **Kiosk-режим**: отдельный сервис-аккаунт `viewer_kiosk`; Dash не хранит токен в браузере, а запрашивает **server-to-server** токен у API (по client credentials) и проксирует данные на фронт. 3) **SSE/WebSocket-фид** от API с проверкой access-токена и автоматическим переподключением при ротации.
- **Слой API** (FastAPI):
- `/api/auth/login` (JWT выдача), `/api/auth/refresh` (ротация refresh), `/api/auth/logout` (инвалидация),
- `/api/status`, `/api/positions`, `/api/pnl`, `/api/orders`, `/api/fills` (GET),
- `/api/strategies` (GET/POST/PUT/DELETE), `/api/control/start`, `/api/control/stop` — только для `admin`.
- **Интеграция**: Dashboard вызывает API; API обращается к PersistencePort и к `runtime.controller` для команд запуска/остановки стратегии.

Конфигурация (пример YAML)

```yaml
exchange: forts
instrument: CCH6
strategy:
  name: cocoa_vwap_reversion
  params:
    k: 1.2
    atr_window: 14
contracts: 3
risk:
  max_day_loss: 0.02   # 2%
  max_position: 5
  forbid_flip: true
runtime:
  mode: live           # live|paper|replay
  replay_csv: data/CCH6_2025-01.csv
```

Схема БД (MariaDB) — `trade_core`

```sql
CREATE TABLE runs (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  started_at DATETIME(6) NOT NULL,
  exchange VARCHAR(16) NOT NULL,
  instrument VARCHAR(32) NOT NULL,
  strategy VARCHAR(64) NOT NULL,
  mode ENUM('live','paper','replay') NOT NULL,
```

```
  config_json JSON NOT NULL
);

CREATE TABLE strategies (
  name VARCHAR(64) PRIMARY KEY,
  params JSON NOT NULL,
  enabled TINYINT(1) NOT NULL DEFAULT 1,
  updated_at DATETIME(6) NOT NULL
);

CREATE TABLE users (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(64) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('admin','viewer') NOT NULL DEFAULT 'viewer',
  created_at DATETIME(6) NOT NULL
);

CREATE TABLE refresh_tokens (
  jti CHAR(36) PRIMARY KEY,
  user_id BIGINT NOT NULL,
  issued_at DATETIME(6) NOT NULL,
  expires_at DATETIME(6) NOT NULL,
  revoked TINYINT(1) NOT NULL DEFAULT 0,
  FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE orders (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  side ENUM('BUY','SELL') NOT NULL,
  qty INT NOT NULL,
  price DECIMAL(18,6) NULL,
  order_type ENUM('MARKET','LIMIT') NOT NULL,
  reason VARCHAR(128) NULL,
  client_order_id VARCHAR(64) NULL,
  ext_order_id VARCHAR(64) NULL,
  status ENUM('NEW','PARTIAL','FILLED','CANCELLED','REJECTED') DEFAULT 'NEW',
  UNIQUE KEY uniq_client (client_order_id),
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_orders_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE fills (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  order_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  fill_qty INT NOT NULL,
  fill_price DECIMAL(18,6) NOT NULL,
  ext_trade_id VARCHAR(64) NULL,
```

```
  KEY idx_order_ts (order_id, ts),
  CONSTRAINT fk_fills_orders FOREIGN KEY (order_id) REFERENCES orders(id)
);

CREATE TABLE positions (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  qty INT NOT NULL,
  avg_price DECIMAL(18,6) NOT NULL,
  realized_pnl DECIMAL(18,6) NOT NULL DEFAULT 0,
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_positions_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE risk_events (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  kind VARCHAR(32) NOT NULL,
  payload JSON NULL,
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_risk_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);
```sql
CREATE TABLE runs (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  started_at DATETIME(6) NOT NULL,
  exchange VARCHAR(16) NOT NULL,
  instrument VARCHAR(32) NOT NULL,
  strategy VARCHAR(64) NOT NULL,
  mode ENUM('live','paper','replay') NOT NULL,
  config_json JSON NOT NULL
);

CREATE TABLE strategies (
  name VARCHAR(64) PRIMARY KEY,
  params JSON NOT NULL,
  enabled TINYINT(1) NOT NULL DEFAULT 1,
  updated_at DATETIME(6) NOT NULL
);

CREATE TABLE users (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  username VARCHAR(64) UNIQUE NOT NULL,
  password_hash VARCHAR(255) NOT NULL,
  role ENUM('admin','viewer') NOT NULL DEFAULT 'viewer',
  created_at DATETIME(6) NOT NULL
);

CREATE TABLE orders (
```
```

```
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  side ENUM('BUY','SELL') NOT NULL,
  qty INT NOT NULL,
  price DECIMAL(18,6) NULL,
  order_type ENUM('MARKET','LIMIT') NOT NULL,
  reason VARCHAR(128) NULL,
  client_order_id VARCHAR(64) NULL,
  ext_order_id VARCHAR(64) NULL,
  status ENUM('NEW','PARTIAL','FILLED','CANCELLED','REJECTED') DEFAULT 'NEW',
  UNIQUE KEY uniq_client (client_order_id),
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_orders_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE fills (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  order_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  fill_qty INT NOT NULL,
  fill_price DECIMAL(18,6) NOT NULL,
  ext_trade_id VARCHAR(64) NULL,
  KEY idx_order_ts (order_id, ts),
  CONSTRAINT fk_fills_orders FOREIGN KEY (order_id) REFERENCES orders(id)
);

CREATE TABLE positions (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  qty INT NOT NULL,
  avg_price DECIMAL(18,6) NOT NULL,
  realized_pnl DECIMAL(18,6) NOT NULL DEFAULT 0,
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_positions_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE risk_events (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  kind VARCHAR(32) NOT NULL,
  payload JSON NULL,
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_risk_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);
```sql
CREATE TABLE runs (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  started_at DATETIME(6) NOT NULL,
```
```

```
  exchange VARCHAR(16) NOT NULL,
  instrument VARCHAR(32) NOT NULL,
  strategy VARCHAR(64) NOT NULL,
  mode ENUM('live','paper','replay') NOT NULL,
  config_json JSON NOT NULL
);

CREATE TABLE orders (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  side ENUM('BUY','SELL') NOT NULL,
  qty INT NOT NULL,
  price DECIMAL(18,6) NULL,
  order_type ENUM('MARKET','LIMIT') NOT NULL,
  reason VARCHAR(128) NULL,
  ext_order_id VARCHAR(64) NULL,
  status ENUM('NEW','PARTIAL','FILLED','CANCELLED','REJECTED') DEFAULT 'NEW',
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_orders_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE fills (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  order_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  fill_qty INT NOT NULL,
  fill_price DECIMAL(18,6) NOT NULL,
  ext_trade_id VARCHAR(64) NULL,
  KEY idx_order_ts (order_id, ts),
  CONSTRAINT fk_fills_orders FOREIGN KEY (order_id) REFERENCES orders(id)
);

CREATE TABLE positions (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  qty INT NOT NULL,
  avg_price DECIMAL(18,6) NOT NULL,
  realized_pnl DECIMAL(18,6) NOT NULL DEFAULT 0,
  KEY idx_run_ts (run_id, ts),
  CONSTRAINT fk_positions_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);

CREATE TABLE risk_events (
  id BIGINT PRIMARY KEY AUTO_INCREMENT,
  run_id BIGINT NOT NULL,
  ts DATETIME(6) NOT NULL,
  kind VARCHAR(32) NOT NULL,
  payload JSON NULL,
  KEY idx_run_ts (run_id, ts),
```

```
  CONSTRAINT fk_risk_runs FOREIGN KEY (run_id) REFERENCES runs(id)
);
```

Валидация сигналов и маппинг в заявки

- `Signal.FLAT` — не генерирует намерения.
- `LONG/SHORT` → функция `strategy_to_order(state, signal, sizing)` учитывает **tick size, lot size, contract multiplier**, скользит/лимит.
- RiskManager дополнительно валидирует: дневной PnL, `max_position`, запрет переворота (если включён), анти-дублирование (idempotency key на intent → order).

Implementation

1) Пакетирование

```
trade-core/
  domain/                # Signal, MarketState, Strategy, RiskManager,
StrategyEngine
  ports/                 # MarketDataPort, ExecutionPort, PositionPort,
ClockPort, PersistencePort
  strategies/            # cocoa_vwap_reversion.py, orb.py, ...
  adapters/
    moex_alfa_ws/       # market_data.py, execution.py, metadata.py,
session.py
    quik_lua/           # (опционально)
    paper/              # simulator.py
    csv_replay/         # reader.py
  infra/
    config.py           # YAML → dataclass
    logging.py, metrics.py
    persistence_maria/  # repo.py, migrations/
  runtime/
    jobs.py             # pyjobkit задачи
    main.py             # точка входа
  configs/
    forts_cch6_live.yaml
    forts_cch6_paper.yaml
```

2) Конфигурация (YAML, с подтверждёнными risk-параметрами)

```
exchange: forts
instrument: CCH6
strategy:
  name: cocoa_vwap_reversion
  params:
    k: 1.2
    atr_window: 14
contracts: 3
risk:
```

```
  max_day_loss: 0.02      # 2%
  max_position: 5
  forbid_flip: true
runtime:
  mode: live              # live|paper|replay
  replay_csv: null
persistence:
  mariadb_dsn: mysql+pymysql://user:pass@host:3306/trade_core
```

3) Адаптер MOEX/Alfa (константы сессии и метаданные)

- **Часы сессии — константа адаптера** (`adapters/moex_alfa_ws/session.py`):

```python
@dataclass
class SessionWindow:
    tz: str = "Europe/Moscow"
    day_open: str = "09:50"   # пример
    day_close: str = "18:45"
    eve_open: Optional[str] = None
    eve_close: Optional[str] = None
```

- **Метаданные инструмента** (`metadata.py`) подтягиваются из API брокера при старте и кэшируются в БД с возможностью override из YAML:

```python
@dataclass
class InstrumentMeta:
    tick_size: float
    lot: int
    multiplier: float
    symbol: str
```

4) Правило «не переносить через ночь»

- Реализуется **в стратегии/бизнес-логике**: во всех стратегиях вводим общий миксин `SessionAware`:

```python
class SessionAware:
    def flat_before_close(self, now, session: SessionWindow, minutes:
int = 5) -> bool:
        # если текущее время в зоне TZ входит в [close - minutes,
close], стратегия форсирует FLAT
        ...
```

- В адаптере часы сессии — константа; стратегия читает их через `ClockPort` + `SessionWindow` из адаптера (передаётся при инициализации стратегии через DI).

5) Идемпотентность заявок

- `OrderIntent.client_order_id` заполняется ядром (`uuid4()`), адаптер использует его при отправке и дедуплирует повторные send на реконнекте.

6) Persistence (MariaDB)

- Применить миграции (таблицы `runs`, `orders`, `fills`, `positions`, `risk_events` из раздела Method). Индексы по `(run_id, ts)` уже указаны.
- Репозитории: `save_order → id`, `update_order_status`, `append_fill`, `snapshot_position`, `log_risk_event`.

7) pyjobkit: задачи и зависимости

```
jobs = {
  "market_data": {
    "target": "adapters.moex_alfa_ws.market_data:run",
    "restart": "always", "backoff": [1,2,5,10]
  },
  "execution": {
    "target": "adapters.moex_alfa_ws.execution:run",
    "restart": "always"
  },
  "strategy": {
    "target": "runtime.main:run_strategy",
    "after": ["market_data", "execution"],
    "restart": "on-failure"
  },
  "persistence": {
    "target": "infra.persistence_maria.repo:run_flush",
    "restart": "on-failure"
  },
  "replay": {
    "target": "adapters.csv_replay.reader:run",
    "enabled": false
  }
}
```

8) Paper и Replay

- **Paper**: симулятор заявок учитывает `tick_size`, `latency_ms`, `slippage_ticks`, частичные исполнения; записывает филлы в Persistence.
- **Replay**: читает CSV/PCAP → нормализует в `MarketState`; `ClockPort` отдаёт «виртуальное время» из файла.

9) Тесты

- Юнит-тесты домена: маппинг сигнал→намерение, правила риска, идемпотентность.
- Контракты портов: фиктивные адаптеры для MarketData/Execution.
- Интеграционные: paper/replay прогоны сценариев (дневной стоп, запрет переворота, flat перед закрытием).

10) Наблюдаемость

- Логи: `order_intent`, `order_sent`, `order_fill`, `risk_blocked`, `reconnect`.
- Метрики: счётчики заявок/филлов/блокировок, время отклика Execution, задержка маркет-данных.

11) Развёртывание (MVP)

- Python 3.11, зависимости: `pydantic`, `websockets` / `httpx`, `pyjobkit`, `SQLAlchemy`, `PyMySQL`, `PyYAML`, `bcrypt`, `PyJWT`, `Dash`, `FastAPI`.
- Один контейнер/процесс; доступ к существующей MariaDB по DSN.

Gathering Results

Критерии приёмки MVP

- **Функциональность**:
- Live-торговля на FORTS CCH6 через адаптер MOEX/Alfa: ≥1 успешный полный цикл (order→fill) под контролем Risk Engine.
- Paper/Replay: воспроизводимый прогон стратегий на историческом CSV с тем же результатом PnL (±1 тик) при одинаковом конфиге.
- Dashboard (Dash) с **JWT**: вход/refresh/logout; страницы статус/позиция/PNL/orders/fills/лог; CRUD стратегий; start/stop.
- **Надёжность**: авто-reconnect WS/API; идемпотентность заявок (нет дублей по `client_order_id`) при реконнекте.
- **Риск-контроль**: срабатывание дневного стопа (2%), `max_position=5`, запрет переворота, авто-FLAT перед закрытием сессии.

KPI/метрики

- Исполнение: fill-ratio, средний slippage (в тиках), средняя задержка `send→ack` и `send→fill`.
- Стабильность: кол-во реконнектов/24ч, аптайм job-ов, частота ошибок Execution.
- Доходность (paper/live): PnL/день, волатильность PnL, max drawdown.

Процедуры валидации

1. **Regression suite (Replay)**: набор фиксированных CSV-кейсов; сравнение PnL/сигналов с эталоном (tolerance по тикам).
2. **Chaos-тест адаптера**: искусственный обрыв WS/HTTP, задержки, дубли ack — проверка идемпотентности и авто-восстановления.
3. **Risk-сценарии**: принудительное превышение дневного убытка, попытка переворота, попытка превысить `max_position` — ожидаемые блокировки/FLAT.
4. **UI-тесты (JWT)**: истечение access (60 мин), ротация refresh (30 дней), Kiosk-режим (server-to-server токен), разграничение ролей admin/viewer.

Наблюдаемость и алерты

- Алерты: дисконнект адаптера, отказ отправки заявки, превышение latency порога, срабатывание kill-switch.
- Дашборд: real-time SSE/WebSocket фид; страница «Сервис» со статусом pyjobkit jobs/версий конфигов.

Need Professional Help in Developing Your Architecture?

Пожалуйста, свяжитесь со мной на sammuti.com :)