**Модуль-за-модулем и файл-за-файлом**

**1) configs/**

**Файлы:**

* configs/.env, configs/.env.test
* configs/exec.yaml, configs/exec.test.yaml, configs/exec.example.yaml

**Роль:** ENV-переменные и исполняемая конфигурация (режимы, ключи, таймауты, риск).

**Сильные стороны:**

* Есть готовые ENV и YAML-настройки для разных окружений.
* Предусмотрены ключи для разных провайдеров и базовый risk-блок в ENV.

**Проблемы / слабости:**

* Несогласованность схем: exec.yaml (ключи provider/network/binance/simulated/ui) отличается от exec.example.yaml (executors.default\_mode) и exec.test.yaml (exchange/testnet/api/...). Это ломает предсказуемую загрузку конфигурации в утилитах.
* В .env лежат реальные ключи. Их нельзя коммитить/публиковать.

**Что исправить:**

1. **Единый формат exec.yaml**. Рекомендую такую схему (совместимую с текущими вызовами):
2. mode: binance # binance | sim | ui
3. binance:
4. testnet: true
5. recv\_window: 5000
6. timeout: 20
7. max\_retries: 5
8. backoff\_base: 0.5
9. backoff\_cap: 8
10. sim:
11. starting\_balance\_usdt: 10000
12. fee\_pct: 0.001
13. ui:
14. enabled: false
15. risk:
16. risk\_pct\_per\_trade: 0.01
17. daily\_max\_loss\_pct: 0.02
18. max\_trades\_per\_day: 15
19. deadman\_max\_stale\_sec: 90
20. tz\_name: Asia/Almaty

И удалить/унифицировать exec.test.yaml/exec.example.yaml.

1. **Секреты** оставить только в .env/.env.test вне VCS. Пример требуемых переменных для тестнета:
2. BINANCE\_TESTNET\_API\_KEY=...
3. BINANCE\_TESTNET\_API\_SECRET=...
4. DB\_URL=sqlite+aiosqlite:///./data/ai\_trader.db
5. LOG\_LEVEL=INFO

Остальное можно подтягивать из exec.yaml.

**2) db/**

**Файлы:**

* db/session.py
* db/models.py
* db/crud.py
* db/crud\_orders.py
* db/models\_orders.py ← **внимание, тут проблема**

**Роль:** подключение БД (SQLite+aiosqlite), модели OHLCV, CRUD по свечам и журналу ордеров.

**Сильные стороны:**

* session.py: аккуратный init, PRAGMA (WAL, synchronous, cache), health-check, единый get\_session. 👍
* models.py: модель OHLCV со стабильной бизнес-уникальностью (symbol, timeframe, ts) и полезными индексами.
* crud.py: батчевый upsert OHLCV и выборки — по делу.

**Критические проблемы:**

* **Нет ORM-модели журнала ордеров**. crud\_orders.py и *особенно* db/models\_orders.py сейчас содержат фрагменты **CRUD-кода вместо определения модели** OrderLog. Из-за этого:
  + services/reconcile.py, routers/trading\_exec.py (при наличии БД) не смогут писать/читать журнал.
  + Тесты и последующая аналитика/сверка пост-фактум будут неполноценны.

**Что исправить:**

1. **Создать/переписать db/models\_orders.py** на полноценную модель OrderLog. Пример готового файла:

# db/models\_orders.py

from \_\_future\_\_ import annotations

from datetime import datetime, timezone

from typing import Optional

from sqlalchemy import String, Float, Integer, BigInteger, Index, text

from sqlalchemy.orm import Mapped, mapped\_column

from .session import Base

class OrderLog(Base):

\_\_tablename\_\_ = "order\_log"

id: Mapped[int] = mapped\_column(Integer, primary\_key=True, autoincrement=True)

# внешние/биржевые идентификаторы

exchange: Mapped[str] = mapped\_column(String(16), nullable=False, server\_default="binance")

testnet: Mapped[int] = mapped\_column(Integer, nullable=False, server\_default=text("1")) # 1/0

order\_id: Mapped[Optional[str]] = mapped\_column(String(64), nullable=True) # биржевой id

client\_order\_id: Mapped[Optional[str]] = mapped\_column(String(64), nullable=True) # newClientOrderId

# базовая торговая информация

symbol: Mapped[str] = mapped\_column(String(24), nullable=False, index=True) # BTCUSDT

side: Mapped[str] = mapped\_column(String(4), nullable=False) # BUY | SELL

type: Mapped[str] = mapped\_column(String(16), nullable=False) # MARKET | LIMIT | ...

status: Mapped[str] = mapped\_column(String(16), nullable=False, server\_default="NEW")

price: Mapped[float] = mapped\_column(Float, nullable=False, server\_default=text("0"))

qty: Mapped[float] = mapped\_column(Float, nullable=False, server\_default=text("0")) # base qty

filled\_qty: Mapped[float] = mapped\_column(Float, nullable=False, server\_default=text("0"))

quote\_qty: Mapped[float] = mapped\_column(Float, nullable=False, server\_default=text("0")) # quote spent/recv

# таймштампы

ts\_ms: Mapped[Optional[int]] = mapped\_column(BigInteger, nullable=True) # event time от биржи

created\_at: Mapped[datetime] = mapped\_column(nullable=False, default=lambda: datetime.now(timezone.utc))

updated\_at: Mapped[datetime] = mapped\_column(nullable=False, default=lambda: datetime.now(timezone.utc))

# сериализация упрощённая (можно расширить по необходимости)

def as\_dict(self) -> dict:

return {

"id": self.id,

"exchange": self.exchange,

"testnet": bool(self.testnet),

"order\_id": self.order\_id,

"client\_order\_id": self.client\_order\_id,

"symbol": self.symbol,

"side": self.side,

"type": self.type,

"status": self.status,

"price": float(self.price),

"qty": float(self.qty),

"filled\_qty": float(self.filled\_qty),

"quote\_qty": float(self.quote\_qty),

"ts\_ms": self.ts\_ms,

"created\_at": self.created\_at.isoformat(),

"updated\_at": self.updated\_at.isoformat(),

}

Index("ix\_orderlog\_symbol\_created", OrderLog.symbol, OrderLog.created\_at)

1. **Привести db/crud\_orders.py в соответствие** (создание записи из биржевого ответа, выборки последнего/по периоду, обновление статуса). Сейчас файл «обрезан» — дописать функции create(...), get\_last(...), get\_between(...), update\_status(...), используя OrderLog.
2. Вызовы записи журнала подключить в местах исполнения ордеров (см. ниже routers/trading\_exec.py).

**3) executors/**

**Файлы:**

* executors/base.py
* executors/simulated.py
* executors/ui\_agent.py
* executors/api\_binance.py ← **критическая ошибка синтаксиса**

**Роль:** Исполнители ордеров: абстрактный интерфейс, симулятор, заглушка UI, реальный Binance spot.

**Сильные стороны:**

* Интерфейс Executor хорошо очерчивает контракт (open/close, positions, balance).
* SimulatedExecutor удобен для быстрой дымовой проверки API (есть позиции, баланс).
* api\_binance.py: продуманная обвязка (recvWindow, ретраи с backoff, клиент \_BinanceSpotClient, нормализация фильтров символа).

**Критические проблемы:**

* **Синтаксическая ошибка** в executors/api\_binance.py:
* async def \_apply\_symbol\_rules(... ) -> Tuple[Optional[float], Optional[float], Optional[float], Optional[float]]]:
* ^ лишняя ']'

Из-за лишней ] модуль **не импортируется**, и все роуты, завязанные на Binance, падают.

**Что исправить (обязательно):**

1. Исправить сигнатуру:

* ) -> Tuple[Optional[float], Optional[float], Optional[float], Optional[float]]]:
* ) -> Tuple[Optional[float], Optional[float], Optional[float], Optional[float]]:

1. Пройтись по методам open\_order, open\_with\_protection, get\_positions, fetch\_balance:
   * Убедиться, что:
     + при type="MARKET" используем quoteOrderQty (если пользователь задаёт quote\_qty) **или** шаг MARKET\_LOT\_SIZE/LOT\_SIZE для qty.
     + для LIMIT|STOP\_LOSS\_LIMIT|TAKE\_PROFIT\_LIMIT корректно применяются PRICE\_FILTER.tickSize и LOT\_SIZE.stepSize.
     + newClientOrderId пробрасывается из API.
   * Возвращаемое тело унифицировать к виду, который ожидают роуты (exchange, testnet, order\_id, client\_order\_id, symbol, side, type, price, qty, status, raw).
2. В клиенте \_BinanceSpotClient проверить:
   * Базовый URL: testnet — https://testnet.binance.vision (у тебя так и есть).
   * Подпись и X-MBX-APIKEY на приватные методы OK.
   * Расширенные ошибки Binance → бросать BinanceAPIError (уже сделано).

**4) routers/**

**Файлы:**

* routers/trading\_exec.py ← REST API торговли
* routers/trading.py ← анализ/бэктест (через БД)
* routers/ohlcv.py ← загрузка/выгрузка свечей (CSV/JSON, источники)
* routers/ui.py ← простой UI через Jinja

**Роль:** Публичные HTTP-эндпоинты.

**Сильные стороны:**

* trading\_exec.py: есть режимы binance|sim, soft-импорты (не валится без БД/риск-модулей), JSON-ответы выровнены.
* ohlcv.py: валидации, безопасные имена файлов для CSV, разделение источников данных.
* ui.py: фронт простого мониторинга (баланс/позиции).

**Проблемы / что допилить:**

1. **Журналирование ордеров в БД.** После фикса db/models\_orders.py:
   * В точках open/close (успешный ответ) вызывать crud\_orders.create(...) с нормализацией тела ответа от исполнителя.
   * При отменах/ошибках — писать status="REJECTED"/"CANCELED".
2. **Риск-чеки перед выставлением ордера**:
   * Сейчас код импортирует utils.risk\_config и state.daily\_limits; убедиться, что проверка can\_open\_more\_trades(...) и дневного стопа вызывается **до** отправки ордера в Binance, и при отказе возвращается 4xx с причиной.
   * При исполнении/закрытии — register\_new\_trade(...)/register\_realized\_pnl(...).
3. **Единая нормализация входных параметров**:
   * В хелпере merge\_params (у тебя уже есть подобный) аккуратно сводить qty/quote\_qty/price/timeInForce/type/side, валидировать их набор (например, для MARKET не требовать price).

**5) src/**

**Файлы:**

* src/main.py (FastAPI-приложение, middlewares, разделы роутов)
* src/data\_loader.py (унифицированная загрузка котировок)
* src/analysis/analyze\_market.py + src/indicators.py, src/strategy.py, src/paper.py

**Сильные стороны:**

* main.py: аккуратные фиче-флаги, список /endpoints на /, health-роуты, GZip/TrustedHost/CORS.
* data\_loader.py: кэширование, адаптеры под разные источники, удобно мокается в тестах.
* analysis/strategy/indicators: компактные, тестируемые, сигналы/агрегации — базовый бэктест/буковки есть.

**Замечания:**

* В main.py проверь, что **все подключаемые роутеры включены** в приложение в зависимости от флагов FEATURES (особенно exec/ohlcv/ui) и что их импорты не падают, если БД/модули отсутствуют (у тебя уже «мягкие» импорты).
* В data\_loader.py убедиться, что ALPHAVANTAGE\_KEY читается только при source=alphavantage, чтобы не блокировать /prices без ключа.

**6) utils/**

**Файлы:** utils/secrets.py, utils/risk\_config.py

**Сильные стороны:**

* secrets.py: есть load\_env\_files(...), get\_binance\_keys(...), load\_exec\_config(...) — правильный путь.
* risk\_config.py: агрегирует конфиг риска из ENV/YAML.

**Проблемы / что допилить:**

* Сопоставь load\_exec\_config(...) с **новой единой схемой** exec.yaml (см. блок configs). Добавь разбор mode/binance/sim/ui/risk.
* Проверь, что get\_binance\_keys(testnet=...) корректно ходит по алиасам ENV (сейчас есть), и **не** смотрит в устаревшую схему exec.test.yaml.

**7) risk/ и state/**

**Файлы:**

* risk/risk\_manager.py, risk/deadman.py, risk/trailing.py (пустой)
* state/daily\_limits.py

**Сильные стороны:**

* daily\_limits.py: хранение состояния в data/state/daily\_limits.json, расчёт начала дня по таймзоне, лимиты — ок.
* risk\_manager.py: интерфейсы к исполнителю через Protocol, расчёт размера позиции от риска — по делу.
* deadman.py: heartbeat-файл data/state/heartbeat.txt и контроль «пульса».

**Замечания / что допилить:**

* risk/trailing.py пуст. Если trailing не нужен прямо сейчас — **оставить на потом** (низкий приоритет).
* Включить deadman.check\_deadman(...) в периодическую задачу или вызовы после inactivity (см. SERVICES).

**8) services/**

**Файлы:** services/trading\_service.py, services/reconcile.py, services/protection.py

**Роль:** сервисный слой (оркестрация торговли, сверка позиций с биржей, защита).

**Сильные стороны:**

* reconcile.py: есть план сверки (бинанс → локальный журнал), fallback на симулятор при отсутствии.
* trading\_service.py: каркас конфигурации, обвязка под несколько исполнителей, RiskLimits.

**Что допилить:**

* После появления OrderLog — дописать **агрегации журнала** и сверки (journal\_between(...) в reconcile) + подпитка UI.

**9) sources/**

**Файлы:** sources/binance.py, sources/alpha\_vantage.py, sources/ohlcv.py, sources/base.py

**Роль:** источники для routers/ohlcv.py (не торговые).

**Сильные стороны:**

* Понятная обвязка, бект-офф при ошибках Binance.

**Замечания:**

* Эти источники **не влияют на торговое исполнение**. Их можно оставлять в текущем виде, фокусируясь на executors/api\_binance.py и routers/trading\_exec.py.

**10) news/\*, templates/\*, static/\*, htmlcov/\*, tests/\***

* news/\* — опциональный функционал, в демо-торговле не критичен.
* templates/\*, static/\* — UI-обвязка, ок.
* htmlcov/\* — артефакты coverage (игнор).
* tests/\* — набор юнитов/смоуков. Полезно для регрессии (см. ниже раздел «Как протестировать локально»).

**Сводный чек-лист готовности**

**Готово ✅**

* Базовая структура FastAPI-сервиса (src/main.py, middlewares, health).
* Симулятор исполнения (executors/simulated.py) и API-роуты вокруг него.
* OHLCV-модель и CRUD, выгрузка/загрузка /ohlcv, /prices (без торговли).
* Risk-состояние дня (state/daily\_limits.py), каркас риск-менеджера.

**Требует доработки ⚠️**

* executors/api\_binance.py: **исправить синтаксис** и довести фильтры символа / тела запросов.
* db/models\_orders.py + crud\_orders.py: **реализовать ORM-модель** OrderLog и CRUD-операции.
* routers/trading\_exec.py: интегрировать журналирование ордеров и принудительные риск-чеки до отправки ордера.
* utils/secrets.py: унифицировать разбор единого exec.yaml.

**Отсутствует ❌ / на потом**

* risk/trailing.py (реализация трейлинг-логики).
* Расширенная сверка services/reconcile.py (завяжется на появление журнала).

**Риски, если не доделать:**

* Без фикса api\_binance.py — **вообще** не заработает режим binance (падение импорта).
* Без OrderLog — нет «чёрного ящика» сделок: нельзя нормально отлаживать, сверять, считать PnL и ограничивать поведение — критично для боевого тестирования.
* Без унификации exec.yaml — путаница источников конфигурации, ошибки в ключах/режимах.

**Абсолютный приоритет (минимум, чтобы выйти на демо)**

1. **FIX:** executors/api\_binance.py — убрать лишнюю ] в сигнатуре \\_apply\_symbol\_rules (и быстро прогнать линтер/импорт).
2. **ADD:** db/models\_orders.py — ввести модель OrderLog (код выше), затем await init\_db\_schema() при старте/первом запросе.
3. **WIRE:** routers/trading\_exec.py — перед open вызвать риск-чек (can\_open\_more\_trades, дневной стоп), после успешного ответа — crud\_orders.create(...).
4. **CONFIG:** унифицировать configs/exec.yaml, а загрузку в utils/secrets.py подстроить под новую схему.
5. **SMOKE:** прогнать симуляторные эндпоинты (mode=sim) и затем mode=binance&testnet=true на **балансе/позициях** (без денег/микро-ордером).

**Можно отложить:**

* Трейлинг, расширенная сверка, UI-полировка, агрегации журналов для витрин, ньюс-модуль.

**Как протестировать проект локально**

**Установка**

# Python 3.11

python -m venv .venv

. .venv/Scripts/activate # Windows

# . .venv/bin/activate # Linux/Mac

pip install -U pip

pip install -r requirements.txt

**ENV**

Создай configs/.env (или используй твой) c минимумом:

DB\_URL=sqlite+aiosqlite:///./data/ai\_trader.db

LOG\_LEVEL=INFO

# Binance testnet

BINANCE\_TESTNET\_API\_KEY=...

BINANCE\_TESTNET\_API\_SECRET=...

# Риск

RISK\_PCT\_PER\_TRADE=0.01

DAILY\_MAX\_LOSS\_PCT=0.02

MAX\_TRADES\_PER\_DAY=15

DEADMAN\_MAX\_STALE\_SEC=90

TZ\_NAME=Asia/Almaty

Ключи **не коммитить**.

**Запуск API**

set PYTHONPATH=%CD% # Windows (в корне проекта)

uvicorn src.main:app --reload --host 127.0.0.1 --port 8001

# или run-dev.bat

**Быстрые проверки (HTTP)**

* Пинг/здоровье:
* GET http://127.0.0.1:8001/ping
* GET /healthz
* Симулятор:
* GET /exec/balance?mode=sim&testnet=true
* POST /exec/open?mode=sim&testnet=true&symbol=BTCUSDT&side=buy&type=market&qty=0.001
* GET /exec/positions?mode=sim&testnet=true
* POST /exec/close\_all?mode=sim&testnet=true&symbol=BTCUSDT
* Binance Testnet (после правок):
* GET /exec/balance?mode=binance&testnet=true
* GET /exec/positions?mode=binance&testnet=true
* # микро-ордер:
* POST /exec/open?mode=binance&testnet=true&symbol=BTCUSDT&side=buy&type=market&quote\_qty=10
* POST /exec/close\_all?mode=binance&testnet=true&symbol=BTCUSDT

**Pytest (то, что уже есть)**

pytest -q tests/test\_ui\_agent\_stub.py

pytest -q tests/test\_prices\_endpoint.py

pytest -q tests/test\_indicators.py tests/test\_strategy.py tests/test\_analyze\_market.py

# когда БД готова:

pytest -q tests/test\_ohlcv.py

# когда exec/binance починен:

pytest -q tests/test\_execution\_api.py -k "sim or binance"

**Как корректно выйти в «боевое тестирование» на Binance Testnet**

1. **Правки кода (минимум):**
   * Исправить синтаксис в executors/api\_binance.py.
   * Ввести db/models\_orders.py: OrderLog и доделать crud\_orders.py.
   * В routers/trading\_exec.py:
     + Перед вызовом исполнителя → load\_risk\_config() + load\_state() → can\_open\_more\_trades()/проверка дневного стопа.
     + После успеха → crud\_orders.create(...) (лог ордера).
   * Унифицировать exec.yaml; utils/secrets.get\_binance\_keys() оставить источником №1 (ENV), YAML — как резерв.
2. **ENV / ключи:**
   * Убедиться, что **тестнет-ключи** лежат в configs/.env и подхватываются.
   * Проверить, что DB\_URL указывает на sqlite+aiosqlite:///./data/ai\_trader.db.
3. **Инициализация БД:**
   * На старте приложения (или отдельно) вызвать await init\_db\_schema() (в src/main.py — в lifespan-хуке или первом запросе к БД), чтобы создать order\_log и ohlcv.
   * Убедиться, что каталог data/ существует и доступен для записи.
4. **Дымовые тесты:**
   * Прогнать mode=sim (баланс → open → positions → close\_all).
   * Прогнать mode=binance&testnet=true:
     + GET /exec/balance должен вернуть актуальные free/locked.
     + POST /exec/open ... quote\_qty=10 — минимум влияния, без ручного расчёта лота.
     + GET /exec/positions — отразит чистое количество по символу.
     + POST /exec/close\_all — вернуть позицию к нулю.
5. **Проверка риск-менеджмента:**
   * В .env установи жёсткие параметры (например, MAX\_TRADES\_PER\_DAY=1, DAILY\_MAX\_LOSS\_PCT=0.0001) и убедись, что второй ордер в день возвращает отказ (4xx) и понятное сообщение.
   * Проверить запись data/state/daily\_limits.json (дата, стартовый equity, счётчики сделок, накопленный PnL).
6. **Пульс/Deadman:**
   * Включить периодическое touch\_heartbeat() (раз в 30–60с) в фоновом таске.
   * Проверить, что при «протухшем» heartbeat (искусственно не обновлять) check\_deadman(...) срабатывает (для симулятора — логирование; для бинанса — план закрытия позиций).

**План перехода в демо-боевой режим (пошагово)**

1. **Быстрые правки кода (сразу):**
   * executors/api\_binance.py — убрать лишнюю ] в сигнатуре \_apply\_symbol\_rules.
   * Добавить db/models\_orders.py (модель OrderLog) и довести db/crud\_orders.py (create/get\_between/update).
   * В routers/trading\_exec.py:
     + до ордера: can\_open\_more\_trades/дневной стоп;
     + после ордера: crud\_orders.create запись журнала.
   * Унифицировать configs/exec.yaml; проверить utils/secrets.load\_exec\_config.
2. **Подготовка окружения:**
   * pip install -r requirements.txt
   * configs/.env с тестнет-ключами и DB\_URL (SQLite aiosqlite).
   * Создать data/ и права на запись.
3. **Инициализация БД:**
   * init\_db\_schema() → таблицы ohlcv, order\_log.
   * Проверка db\_healthcheck.
4. **Смоук-прогон:**
   * Симуляторные эндпоинты: open/positions/close\_all.
   * /prices и /ohlcv/\* (без ключей AlphaVantage, чтобы не мешало).
5. **Тестнет-прогон:**
   * /exec/balance?mode=binance&testnet=true
   * /exec/open?mode=binance&testnet=true&symbol=BTCUSDT&side=buy&type=market&quote\_qty=10
   * /exec/positions?...
   * /exec/close\_all?...
   * Проверить, что записи появились в order\_log, а daily\_limits.json обновился.
6. **Риск-проверка:**
   * Ужать лимиты в .env, убедиться в отказе «вторая сделка в день».
   * Имитировать просадку (через симулятор или принудительную запись realized PnL) → проверить блокировку по дневному стопу.
7. **Фоновая защита:**
   * Запустить периодический heartbeat.
   * Включить (по крону/фоновому таску) reconcile\_positions для сверки и журналирования расхождений (можно раз в 5–10 минут).
8. **Дальше по необходимости (не блокирует демо):**
   * Реализовать risk/trailing.py.
   * Расширить reconcile (агрегации, отчёты).
   * Доработать UI (витрины по журналу, дневные метрики).