需求文档

Aipha Vibe Trading System 需求文档（MVP）

0. 目标与范围

MVP 目标

构建最基础版本的 vibe trading system，支持以下闭环：

1. 用户输入自然语言策略意图
2. AI 自动生成结构化策略（五层：原子/时间/信号/逻辑/动作）
3. 自动回测并给出报告
4. 一键确认后进入 **Paper Trading**（优先）或 **Live Trading**（可开关）
5. 具备最小可用的可观测性（日志、状态、告警）

MVP 验收任务

> 在 QQQ / NDX 上确认 **4**小时 **MACD** 死叉，且在收盘前 **2** 分钟仍未收回"跌破的5日 MA"（仍在 MA5 下方），则卖出一部分 **TQQQ**，认为反弹结束。

核心点：多标的（信号标的 vs 交易标的）、多周期（4H + "收盘前2分钟" ）、事件+状态组合、执行动作（减仓 ）。

---

1. 用户体验与产品形态（MVP）

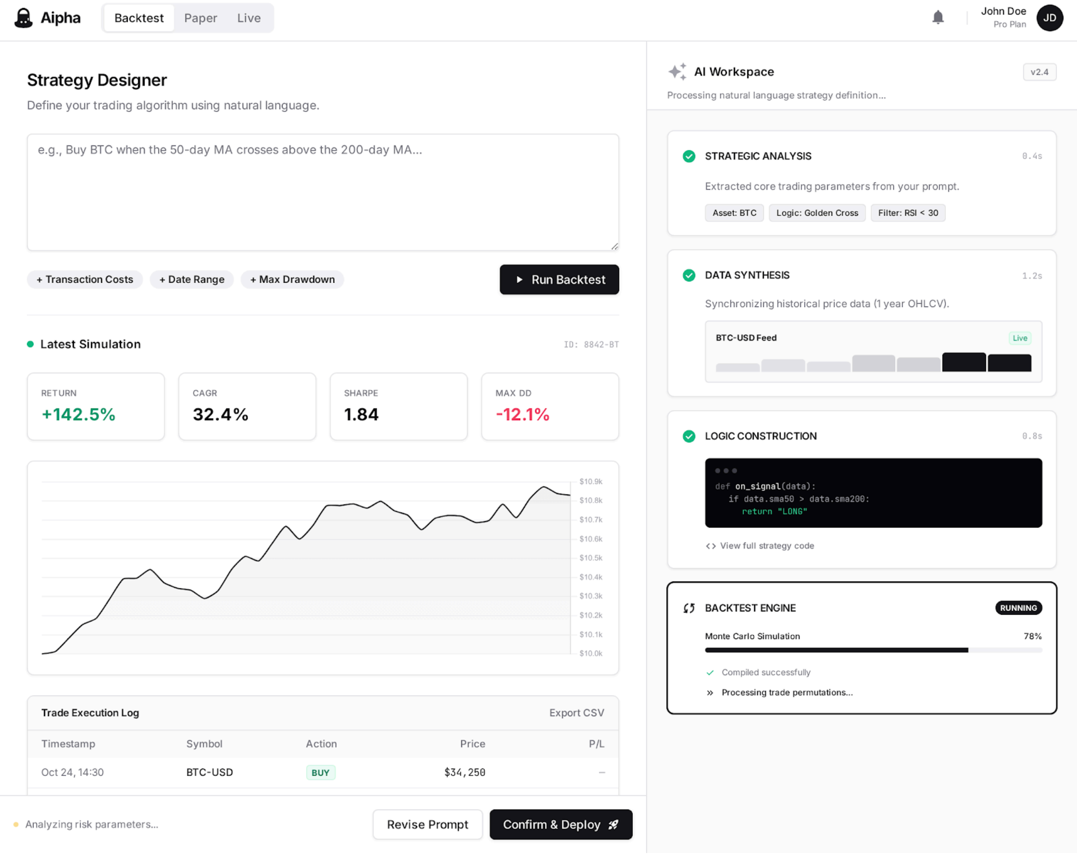
1.1 交互原则

- 用户只输入一句自然语言（或很少补充），系统自动完成拆解与执行。
- 输出结果后用户只做：**Confirm**（部署）**/ Revise**（修改）。

1.2 页面结构（**v0**）

- 左侧：对话输入 + 回测结果（KPI、曲线、交易列表）+ Confirm/Revise

- 右侧：AI 工作台（Planner/Runner 状态）展示进度与产物（DSL、数据检查、回测 run_id）



Strategy Designer

Define your trading algorithm using natural language.

e.g., Buy BTC when the 50-day MA crosses above the 200-day MA...

● Latest Simulation ID: 8842-BT

RETURN	CAGR	SHARPE	MAX DD
+142.5%	32.4%	1.84	-12.1%

Trade Execution Log Export CSV

Timestamp	Symbol	Action	Price	P/L
Oct 24, 14:30	BTC-USD	BUY	$34,250	–

● Analyzing risk parameters...

AI Workspace v2.4

Processing natural language strategy definition...

✓ STRATEGIC ANALYSIS 0.4s

Extracted core trading parameters from your prompt.

Asset: BTC | Logic: Golden Cross | Filter: RSI < 30

✓ DATA SYNTHESIS 1.2s

Synchronizing historical price data (1 year OHLCV).

BTC-USD Feed Live

✓ LOGIC CONSTRUCTION 0.8s

```
def on_signal(data):
  if data.sma50 > data.sma200:
    return "LONG"
```

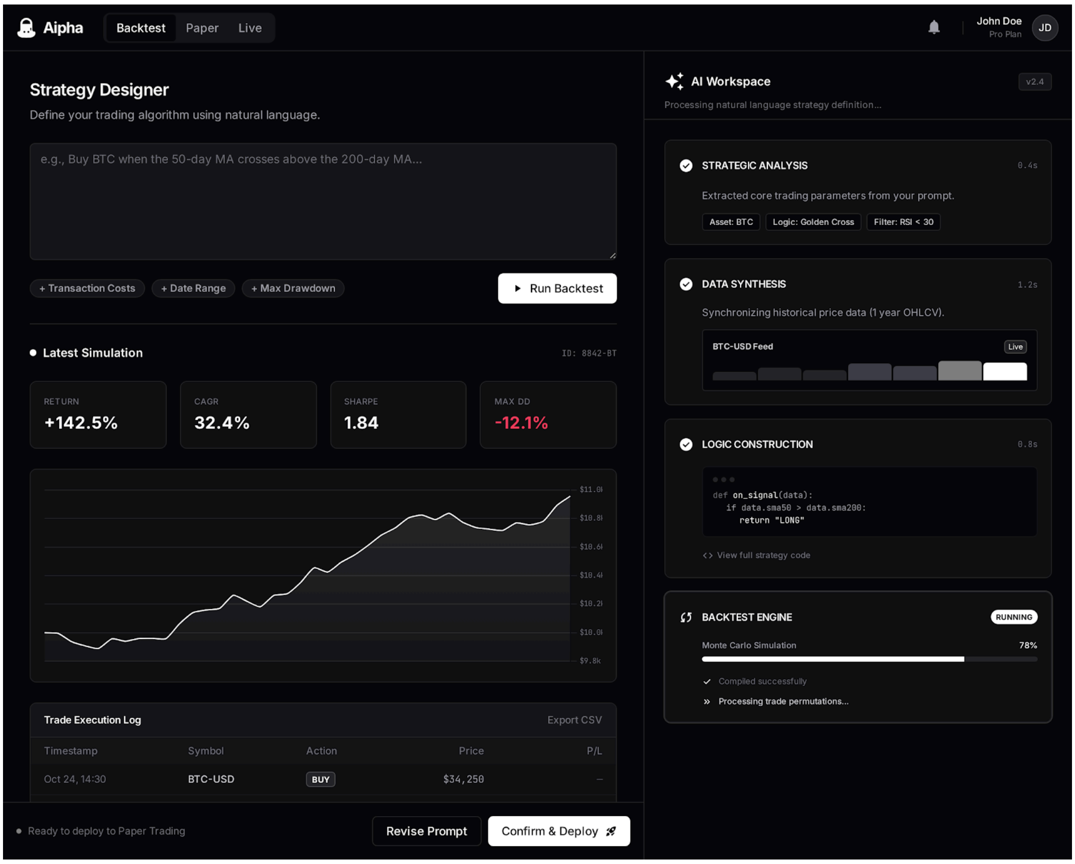
<> View full strategy code

BACKTEST ENGINE RUNNING

Monte Carlo Simulation 78%

✓ Compiled successfully

» Processing trade permutations...



Strategy Designer

Define your trading algorithm using natural language.

e.g., Buy BTC when the 50-day MA crosses above the 200-day MA...

● Latest Simulation ID: 8842-BT

RETURN	CAGR	SHARPE	MAX DD
+142.5%	32.4%	1.84	-12.1%

Trade Execution Log Export CSV

Timestamp	Symbol	Action	Price	P/L
Oct 24, 14:30	BTC-USD	BUY	$34,250	–

● Ready to deploy to Paper Trading

AI Workspace v2.4

Processing natural language strategy definition...

✓ STRATEGIC ANALYSIS 0.4s

Extracted core trading parameters from your prompt.

Asset: BTC | Logic: Golden Cross | Filter: RSI < 30

✓ DATA SYNTHESIS 1.2s

Synchronizing historical price data (1 year OHLCV).

BTC-USD Feed Live

✓ LOGIC CONSTRUCTION 0.8s

```
def on_signal(data):
  if data.sma50 > data.sma200:
    return "LONG"
```

<> View full strategy code

BACKTEST ENGINE RUNNING

Monte Carlo Simulation 78%

✓ Compiled successfully

» Processing trade permutations...

2. 策略表达：五层 DSL（必须）

2.1 五层定义

1. 原子层 **Atom**：指标/特征（MACD、MA）
2. 时间层 **Timeframe**：1m/1h/4h/1d + 对齐规则
3. 信号层 **Signal**：死叉、收盘前条件、MA 下方状态
4. 逻辑层 **Logic**：AND/OR、窗口、确认、冷却、优先级
5. 动作层 **Action**：卖出多少、订单类型、执行保护

2.2 测试用例的 DSL 拼装结果（明确交付）

下面是 v0 运行必须能生成/保存/回测的“结构化策略 Spec”（用自然语言解析后产出）：

标的与角色

- Signal Underlying：`QQQ`（或 `NDX`，两者择一/同时支持）
- Trade Instrument：`TQQQ`

原子层

- `MACD(12,26,9)` on 4H for QQQ/NDX
- `MA(5, type=SMA)` on 1D for QQQ/NDX（“5日MA”日频）
- `LastPrice` on 1m for QQQ/NDX（用于收盘前2分钟检查）

时间层

- 4H bars：用于 MACD（基于分钟数据聚合或直接用4H数据）
- 1D bars：用于 MA5
- 1m bars：用于临近收盘检查（最后2分钟）

- 对齐：在“收盘前2分钟”决策点，只能使用已完成的 4H bar（无未来函数）
    - `carry_forward_last_closed_4H`（把最近完成4H状态带到当前时点）

信号层

- `S1_event`: 4H MACD 死叉（QQQ/NDX）
    - event：`macd_line crosses below signal_line`（在 4H bar close 判定）
    - confirm：可选 `confirm_bars=0/1`（v0 可不确认）
- `S2_state`: 当天收盘前 2 分钟（例如 15:58:00 ET）时，QQQ/NDX 价格仍在 MA5 下方
    - state：`price_1m(15:58) < MA5_today`（MA5 取当日收盘前可得的“昨日为止 MA5” or “当日实时MA5”二选一，v0 建议用昨日收盘计算的 **MA5**，避免当日未收盘导致定义歧义）
- `S3_filter`（可选）：当日是否为交易日；数据完整性 OK

逻辑层

- `TRIGGER = S1_event_within(lookback_days=5 trading days) AND S2_state`
    - 解释：4H 死叉发生后的一段时间内（比如5个交易日）有效，避免“死叉发生了很久还触发”
    - cooldown：触发后 `cooldown=1 trading day`，避免每天15:58重复卖

动作层

- `ACTION = REDUCE_POSITION(symbol=TQQQ, by_pct=25%)`
    - 订单：`MKT` 或 `LMT`（v0 建议 MKT + 滑点保护阈值）
    - 执行保护：
        - max_slippage_bps（例如 30bps）
        - only_regular_session（只在正常交易时段）
        - idempotency_key（按日期+策略+symbol 防重复下单）

以上就是“拼搭结果”，你们的系统必须能自动生成并运行它。

---

3. 回测需求（MVP 必须有）

3.1 回测引擎必须支持

- 多标的：信号标的（QQQ/NDX）触发交易标的（TQQQ）
- 多周期：4H、1D、1m 的对齐与无未来函数
- 执行规则：在指定时刻（15:58）下单
- 成本模型：手续费 + 滑点（先固定bps，后续可升级）
- 输出：
  - KPI：年化/回撤/Sharpe/交易次数/胜率/平均持有期
  - 交易列表：每笔交易原因（S1/S2/逻辑触发）
  - 诊断：信号触发频率、触发后收益分布、失效环境提示

3.2 数据颗粒度与回测时间

- 最低要求：**1**分钟级数据（用于 15:58 检查与模拟下单）
- 4H 可由 1m 聚合
- 1D 用于 MA5（可由 1m 聚合或直接日线）
- 针对MOC订单，使用完整的从开盘到收盘的数据进行回测模拟，成交价格必须使用该交易日16:00的收盘价(Close Price)，而不是15:58的瞬时价

---

4. 实盘/纸面交易需求（MVP）

4.1 交易接口（建议路径）

- 优先：IBKR API（纸面/实盘一致）
- 需要的能力：
  - 查询持仓（TQQQ 当前仓位）

    - 下单（市价/限价）
    - 订单状态回报（成交/失败）
    - 幂等与重试（避免重复卖）

4.2 定时与触发

- 每个交易日 **15:58:00 ET**（美股东部时间）触发策略评估与下单
- 同时需要持续更新/缓存信号状态（4H MACD 是否在有效窗口内）

---

5. 工程架构（v0 最小模块）

5.1 服务划分

1. **UI Web**：对话、工作台、报告展示、Confirm/Revise
2. **Strategy Service**（**NL→Spec**）
    - 把自然语言解析成五层 DSL（带默认值）
    - 校验（字段完整性、无未来函数规则）
3. **Planner/Runner**
    - 把 Spec 编译成 DAG 任务（数据→特征→信号→回测→报告）
    - 记录 run_id、步骤状态、产物链接
4. **Market Data Service**
    - 拉取/缓存 1m 数据、聚合 4H/1D
    - 数据完整性检查（缺口、时区、交易日）
5. **Backtest Engine**
6. **Execution Service**
    - IBKR 下单、风控、幂等、告警

5.2 存储与产物

- DB（Postgres）：策略 spec、run_id、任务状态、交易记录、配置
- Object Storage：回测报告、图表、交易 CSV、日志快照
- Cache（Redis）：最新行情/信号状态、任务队列锁

5.3 可观测性

- structured logs（JSON）
- metrics：下单成功率、延迟、数据缺口率、回测耗时
- alerts：15:58 触发失败、数据缺失、下单失败、重复下单防护触发

---

6. 数据需求与选型

6.1 必要数据

- QQQ（或 NDX）1-minute OHLCV
- TQQQ 1-minute OHLCV（回测下单成交模拟）
- 交易日历（US equities calendar）
- 公司行为（拆股/分红）处理：v0 可只用调整后价格（adjusted）或明确只做不复权回测并声明

6.2 数据源策略（不绑定单一供应商）

- v0 推荐：选择一个稳定的分钟数据供应（付费通常更稳定）
- 备选：两路数据（主/备）+ 缺口自动切换（可 v1）

---

7. 服务器与部署要求（MVP）

7.1 技术设想（可修改）

- 足够跑原型+少量用户
- 前后端
  - Vite + React + Python + Fast API
  - 要求可以达到快速部署验证的要求
  - 框架完整，遵循clean code代码风格
  - 前端风格简练，采用Shadcn/UI作为前端组件和默认配色模板
- Infra
  - GitHub
  - Supabase支持PostgreSQL + Auth + Storage + Realtime
  - Redis 缓存
  - Vercel快速部署

7.2 网络与安全

- API 必须鉴权（JWT/Session）

- Broker 凭证加密存储（KMS/Secret Manager）

- 下单服务独立网络策略（最小权限）

- 审计日志：谁在什么时候 Confirm、生成了什么订单意图

---

8. 风控与防呆（**MVP** 必须 ）

- **No Lookahead**：跨周期只能使用已完成 bar

- 时间点明确：15:58 ET 执行；若遇到半日市/提前收盘，按交易日历调整

- 幂等：同一策略同一交易日最多触发一次卖出

- 失败降级：数据缺失/下单失败 → 不交易 + 告警 + UI 展示原因

- **Paper** 默认：Confirm 默认部署到 Paper；Live 需要额外开关

---

9. 里程碑与验收标准（建议 ）

Phase 0（**3–7** 天）：策略 **DSL + mock** 回测

- NL→Spec 能输出五层 DSL

- Runner 能跑通流程（先用 mock 数据/结果）

- UI 两区 + Confirm/Revise

Phase 1（1–2 周）：真实数据 + 回测可用

- 接入分钟数据
- 回测引擎跑出 KPI、交易列表、曲线
- 测试用例可以回测复现（15:58 检查 + 4H MACD）

Phase 2（1–2 周）：Paper Trading

- IBKR paper 下单、订单状态、幂等、告警
- 每个交易日自动触发一次评估

验收标准（针对你这个策略）

- 回测：能输出 “触发日/触发时刻/卖出比例/原因”
- 实盘（paper）：在触发日 15:58 ET 产生一笔减仓单，且不会重复下单

---

10. 工程任务清单

Backend

- Spec Schema（五层 DSL）定义 + 校验器
- NL→Spec 解析器（默认值、输出假设 ）
- Data Service：分钟数据拉取/缓存/聚合（1m→4H/1D）
- Backtest Engine：多标的+多周期对齐+成本模型
- Report Generator：KPI+图+交易列表+解释文本
- Execution Service：IBKR paper 下单 + 幂等 + 告警

Frontend

- Strategy Designer: chat 输入与历史
- 右侧 AI Workspace（步骤卡片 + artifact 链接）
- Backtest Summary 部分（KPI + chart + trade table）
- Confirm/Revise 流程（paper 默认 ）

Infra/DevOps

- 容器化、CI/CD、环境变量与密钥管理
- Postgres/Redis 部署
- 日志与告警（最小版也要有）

:

- **DSL 的 JSON Schema**（字段名、类型、默认值 ）
- Runner 的 DAG 节点定义（input/output）
- 回测引擎的对齐规则伪代码（避免未来函数）
- IBKR 下单幂等与重试策略（状态机）

后端技术总结

Vibe Trading 后端技术总结

0. Scope & Non-goals（v0 边界）

In Scope

- 自然语言 → 五层 DSL → 执行计划 → 回测 → 报告 → Paper（可选 Live）
- 多周期信号（1m / 4H / 1D）对齐
- 决策 vs 成交时点严格区分
- Workspace 可观测（steps / logs / artifacts）

Out of Scope（v0 不做）

- 高频（<1m）数据
- 盘中多次决策
- 复杂组合优化 / 资金再平衡
- Partial fill / VWAP / TWAP 等复杂执行

---

1. Global Assumptions（全局写死规则 ）

1. 时区与日历
   - 统一使用 `America/New_York`
   - 使用交易所日历（处理 DST / 提前收盘 / 假期）
   - 禁止使用 EST / 本地时间
2. 决策与执行
   - 决策时点：`market_close - 2 minutes`
     - 常规交易日：15:58 ET
     - 提前收盘：`actual_close - 2min`
   - 执行模型：MOC（Market-On-Close）
   - 回测成交价：当日收盘价（16:00 close 或提前收盘 close）+ 滑点/成本
3. MA5 定义（强约束）
   - MA5 = 截至昨日收盘的 5 日 SMA
   - 使用 `LAST_CLOSED_1D` bar
   - 禁止使用当日未收盘数据（避免未来函数）
4. 多周期数据真源
   - 1m 为唯一真源

    - 4H / 1D 必须由 1m 聚合生成

5. 信号标的降级
    - 默认信号标的：`QQQ`
    - `NDX` 分钟数据不稳定 → 自动 fallback 到 QQQ
    - 降级必须在 DataHealth 中显式标记

---

2. System Architecture（模块划分）

- `NL Input`
- `  ↓`
- `Parser / SpecBuilder`
- `  ↓ StrategySpec (DSL)`
- `Planner`
- `  ↓ ExecutionPlan`
- `Runner (Orchestrator)`
- `  ├─ Data Factory`
- `  ├─ Indicator Engine`
- `  ├─ Backtest Engine`
- `  ├─ Execution Service (Paper/Live)`
- `  ↓`
- `Workspace (steps / logs / artifacts)`

---

3. Core Domain Models & Schemas（核心结构）

3.1 StrategySpec（策略规范）

包含：

- universe（signal / trade symbols）
- decision / execution / risk
- DSL 五层：
    - Atomic
    - Time
    - Signal

    - Logic
    - Action

StrategySpec 是 策略的唯一语义源，Planner / Backtest / Execution 不得自行推断规则。

3.2 ExecutionPlan（执行计划，plan.json）

ExecutionPlan 是 StrategySpec 的可执行编译结果。

必须包含

```
{ "version": "v0", "decision_schedule": { "type": "MARKET_CLOSE_OFFSET",
"offset": "-2m", "timezone": "America/New_York" }, "nodes": [ { "id":
"data_signal_1m", "type": "DATA", "symbol": "QQQ", "timeframe": "1m",
"outputs": ["bars_qqq_1m"] } ] }
```

强制规则

- 每个 DATA / INDICATOR 节点必须显式声明：
    - `symbol`
    - `timeframe`
- 禁止 Runner / DataFactory 从 DSL 反推 symbol

---

4. Time & Multi-Timeframe Semantics（关键写死点）

4.1 4H Bar 切分规则（非常重要）

SESSION_ALIGNED_4H 定义：

- 以 交易所 session open（NYSE 09:30 ET）为锚点
- 按 session 内切分 4H bar
- 提前收盘日：最后一个 4H bar 自动缩短
- 禁止使用自然日 / UTC 对齐 / pandas resample 默认行为

4.2 多周期对齐规则（No Future Function）

周期	取值规则
1m	decision_time 前最后一个已闭合 bar
4H	decision_time 前最后一个 已闭合 4H bar
1D	昨日收盘（LAST_CLOSED_1D）

高周期值在决策点（15:58）使用 carry-forward 语义。

---

5. Signal & Event Semantics（信号语义）

5.1 Indicator

- 所有 indicator 只允许基于 已闭合 bar
- lookback/window 必须带单位（如 `"5d"` / `"20bars@4h"`）

5.2 Event（非常关键）

```
Event semantics: - Events are edge-triggered. - macd_bear_cross is TRUE only at the bar where the cross occurs. - It is NOT a persistent state.
```

MACD 死叉 ≠ “MACD 当前在 signal 下方”

---

6. Logic Layer（策略条件）

v0 示例逻辑（语义级）：

```
IF (4H MACD bearish cross occurred on last closed 4H bar) AND (15:58 1m close < MA5_last_closed_1D) THEN sell part of TQQQ position via MOC
```

---

7. Action & Execution Semantics

7.1 Quantity Resolution

```
"qty": { "mode": "FRACTION_OF_POSITION", "value": 0.3 }
```

最小成交约束（必须实现）

```
If computed quantity < broker.min_qty: - v0 behavior: skip action - log:
"qty_too_small" - do NOT place order
```

7.2 Idempotency & Cooldown

- Idempotency key：
    - `strategy_version + trading_day + action_id`
-
- Cooldown（v0 默认）：
    - 同一 symbol + side
    - 1 trading day 内只允许一次

---

8. Backtest Engine Semantics

8.1 时间线

阶段	时间
Decision	15:58
Order Submit	15:58
Fill	Market Close

8.2 Trade 结构（必须区分）

```
{ "decision_time": "...15:58", "fill_time": "...16:00", "fill_price": 50.12,
"cost": { "slippage": 0.02, "commission": 0.0 }, "why": { "macd_4h_cross":
true, "close_1558": 408.1, "ma5_last_closed": 410.25, "signal_symbol": "QQQ",
"is_fallback": false } }
```

---

9. Data Factory & Health

9.1 数据真源

- 所有 4H / 1D 数据必须由 1m 聚合
- 禁止混用第三方 4H / 1D

9.2 DataHealth（必须产出）

```
{ "source": "primary | fallback", "is_fallback": false, "missing_ratio": 0.0,
"gaps": [ { "start": "...", "end": "...", "bars_missing": 3 } ] }
```

---

10. Workspace & Observability（前端可观测 ）

10.1 Workspace Steps（固定枚举 ）

- `parse → plan → data → backtest → report → deploy`

10.2 必备 Artifacts

Artifact	用途
dsl.json	策略语义快照
plan.json	执行计划
inputs_snapshot.json	最终解析输入（强烈要求）
report.md	回测解释性报告
equity.png	净值曲线

inputs_snapshot.json 内容：

- strategy_version
- resolved universe
- resolved calendar
- execution model
- fallback 是否发生

---

11. Error Model（统一）

```
{ "code": "VALIDATION_ERROR | DATA_UNAVAILABLE | EXECUTION_GUARD_BLOCKED |
INTERNAL", "message": "string", "details": {} }
```

---

12. v0 必跑测试用例（验收标准）

用例：

在 QQQ 确认 4H MACD 死叉，且 15:58 收盘价仍低于“昨日收盘计算的 MA5”，

则于当日收盘通过 MOC 卖出部分 TQQQ。

验收点：

- decision_time = 15:58
- fill_time = market close
- MA5 使用 LAST_CLOSED_1D
- 4H MACD 使用 last closed 4H bar
- Trade.why 可完整解释

---

13. Final Statement（定稿声明）

本文档定义了 Vibe Trading System v0 的唯一后端语义标准。

所有实现必须遵循本文档，不允许自行合理推断或扩展。

工程 Implementation Checklist

工程 Implementation Checklist

0. 全局约束（必须先确认）

- 系统时区固定：`America/New_York`
- 使用 交易所日历（支持 DST / 提前收盘）
- 禁止使用 EST / 本地时间 / UTC shortcut
- 所有 lookback / window 必须带单位（禁止裸数字）

---

1. NL → StrategySpec（Parser / SpecBuilder）

输入

- 支持 `NaturalLanguageStrategyRequest`
- 支持 `mode = BACKTEST_ONLY | PAPER | LIVE`
- 支持 overrides（universe / execution / risk）

输出：**StrategySpec**

- 生成 `strategy_id`
- 生成 确定性的 `strategy_version`（内容 hash）
- universe 明确区分：
  - `signal_symbol`
  - `trade_symbol`
- decision / execution / risk 字段齐全
- DSL 五层结构完整：
  - atomic
  - time
  - signal
  - logic
  - action

强制校验（失败即 **VALIDATION_ERROR**）

- MA5 = `LAST_CLOSED_1D`（禁止当日 MA）
- 所有 lookback 带单位（如 `"5d"` / `"20bars@4h"`）
- timezone 只能是 `America/New_York`

---

2. Planner → ExecutionPlan

ExecutionPlan 结构

- 生成 `decision_schedule`
  - type = `MARKET_CLOSE_OFFSET`
  - offset = `-2m`
  - timezone = `America/New_York`

Node 编译（重点）

对每个 node：

- 有 `id`
- 有 `type`（DATA / INDICATOR / LOGIC / ACTION）
- 显式声明 `symbol` + `timeframe`
- 不允许 Runner / DataFactory 从 DSL 反推 symbol

❗如果这里漏了 symbol / tf，直接算不合格

---

3. Data Factory（数据层 ）

数据真源

- 1m 是唯一真源
- 4H / 1D 只能由 1m 聚合
- 禁止混用第三方 4H / 1D

4H 聚合规则（必须一致）

- 使用 `SESSION_ALIGNED_4H`
- 锚点 = 交易所开盘（NYSE 09:30 ET）
- 提前收盘日：最后一个 4H bar 自动缩短
- 禁止 UTC / 自然日 resample

NDX Fallback

- 若 NDX 分钟数据不可用 → fallback 到 QQQ
- 在 DataHealth 中显式标记：
  - `is_fallback = true`
  - `source = fallback`

DataHealth

- 输出 `missing_ratio`
- 输出 gaps（即便 v0 先为空数组 ）

---

4. Indicator Engine（信号层 ）

通用规则

- 所有指标 只使用已闭合 bar
- 禁止未来函数

MA5

- tf = `1d`
- window = `"5d"`
- bar_selection = `LAST_CLOSED_1D`
- align = `CARRY_FORWARD`

4H MACD

- 使用 最近已闭合 4H bar
- 值在 15:58 决策点 carry-forward

Event 语义（非常重要）

- MACD 死叉是 edge-triggered event
- 只在发生的那个 4H bar 为 true
- 不是持续状态

---

5. Logic Engine（策略逻辑 ）

- 支持 AND / OR / ALL / ANY
- 逻辑只消费：
    - indicator 值
    - event
- 不允许在 Logic 层直接访问原始数据

---

6. Action → ExecutionIntent

Quantity 解析

- 支持 `FRACTION_OF_POSITION`
- qty 在 execution-time 计算

最小成交约束（必须 ）

- 若 qty < broker.min_qty：
    - 不下单
    - 记录 log：`qty_too_small`
    - Action 视为 skipped（不是 failed）

幂等 & 冷却

- idempotency_key =

```
strategy_version + trading_day + action_id
```

-
- cooldown = 1 trading day
- cooldown 命中 → `EXECUTION_GUARD_BLOCKED`

---

7. Backtest Engine（核心验收点）

时间线（必须严格）

- decision_time = market_close - 2m
- order_submit_time = decision_time
- fill_time = market_close

成交价

- 使用当日收盘价（或提前收盘 close）
- 应用 slippage + commission

Trade 结构（必须齐 ）

- decision_time
- fill_time
- fill_price
- cost（slippage / commission）
- why：
  - macd_4h_cross
  - close_15_58
  - ma5_last_closed
  - signal_symbol
  - is_fallback

---

8. Execution Service（Paper / Live）

Paper

- 复用 Backtest fill 逻辑
- 订单 / 成交状态完整

Live（若启用）

- 仅支持 MOC
- 支持 reject / cancel
- 与 Backtest 使用同一 idempotency 逻辑

---

9. Workspace / Observability（前端强依赖）

Workspace Steps（固定顺序）

- parse
- plan
- data
- backtest
- report
- deploy

Step 状态

- PENDING / RUNNING / DONE / FAILED / SKIPPED
- 每 step 支持 logs[]

必备 Artifacts

- `dsl.json`
- `plan.json`
- `inputs_snapshot.json`（必须）
- `report.md`
- `equity.png`

---

10. Error Model（统一）

- `VALIDATION_ERROR`
- `DATA_UNAVAILABLE`
- `EXECUTION_GUARD_BLOCKED`
- `INTERNAL`

所有 API 错误必须返回 `{ code, message, details }`

---

11. v0 验收测试（必须跑通）

策略：

QQQ 出现 4H MACD 死叉，且 15:58 close < 昨日收盘 MA5 →

MOC 卖出部分 TQQQ

验收点

- decision_time = 15:58
- fill_time = 收盘
- MA5 使用 LAST_CLOSED_1D
- 4H MACD 使用 last closed 4H bar
- Trade.why 可完整解释

✅ 完成定义（**Definition of Done**）

- 所有 checklist 项均可勾选
- 不存在“工程自行理解”的行为
- 回测结果可解释、可回放
- 前端 Workspace 无需 hardcode 语义

Schema 分层要求

Schema 分层要求

1. 总览分层

层级	Schema 名	角色	对前端/外部是否是 Contract	备注
API	`NaturalLanguageStrategyRequest`	POST /runs 请求体	✅ 是	前端直接构造
API	`RunStatusResponse`（含 `RunStatus` + `Workspace`）	GET /runs/{id}/status	✅ 是	Workspace 驱动 UI
API	`BacktestReportResponse`（含 `BacktestReport`）	GET /runs/{id}/report	✅ 是	报告页主数据
API	`DeploymentRequest`	POST /runs/{id}/deploy 请求体	✅ 是	部署入口
API	`Deployment` / `DeploymentResponse`	POST /runs/{id}/deploy 响应体	✅ 是	展示部署状态
API	`ErrorObject`	所有 4xx/5xx 错误	✅ 是	统一错误模型
API-共享	`Workspace` / `WorkspaceStep` / `ArtifactRef` / `LogEntry`	嵌入在 RunStatus 中	✅ 是	前端按此渲染 steps/logs/artifacts
API-共享	`BacktestMetrics`	BacktestReport 内部	✅ 是	前端图表 / KPI

API-共享	`Trade`	BacktestReport 内部	✅ 是	交易明细表

---

2. Internal Schema（对内使用，不视为前端 Contract）

模块	Schema 名	Contract 归属	说明
Strategy	`StrategySpec`	❌ Internal	Parser 输出，后端内部标准，不承诺前端直接使用
Strategy / DSL	`AtomicLayer` / `TimeLayer` / `SignalLayer` / `LogicLayer` / `ActionLayer`	❌ Internal	作为 `StrategySpec.dsl` 的细分 schema，前端只看 dsl.json artifact，不做编译耦合
Planner	`ExecutionPlan`	❌ Internal	plan.json 是 artifact，可变结构；只对 Runner/Worker 是 contract
Data	`DataRequest`	❌ Internal	DataFactory 内部协议
Data	`Bar` / `BarSeries` / `DataHealth`	❌ Internal	仅内部使用，外部只看到聚合结果（在 report / why 里）
Indicator	`IndicatorJob` / `IndicatorSeries` / `EventSeries`	❌ Internal	指标计算层内部结构
Backtest	`BacktestJob`	❌ Internal	Runner → BacktestEngine 内部结构
Execution	`ExecutionIntent`	❌ Internal	Logic → ExecutionService 内部结构

Execution	`Order` / `Fill`	❌ Internal	仅 paper/live 执行层使用；前端只通过 `Trade` & 状态 artifact 间接看到
Infra	`Run`（内部实体）	❌ Internal	DB 模型；对外只暴露 RunStatus/Report/Deploy 三套 API 结构

3. Contract 稳定性说明

- 强稳定（尽量不改）
  - 所有 API 层 schema：
    - `NaturalLanguageStrategyRequest`
    - `RunStatusResponse`（含 Workspace 结构）
    - `BacktestReportResponse`（含 Trades / Metrics）
    - `DeploymentRequest` / `DeploymentResponse`
    - `ErrorObject`
  - 改动需要：
    - 版本协商（v0 → v1）
    - 或向后兼容（仅增加可选字段）
- 中等稳定（后端内部模块间的 contract）
  - `StrategySpec`
  - `ExecutionPlan`
  - `DataRequest` / `BarSeries` / `DataHealth`
  - `ExecutionIntent`
  - 这些可以随 v0.x 优化，但要：
    - 同步给所有 Worker / Service owner
    - 确保 artifact（比如 `dsl.json` / `plan.json`）旧版本仍可读取或有 migrate
- 低稳定（实现细节 ）
  - `IndicatorJob` / `IndicatorSeries` / `EventSeries`
  - `BacktestJob`
  - `Order` / `Fill`
  - 可以随着性能/实现优化调整，只要对上层 `Trade`、`Report` 的语义不变即可。

Schema Json示例

Schema Json示例

1. `strategy_spec.schema.json`

（StrategySpec + DSL 五层，Internal Schema）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"strategy_spec.schema.json", "title": "StrategySpec", "type": "object",
"required": [ "strategy_id", "strategy_version", "name", "timezone",
"calendar", "universe", "decision", "execution", "risk", "dsl", "meta" ],
"properties": { "strategy_id": { "type": "string" }, "strategy_version": {
"type": "string" }, "name": { "type": "string" }, "timezone": { "type":
"string", "const": "America/New_York" }, "calendar": { "type": "object",
"required": ["type", "value"], "properties": { "type": { "type": "string",
"const": "exchange" }, "value": { "type": "string", "const": "XNYS" } } },
"universe": { "type": "object", "required": ["signal_symbol",
"trade_symbol"], "properties": { "signal_symbol": { "type": "string" },
"signal_symbol_fallbacks": { "type": "array", "items": { "type": "string" }
}, "trade_symbol": { "type": "string" } } }, "decision": { "type": "object",
"required": ["decision_time_rule"], "properties": { "decision_time_rule": {
"type": "object", "required": ["type", "offset"], "properties": { "type": {
"type": "string", "const": "MARKET_CLOSE_OFFSET" }, "offset": { "type":
"string", "pattern": "^-?[0-9]+m$", "const": "-2m" } } } } }, "execution": {
"type": "object", "required": ["model"], "properties": { "model": { "type":
"string", "enum": ["MOC"] }, "slippage_bps": { "type": "number", "default": 0
}, "commission_per_share": { "type": "number", "default": 0 },
"commission_per_trade": { "type": "number", "default": 0 } } }, "risk": {
"type": "object", "properties": { "cooldown": { "type": "object", "required":
["scope", "value"], "properties": { "scope": { "type": "string", "enum":
["SYMBOL_ACTION"] }, "value": { "type": "string" } } }, "max_orders_per_day":
{ "type": "integer", "default": 1 } } }, "dsl": { "type": "object",
"required": ["atomic", "time", "signal", "logic", "action"], "properties": {
"atomic": { "$ref": "#/definitions/AtomicLayer" }, "time": { "$ref":
"#/definitions/TimeLayer" }, "signal": { "$ref": "#/definitions/SignalLayer"
}, "logic": { "$ref": "#/definitions/LogicLayer" }, "action": { "$ref":
"#/definitions/ActionLayer" } } }, "meta": { "type": "object", "properties":
{ "created_at": { "type": "string", "format": "date-time" }, "author": {
"type": "string" }, "notes": { "type": "string" } } } }, "definitions": {
"Duration": { "oneOf": [ { "type": "string", "pattern": "^[0-9]+(d|h|m|s)$"
```

```
}, { "type": "string", "pattern": "^[0-9]+bars@(1m|5m|15m|30m|1h|4h|1d)$" },
{ "type": "object", "required": ["tf", "bars"], "properties": { "tf": {
"type": "string", "enum": ["1m", "5m", "15m", "30m", "1h", "4h", "1d"] },
"bars": { "type": "integer", "minimum": 1 } } } ] }, "Timeframe": { "type":
"string", "enum": ["1m", "5m", "15m", "30m", "1h", "4h", "1d"] },
"AlignRule": { "type": "string", "enum": ["LAST_CLOSED_BAR", "CARRY_FORWARD"]
}, "AtomicLayer": { "type": "object", "properties": { "symbols": { "type":
"array", "items": { "type": "object", "required": ["name", "ticker"],
"properties": { "name": { "type": "string" }, "ticker": { "type": "string" }
} } }, "constants": { "type": "object", "properties": { "sell_fraction": {
"type": "number", "minimum": 0, "maximum": 1, "default": 0.3 } },
"additionalProperties": true } } }, "TimeLayer": { "type": "object",
"required": ["primary_tf", "derived_tfs", "session", "aggregation"],
"properties": { "primary_tf": { "$ref": "#/definitions/Timeframe" },
"derived_tfs": { "type": "array", "items": { "$ref":
"#/definitions/Timeframe" } }, "session": { "type": "object", "required":
["calendar", "timezone"], "properties": { "calendar": { "type": "string",
"const": "XNYS" }, "timezone": { "type": "string", "const":
"America/New_York" } } }, "aggregation": { "type": "object", "properties": {
"4h": { "type": "object", "required": ["source_tf", "bar_close_rule",
"align"], "properties": { "source_tf": { "type": "string", "const": "1m" },
"bar_close_rule": { "type": "string", "const": "SESSION_ALIGNED_4H" },
"align": { "$ref": "#/definitions/AlignRule" } } }, "1d": { "type": "object",
"required": ["source_tf", "bar_close_rule", "align"], "properties": {
"source_tf": { "type": "string", "const": "1m" }, "bar_close_rule": { "type":
"string", "const": "EXCHANGE_DAILY" }, "align": { "$ref":
"#/definitions/AlignRule" } } } }, "additionalProperties": true } } },
"SignalIndicator": { "type": "object", "required": ["id", "symbol_ref", "tf",
"type", "params", "align"], "properties": { "id": { "type": "string" },
"symbol_ref": { "type": "string" }, "tf": { "$ref": "#/definitions/Timeframe"
}, "type": { "type": "string" }, "params": { "type": "object" }, "align": {
"$ref": "#/definitions/AlignRule" } } }, "SignalEvent": { "type": "object",
"required": ["id", "type"], "properties": { "id": { "type": "string" },
"type": { "type": "string", "enum": ["CROSS", "THRESHOLD"] }, "left": {
"type": "object" }, "right": { "type": "object" }, "direction": { "type":
"string", "enum": ["UP", "DOWN", "ANY"] }, "scope": { "type": "string",
"enum": ["LAST_CLOSED_4H_BAR", "LAST_CLOSED_1D", "BAR"] } } }, "SignalLayer":
{ "type": "object", "properties": { "indicators": { "type": "array", "items":
```

```
{ "$ref": "#/definitions/SignalIndicator" } }, "events": { "type": "array",
"items": { "$ref": "#/definitions/SignalEvent" } } } }, "LogicCondition": {
"type": "object", "properties": { "all": { "type": "array", "items": {
"$ref": "#/definitions/LogicCondition" } }, "any": { "type": "array",
"items": { "$ref": "#/definitions/LogicCondition" } }, "not": { "$ref":
"#/definitions/LogicCondition" }, "event_id": { "type": "string" }, "scope":
{ "type": "string" }, "op": { "type": "string", "enum": ["<", "<=", ">",
">=", "==", "!="] }, "left": { "type": "object" }, "right": { "type":
"object" } }, "additionalProperties": false }, "LogicRule": { "type":
"object", "required": ["id", "when", "then"], "properties": { "id": { "type":
"string" }, "when": { "$ref": "#/definitions/LogicCondition" }, "then": {
"type": "array", "items": { "type": "object", "required": ["action_id"],
"properties": { "action_id": { "type": "string" } } } } } }, "LogicLayer": {
"type": "object", "properties": { "rules": { "type": "array", "items": {
"$ref": "#/definitions/LogicRule" } } } }, "Action": { "type": "object",
"required": [ "id", "type", "symbol_ref", "side", "qty", "order_type" ],
"properties": { "id": { "type": "string" }, "type": { "type": "string",
"enum": ["ORDER"] }, "symbol_ref": { "type": "string" }, "side": { "type":
"string", "enum": ["BUY", "SELL"] }, "qty": { "type": "object", "required":
["mode"], "properties": { "mode": { "type": "string", "enum": [
"FRACTION_OF_POSITION", "ABSOLUTE", "NOTIONAL_USD" ] }, "value": { "type":
"number" }, "value_ref": { "type": "string" } } }, "order_type": { "type":
"string", "enum": ["MOC"] }, "time_in_force": { "type": "string", "enum":
["DAY"], "default": "DAY" }, "cooldown": { "$ref": "#/definitions/Duration"
}, "idempotency_scope": { "type": "string", "enum": ["DECISION_DAY"] } } },
"ActionLayer": { "type": "object", "properties": { "actions": { "type":
"array", "items": { "$ref": "#/definitions/Action" } } } } } }
```

---

2. run_api.schema.json

（/runs + /runs/{id}/status，API Contract）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"run_api.schema.json", "title": "Run API Schemas", "type": "object",
"definitions": { "NaturalLanguageStrategyRequest": { "type": "object",
"required": ["input_type", "nl", "mode"], "properties": { "input_type": {
"type": "string", "enum": ["NATURAL_LANGUAGE"] }, "nl": { "type": "string" },
"mode": { "type": "string", "enum": ["BACKTEST_ONLY", "PAPER", "LIVE"] },
"as_of": { "type": "string", "format": "date-time" }, "overrides": { "type":
"object", "properties": { "universe": { "type": "object", "properties": {
"signal_symbol": { "type": "string" }, "trade_symbol": { "type": "string" } }
}, "execution": { "type": "object", "properties": { "model": { "type":
"string", "enum": ["MOC"] }, "slippage_bps": { "type": "number" } } },
"risk": { "type": "object", "properties": { "cooldown": { "type": "string" }
} } }, "additionalProperties": true } }, "additionalProperties": false },
"ErrorObject": { "type": "object", "required": ["code", "message"],
"properties": { "code": { "type": "string", "enum": [ "VALIDATION_ERROR",
"DATA_UNAVAILABLE", "EXECUTION_GUARD_BLOCKED", "INTERNAL" ] }, "message": {
"type": "string" }, "details": { "type": "object" } } }, "LogEntry": {
"type": "object", "required": ["ts", "level", "msg"], "properties": { "ts": {
"type": "string", "format": "date-time" }, "level": { "type": "string",
"enum": ["DEBUG", "INFO", "WARN", "ERROR"] }, "msg": { "type": "string" },
"kv": { "type": "object" } } }, "ArtifactRef": { "type": "object",
"required": ["id", "type", "name", "uri"], "properties": { "id": { "type":
"string" }, "type": { "type": "string", "enum": ["json", "markdown", "image",
"csv", "binary"] }, "name": { "type": "string" }, "uri": { "type": "string" }
} }, "WorkspaceStep": { "type": "object", "required": ["id", "state",
"label"], "properties": { "id": { "type": "string" }, "state": { "type":
"string", "enum": ["PENDING", "RUNNING", "DONE", "FAILED", "SKIPPED"] },
"label": { "type": "string" }, "progress": { "type": "number", "minimum": 0,
"maximum": 1 }, "started_at": { "type": "string", "format": "date-time" },
"ended_at": { "type": "string", "format": "date-time" }, "logs": { "type":
"array", "items": { "$ref": "#/definitions/LogEntry" } } } }, "Workspace": {
"type": "object", "properties": { "steps": { "type": "array", "items": {
"$ref": "#/definitions/WorkspaceStep" } }, "artifacts": { "type": "array",
"items": { "$ref": "#/definitions/ArtifactRef" } } } }, "RunStatusResponse":
{ "type": "object", "required": ["run_id", "status", "updated_at",
"workspace"], "properties": { "run_id": { "type": "string" }, "status": {
"type": "string", "enum": ["CREATED", "RUNNING", "SUCCEEDED", "FAILED",
"CANCELED"] }, "updated_at": { "type": "string", "format": "date-time" },
"workspace": { "$ref": "#/definitions/Workspace" }, "error": { "$ref":
"#/definitions/ErrorObject" } } }, "RunCreateResponse": { "type": "object",
"required": ["run_id", "status", "workspace"], "properties": { "run_id": {
"type": "string" }, "status": { "type": "string", "enum": ["CREATED",
"RUNNING", "FAILED"] }, "strategy_id": { "type": "string" },
"strategy_version": { "type": "string" }, "workspace": { "$ref":
"#/definitions/Workspace" } } } } }
```

---

3. `backtest_report.schema.json`

（/runs/{id}/report，API Contract）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"backtest_report.schema.json", "title": "Backtest Report", "type": "object",
"definitions": { "BacktestMetrics": { "type": "object", "properties": {
"cagr": { "type": "number" }, "max_drawdown": { "type": "number" }, "sharpe":
{ "type": "number" }, "trades": { "type": "integer" }, "decision_days": {
"type": "integer" }, "trade_days": { "type": "integer" } } }, "Trade": {
"type": "object", "required": [ "decision_time", "fill_time", "symbol",
"side", "qty", "fill_price" ], "properties": { "decision_time": { "type":
"string", "format": "date-time" }, "fill_time": { "type": "string", "format":
"date-time" }, "symbol": { "type": "string" }, "side": { "type": "string",
"enum": ["BUY", "SELL"] }, "qty": { "type": "number" }, "fill_price": {
"type": "number" }, "cost": { "type": "object", "properties": { "slippage": {
"type": "number" }, "commission": { "type": "number" } } }, "why": { "type":
"object" } } }, "ArtifactRef": { "type": "object", "required": ["id", "type",
"name", "uri"], "properties": { "id": { "type": "string" }, "type": { "type":
"string", "enum": ["json", "markdown", "image", "csv", "binary"] }, "name": {
"type": "string" }, "uri": { "type": "string" } } },
"BacktestReportResponse": { "type": "object", "required": ["run_id",
"summary", "metrics", "trades"], "properties": { "run_id": { "type": "string"
}, "summary": { "type": "object", "properties": { "strategy_name": { "type":
"string" }, "mode": { "type": "string" }, "symbols": { "type": "object",
"properties": { "signal": { "type": "string" }, "trade": { "type": "string" }
} }, "decision_time_rule": { "type": "string" }, "execution_model": { "type":
"string", "enum": ["MOC"] } } }, "metrics": { "$ref":
"#/definitions/BacktestMetrics" }, "trades": { "type": "array", "items": {
"$ref": "#/definitions/Trade" } }, "artifacts": { "type": "array", "items": {
"$ref": "#/definitions/ArtifactRef" } } } } } }
```

---

4. `deployment_api.schema.json`

（/runs/{id}/deploy，API Contract）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"deployment_api.schema.json", "title": "Deployment API", "type": "object",
"definitions": { "DeploymentRequest": { "type": "object", "required":
["target"], "properties": { "target": { "type": "string", "enum": ["PAPER",
"LIVE"] }, "effective_from": { "type": "string", "format": "date" },
"broker": { "type": "object", "properties": { "name": { "type": "string" } }
}, "guards": { "type": "object", "properties": {
"require_backtest_succeeded": { "type": "boolean" }, "max_daily_orders": {
"type": "integer" } } } }, "additionalProperties": false },
"DeploymentResponse": { "type": "object", "required": ["deployment_id",
"run_id", "status", "scheduler"], "properties": { "deployment_id": { "type":
"string" }, "run_id": { "type": "string" }, "status": { "type": "string",
"enum": ["DEPLOYED", "DISABLED"] }, "scheduler": { "type": "object",
"properties": { "decision_time_rule": { "type": "string" }, "timezone": {
"type": "string" } } } } } } }
```

---

5. `data_internal.schema.json`

（DataFactory 内部，Internal Schema）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"data_internal.schema.json", "title": "Data Internal", "type": "object",
"definitions": { "DataRequest": { "type": "object", "required": ["symbol",
"timeframe", "start", "end"], "properties": { "symbol": { "type": "string" },
"timeframe": { "type": "string", "enum": ["1m", "5m", "15m", "30m", "1h",
"4h", "1d"] }, "start": { "type": "string", "format": "date-time" }, "end": {
"type": "string", "format": "date-time" }, "calendar": { "type": "string" } }
}, "Bar": { "type": "object", "required": ["start", "end", "open", "high",
"low", "close"], "properties": { "start": { "type": "string", "format":
"date-time" }, "end": { "type": "string", "format": "date-time" }, "open": {
"type": "number" }, "high": { "type": "number" }, "low": { "type": "number"
}, "close": { "type": "number" }, "volume": { "type": "number" } } },
"DataGap": { "type": "object", "properties": { "start": { "type": "string",
"format": "date-time" }, "end": { "type": "string", "format": "date-time" },
"bars_missing": { "type": "integer" } } }, "DataHealth": { "type": "object",
"properties": { "source": { "type": "string" }, "is_fallback": { "type":
"boolean", "default": false }, "missing_ratio": { "type": "number" }, "gaps":
{ "type": "array", "items": { "$ref": "#/definitions/DataGap" } } } },
"BarSeries": { "type": "object", "required": ["symbol", "timeframe", "bars"],
"properties": { "symbol": { "type": "string" }, "timeframe": { "type":
"string" }, "bars": { "type": "array", "items": { "$ref": "#/definitions/Bar"
} }, "health": { "$ref": "#/definitions/DataHealth" } } } } }
```

---

6. `plan_internal.schema.json`

（ExecutionPlan，Internal Schema）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"plan_internal.schema.json", "title": "ExecutionPlan", "type": "object",
"definitions": { "PlanNode": { "type": "object", "required": ["id", "type"],
"properties": { "id": { "type": "string" }, "type": { "type": "string",
"enum": ["DATA", "INDICATOR", "LOGIC", "ACTION"] }, "symbol": { "type":
"string" }, "timeframe": { "type": "string" }, "inputs": { "type": "array",
"items": { "type": "string" } }, "outputs": { "type": "array", "items": {
"type": "string" } }, "config": { "type": "object" } } }, "ExecutionPlan": {
"type": "object", "required": ["version", "decision_schedule", "nodes"],
"properties": { "version": { "type": "string" }, "decision_schedule": {
"type": "object", "required": ["type", "offset", "timezone"], "properties": {
"type": { "type": "string", "enum": ["MARKET_CLOSE_OFFSET"] }, "offset": {
"type": "string", "pattern": "^-?[0-9]+m$" }, "timezone": { "type": "string"
} } }, "nodes": { "type": "array", "items": { "$ref":
"#/definitions/PlanNode" } } } } } }
```

---

7. `execution_internal.schema.json`

（ExecutionIntent / Order / Fill，Internal Schema）

```
{ "$schema": "https://json-schema.org/draft/2020-12/schema", "$id":
"execution_internal.schema.json", "title": "Execution Internal", "type":
"object", "definitions": { "ExecutionIntent": { "type": "object", "required":
[ "symbol", "side", "qty", "order_type", "time_in_force", "decision_time",
"idempotency_key" ], "properties": { "symbol": { "type": "string" }, "side":
{ "type": "string", "enum": ["BUY", "SELL"] }, "qty": { "type": "number" },
"order_type": { "type": "string", "enum": ["MOC"] }, "time_in_force": {
"type": "string", "enum": ["DAY"] }, "decision_time": { "type": "string",
"format": "date-time" }, "idempotency_key": { "type": "string" }, "metadata":
{ "type": "object" } } }, "Order": { "type": "object", "required": [
"order_id", "symbol", "side", "qty", "order_type", "status" ], "properties":
{ "order_id": { "type": "string" }, "symbol": { "type": "string" }, "side": {
"type": "string", "enum": ["BUY", "SELL"] }, "qty": { "type": "number" },
"order_type": { "type": "string", "enum": ["MOC"] }, "status": { "type":
"string", "enum": ["NEW", "SUBMITTED", "FILLED", "REJECTED", "CANCELED"] } }
}, "Fill": { "type": "object", "required": ["order_id", "fill_time",
"fill_price", "qty"], "properties": { "order_id": { "type": "string" },
"fill_time": { "type": "string", "format": "date-time" }, "fill_price": {
"type": "number" }, "qty": { "type": "number" } } } } }
```

代码-Prompt示例

代码示例

1. NL Strategy Parser

```
NL API 层 (/runs)
  ↓
ParserService（你定义的接口）
  ├─ LlmClient（对接 OpenAI / 内部模型）
  ├─ OutputValidator（用 strategy_spec.schema.json 校验 ）
  └─ DefaultFiller（把所有默认值写进 StrategySpec）
```

注意点：

1. `LlmClient`：只负责 prompt + 调模型 + 拿回 raw JSON/string。
2. `ParserService`：
    - 负责重试、限流、缓存（同一 NL 文本不重复烧钱 ）。
    - 出口只有两种结果：
        - 成功 → 完整 `StrategySpec`
        - 失败 → 标准 `VALIDATION_ERROR` 或 `INTERNAL`。
3. 其他模块（Planner / Runner / Backtest）不认识 LLM，只认识 StrategySpec。

1.1 模块结构

```
ParserService
├── LlmClient
│   └── call(prompt) -> raw_text
├── PromptBuilder
│   └── build(nl_text, context) -> prompt
├── JsonExtractor
│   └── extract(raw_text) -> dict
├── SchemaValidator
│   └── validate(dict, strategy_spec.schema.json)
├── DefaultResolver
│   └── fill_defaults(dict) -> StrategySpec
└── Cache
    └── get/set(hash(nl_text))
```

1.2 核心接口定义

1.2.1. ParserService 接口

```
class ParserService: def parse_nl( self, user_id: str, request:
NaturalLanguageStrategyRequest ) -> StrategySpec: ...
```

保证：

- 返回的一定是 schema-valid + fully-resolved 的 StrategySpec
- 或直接抛结构化错误

1.2.2 LlmClient（最小能力）

```
class LlmClient: def call(self, prompt: str, timeout_s: int = 20) -> str: """
- 只负责调用模型 - 不关心策略语义 - 不做 JSON 校验 """
```

实现里可以是 OpenAI / Azure / 内部模型，Parser 不关心。

1.3 主流程伪代码（重点）

```
def parse_nl(self, user_id, request): # 0. 缓存（省钱 & 稳定） cache_key =
hash(user_id + request.nl) cached = cache.get(cache_key) if cached: return
cached # 1. 构建 Prompt prompt = PromptBuilder.build( nl_text=request.nl,
context={ "timezone": "America/New_York", "execution_model": "MOC", "rules":
[ "MA5 must be LAST_CLOSED_1D", "decision_time = market_close - 2m",
"lookback must include units" ] } ) # 2. 调用 LLM try: raw_text =
llm_client.call(prompt) except TimeoutError: raise
InternalError("LLM_TIMEOUT") # 3. 提取 JSON try: raw_dict =
JsonExtractor.extract(raw_text) except Exception: raise
ValidationError("INVALID_LLM_OUTPUT") # 4. Schema 校验（强） errors =
SchemaValidator.validate( raw_dict, schema="strategy_spec.schema.json" ) if
errors: raise ValidationError( message="StrategySpec schema validation
failed", details=errors ) # 5. 填默认值（非常关键） spec =
DefaultResolver.fill_defaults(raw_dict) # 6. 强制写死规则再校验一遍
self._enforce_hard_rules(spec) # 7. 写 meta spec.meta["parser_version"] =
"nl_v1" spec.meta["llm_model"] = "gpt-4.1" spec.meta["source"] = "nl" # 8. 缓
存 cache.set(cache_key, spec) return spec
```

1.4 Hard Rules（二次校验，不能只信 LLM）

```
def _enforce_hard_rules(self, spec: StrategySpec): assert spec.timezone ==
"America/New_York" # MA5 for ind in spec.dsl.signal.indicators: if ind.type
== "SMA" and ind.params.get("window") == "5d": assert
ind.params.get("bar_selection") == "LAST_CLOSED_1D" # decision time assert
spec.decision.decision_time_rule.type == "MARKET_CLOSE_OFFSET" assert
spec.decision.decision_time_rule.offset == "-2m" # lookback 单位 for ind in
spec.dsl.signal.indicators: assert lookback_has_unit(ind.params)
```

1.5 Prompt模板

要求这里的输入输出按照规定json格式
主要目的就是从用户的自然语言中抓取关键词，对策略进行填充

1.5.1. System Prompt（一次性配置）

```
You are a trading strategy compiler. Your job: - Convert natural language
trading strategy descriptions into a STRICT JSON object called StrategySpec.
- StrategySpec MUST be fully specified and valid according to the provided
rules. - You are NOT allowed to leave fields ambiguous or "to be defined
later". VERY IMPORTANT: - You MUST obey all "Hard Rules" below, even if the
user's description is ambiguous or contradicts them. - If the user
description conflicts with a Hard Rule, you MUST follow the Hard Rule and
still produce a consistent StrategySpec. Hard Rules (Vibe Trading v0): 1)
Timezone is always "America/New_York". 2) Exchange calendar is always "XNYS"
(US equities). 3) Decision time is always "market_close - 2 minutes": -
normal day: 15:58 ET - early close: close_time - 2 minutes 4) Execution model
is always "MOC" (Market-On-Close): - Orders decided at decision_time. - Fills
occur at the official market close price for the day. 5) MA5 definition is
FIXED: - MA5 is based on LAST_CLOSED 1D bars (end of yesterday's session). -
Never use today's partially formed 1D bar for MA5. 6) Timeframes: - Primary
timeframe: 1m. - 4h and 1d bars MUST be aggregated from 1m data. - 4h bars
are aligned to the trading session (SESSION_ALIGNED_4H), starting from
session open. 7) Multi-timeframe alignment: - For decision at 15:58: * 1m
values use the last closed 1m bar at or before 15:58. * 4h indicators use the
last CLOSED 4h bar (carry-forward semantics). * 1d indicators use
LAST_CLOSED_1D (yesterday’s close). 8) Lookback windows MUST always include
units, such as "5d", "20bars@4h", or { "tf": "4h", "bars": 5 }. - Bare
integers without units are NOT allowed. 9) Signals MUST NOT use future
information: - You can only use bars that are fully closed as of the
decision_time. Universe & Symbols: - There is always at least one "signal"
symbol and one "trade" symbol. - If the user does not explicitly specify,
assume: - signal_symbol = "QQQ" - trade_symbol = "TQQQ" - You may add
additional internal symbol references ("signal", "trade") but the underlying
tickers must be clear. Events: - CROSS events are edge-triggered: - A MACD
bearish cross (macd_bear_cross) is TRUE only at the bar where MACD crosses
below its signal. - It is NOT a persistent boolean state. Your output: - MUST
be valid JSON. - MUST match the StrategySpec skeleton described below. - MUST
be syntactically correct (parsable JSON). - MUST not contain comments or
trailing commas. StrategySpec skeleton (high-level): { "strategy_id":
```

```
"string", "strategy_version": "string (you may leave an empty string)",
"name": "string", "timezone": "America/New_York", "calendar": { "type":
"exchange", "value": "XNYS" }, "universe": { "signal_symbol": "QQQ",
"signal_symbol_fallbacks": ["NDX", "QQQ"], "trade_symbol": "TQQQ" },
"decision": { "decision_time_rule": { "type": "MARKET_CLOSE_OFFSET",
"offset": "-2m" } }, "execution": { "model": "MOC", "slippage_bps": 2,
"commission_per_share": 0.0, "commission_per_trade": 0.0 }, "risk": {
"cooldown": { "scope": "SYMBOL_ACTION", "value": "1d" },
"max_orders_per_day": 1 }, "dsl": { "atomic": { ... }, "time": { ... },
"signal": { ... }, "logic": { ... }, "action": { ... } }, "meta": {
"created_at": "2026-01-01T00:00:00Z", "author": "nl_user", "notes": "" } }
Atomic layer: - Define symbol references and constants (e.g. sell_fraction).
Time layer: - primary_tf: "1m" - derived_tfs: ["4h", "1d"] - aggregation for
4h and 1d as described in the Hard Rules. Signal layer: - indicators[]: each
with {id, symbol_ref, tf, type, params, align} - events[]: for cross/down
logic (e.g. "macd_bear_cross"). Logic layer: - rules[]: each with {id, when,
then[]} - when: boolean expression built from events and indicator
comparisons. Action layer: - actions[]: each with {id, type, symbol_ref,
side, qty, order_type, ...} - For partial sells, use: { "id":
"sell_trade_symbol_partial", "type": "ORDER", "symbol_ref": "trade", "side":
"SELL", "qty": { "mode": "FRACTION_OF_POSITION", "value": 0.3 },
"order_type": "MOC", "time_in_force": "DAY", "cooldown": "1d",
"idempotency_scope": "DECISION_DAY" } IMPORTANT: - When the user’s
description is vague, you MUST choose reasonable, consistent defaults that
respect the Hard Rules. - Never ask follow-up questions; always return a
complete StrategySpec. Output instructions: - Output ONLY the JSON object. -
DO NOT include any explanation, commentary, markdown, or backticks. - If you
need to approximate something, choose a clear, concrete value.
```

1.5.2 User Prompt 模板（在代码里填充的那一层）

假设你在后端拿到 `request.nl`（用户自然语言描述），以及可能的 `user_id`、模式等，可以这样构造 user message：

```
User natural language strategy description: "{nl_text}" Additional context: -
mode: "{mode}" # e.g. BACKTEST_ONLY / PAPER / LIVE - user_id: "{user_id}" #
do NOT include PII, use internal id only Task: 1) Read the strategy
description above. 2) Infer a concrete, executable StrategySpec that follows
all Hard Rules. 3) Fill in all required fields, including: - universe
```

```
(signal_symbol, trade_symbol) - decision rules - execution model (MOC) - risk
(cooldown, max_orders_per_day) - DSL layers: atomic, time, signal, logic,
action 4) Make sure the DSL implements the described behavior as closely as
possible. For this specific product (Vibe Trading v0), always: - Use "QQQ" as
signal symbol and "TQQQ" as trade symbol if the user does not specify. - Use
1m data as the primary timeframe, and aggregate 4h and 1d from 1m. - For MA5,
always use LAST_CLOSED_1D (yesterday's close) and window "5d". - For
MACD(4h), compute on 4h bars and use the last CLOSED 4h bar at decision time.
- Define at least one logic rule that connects the signals to the actions.
Example user intent (for your reference of style): - "When 4H MACD on QQQ
turns bearish and the 15:58 close is still below the 5-day moving average
(based on yesterday's close), sell part of my TQQQ position into the close."
IMPORTANT: - Return ONLY the JSON of StrategySpec. - Do not wrap it in quotes
or Markdown. - The JSON MUST be syntactically valid.
```

2. 回测引擎关键（多周期对齐 / MOC / 成本 / 幂等 / cooldown）

2.1. 多周期对齐：4H carry-forward 到 15:58

```
def get_decision_timestamp(trading_day, calendar): close_ts =
calendar.market_close(trading_day) # handles early close + DST return
close_ts - timedelta(minutes=2) # 15:58 or early_close-2m def
last_closed_bar(bars_tf, ts): # bars_tf: list of bars with [start, end, ohlc]
# return the bar whose end <= ts and is the latest return max([b for b in
bars_tf if b.end <= ts], key=lambda b: b.end) def
compute_signals_at(decision_ts): # 1m close at 15:58 bar_1m =
last_closed_bar(bars_1m, decision_ts) close_1m = bar_1m.close # 4h MACD uses
LAST_CLOSED_4H bar, then carry-forward bar_4h = last_closed_bar(bars_4h,
decision_ts) macd_val, macd_sig = macd_series_4h.value_at(bar_4h.end),
macd_series_4h.signal_at(bar_4h.end) macd_cross_down =
crossed_down(macd_series_4h, macd_signal_4h, at=bar_4h.end) # MA5 uses
LAST_CLOSED_1D (yesterday) bar_1d_last_closed = last_closed_bar(bars_1d,
decision_ts).previous_trading_day_bar ma5 = sma(bars_1d_close, window=5,
end_at=bar_1d_last_closed.end) return { "close_1m": close_1m,
"macd_cross_down": macd_cross_down, "ma5_last_closed": ma5 }
```

注：`previous_trading_day_bar` 必须走日历（周末/假期跳过），保证“截至昨日收盘”。

2.2. 15:58 决策 + MOC 成交（fill 用 close）

```
def simulate_day(trading_day): decision_ts =
get_decision_timestamp(trading_day, calendar) signals =
compute_signals_at(decision_ts) decision = (signals["macd_cross_down"] and
(signals["close_1m"] < signals["ma5_last_closed"])) if decision: order =
build_order(symbol="TQQQ", side="SELL", qty=fraction_of_position(0.3),
order_type="MOC") # fill at market close fill_ts =
calendar.market_close(trading_day) fill_price = bars_1m_close_at("TQQQ",
fill_ts) # or daily close if easier fill_price = apply_slippage(fill_price,
side="SELL", bps=slippage_bps) cost = commission_model(order)
execute_fill(order, fill_ts, fill_price, cost, decision_ts, why=signals)
```

2.3. 成本/滑点模型（v0 简化）

```
def apply_slippage(price, side, bps): # SELL gets worse: price * (1 -
bps/10000) mult = (1 - bps/10000) if side == "SELL" else (1 + bps/10000)
return price * mult
```

2.4. 幂等 + cooldown（避免重复卖 ）

```
def idempotency_key(strategy_version, trading_day, action_id): return
f"{strategy_version}:{trading_day}:{action_id}" def can_fire(action_id,
trading_day): # cooldown "1d" => if already fired today, block return not
store.exists(idempotency_key(strategy_version, trading_day, action_id)) def
mark_fired(action_id, trading_day):
store.put(idempotency_key(strategy_version, trading_day, action_id), True) #
in runner: if decision and can_fire("sell_trade_symbol_partial",
trading_day): place_order(...) mark_fired(...)
```