01_Project Plan

(Can be overridden, with comprehensive explanation why)

1. Goals

- Primary outcome (backtests): reliable, reproducible backtests with realistic P&L
- Primary outcome (live): Small, and very safe live deployment that executes exactly
- Secondary: For the MVP, great a solid (safe and reliable), scalable architecture. Scalability means the ability to add of additional asset classes, venues, backtest modeling tools for backtest, strategies, risk, sizing and portfolio functionality later.
- Important: Prevention of Survivorship bias, lookahead bias and overfitting

2. Scope and Constraints

2.1. Business & use case scope:

- **MVP Operating Modes (one line):** “Backtest/Research, Paper Trading, Live Trading (GTC only, market/limit).”
- User model: solo
- Strategy count: single, with option to add additional\
- Manual intervention level: fully automated
- Deliverables: notebooks

2.2. Markets & Instruments

- Asset classes: crypto (spot, options, futures), later equities etc.
- Venues: Binance (MVP), other later

2.3. Data Model

- Data: Flexible, OHLCV first
- Granularity: 1 min smallest for MVP
- History length: venue availability
- Live data sources: REST first, WEBSOCKET or FIX only added in later version
- Persistence: disk
- Time & TZ: Canonical timezone, UTC
- Storage: raw (HDD) + parquet (SDD)

2.4. Strategy

- Style: intraday trading mostly (trend, mean reversion, option modelling)
- Signal inputs: mostly price/volume
- Modelling: rules, ML

- Rebalance: even driven
- Transaction costs model: all, i.e., commissions, fees, spreads, slippage (borrow/funding, rebates not for MVP)

2.5. Risk& Sizing & Portfolio

- **Capital base & base currency: USDT
- **Leverage & margin:** (out of MVP scope)
- **Shorting:** (Out of MVP scope)
- **Sizing:** fixed for MVP (volatility-targeting, Kelly-capped, risk parity later)
- **Limits:** MVP: per-order notional (per-symbol exposure, gross/net, VaR/DD guards)
- **Kill-switches:** YES! Simple for MVP

2.6. Order Execution

- All order types
- Time in Force: GTC, IOC
- Partial fills:YES
- Routing/algos: single venue
- Latency budget: <1s
- Fill/impact models: see below

2.7. Accounting & P&L

- Valuation: mark-to-last bar close
- Cost basis: FIFO, realized & unrealized PnL
- Cash flows: Not relevant for MVP (crypto)
- Multi-currency: Not relevant for MVP (crypto)
- Corporate actions: Not relevant for MVP (crypto)

2.8. Architecture & performance

- Programming: Python and C++ (later for performance)
- Engine design: event-driven (MVP), vectorized later version
- Determinism: RNG seeding, single source of truth for state
- Replay & recovery: Checkpointing, durable queues (append-only logs + snapshots), idempotent actions
- Config: env-based configs, schema evolution

2.9. Broker & External Integration

- Exchange: binance for now
- Rate limits: Binance specific
- Connectivity: REST MVP, FIX/Websockets later
- Reconcilation: MVP: Periodicit full state pulls, drift detection late

2.9. Reliability, Observability & Ops

- Monitoring: metrics and logs
- Audit trail!
- Disaster control: global cancel, position flatten

2.10. Testing & validation

- Unit & property tests!
- Backtest - paper - live parity tests

2.11. Compliance

- Record-keeping: trade logs, timestamping

2.12. Constraints:

- Budget: no money for MVP
- Hardware: MacBookPro M4 Pro, 24 gb ram, potentially server later
- Data: <100gb for MVP (depending on strategy)
- MVP: Absolut minimal version to backtest and deploy simple strategies.
- Simple MVP, simplicity over complexity; more complex features (above bare minimum) to be added later

3. SLO

Service: Personal Trader (live)
SLOs (monthly):

- Availability ≥ 99.5%
- Missed/duplicate bars = 0
- Safety triggers: 100% enforce; time-to-halt ≤ 1 sec
  Error budget: 3.6h/mo
  Policy: Disable new orders if budget < 50%; switch to paper if < 25%

4 Domain Model

4.1. Ubiquitous language

Instrument / Symbol (e.g., `BTCUSDT`), **Bar(OHLCV, 1m)**, **Tick** (later), **Order** (market/limit, GTC), **ExecutionReport / Fill**, **Trade** (execution normalized), **Position** (qty, avg price, PnL), **Portfolio** (USDT base), **Signal** (decision inputs/outputs), **RiskLimit** (per-order notional, daily loss halt), **Snapshot** (durable state), **Halt / Kill-switch**. Long-only, spot, USDT base currency; multi-currency, margin, shorting out for MVP.

4.2. Bounded contexts

- Market data: Ingests 1-minute OHLCV, normalizes calendars/time (UTC internally), persists raw→parquet, detects gaps/dupes. Interfaces: `BarFeed`, `HistoryStore`.
- Strategy & Research: Deterministic feature calc → signal → order intent. Interfaces: `Strategy.compute_signals(bars, state) -> [OrderIntent]`.
- Backtets Simulator: Event-driven bar loop, models fees + simple slippage (fixed bps), order lifecycle, positions, P&L (FIFO), audit logs. Parity with live semantics is the goal.
- Execution (Paper/Live): Broker adapters (Paper, Binance REST) with **idempotent** client order IDs, order state machine, reconcilers, global cancel/flatten. Partial fills are out-of-scope for MVP (treat as atomic).
- **Risk & Sizing**: Fixed sizing now; checks: per-order notional cap, daily drawdown halt, exposure caps; hard **kill-switch**. Kelly/vol targeting later.
- **Accounting & P&L**: Positions, realized/unrealized P&L (FIFO), mark-to-close valuation. CSV/Parquet exports.
- **Orchestration & Ops**: Runtime, config, checkpoints, deterministic RNG seeds, metrics/logging, disaster controls, replay/recovery. Latency budget <1s, availability targets & error budget.

4.3. Events & commands (engine heartbeat)

- **Events:** `BarReady`, `SignalComputed`, `OrderIntentCreated`, `OrderSubmitted`, `OrderAcked`, `OrderFilled`, `PositionChanged`, `PnLUpdated`, `RiskBreach`, `TradingHalted`, `SnapshotSaved`.
- **Commands:** `SubmitOrder`, `CancelOrder`, `FlattenAll`, `ResumeTrading`, `SaveSnapshot`, `ReconcileState`.

4.4. Key Aggregates:

- Order: New - Stubmitted - Acked - Filled (or canceled, rejected), no partial for MVP
- Invariant: At most one live order per symbol / strategy unless explicitly allowed (GTC only for MVP)
- Position: long only for MVP, avg price recomputed on fills, PnL = Mark-to-close, FIFO realization
- Portfolio: USDT Base currency, exposure ≤ configured caps; daily loss halt flips engine to halted until manual resume.

4.5. Ports & adapters

- Ports

```
DataFeed.get_history(symbol, start, end, granularity) -> DataFrame
DataFeed.stream_bars(symbols, granularity) -> iterator[Bar]

Broker.submit(order: Order) -> ClientOrderId
Broker.cancel(client_order_id) -> None
Broker.get_order(client_order_id) -> ExecutionReport
Broker.get_positions() -> list[Position]
Broker.flatten_all() -> None
```

```
Storage.put(table, records) / get(table, query)
Clock.now() -> UTC timestamp
```

- Adapters
  `ParquetHistoryStore`, `RESTPollingFeed(Binance)`, `PaperBroker`, `BinanceRESTBroker`, `CSVLogSink`. (WebSocket/FIX later.)

4.6. Architecture shape (C4 in words)

- System context: “Personal Trader” interacts with **Binance REST API** (orders, account, market data) and local disk (raw→Parquet). CLI/daemon controlled by you.
- **Containers:**
  - **Trader Core (Python)**: event bus, app services (BacktestRunner, LiveRunner), Risk, Accounting.
  - **Broker Adapter**: Paper + Binance REST.
  - **Market Data Service**: Ingest + gap detection + store.
  - **Storage**: Parquet/CSV on disk; snapshots/checkpoints.
  - **CLI**: `bt run`, `live paper`, `live real`; emits metrics/logs. (not in MVP)
- **Hot path sequence (live):** `BarReady → Strategy → OrderIntent → RiskCheck → Broker.submit → Ack/Fill → Position/PnL → Logs + Snapshot`. Keep within p95 < 1s budget (mostly broker RTT).

4.7. Slicing Plan

- (Breaking down large project into thin, end-to-end functional pieces).
- See Slices Plan 04_Slices

4.8. Contract tests

- (Definition: A way to ensure that two separate services (API, provider) and client (consumer) can communicate with each other correctly)
- DataFeed contract:
  - `get_history` returns UTC-indexed 1-min bars, no gaps for a known period; schema: time, open, high, low, close, volume, ntrades
  - `stream_bars` emits strictly increasing timestamps; tolerates network hiccups with backoff.
- **Broker contract**
  - `submit` is idempotent (same clientOrderId → same order).
  - `get_order` transitions only along allowed edges; no PARTIAL in MVP.
  - `flatten_all` results in zero positions and emits audit logs.
- **Storage contract**
  - `put/get` round-trips schema without precision loss (prices/qty).
  - Durability: crash after `put` → data present after restart.

4.9. Data model (minimal schemas)

- **bars.parquet**: `ts_utc, symbol, open, high, low, close, volume, source, ntrades`
- **orders.csv**: `ts_utc, strategy, symbol, side, type, qty, limit_price, client_order_id, state`
- **trades.csv**: `ts_utc, client_order_id, symbol, price, qty, fee, fee_ccy`
- **positions.csv**: `ts_utc, symbol, qty, avg_price, unrealized_pnl`
- **equity.csv**: `ts_utc, equity, drawdown`
- (All timestamps canonical/UTC; Europe/Berlin only for display)

4.10. Guardrails tied to your MVP

- **Scope locks:** Crypto spot, Binance, 1–3 symbols, 1-min bars, REST polling, GTC only, no partial fills, fixed-bps slippage, FIFO P&L, disk persistence (raw + parquet), event-driven engine, audit trail, kill-switch, latency p95 <1s, “paper → tiny live” path.

5 ADRs

ADR-001: Modular Monolith over Microservices

Context / Options:
A) Microservices from day 1; B) Modular monolith with clear boundaries (ports/adapters).
Decision: B. Single repository and process, with explicit interfaces: `DataFeed`, `Broker`, `Storage`, `Clock`, `Strategy`.
Consequences:

- Lowest ops overhead; easy debugging; deterministic.
  − Harder to scale across machines (not needed for personal use).
  Revisit when: You truly need >1 venue concurrently, or CPU/IO contention appears and profiling shows isolation is needed.

ADR-002: Event-Driven Core Loop (In-Process Bus)

Context / Options:
A) Synchronous “for each bar do everything”; B) In-proc event bus with typed events; C) External queue (Kafka/Rabbit).
Decision: B. Use an in-process event bus emitting `BarReady → SignalComputed → OrderIntent → ...`.
Consequences:

- Testable, decoupled modules; natural replay; timing hooks.
  − Slight complexity over a linear loop.
  Revisit when: Cross-process scaling or multiple strategies need independent pacing.

ADR-003: Deterministic, Single-Threaded Backtests

Context / Options:
A) Multithread parallelism; B) Single-thread determinism; C) Multiprocess per-strategy.

Decision: B. Deterministic single-thread for a given strategy/run; allow separate processes for parallel strategies if needed.
Consequences:

- Bitwise reproducibility; simpler accounting.
  − Throughput capped per process (acceptable for 1-min bars).
  Revisit when: You need faster than ~25k bars/sec per run on your laptop.

ADR-004: REST Polling for Live Bars (WebSocket Later)

Context / Options:
A) WebSocket streaming now; B) REST polling now; C) Both.
Decision: B. Start with REST polling (easier error handling & retries). Keep an interface to swap to WS later.
Consequences:

- Simpler; fewer moving parts; easier offline tests.
  − Higher latency and risk of missed intra-minute micro-moves.
  Revisit when: You move to sub-minute bars or need lower latency and better gap behavior → add WS adapter.

ADR-005: One Venue, One Asset Class (Spot) for V1

Context / Options:
A) Multi-venue abstraction; B) Single venue; C) Single venue + paper.
Decision: C. Build parity with **PaperBroker** first; then one real **BrokerAdapter** (same contracts).
Consequences:

- Faster to reliable parity; less API variance.
  − No immediate multi-venue/arbitrage.
  Revisit when: Strategy needs routing or you want to A/B execution across venues.

ADR-006: Storage = Parquet for History, Append-Only Logs + Snapshots for State

Context / Options:
A) SQLite/Timeseries DB; B) Parquet for bars + JSONL/CSV logs + snapshots; C) DuckDB for everything.
Decision: B.

- Historical bars: columnar **Parquet** (compressed, portable).
- Operational durability: **append-only log** (orders/trades/decisions) flushed fsync; **periodic snapshots** (positions/equity).

- Crash-replay on startup.

Consequences:

- Zero external deps; easy diff/inspect; good perf.
  − No concurrent writers; manual care with fsync.
  Revisit when: You need ad-hoc SQL over large histories → consider DuckDB/SQLite in addition.

ADR-007: UTC Everywhere Internally

Context / Options:
A) Local TZ; B) UTC internal + UI TZ conversion.
Decision: B. All timestamps and calendars in UTC; convert only at the edges (CLI/UI) to Europe/Berlin.
Consequences:

- No DST bugs; clean replay across time boundaries.
  − Must be disciplined about conversion in user output.
  Revisit when: Never (strong default).

ADR-008: Execution Model Without Partial Fills (V1)

Context / Options:
A) Model partial fills & amendments; B) Atomic orders (all-or-none from engine perspective) with GTC/IOC as supported.
Decision: B. Treat fills as atomic for both sim and paper/live parity; if exchange returns partials, adapter aggregates and presents a single final “FILLED” or “CANCELED”.
Consequences:

- Sim↔live parity, simpler state machine, fewer edge cases.
  − Loses realism for thin books; may hide partial execution dynamics.
  Revisit when: You trade thin pairs or larger size → add true partial-fill lifecycle.

ADR-009: Idempotent Client Order IDs + Reconciliation

Context / Options:
A) Opaque broker IDs; B) Deterministic client IDs + periodic reconciliation.
Decision: B. Generate stable `client_order_id` (e.g., `STRAT-SYM-TS-NONCE`); broker adapter retries are idempotent; a reconciler runs on startup and periodically.
Consequences:

- No duplicate orders on retries; safe crash recovery.
  − Must carefully scope ID namespace to avoid collisions.
  Revisit when: Multiple processes/strategies submit to same account → include process/strategy UUID in IDs.

ADR-010: Simple Slippage = Spread + Participation-of-Volume

Context / Options:
A) No slippage; B) Fixed bps; C) Spread + POV; D) OB-based microstructure.
Decision: C. Use mid+½spread plus an impact term proportional to your **participation of bar volume**.
Consequences:

- Better than fixed-bps; cheap to compute; deterministic.
  − Still an approximation; not suitable for HFT.
  Revisit when: You capture L2 or trade sub-minute bars → consider OB snapshots.

ADR-011: Risk Rails in Pre-Trade Path + Global Kill-Switch

Context / Options:
A) Post-trade monitoring only; B) Pre-trade hard checks + circuit breaker.
Decision: B. Enforce notional caps, daily loss halt, and exposure limits **before** submit. Global `halt/flatten` commands available.
Consequences:

- Prevents bad orders; limits blast radius.
  − Requires careful ordering to not block legitimate exits (exits bypass some checks).
  Revisit when: You add leverage/shorts or multiple symbols → expand to portfolio-aware checks.

ADR-012: Config & Secrets

Context / Options:
A) All in code; B) YAML/TOML + env overrides; C) Key-value store.
Decision: B. Typed config file (committed) + environment variable overrides; secrets only via env/OS keychain; never in repo.
Consequences:

- Repeatable runs; easy diff; safe secrets handling.
  − Two places to look (file + env) when debugging.
  Revisit when: You share configs across machines → add `.env` loader or OS keychain integration.

ADR-013: Observability: Logs + Minimal Metrics, No External Stack

Context / Options:
A) Full Prometheus/Grafana; B) Minimal counters/timers + structured logs.
Decision: B. Emit structured logs (JSONL) and a few counters/timers: `order_rtt`, `bars_lag`, `error_rate`, `pnl_intraday`. Three alerts: bar gap, order error rate, daily loss halt.
Consequences:

- Lightweight; fits local-only use; good enough to debug.
  − No long-term dashboards unless you add them.
  Revisit when: You regularly run unattended → add a tiny local Grafana/Prom stack.
