Software Factory fürs Trading – Architektur & Plan

Mega-Projekt „Software Factory fürs Trading“ — Iteration 1 & 2 (Kompakt-Dokument)

Ziele & Leitplanken
- Durchgängige Pipeline: Experiment → Lab/Backtests → Paper → Live.
- Einheitliches Datenmodell: OHLCV (Open, High, Low, Close, Volume) konsequent GROSS, Index in UTC.
- Anzeige/Bedienung in Europe/Berlin; interne Berechnungen und Datenhaltung in UTC.
- Reproduzierbarkeit: feste Seeds, Artefakt-Manifeste, versionierte Runs (Code & Daten).
- Sichere Order-Strecke: OCO, TIF/Session-Gültigkeit, Idempotenz-Key, MOC-Guard.
- Beobachtbarkeit: strukturierte Logs, Metriken, Dashboards.
- Erweiterbarkeit: Strategie-Plugins, Market-Data-Adapter, Broker-Adapter.

End-to-End-Architektur (gemappt auf bestehende Bausteine)
Experiment (lokal)
  - fetch D1/M1 (EODHD Paketfiles) → assemble/resample → signals (InsideBar)
  - axiom-bt: cli_data + engines/replay + artifacts/*
Lab / Backtests (lokal)
  - runner (axiom_bt.runner) → simulate (replay_engine) → metrics, curves
  - Streamlit-Dashboard: Runs browsen, KPIs, Equity, Trades, Orders
Promotion bei Greenlight → Paper Trading (Debian-Server)
  - Realtime-Data (WebSocket, z. B. Polygon oder IBKR) → Filters/Signals → OCO Orders via ib_insync (Paper)
  - systemd Worker, Healthchecks, strukturierte Logs
Promotion bei Stabilität → Live
  - identische Pipeline, härtere Guards: Limits/Exposure/KillSwitch, Audit

Datenquellen & Adapter
- Historisch: EODHD (D1/M1 Paket-Parquets) für Backtests; normalisierte Spalten OHLCV, Index UTC.
- Realtime: IBKR (ib_insync) oder Polygon (bei euch installiert) — möglichst nur EINE Quelle in Paper/Live für Parität.
- Handelskalender: pandas-market-calendars (NYSE/NASDAQ) für Sessions/Holidays; keine Localtime im Index.

Broker-Anbindung
- IBKR via ib_insync: OCO/Bracket, Order-Status, Paper/Live kompatibel. Idempotente Referenzen für sichere Retries.

Artefakte & Speicher
- Parquet für History (M1, M5, D1), artifacts/ strikt versioniert; manifests/metrics pro Run.
- Optional: Data-Versionierung mit DVC oder lakeFS, falls Governance/Audit auf Datenebene benötigt.

Modul-Schnittstellen (konkret)
1) Data Layer
- History Loader (EODHD): liest Paket-Parquets → normalisiert Spaltennamen (OHLCV groß), Index UTC.
- Assembler: assemble-m1 (merge/trim/validate), ensure-m5 (resample M1→M5, Lücken-Policy, Session-Cut).
- Kalender/TZ: Session-Filter (NYSE/NASDAQ). Intern UTC, UI in Europe/Berlin.

- Learnings: Fehlerquellen waren fehlende/ungültige M5-Dateien und unterschiedliche Spalten-Cases. Fix: harte Validierung + klare Logs.

2) Strategy Layer (Plugins)
- Signal-Generatoren als Module: insidebar_intraday, insidebar_breakout (künftig: marubozu_breakout etc.).
- Einheitliche Rückgabe: pro Symbol/Bar – Spalten long_entry, short_entry, sl_long, sl_short, tp_long, tp_short (+Meta).
- Filter (Universe/Volumen/ROC/News-Flags) vorgelagert; optional Zwei-Stufen-Pipeline (Top-N → Intraday Trigger).

3) Order-Export (OCO, TIF, Sessions, Sizing)
- cli_export_orders.py: tick-basiertes Runden (Decimal), Sessions als Gültigkeit (valid_to bis Sessionende), oco_id pro Zeile.
- Positionsgrößen-Modi:
  (a) fixed qty
  (b) Prozent vom Portfolio (z. B. 10 %; Rundung „Modulo 10 %“ wie besprochen)
  (c) risk-based (z. B. 1 % vom Portfolio, Stop-Abstand definiert; Tick-Rundung Pflicht).
- TP/SL aus Signalen; OCO-Verknüpfung zwischen Entry und Exit-Limits.

4) Backtesting Engine (Replay)
- First-Touch-Entry, SL/TP/EOD Exit, Slippage (bps), Fees (bps).
- Metriken: initial_cash, final_cash, net_pnl/pnl, num_trades, win_rate, sharpe, drawdown (falls vorhanden).
- Artefakte: filled_orders.csv, trades.csv, equity_curve.csv/png, metrics.json, manifest.json.
- Equity-Kurve & Drawdown im Dashboard; Units/Spalten: €-Format, „Stk“, Notional (qty*entry_price).

5) Dashboard (Streamlit)
- Runs-Browser (links Auswahl, Labelierung via label.txt pro Run).
- Sprechende Titel: „IB • Risk 1% • 10k • fees 2bps • 20 trades • WR 0.75 • PnL +3.61“.
- KPIs, Equity/Drawdown; Tabellen mit Einheiten (€, Stück) & Notional.
- Vergleich von Varianten über getrennte Runs (YAML-Configs).

Promotionspfad & Betrieb
Stufen & Checks
1) Experiment: kleine Samples, harte Validierung, lautes Fail-Fast.
2) Lab/Backtests: vollständige Metriken, Param-Sweeps (Grid/Random), Backtest-vs-Realtime-Paritätstests (Stichproben).
3) Paper: Realtime-Quelle, Order-Router (ib_insync), Guards (Exposure, MaxTrades, KillSwitch), strukturierte Logs.
4) Live: identische Deploys, Audit-Trail (idempotency_key, order-lifecycle), Alerting & Limits/KillSwitch.

Orchestrierung
- Heute: Make + systemd (pragmatisch, vorhanden).
- Später bei Bedarf: Prefect (dev-freundlich) oder Dagster (Software-Assets), ggf. Airflow. Start schlank, später wechseln.

Messaging/Streaming (optional)

```
- Leicht: Dateien/Funktionsaufrufe.
- Mittel: Redis Streams.
- Skalierung: Kafka/Redpanda (API-kompatibel, einfacher Betrieb).

Observability
- Logs: JSON-Struktur (run_id, oco_id, order_ref, symbol, side, prices, reason).
- Metriken: Prometheus-Exporter (orders_sent_total, fills_total, rejects_total, pnl_realized,
latency_ms).
- Dashboards: Grafana + Streamlit (Research/KPIs).

Repo-/Konfig-Struktur (vereinheitlicht)
repo/
  configs/
    runs/
      insidebar_replay.yml
      insidebar_replay_risk1.yml
      insidebar_replay_pos10pct.yml
      insidebar_replay_fixed1.yml
  src/
    data/       (adapters: eodhd, calendars, resample)
    signals/    (insidebar_intraday.py, filters, sizing utils)
    broker/     (ibkr_adapter.py, polygon_ws.py)
    engines/    (replay_engine.py)
    cli/        (cli_data, cli_signals, cli_export_orders)
  dashboards/
    run_dashboard.py (Runs-Browser, KPIs, Aktien/€-Einheiten, Notional-Spalten)
  artifacts/
    data_m1/, data_m5/, backtests/ (equity_curve.png, drawdown_curve.png, metrics.json,
label.txt, …)
  Makefile
  requirements.txt / pyproject.toml

YAML-Varianten (Beispiele)
- insidebar_replay_pos10pct.yml
  name: ib_pos10pct
  engine: replay
  mode: insidebar_intraday
  data: { path: artifacts/data_m5, tz: UTC }
  costs: { fees_bps: 2.0, slippage_bps: 1.0 }
  sizing:
    mode: pct_of_equity
    equity: 10000
    pos_pct: 10.0
  initial_cash: 10000

- insidebar_replay_risk1.yml
  sizing:
    mode: risk
    equity: 10000
    risk_pct: 1.0
```

```
    max_pos_pct: 20.0
    min_qty: 1

- insidebar_replay_fixed1.yml
  sizing:
    mode: fixed
    qty: 1
```

Teststrategie (QS)
- Unit-Tests: Resample M1→M5 (Ränder, leere Tage, DST), Signals (InsideBar), Sizing (fixed/%/risk inkl. Tickrundung).
- Property-Tests (Hypothesis): keine Duplikat-Orders, OCO-Kohärenz.
- Integration: Signals → Orders → Replay reproduziert erwartete Fills.
- Backtest-Parität: Stichproben Live-Bars gegen Historie (gleiche Session-Definition).
- Broker-Sandbox: IBKR Paper Flow (Create → Status → Fill → Bracket/OCO).

Kritische Punkte / Fallstricke (Learnings aus dem Thread)
- TZ & Sessions: Alles intern UTC; Sessions über Börsenkalender. UI/Anzeige Berlin.
- OHLCV-Casing: Immer Open/High/Low/Close/Volume groß, sonst bricht vieles still.
- Resample: Nur aus vollständigem M1 resamplen; keine ffill-Bars „über Nacht“. Session-Cuts beachten.
- Risk-Sizing: Stop-Abstand tick-runden (Decimal) – sonst Fill-/Stop-Drift.
- Order-Idempotenz: oco_id + idempotency_key; Retries dürfen keine Doppel-Orders erzeugen.
- Backtest-Bias: Keine Look-ahead-Infos; First-Touch-Entry, Reihenfolge SL/TP strikt einhalten.
- Monitoring: Ohne Metriken ist man blind – PnL/Orders/Latency erfassen und alarmeren.

Empfohlene Tools & Alternativen
- Orchestrierung: Start mit Make + systemd; später ggf. Prefect oder Dagster (erst bei Komplexität).
- Streaming: Dateien → Redis Streams → Redpanda/Kafka (skaliert).
- Data-Versionierung: Parquet + Manifeste; DVC/lakeFS, wenn Daten-Governance/Audit gebraucht wird.
- Backtesting-Engines: Eigener Replay (leicht, passgenau); Alternativ Backtesting.py, vectorbt, Lean (mehr Features, aber mehr Abhängigkeiten).
- Kalender: pandas-market-calendars; alternativ exchange_calendars (beides ok).

Konkrete nächste Schritte
1) Repo konsolidieren (Signals/Engines/Adapter trennen, configs/runs/* anlegen).
2) Drei InsideBar-Varianten (fixed / pos 10 % / risk 1 %) als YAML + drei Backtest-Runs erzeugen.
3) Dashboard nutzen: Runs vergleichen; label.txt je Run für sprechende Namen.
4) Paper-Stufe: IBKR Paper mit ib_insync OrderRouter (Idempotenz/OCO), Realtime-Quelle konsolidieren, Guards aktivieren.
5) Observability: Prometheus-Exporter + Log-Schema finalisieren.
6) Optional Orchestrator: Prefect-Flow „daily_backtest → report → promote_if_green“.

Hinweis: Dieses Dokument fasst den besprochenen Architekturentwurf und unsere Learnings (z. B. OHLCV-Casing, UTC/Session-Handling, Sizing/Decimal-Rundung, Dashboard-Einheiten und Notional) zusammen und dient als Grundlage für die Auswertung durch factory.ai.