



TRADING
JOURNAL

| TradingJournal

Technisches Handbuch

Tipp zum Lesen am Bildschirm: Links mit der **rechten Maustaste** anklicken und „Link in neuem Tab öffnen“ wählen — ein normaler Klick öffnet die Seite im selben Fenster, und das PDF ist weg.

Version v1.4.0 · Stand 30.10.2026



Inhalt

Architektur und Import-Pipeline 8

Inhaltsverzeichnis 8

1. Überblick & Ziele 8

2. Datenquellen 9

3. Pipeline-Architektur 11

4. Phase 0 — E-Mail-Abruf (get_flexquery_emails.py) 13

5. Phase 1 — Import (XML/CSV → trades_raw) 15

6. Phase 2 — Processing (trades_raw → Endtabellen) 21

7. Kernkonzepte 30

8. Datenbankstruktur 32

9. Konfiguration & Multi-User 36

10. Fehlerbehandlung & Robustheit 41

11. Ausblick — Erweiterungen & Verbesserungen 42

Anhang: Technologie-Stack 45

Anhang: Wichtige Dateipfade 45

Aufbau einer Flex Query 48

1. Beispiel 48

2. Welche Felder der Import liest 49

3. Mail- und API-Query 50



Befehlsreferenz 51

Environments	51
Table-Prefix	51
Account-Zuordnung (Routing)	51
Verzeichnisstruktur (XML-Dateien)	52
TEST-Environment — manuelle XMLs automatisch erkennen	53
run_pipeline.py — Vollständiger Pipeline-Runner	54
get_flexquery_emails.py — E-Mails abrufen	56
get_flexquery_api.py — Flex Query per Web-API abrufen	56
import_csv_snapshots.py — account_snapshots	57
import_ibkr_trades.py — XML → trades_raw	58
process_ibkr_trades.py — trades_raw → trades_open / trades_close	58
enrich_greeks.py — Delta, IV, IVR, IVP über die TWS-API (optional)	59
xml_append_summary.py — Einzel-XMLs zu summary.xml zusammenfassen	60
reset_IBKR_DB.py — Datenbank initialisieren	61
db-backup.ps1 — Datenbanken sichern und wiederherstellen	62
Typischer Ablauf	63



Installation: Windows-Grundausrüstung 65

Übersicht	65
1. Git for Windows	66
2. XAMPP (Apache + PHP + MySQL)	66
3. Composer	70
4. Node.js + npm	71
5. Python 3.14	71
6. Projektverzeichnis anlegen und Repos klonen	71
Zu „Artisan“	73
Verifikations-Checkliste	73

Installation: TradingJournal-Weaver 75

Voraussetzungen	75
1. Repository klonen	75
2. Virtuelle Umgebung anlegen	76
3. .env anlegen	76
4. Datenbanken initialisieren	78
5. pipeline.toml anlegen	78
6. Ersten Lauf testen	79
7. Altdaten nachholen (einmalig)	79
8. Web-API: DNS-Auflösung des IBKR-Hosts	80
Weiterführende Dokumentation	81



Installation: TradingJournal

82

Voraussetzungen

82

1. Repository klonen + Abhängigkeiten

82

2. .env anlegen

83

3. Laravel-eigene Tabellen anlegen

84

4. Trading-Tabellen bereitstellen

84

5. Ersten User anlegen

85

6. Assets bauen

86

7. Starten

86

Weiterführende Dokumentation

87



Server-Betrieb — TradingJournal auf einem eigenen Server 88

Voraussetzungen	88
Notation in dieser Anleitung	88
Schritt 1 — Git for Windows	89
Schritt 2 — MySQL 8.4 LTS	89
Schritt 3 — Python 3.14	91
Schritt 4 — PHP 8.3 (Thread-Safe)	91
Schritt 5 — Apache HTTP Server	92
Schritt 6 — Composer	94
Schritt 7 — Code deployen	94
Schritt 8 — .env-Dateien	95
Schritt 9 — Laravel-Grundgerüst	96
Schritt 10 — Datenbank einrichten	96
Schritt 11 — pipeline.toml + accounts.toml	98
Schritt 12 — Abschlusstest	99
Troubleshooting	100

Verteilter Abruf — minimaler Client für den Datenabruf 103

Warum das minimal bleiben kann	103
Server-Seite: SSH-Empfang einrichten (einmalig)	103
Client-Seite: Minimal-Installation	104
Täglicher Ablauf	108



Bekannte Grenzen und offene Fragen

109

1. Bekannte Grenzen

109

2. Ungetestet – Beispiele gesucht

111

3. Mithelfen

111

Architektur und Import-Pipeline

Projekt: TradingJournal-Weaver — Trading-Journal Import-System

Ziel: Vollständige Dokumentation des Datenflusses von IBKR-Rohdaten (XML/CSV) bis zur befüllten MySQL-Datenbank

Inhaltsverzeichnis

1. Überblick & Ziele
2. Datenquellen
3. Pipeline-Architektur
4. Phase 0 — E-Mail-Abruf
5. Phase 1 — Import (XML/CSV → trades_raw)
6. Phase 2 — Processing (trades_raw → Endtabellen)
7. Kernkonzepte
8. Datenbankstruktur
9. Konfiguration & Multi-User
10. Fehlerbehandlung & Robustheit
11. Ausblick — Erweiterungen & Verbesserungen

1. Überblick & Ziele

Was ist das System?

Das TradingJournal ist eine **Webanwendung, die auf dem eigenen Rechner (oder einem eigenen Server) läuft** und im Browser bedient wird — weder eine App aus einem App-Store noch eine Online-Plattform. Es besteht aus zwei Programmen, die sich eine MySQL-Datenbank teilen: **TradingJournal-Weaver** (Python, ohne eigene Oberfläche) holt und verarbeitet die Daten, **TradingJournal** (Laravel) zeigt sie im Browser an. Dieses Kapitel beschreibt vor allem den Weaver-Teil.

Das System liest automatisch **IBKR Flex-Query-Reports** (XML) und **Activity Statements** (CSV) ein und überführt die Rohdaten in ein strukturiertes Trading-Journal mit:

- **FIFO-Lot-Matching** für Aktien und Optionen (Devisentausch wird separat erfasst, ohne FIFO)
- **Kampagnen-Tracking** (gebündelte Trades je Symbol/Zeitraum)
- **Options-Strategie-Erkennung** (Spreads, Iron Condors, Straddles, ...)
- **Vollständigem Audit-Trail** (jede Aktion im `campaign_ledger`)
- **Multi-User/Multi-Environment-Support** (Echtgeld, Paper, Test, Automatisiert)



Warum zwei Phasen?

Die Trennung in **Import** (XML → `trades_raw`) und **Processing** (Matching + Kampagnen) ist bewusst gewählt:

Vorteil	Erklärung
Wiederholbarkeit	Import kann mehrfach ausgeführt werden (Checksums verhindern Doppelimport)
Debugging	<code>trades_raw</code> zeigt exakt was IBKR geliefert hat, unverändert
Trennung von Verantwortlichkeiten	Parser-Logik vs. Business-Logik klar getrennt
Datenbank-Reset	<code>reset_IBKR_DB.py</code> setzt alle Tabellen zurück (inkl. <code>trades_raw</code>) — Summary-Files ermöglichen den vollständigen Neuaufbau der Trades ohne Originaldateien (Kontoauszüge/ <code>account_snapshots</code> sind nicht enthalten)

2. Datenquellen

2.1 Flex-Query-XML

IBKR generiert **täglich** einen Flex-Query-Report als XML. Die Datei enthält alle Aktivitäten des Kontos für den gewählten Zeitraum.



Struktur:

```
<FlexQueryResponse>
  <FlexStatements>
    <FlexStatement accountId="U1234567" fromDate="..." toDate="...">
      <Trades>
        <Trade tradeID="..." symbol="AAPL" buySell="BUY" quantity="100" ... />
        <Trade tradeID="..." assetCategory="CASH" symbol="EUR.USD" ... />
      </Trades>
      <OptionEAE>
        <OptionEAE tradeID="..." transactionType="Expiration" ... />
      </OptionEAE>
      <CorporateActions>
        <CorporateAction type="RS" symbol="APDN.OLD" quantity="-1500" ... /> ←!—
        Reverse Split 1:50, alter Teil →
        <CorporateAction type="RS" symbol="APDN" quantity="30" ... /> ←!— neuer
        Teil, gleiche actionID →
      </CorporateActions>
    </FlexStatement>
  </FlexStatements>
</FlexQueryResponse>
```

Asset-Kategorien:

Code	Bedeutung	Beispiel
STK	Aktien	AAPL, TSLA
OPT	Optionen	AAPL 250117C00150000
CASH	FX-Trades	EUR.USD, GBP.USD

2.2 Activity Statement CSV

IBKR liefert täglich auch ein **Activity Statement CSV** — ein mehrteiliges Dokument mit Kontosnapshots (Cash, Net Liquidation, Margin, etc.). Es enthält **keine Einzeltrades**, sondern Kontostands-Daten.

Die CSV landet in einem **separaten Verzeichnis**: `C:\Trade_CSV\incoming\` (nur per Mail — kein API-Äquivalent für Activity Statements, siehe Phase 0). XML-Files kommen nach `C:\Trade_XML\incoming_mail\` (Mail) bzw. `C:\Trade_XML\incoming_api\` (Flex-Query-Web-API, zweiter gleichwertiger Kanal für Trades).

`C:\Trade_XML\incoming_mail\` und `C:\Trade_CSV\incoming\` werden über `get_flexquery_emails.py` befüllt, `C:\Trade_XML\incoming_api\` über `get_flexquery_api.py` (siehe Phase 0).



```
import_ibkr_trades.py

flex_parser.py (Orchestrator)
├─ day_trade_processor.py → Trades, OptionEAE,
│                           CorporateActions
├─ option_strategies.py → Spread-Erkennung
├─ book_trades.py → BookTrade-Matching
├─ Checksum-Prüfung → Duplikat-Schutz

INSERT → trades_raw (processed=0)
INSERT → trades_fx (FX-Trades separat)
```



```
PHASE 2: PROCESSING
process_ibkr_trades.py

inventory_tracker.py → Kampagnen-Lebenszyklus
open_trade.py → FIFO-Lot-Schema
closed_trade.py → Closed-Trade-Schema

Ergebnis:
├─ trades_open ← Aktive FIFO-Tranchen
├─ trades_close ← Abgeschlossene Trades
├─ trading_campaigns ← Kampagnen je Symbol
├─ campaign_ledger ← Vollständiges Audit-Log
```



```
PHASE 3: (OPTIONAL) SUMMARY
xml_append_summary.py
├─ Erstellt Zusammenfassungs-XML für nächsten Import
```

3.2 Pipeline-Runner

Alle Schritte werden über `run_pipeline.py` gesteuert:

```
python run_pipeline.py          # alle Schritte
python run_pipeline.py --steps import  # nur Import
python run_pipeline.py --steps process  # nur Processing
python run_pipeline.py --envs money    # nur Echtgeld
python run_pipeline.py --users dein_user  # nur User dein_user
```



pipeline.toml — Konfiguration:

```
[settings]
steps = ["fetch_emails", "fetch_api", "snapshot_csv", "import", "process", "summary"]
#       ↑ Phase 0 Mail  ↑ Phase 0 API ↑ Snapshots   ↑ Phase 1  ↑ Phase 2  ↑ Phase 3
# optional nach "process": "enrich" (Delta/IV/IVR/IVP über die TWS-API, § 6.5)

[users.dein_user]
environments = ["money", "paper", "autom_money", "autom_paper"]

[users.zweiter_user]
environments = ["money"]
```

Tabellen-Prefix kommt aus `users.toml` (zentrale, lokale Registry, siehe § 9.1) — in `pipeline.toml` nur bei Bedarf per `prefix = "..."` explizit überschreibbar.

`account_snapshots` kommt ausschließlich aus dem Kontoauszug (CSV per Mail) über `snapshot_csv`, siehe § 8.2 — die Flex Queries enthalten keine Kontostands-Daten.

Der `import`-Step läuft ohne `--users / --envs`-Filter **einmal global** (accountId-Routing deckt intern alle User/Envs ab); mit gesetztem Filter erzeugt `run_pipeline.py` einen Task pro passendem User x Environment und nutzt dafür das CLI-Override `import_ibkr_trades.py <env> <prefix>` (§ 5.1).

4. Phase 0 — E-Mail-Abruf (`get_flexquery_emails.py`)

IBKR stellt Flex-Query-Reports und Activity Statements **automatisiert per E-Mail** zu.

`get_flexquery_emails.py` übernimmt den Abruf und die Verteilung in die richtigen Eingangsverzeichnisse.



```
get_flexquery_emails.py
|
|— Verbindet sich per IMAP mit dem konfigurierten Postfach (.env)
|— Durchsucht Inbox nach IBKR-Report-Mails (Betreff-Filter)
|
|— Für XML-Anhänge (Flex-Query-Report):
|   └ Speichert in C:\Trade_XML\incoming_mail\
|       (enthält Trades, OptionEAE, CorporateActions)
|
|— Für CSV-Anhänge (Activity Statement):
|   └ Speichert in C:\Trade_CSV\incoming\
|       (enthält Kontostands-Snapshots, keine Einzeltrades)
|
|— Löscht verarbeitete IBKR-Mails aus dem Postfach
```

Konfiguration in `.env` :

```
IMAP_USERNAME=trading@example.com
IMAP_PASSWORD=...
IMAP_SERVER=imap.gmx.com
IBKR_ACTIVITY_SUBJECT=Activity Statement
IBKR_ACTIVITY_SUBJECT_DE=Kontoauszug
```

Wichtig: Phase 0 ist optional — XML/CSV-Dateien können auch manuell in die Eingangsverzeichnisse kopiert werden. Die nachfolgenden Phasen 1 und 2 sind davon unabhängig. Ein Namensmuster ist im Normalbetrieb **nicht** erforderlich (siehe Phase 1) — nur der dedizierte TEST-Modus verlangt

`flex.*.xml` .

Zweiter Kanal `fetch_api` (`get_flexquery_api.py`): Abruf der Flex-Query-XMLs direkt über die IBKR-Web-API (`.env` : `IBKR_FLEX_QUERIES=token:query_id,...`), läuft in `pipeline.toml` gleichwertig neben `fetch_emails` als zweite produktive Quelle für `trades_raw` . Schreibt nach `C:\Trade_XML\incoming_api\` (getrennt von `incoming_mail\` , damit beide Quellen unabhängig nachvollziehbar bleiben); der `import` -Step scannt **beide** Ordner. Checksum-Dedup verhindert doppelte Trades, falls dieselben Daten über beide Kanäle ankommen. Ist die API-Query mit denselben Sektionen eingerichtet wie die Mail-Query (Trades, Option Exercises/Assignments/Expirations, Corporate Actions), liefert sie auch `OptionEAE` /BookTrades und CorporateActions. Timing: Der Report wird bei Anfrage on-demand generiert, es gibt kein festes Tages-Zeitfenster — die Daten des Vortags stehen in der Regel schon am frühen Morgen bereit. Retry-Logik fängt die kurze „noch in Verarbeitung“-Phase ab (typisch wenige Sekunden, bis ~30s bei Accounts mit geringerer IBKR-Priorität, z.B. Paper-Konten). Damit auch ein unabhängiger zweiter Kanal für Ausfallsicherheit, falls die Mail an einem Tag ausbleibt.



5. Phase 1 — Import (XML/CSV → trades_raw)

5.1 Einstiegspunkt: `import_ibkr_trades.py`

```
import_ibkr_trades.py
|
| — Für jede *.xml-Datei (aus C:\Trade_XML\incoming_mail\ UND incoming_api\,
|   incoming_mail zuerst – Quelle ist fuer den Import irrelevant):
|   | — Lese <FlexStatement> (kann mehrere je Datei enthalten)
|   | — Extrahiere accountId → route zu Environment/User (via config.py)
|   | — Rufe flex_parser.process_flex_file() auf
|   | — Verschiebe Datei in <env>/<user>/processed/ oder deleted/
| — Scanne C:\Trade_CSV\incoming\ nach CSV-Dateien:
|   | — activity_csv_parser → account_snapshots
```

Kein Dateinamen-Filter (Normalbetrieb): Verarbeitet werden alle `*.xml` -Dateien in beiden Ordnern, unabhängig vom Namen — Mail- und API-Dateien haben verschiedene Namensschemata. Einzige Ausnahme: der TEST-Modus (`test\<user>\incoming\`) verlangt weiterhin `flex.*.xml` .

CLI-Override — gezielt ein Environment importieren:

```
python -m src.import_ibkr_trades money dein_user_
```

Analog zu `process_ibkr_trades.py` : nur Dateien, deren accountId auf dieses Environment/Prefix routet, werden importiert und verschoben — alle anderen bleiben in `incoming_mail\` / `incoming_api\` liegen. `run_pipeline.py` nutzt diesen Modus automatisch, sobald `--users` / `--envs` gesetzt sind (§ 3.2).

Routing via `accounts.toml` (`src/config.py` → `get_account_config()`):

Das Mapping steht — wie `pipeline.toml` — in einer maschinen-/personenspezifischen, nicht versionierten Datei (`accounts.toml` , Vorlage `accounts.toml.example`), nicht im Code:

```
# Key = letzte 4 Ziffern der IBKR accountId
# "U**1234" und "U12341234" ergeben denselben Key "1234"
[1234]
user = "dein_user"
environment = "money"
```

Tabellen-Prefix steht bewusst nicht hier, sondern wird zur Laufzeit aus `users.toml` aufgelöst (`get_account_config()`) — einzige Quelle für `user` → `prefix` im Projekt.



Neue IBKR-Konten müssen in `accounts.toml` ergänzt werden — XMLs mit unbekannter `accountId` werden beim Import übersprungen (Warnung im Log). Neue User zusätzlich in `users.toml` eintragen.

Jede XML-Datei enthält immer nur Daten eines einzigen Kontos. Die `accountId` im `<FlexStatement>` bestimmt, welchem User und Environment die Trades zugeordnet werden.

5.2 Kern-Parser: `flex_parser.py`

Der Flex-Parser ist der **Orchestrator** der Import-Phase. Er koordiniert alle Sub-Parser.

```
process_flex_file(xml_path, db, repo, logger, environment)

|
|— 1. Extrahiere Rohdaten (parallel als DataFrames):
|   |— extract_trades_for_db()          → df_trades
|   |— extract_optioneae_for_db()      → df_eae
|   |— extract_corporateaction_for_db() → df_corp
|
|— 2. Aggregiere gleichartige Trades:
|   |— Gruppenbildung: (conid, symbol, buySell, ibOrderID)
|       Hintergrund: Exchange-Fills können in mehreren Teil-Trades kommen
|
|— 3. Options-Strategie-Erkennung:
|   |— option_strategies.detect_strategies(df_trades)
|
|— 4. BookTrade-Matching:
|   |— book_trades.handle_booktrades(df_trades, df_eae)
|
|— 5. CorporateActions sortieren:
|   |— sort_corporate_actions(df_corp)
|
|— 6. Checksum-Berechnung & Duplikat-Check:
|   |— MD5(tradeID + datetime + symbol + qty + price)
|   |— SELECT checksums FROM trades_raw WHERE checksum IN (...)
|
|— 7. Symbol-Lookup/-Anlage:
|   |— get_symbols_bulk() → INSERT neue Instrumente
|
|— 8. INSERT in Datenbank:
|   |— INSERT INTO trades_raw (processed=0)
|   |— INSERT INTO trades_fx
```

5.3 Element-Extraktion: `day_trade_processor.py`

Normale Trades (`extract_trades_for_db`):

Liest alle `<Trade>`-Elemente und mappt IBKR-Attribute auf interne Spaltennamen.



Sonderfall CASH-Trades (FX):

IBKR meldet einen Währungstausch (`assetCategory="CASH"` , Symbol `<Basis>.<Quote>` , z. B. `EUR.USD`) als **einen** `<Trade>` . Daraus entstehen drei Zeilen — eine in `trades_fx` , zwei in `trades_raw` :

```
trades_fx (1 Zeile je Tausch – Grundlage der Ansicht „FX-Trades“):
    baseCurrency/quoteCurrency  aus dem Symbol (EUR / USD)
    baseAmount                  = quantity  (Betrag in Basiswährung, Vorzeichen von IBKR)
    quoteAmount                  = proceeds  (Betrag in Gegenwährung)
    fxRate                       = tradePrice
    commission                   = ibCommission

trades_raw (2 Zeilen, gemeinsame linkID – „von“ und „nach“):
    Zeile „von“: quantity = 1, currency = Basis, tradePrice = Kurs, netCash =
    Basisbetrag,
                ibCommission = Kommission
    Zeile „nach“: quantity = 2, currency = Quote, tradePrice = 1/Kurs, netCash =
    Gegenbetrag,
                ibCommission = 0
```

`quantity` ist hier **keine Menge**, sondern nur die Kennung 1 = „von“, 2 = „nach“. Beispiel aus echten Daten: EUR → USD zum Kurs 1,1686 ergibt in `trades_raw` die Zeilen `(quantity 1, EUR, 1,1686, netCash -498,30)` und `(quantity 2, USD, 0,8557, netCash +582,31)` , in `trades_fx` eine Zeile `EUR/USD, -498,30 / +582,31, Kurs 1,1686` . Ein FIFO-Matching findet für FX **nicht** statt (siehe § 6.2, Fall O). Die Umrechnung von Trades in Euro hängt nicht an diesen Zeilen, sondern am `fxRateToBase` , das IBKR bei jedem Trade mitliefert. Die so mitgeführten EUR-Werte zeigt die Weboberfläche derzeit nur auf den Detailseiten geschlossener Trades neben dem USD-Wert (siehe § 11, Punkt H).

Kein Forex-Trading: Erfasst wird nur der Umtausch als solcher (z. B. EUR → USD für den Aktienhandel). Devisenhandel als Strategie — Währungspositionen mit Gewinn/Verlust aus Kursbewegungen — ist nicht umgesetzt: keine Tranchen, kein P&L, keine Kampagnen für `CASH` .

OptionEAE-Extraktion (`extract_optioneae_for_db`):

<code>transactionType</code>	Bedeutung
<code>Expiration</code>	Option verfallen (wertlos)
<code>Exercise</code>	Option ausgeübt (durch Holder)
<code>Assignment</code>	Option zugewiesen (durch Writer)
<code>Cash Settlement</code>	Barzahlung statt Delivery (z.B. SPX, XSP)



CorporateActions (`extract_corporateaction_for_db`):

IBKR-Type	Bedeutung
RS	Reverse Split
SS	Stock Split (Forward)
FS	Forward Split
DI	Dividende
TC	Takeover / Merger

5.4 Options-Strategie-Erkennung: `option_strategies.py`

IBKR liefert in den Flex-Query-Daten **keine Information über synthetische Produkte** wie Spreads oder andere Multi-Leg-Konstrukte — jedes Leg erscheint als eigenständiger, isolierter Trade ohne Verweis auf eine übergeordnete Strategie. Das System muss daher **heuristisch** erkennen, welche Einzeltrades zusammengehören.

`detect_strategies()` läuft beim Import je FlexStatement (also je Handelstag), nachdem Teilausführungen zusammengefasst sind, und betrachtet nur echte Börsen-Trades (`transactionType="ExchTrade"` ; BookTrades aus Verfall/Zuteilung nie). Zwei Gruppierungen werden nacheinander angewendet:

Methode 1: `ibOrderID`-Gruppierung (primär)

IBKR vergibt allen Legs einer Combo-Order dieselbe `ibOrderID`.
→ Options-Trades mit gleicher `ibOrderID` (mind. 2) = Kandidat

Methode 2: 3-Sekunden-Raster (Fallback)

Für einzeln eingegebene Legs ohne gemeinsame `ibOrderID`:
→ übrige Options-Trades desselben Underlyings, deren `dateTime` in dasselbe feste 3-Sekunden-Raster fällt (`dateTime` abgerundet auf 3 s) = Kandidat

Das Raster ist fest, kein gleitendes Fenster: Legs um 10:00:02 und 10:00:04 liegen in verschiedenen Rastern (10:00:00 und 10:00:03) und werden **nicht** zusammengefasst.

Klassifizierung: Je Kandidat werden die Legs nach Strike/Typ/Verfall zusammengefasst (Mengen summiert) und nach Anzahl der Legs eingeordnet. Nur wenn ein Name passt, bekommen alle Legs dieselbe `opt_strategyID` (UUID), `opt_strategyName` und `opt_strategy=1` ; sonst bleiben sie Einzeloptionen.



Name	Legs	Bedingung im Code
Bull Put Spread	2	Puts, gleicher Verfall, gleiche Menge, entgegengesetzt; höherer Strike short
Bear Put Spread	2	wie oben, höherer Strike long
Bear Call Spread	2	Calls, gleicher Verfall, gleiche Menge, entgegengesetzt; tieferer Strike short
Bull Call Spread	2	wie oben, tieferer Strike long
Calendar Spread	2	gleicher Typ und Strike, verschiedener Verfall, entgegengesetzt
Diagonal Spread	2	gleicher Typ, verschiedener Strike und Verfall, entgegengesetzt
Straddle	2	Call + Put, gleicher Strike und Verfall, gleiche Richtung
Strangle	2	Call + Put, verschiedener Strike, gleicher Verfall, gleiche Richtung
Call / Put Butterfly	3	ein Typ, ein Verfall, Mengen 1 : 2 : 1, Summe 0
Iron Condor	4	2 Calls + 2 Puts, ein Verfall, Calls und Puts heben sich je auf

Nicht erkannt (bleiben Einzeloptionen): ungleiche Mengen bei 2 Legs (Ratio Spreads), mehr als 4 Legs, Legs an verschiedenen Tagen.

Was mit dem Namen passiert: `trades_raw` speichert ihn an jedem Leg. `trades_open` übernimmt ihn beim Eröffnen, `trades_close` übernimmt ihn **von der Eröffnung** — die Spreads-Ansichten in TradingJournal gruppieren `trades_close` nach dieser `opt_strategyID`. Die Kampagne hängt unabhängig davon am Underlying (§ 6.3).

Grenzen (gegen Code und Echtdateien geprüft):

- **Schließende Combo-Orders bekommen einen eigenen, „umgekehrten“ Namen.** Die Erkennung kennt keine Eröffnung/Schließung: Wer einen Bull Put Spread per Combo-Order zurückkauft, erzeugt in `trades_raw` einen „Bear Put Spread“ (in einem Konto der Echtdateien sind alle 32 „Bear Put Spreads“ und alle 10 „Bull Call Spreads“ solche Schließungen). Auswirkung nur auf die Ansicht *Raw Data* — `trades_close` und die Spreads-Auswertung verwenden die Strategie der Eröffnung.
- **Rolls werden als Spread erkannt.** Ein Roll als Combo-Order (alte Option zurückkaufen, neue verkaufen) sieht aus wie ein Calendar- oder Diagonal-Spread. Die neu eröffnete Option trägt dann diesen Namen und erscheint später als „Spread“ mit nur einem Leg (PAPER: 2 Diagonal Spreads, außerdem je ein bis drei gemischte Fälle in zwei weiteren Konten).
- **Iron Butterfly** (Short Call und Short Put am selben Strike) erfüllt die Iron-Condor-Regel und heißt deshalb „Iron Condor“; die Reihenfolge der Strikes wird nicht geprüft.

5.5 Verfall, Ausübung, Zuteilung: `book_trades.py`

Schließt IBKR eine Option automatisch (Verfall, Ausübung, Zuteilung, Barausgleich), stehen im XML zwei Dinge: der **BookTrade** im `Trades`-Block (`transactionType="BookTrade"`), der die Option ausbucht, und das **OptionEAE**-Ereignis, das den Grund nennt. Bei einer Zuteilung oder Ausübung mit



Aktienlieferung kommt ein dritter Eintrag dazu: der **Aktien-Trade** zum Strike, als eigener Eintrag im **Trades**-Block. `flex_parser.py` trennt die BookTrades ab und gibt sie mit den Options-EAE (`assetCategory="OPT"`) an `handle_booktrades()`.

Schritt 1 — Status am Options-BookTrade (Parser):

Assignment / Exercise / Expiration:

BookTrade.tradeID = OptionEAE.tradeID → Ereignistyp übernehmen

Cash Settlement (eigene tradeID ohne Bezug zum BookTrade):

BookTrade.conid = OptionEAE.conid und Datum höchstens 1 Tag auseinander

Status (Spalte `trades_raw.status`):

Assignment oder Exercise → "Assigned" (auch die Ausübung – einen eigenen Status "Exercised" gibt es nicht)

zusätzlich Cash Settlement → "CashSettled", proceeds = Barausgleich
(SPX: Exercise/Assignment + Cash Settlement)

Expiration → "Expired"

nur Cash Settlement → "CashSettled" + Warnung (sollte nicht vorkommen)

Schritt 2 — Option mit Aktie verknüpfen (Parser, nur bei Assignment):

Voraussetzung: OptionEAE hat `underlyingConid` (fehlt beim Barausgleich)

1. `linkID` = neue UUID → auf den Options-BookTrade

2. Aktien-Trade suchen:

`assetCategory` = "STK"

`conid` = `underlyingConid` der Option

`|quantity|` = `|Kontrakte|` × `multiplier` (z. B. 1 × 100 = 100 Aktien)

noch nicht mit einer anderen Zuteilung verknüpft

bei mehreren Treffern: der zeitlich nächste zum EAE-Datum

3. Treffer → gleiche `linkID` + status "Assigned" auf den Aktien-Trade

kein Treffer → Logzeile "Kein passender Stock-Trade gefunden", Option behält `linkID`

Beide Einträge landen so in `trades_raw` mit derselben `linkID`.

Schritt 3 — Verarbeitung (`process_ibkr_trades.py`): Der Options-BookTrade schließt die offene Options-Tranche (FIFO, § 6.2). In `trades_close` werden `status` und `closedBy` gesetzt (`Assigned` → `ASSIGNMENT`, `Expired` / `CashSettled` → `EXPIRY`); die `linkID` kommt dabei vom schließenden BookTrade, nicht von der Eröffnung (die nie eine hat). Im Fall 2c (exakter Gegenabschluss) schreibt das Ledger `OPTION_EXERCISE` bzw. `OPTION_EXPIRATION` / `CASH_SETTLEMENT`. Bei `OPTION_EXERCISE` merkt sich der Tracker die `linkID` als `pending_link_id` am Underlying; eröffnet der nächste Aktien-Trade dieses Underlyings eine Position (Fall 1), bekommt sein Ledger-Eintrag diese `linkID`. Schließt der Aktien-Trade dagegen eine bestehende Position (z. B. Covered Call zugeteilt → Aktien gehen weg), trägt er die `linkID` aus Schritt 2 direkt in seinen `trades_close`-Eintrag. Option und Aktie landen in derselben Kampagne, weil die Kampagne am Underlying hängt (§ 6.3).



Nutzung in TradingJournal: `SpreadList` / `SpreadDetail` addieren das Ergebnis der über `linkID` verknüpften Aktien-Trades zum Spread-Ergebnis (sonst wäre ein zugeteilter Bull Put Spread zu optimistisch, weil der Verlust aus der Aktienzuteilung fehlt). Die Detailseiten offener und geschlossener Trades zeigen verknüpfte Trades mit gleicher `linkID` an.

Grenzen (gegen Code und Echtdaten geprüft):

- **Ausübung eigener Optionen** (*Exercise*): Der Parser vergibt keine `linkID`, der Aktien-Trade wird nicht gesucht. Nur im Fall 2c erzeugt die Verarbeitung eine `linkID` für den Ledger-Eintrag des nächsten eröffnenden Aktien-Trades; in `trades_close` / `trades_open` bleibt sie leer.
- **Ohne Aktien-Gegenstück** (z. B. Barausgleich) bleibt die `linkID` des Options-Eintrags ohne Partner. Ältere Datenbestände lassen sich mit `dev/migrate_backfill_assignment_linkid.py` nachträglich verknüpfen (idempotent).

5.6 Duplikat-Schutz: Checksums

Konzept: Jeder Trade erhält vor dem Insert einen MD5-Hash:

```
checksum = MD5(tradeID + dateTime + symbol + quantity + price)
```

Ablauf:

1. Berechne Checksums für alle neuen Trades
2. `SELECT checksums FROM trades_raw WHERE checksum IN (...)`
3. Filtere bekannte Checksums heraus
4. `INSERT` nur wirklich neue Trades

Vorteil: Dasselbe XML kann mehrfach importiert werden (z.B. nach Neustart) — keine Duplikate.

6. Phase 2 — Processing (trades_raw → Endtabellen)

6.1 Einstiegspunkt: `process_ibkr_trades.py`

```
# Hole alle unverarbeiteten Trades (processed=0), chronologisch sortiert
trades = repo.get_unprocessed_trades(environment)

for trade in trades:
    process_raw_trade(logger, trade, repo, tracker, campaign_id)
    repo.mark_trade_processed("trades_raw", trade["id"])
```



Schlüsselprinzip: Trades werden in **Zeitreihenfolge** verarbeitet. FIFO ist nur korrekt, wenn ältere Positionen zuerst geschlossen werden.

Reihenfolge-Garantie — und ihre Grenze: Sortiert wird nur die aktuelle Menge

`trades_raw.processed=0 ( DBAccess.get_all_raw_trades : ORDER BY dateTime ASC, symbol ASC )`. Innerhalb eines Laufs sind Dateireihenfolge, Quelle (`incoming_mail / incoming_api`) und Mehrfachlieferungen (Checksum-Dedup beim Import) deshalb irrelevant. Bereits verarbeitete Trades werden aber nicht neu einsortiert: Ein Trade mit `dateTime` vor dem jüngsten verarbeiteten Trade desselben Environments liefe gegen den aktuellen `trades_open` -/Tracker-Stand. Davor schützt ein **Guard** in `process_trades()` (`find_out_of_order_trades()` , seit 2026-10-28): Liegen unverarbeitete Trades (ohne CASH) vor `MAX(dateTime)` der verarbeiteten (`DBAccess.get_last_processed_raw_datetime`), wird das Environment nicht verarbeitet, das Modul endet mit Exit-Code 1 (`run_pipeline.py` meldet *process* als FEHLER). Ausgenommen sind Options-Ereignisse (`OPT` mit Status `Expired / Assigned / CashSettled`), die IBKR gelegentlich erst mit dem Folgebericht liefert — nur Hinweis im Log; die Aktie aus einer Zuteilung bleibt geprüft. Erzwingen: `--reihenfolge-ignorieren` . Beispiel ohne Guard: Kauf 02.03. kommt nach dem bereits verarbeiteten Verkauf 10.03. → der Verkauf hat zuvor ohne Bestand per Fall 1 eine Short-Tranche eröffnet, der Kauf schließt sie (Fall 2c) — Bestand 0, aber Richtung, `dateTimeOpen / dateTimeClose` und Kampagnenzuordnung falsch. Im Regelbetrieb tritt das nicht auf (jeder Bericht überlappt die letzten n Tage, ein Trade kommt im ersten Bericht nach dem Handelstag), relevant ist es beim Nachholen von Altdaten nach dem ersten produktiven Lauf oder bei lange liegengebliebenen Dateien (z. B. unbekannte `accountId` , später in `accounts.toml` ergänzt). Kampagnen-Ablauf (`close_old_pending_campaigns`) und PENDING→CLOSED richten sich dagegen nach Trade-Datum (`reference_date` = erster Trade des Laufs), nicht nach `now()` . **Korrektur:** Neuaufbau per `reset_IBKR_DB.py + --from-summary` (Summary ist nach `fromDate` sortiert). `reset_IBKR_DB.py` legt dabei u. a. auch `trade_planner` , `trade_planner_legs` und `account_snapshots` neu an — Trade-Log-Einträge vorher sichern, Snapshots durch Zurücklegen der CSVs aus `C:\Trade_CSV\incoming\<benutzer>\processed\` neu importieren. `account_snapshots` selbst sind reihenfolgeunabhängig (ein Datensatz je `snapshot_date` , Dedup).

6.2 FIFO-Matching: Die Fälle

IBKR-Vorzeichenkonvention:

- **BUY:** `netCash` negativ (Geld fließt raus)
- **SELL:** `netCash` positiv (Geld fließt rein)

```
Aktueller Bestand im Tracker: current_qty
Neuer Trade:                new_qty (positiv = Long, negativ = Short)

FIFO-Queue: offene Tranchen in trades_open (processed=0) je Symbol/conid,
            ältester Eintrag zuerst (ORDER BY dateTime ASC)
```

Geschlossene Tranchen werden in `trades_open` nicht gelöscht, sondern auf `processed=1` gesetzt (History). Wirklich offen ist nur, was `processed=0` hat.

**Fall 0: CASH / FX-Trade**

```
assetCategory = "CASH"    (die beiden „von“/„nach“-Zeilen aus § 5.3)
→ Keine Kampagne, kein FIFO, kein Eintrag in trades_open/trades_close
→ trades_raw markieren: processed=1
    (trades_fx wird bereits beim Import befüllt, nicht hier)
```

Fall 1: Gleiche Richtung (Positionsaufbau)

```
Bedingung: current_qty * new_qty > 0 ODER current_qty = 0
            (Produkt > 0 ⇔ gleiches Vorzeichen ⇔ gleiche Richtung;
             eine Summe würde nicht funktionieren: 100 + (-50) = 50 > 0)

Beispiel: Bestand = 100 AAPL, neuer Trade = BUY 50 AAPL
→ 100 × 50 > 0, Bestand wird 150
Bestand = -100 AAPL (short), neuer Trade = SELL 50 AAPL
→ (-100) × (-50) > 0, Short wird -150
Bestand = 100 AAPL, neuer Trade = SELL 50 AAPL
→ 100 × (-50) < 0 → kein Fall 1, sondern Fall 2a (Teilschließung)

Aktion: INSERT trades_open (status=OPEN)
Kampagne: TRADE_OPEN im Ledger
```

Fall 2c: Exakter Gegenabschluss

```
Bedingung: abs(open_tranche.qty) = abs(new_qty), entgegengesetzt

Beispiel: Bestand = 100 AAPL (1 Tranche), neuer Trade = SELL 100 AAPL
→ Position wird 0

Aktion: INSERT trades_close (status=Closed)
        trades_open markieren: processed=1
        Kampagne: TRADE_CLOSE_FULL → Status PENDING
```

Fall 2a: Teilabschluss



Bedingung: `abs(open_tranche.qty) > abs(new_qty)`, entgegengesetzt

Beispiel: Bestand = 100 AAPL (1 Tranche), neuer Trade = SELL 60 AAPL
→ 60 AAPL geschlossen, 40 AAPL bleiben offen

Aktion: `INSERT trades_close` (60 Stück, status=Closed)
alte Tranche in `trades_open` markieren: `processed=1`
`INSERT trades_open`: neue Zeile mit den 40 Reststück (status=SPLIT_REST),
netCash/proceeds/ibCommission anteilig (40/100)
Kampagne: `TRADE_CLOSE_PARTIAL` im Ledger

Schlüsselkonzept: Ratio-Skalierung

```
ratio = close_qty / open_qty # z.B. 60/100 = 0.6

# Die Open-Werte werden auf den geschlossenen Anteil skaliert:
netCash_open_anteil    = open_trade.netCash      * ratio
proceeds_open_anteil   = open_trade.proceeds     * ratio
commission_open_anteil = open_trade.ibCommission * ratio
# → ROI und P&L sind korrekt trotz Teilabschluss (Details: § 6.4)
```

Fall 2b: Neuer Trade größer als die Tranche

Bedingung: `abs(new_qty) > abs(open_tranche.qty)`, entgegengesetzt

Beispiel: Tranchen A = 100 AAPL, B = 50 AAPL, neuer Trade = SELL 120 AAPL
→ A komplett geschlossen, die übrigen 20 laufen weiter gegen B (dort Fall 2a)

Aktion: `INSERT trades_close` (Tranche komplett, status=Closed)
Tranche in `trades_open` markieren: `processed=1`
Restmenge des neuen Trades (anteilig netCash/proceeds/ibCommission)
→ nächste Tranche der Schleife
Kampagne: `TRADE_CLOSE_FULL` im Ledger je Tranche

Bei **Optionen** übernehmen alle drei Fälle (2a, 2b, 2c) den Status des schließenden Events: Verfall, Zuteilung oder Barausgleich stehen in `trades_close` als `Expired` / `Assigned` / `CashSettled`, im Ledger als `OPTION_EXPIRATION` / `OPTION_EXERCISE` / `CASH_SETTLEMENT` — auch wenn nur ein Teil der Tranche betroffen ist (z. B. 2 von 5 Kontrakten zugeteilt) oder ein Event mehrere Tranchen schließt. Bei einer Zuteilung wird zusätzlich die `linkID` für den folgenden Aktien-Trade gemerkt (§ 5.5).

Umkehrung (Reversal): Restmenge nach allen Tranchen



Bedingung: nach der Schleife über alle Tranchen ist `remaining_qty` $\neq 0$

Beispiel: Bestand = 100 AAPL (mehrere Tranchen), neuer Trade = SELL 250 AAPL
→ Alle Tranchen schließen (FIFO, Fälle 2b/2c), 150 AAPL werden Short

Aktion: Restmenge → INSERT `trades_open` (`status=REV_OPEN`)
Kampagne: `TRADE_REVERSAL_OPEN` im Ledger, gleiche Kampagne
(die Position war nie flat, daher keine neue Kampagne)

Fall SPLIT: Corporate Action (Aktiensplit)

Woher die Daten kommen: Aus dem Abschnitt *Kapitalmaßnahmen* der Flex Query (`<CorporateAction>`, Option *Detail*). `extract_corporateaction_for_db()` übernimmt jede Kapitalmaßnahme als Zeile in `trades_raw` (`source="split"`, `dateTime = reportDate`) und setzt `is_split=1` für die Typen `RS` (Reverse Split), `FS` (Forward Split) und `SS` (Split) — die Liste steht an einer Stelle in `src/utils/corporate_utils.py`. IBKR liefert einen Split als **Paar mit derselben `actionID`**: die alten Aktien gehen raus (negative Menge, Symbol meist mit Endung `.OLD`), die neuen kommen rein (positive Menge, neues Symbol). `sort_corporate_actions()` ordnet die Paare (alter Teil zuerst).

Verarbeitung (`process_ibkr_trades.py`, vor der Kampagnenlogik — Split-Zeilen lösen keinen PENDING/ACTIVE-Wechsel aus):

1. `_handle_split_trade()`: Zeilen je `actionID` puffern
erster Teil → warten, als verarbeitet markieren
zweiter Teil → Faktor = $|Menge\ neu| / |Menge\ alt|$, Symbol = neues Symbol,
ebenfalls als verarbeitet markieren
2. `process_stock_split()`: alle offenen Tranchen des Symbols (`trades_open`, `processed=0`)
`quantity` = `quantity` × Faktor
`tradePrice` = `tradePrice` / Faktor (Einstand gesamt bleibt gleich)
Menge 0 → Tranche geschlossen (`processed=1`), nicht als 0-Menge stehen lassen
Ledger → `STOCK_SPLIT` mit dem Datum des Splits, `inventory_delta` = neue - alte Menge,
kein Cash, kein P&L
3. Tracker: Bestand der Kampagne × Faktor, danach Flat-Check (Bestand 0 → PENDING)

Kein Cash, kein Gewinn/Verlust: Ein Split ändert nur Stückzahl und Preis je Aktie. Spätere Trades finden die Tranchen über das Symbol; ändert IBKR beim Split die `conid`, greift der Fallback in `get_open_trades()` nicht ein, weil das Symbol gleich bleibt.

Beispiele aus echten Daten (beide Reverse Splits):



Symbol	Datum	alt → neu	Faktor	Ledger <code>inventory_delta</code>
CYN	18.02.2025	4.152 → 27,68	≈ 0,00667 (1:150)	-4.124,32
APDN	14.03.2025	1.500 → 30	0,02 (1:50)	-1.470

Die neue Menge kann krumm werden (27,68 CYN-Aktien) — sie wird so übernommen, wie IBKR sie liefert.

Grenzen (gegen Code und Echtdaten geprüft):

- **Optionen auf die gesplittete Aktie** werden erkannt (`is_split=1`), aber offene Options-Tranchen werden nicht umgestellt: `process_stock_split()` sucht nach dem *neuen* Options-Symbol, die Tranche steht aber unter dem alten. Einmal mit offener Position vorgekommen (PAPER, TQQQ-Forward-Split 2:1 am 20.11.2025, Bull Put Spread 75/85). So steht das im XML: je Leg eine `CorporateAction` -Zeile mit gemeinsamer `actionID` , ohne `.OLD` -Zeilen; Symbol mit neuem Strike (37,5/42,5), `quantity` nur der **Zuwachs** (+1 beim Long, -1 beim Short), `conid` unverändert. Das Paar-Schema behandelt die zwei Legs dann als altes und neues Symbol (Faktor 1, also keine Umstellung). Der Verfall am 19.12. mit je 2 Kontrakten trifft über den `conid` -Rückfall die alte Tranche mit 1 Kontrakt. Für **Reverse Splits** mit offener Option liegen noch keine Daten vor.
- **Mehr als zwei Zeilen je `actionID`** (z. B. mehrere Options-Strikes in einer Kapitalmaßnahme) passen nicht ins Paar-Schema; ab der dritten Zeile beginnt ein neues Paar.
- **Andere Kapitalmaßnahmen** (Fusion, Spin-off, Symbolwechsel) bekommen `is_split=0` und würden wie normale Trades (Preis 0) durch die FIFO-Logik laufen — eine eigene Behandlung gibt es nicht. In den Echtdaten bisher nicht vorgekommen.

6.3 Kampagnen-Lebenszyklus: `inventory_tracker.py`

Ursprung: Die Kampagne wurde konzipiert, um die **Wheel-Strategie** (auch „Bonus-Strategie“) vollständig zu dokumentieren. Bei der Wheel-Strategie werden auf dasselbe Underlying zyklisch Cash-Secured Puts und Covered Calls verkauft — oft über Wochen oder Monate. Eine Kampagne bildet genau diesen Zyklus ab: vom ersten Put bis zur letzten Aktienlieferung oder Ausübung.

Die **10-Tage-Wartezeit** (PENDING-Phase) ist ebenfalls auf die Wheel-Strategie abgestimmt: Nach dem scheinbaren Ende eines Zyklus (Position = 0) kann innerhalb weniger Tage ein neuer Put auf dasselbe Underlying eröffnet werden. Die PENDING-Phase verhindert, dass dafür sofort eine neue Kampagne angelegt wird — stattdessen wird der Trade in die bestehende aufgenommen, solange das Zeitfenster noch offen ist. Die Frist ist per `.env` einstellbar (`MAX_GAP_DAYS` , Kalendertage, Standard 10, empfohlen 7–14; gelesen in `src/config.py` , ungültige Werte brechen ab). Sie gilt nur für künftige Verarbeitung — bestehende Kampagnen ändern sich erst nach einem Neuaufbau.

Konzept: Eine *Kampagne* fasst alle Trades zu einem Symbol in einem zusammenhängenden Zeitraum zusammen.



AKTIVE Kampagne:

- └ Symbol: AAPL
- └ Start: 2025-01-10
- └ Bestand: 150 Aktien + 2 Calls
- └ Status: ACTIVE
- └ Ledger: TRADE_OPEN (x3), TRADE_CLOSE_PARTIAL (x1)

Nach vollständigem Schließen:

- └ Status: PENDING (Wartezeit: 10 Tage)
 - Sicherheitsnetz: Falls IBKR nachträgliche Korrekturtrades sendet

Nach 10 Tagen ohne Aktivität:

- └ Status: CLOSED
 - └ closed_at: 2025-02-05
 - └ Finale P&L berechnet

Ledger-Aktionen (`LedgerAction` -Enum in `src/trade_enums.py`):

Seit 2026-05 werden **feingranulare Actions** geschrieben – statt generischem `TRADE_OPEN` / `TRADE_CLOSE_*` benennt die Action Instrument, Richtung und Abschlussart. Die Zuordnung erfolgt über `_open_action()` / `_close_action()` in `process_ibkr_trades.py`.

Gruppe	Actions	Auslöser
Aktien: Eröffnung	<code>STK_BUY</code> , <code>STK_SELL</code>	Fall 1 (Long- bzw. Short-Einstieg)
Aktien: Schließung	<code>STK_SELL_CLOSE</code> , <code>STK_BUY_CLOSE</code> , <code>STK_SELL_PARTIAL</code> , <code>STK_BUY_PARTIAL</code>	Fall 2c (voll) bzw. 2a (teilweise)
Optionen: Eröffnung	<code>OPT_PUT_SELL</code> , <code>OPT_PUT_BUY</code> , <code>OPT_CALL_SELL</code> , <code>OPT_CALL_BUY</code>	Fall 1
Optionen: Schließung	<code>OPT_PUT_BUY_CLOSE</code> , <code>OPT_PUT_SELL_CLOSE</code> , <code>OPT_CALL_BUY_CLOSE</code> , <code>OPT_CALL_SELL_CLOSE</code> + <code>*_PARTIAL</code> -Varianten	Fall 2c bzw. 2a
Options- Events	<code>OPTION_EXPIRATION</code> , <code>OPTION_EXERCISE</code> , <code>CASH_SETTLEMENT</code>	OptionEAE (Verfall / Ausübung+Assignment / Barausgleich)
Sonstiges	<code>TRADE_REVERSAL_OPEN</code> , <code>STOCK_SPLIT</code>	Fall 2b Remainder / CorporateAction

Hinweis: Die generischen Bezeichnungen `TRADE_OPEN`, `TRADE_CLOSE_FULL`, `TRADE_CLOSE_PARTIAL` in den Fallbeschreibungen oben (Abschnitt 6.2) stehen stellvertretend für die jeweilige feingranulare Action. Ältere Datenbestände können noch die generischen Actions enthalten; TradingJournal kennt beide Varianten, ein Neuaufbau per Summary-Re-Import schreibt die feingranularen.



6.4 Schema-Validierung: `open_trade.py` / `closed_trade.py`

Warum Schemas?

Ohne explizite Validierung können falsche Datentypen, fehlende Felder oder falsche Vorzeichen unbemerkt in die Datenbank gelangen und P&L-Berechnungen verfälschen.

`OpenTradeSchema.validate()` :

```
REQUIRED_FIELDS = ["quantity", "tradePrice", "netCash", "status", ...]
FIELD_TYPES = {"quantity": Decimal, "tradePrice": Decimal, ...}

# Status-Validierung:
assert status in [TradeStatus.OPEN, TradeStatus.REV_OPEN, TradeStatus.SPLIT_REST]
```

`to_closed_trade(open_trade, close_data, ratio)` (Fall 2b/2c) und `partial_close_new_open(...)` (Fall 2a):

Dies sind die **kritischsten Funktionen** im System. Sie erzeugen aus einer offenen Tranche und dem schließenden Trade einen abgeschlossenen Trade (`trades_close`). Vereinfacht für `to_closed_trade` — die offene Tranche wird hier immer vollständig geschlossen, skaliert wird nur der Anteil des schließenden Trades (`ratio`):

```
# Open-Seite (Tranche vollständig):
proceeds_open    = open_trade.proceeds
open_commission  = open_trade.ibCommission

# Close-Seite (nur der Anteil, der diese Tranche schließt):
proceeds_close   = close_data.proceeds    * ratio
close_commission = close_data.ibCommission * ratio

# P&L – BRUTTO, ohne Kommissionen:
profit          = proceeds_open + proceeds_close
profit_EUR      = proceeds_open * fxRate_open + proceeds_close * fxRate_close

# Kommissionen separat (IBKR: negativ):
ibCommission    = open_commission + close_commission
# Netto-P&L wird erst in der Anzeige gebildet: profit + ibCommission

# ROI in Prozent, Basis: eingesetztes Kapital (netCashOpen), Vorzeichen wie profit:
ROI = abs(profit / netCashOpen * 100)    # negativ, wenn profit < 0

# Haltezeit:
days_held, total_seconds, duration_text = calculate_hold_metrics(open_dt, close_dt)
```



Bei `partial_close_new_open` (Fall 2a) ist es umgekehrt: Der schließende Trade zählt vollständig, die Open-Seite (`netCash` , `proceeds` , `ibCommission`) wird mit dem geschlossenen Anteil skaliert, der Rest bleibt als `SPLIT_REST` offen.

Warum `proceeds` statt `netCash` ? `proceeds` ist der reine Handelsbetrag ohne Kommission und funktioniert für alle Abschlussarten — auch für Cash Settlement, bei dem `netCashClose` 0 ist, `proceeds` aber den Barausgleich enthält. Alle P&L-Felder (`trades_close.profit` , `campaign_ledger.realized_pnl`) sind deshalb **brutto**; die Kommission steht immer separat.

6.5 Optional: Anreicherung `enrich_greeks.py` (Delta, IV, IVR, IVP)

Die Flex Query liefert zu einer Options-Ausführung weder Griechen noch implizite Volatilität noch den Kurs des Basiswerts. Der Pipeline-Schritt `enrich` (nach `process` , nur wenn in `pipeline.toml` unter `steps` eingetragen) rekonstruiert sie **nachträglich** über die TWS-API (`ib_async`):

- 1 Basiswert-Kurs zur Ausführungsminute:** historische 1-Minuten-Kurse (`TRADES` , nur Handelszeit) für Aktie/ETF (`SMART`) bzw. Index (`SPX` / `SPXW` → SPX an CBOE, analog NDX, RUT, XSP, VIX).
- 2 IV und Delta der Option:** Black-Scholes (europäisch, ohne Dividende, Zins `GREEKS_RATE`) aus dem tatsächlichen Ausführungspreis — IV per Nullstellensuche, daraus das Delta. Restlaufzeit bis Verfall 16:00 New York; Ausführungszeiten gelten als `GREEKS_TRADE_TZ` .
- 3 IVR / IVP:** aus den IV-Tageswerten des Basiswerts (`OPTION_IMPLIED_VOLATILITY`) über das Jahr bis zum Handelstag; unter 20 Tageswerten bleibt es leer.

Berücksichtigt werden Options-Trades mit Ausführungspreis > 0, aber **nicht** Verfall, Zuteilung oder Barausgleich (Status `Expired` / `Assigned` / `CashSettled`). Je Lauf die neuesten noch nicht angereicherten Trades der letzten 14 Tage (`--seit` für Rückwirkendes, `--max` begrenzt je Ziel); Abfragen sind gedrosselt (Abfragegrenzen von IBKR).

Ergebnis in zwei Tabellen je Prefix, beide **abgeleitet** und deshalb nicht in `reset_IBKR_DB.py` — `enrich_greeks.py` legt sie selbst an und berechnet nach einem Neuaufbau einfach neu:

Tabelle	Inhalt
<code>trade_greeks</code>	je <code>tradeID</code> : Basiswert-Kurs, IV der Option, Delta, IV des Basiswerts, IVR, IVP, Zahl der IV-Tage, Hinweis bei fehlenden Daten (<code>error</code>), Zeitpunkt
<code>iv_history</code>	Zwischenspeicher der IV-Tageswerte je Basiswert und Tag (für IVR/IVP auch zu anderen Tagen, z. B. bei der Schließung)

Trades mit Hinweis werden nicht bei jedem Lauf erneut abgefragt, sondern nur mit `--fehler-wiederholen` .



Verbindung: nur lesend (`readOnly=True` , keine Order-Aufrufe) zu `TWS_HOST:TWS_PORT` mit eigener `TWS_CLIENT_ID` (Standard 630). Ist keine TWS / kein IB Gateway erreichbar, endet der Schritt mit einem Hinweis und Exit-Code 0 — der Rest der Pipeline läuft weiter, beim nächsten Lauf wird nachgeholt.

Daten-Berechtigung (Machbarkeitsprobe 2026-10-29, Papierkonto): historische Minutenkurse und IV-Historie für Index und Aktien ohne Zusatz-Abo verfügbar, auch mehr als zwei Jahre zurück. Live-Griechen (`modelGreeks`) würden dagegen ein API-Marktdaten-Abo brauchen und werden nicht genutzt.

Genauigkeit: Black-Scholes ist für amerikanische Aktienoptionen eine Näherung (Abweichung vor allem tief im Geld und vor Dividenden); SPX-Monatsoptionen werden am Morgen abgerechnet (AM-Settlement), gerechnet wird einheitlich bis 16:00. Für Journal-Werte wie „Delta 0,16“ genügt das.

7. Kernkonzepte

7.1 Warum FIFO?

IBKR-Steuerberichte verwenden FIFO (First In, First Out) für Lot-Matching — das System bildet diese Logik nach, damit P&L-Berechnungen mit offiziellen IBKR-Berichten übereinstimmen. Die FIFO-Queue in `trades_open` ist nach `datetime` aufsteigend sortiert: älteste Tranche wird zuerst geschlossen.

Einschränkung beim Automations-Trading (TAT): Der automatisierte Trader (TAT) verwaltet intern seine eigene Lot-Zuordnung und schließt Positionen nicht zwingend in FIFO-Reihenfolge — insbesondere wenn mehrere offene Tranchen denselben Strike haben. In diesen Fällen kann die FIFO-Zuordnung dieses Systems von der TAT-internen Zuordnung abweichen: P&L und Haltezeit einzelner Trades werden dann anders berechnet als TAT es intern ausweist, auch wenn das Gesamtergebnis der Kampagne übereinstimmt.

7.2 Warum `Decimal` statt `float` ?

```
# float-Problem:
0.1 + 0.2 == 0.3 # → False (0.30000000000000004)

# In der Finanzwelt ist das inakzeptabel:
# Summe aus 1000 Trades kann um Cent-Beträge abweichen

# Lösung: Python Decimal (exakte Dezimalarithmetik)
from decimal import Decimal
to_decimal("0.1") + to_decimal("0.2") == Decimal("0.3") # → True
```

`to_decimal()` aus `src/utils/div_utils.py` ist robust gegen verschiedene Eingabeformate:

- Strings mit Tausenderpunkten (`"1.234,56"`)
- Float (konvertiert über String, vermeidet Float-Fehler)



- None / leer → `Decimal(0)`

7.3 Multi-User / Multi-Environment

Das System verwaltet bis zu 5 Environments je User:

Environment	Datenbank (Default)	Zweck
<code>money</code>	<code>ibkr_money</code>	Echtes Kapital
<code>paper</code>	<code>ibkr_paper</code>	Papierkonto / Simulation
<code>test</code>	<code>ibkr_test</code>	Entwicklung und Tests
<code>autom_money</code>	<code>ibkr_autom_money</code>	Automatisiertes Echtgeld
<code>autom_paper</code>	<code>ibkr_autom_paper</code>	Automatisiertes Paper

DB-Namen können per `.env` überschrieben werden (`DB_NAME_MONEY` , `DB_NAME_PAPER` , ...).

Tabellen-Präfix (`TABLE_PREFIX` aus `.env`):

```
-- User "dein_user" mit prefix "dein_user_":
SELECT * FROM dein_user_trades_raw;
SELECT * FROM dein_user_trading_campaigns;

-- User "zweiter_user" mit prefix "zweiter_user_":
SELECT * FROM zweiter_user_trades_raw;
```

Dies erlaubt mehrere User in **einer MySQL-Datenbank** ohne Kollisionen.

7.4 Transaktionen und Atomizität

Jeder Trade wird innerhalb einer Datenbank-Transaktion verarbeitet:

```
with db.transaction():
    # Alle INSERTs/UPDATES für einen Trade
    open_id = repo.insert_open_trade(trade_dict)
    repo.link_to_campaign_new(campaign_id, ...)
    repo.mark_trade_processed("trades_raw", raw_id)
    # Bei Fehler: Rollback → keine halbfertigen Zustände
```

Ohne Transaktionen könnten bei einem Absturz inkonsistente Zustände entstehen (z.B. `trades_close` vorhanden, aber `trades_open` nicht als processed markiert).



8. Datenbankstruktur

8.1 Tabellen-Übersicht

```
trades_raw      ← Rohdaten (temporärer Puffer)
  |
  ▼
trades_open     ← Aktive FIFO-Tranchen
  |
  ├── trades_close ← Abgeschlossene Trades (P&L hier)
  |
  └── trading_campaigns ← Kampagne je Symbol
      |
      └── campaign_ledger ← Vollständiger Audit-Trail

symbols         ← Instrument-Referenzdata (conid, ISIN, ...)
trades_fx       ← FX-Trades (separat)
account_snapshots ← Kontostand-Snapshots (aus CSV)

trade_greeks    ← optional: Delta/IV/IVR/IVP je Options-Trade (enrich, abgeleitet, § 6.5)
iv_history      ← optional: IV-Tageswerte je Basiswert (enrich, Zwischenspeicher)
```

Dazu die Tabellen, die nur TradingJournal schreibt (Eingaben in der Oberfläche):

Tabelle	Inhalt (UI)	Angelegt von
<code>trade_planner</code> , <code>trade_planner_legs</code>	Log: Kommentar (Begründung, Exit-Plan, Notizen) je Eröffnung, per <code>campaign_id</code> + <code>opened_at</code> einer Eröffnung der Kampagne zugeordnet; die Zahlen liest <code>App\Support\CampaignOpenings</code> aus <code>campaign_ledger</code> . Die Zahlenspalten und <code>trade_planner_legs</code> gelten nur noch für Einträge ohne Kampagne	<code>reset_IBKR_DB.py</code> (wird beim Neuaufbau mit geleert)
<code>psychology_rulesets</code> , <code>psychology_rules</code>	Psychologie → Regelwerke	<code>tools/maintenance/setup_psychology_tables.py</code> (nicht durch <code>reset_IBKR_DB.py</code> berührt)
<code>psychology_entries</code> , <code>psychology_compliance_results</code>	Psychologie → Einträge , Einhaltung je Regel und Eintrag	wie oben



8.2 Schlüsseltabellen im Detail

Die folgenden Spaltenlisten sind auf die fachlich relevanten Felder gekürzt, aber **wörtlich aus** `TradingJournal-Weaver/sql/*.sql` **übernommen** (keine erfundenen/veralteten Spalten) — für die vollständige Liste inkl. Typen/Defaults ist `sql/*.sql` die Quelle der Wahrheit.

`trades_raw` — Rohdaten-Puffer

```
id, checksum,                -- Eindeutigkeit
environment,                 -- Routing (accountId selbst wird nicht
                             -- gespeichert,
                             -- nur beim Import ausgewertet)
symbol, conid, assetCategory, buySell, -- Instrument
quantity, tradePrice, netCash,      -- Mengen/Preise
ibOrderID, tradeID,            -- IBKR-Referenzen
opt_strategyID, opt_strategyName,   -- Strategie-Info
is_split, split_type,          -- Corporate-Action-Kennzeichnung
source,                       -- Ursprungs-Elementtyp: trade | optioneae | split
processed,                   -- Status (0/1)
dateTime                     -- Zeitstempel
```

`trades_open` — Aktive Tranchen

```
id, symID,
symbol, conid, assetCategory,
quantity, tradePrice, netCash,      -- Einstiegswerte
ibCommission, fxRateToBase,
status,                             -- OPEN | REV_OPEN | SPLIT_REST
opt_strategyID, opt_strategyName, linkID,
environment, dateTime,
processed                           -- 0=offen, 1=geschlossen (History-Zeile, siehe §
6.1)
```

Kein `campaign_id` in dieser Tabelle — die Verknüpfung zur Kampagne läuft über `campaign_ledger.source_table / source_id`, nicht über eine FK-Spalte hier.

`trades_close` — Abgeschlossene Trades



```

id, symID, symbol, conid,
-- Open-Seite:
tradeIDOpen, tradePriceOpen, dateTimeOpen, netCashOpen,
-- Close-Seite:
tradeIDClose, tradePriceClose, dateTimeClose, netCashClose,
quantity,                -- geschlossene Menge (eine Spalte, kein
Open/Close-Paar)
-- Ergebnis:
profit, profit_EUR, ROI,
daysHeld, durationStr,
ibCommission, ibCommission_EUR,
status,                  -- Closed | Expired | Assigned | CashSettled
closedBy,                -- z.B. SELL / EXPIRY / ASSIGNMENT
opt_strategyID, opt_strategyName, linkID

```

Ebenfalls **kein** `campaign_id` — gleiches Prinzip wie bei `trades_open`.

`trading_campaigns` — Kampagnen

```

id, name, symbol, strategy, assetCategory,
status,                -- ACTIVE | PENDING | CLOSED
start_date, end_date, closed_at,
environment

```

Keine gespeicherten Aggregate (`total_profit`, `total_commission`, `inventory`) — diese Werte werden zur Laufzeit ermittelt: Bestand aus `trades_open`, P&L-Stand aus dem jeweils letzten `cum_pnl` / `cum_commission` / `cum_premium`-Eintrag in `campaign_ledger` (§ 8.3).

`campaign_ledger` — Audit-Trail (ES-Lite)

```

id, campaign_id,
action,                -- LedgerAction: STK_BUY | OPT_PUT_SELL | ... (s.
6.3)
inventory_delta,       -- Bestandsänderung
netcash, proceeds, commission, -- Cashflows dieser Zeile
realized_pnl,          -- P&L dieses Events
cum_pnl,               -- kumulierter P&L der Kampagne bis hier
cum_commission,        -- kumulierte Kommissionen bis hier
cum_premium,           -- kumulierte Erlöse bis hier
source_table, source_id, -- Rückverweis (TRADES_OPEN / TRADES_CLOSE /
TRADES_RAW)
dateTime

```



Die drei `cum_*`-Spalten ermöglichen Zeitreise-Abfragen ohne vollständigen Replay:

```
-- P&L-Stand der Kampagne am 15. März:  
SELECT cum_pnl FROM dein_user_campaign_ledger  
WHERE campaign_id = 42 AND dateTime ≤ '2025-03-15'  
ORDER BY dateTime DESC, id DESC LIMIT 1;
```

`account_snapshots` — Kontostand-Snapshots (täglich)

```
id, snapshot_date, environment,  
nav_prior, nav_current, nav_change, twrr,      -- Nettovermögenswert  
starting_value, mtm_pnl, ending_value,        -- Veränderung des NAV  
cash_eur, cash_usd,                          -- Cash-Positionen (siehe Limitierung  
unten!)  
commissions_eur, forex_translation_pnl,  
realized_pnl_eur, unrealized_pnl_eur,  
mtd_pnl, ytd_pnl,  
imported_at
```

Befüllt **ausschließlich** von `import_csv_snapshots.py` (`activity_csv_parser.py` , `snapshot_csv` -Step, TWRR exakt von IBKR) aus dem **Kontoauszug (Activity Statement, CSV)**, den IBKR nur per Mail zustellt — eine Abfrage per Web-API gibt es dafür nicht, und die Flex Queries (Mail wie API) enthalten keine Snapshot-Sektionen. Bleibt die Mail an einem Tag aus, fehlt der Snapshot dieses Tages.

⚠ Limitierung: fest auf EUR + USD verdrahtet, nicht generisch.

Anders als bei Einzeltrades (`trades_raw.currency` ist freier Text, `trades_fx` speichert Basis- und Gegenwährung je Tausch aus dem Symbol) ist die Cash-Position im Portfolio-Snapshot **hartverdrahtet auf genau zwei Währungen**:

- Schema: `sql/account_snapshots.sql` hat feste Spalten `cash_eur` / `cash_usd` , keine generische Struktur.
- CSV-Import: `activity_csv_parser.py::_extract_cash()` ruft `find('Ending Cash', 'EUR')` und `find('Ending Cash', 'USD')` auf.

Hält das IBKR-Konto Cash in einer dritten Währung (z.B. GBP), wird dieser Betrag beim Snapshot-Import **stillschweigend nicht erfasst** — kein Fehler, keine Warnung, der Wert fehlt einfach in `account_snapshots` . Bewusste Vereinfachung für ein Konto, das nur in EUR (Basiswährung) und USD (Handelswährung für US-Optionen/Aktien) bewegt wird. Für weitere Cash-Währungen braucht es eine bewusste Erweiterung (neue Spalte(n) + Parser-Anpassung).



8.3 Running Balance — ES-Lite-Upgrade

Das `campaign_ledger` wurde um drei kumulierte Salden erweitert, die bei jedem INSERT automatisch berechnet und mitgeschrieben werden (`inventory_tracker.py` → `_get_running_totals()`):

Spalte	Berechnung	Nutzen
<code>cum_pnl</code>	SUM(realized_pnl) je Kampagne	historischer P&L-Stand
<code>cum_commission</code>	SUM(commission) je Kampagne	Gesamtkosten-Verlauf
<code>cum_premium</code>	SUM(proceeds) je Kampagne	Prämieinnahmen (Wheel-Tracking)

Bestehende Daten wurden per `dev/migrate_running_balance.py` nachbefüllt (Window-Functions, liegt seit einer Aufräum-Aktion in `dev/` statt `src/` — reine Einmal-Migrationsskripte, auf einer frischen Installation nie gebraucht):

```
python -m dev.migrate_running_balance          # alle Environments
python -m dev.migrate_running_balance money dein_user_      # einzelnes Target
```

9. Konfiguration & Multi-User



9.1 Konfigurationsdateien

Datei	Zweck	Versioniert?
<code>.env</code>	Secrets (DB-Passwort, API-Keys)	Nein (gitignore, siehe <code>.env.example</code>)
<code>users.toml</code>	Zentrale User↔Prefix-Registry (einzige Quelle für z.B. <code>dein_user</code> → <code>dein_user_</code>)	Nein (gitignore, siehe <code>users.toml.example</code>) — enthält die echten Benutzernamen
<code>config.py</code>	DB-Konfiguration, lädt Account-Mapping aus <code>accounts.toml</code> (<code>get_account_config()</code>)	Ja
<code>accounts.toml</code>	Account-Mapping (accountId-Suffix → User/Environment, ohne Prefix)	Nein (gitignore, siehe <code>accounts.toml.example</code>) — enthält reale Kontodaten
<code>pipeline.toml</code>	Pipeline-Schritte, aktive User dieser Maschine	Nein (gitignore, siehe <code>pipeline.toml.example</code>) — maschinenspezifisch: jede Maschine verarbeitet nur die dort tatsächlich per <code>reset_IBKR_DB.py</code> initialisierten User/Environments
<code>CLAUDE.md</code>	Entwickler-Dokumentation	Ja

9.2 `.env` –Struktur (Auszug)

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_USER=root
DB_PASS=...
DEFAULT_USER=dein_user
# Alternative/Override statt DEFAULT_USER: TABLE_PREFIX=dein_user_ direkt

IMAP_USERNAME=deine@email.at
IMAP_PASSWORD=...
IMAP_SERVER=imap.gmx.com

IBKR_FLEX_QUERIES=TOKEN1:111111,TOKEN1:222222
```

(vollständige Variablenliste inkl. `DB_NAME_*` –Overrides: siehe `.env.example`, § 4 oben für die Mail-/API-spezifischen Variablen im Detail)



9.3 DB-Engine-Pattern (src/config.py)

```
from src.config import get_engine

# Environment bestimmt Datenbank-Name (aus DB_MAP + .env):
engine = get_engine("money")
# → mysql+pymysql://user:pass@localhost:3306/ibkr_money
```

MySQL-Modus für Windows 11 Kompatibilität — wird bei jeder Verbindung gesetzt:

```
# Verhindert MySQL-Fehler bei NULL-Datumswerten (Win11 ist strenger als Win10)
SET SESSION sql_mode='NO_ZERO_IN_DATE,NO_ZERO_DATE,NO_ENGINE_SUBSTITUTION'
```

9.4 Zusammenspiel .env / accounts.toml / pipeline.toml

Alle drei sind gitignored und maschinen-/personenspezifisch (im Gegensatz zum versionierten `users.toml`), haben aber unterschiedliche Aufgaben und greifen an unterschiedlichen Stellen im Ablauf:

<code>.env</code>	→ WER bin ich (DB-Zugang, Mail-/API-Zugang, optional DEFAULT_USER)
<code>accounts.toml</code>	→ WEM gehört diese XML-Datei (accountId → User + Environment)
<code>pipeline.toml</code>	→ WAS läuft auf DIESER Maschine (welche User/Environments werden verarbeitet)

Ablauf beim Import einer neuen Flex-Query-XML:

1. `.env` liefert die DB-Zugangsdaten für die Verbindung selbst — unabhängig davon, welchem User/Environment die Datei am Ende zugeordnet wird.
2. `import_ibkr_trades.py` liest die `accountId` aus dem `<FlexStatement>` der XML und schlägt sie in `accounts.toml` nach (§ 5.1) → ergibt User (z.B. `dein_user`) + Environment (z.B. `money`). Der Tabellen-Prefix selbst steht nicht in `accounts.toml`, sondern wird erst zur Laufzeit aus `users.toml` aufgelöst (`get_account_config()`) — `accounts.toml` kennt also nur den User-Namen, nicht das Prefix.
3. Im Auto-Modus (kein CLI-Override) ist dieses Routing für den `import`-Step vollständig unabhängig von `pipeline.toml` — jede Datei landet dort, wohin ihre `accountId` zeigt, auch wenn dieser User/dieses Environment auf der aktuellen Maschine gar nicht in `pipeline.toml` eingetragen ist.
4. `pipeline.toml` wirkt erst beim `process`-Step (und bei `run_pipeline.py` generell für die Auswahl, welche UserxEnvironment-Kombinationen ein Lauf überhaupt anfasst, § 3.2): nur wer hier eingetragen ist, wird verarbeitet — Daten für einen User, dessen Tabellen auf dieser Maschine nie per `reset_IBKR_DB.py` angelegt wurden, blieben sonst mit „Tabelle nicht gefunden“ hängen.



Praktische Konsequenz: Ein neues IBKR-Konto anzulegen heißt **immer zwei Schritte**: `accounts.toml` (Routing: welcher User/welches Environment) **und**, falls es sich um einen komplett neuen User handelt, zusätzlich `users.toml` (Prefix) — sonst bricht `get_account_config()` mit einer Meldung ab (fehlender User in `users.toml`), statt ohne Prefix weiterzuarbeiten. `pipeline.toml` muss nur ergänzt werden, wenn diese Maschine den neuen User/das neue Environment auch tatsächlich lokal verarbeiten soll.

9.5 Konkretes Beispiel

Fiktives Setup: **Anna** nutzt die App allein, auf einem Rechner, mit einem Live-Konto und einem Paper-Konto. So sehen ihre drei Dateien aus:

```
# .env — WER bin ich (DB-Zugang)
DB_USER=root
DB_PASS=geheim123
DEFAULT_USER=anna
```

```
# users.toml (lokal) — Anna → Prefix
[anna]
prefix = "anna_"
```

```
# accounts.toml — welches IBKR-Konto gehört zu welchem User/Environment
[4321]
user = "anna"
environment = "money"

[8765]
user = "anna"
environment = "paper"
```

```
# pipeline.toml — was verarbeitet DIESER Rechner
[users.anna]
environments = ["money", "paper"]
```

Was passiert, wenn eine neue Flex-Query-XML von Annas Live-Konto ankommt (`accountId` in der XML z.B. `U11114321`, maskiert oder voll — nur die letzten 4 Ziffern zählen):

1. `import_ibkr_trades.py` öffnet die XML, liest `accountId = U11114321` aus dem `<FlexStatement>`-Tag.
2. `config.py` extrahiert die letzten 4 Ziffern (`4321`) und schlägt sie in `accounts.toml` nach → Treffer: `user = "anna", environment = "money"`.
3. Für den Tabellen-Prefix wird zusätzlich `users.toml` nachgeschlagen: `anna` → `anna_`.



4. Ergebnis: Die Trades aus dieser Datei landen in `ibkr_money.anna_trades_raw` — DB-Zugang dafür kommt aus `.env` (Schritt 1 dort war nur „wie verbinde ich mich“, nicht „wessen Daten sind das“).
5. Beim nächsten `run_pipeline.py` -Lauf (ohne `--users` / `--envs` -Filter) prüft `pipeline.toml` : ist `anna` / `money` dort eingetragen? Ja → wird im `process` -Step mitverarbeitet.

Jetzt zieht Annas Bruder Tom mit ein, eigenes Paper-Konto, dieselbe Maschine:

- `users.toml` bekommt einen neuen Block `[tom]` mit `prefix = "tom_"` (lokal — auf jeder Maschine, die Toms Daten verarbeitet, ebenso ergänzen).
- `accounts.toml` bekommt einen neuen Eintrag für Toms accountId-Suffix, z.B. `[2468] user = "tom" / environment = "paper"` (nur lokal, nicht geteilt).
- `pipeline.toml` bekommt `[users.tom] environments = ["paper"]` — **aber erst, nachdem** `python -m tools.maintenance.reset_IBKR_DB paper tom_` gelaufen ist. Vorher würde `run_pipeline.py` beim `process` -Step mit „Tabelle nicht gefunden“ abbrechen, weil `tom_trades_raw` noch gar nicht existiert.
- `.env` bleibt unverändert — DB-Zugangsdaten sind nicht pro Person, sondern pro Maschine.

Alle vier Dateien im Endzustand, nachdem Tom eingezogen ist:

```
# .env — unverändert
DB_USER=root
DB_PASS=geheim123
DEFAULT_USER=anna
```

```
# users.toml (lokal) — neuer Block [tom]
[anna]
prefix = "anna_"

[tom]
prefix = "tom_"
```



```
# accounts.toml – neuer Eintrag [2468]
[4321]
user = "anna"
environment = "money"

[8765]
user = "anna"
environment = "paper"

[2468]
user = "tom"
environment = "paper"
```

```
# pipeline.toml – neuer Block [users.tom]
[users.anna]
environments = ["money", "paper"]

[users.tom]
environments = ["paper"]
```

10. Fehlerbehandlung & Robustheit

10.1 Duplikat-Schutz

- **Checksum-basiert:** Verhindert Doppel-Import derselben XML
- **processed-Flag:** Verhindert Doppel-Processing desselben Trades
- **campaign_ledger-Atomizität:** Kein Ledger-Eintrag ohne dazugehörigen Trade

10.2 Logging

Jede Phase schreibt in `log/` mit Timestamp. Logger-Ebenen:

```
DEBUG    → detaillierte FIFO-Schritte
INFO     → Trade-Ergebnisse, Kampagnen-Status
WARNING  → fehlende Daten, unerwartete Fälle
ERROR    → Fehler mit Stacktrace (kein Absturz der Pipeline)
```



11. Ausblick — Erweiterungen & Verbesserungen

11.1 Kurzfristige Verbesserungen

A) Streaming-Processing statt Batch

Problem: Alle Trades werden in RAM geladen (DataFrame)
Risiko: Bei sehr großen Historien (10.000+ Trades) hoher Speicherbedarf
Lösung: Generator-basiertes Streaming – trades_raw zeilenweise verarbeiten
(yield-Pattern bereits in Teilen vorhanden)

B) Automatische Retry-Logik

Problem: Bei Datenbankfehler mid-Processing: unklar welche Trades schon verarbeitet
Lösung: Idempotente Operationen + Retry-Counter in trades_raw
(max_retries=3 → FAILED-Status nach Erschöpfung)

C) Validierungs-Report beim Import

Aktuell: Fehler landen im Log, keine strukturierte Auswertung
Verbesserung:
- Anzahl importierter / übersprungener / fehlerhafter Trades als JSON
- Webhook-Notification bei Fehlern (Slack, E-Mail)

11.2 Mittelfristige Erweiterungen

D) Broker-Agnostik: Weitere Datenquellen

Problem: System ist auf IBKR Flex Query fixiert
Potential: Andere Broker (Tastyworks, TD Ameritrade) über Adapter anbinden

Ansatz: BrokerAdapter-Interface
├ IBKRFlexAdapter (bestehend)
├ TastyworksAdapter (CSV-Format)
└ GenericCSVAdapter (konfigurierbar)

Der Parser-Orchestrator (flex_parser.py) ruft nur das Interface auf.

E) Real-Time-Import via IBKR TWS API



Aktuell: Täglicher Batch-Import via E-Mail
Potential: Live-Import via IBKR TWS API (Interactive Brokers API)
→ Trades werden sekundlich importiert
→ Dashboard zeigt Live-P&L

Anforderungen:

- ibapi Python-Paket
- Separater Service-Prozess (always-on)
- trades_raw als Puffer bleibt bestehen

F) Erweitertes Strategie-Tracking

Aktuell: Strategie-Name wird beim Import gesetzt und bleibt statisch
Potential:

- Strategie-Zustand über Laufzeit tracken (z.B. IC: welcher Flügel getroffen?)
- Automatisches Rollover-Erkennen (gleiche Strategie, neue Expiry)
- Greeks-Snapshot bei Eröffnung (via TWS API)

G) Steuer-Export

Potential: Automatischer Export als:

- FIFO-Steuerreport (Österreich/Deutschland)
- CSV für Steuerberater
- Vergleich mit IBKR Annual Tax Report

Basis: trades_close enthält alle nötigen Daten

H) Währungsdarstellung USD/EUR in der Weboberfläche

Aktuell: Weaver führt alle Beträge zusätzlich in EUR mit (*_EUR / *_eur-Spalten in trades_close und campaign_ledger, fxRateToBase in trades_raw / trades_open). Beide Währungen gleichzeitig zeigen aber nur die Detailseiten geschlossener Trades; Listen, Kampagnen, Spreads und Statistiken zeigen die Handelswährung.
Ziel: Globaler Umschalter USD/EUR (analog zum Environment-Schalter) oder EUR-Werte überall mit anzeigen.
Basis: Daten liegen vollständig vor – reine Erweiterung der Laravel-Oberfläche.

11.3 Langfristige Architektur-Ideen

I) Event-Sourcing statt FIFO-Matching on-the-fly



Aktuelles Modell:

trades_raw → FIFO-Matching → trades_open + trades_close

Event-Sourcing-Ansatz:

Alle Ereignisse (TradeOpened, TradeClosed, PositionSplit, ...) in einer "events"-Tabelle speichern.

Aktueller Zustand = Replay aller Events.

Vorteil: Zeitreise möglich (wie sah P&L am 15.03. aus?)

Kampagnen rückwirkend neu berechnen ohne Datenverlust

Nachteil: Komplexer, langsamer Read-Pfad

J) Plugin-System für Corporate Actions

Aktuell: Splits werden hardcoded behandelt

Potential: Jeder CorporateAction-Typ = eigener Handler (Plugin)

→ einfaches Erweitern ohne Kerncode zu ändern

```
class CorporateActionHandler(ABC):  
    @abstractmethod  
    def handle(self, action, repo, tracker): ...  
  
class SplitHandler(CorporateActionHandler): ...  
class DividendHandler(CorporateActionHandler): ...  
class MergerHandler(CorporateActionHandler): ...
```

K) Tagesabschluss-Snapshot

Potential: Tägliche Bestandsaufnahme als separater DB-Eintrag:

- Open P&L je Symbol (unrealisiert)
- Margin-Verbrauch
- Greeks-Summe (Delta, Theta, Vega)

Basis: Kombination aus trades_open + account_snapshots + TWS API



Anhang: Technologie-Stack

Komponente	Technologie	Version
Sprache	Python	3.14
Datenbank	MySQL	8.4
ORM/DB-Driver	SQLAlchemy + PyMySQL	2.0.x + 1.1.x
Daten-Manipulation	pandas	aktuelle
Dezimal-Arithmetik	Python <code>decimal</code>	stdlib
XML-Parsing	Python <code>xml.etree.ElementTree</code>	stdlib
Konfiguration	python-dotenv	aktuelle
Pipeline-Config	TOML	stdlib (Python 3.11+)
Tests	pytest	aktuelle

Anhang: Wichtige Dateipfade

```
TradingJournal-Weaver/  
├─ run_pipeline.py          ← Einstiegspunkt (orchestriert alle Phasen)  
├─ pipeline.toml           ← Pipeline-Konfiguration (aktive User/Environments,  
Steps) – maschinenspezifisch, gitignored  
├─ users.toml              ← Zentrale User⇒Prefix-Registry – lokal, gitignored  
                           (Vorlage: users.toml.example)  
├─ README.md               ← Projekt-Kurzbeschreibung  
├─ src/  
│   └─ get_flexquery_emails.py ← Phase 0: IMAP-Abruf → incoming_mail/ +  
Trade_CSV/incoming  
│   └─ get_flexquery_api.py   ← Phase 0 zweiter Kanal: Flex Query via Web-API →  
incoming_api/  
│   └─ import_ibkr_trades.py  ← Phase 1: XML/CSV-Import → trades_raw  
│   └─ process_ibkr_trades.py ← Phase 2: FIFO-Processing → Endtabellen  
│   └─ xml_append_summary.py  ← Phase 3: Summary-XML für DB-Rebuild  
│   └─ import_activity_statement.py ← CSV-Snapshot-Import (account_snapshots)  
│   └─ import_csv_snapshots.py ← Snapshot-Step der Pipeline (Activity-CSVs, TWRR  
exakt)  
│   └─ config.py             ← get_engine(), get_account_config(), DB_MAP,  
load_users(), TABLE_PREFIX
```



```
| |─ trade_enums.py          ← TradeStatus, AssetCategory, CampaignStatus
| |─ options_pnl.py         ← calculate_pnl()/get_position_summary(), von
flex_parser.py verwendet (Produktivcode, kein dev-Tool)
| |─ parsers/
| |   |─ flex_parser.py      ← XML-Orchestrator (process_flex_file)
| |   |─ day_trade_processor.py ← Element-Extraktion (Trades, EAE, CorpActions)
| |   |─ option_strategies.py ← Strategie-Erkennung (ibOrderID + 3s-Fallback)
| |   |─ inventory_tracker.py ← Kampagnen-Lifecycle + Running Balance
| |   |─ activity_csv_parser.py ← Activity-Statement-CSV-Parser
| |─ book_trades.py         ← BookTrade-Klassifizierung via OptionEAE (Top-Level,
nicht in parsers/)
| |─ trade_schemas/
| |   |─ base_schema.py      ← Abstrakte Basis (validate, REQUIRED_FIELDS)
| |   |─ open_trade.py       ← OpenTradeSchema + to_closed_trade() + ratio-
Skalierung
| |   |─ closed_trade.py     ← ClosedTradeSchema
| |─ db/
| |   |─ db_access.py        ← DBAccess (SQL) + TradeRepository (Domain-Queries)
| |   |─ utils/
| |       |─ div_utils.py    ← to_decimal(), compute_checksum(),
calculate_hold_metrics()
| |       |─ corporate_utils.py ← SPLIT_TYPES (RS, SS, FS)
| |       |─ path_utils.py   ← get_project_root()
|─ tools/
|   |─ maintenance/
|   |   |─ reset_IBKR_DB.py  ← DB-Reset (destruktiv! alle Tabellen inkl. trades_raw)
|   |   |─ setup_psychology_tables.py ← legt psychology_*-Tabellen an (nicht-
destruktiv)
|   |   |─ seed_psychology_rulesets.py ← Basisdaten für Psychology-Modul
|   |   |─ fix_symbol_mismatch.py / purge_stale_trades.py ← punktuelle Datenkorrekturen
|   |   |─ xml_summary_creator.py ← Summary-XML manuell erzeugen
|─ sql/
|   |─ trades_raw.sql
|   |─ trades_open.sql
|   |─ trades_close.sql
|   |─ trading_campaigns.sql
|   |─ campaign_ledger.sql   ← inkl. cum_pnl, cum_commission, cum_premium
|─ docs/
|   |─ handbuch/
|   |   |─ benutzer/        ← Überblick, IBKR einrichten, Installation für
Einsteiger, Alltag, Updates
|   |   |─ technik/         ← u. a. diese Dokumentation (10-...), Befehlsreferenz
(30-...), Installation
|─ dev/
|   |─                               ← Einmal-Migrationen, persönliche Tools – nie für einen
frischen Rechner nötig, absichtlich nicht in
```



```
| | sql/src/tools/maintenance (siehe Sparse-Checkout-
Hinweis
| | in 40-installation-windows.md für schlanke
Installationen)
| └─ migrate_running_balance.py ← Migration: cum_pnl/cum_commission/cum_premium
| └─ migrate_add_campaign_id_to_trade_planner.py
| └─ migrate_add_fees_open_to_trade_planner.py
| └─ migrate_add_opened_at_to_trade_planner.py ← Log-Eintrag → Eröffnung; setzt
bestehende Einträge auf die erste Eröffnung
| └─ migrate_add_open_types_to_psychology.py
| └─ migrate_backfill_assignment_linkid.py
| └─ compare_tat_ibkr.py ← manueller TAT-vs-IBKR-Abgleich
| └─ setup.py ← veraltet/ungenutzt (echter Weg: requirements.txt)
└─ tests/ ← pytest (DummyLogger, DummyDBAccess)
└─ input/ ← Test-XML/CSV-Dateien
└─ log/ ← Anwendungslogs (RotatingFileHandler)
```

Aufbau einer Flex Query

Dieses Kapitel zeigt, wie eine von IBKR gelieferte Flex Query (XML) aufgebaut ist und welche Felder der Import (`import_ibkr_trades.py` → `flex_parser.py` → `day_trade_processor.py`) daraus liest. Wie die Query im Kundenportal eingerichtet wird, beschreibt das Benutzerhandbuch im Kapitel IBKR einrichten.

1. Beispiel

Ein gekürztes Beispiel mit je einem Eintrag pro Abschnitt: ein Aktienkauf, ein verfallener Short Put und ein Reverse Split. **Alle Werte sind erfunden.** Echte Dateien enthalten pro Eintrag deutlich mehr Attribute (bei „Alle auswählen“: 84 je `Trade`, 44 je `OptionEAE`, 46 je `CorporateAction`) — hier stehen nur die, die der Import auswertet, plus einige zur Orientierung.

```
<FlexQueryResponse queryName="money_report" type="AF">
  <FlexStatements count="1">
    <FlexStatement accountId="U1234567" fromDate="10/03/2026" toDate="17/03/2026"
      period="LastNCalendarDays" whenGenerated="18/03/2026;061502">

      <Trades>
        <Trade accountId="U1234567" currency="USD" fxRateToBase="0.9174"
          assetCategory="STK" subCategory="COMMON" symbol="AAPL"
          description="APPLE INC" conid="265598" isin="US0378331005"
          tradeID="7012345678" dateTime="16/03/2026;153012"
          transactionType="ExchTrade" buySell="BUY" quantity="100"
          tradePrice="212.5" proceeds="-21250" ibCommission="-1"
          netCash="-21251" openCloseIndicator="0" fifoPnlRealized="0"
          ibOrderID="4401234567" orderReference="" multiplier="1"
          putCall="" strike="" expiry="" underlyingConid="" underlyingSymbol="" />
        </Trades>

        <OptionEAE>
          <OptionEAE accountId="U1234567" currency="USD" fxRateToBase="0.9174"
            assetCategory="OPT" subCategory="P" symbol="XYZ 260313P00040000"
            description="XYZ 13MAR26 40 P" conid="801234567"
            underlyingConid="45678901" underlyingSymbol="XYZ"
            multiplier="100" strike="40" expiry="13/03/2026" putCall="P"
            date="13/03/2026" transactionType="Expiration"
            quantity="1" tradePrice="0" proceeds="0" tradeID="7012340000" />
          </OptionEAE>

```



```
<CorporateAction accountId="U1234567" currency="USD" fxRateToBase="0.9174"
    assetCategory="STK" symbol="XYZ" conid="45678901"
    description="XYZ(US0000000000) SPLIT 1 FOR 25"
    reportDate="12/03/2026" dateTime="11/03/2026;202500"
    type="RS" quantity="-96" proceeds="0" actionID="150123456" />

</CorporateActions>

</FlexStatement>
</FlexStatements>
</FlexQueryResponse>
```

Aufbau in Kürze:

- Eine Datei enthält ein oder mehrere `FlexStatement`-Elemente — **eines je Konto**. `accountId` am `FlexStatement` bestimmt, welchem Benutzer und Environment die Daten zugeordnet werden (`accounts.toml`, Schlüssel = letzte vier Ziffern).
- Die drei Abschnitte entsprechen den drei in der Query aktivierten Sektionen. Das Container- und das Eintragsselement heißen bei den Options-Ereignissen beide `OptionEAE`.
- Datum mit Uhrzeit hat das Format `dd/MM/yyyy;HHmmss` (z. B. `16/03/2026;153012`), reine Datumsfelder `dd/MM/yyyy`. Ein anderes in der Query eingestelltes Format kann der Import nicht lesen (`parse_datetime_flex()` in `src/utils/div_utils.py`).
- Leere Attribute (`strike=""` bei Aktien) sind normal — IBKR schreibt jedes ausgewählte Feld, auch wenn es für den Eintrag keinen Wert hat.

2. Welche Felder der Import liest

Trade (`extract_trades_for_db`)

Gruppe	Felder
Identität	<code>tradeID</code> , <code>ibOrderID</code> , <code>orderReference</code> , <code>transactionType</code>
Instrument	<code>assetCategory</code> , <code>subCategory</code> , <code>symbol</code> , <code>description</code> , <code>conid</code> , <code>isin</code> , <code>currency</code>
Optionen	<code>putCall</code> , <code>strike</code> , <code>expiry</code> , <code>multiplier</code> , <code>underlyingConid</code> , <code>underlyingSymbol</code>
Ausführung	<code>dateTime</code> , <code>buySell</code> , <code>quantity</code> , <code>tradePrice</code> , <code>openCloseIndicator</code>
Geld	<code>proceeds</code> , <code>netCash</code> , <code>ibCommission</code> , <code>fxRateToBase</code> , <code>fifoPnlRealized</code>

`ibOrderID` ist doppelt wichtig: Er bündelt Teilausführungen derselben Order und erkennt Mehrbein-Strategien (Spreads, Iron Condors), die IBKR als einzelne Trades liefert. Reine Devisentausch-Trades (`assetCategory="CASH"`, `symbol="EUR.USD"`) werden in zwei Zeilen (Basis- und Gegenwährung) zerlegt.



OptionEAE (`extract_optioneae_for_db`)

`transactionType` (`Expiration` , `Exercise` , `Assignment` , `Cash Settlement`), `date` , `tradeID` , `conid` , `symbol` , `description` , `assetCategory` , `subCategory` , `putCall` , `strike` , `expiry` , `multiplier` , `underlyingConid` , `underlyingSymbol` , `quantity` , `proceeds` , `currency` , `fxRateToBase` , `isin` .

Diese Einträge beschreiben das *Ereignis*. Die zugehörige Buchung liefert IBKR zusätzlich als automatischen Trade im Abschnitt `Trades` ; `book_trades.py` verbindet beide über `tradeID` bzw. `conid` + Datum und überträgt den Status (*Expired*, *Assigned* — auch bei Ausübung —, *CashSettled*); bei einer Zuteilung verknüpft es außerdem Option und Aktien-Trade über eine gemeinsame `linkID` (siehe Architektur und Import-Pipeline, § 5.5).

CorporateAction (`extract_corporateaction_for_db`)

`type` (z. B. `RS` = Reverse Split, `FS` / `SS` = Split), `actionID` , `reportDate` , `description` , `symbol` , `conid` , `isin` , `assetCategory` , `subCategory` , `quantity` , `proceeds` , `currency` , `fxRateToBase` , `multiplier` , `underlyingConid` , `underlyingSymbol` .

Aus Splits passt die Verarbeitung Menge und Einstandspreis aller offenen Tranchen an (siehe Architektur und Import-Pipeline, Fall SPLIT).

3. Mail- und API-Query

Die per E-Mail zugestellte und die per Web-API abgerufene Query sind **gleich aufgebaut**, sofern sie mit denselben Sektionen eingerichtet sind. Der Import behandelt beide Ordner (`incoming_mail\` , `incoming_api\`) gleich; doppelt gelieferte Trades erkennt er an einer Prüfsumme (`tradeID` , `dateTime` , `symbol` , `quantity` , `tradePrice`).

Snapshot-Daten (Kontostand, Cash, Rendite) enthalten die Flex Queries nicht — sie kommen ausschließlich aus dem täglichen Kontoauszug (CSV per Mail), siehe Architektur und Import-Pipeline, § 8.2 `account_snapshots` .



Befehlsreferenz

Environments

Kürzel	Datenbank	Beschreibung
<code>money</code>	<code>ibkr_money</code>	Live-Trading
<code>paper</code>	<code>ibkr_paper</code>	Paper-Trading
<code>test</code>	<code>ibkr_test</code>	Test/Entwicklung
<code>autom_money</code>	<code>ibkr_autom_money</code>	Automatisiertes Live-Trading
<code>autom_paper</code>	<code>ibkr_autom_paper</code>	Automatisiertes Paper-Trading

Table-Prefix

Jeder User hat einen eigenen Tabellen-Prefix, zentral definiert in `users.toml` (einzige Quelle — `config.py` und `pipeline.toml` lösen den Prefix von dort auf):

User	Prefix
<code>dein_user</code>	<code>dein_user_</code>
<code>zweiter_user</code>	<code>zweiter_user_</code>

Beispiel: Prefix `dein_user_` → Tabellen `dein_user_trades_raw`, `dein_user_trades_open`, ...

Für manuelle Einzelaufrufe ohne CLI-Prefix-Argument: `.env` → `DEFAULT_USER=<name>` (Prefix wird aus `users.toml` aufgelöst) oder direkt `TABLE_PREFIX=<prefix>` (Override, gewinnt falls beides gesetzt ist).

Account-Zuordnung (Routing)

Die Zuordnung XML/CSV → User + Environment steht in `accounts.toml` (nicht versioniert, wie `pipeline.toml` — Vorlage: `accounts.toml.example`), aufgelöst über `src/config.py` → `get_account_config()`. Key sind die **letzten 4 Ziffern** der IBKR-accountId — funktioniert daher auch



mit maskierten IDs (`U***1234` und `U12341234` ergeben denselben Key).

```
[1234]
user = "dein_user"
environment = "money"
```

Der Importer liest die accountId aus dem ersten `<FlexStatement>` der XML. Unbekannte accountIds werden übersprungen (Warnung im Log). Neue Konten müssen in `accounts.toml` ergänzt werden.

Kein Dateinamen-Filter mehr (Normalbetrieb): Der Importer verarbeitet alle `*.xml`-Dateien in `incoming_mail\` und `incoming_api\` gleichwertig — die Quelle (Mail vs. Flex-Query-API) ist irrelevant, Checksum-Dedup (tradeID+dateTime+symbol+qty+price) verhindert doppelte Trades, falls dieselben Daten über beide Kanäle ankommen. Einzige Ausnahme: der dedizierte **TEST-Modus** verlangt weiterhin das Muster `flex.*.xml` (siehe TEST-Environment).

Verzeichnisstruktur (XML-Dateien)

Gezeigt mit den Standardordnern unter Windows. Die Basisordner lassen sich in der `.env` ändern (`TRADE_XML_DIR` , `TRADE_CSV_DIR` ; unter Linux/macOS standardmäßig `~/Trade_XML` und `~/Trade_CSV`), die Unterordner bleiben gleich.

```
C:\Trade_XML\
  incoming_mail\           ← eingehende XML-Dateien per E-Mail (accountId-Routing)
  incoming_api\           ← eingehende XML-Dateien per Flex-Query-Web-API
(accountId-Routing)
  <env>\
    <user>\
      processed\          ← erfolgreich importierte Dateien (aus beiden Quellen)
      deleted\            ← Dateien ohne neue Trades
      summary_<prefix><env>.xml ← zusammengefasste Summary
  test\
    <user>\
      incoming\           ← manuell abgelegte Test-XMLs (flex.*.xml), kein
accountId-Routing
      processed\
      deleted\
C:\Trade_CSV\
  incoming\               ← eingehende Activity-CSVs (nur per Mail, kein API-
Äquivalent)
  <user>\
    processed\           ← erfolgreich importierte CSVs
```



Der `test\<user>\` -Ordner existiert nur für User mit einem `[[test]]` -Eintrag in `pipeline.toml` (siehe TEST-Environment).

TEST-Environment — manuelle XMLs automatisch erkennen

Für TEST gibt es kein echtes IBKR-Konto, also auch keinen `accounts.toml` -Eintrag — Environment und Prefix werden stattdessen durch das **Verzeichnis** bestimmt, nicht durch die `accountId` in der XML. Kein Umbenennen der `accountId`, kein manuelles Patchen der XML nötig.

Konfiguration in `pipeline.toml` : Ein `[[test]]` -Eintrag pro User — kein `enabled` -Schalter, ein vorhandener Eintrag ist automatisch aktiv:

```
[[test]]
user = "dein_user"

[[test]]
user = "zweiter_user"
```

Zum Deaktivieren einfach den Eintrag aus der TOML entfernen/auskommentieren.

Workflow:

```
# 1. Datei muss dem flex.*.xml-Muster entsprechen (nur im TEST-Modus noch Pflicht)
Rename-Item C:\Trade_XML\incoming_mail\meine_datei.xml flex.meine_datei.xml

# 2. In den passenden User-Ordner legen — kein incoming_mail/incoming_api der
Hauptroute!
Move-Item flex.meine_datei.xml C:\Trade_XML\test\dein_user\incoming\
```

```
# 3. Ganz normal laufen lassen — TEST wird automatisch mit erfasst
python run_pipeline.py
# oder einzeln:
python -m src.import_ibkr_trades
python -m src.process_ibkr_trades
```

`import_ibkr_trades.py` scannt bei jedem Auto-Lauf zusätzlich zum normalen `accountId`-Routing alle `[[test]]` -Ordner (`C:\Trade_XML\test\<benutzer>\incoming\`) und importiert sie direkt mit dem konfigurierten Prefix in die TEST-DB. `process_ibkr_trades.py` verarbeitet TEST im Auto-Modus automatisch mit, da es ebenfalls `load_active_targets()` aus `pipeline.toml` liest.



Kein Summary für TEST: `xml_append_summary.py` lässt TEST im Auto-Modus bewusst aus — Testdaten brauchen kein Summary-Replay, sie lassen sich jederzeit erneut importieren. Ein `summary_<prefix>test.xml` entsteht daher nicht automatisch (nur bei explizitem CLI-Aufruf `python -m src.xml_append_summary test dein_user_`).

Alternative — Einzeldatei manuell testen, ohne `pipeline.toml`-Eintrag:

```
# Datei liegt in C:\Trade_XML\incoming_mail\ (Hauptordner), Prefix explizit angeben
python -m src.import_ibkr_trades test dein_user_
python -m src.process_ibkr_trades test dein_user_
```

Ohne Prefix-Argument wird der erste `[[test]]`-Eintrag aus `pipeline.toml` verwendet, sonst `TABLE_PREFIX` aus `.env`. Dieser Modus scannt den **Hauptordner** `incoming_mail\` (nicht `test\<user>\incoming\`, nicht `incoming_api\`) und ignoriert die accountId komplett.

Neu testen mit derselben Datei: Der Checksum-Schutz verhindert Doppel-Import (0 Trades importiert ist dann erwartetes Verhalten, kein Fehler). Für einen sauberen Neustart die TEST-DB des Prefixes zurücksetzen (⚠️ löscht alle bisherigen TEST-Daten dieses Prefixes):

```
python -m tools.maintenance.reset_IBKR_DB test dein_user_
```

run_pipeline.py — Vollständiger Pipeline-Runner

Konfiguration in `pipeline.toml`. Führt alle Steps automatisch aus.

```
# Alles ausführen (laut pipeline.toml)
python run_pipeline.py

# Nur bestimmte User
python run_pipeline.py --users mike
python run_pipeline.py --users dein_user zweiter_user

# Nur bestimmte Environments
python run_pipeline.py --envs money
python run_pipeline.py --envs money paper

# Nur bestimmte Steps
python run_pipeline.py --steps fetch_emails
python run_pipeline.py --steps import process
python run_pipeline.py --steps fetch_emails import process summary
```



```
# Kombinationen
python run_pipeline.py --users mike --envs money
python run_pipeline.py --users mike --envs money --steps import process

# Import aus summary.xml statt einzelner XML-Dateien
python run_pipeline.py --steps import --from-summary
python run_pipeline.py --steps import --from-summary --users mike
python run_pipeline.py --steps import --from-summary --envs autom_paper

# DB-Reset (destruktiv – Sicherheitsabfrage erscheint)
python run_pipeline.py --steps reset --users mike
python run_pipeline.py --steps reset --users mike --envs money
python run_pipeline.py --steps reset --force          # ohne Rückfrage

# Vorschau ohne Ausführung (zeigt alle Steps und Befehle)
python run_pipeline.py --dry-run
python run_pipeline.py --users mike --dry-run

# Andere Konfigurationsdatei verwenden
python run_pipeline.py --config meine_pipeline.toml
```

Verfügbare Steps:

Step	Modul	Läuft pro
fetch_emails	src.get_flexquery_emails	Einmal gesamt
fetch_api	src.get_flexquery_api	Einmal gesamt
snapshot_csv	src.import_csv_snapshots	Einmal gesamt (intern pipeline.toml)
import	src.import_ibkr_trades	Ohne <code>--users</code> / <code>--envs</code> : einmal gesamt. Mit Filter: pro passendem User x Environment
process	src.process_ibkr_trades	Einmal gesamt (intern pipeline.toml)
enrich	src.enrich_greeks	Einmal gesamt (intern pipeline.toml), optional — braucht TWS
summary	src.xml_append_summary	Einmal gesamt (intern pipeline.toml)
reset	tools.maintenance.reset_IBKR_DB	User x Environment ⚠ destruktiv



`import_ibkr_trades.py` unterstützt ein CLI-Override `<env> <prefix>` (analog `process_ibkr_trades.py`) — nur Dateien dieses einen Ziels werden importiert, alle anderen bleiben unangetastet liegen. `run_pipeline.py` nutzt das automatisch: ohne `--users / --envs` läuft `import` einmal global (effizient, accountId routet intern), mit gesetztem Filter läuft es pro passendem User x Environment.

`account_snapshots` kommt ausschließlich aus dem Kontoauszug (Activity Statement, CSV per Mail) über `snapshot_csv`. Den Kontoauszug gibt es nicht per Web-API, und die Flex Queries (Mail wie API) enthalten keine Snapshot-Sektionen (EquitySummary, CashReport, MTM).

get_flexquery_emails.py — E-Mails abrufen

Lädt IBKR Flex-Query-XML- und Activity-CSV-Anhänge aus dem Mail-Postfach und speichert sie in `C:\Trade_XML\incoming_mail\` bzw. `C:\Trade_CSV\incoming\`. Routing (welcher User / welches Environment) erfolgt erst beim Import per accountId.

```
python -m src.get_flexquery_emails
```

Keine Argumente. Relevante `.env`-Variablen:

Variable	Bedeutung
<code>IMAP_USERNAME</code>	Mail-Adresse
<code>IMAP_PASSWORD</code>	Mail-Passwort
<code>IMAP_SERVER</code>	IMAP-Server (Standard: <code>imap.gmx.com</code>)
<code>IBKR_ACTIVITY_SUBJECT</code>	Betreff Activity Statement EN (Standard: <code>Activity Statement</code>)
<code>IBKR_ACTIVITY_SUBJECT_DE</code>	Betreff Activity Statement DE (Standard: <code>Kontoauszug</code>)

get_flexquery_api.py — Flex Query per Web-API abrufen

Holt Flex-Query-XMLs direkt über die IBKR-Web-API, **parallel** zur Mail-Pipeline (`get_flexquery_emails.py`) — beide Kanäle sind gleichwertig aktiv. Schreibt nach `C:\Trade_XML\incoming_api\` (getrennt von `incoming_mail\`, damit beide Quellen unabhängig nachvollziehbar bleiben). Der `import`-Step scannt **beide** Ordner und importiert daraus `trades_raw` — die Query liefert dieselben Trades wie die Mail-XML, Checksum-Dedup verhindert doppelte Einträge.



Mit denselben Sektionen eingerichtet wie die Mail-Query (Trades, Option Exercises/Assignments/Expirations, Corporate Actions) liefert die API-Query auch Assignment-/Expiry-Events und Splits — vollwertiger zweiter Kanal.

Timing: Der Report wird bei Anfrage **on-demand generiert**, es gibt kein festes Tages-Zeitfenster. Voraussetzung ist nur, dass der Börsentag abgeschlossen ist — die Daten des Vortags stehen in der Regel schon am frühen Morgen bereit. Die Retry-Logik (`RETRY_DELAY` / `MAX_RETRIES`) fängt die kurze „noch in Verarbeitung“-Phase ab — typischerweise wenige Sekunden, bis zu ~30s bei Accounts mit geringerer IBKR-Priorität (z.B. Paper-Konten).

```
python -m src.get_flexquery_api
```

Variable	Bedeutung
<code>IBKR_FLEX_QUERIES</code>	Kommagetrennte <code>token:query_id</code> -Paare (siehe Docstring im Script)

Ohne `IBKR_FLEX_QUERIES` in `.env` ist der Schritt ein No-Op (kein Fehler, Exit-Code 0) — kann also gefahrlos im Standard-Pipeline-Lauf aktiv bleiben, auch wenn (noch) nicht für alle Accounts ein Token/Query konfiguriert ist.

`import_csv_snapshots.py` — `account_snapshots`

Importieren Kontostands-Snapshots in `account_snapshots` :

- `import_csv_snapshots` — aus den Activity-CSVs (Kontoauszug per Mail, **einzige Quelle**, TWRR exakt); Teil der Standard-Pipeline (`snapshot_csv`)

```
# Auto-Modus: alle aktiven Targets aus pipeline.toml
python -m src.import_csv_snapshots
```

Achtung: Cash-Positionen werden fest auf EUR + USD verdrahtet erfasst (`cash_eur` / `cash_usd` in `sql/account_snapshots.sql` , der CSV-Parser sucht explizit nach diesen zwei Währungen). Cash in einer dritten Währung wird stillschweigend nicht importiert. Details + Begründung: [docs/handbuch/technik/10-architektur-und-pipeline.md § 8.2 account_snapshots](#) .



import_ibkr_trades.py — XML → trades_raw

Scannt `C:\Trade_XML\incoming_mail\` und `C:\Trade_XML\incoming_api\` nach `*.xml` -Dateien (kein Dateinamen-Filter im Normalbetrieb). Routing erfolgt automatisch per `accountId` aus der XML → `accounts.toml` (via `src/config.py::get_account_config()`) → Environment + User + Prefix. Verarbeitete Dateien landen in `C:\Trade_XML\<environment>\<benutzer>\processed\` (aus beiden Quellordnern gleichermaßen).

```
# Auto-Modus: scannt beide incoming-Ordner, Routing per accountId (kein Argument nötig)
python -m src.import_ibkr_trades

# CLI-Override: nur ein Environment/Prefix importieren, alles andere bleibt liegen
python -m src.import_ibkr_trades money dein_user_
python -m src.import_ibkr_trades paper dein_user_

# Aus summary.xml importieren (explizites Environment + Prefix erforderlich)
python -m src.import_ibkr_trades from_summary money dein_user_
python -m src.import_ibkr_trades from_summary money zweiter_user_
python -m src.import_ibkr_trades from_summary autom_paper dein_user_
```

Argumente:

Position	Wert	Pflicht
1	Environment (CLI-Override) oder <code>from_summary</code> oder <code>test</code>	Nein — ohne Argument: Auto-Modus
2	Table-Prefix (z.B. <code>dein_user_</code>)	Bei CLI-Override/ <code>from_summary</code> : Ja. Sonst optional/Standard aus <code>.env</code>
3	<code>from_summary</code>	Nur im <code>from_summary</code> -Modus, an letzter Position

Bei CLI-Override (`<env> <prefix>`) werden Dateien anderer Environments/Prefixe in beiden Ordnern übersprungen und bleiben liegen — für den nächsten Lauf ohne Filter oder mit anderem Ziel.

process_ibkr_trades.py — trades_raw → trades_open / trades_close

Verarbeitet unverarbeitete Einträge aus `trades_raw` per FIFO-Matching. Schreibt Ergebnisse in `trades_open`, `trades_close`, `trading_campaigns`, `campaign_ledger`.



```
# Auto-Modus: alle aktiven Targets aus pipeline.toml (kein Argument nötig)
python -m src.process_ibkr_trades

# CLI-Override: explizites Environment (Überschreibt pipeline.toml)
python -m src.process_ibkr_trades money
python -m src.process_ibkr_trades money dein_user_
python -m src.process_ibkr_trades money zweiter_user_

# Trotz Trades in falscher Reihenfolge verarbeiten (siehe unten)
python -m src.process_ibkr_trades money dein_user_ --reihenfolge-ignorieren
```

Argumente (CLI-Override):

Position	Wert	Pflicht
1	Environment	Ja (für CLI-Override)
2	Table-Prefix	Nein (Standard aus <code>.env</code>)
beliebig	<code>--reihenfolge-ignorieren</code>	Nein

Reihenfolge-Prüfung: Liegt ein unverarbeiteter Trade zeitlich vor dem zuletzt verarbeiteten desselben Environments, wird dieses Environment nicht verarbeitet (Exit-Code 1, die übrigen laufen weiter). Ausgenommen: Verfall/Zuteilung/Barausgleich von Optionen. Hintergrund: Architektur und Import-Pipeline, § 6.1.

enrich_greeks.py — Delta, IV, IVR, IVP über die TWS-API (optional)

Rekonstruiert für Options-Trades den Basiswert-Kurs zur Ausführungsminute, IV und Delta der Option sowie IVR/IVP des Basiswerts und schreibt sie nach `trade_greeks` (Details: Architektur und Import-Pipeline, § 6.5). Braucht eine laufende TWS bzw. IB Gateway mit aktivierter API; ohne sie wird der Schritt übersprungen (Exit-Code 0). Verbindung nur lesend.



```
# Auto-Modus: alle Ziele aus pipeline.toml, neue Options-Trades der letzten 14 Tage
python -m src.enrich_greeks

# Einzelnes Ziel
python -m src.enrich_greeks paper dein_user_

# Rückwirkend, in Etappen (je Lauf höchstens 300 Trades je Ziel, die neuesten zuerst)
python -m src.enrich_greeks --seit 2025-01-01 --max 300

# Trades mit gespeichertem Hinweis (z. B. fehlende Daten) erneut versuchen
python -m src.enrich_greeks --fehler-wiederholen
```

Argumente:

Argument	Wert	Pflicht
1/2	Environment / Table-Prefix (CLI-Override)	Nein
<code>--seit</code>	Datum <code>JJJJ-MM-TT</code> (Standard: vor 14 Tagen)	Nein
<code>--max</code>	höchstens so viele Trades je Ziel (Standard 200)	Nein
<code>--fehler-wiederholen</code>	Trades mit Hinweis erneut berechnen	Nein

In der Pipeline als Step `enrich` nach `process` (in `pipeline.toml` unter `steps` eintragen).

xml_append_summary.py — Einzel-XMLs zu summary.xml zusammenfassen

Liest neue `*.xml` -Dateien aus `<env>/<user>/processed/` (Mail- und API-Dateien gleichermaßen, kein Dateinamen-Filter) und aktualisiert `summary_<prefix><env>.xml`. Keine Duplikate (fromDate eindeutig), leere FlexStatements werden übersprungen, aufsteigende Reihenfolge, Performance-Cache (`summary_<prefix><env>.cache.json` : bereits verarbeitete Dateinamen). Enthält nur Flex-Query-Daten, keine Kontoauszüge (CSV) — `account_snapshots` lässt sich daraus nicht wiederherstellen.



```
# Auto-Modus: alle aktiven Targets aus pipeline.toml (kein Argument nötig)
python -m src.xml_append_summary

# CLI-Override: explizites Environment + Prefix (Überschreibt pipeline.toml)
python -m src.xml_append_summary money dein_user_
python -m src.xml_append_summary paper dein_user_
python -m src.xml_append_summary autom_money dein_user_
python -m src.xml_append_summary money zweiter_user_
```

Argumente (CLI-Override):

Position	Wert	Pflicht
1	Environment	Ja (für CLI-Override)
2	Table-Prefix	Ja (für CLI-Override)

reset_IBKR_DB.py — Datenbank initialisieren

Legt die Datenbank und alle Tabellen neu an (aus `sql/` -Schemas). **Achtung: Löscht alle bestehenden Daten!** Nur beim ersten Setup oder nach Reset verwenden. Das Script liegt in `tools/maintenance/` (nicht in `src/`).

```
# Pflichtargument: Environment; optional: Prefix
python -m tools.maintenance.reset_IBKR_DB money dein_user_
python -m tools.maintenance.reset_IBKR_DB paper dein_user_
python -m tools.maintenance.reset_IBKR_DB money zweiter_user_
python -m tools.maintenance.reset_IBKR_DB autom_money dein_user_
python -m tools.maintenance.reset_IBKR_DB autom_paper dein_user_
```

Hinweis: Neue Zusatz-Tabellen werden bewusst **nicht** über `reset_IBKR_DB` angelegt, sondern über separate Setup-Scripts in `tools/maintenance/` — z.B. `setup_psychology_tables.py` (legt die `psychology_*`-Tabellen an, ohne bestehende Tabellen zu löschen).

Argumente:

Position	Wert	Pflicht
1	Environment	Ja
2	Table-Prefix	Nein (Standard aus <code>.env</code>)



db-backup.ps1 — Datenbanken sichern und wiederherstellen

PowerShell-Skript im Ordner `TradingJournal-Weaver`. Sichert alle Datenbanken (die fünf Environments aus `DB_NAME_*` der Weaver- `.env` plus `DB_DATABASE` aus der TradingJournal- `.env`, Standard `ibkr_laravel`); Datenbanken, die es auf dem Server nicht gibt, werden übersprungen.

```
.\db-backup.ps1                                # Backup → database\db\  
<db>_backup_JJJJMMTT_HHMM.sql  
.\db-backup.ps1 -UsbLabel "NAME"              # zusätzlich auf USB-Laufwerk  
(Datenträgerbezeichnung) kopieren  
.\db-backup.ps1 -Restore -DryRun              # zeigt die je Datenbank neueste  
Sicherung  
.\db-backup.ps1 -Restore                      # einspielen, Rückfrage je Datenbank  
.\db-backup.ps1 -MysqlBin "C:\xampp\mysql\bin" # Ordner mit mysqldump.exe/mysql.exe  
vorgeben
```

Parameter	Bedeutung
<code>-Restore</code>	statt sichern: neueste Sicherung je Datenbank einspielen
<code>-DryRun</code>	nur mit <code>-Restore</code> : nichts einspielen, nur anzeigen
<code>-UsbLabel</code>	Datenträgerbezeichnung eines USB-Laufwerks; Backup kopiert nach <code><Laufwerk>:\db</code> , Restore sucht dort mit
<code>-BackupDir</code>	anderer Sicherungsordner (Standard <code>database\db</code> , gitignored)
<code>-MysqlBin</code>	Ordner mit <code>mysqldump.exe</code> / <code>mysql.exe</code> ; sonst PATH, <code>C:\xampp\mysql\bin</code> , <code>C:\Program Files\MySQL\MySQL Server *\bin</code>

- Zugangsdaten (`DB_USER`, `DB_PASS`, `DB_HOST`, `DB_PORT`) kommen aus der Weaver- `.env` und werden nur in eine temporäre MySQL-Optionsdatei geschrieben (nicht auf die Kommandozeile), die danach gelöscht wird.
- Dumps entstehen über `--result-file`, nicht über PowerShells `>` (das in Windows PowerShell 5.1 UTF-16 schreibt und den Dump unbrauchbar macht); das Einspielen liest die Datei unverändert über `Start-Process -RedirectStandardInput`.
- Vor dem Überschreiben einer Datenbank sichert `-Restore` deren aktuellen Stand als `<db>_prerestore_JJJJMMTT_HHMMSS.sql`; diese Dateien zählen bei der Suche nach der neuesten Sicherung nicht mit.



Typischer Ablauf

Erster Setup (einmalig)

```
python -m tools.maintenance.reset_IBKR_DB money dein_user_  
python -m tools.maintenance.reset_IBKR_DB paper dein_user_  
python -m tools.maintenance.reset_IBKR_DB money zweiter_user_
```

Täglicher Lauf (automatisch, alle User/Envs laut pipeline.toml)

```
python run_pipeline.py
```

Täglicher Lauf (manuell, Schritt für Schritt)

```
python -m src.get_flexquery_emails      # Mails holen → incoming_mail/  
python -m src.get_flexquery_api         # Flex-Query-API holen → incoming_api/  
python -m src.import_ibkr_trades        # XML importieren (auto-routing per accountId,  
beide Ordner)  
python -m src.process_ibkr_trades       # FIFO-Verarbeitung (alle Targets)  
python -m src.xml_append_summary        # Summary aktualisieren (alle Targets)
```

Nur einen User neu verarbeiten

```
python run_pipeline.py --users mike --steps process
```

Nur Emails holen und importieren

```
python run_pipeline.py --steps fetch_emails import
```

Nur ein Environment importieren (CLI-Override, seit run_pipeline-Update)

```
python run_pipeline.py --steps import --envs money  
# entspricht intern: python -m src.import_ibkr_trades money dein_user_  
#                       python -m src.import_ibkr_trades money zweiter_user_
```

Einzelnen Account aus summary.xml neu importieren

```
python -m src.import_ibkr_trades from_summary money dein_user_  
python -m src.process_ibkr_trades money dein_user_
```



Manuell erhaltene XML importieren (z.B. für einen zweiten Benutzer)

```
# Einfach ablegen – Umbenennen ist nicht nötig (kein Dateinamen-Filter im Normalbetrieb)
```

```
Move-Item Bericht_zweiter_user.xml C:\Trade_XML\incoming_mail\
```

```
# Import (Routing per accountId automatisch) + Processing + Summary
```

```
python -m src.import_ibkr_trades
```

```
python -m src.process_ibkr_trades money zweiter_user_
```

```
python -m src.xml_append_summary money zweiter_user_
```

XML schnell im TEST-Environment testen

Siehe TEST-Environment für Details.

```
Rename-Item C:\Trade_XML\incoming_mail\meine_datei.xml flex.meine_datei.xml
```

```
Move-Item C:\Trade_XML\incoming_mail\flex.meine_datei.xml
```

```
C:\Trade_XML\test\dein_user\incoming\
```

```
python run_pipeline.py
```



Installation: Windows-Grundausrüstung

Ausführlichere, für Einsteiger geschriebene Version mit mehr Erklärungen:
Benutzerhandbuch: Windows vorbereiten.

Diese Seite deckt die **Basis-Software** ab, die auf einem neuen Windows-Rechner installiert sein muss, bevor die projektspezifischen Setup-Anleitungen (Installation: TradingJournal-Weaver, Installation: TradingJournal) funktionieren. Beschreibt lokale Entwicklungsrechner.

Wichtig

Läuft nur unter Windows. Die gesamte Installation ist auf Windows zugeschnitten: die Schritt-für-Schritt-Anleitungen, die Hilfsskripte (PowerShell) und die Ablageordner für die IBKR-Berichte (standardmäßig `C:\Trade_XML\`, `C:\Trade_CSV\`). Andere Betriebssysteme werden derzeit nicht unterstützt.

Der Datenimport (TradingJournal-Weaver) ist für Linux und macOS **vorbereitet**: Die Ablageordner lassen sich in der `.env` einstellen (`TRADE_XML_DIR`, `TRADE_CSV_DIR`, sonst `~/Trade_XML` und `~/Trade_CSV`), und die automatischen Tests laufen unter Windows, Linux und macOS. Offiziell unterstützt — mit Anleitungen und Hilfsskripten — wird bisher nur Windows.

Übersicht

Software	Wofür	Woher
Git for Windows	Repos klonen/verwalten	git-scm.com/install/windows
XAMPP	Apache, PHP, MySQL (lokale DB)	apachefriends.org
Composer	PHP-Abhängigkeiten (TradingJournal)	getcomposer.org
Node.js + npm	Frontend-Assets bauen (Vite)	nodejs.org (LTS)
Python 3.14	Import-Pipeline (TradingJournal-Weaver)	python.org

(„Artisan“ ist **kein** eigenes Programm — siehe Hinweis am Ende.)



1. Git for Windows

- Installer ausführen, Option „**Git from the command line and also from 3rd-party software**“, restliche Defaults übernehmen.
- Prüfen: `git --version`
- Einmalig konfigurieren (PowerShell) — nur nötig, wenn eigene Commits gemacht werden; Klonen und Updates (`git pull` , `update.ps1`) brauchen es nicht:

```
git config --global user.name "<Name>"
git config --global user.email "<Mail>"
```

- Prüfen, ob die Werte gesetzt sind:

```
git config --global --list
```

- **GitHub-Authentifizierung (HTTPS + Windows Credential Manager):** Der Git-for-Windows-Installer bringt den **Git Credential Manager** bereits mit (`credential.helper` ist danach automatisch auf `manager` gesetzt — mit `git config --global --list` prüfbar, Zeile `credential.helper=manager`). Beim ersten `git push` / `git pull` gegen ein GitHub-Repo öffnet sich ein Browser-Fenster zum GitHub-Login; die Zugangsdaten landen danach im Windows-Anmeldeinformationsspeicher und werden bei künftigen Aufrufen automatisch wiederverwendet — kein `gh` -CLI nötig (ist in diesem Projekt bewusst nicht installiert).

2. XAMPP (Apache + PHP + MySQL)

Dieses Projekt nutzt lokal den **kompletten XAMPP-Stack** — Apache, PHP und MySQL laufen alle über das XAMPP-Control-Panel. Wie im restlichen Projekt üblich, ist hier durchgängig von **MySQL** die Rede (auch wenn XAMPP technisch die MySQL-kompatible MariaDB bündelt — für dieses Projekt ohne relevanten Unterschied, PyMySQL/PDO-MySQL sprechen dasselbe Protokoll).

Wichtig bei der Wahl der XAMPP-Version: Apache Friends bietet mehrere parallele Versionszweige an. `composer.json` von TradingJournal verlangt `php: ^8.2` (Laravel 12) — also den **XAMPP-8.2.x-Zweig** (oder neuer) herunterladen, nicht die jeweils „neueste“ Version ungeprüft nehmen, falls die aktuellste einen anderen PHP-Major bündelt.

Installation

1. Installer von [apachefriends.org](https://www.apachefriends.org) laden, Standardpfad `C:\xampp` verwenden.
2. Bei der Komponentenauswahl mindestens **Apache, MySQL, PHP** aktiviert lassen — restliche Komponenten (FileZilla, Mercury, Tomcat) werden nicht gebraucht.



3. **XAMPP Control Panel** starten, **Apache** und **MySQL** über die jeweiligen „Start“-Buttons hochfahren. Für den Standardbetrieb mit `php artisan serve` reicht MySQL; Apache wird nur für den VirtualHost-Betrieb gebraucht (Installation: TradingJournal, Variante B).

4. Prüfen: `php -v` (muss `PHP 8.2.x` oder höher zeigen), MySQL-Client testen mit `C:\xampp\mysql\bin\mysql -u root -p` (Standard-Passwort bei frischer XAMPP-Installation: leer).

PHP zum PATH hinzufügen

XAMPP trägt sich **nicht automatisch** in die `PATH`-Umgebungsvariable ein — ohne diesen Schritt funktionieren `php`, `php artisan` und `composer` nur, wenn man direkt im XAMPP-Ordner steht.

Über die Windows-Oberfläche:

1. Windows-Taste drücken, „Umgebungsvariablen“ eingeben → „**Umgebungsvariablen für dieses Konto bearbeiten**“ öffnen.
2. Unter „**Benutzervariablen**“ die Variable `Path` markieren → **Bearbeiten**.
3. **Neu** → `C:\xampp\php` eintragen → optional zweiten Eintrag `C:\xampp\mysql\bin` ergänzen (damit auch `mysql` / `mysqldump` direkt ohne vollen Pfad aufrufbar sind) → mit **OK** überall bestätigen.
4. **Alle offenen Terminals/IDEs schließen und neu öffnen** — PATH-Änderungen wirken erst in neu gestarteten Prozessen.

Alternative per PowerShell (kein Admin nötig, wirkt nur für den aktuellen Windows-Account):

```
[Environment]::SetEnvironmentVariable(  
    "Path",  
    $env:Path + ";C:\xampp\php;C:\xampp\mysql\bin",  
    "User"  
)
```

Danach ebenfalls alle offenen Terminals neu starten. Prüfen mit `php -v` in einem **neuen** Terminal-Fenster.

PHP-Extension `zip` aktivieren

XAMPP liefert PHP standardmäßig mit der `zip`-Extension **deaktiviert** aus — Composer (für `TradingJournal`) braucht sie aber, um Pakete zu entpacken. Ohne diesen Schritt bricht `composer install` später mit einem Hinweis wie „`zip extension not found`“ oder „`... zip extension and unzip/7z commands are both missing`“ ab bzw. warnt zumindest.

1. `C:\xampp\php\php.ini` in einem Text-Editor öffnen.
2. Zeile `;extension=zip` suchen (Suche nach `zip`) und das führende `;` entfernen, sodass dort `extension=zip` steht.
3. Datei speichern.
4. Neues Terminal öffnen (Ini-Änderungen wirken erst dort). Prüfen:

```
php -m | findstr -i zip
```



sollte `zip` auflisten. Falls Apache bereits läuft, danach ebenfalls neu starten.

XAMPP als Windows-Dienst installieren

Standardmäßig laufen Apache/MySQL nur, solange das XAMPP Control Panel offen ist und die Module manuell gestartet wurden. Als Windows-Dienst installiert, starten sie automatisch beim Systemstart und laufen unabhängig vom Control Panel weiter — praktisch, wenn man nicht jedes Mal vor der Arbeit erst das Control Panel öffnen und beide Module anklicken will.

Über das XAMPP Control Panel (einfachster Weg):

1. XAMPP Control Panel **als Administrator** starten (Rechtsklick → „Als Administrator ausführen“) — ohne Admin-Rechte lässt sich kein Dienst registrieren.
2. In der Zeile **Apache** die Checkbox ganz links in der Spalte „**Svc**“ anklicken → registriert Apache als Windows-Dienst. Gleiches für die Zeile **MySQL**. Die dabei meist erscheinende UAC-Abfrage (Benutzerkontensteuerung) mit **Ja** bestätigen.
3. Beide Zeilen zeigen danach „Running“ auch nach Neustart, ohne dass sie manuell erneut gestartet werden müssen. Kontrolle: `services.msc` öffnen → **Apache2.4** und **MySQL** sind gelistet, Starttyp auf **Automatisch** setzen (Rechtsklick → Eigenschaften), falls das nicht schon der Fall ist.

Alternative per Kommandozeile (falls die Checkbox-Methode nicht greift — PowerShell **als Administrator**):

```
C:\xampp\apache\bin\httpd.exe -k install -n "Apache2.4"
C:\xampp\mysql\bin\mysqld.exe --install MySQL --defaults-
file="C:\xampp\mysql\bin\my.ini"

net start Apache2.4
net start MySQL
```

Rückgängig machen (Dienst entfernen, z.B. um zurück auf manuellen Start via Control Panel zu wechseln):

```
C:\xampp\apache\bin\httpd.exe -k uninstall -n "Apache2.4"
C:\xampp\mysql\bin\mysqld.exe --remove MySQL
```

Datenbank-Zugangsdaten: Frisches XAMPP hat den User `root` **ohne Passwort**. Für dieses Projekt in `.env` / `TradingJournal-Weaver/.env` entsprechend `DB_PASS=` / `TRADING_DB_PASSWORD=` leer lassen, oder — empfohlen — ein Passwort setzen (`ALTER USER 'root'@'localhost' IDENTIFIED BY '...'`) und in beiden `.env`-Dateien eintragen.

Port-Konflikte: Apache belegt standardmäßig Port 80 — falls das fehlschlägt (Skype, IIS, ein anderer Webserver), im XAMPP Control Panel unter „Config“ → `httpd.conf` den Port ändern. MySQL läuft auf dem für dieses Projekt erwarteten Standardport 3306, hier normalerweise keine Anpassung nötig.



Ein Produktionsserver läuft üblicherweise mit einer **Standalone-MySQL-Installation** statt XAMPP — ein bewusster, bekannter Unterschied zwischen lokaler Entwicklung und Produktion, hier nicht weiter behandelt.

PHP aktualisieren, ohne auf ein neues XAMPP zu warten

XAMPP bündelt eine feste PHP-Version und liefert neue Versionen oft erst Wochen nach einem PHP-Sicherheitsupdate aus. Bei kritischen Lücken (angekündigt unter [php.net](https://www.php.net/news) → News) lässt sich PHP innerhalb desselben Versionszweigs (z. B. 8.2.12 → 8.2.35) selbst austauschen. `php.ini`, PEAR und die XAMPP-Extras bleiben dabei erhalten. Für einen Wechsel des Zweigs (z. B. 8.2 → 8.3) ist dieser Weg nicht gedacht — dann besser ein passendes XAMPP neu installieren.

1. Passendes Build bestimmen — [PowerShell]:

```
C:\xampp\php\php.exe -v
```

Die erste Zeile zeigt Version und Build, z. B. `PHP 8.2.12 (cli) ... (ZTS Visual C++ 2019 x64) . ZTS` bedeutet **Thread Safe** (Pflicht für Apache mit `mod_php`), `Visual C++ 2019` entspricht **VS16**. Auf windows.php.net/download **[Browser]** im selben Zweig die neueste Version als **VS16 x64 Thread Safe** (ZIP) laden — nicht die Non-Thread-Safe-Variante.

2. Download prüfen — [PowerShell]: Die SHA256-Summe mit der Angabe auf der Download-Seite (bzw. in `sha256sum.txt` im selben Verzeichnis) vergleichen:

```
Get-FileHash $env:USERPROFILE\Downloads\php-8.2.35-Win32-vs16-x64.zip -Algorithm SHA256
```

3. Entpacken und sichern — [PowerShell]:

```
Expand-Archive $env:USERPROFILE\Downloads\php-8.2.35-Win32-vs16-x64.zip -DestinationPath  
C:\xampp\php-neu  
robocopy C:\xampp\php C:\xampp\php-backup /E
```

Prüfen, dass jede in `php.ini` aktivierte Erweiterung (`extension=...`) auch im neuen Build liegt (`C:\xampp\php-neu\ext\php_<name>.dll`). XAMPP bringt einzelne Zusätze mit, die das offizielle Build nicht enthält (z. B. Oracle-Treiber) — für dieses Projekt nicht nötig.

4. Austauschen — [PowerShell – Administrator]: Apache muss dafür gestoppt sein, sonst sind die PHP-DLLs gesperrt. Die neuen Dateien werden **über** den bestehenden Ordner kopiert; `php.ini` ist nicht im ZIP und bleibt deshalb unverändert.

```
net stop Apache2.4  
robocopy C:\xampp\php-neu C:\xampp\php /E  
net start Apache2.4
```



Läuft Apache nicht als Dienst, stattdessen im XAMPP Control Panel stoppen bzw. starten.

5. Prüfen:

- **[PowerShell]** `php -v` zeigt die neue Version.
- **[Browser]** `http://localhost/dashboard/phpinfo.php` zeigt in Apache ebenfalls die neue Version — das ist der Teil, der für die Weboberfläche zählt.
- **[Text-Editor]** `C:\xampp\apache\logs\error.log` : Nach dem Neustart darf keine Zeile `PHP Warning: PHP Startup: Unable to load dynamic library` stehen.

Bekannter Stolperstein — curl lädt in Apache nicht: Erscheint im Apache-Log `Unable to load dynamic library 'curl'`, während `php -m` in der PowerShell `curl` auflistet, fehlt dem Apache-Prozess eine DLL. Neuere `php_curl.dll`-Versionen brauchen Bibliotheken (z. B. `brotlidedc.dll`), die nur im PHP-Ordner liegen. Apache sucht dort nicht, sondern nimmt ältere oder gar keine Fassungen aus `C:\xampp\apache\bin`. Abhilfe: In `C:\xampp\apache\conf\extra\httpd-xampp.conf` **[Text-Editor]** direkt unter den vorhandenen `LoadFile`-Zeilen (vor `LoadModule php_module`) ergänzen:

```
LoadFile "C:/xampp/php/brotlicommon.dll"
LoadFile "C:/xampp/php/brotlidedc.dll"
LoadFile "C:/xampp/php/nghttp2.dll"
LoadFile "C:/xampp/php/libssh2.dll"
```

Danach Apache neu starten und das Log erneut prüfen.

Zurück zur alten Version — [PowerShell – Administrator]:

```
net stop Apache2.4
robocopy C:\xampp\php-backup C:\xampp\php /E
net start Apache2.4
```

Läuft alles, können `C:\xampp\php-neu` und nach einigen Tagen auch `C:\xampp\php-backup` gelöscht werden.

3. Composer

- Installer von getcomposer.org (`Composer-Setup.exe`) ausführen — erkennt eine vorhandene PHP-Installation automatisch; falls nicht gefunden, manuell auf `C:\xampp\php\php.exe` zeigen.
- Prüfen: `composer -V`



4. Node.js + npm

- LTS-Installer von nodejs.org — Standardoptionen reichen.
- Prüfen: `node -v` und `npm -v`

5. Python 3.14

- **Python 3.14**, nicht die neueste Version: `requirements.txt` pinnt Pakete (u. a. `pandas==3.0.0`), für die es unter Python 3.15 keine fertigen Wheels gibt — `pip install` scheitert dann. Eigenständigen Installer nehmen: python.org/downloads/windows → oberster *Python 3.14.x*-Eintrag → *Windows installer (64-bit)*; nicht den *Python install manager* (installiert die neueste Version).
- **Wichtig:** Beim Installer die Checkbox „Add python.exe to PATH“ aktivieren — sonst funktionieren `python` / `pip` nicht direkt im Terminal.
- Prüfen: `python --version` und `pip --version`

6. Projektverzeichnis anlegen und Repos klonen

PowerShell startet **nicht** automatisch im gewünschten Arbeitsverzeichnis — das Startverzeichnis hängt davon ab, wie das Fenster geöffnet wurde:

Wie geöffnet	Standard-Startverzeichnis
Normal übers Startmenü / Windows Terminal	meist <code>C:\Users\<Benutzername></code>
„Als Administrator ausführen“ (Rechtsklick)	<code>C:\Windows\System32</code> — häufigste Ursache für „ich lande immer in system32“

`Set-Location` (Alias: `cd`) wechselt das Verzeichnis, `Get-Location` (Alias: `pwd`) zeigt, wo man gerade steht. Root-Ordner fürs Projekt anlegen und hineinwechseln:

```
New-Item -ItemType Directory -Force -Path C:\Users\<Benutzername>\trading
Set-Location C:\Users\<Benutzername>\trading
```

`C:\Users\<Benutzername>\trading\` ist frei wählbar, keine technische Vorgabe — es ist nur ein Vorschlag für den Ordner, unter dem das Projekt lokal liegt. Jeder beliebige Ordner funktioniert genauso, z.B. `D:\dev\trading` oder `C:\trading`. Was tatsächlich zählt: `TradingJournal-Weaver` und



`TradingJournal` müssen als **direkte Geschwisterordner** im selben Elternverzeichnis liegen — `TradingJournal/update.ps1` erwartet den Weaver unter `../TradingJournal-Weaver` (relativ zum eigenen Skript). Liegt einer der beiden woanders, bricht das Update-Skript ab.

Unabhängig davon fest verdrahtet: `C:\Trade_XML\` und `C:\Trade_CSV\` (die IBKR-Import-Ordner) stehen in `import_ibkr_trades.py`, `get_flexquery_emails.py` und `get_flexquery_api.py` als absolute Pfade **direkt an der Wurzel von Laufwerk C:**, unabhängig davon, wo das Projekt selbst liegt — nicht konfigurierbar, nicht verschiebbar, ohne Codeänderung.

Manuelles Anlegen ist aber nicht nötig — beide Ordner samt kompletter Unterstruktur (`incoming_mail\`, `incoming_api\`, `incoming\`, `<env>\<user>\processed\`, `test\<user>\incoming\`, etc.) werden von den Scripts selbst per `mkdir(exist_ok=True)` beim ersten Lauf automatisch erzeugt, falls sie fehlen. Einzige Voraussetzung: der Prozess braucht Schreibrechte auf `C:\` — bei einem normalen Benutzerkonto ohne Sonderbeschränkungen ist das im Regelfall automatisch der Fall.

Repos klonen

Bei `TradingJournal-Weaver` per Voreinstellung nur die zum **Betreiben** nötigen Dateien holen, ohne Test-Suite/Tiefendokumentation/persönliche Einmal-Skripte (per Git **Sparse-Checkout** — kein Abweichen von „alles kommt aus Git“, `git pull` für spätere Updates funktioniert danach ganz normal weiter). `TradingJournal` ist bereits ohne Zusatz-Ballast, dort reicht ein normaler `git clone`:

```
# Im leeren Root-Ordner (siehe oben):
git clone --filter=blob:none --no-checkout https://github.com/mike-1140/TradingJournal-Weaver.git
cd TradingJournal-Weaver
git sparse-checkout init --cone
git sparse-checkout set sql src tools/maintenance dev docs
git checkout master
cd ..

git clone https://github.com/mike-1140/TradingJournal.git
```

Ergebnis:

```
<gewählter Root-Ordner>\
├─ TradingJournal-Weaver\    (nur sql/, src/, tools/maintenance/, dev/, docs/ + Root-
                             Dateien wie run_pipeline.py)
└─ TradingJournal\
```

`dev/` gehört dazu, weil dort die Datenbank-Migrationen (`migrate_*.py`) liegen, die `update.ps1` bei jedem Update ausführt. Ein bestehender Sparse-Klon ohne `dev/` lässt sich nachträglich ergänzen — **[PowerShell]** im Ordner `TradingJournal-Weaver`:



```
git sparse-checkout add dev
```

Danach weiter mit Installation: TradingJournal-Weaver (ab Schritt 2 — Repo ist ja bereits geklont) und der TradingJournal-Setup-Anleitung.

Nur für Entwicklung: vollständiger TradingJournal-Weaver –Klon

Wer an TradingJournal-Weaver selbst mitentwickeln will, braucht zusätzlich die Test-Suite, Tiefendokumentation und die persönlichen Einmal-Skripte aus `dev/` — dafür einfach ganz normal klonen, ohne Sparse-Checkout, **statt** der obigen drei TradingJournal-Weaver –Zeilen:

```
git clone https://github.com/mike-1140/TradingJournal-Weaver.git
```

Zu „Artisan“

`artisan` ist **kein separat zu installierendes Programm**, sondern ein PHP-Skript (`TradingJournal/artisan`), das mit jedem Laravel-Projekt mitkommt. Voraussetzung dafür sind nur PHP (Schritt 2) und die per Composer installierten Abhängigkeiten (TradingJournal-Setup, Schritt 1). Aufruf immer aus dem Projektordner:

```
cd TradingJournal  
php artisan --version
```

Verifikations-Checkliste

Nach Installation aller Programme, jeweils in einem neuen Terminal:

```
git --version  
php -v          # ≥ 8.2  
composer -V  
node -v  
npm -v  
python --version # 3.14
```

Sowie XAMPP Control Panel: Apache und MySQL zeigen „Running“ (grün), und Abschnitt 6 oben (Repos geklont, Ordnerstruktur stimmt).

Danach weiter mit Installation: TradingJournal-Weaver und der Installation: TradingJournal für die projektspezifische Einrichtung.

Installation: TradingJournal-Weaver

Ausführlichere, für Einsteiger geschriebene Version mit mehr Erklärungen:
Benutzerhandbuch: TradingJournal-Weaver installieren.

Installations- und Erstinitialisierungs-Anleitung für `TradingJournal-Weaver` auf einem neuen Rechner. Für Befehlsreferenz siehe `docs/handbuch/technik/30-befehlsreferenz.md`, für Architektur/FIFO-Logik/DB-Schema im Detail siehe `docs/handbuch/technik/10-architektur-und-pipeline.md` (beide im `TradingJournal-Weaver`-Repo).

Grundlegende Software (Git, Python, MySQL/MariaDB via XAMPP) muss vorher installiert sein — siehe Installation: Windows-Grundausrüstung. Diese Seite setzt das voraus und beschreibt nur noch die projektspezifische Einrichtung.

Voraussetzungen

Software	Version	Hinweis
Python	3.14	Bei der Installation „Add python.exe to PATH“ aktivieren
MySQL/MariaDB Server	via XAMPP	Lokal, Port 3306, laufend und erreichbar; Zugangsdaten mit Recht zum Anlegen neuer Datenbanken — siehe „Installation: Windows-Grundausrüstung“
Git	aktuell	Für Clone + Versionsverwaltung

1. Repository klonen

`TradingJournal-Weaver` hat ein eigenes Git-Repository. Falls schon per Installation: Windows-Grundausrüstung Abschnitt 6 geklont, diesen Schritt überspringen.

```
git clone https://github.com/mike-1140/TradingJournal-Weaver.git
cd TradingJournal-Weaver
```



2. Virtuelle Umgebung anlegen

```
python -m venv .venv
.venv\Scripts\activate
pip install -r requirements.txt
```

Komfort-Skript für künftige Sitzungen: `activate.ps1` (im `TradingJournal-Weaver` -Ordner) wechselt in dieses Verzeichnis und aktiviert die venv in einem Schritt — mit vorangestelltem Punkt aufrufen, damit es im aktuellen Fenster wirkt, nicht nur im Skript selbst:

```
.. \TradingJournal-Weaver\activate.ps1
```

Falls dabei eine Fehlermeldung wie „...kann nicht geladen werden, da die Ausführung von Skripts auf diesem System deaktiviert ist“ kommt (Standardeinstellung auf vielen frischen Windows-Rechnern): einmalig als Administrator `Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned`, danach neues Terminal öffnen und den Befehl erneut ausführen.

Desktop-Verknüpfung (optional, einmalig einrichten): `create-desktop-shortcut.ps1` (ebenfalls im `TradingJournal-Weaver` -Ordner) legt eine `.lnk` -Verknüpfung auf dem Desktop an, die per Doppelklick PowerShell öffnet, direkt in `TradingJournal-Weaver` mit aktivierter venv — danach z.B. sofort `python run_pipeline.py` möglich. Ein `.ps1` selbst startet bei Doppelklick **nicht** automatisch PowerShell (öffnet sich standardmäßig im Text-Editor), deshalb der Umweg über eine echte `.lnk` -Verknüpfung:

```
.. \TradingJournal-Weaver\create-desktop-shortcut.ps1
```

3. `.env` anlegen

```
copy .env.example .env
```

Werte ausfüllen (siehe `.env.example` für alle Felder):



Variable	Bedeutung
<code>DB_USER</code> / <code>DB_PASS</code>	MySQL-Zugangsdaten
<code>DB_HOST</code> / <code>DB_PORT</code>	Standard: <code>127.0.0.1</code> / <code>3306</code>
<code>DEFAULT_USER</code>	Eigener Username (z.B. <code>dein_user</code>) — Tabellen-Prefix wird daraus aus <code>users.toml</code> aufgelöst. Alternativ direkt <code>TABLE_PREFIX</code> setzen (Override)
<code>IMAP_USERNAME</code> / <code>IMAP_PASSWORD</code> / <code>IMAP_SERVER</code>	Nur nötig, falls <code>get_flexquery_emails.py</code> genutzt wird
<code>IBKR_FLEX_QUERIES</code>	Optional — nur für den zweiten Import-Kanal <code>get_flexquery_api.py</code>
<code>MAX_GAP_DAYS</code>	Optional — Tage, die eine glattgestellte Kampagne <i>PENDING</i> bleibt, bevor sie <i>CLOSED</i> wird (Standard <code>10</code> , ganze Zahl ≥ 1)
<code>TRADE_XML_DIR</code> / <code>TRADE_CSV_DIR</code>	Optional — Ablageordner für die IBKR-Berichte (XML bzw. Kontoauszug-CSV). Standard: Windows <code>C:\Trade_XML</code> / <code>C:\Trade_CSV</code> , Linux/macOS <code>~/Trade_XML</code> / <code>~/Trade_CSV</code> . Alle Unterordner werden darunter angelegt
<code>TWS_HOST</code> / <code>TWS_PORT</code> / <code>TWS_CLIENT_ID</code>	Optional, nur für den Schritt <code>enrich</code> — Verbindung zur TWS bzw. zum IB Gateway (Standard <code>127.0.0.1</code> , <code>7497</code> = Papierkonto, <code>630</code>). Die clientId darf keine andere TWS-Anwendung nutzen
<code>GREEKS_RATE</code> / <code>GREEKS_TRADE_TZ</code>	Optional, nur für <code>enrich</code> — Zins für Black-Scholes (Standard <code>0.04</code>) und Zeitzone der Ausführungszeiten aus der Flex Query (Standard <code>America/New_York</code>)

Sicherheit: `.env` ist in `.gitignore` — niemals committen.

`users.toml` (lokal, gitignored — aus `users.toml.example` kopieren: `copy users.toml.example users.toml`) ist die zentrale, einzige Quelle für die User↔Prefix-Zuordnung (`dein_user` → `dein_user_`, `zweiter_user` → `zweiter_user_`) — `DEFAULT_USER` oben, `accounts.toml` (Kontozuordnung) und `pipeline.toml` (Schritt 5) lösen den Prefix alle von dort auf. Jeder dort genannte User muss in `users.toml` stehen, sonst bricht der Import mit einer Meldung ab. Ebenso anlegen: `accounts.toml` aus `accounts.toml.example` (Kontozuordnung, letzte vier Ziffern der IBKR-Kontonummer → User + Environment).

Die `DB_NAME_*`-Variablen (`ibkr_money`, `ibkr_paper`, `ibkr_test`, `ibkr_autom_money`, `ibkr_autom_paper`) müssen normalerweise nicht angepasst werden — sie legen die tatsächlichen MySQL-Datenbanknamen fest und werden von TradingJournal (`config/database.php`) in gleicher Form erwartet.



4. Datenbanken initialisieren

Destruktiv

— legt Datenbank + alle Tabellen aus `sql/` neu an, löscht bestehende Daten des jeweiligen Environment/Prefix-Paars. Nur beim ersten Setup bzw. bewusstem Reset ausführen.

Für jede benötigte Kombination aus Environment × Nutzer-Prefix einmal ausführen — `test` immer (Start-Environment der Weboberfläche, solange `TRADING_DEFAULT_ENV` nicht gesetzt ist):

```
python -m tools.maintenance.reset_IBKR_DB test dein_user_  
python -m tools.maintenance.reset_IBKR_DB money dein_user_  
python -m tools.maintenance.reset_IBKR_DB paper dein_user_
```

Zusätzliche Tabellen außerhalb des Kern-Schemas (z.B. Psychology-Modul) werden **nicht** über `reset_IBKR_DB.py` angelegt, sondern über eigene Setup-Skripts in `tools/maintenance/`, damit ein Reset nicht versehentlich fachfremde Tabellen mitlöscht:

```
python -m tools.maintenance.setup_psychology_tables money dein_user_
```

Für einen zusätzlichen, weiteren User (z.B. ein Familienmitglied, nachdem die Grundinstallation schon steht): diesen Schritt mit dem neuen Prefix wiederholen (`users.toml` ggf. vorher um den neuen User ergänzen, siehe Schritt 3), danach den passenden Laravel-User anlegen (siehe TradingJournal-Setup-Anleitung, Schritt 5) — **mit demselben Prefix** auf beiden Seiten.

5. `pipeline.toml` anlegen

```
copy pipeline.toml.example pipeline.toml
```

`pipeline.toml` ist **maschinenspezifisch** und daher wie `.env` in `.gitignore` (nicht versioniert) — jede Maschine trägt hier nur die User/Environments ein, die auf **ihr** tatsächlich per Schritt 4 initialisiert wurden. Ein Eintrag für einen User, dessen Tabellen auf dieser Maschine nicht existieren, lässt `run_pipeline.py` im Auto-Modus mit „Tabelle nicht gefunden“ fehlschlagen.



```
[users.dein_user]
environments = ["money"]    # nur die Environments, die in Schritt 4 initialisiert wurden

[[test]]                  # TEST je User als eigener Block (steht schon in der
Vorlage),
user = "dein_user"        # nicht zusätzlich in environments – sonst läuft TEST doppelt
```

Kein `prefix` = nötig – der wird automatisch aus `users.toml` aufgelöst (User-Name = Name der Sektion, hier `dein_user`). Nur falls ein User (noch) nicht in `users.toml` steht, hier per `prefix = "..."` explizit setzen.

6. Ersten Lauf testen

```
python run_pipeline.py --dry-run
```

Zeigt alle konfigurierten Steps/Befehle laut `pipeline.toml`, ohne etwas auszuführen. Danach für einen echten Testlauf (User-/Env-Namen an die eigene `pipeline.toml` anpassen):

```
python run_pipeline.py --users dein_user --envs test
```

7. Altdaten nachholen (einmalig)

Die tägliche Query deckt nur die letzten n Tage ab. Historie vor der Installation einmalig im Kundenportal abrufen: Flex Query **Starten**, Zeitraum *Benutzerdefinierter Zeitraum*, Format XML (max. 365 Tage je Abruf, vom Portal erzwungen; Paper-Konten nur rund 180 Tage rückwirkend, frühere Tage sind im Kalender gesperrt). Die `.xml`-Dateien nach `C:\Trade_XML\incoming_mail\` legen (Dateiname egal, Routing über `accountId` → `accounts.toml`), dann regulär `python run_pipeline.py`.

Vor dem ersten produktiven `process`-Lauf einspielen bzw. im selben Lauf: `process` sortiert nur die unverarbeiteten Zeilen (`trades_raw.processed=0`) nach `dateTime`. Alte Trades, die nach bereits verarbeiteten neueren kommen, laufen danach durch das FIFO-Matching und verfälschen Tranchen und Kampagnen. Überlappungen mit den täglichen Berichten fängt der Checksum-Dedup ab. Kontostände (`account_snapshots`) kommen nur aus Kontoauszug-CSVs mit **einem** Stichtag (`Statement, Data, Period`): rückwirkend nur per Einzel-Download je Tag im Portal (*Täglich* + Datum →



CSV) nach `C:\Trade_CSV\incoming\`, Routing über die `accountId` in der CSV, Reihenfolge egal (Dedup je `snapshot_date`). Monats-/Zeitraumauszüge haben kein parsebares Einzeldatum → *Kein Datum im CSV – übersprungen*, Datei bleibt in `incoming\` liegen.

8. Web-API: DNS-Auflösung des IBKR-Hosts

`get_flexquery_api.py` ruft `gdcdyn.interactivebrokers.com` auf. Der Name ist kein einfacher A-Eintrag, sondern eine Akamai-CNAME-Kette:

```
gdcdyn.interactivebrokers.com
→ www.interactivebrokers.com.edgekey.net
→ e72360.a.akamaiedge.net → A-Records (wechselnd)
```

DNS-Forwarder in Handy-Hotspots (Android: `192.168.43.x`) und manchen kleinen Routern bzw. der DNS des Mobilfunkanbieters dahinter scheitern an dieser Kette (`nslookup` → `Server failed`), während kurze Namen weiter aufgelöst werden. Folge im Log: `NameResolutionError ... [Errno 11002] getaddrinfo failed`. Die Anfrage erreicht IBKR nicht – kein Token-/Query-Fehler und kein fehlgeschlagener Versuch im Sinne der Sperre 1025. Der Mail-Abruf (`get_flexquery_emails.py`) nutzt andere Hostnamen und ist nicht betroffen.

Behebung: feste DNS-Server (1.1.1.1, 8.8.8.8) auf dem Netzwerkadapter, Schritte in Fehlerbehebung und häufige Fragen. Hintergrund dazu:

- Windows speichert die Einstellung **je Adapter** in der Registry (`HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters\Interfaces\{Adapter-GUID}`): `NameServer` = feste Server, `DhcpNameServer` = Vorschlag des Routers/Hotspots. Ist `NameServer` gesetzt, hat er Vorrang.
- Sie gilt unabhängig vom verbundenen Netz und übersteht Netzwechsel, Neustart und Neustart des Hotspot-Handys. Sie gilt aber **nur für diesen Adapter** – WLAN, Ethernet und USB-Tethering haben jeweils eigene Einträge.
- Tritt der Fehler trotz Fix wieder auf, zuerst prüfen, ob die Einstellung noch gesetzt ist (PowerShell, ohne Administrator):

```
Get-DnsClientServerAddress -InterfaceAlias "WLAN" -AddressFamily IPv4
```

Steht dort die Adresse des Routers/Hotspots statt `1.1.1.1`, `8.8.8.8`, wurde sie zurückgesetzt (z. B. durch Netzwerk-Reset oder Windows-Update) und muss neu gesetzt werden.

- Offene WLANs mit Anmeldeseite (Hotel, Bahn) zeigen diese mit festen DNS-Servern teils nicht an. Dann vorübergehend `-ResetServerAddresses`, anmelden, danach wieder setzen.



Weiterführende Dokumentation

Datei	Inhalt
docs/handbuch/technik/30-befehlsreferenz.md	Vollständige Befehlsreferenz aller Scripts
docs/handbuch/technik/10-architektur-und-pipeline.md	Architektur, FIFO-Logik, DB-Schema im Detail
Installation: TradingJournal	Setup der Weboberfläche (liest dieselben Datenbanken)

Installation: TradingJournal

Ausführlichere, für Einsteiger geschriebene Version mit mehr Erklärungen:
Benutzerhandbuch: TradingJournal installieren.

Installations- und Erstinitialisierungs-Anleitung für [TradingJournal](#) auf einem neuen Rechner.

Voraussetzung: [TradingJournal-Weaver](#) ist bereits eingerichtet (siehe Installation: TradingJournal-Weaver) — TradingJournal zeigt nur Daten an, das Trading-Datenbankschema wird ausschließlich über [TradingJournal-Weaver](#) verwaltet.

Grundlegende Software (XAMPP für PHP/Apache/MariaDB, Composer, Node.js) muss vorher installiert sein — siehe Installation: Windows-Grundausrüstung. Diese Seite setzt das voraus und beschreibt nur noch die projektspezifische Einrichtung.

Voraussetzungen

Software	Version	Hinweis
PHP	^8.2 (Projekt läuft mit Laravel 12)	Aus XAMPP, siehe Windows-Grundausrüstung
Composer	aktuell	
Node.js + npm	aktuell	Für Vite-Asset-Build
MySQL/MariaDB Server	via XAMPP	Dieselbe Instanz wie TradingJournal-Weaver , Port 3306

1. Repository klonen + Abhängigkeiten

[TradingJournal](#) hat ein eigenes Git-Repository. Falls schon per Windows-Grundausrüstung Abschnitt 6 geklont, diesen Schritt überspringen.

```
git clone https://github.com/mike-1140/TradingJournal.git
cd TradingJournal

composer install
npm install
```



Falls `composer install` mit einem Hinweis auf die fehlende `zip` -Extension abbricht bzw. warnt (z.B. „`zip extension not found`“ oder „`... zip extension and unzip/7z commands are both missing`“):

XAMPP liefert PHP standardmäßig mit der `zip` -Extension **deaktiviert** aus.

Fix:

1. `C:\xampp\php\php.ini` in einem Text-Editor öffnen.
2. Zeile `;extension=zip` suchen (Suche nach `zip`) und das führende `;` entfernen, sodass dort `extension=zip` steht.
3. Datei speichern.
4. Neues Terminal öffnen (PATH-/Ini-Änderungen wirken erst dort) und `composer install` erneut ausführen. Falls Apache/ `php artisan serve` bereits läuft, diese(s) danach auch neu starten.

Prüfen, ob die Extension aktiv ist: `php -m | findstr -i zip` sollte `zip` auflisten.

Komfort-Skript für künftige Sitzungen: `open.ps1` (im `TradingJournal` -Ordner) wechselt in dieses Verzeichnis — mit vorangestelltem Punkt aufrufen, damit es im aktuellen Fenster wirkt, nicht nur im Skript selbst:

```
. .\TradingJournal\open.ps1
```

Falls dabei eine Fehlermeldung wie „...kann nicht geladen werden, da die Ausführung von Skripts auf diesem System deaktiviert ist“ kommt (Standardeinstellung auf vielen frischen Windows-Rechnern): einmalig als Administrator `Set-ExecutionPolicy -Scope CurrentUser -ExecutionPolicy RemoteSigned`, danach neues Terminal öffnen und den Befehl erneut ausführen.

Desktop-Verknüpfung (optional, einmalig einrichten): `create-desktop-shortcut.ps1` (ebenfalls im `TradingJournal` -Ordner) legt eine `.lnk` -Verknüpfung auf dem Desktop an, die per Doppelklick PowerShell direkt im `TradingJournal` -Ordner öffnet — danach z.B. sofort `php artisan serve` möglich. Genau wie bei `TradingJournal-Weaver` der Umweg über eine `.lnk` -Verknüpfung, weil ein `.ps1` bei Doppelklick nicht automatisch PowerShell startet:

```
. .\TradingJournal\create-desktop-shortcut.ps1
```

2. `.env` anlegen

```
copy .env.example .env  
php artisan key:generate
```



`.env.example` ist bereits auf MySQL mit XAMPP-Standardwerten eingestellt (`root` , kein Passwort) — bei frischer XAMPP-Installation ist nichts zu ändern. Bei abweichenden Zugangsdaten (z.B. Server) diese Werte anpassen:

```
DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ibkr_laravel
DB_USERNAME=<mysql-user>
DB_PASSWORD=<mysql-pass>

TRADING_DB_HOST=127.0.0.1
TRADING_DB_PORT=3306
TRADING_DB_USERNAME=<mysql-user>
TRADING_DB_PASSWORD=<mysql-pass>
```

`DB_*` steuert die Laravel-eigene Datenbank (`users` , `sessions` , `cache` — Name frei wählbar, z.B. `ibkr_laravel`). `TRADING_DB_*` steuert Zugangsdaten für die fünf Trading-Datenbanken. Deren **Namen selbst** (`ibkr_money` , `ibkr_paper` , `ibkr_test` , `ibkr_autom_money` , `ibkr_autom_paper`) sind in `config/database.php` fest hinterlegt und nicht über `.env` konfigurierbar — sie müssen mit den in `TradingJournal-Weaver/.env` (`DB_NAME_*`) definierten Namen übereinstimmen.

3. Laravel-eigene Tabellen anlegen

```
php artisan migrate
```

Legt `users` (inkl. `table_prefix` , `filter_preset` / `filter_from` / `filter_to` , `dashboard_config`), `cache` , `sessions` an. **Nicht vergessen** — ohne diesen Schritt existiert keine `users` -Tabelle und kein Login ist möglich.

4. Trading-Tabellen bereitstellen

Kein Laravel-Migrationsschritt. Falls noch nicht geschehen, in `TradingJournal-Weaver` ausführen (siehe Installation: TradingJournal-Weaver Schritt 4):



```
cd ../TradingJournal-Weaver
python -m tools.maintenance.reset_IBKR_DB test dein_user_
python -m tools.maintenance.reset_IBKR_DB money dein_user_
```

`test` immer anlegen: Ohne `TRADING_DEFAULT_ENV` in der `.env` startet die Weboberfläche im Environment TEST. Prefix (`dein_user_` etc.) muss zu dem Prefix passen, der beim Anlegen des Users in Schritt 5 verwendet wird.

5. Ersten User anlegen

Vorschritt, einmalig: Die Spalte `is_admin` in der `users` -Tabelle hat **keine** Laravel-Migration — sie wird nur durch diesen separaten Befehl per `ALTER TABLE` angelegt (idempotent, kann gefahrlos mehrfach laufen). Ohne diesen Schritt bricht `auth:create-user --admin` mit `SQL: Column not found: is_admin` ab:

```
php artisan setup:auth
```

(„User ID 1 nicht gefunden“ ist an dieser Stelle normal, wenn noch kein User existiert — wichtig ist nur, dass die Spalte danach da ist.)

```
php artisan auth:create-user "Dein Name" dein_name@example.com "PASSWORT" --
prefix=dein_user_ --admin
```

Option	Bedeutung
<code>name</code> , <code>email</code> , <code>password</code>	Pflichtargumente
<code>--prefix=</code>	Tabellen-Prefix, muss zum in Schritt 4 initialisierten Prefix passen
<code>--admin</code>	Setzt <code>is_admin=1</code> — nur lokal (<code>APP_ENV=local</code>) wirksam: Benutzer-Dropdown zum Wechsel auf andere User (Gate <code>switch-user</code>)

Der Befehl gibt nach dem Anlegen den passenden `reset_IBKR_DB` -Aufruf für den gewählten Prefix aus, falls Schritt 4 noch nicht erledigt wurde.

Für einen zusätzlichen, weiteren User (z.B. ein Familienmitglied, nachdem die Grundinstallation schon steht): Schritt 4 (`TradingJournal-Weaver` -Tabellen) mit dem neuen Prefix wiederholen, danach diesen Schritt hier noch einmal mit den neuen Zugangsdaten und **demselben Prefix** ausführen.



6. Assets bauen

```
npm run build      # Produktion
# oder für Entwicklung mit Hot Reload:
npm run dev
```

7. Starten

Zwei Möglichkeiten — beide führen zur selben App, unterschiedlich bequem für den Alltag:

Variante A: `php artisan serve` (einfach, aber manuell)

```
php artisan serve
```

`http://localhost:8000` aufrufen. Mit `APP_ENV=local` gibt es keine Login-Seite: Die Middleware `SetDefaultUser` meldet einmal pro Browser-Sitzung automatisch den User mit der ID 1 an (daher den eigenen Admin-Zugang als ersten anlegen). *Abmelden* beendet die Sitzung, die nächste Anfrage meldet wieder User 1 an — so kehrt der Admin nach einem Benutzerwechsel zurück. Login-Seite und Passwort gelten nur mit `APP_ENV=production` (Server-Betrieb). **Nachteil:** läuft nur, solange dieses Terminal-Fenster offen bleibt — bei jedem Neustart erneut manuell ausführen.

Variante B: Apache-VirtualHost (dauerhaft, kein manueller Start nötig)

Voraussetzung: XAMPPs Apache läuft (idealerweise als Windows-Dienst, siehe Installation: Windows-Grundausstattung). Dann liefert Apache die Seite automatisch mit Windows — kein `php artisan serve` mehr nötig.

Zwei Teile:

1. Windows-hosts-Datei — ordnet den Namen (hier z.B. `trading.local`) der eigenen Maschine zu, sonst findet der Browser den Namen gar nicht:

```
# Prüfen, ob der Eintrag schon da ist:
Get-Content C:\Windows\System32\drivers\etc\hosts | Select-String "trading.local"

# Falls kein Ergebnis: als Administrator (PowerShell "Als Administrator ausführen")
ergänzen:
Add-Content -Path C:\Windows\System32\drivers\etc\hosts -Value "127.0.0.1 trading.local"
```



2. Apache-VirtualHost — `C:\xampp\apache\conf\httpd.conf` (oder `C:\xampp\apache\conf\extra\httpd-vhosts.conf`, falls dort schon andere VHosts stehen) öffnen und ergänzen (Pfad an den eigenen Root-Ordner anpassen):

```
<VirtualHost *:80>
    ServerName trading.local
    DocumentRoot "C:/Users/<Benutzername>/trading/TradingJournal/public"

    <Directory "C:/Users/<Benutzername>/trading/TradingJournal/public">
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>
</VirtualHost>
```

Falls in `httpd.conf` bei `NameVirtualHost` /mehreren VHosts noch keine Grundkonfiguration für Port 80 existiert, reicht in der Regel der obige Block allein (XAMPPs Standard- `Listen 80` bleibt bestehen).

Danach Apache im XAMPP Control Panel neu starten (Stop → Start), und `http://trading.local` im Browser aufrufen.

Troubleshooting: Falls `http://trading.local` nicht erreichbar ist — zuerst den hosts-Eintrag prüfen (Befehl oben), dann `C:\xampp\apache\logs\error.log` auf Konfigurationsfehler prüfen, danach `C:\xampp\apache\bin\httpd.exe -t` (testet die `httpd.conf` -Syntax, ohne zu starten).

Weiterführende Dokumentation

Datei	Inhalt
<code>CLAUDE.md</code>	Entwickler-Notizen, Architektur-Patterns (Environment/Zeitfilter-Traits)
Installation: TradingJournal-Weaver	Setup der Datenimport-Pipeline (Python)

Server-Betrieb — TradingJournal auf einem eigenen Server

Alternative zur lokalen Installation (siehe Installation: TradingJournal): TradingJournal + TradingJournal-Weaver laufen auf einem eigenen Windows Server (physisch, VPS, oder ein Rechner, der dauerhaft eingeschaltet bleibt) statt auf dem lokalen Arbeitsplatz-PC. Sinnvoll, wenn du von mehreren Geräten (Laptop, Reise-Notebook, Tablet) auf dasselbe Journal zugreifen willst, ohne die Datenbank mehrfach zu pflegen.

Diese Anleitung ersetzt nicht Installation: TradingJournal-Weaver/ Installation: TradingJournal — sie überträgt dieselben Konzepte auf einen Server-Kontext. Grundkenntnisse aus diesen beiden Dokumenten werden vorausgesetzt.

Voraussetzungen

- Ein Windows-Server oder dauerhaft laufender Windows-Rechner, erreichbar per Remote-Zugriff (RDP, ein Remote-Desktop-Tool deiner Wahl, oder physischer Zugriff)
- **Ein eigenes VPN oder ein Reverse-Proxy mit Authentifizierung**, um den Server sicher von außen erreichbar zu machen — diese Anleitung geht davon aus, dass du diesen Teil bereits gelöst hast. Ein einfacher, kostenloser Einstieg: [Tailscale](#) (baut ein privates Mesh-VPN zwischen deinen Geräten auf, jedes Gerät bekommt eine feste, ausschließlich innerhalb des VPNs erreichbare IP-Adresse). Andere Optionen: WireGuard, klassisches Site-to-Site-VPN, oder ein Reverse-Proxy mit eigener Auth-Schicht.

Wichtig — niemals ungeschützt am offenen Internet betreiben: Diese Anleitung bindet Apache bewusst nur an die private VPN-Adresse des Servers, nicht an die öffentliche IP (siehe Firewall-Abschnitt unten). TradingJournal hat kein eigenes Rate-Limiting/Brute-Force-Schutz für den Login und ist nicht für einen öffentlich erreichbaren Betrieb konzipiert.

Notation in dieser Anleitung

Jeder Befehlsblock ist mit einem Label gekennzeichnet, welches Fenster/Programm gemeint ist:

- **[PowerShell]** — normales PowerShell-Fenster
- **[PowerShell – Administrator]** — PowerShell-Icon rechtsklicken → „Als Administrator ausführen“ (nötig für Service-Steuerung, Firewall-Regeln, Schreibrechte-Änderungen)
- **[MySQL-Client]** — am `mysql>`-Prompt (siehe Schritt 2)
- **[Text-Editor]** — eine Datei mit einem Texteditor bearbeiten (Editor/Notepad reicht)



- **[Installer-GUI]** — Klicks innerhalb eines grafischen Installations-Assistenten
- **[Windows-Einstellungen]** — Systemsteuerung/Einstellungen-App, kein Terminal

Platzhalter: `<server-vpn-ip>` (die private VPN-Adresse des Servers), `<db-passwort>` (selbst gewähltes, sicheres Passwort für den App-Datenbank-User).

Schritt 1 — Git for Windows

Download: <https://git-scm.com/install/windows> („Click here to download“)

- **[Installer-GUI]** Installieren mit Option „**Git from the command line and also from 3rd-party software**“, sonst Defaults
- **[PowerShell]** Prüfen: `git --version`

Schritt 2 — MySQL 8.4 LTS

Download: <https://dev.mysql.com/downloads/installer/>

- **[Installer-GUI]** Setup Type: **Custom** → **MySQL Server 8.4.x** auswählen — die Versionsnummer beim Auswählen genau prüfen, der Installer listet oft mehrere Versionen gleichzeitig auf. Ältere Versionen (z. B. 8.0) haben ihren offiziellen Support-Zeitraum bereits beendet und sollten für einen Server mit echten Finanzdaten nicht verwendet werden.
- **[Installer-GUI]** Config Type: **Server Computer**, Port **3306**
- **[Installer-GUI]** Root-Passwort setzen und sicher notieren
- **[Installer-GUI]** Windows Service Name: **MySQL84**, Autostart: **Ja**

Passwort vergessen/zurücksetzen

Falls das Root-Passwort verloren geht: der MySQL Installer (Reconfigure → „Accounts and Roles“) verlangt dafür das *aktuelle* Passwort und hilft bei einem vergessenen Passwort nicht. Stattdessen über eine **Init-Datei** (der Server führt sie beim Start einmalig aus, ganz ohne bestehendes Passwort):

1. **[PowerShell – Administrator]** Service stoppen: `net stop MySQL84`
2. **[Text-Editor]** Init-Datei anlegen (Notepad: Zeile eintippen, Speichern unter `C:\mysql-init.txt`, Dateityp „Alle Dateien“, Kodierung **ANSI**):

```
ALTER USER 'root'@'localhost' IDENTIFIED BY 'NeuesSicheresPasswort123!';
```

3. **[PowerShell – Administrator]** MySQL manuell im Vordergrund starten, mit Verweis auf die echte `my.ini` (ohne `--defaults-file` sucht `mysqld` das Datenverzeichnis am falschen Ort — Fehler `Failed to set datadir ... No such file or directory`). Dieses Fenster offen lassen:



```
& "C:\Program Files\MySQL\MySQL Server 8.4\bin\mysqld.exe" --defaults-  
file="C:\ProgramData\MySQL\MySQL Server 8.4\my.ini" --init-file=C:\mysql-init.txt --  
console
```

Warten, bis `ready for connections` im Log erscheint.

4. **[PowerShell]** — ein neues, zweites Fenster öffnen, Login testen:

```
& "C:\Program Files\MySQL\MySQL Server 8.4\bin\mysql.exe" -u root -p
```

5. Zurück zum ersten Fenster, dort `Strg+C` zum Stoppen.

6. **[PowerShell – Administrator]** Init-Datei löschen (enthält Passwort im Klartext): `Remove-Item C:\mysql-init.txt`, dann Service normal starten: `net start MySQL84`

Datenbanken + App-User anlegen — [MySQL-Client]

Verbinden entweder über die Startmenü-Verknüpfung „MySQL 8.4 Command Line Client“, oder aus **[PowerShell]** heraus mit `& "C:\Program Files\MySQL\MySQL Server 8.4\bin\mysql.exe" -u root -p` (der Prompt wechselt dann im selben Fenster von `PS C:\...>` zu `mysql>`).

```
CREATE DATABASE ibkr_laravel      CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
CREATE DATABASE ibkr_money       CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
CREATE DATABASE ibkr_paper       CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
CREATE DATABASE ibkr_autom_money CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
CREATE DATABASE ibkr_autom_paper CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci;  
  
CREATE USER 'trading_app'@'%' IDENTIFIED BY '<db-passwort>';  
GRANT ALL PRIVILEGES ON ibkr_laravel.*      TO 'trading_app'@'%';  
GRANT ALL PRIVILEGES ON ibkr_money.*        TO 'trading_app'@'%';  
GRANT ALL PRIVILEGES ON ibkr_paper.*        TO 'trading_app'@'%';  
GRANT ALL PRIVILEGES ON ibkr_autom_money.*  TO 'trading_app'@'%';  
GRANT ALL PRIVILEGES ON ibkr_autom_paper.*  TO 'trading_app'@'%';  
FLUSH PRIVILEGES;
```

Nur die Environments anlegen, die du tatsächlich brauchst — `ibkr_test` kann entfallen, wenn kein Test-Environment genutzt wird. Die App startet dann im Environment aus `TRADING_DEFAULT_ENV` bzw. automatisch im ersten Environment mit Tabellen (siehe Schritt 8).

Host bewusst % statt 127.0.0.1 : MySQL löst eine eingehende Verbindung von `127.0.0.1` serverseitig per Reverse-Lookup standardmäßig zu `localhost` auf und matcht die Rechte-Tabelle gegen den aufgelösten Namen, nicht die rohe IP. Ein User mit der exakten IP `127.0.0.1` als Host matcht dann nie und liefert `Access denied for user '...'@'localhost'` — auch mit explizitem `-h 127.0.0.1` am Client, da das Problem serverseitig sitzt. `%` umgeht dieses Hostname-Auflösungs-



Problem komplett. Sicherheitsmäßig unkritisch, solange `bind-address` (nächster Schritt) auf `127.0.0.1` steht — von außen kommt dann ohnehin nie eine TCP-Verbindung an, unabhängig vom User-Host-Feld.

MySQL nur lokal lauschen lassen — [Text-Editor] + [PowerShell – Administrator]:

`exit` im MySQL-Client, dann `C:\ProgramData\MySQL\MySQL Server 8.4\my.ini` öffnen:

```
[mysqld]
bind-address = 127.0.0.1
```

Speichern, dann `net stop MySQL84 && net start MySQL84`.

Verifizieren (kein phpMyAdmin nötig — reicht der MySQL-Client selbst):

```
SHOW DATABASES;
SELECT User, Host FROM mysql.user WHERE User = 'trading_app';
SHOW VARIABLES LIKE 'bind_address'; -- erwartet: 127.0.0.1
```

Schritt 3 — Python 3.14

Download: <https://www.python.org/downloads/>

- [] [Installer-GUI] „Add python.exe to PATH“ + „Install for all users“ aktivieren
- [] [PowerShell] — neues Fenster, prüfen: `python --version`, `pip --version`

Schritt 4 — PHP 8.3 (Thread-Safe)

Download: <https://windows.php.net/download/> → **PHP 8.3.x VS16 x64 Thread Safe (ZIP)**

- [] ZIP-Inhalt nach `C:\php` entpacken
- [] [PowerShell] `copy C:\php\php.ini-production C:\php\php.ini`
- [] [Text-Editor] `C:\php\php.ini` anpassen:



```
extension_dir = "C:/php/ext"
extension=curl
extension=fileinfo
extension=mbstring
extension=openssl
extension=pdo_mysql
extension=zip
extension=gd
upload_max_filesize = 64M
post_max_size = 64M
max_execution_time = 120
memory_limit = 256M
```

- [] **[Windows-Einstellungen]** `C:\php` zu System-PATH hinzufügen: Windows-Suche → „Umgebungsvariablen bearbeiten“ → **Systemvariablen** → **Path** → **Bearbeiten** → **Neu** → `C:\php` eintragen
- [] **[PowerShell]** — neues Fenster, prüfen: `php --version`, `php -m | findstr -i "pdo_mysql mbstring"`

PHP später aktualisieren (Sicherheitsupdates innerhalb von 8.3.x): gleicher Ablauf wie in Installation: Windows-Grundausstattung, Abschnitt „PHP aktualisieren, ohne auf ein neues XAMPP zu warten“ — nur mit `C:\php` statt `C:\xampp\php` und `C:\Apache24` statt `C:\xampp\apache`. `php.ini` bleibt beim Drüberkopieren erhalten. Da `C:\php` im System-PATH steht, findet Apache die Begleit-DLLs von curl in der Regel ohne zusätzliche `LoadFile`-Zeilen; nach dem Neustart trotzdem `C:\Apache24\logs\error.log` auf `Unable to load dynamic library` prüfen.

Schritt 5 — Apache HTTP Server

Voraussetzung: Visual C++ Redistributable 2015–2022 x64 (falls noch nicht vorhanden, bei Microsoft herunterladen und installieren).

Download: <https://www.apachelounge.com/download/> (aktueller Build, z. B. `httpd-2.4.x-win64-VS18.zip`). Die VC-Runtime ist rückwärtskompatibel zu PHPs VS16-Build, das passt problemlos zusammen.

- [] ZIP-Inhalt nach `C:\Apache24` entpacken
- [] **[Text-Editor]** `C:\Apache24\conf\httpd.conf` ergänzen (an den passenden Stellen, nicht die ganze Datei ersetzen):



```
LoadModule php_module "C:/php/php8apache2_4.dll"
AddHandler application/x-httpd-php .php
PHPIniDir "C:/php"

LoadModule rewrite_module modules/mod_rewrite.so

DirectoryIndex index.php index.html

Listen <server-vpn-ip>:80

<VirtualHost <server-vpn-ip>:80>
    DocumentRoot "C:/trading/TradingJournal/public"
    ServerName trading

    <Directory "C:/trading/TradingJournal/public">
        Options -Indexes +FollowSymLinks
        AllowOverride All
        Require all granted
    </Directory>

    ErrorLog "C:/Apache24/logs/trading-error.log"
    CustomLog "C:/Apache24/logs/trading-access.log" combined
</VirtualHost>
```

Wichtig: `<server-vpn-ip>` durch die tatsächliche VPN-Adresse des Servers ersetzen (zweimal — bei `Listen` und `<VirtualHost>`), sonst lauscht Apache auf eine nicht existierende Adresse.

[PowerShell – Administrator]:

```
C:\Apache24\bin\httpd.exe -k install -n "Apache2.4"
sc.exe config Apache2.4 start= delayed-auto
sc.exe failure Apache2.4 reset= 86400 actions=
restart/60000/restart/60000/restart/120000
net start Apache2.4
```

Warum verzögerter Start: Apache lauscht nur auf der VPN-Adresse. Beim Hochfahren des Servers ist das VPN oft noch nicht verbunden, wenn der normale Autostart Apache startet — die Adresse existiert dann noch nicht, Apache bricht ab (`could not bind to address ...:80` im Ereignisprotokoll) und bleibt gestoppt, bis ihn jemand von Hand startet. `delayed-auto` startet Apache erst nach den übrigen Autostart-Diensten, die Fehleraktionen (`sc.exe failure`) versuchen den Start bei Misserfolg nach 1, 1 und 2 Minuten erneut.

Firewall — nur das VPN-Netz erlauben (Beispiel für Tailscapes Standard-Adressbereich `100.64.0.0/10`, bei anderem VPN entsprechend anpassen) — **[PowerShell – Administrator]:**



```
New-NetFirewallRule -DisplayName "Trading HTTP (VPN only)" `
-Direction Inbound -LocalPort 80 -Protocol TCP `
-RemoteAddress 100.64.0.0/10 -Action Allow
```

Keine zusätzliche „Block“-Regel anlegen: Eine `Action Block`-Regel ohne eigene `-RemoteAddress`-Einschränkung würde *alle* Adressen auf Port 80 blocken, einschließlich des oben erlaubten VPN-Bereichs — Windows Firewall lässt eine passende Block-Regel immer gegen eine passende Allow-Regel gewinnen, unabhängig von der Reihenfolge. Unnötig ohnehin: Windows Firewall blockt standardmäßig bereits alles, was nicht explizit erlaubt ist.

- [] Apache-Service läuft, Autostart verzögert (`sc.exe qc Apache2.4` zeigt `AUTO_START (DELAYED)`)
- [] Test: Browser über die VPN-Adresse erreichbar; über die öffentliche IP des Servers **nicht** erreichbar (mit einem Gerät außerhalb des VPNs testen)

Schritt 6 — Composer

Download: <https://getcomposer.org/Composer-Setup.exe> — PHP-Pfad `C:\php\php.exe` wird automatisch erkannt. **[PowerShell]** prüfen: `composer --version`

Schritt 7 — Code deployen

[PowerShell]:

```
mkdir C:\trading
cd C:\trading

git clone https://github.com/mike-1140/TradingJournal-Weaver.git
git clone https://github.com/mike-1140/TradingJournal.git

mkdir C:\Trade_XML
mkdir C:\Trade_CSV
mkdir C:\Trade_XML\incoming_mail

cd C:\trading\TradingJournal-Weaver
python -m venv .venv
.venv\Scripts\activate
pip install -r requirements.txt
deactivate
```



Schritt 8 — .env -Dateien

`C:\trading\TradingJournal-Weaver\.env` — Vorlage `.env.example` kopieren und anpassen:

```
DB_USER=trading_app
DB_PASS=<db-passwort>
DB_HOST=127.0.0.1
DB_PORT=3306

DB_NAME_MONEY=ibkr_money
DB_NAME_PAPER=ibkr_paper
DB_NAME_AUTOM_MONEY=ibkr_autom_money
DB_NAME_AUTOM_PAPER=ibkr_autom_paper
```

Falls der Datenabruf (`get_flexquery_emails.py` / `get_flexquery_api.py`) weiterhin auf einem anderen Gerät läuft und nur die fertigen Dateien auf diesen Server übertragen werden (siehe Verteilter Abruf — minimaler Client für den Datenabruf), bleiben `IMAP_*` / `IBKR_FLEX_QUERIES` hier leer.

`C:\trading\TradingJournal\.env` — Vorlage `.env.example` kopieren und anpassen:

```
APP_NAME="Trading Journal"
APP_ENV=production
APP_DEBUG=false
APP_URL=http://<server-vpn-ip>

DB_CONNECTION=mysql
DB_HOST=127.0.0.1
DB_PORT=3306
DB_DATABASE=ibkr_laravel
DB_USERNAME=trading_app
DB_PASSWORD=<db-passwort>

TRADING_DB_HOST=127.0.0.1
TRADING_DB_PORT=3306
TRADING_DB_USERNAME=trading_app
TRADING_DB_PASSWORD=<db-passwort>

TRADING_DEFAULT_ENV=MONEY
```

`TRADING_DEFAULT_ENV` legt fest, in welchem Environment die App nach dem Login startet (ohne die Zeile: `TEST`). Auf dem Server ist `MONEY` meist die bessere Wahl. Hat der Benutzer im eingestellten Environment keine Tabellen — etwa weil `ibkr_test` gar nicht angelegt wurde —, startet die App



automatisch im ersten Environment, in dem er welche hat (`TradingHelper::getStartEnvironment()` , Reihenfolge wie in `config/trading.php`).

`APP_KEY` wird im nächsten Schritt generiert, nicht manuell eintragen.

Schritt 9 — Laravel-Grundgerüst

[PowerShell – Administrator] (die `icacls` -Befehle brauchen Admin-Rechte):

```
cd C:\trading\TradingJournal
composer install --no-dev --optimize-autoloader
php artisan key:generate

icacls "C:\trading\TradingJournal\storage" /grant "SYSTEM:(OI)(CI)F" /T
icacls "C:\trading\TradingJournal\bootstrap\cache" /grant "SYSTEM:(OI)(CI)F" /T
```

Frontend-Build (Node.js/npm) — sonst „Vite manifest not found“

TradingJournal nutzt Vite (`@vite` -Direktiven in den Blade-Views) für kompiliertes CSS/JS — ohne Build fehlt `public/build/manifest.json` und jede Seite mit `@vite` (inkl. Login) liefert einen 500er.

1. Download: <https://nodejs.org/> (aktuelle LTS-Version, Windows Installer), Standardoptionen.
2. [PowerShell] — neues Fenster, prüfen: `node --version` , `npm --version`
3. [PowerShell]:

```
cd C:\trading\TradingJournal
npm install
npm run build
```

Erzeugt `public/build/manifest.json` + die kompilierten Assets.

Schritt 10 — Datenbank einrichten



Option A — frisches, leeres Journal:

```
cd C:\trading\TradingJournal-Weaver
.venv\Scripts\activate
python -m tools.maintenance.reset_IBKR_DB money dein_user_
# pro benötigtem Environment wiederholen (paper/autom_money/autom_paper)
deactivate

cd C:\trading\TradingJournal
php artisan migrate --force
php artisan setup:auth
php artisan auth:create-user "Dein Name" dein_name@example.com "PASSWORT" --
prefix=dein_user_ --admin
```

Option B — bestehende Daten von einem lokalen Rechner übernehmen:

Auf dem **lokalen Rechner** (nicht auf dem Server), frische Dumps erzeugen — `mysqldump` s eigenes `--result-file` -Flag nutzen, **nicht** PowerShells `>` -Redirection: Windows PowerShell 5.1s `>` (= `Out-File` ohne `-Encoding`) schreibt standardmäßig UTF-16, was die Textausgabe mit Null-Bytes zwischen jedem Zeichen zerschießt (`mysql` meldet beim Restore dann `ASCII '\0' appeared in the statement`):

```
& "<pfad-zu-mysqldump>" -u root --result-file=ibkr_laravel.sql ibkr_laravel
& "<pfad-zu-mysqldump>" -u root --result-file=ibkr_money.sql ibkr_money
# ... für jede weitere Environment-DB
```

Dumps zum Server übertragen (z. B. über dein Remote-Desktop-Tool, nicht über die Zwischenablage — für größere Dateien unzuverlässig). Auf dem **Server** einspielen — auch hier: PowerShell kennt den `<` -Umleitungsoperator nicht (`RedirectionNotSupported`), deshalb über `cmd /c` mit einfachen Anführungszeichen außen:

```
cmd /c '"C:\Program Files\MySQL\MySQL Server 8.4\bin\mysql.exe" -h 127.0.0.1 -u
trading_app -p ibkr_laravel < ibkr_laravel.sql'
cmd /c '"C:\Program Files\MySQL\MySQL Server 8.4\bin\mysql.exe" -h 127.0.0.1 -u
trading_app -p ibkr_money < ibkr_money.sql'
# ... für jede weitere Environment-DB
```

Damit sind Schema, Daten und echte Logins (`ibkr_laravel.users`) in einem Schritt übernommen — keine separate `auth:create-user` /Schema-Anlage nötig. Danach trotzdem als Sicherheitsnetz (überspringt bereits gelaufene Migrationen):



```
cd C:\trading\TradingJournal
php artisan migrate --force
php artisan config:cache
php artisan route:cache
php artisan view:cache
net stop Apache2.4 && net start Apache2.4
```

- [] Login-Seite erscheint unter `http://<server-vpn-ip>/login`
- [] Login funktioniert

Schritt 11 — `pipeline.toml` + `accounts.toml`

[PowerShell]:

```
cd C:\trading\TradingJournal-Weaver
copy pipeline.toml.example pipeline.toml
```

Wichtig — mitkopierten `[[test]]` -Block entfernen: `pipeline.toml.example` liefert standardmäßig einen aktiven (nicht auskommentierten) Block:

```
[[test]]
user = "dein_user"
```

Ein `[[test]]` -Eintrag ist laut `src/config.py::load_active_targets()` immer aktiv — kein Enabled-Schalter, zum Deaktivieren muss der Eintrag komplett aus der Datei entfernt werden. Bleibt er stehen und existiert keine `ibkr_test` -Datenbank, bricht die Pipeline mit `Access denied ... to database 'ibkr_test'` ab.

Eigene User/Environments eintragen:

```
[settings]
steps = ["import", "process", "summary"]

[users.dein_user]
environments = ["money", "paper"]
```

(`fetch_emails` / `fetch_api` / `snapshot_csv` nur ergänzen, falls der Datenabruf auch auf diesem Server laufen soll — sonst reichen `import` / `process` / `summary` , wenn die Rohdaten von woanders hierher übertragen werden, siehe Verteilter Abruf — minimaler Client für den Datenabruf.)



Zusätzlich nötig: `accounts.toml` — das accountId→User/Environment-Mapping (`src/config.py::load_account_map()`), eine eigene, gitignored Konfigurationsdatei neben `.env` und `pipeline.toml`. Ohne sie meldet der Import „Unbekannte accountId ... übersprungen“ für jede Datei, nichts wird importiert. Vorlage `accounts.toml.example` kopieren und mit den eigenen IBKR-accountId-Suffixen befüllen (siehe Kommentare in der Vorlage).

Schritt 12 — Abschlusstest

- [] `services.msc` : `MySQL84` → Gestartet, Autostart; `Apache2.4` → Gestartet, Automatisch (Verzögerter Start)
- [] Server einmal neu starten: Apache kommt ohne Eingriff wieder hoch
- [] Browser über die VPN-Adresse: `http://<server-vpn-ip>/login` erreichbar
- [] Login funktioniert, zeigt die erwarteten Daten
- [] Von einem Gerät **ohne** VPN-Verbindung: Server über seine öffentliche IP → keine Verbindung (Firewall greift)
- [] Ein Journal-Eintrag über die UI anlegen → wird korrekt gespeichert



Troubleshooting

Problem	Ursache	Lösung
MySQL-Client-Fenster (Startmenü-Verknüpfung) schließt sich sofort nach Passwort-Eingabe, keine Fehlermeldung	Falsches Passwort — die Verknüpfung schließt bei Fehler kommentarlos	Stattdessen über PowerShell verbinden (<code>&</code> <code>"... \mysql.exe"</code> <code>-u root -p</code>), Fehlermeldung bleibt dort stehen
<code>Access denied for user 'root'@'localhost'</code> trotz korrektem Passwort in einer Remote-Sitzung	Tastaturlayout-Mismatch zwischen lokalem Gerät und Server (Sonderzeichen im Passwort werden anders übertragen)	Tastaturlayout abgleichen, oder Passwort ohne Sonderzeichen setzen
<code>mysqld --init-file=...</code> bricht ab mit <code>Failed to set datadir ... No such file or directory</code>	<code>mysqld.exe</code> ohne <code>--defaults-file</code> gestartet, sucht Datenverzeichnis relativ zum <code>bin</code> - Ordner statt in <code>C:\ProgramData\MySQL\...</code>	<code>--defaults-file="C:\ProgramData\MySQL\MySQL Server 8.4\my.ini"</code> mit angeben
<code>Access denied for user 'trading_app'@'localhost'</code> trotz <code>-h 127.0.0.1</code> am Client	User wurde mit Host <code>127.0.0.1</code> statt <code>%</code> angelegt — MySQL löst die Verbindung serverseitig zu <code>localhost</code> auf	User mit Host <code>%</code> neu anlegen (siehe Schritt 2)
<code>mysqldump</code> -Restore scheitert mit <code>ASCII '\0' appeared in the statement</code>	Dump wurde über PowerShells <code>></code> - Operator erzeugt (schreibt UTF-16 statt UTF-8)	Dump neu erzeugen mit <code>--result-file=</code> statt <code>></code>
PowerShell meldet <code>Unerwartetes Token</code> bei <code>"pfad\programm.exe" -argument</code>	Ein in Anführungszeichen gesetzter Pfad wird ohne Call-Operator nur als String interpretiert	<code>&</code> vor den Pfad setzen: <code>&</code> <code>"pfad\programm.exe" -argument</code>
PowerShell meldet <code>Der Operator "<" ist für zukünftige Versionen reserviert</code>		



Problem	Ursache	Lösung
MySQL-Client-Fenster (Startmenü-Verknüpfung) schließt sich sofort nach Passwort-Eingabe, keine Fehlermeldung	Falsches Passwort — die Verknüpfung schließt bei Fehler kommentarlos	Stattdessen über PowerShell verbinden (<code>&</code> <code>"...\mysql.exe" -u root -p</code>), Fehlermeldung bleibt dort stehen
<code>Access denied for user 'root'@'localhost'</code> trotz korrektem Passwort in einer Remote-Sitzung	Tastaturlayout-Mismatch zwischen lokalem Gerät und Server (Sonderzeichen im Passwort werden anders übertragen)	Tastaturlayout abgleichen, oder Passwort ohne Sonderzeichen setzen
<code>mysqld --init-file=...</code> bricht ab mit <code>Failed to set datadir ... No such file or directory</code>	<code>mysqld.exe</code> ohne <code>--defaults-file</code> gestartet, sucht Datenverzeichnis relativ zum <code>bin</code> -Ordner statt in <code>C:\ProgramData\MySQL\...</code>	<code>--defaults-file="C:\ProgramData\MySQL\Server 8.4\my.ini"</code> mit angeben
<code>Access denied for user 'trading_app'@'localhost'</code> trotz <code>-h 127.0.0.1</code> am Client	User wurde mit Host <code>127.0.0.1</code> statt <code>%</code> angelegt — MySQL löst die Verbindung serverseitig zu <code>localhost</code> auf	User mit Host <code>%</code> neu anlegen (siehe Schritt 2)
<code>mysqldump</code> -Restore scheitert mit <code>ASCII '\0' appeared in the statement</code>	Dump wurde über PowerShells <code>></code> -Operator erzeugt (schreibt UTF-16 statt UTF-8)	Dump neu erzeugen mit <code>--result-file=</code> statt <code>></code>
PowerShell meldet <code>Unerwartetes Token</code> bei <code>"pfad\programm.exe" -argument</code>	Ein in Anführungszeichen gesetzter Pfad wird ohne Call-Operator nur als String interpretiert	<code>&</code> vor den Pfad setzen: <code>&</code> <code>"pfad\programm.exe" -argument</code>



PowerShell meldet Der Operator "<" ist für zukünftige Versionen reserviert	PowerShell kennt keine < - Umleitung (im Gegensatz zu cmd/bash)	Über <code>cmd /c</code> <code>"pfad\programm.exe</code> <code>" ... < datei'</code> ausführen (einfache Anführungszeichen außen)
Seite liefert 500 mit „Vite manifest not found“	Kein Frontend- Build durchgeführt	<code>npm install && npm</code> <code>run build</code> im TradingJournal- Ordner
„Diese Seite funktioniert nicht — zu oft weitergeleitet“ nach Login	Meist eine fehlgeschlagene DB-Verbindung, die <code>bootstrap/app.p</code> <code>hp</code> ’s Exception- Handler abfängt und zu einem Redirect-Loop mit <code>/login</code> führt	Laravel-Log prüfen (<code>storage/logs/larav</code> <code>el.log</code>), nach <code>\]\s+\w+\.ERROR:</code> filtern) — üblicher Auslöser: <code>TRADING_DB_USERNAM</code> <code>E / _PASSWORD</code> falsch/fehlend, oder der Benutzer hat in keinem Environment Tabellen (Prefix in der <code>users</code> -Tabelle prüfen, <code>reset_IBKR_DB.py</code> für diesen Prefix gelaufen?)
<code>run_pipeline.py</code> meldet „Unbekannte accountId ... übersprungen“ für jede Datei	<code>accounts.toml</code> fehlt oder ist unvollständig	Siehe Schritt 11
<code>run_pipeline.py</code> verarbeitet unerwartet auch ein TEST- Environment	Der mitkopierte <code>[[test]]</code> -Block aus <code>pipeline.toml.e</code> <code>xample</code> wurde nicht entfernt	Siehe Schritt 11

Verteilter Abruf — minimaler Client für den Datenabruf

Nur relevant, wenn du bereits Server-Betrieb — TradingJournal auf einem eigenen Server umgesetzt hast und zusätzlich von einem zweiten, leichtgewichtigen Gerät (Reise-Notebook, Tablet mit Windows, etc.) aus IBKR-Daten abrufen und zum Server übertragen willst — **ohne** auf diesem zweiten Gerät die komplette App (MySQL/PHP/Apache/Node.js) zu installieren.

Typisches Szenario: Dein Hauptrechner (der sonst Mail-Abruf + Import erledigt) ist unterwegs nicht verfügbar oder ausgeschaltet. Der Server läuft aber weiter und soll trotzdem täglich neue Daten bekommen.

Warum das minimal bleiben kann

`get_flexquery_emails.py` / `get_flexquery_api.py` laden nur Rohdaten (XML/CSV) in einen lokalen Ordner herunter — sie brauchen **keine** Datenbankverbindung und kein `accounts.toml` / `pipeline.toml`. Das komplette User/Environment-Routing passiert erst beim `import`-Step, der auf dem Server läuft. Der Client braucht deshalb nur:

- Python 3.14
- Zwei Pakete: `python-dotenv`, `requests` (**nicht** die volle `requirements.txt` — die zieht zusätzlich `pandas` / `sqlalchemy` / `pymysql`, für reines Abrufen unnötig)
- Eine `.env` mit den IMAP-/API-Zugangsdaten
- Einen SSH-Client, um die Dateien zum Server zu übertragen

Kein MySQL, kein PHP, kein Apache, kein Node.js.

Server-Seite: SSH-Empfang einrichten (einmalig)

Auf dem Server, der bereits nach „Server-Betrieb — TradingJournal auf einem eigenen Server“ eingerichtet ist — **[PowerShell – Administrator]**:



```
Add-WindowsCapability -Online -Name OpenSSH.Server~~~~0.0.1.0
Start-Service sshd
Set-Service -Name sshd -StartupType Automatic

New-NetFirewallRule -DisplayName "SSH (VPN only)" `
  -Direction Inbound -LocalPort 22 -Protocol TCP `
  -RemoteAddress 100.64.0.0/10 -Action Allow
```

(Adressbereich `100.64.0.0/10` ist Tailscapes Standard — bei anderem VPN entsprechend anpassen.)

Client-Seite: Minimal-Installation

1. Python 3.14

Download: <https://www.python.org/downloads/> — „Add python.exe to PATH“ + „Install for all users“ aktivieren.

2. Git for Windows

Download: <https://git-scm.com/install/windows> („Click here to download“), Standardoptionen.

3. Nur TradingJournal-Weaver klonen

```
mkdir C:\trading
cd C:\trading
git clone https://github.com/mike-1140/TradingJournal-Weaver.git
```

4. Minimal-Dependencies

```
cd C:\trading\TradingJournal-Weaver
python -m venv .env
.\env\Scripts\python.exe -m pip install python-dotenv requests
```

Der venv-Python wird bewusst direkt über seinen Pfad aufgerufen statt über `activate` — das funktioniert auch ohne Skript-Ausführungsrechte und ohne Aktivierung.



5. .env anlegen

```
IMAP_USERNAME=deine@email.at
IMAP_PASSWORD=DEIN_MAIL_PASSWORT
IMAP_SERVER=imap.deinanbieter.de

# Optional – zweiter, gleichwertiger Kanal falls der Mail-Abruf mal nicht geht:
IBKR_FLEX_QUERIES=DEIN_TOKEN:QUERY_ID_1,DEIN_TOKEN:QUERY_ID_2, ...
```

`C:\Trade_XML\incoming_mail\` und `C:\Trade_CSV\incoming\` werden von den Skripten automatisch angelegt, kein manueller Schritt nötig.

Für die Fetch-Skripte reichen nur die IMAP-Zeilen und/oder `IBKR_FLEX_QUERIES` — keine Datenbank-Zugangsdaten.

Achtung: Der Mail-Abruf löscht die IBKR-Mails nach dem Herunterladen im Postfach. Ein „Testlauf“ ist deshalb ein echter Lauf — die XMLs liegen danach nur lokal in `C:\Trade_XML\incoming_mail` und müssen mit dem Push-Skript (Schritt 7) zum Server.

Vorsicht beim Flex-Query-API-Kanal: Wiederholte fehlgeschlagene Abrufversuche können bei IBKR zu einer temporären Sperre führen („Too many failed attempts“). Nicht wiederholt manuell testen, falls ein Abruf einmal fehlschlägt — meist löst sich das von selbst innerhalb von Stunden.

6. SSH-Client + eigener Schlüssel

Meist bereits vorinstalliert (Windows 10/11). Prüfen: `ssh -V`. Falls nicht vorhanden: Windows-Einstellungen → Apps → Optionale Features → „OpenSSH-Client“.

```
ssh-keygen -t ed25519 -f %env:USERPROFILE\.ssh\fetch_client
```

Public Key zum Server übertragen und dort hinterlegen. **Wichtig bei Administrator-Konten unter Windows:** Falls das Server-Konto Mitglied der Administrators-Gruppe ist (prüfen mit `whoami /groups` | `findstr "Administrators"` auf dem Server), greift **nicht** `~\.ssh\authorized_keys`, sondern eine zentrale Datei: `C:\ProgramData\ssh\administrators_authorized_keys`.

Bei Admin-Konto:

```
# Auf dem Server, nach Übertragung von fetch_client.pub:
$key = Get-Content C:\temp\fetch_client.pub
Add-Content -Path C:\ProgramData\ssh\administrators_authorized_keys -Value $key

icacls C:\ProgramData\ssh\administrators_authorized_keys /inheritance:r
icacls C:\ProgramData\ssh\administrators_authorized_keys /grant "SYSTEM:F"
icacls C:\ProgramData\ssh\administrators_authorized_keys /grant "Administrators:F"
```



Die `icacls` -Befehle sind zwingend — `sshd` verweigert den Key sonst mit „Bad owner or permissions“ bei zu offenen (geerbten) Dateirechten. Mehrere Clients können denselben Server nutzen: einfach jeden weiteren Public Key zusätzlich anhängen (`Add-Content`), nicht überschreiben.

Bei normalem (Nicht-Admin-)Konto:

```
Get-Content $env:USERPROFILE\.ssh\fetch_client.pub | ssh <konto>@<server-vpn-ip> "cat >>
~/.ssh/authorized_keys"
```

Fragt beim ersten Mal noch nach dem Windows-Passwort (Key-Login ist ja noch nicht eingerichtet) — danach klappt's per Key.

Kontrolle: `ssh-keygen -lf <pfad-zur-datei>` auf beiden Seiten, Fingerprints müssen übereinstimmen.

7. Push-Skript

Hinweise für Windows PowerShell 5.1:

- Das Skript darf in den Zeichenketten **nur ASCII** enthalten (kein Gedankenstrich `—`, keine typografischen Anführungszeichen). Eine ohne BOM gespeicherte Datei liest PowerShell 5.1 als ANSI; ein `—` wird zu `"` und beendet den String vorzeitig. Die Fehlermeldungen (`&&` „kein gültiges Anweisungstrennzeichen“, „kein Abschlusszeichen“, „`}` fehlt“) zeigen dann auf andere Stellen als die eigentliche Ursache.
- Mit Notepad bearbeiten (nicht WordPad/Word) und alten Inhalt vollständig ersetzen. Speicherort in diesem Beispiel: `C:\trading\push-to-server.ps1`.
- Auf frischen Windows-Rechnern einmalig erlauben: `Set-ExecutionPolicy -Scope CurrentUser RemoteSigned`.

```
$ErrorActionPreference = "Stop"

$ServerIp      = "<server-vpn-ip>"
$ServerUser    = "<konto>"
$KeyPath       = "$env:USERPROFILE\.ssh\fetch_client"
$RemoteTarget  = "$ServerUser@$ServerIp"

# Lokaler Ordner → Ordner auf dem Server (ohne abschliessenden Slash).
# incoming_mail = get_flexquery_emails, incoming_api = get_flexquery_api (XML, die
# Pipeline
# liest beide); Trade_CSV\incoming = Kontoauszuege (CSV) aus get_flexquery_emails.
# Pushed = lokale Ablage: erfolgreich uebertragene Dateien wandern dorthin, damit sie
# beim
# naechsten Lauf nicht erneut uebertragen werden.
$Folders = @(
    @{ Local = "C:\Trade_XML\incoming_mail"; Remote = "C:/Trade_XML/incoming_mail";
      Filter = "*.xml"; Pushed = "C:\Trade_XML\pushed\incoming_mail" },
```



```
@{ Local = "C:\Trade_XML\incoming_api"; Remote = "C:/Trade_XML/incoming_api"; Filter =
"*.xml"; Pushed = "C:\Trade_XML\pushed\incoming_api" },
    @{ Local = "C:\Trade_CSV\incoming"; Remote = "C:/Trade_CSV/incoming";
Filter = "*.csv"; Pushed = "C:\Trade_CSV\pushed" }
)

function Assert-LastExit($step) {
    # PowerShell faengt Exit-Codes externer Tools (scp/ssh) mit
    # $ErrorActionPreference="Stop" nicht automatisch ab → manuell pruefen.
    if ($LASTEXITCODE -ne 0) {
        Write-Host "FEHLER bei '$step' (Exit-Code $LASTEXITCODE)" -ForegroundColor Red
        exit 1
    }
}

$total = 0
foreach ($f in $Folders) {
    # Nur Dateien direkt im Ordner (keine Unterordner) - genau diese werden uebertragen
    und danach verschoben
    $files = @(Get-ChildItem -Path $f.Local -Filter $f.Filter -File -ErrorAction
SilentlyContinue)
    if ($files.Count -eq 0) {
        Write-Host "Keine Dateien ($($f.Filter)) in $($f.Local)"
        continue
    }
    $total += $files.Count
    Write-Host "Uebertrage $($files.Count) Datei(en) aus $($f.Local) ..."

    # Zielordner auf dem Server anlegen, falls er fehlt (incoming_api existiert dort
    evtl. noch nicht)
    $remoteWin = $f.Remote.Replace("/", "\")
    ssh -i $KeyPath $RemoteTarget "if not exist $remoteWin mkdir $remoteWin"
    Assert-LastExit "Zielordner anlegen (Server)"

    scp -i $KeyPath $files.FullName "${RemoteTarget}:${($f.Remote)}/"
    Assert-LastExit "scp"

    # Erst nach erfolgreicher Uebertragung lokal in die Ablage verschieben (nicht
    loeschen)
    New-Item -ItemType Directory -Force -Path $f.Pushed | Out-Null
    $files | Move-Item -Destination $f.Pushed -Force
}

if ($total -eq 0) {
    Write-Host "Keine Dateien gefunden - nichts zu tun."
    exit 0
}
```



```
}  
  
ssh -i $KeyPath $RemoteTarget "cd C:\trading\TradingJournal-Weaver &&  
.venv\Scripts\python run_pipeline.py"  
Assert-LastExit "run_pipeline.py (Server)"  
  
Write-Host "Fertig." -ForegroundColor Green
```

Bricht bei jedem Fehler mit klarer Meldung ab (Exit-Code-Check), statt stillschweigend weiterzulaufen.

Übertragene Dateien wandern lokal in die Ablage (`C:\Trade_XML\pushed\...` bzw. `C:\Trade_CSV\pushed`), und zwar erst, nachdem `scp` für diesen Ordner erfolgreich war. Ohne diesen Schritt blieben sie in den Eingangsordnern liegen und würden bei jedem Lauf erneut übertragen und auf dem Server erneut eingelesen — der Duplikat-Schutz verhindert zwar doppelte Trades, aber die Datenmenge wächst mit jedem Lauf. Die Ablage kann gelegentlich geleert werden; die Daten liegen auf dem Server in `processed`.

Kontoauszüge (CSV) werden mit übertragen, sonst fehlen auf dem Server die Kontostände der Tage, an denen dieser Rechner die Mails abgeholt hat (der Mail-Abruf löscht die Mails, der Server bekommt sie also nicht mehr selbst).

Täglicher Ablauf

```
cd C:\trading\TradingJournal-Weaver  
.\.venv\Scripts\python.exe -m src.get_flexquery_emails  
  
cd C:\trading  
.\push-to-server.ps1 # oder wie das Skript aus Schritt 7 benannt wurde
```

Bekannte Grenzen und offene Fragen

Dieses Kapitel sammelt ehrlich, was `TradingJournal-Weaver` und TradingJournal **nicht** oder **noch nicht sicher** können. Es hat drei Teile:

- 1 **Bekannte Grenzen** — im Code nachgewiesen, teils auch in echten Daten beobachtet.
- 2 **Ungetestet — Beispiele gesucht** — Fälle, für die der Code vermutlich passt, für die aber keine echten Daten vorliegen.
- 3 **Mithelfen** — wie du mit einem eigenen Beispiel helfen kannst.

Jeder Eintrag verweist auf die Stelle im Handbuch, an der der Ablauf ausführlich beschrieben ist. Wird eine Grenze behoben, verschwindet sie aus diesem Kapitel.

Hinweis

Geprüft gegen den Code und gegen die echten Daten des Autors (mehrere Echtgeld- und Paper-Konten). „Schon vorgekommen“ bezieht sich auf diese Daten.

1. Bekannte Grenzen

1.1 Import und Verarbeitung

Andere Kapitalmaßnahmen werden nicht eigens behandelt

Nur Splits (`RS` , `FS` , `SS`) haben eine eigene Logik. Fusionen, Spin-offs, Symbolwechsel usw. würden wie normale Trades zum Preis 0 verarbeitet und Positionen sowie Gewinn/Verlust verfälschen. *Schon vorgekommen*: nein. *Details*: Architektur und Import-Pipeline, § 6.2 „Fall SPLIT“.

Kein Forex-Trading

Devisentausch wird nur als Umtausch festgehalten (Ansicht *FX-Trades*). Währungspositionen mit Gewinn/Verlust aus Kursbewegungen gibt es nicht. *Details*: Architektur und Import-Pipeline, § 5.3.

Cash im Kontostand nur in EUR und USD



Die Kontostände aus dem Kontoauszug kennen genau zwei Währungen; Cash in einer dritten Währung wird stillschweigend nicht erfasst. *Details: Architektur und Import-Pipeline, § 8.2*

„`account_snapshots`“.

1.2 Optionen: Verfall, Ausübung, Zuteilung

Ausübung eigener Optionen ohne Verknüpfung zur Aktie

Bei einer *Zuteilung* werden Option und gelieferte Aktien über eine gemeinsame Kennung verknüpft (für die Spreads-Auswertung und die Detailseiten). Bei einer *Ausübung* eigener Long-Optionen fehlt diese Verknüpfung. *Details: Architektur und Import-Pipeline, § 5.5.*

1.3 Splits

Optionen auf eine gesplittete Aktie werden nicht umgestellt

Der Split wird erkannt, offene Options-Positionen bleiben aber unter dem alten Kontrakt stehen. Ein späterer Verfall oder Rückkauf unter dem neuen Kontrakt eröffnet dann eine falsche Gegenposition. Kapitalmaßnahmen mit mehr als zwei Zeilen (mehrere Strikes) passen außerdem nicht ins Paar-Schema der Split-Verarbeitung. *Schon vorgekommen: einmal, bei einem Forward Split mit offenem Spread; zu Reverse Splits mit offener Option fehlen noch Daten. Details: Architektur und Import-Pipeline, § 6.2 „Fall SPLIT“.*

1.4 Spread-Erkennung

Rolls werden als Spread erkannt

Ein Roll als Combo-Order (alte Option zurückkaufen, neue verkaufen) sieht aus wie ein Calendar- oder Diagonal-Spread. Die neu verkaufte Option erscheint dann in der Spreads-Ansicht als „Spread“ mit nur einem Leg. *Schon vorgekommen: ja. Details: Architektur und Import-Pipeline, § 5.4.*

Schließende Combo-Orders heißen „umgekehrt“

Wer einen Bull Put Spread per Combo-Order zurückkauft, sieht die Schließung in *Raw Data* als „Bear Put Spread“. Die Spreads-Auswertung ist davon nicht betroffen, sie verwendet die Strategie der Eröffnung. *Schon vorgekommen: ja, regelmäßig. Details: wie oben.*

Nicht erkannte Strategien lassen sich nicht von Hand zusammenfassen

Erkannt wird nur, was als Combo-Order oder innerhalb derselben 3 Sekunden ausgeführt wurde. Einzelne mit Abstand eingegebene Legs, Legs an verschiedenen Tagen und Ratio Spreads bleiben Einzeloptionen — auch wenn man selbst weiß, dass sie zusammengehören. Eine manuelle Zuordnung ist geplant; schwierig sind dabei vor allem Calendar-Spreads, deren Legs zu verschiedenen Terminen enden. Ein Iron Butterfly heißt außerdem „Iron Condor“. *Details: Architektur und Import-Pipeline, § 5.4.*

1.5 Vorgaben von IBKR

Keine Grenzen des Journals, aber wichtig zu wissen:



- Eine Flex Query lässt sich je Abruf für höchstens **365 Tage** starten.
- **Paper-Konten** liefern nur die letzten rund **180 Tage** — Trades wie Kontoauszüge.
- **Kontostände** gibt es nur aus dem täglichen Kontoauszug; rückwirkend nur Tag für Tag von Hand.
- Die **Web-API** verträgt keine häufigen Abrufe; zu viele (fehlgeschlagene) Anfragen führen zu einer vorübergehenden Sperre.

Details: Benutzerhandbuch TradingJournal-Weaver installieren, Abschnitt 8, und Alltag mit dem TradingJournal, Abschnitt 1.

2. Ungetestet — Beispiele gesucht

Für einige Fälle gibt es im Code eine Behandlung oder eine naheliegende Erwartung, aber keine echten Daten, an denen sie geprüft wurde — etwa ein Aktiensplit während einer offenen Option, Forward Splits, Teil- und vorzeitige Zuteilungen, die Ausübung eigener Optionen, Fusionen und Spin-offs, Futures oder ein Konto mit anderer Basiswährung als EUR.

Die vollständige Liste mit dem, was jeweils schon bekannt ist, konkreten Fragen und den hilfreichen Daten steht in einem eigenen Dokument im Repository: [Beispiele gesucht](#). Was davon als Nächstes umgesetzt wird, zeigt die [Roadmap](#).

3. Mithelfen

Wer einen der Fälle aus [Beispiele gesucht](#) im eigenen Konto hat, kann sehr helfen — mit der **Flex-Query-XML** des betreffenden Tages:

1. Im Kundenportal die Flex Query für den Tag des Ereignisses starten (siehe Benutzerhandbuch TradingJournal-Weaver installieren, Abschnitt 8) oder die Datei aus `C:\Trade_XML\<environment>\<benutzer>\processed` nehmen.
2. **Anonymisieren:** In einem Text-Editor die **Kontonummer** überall ersetzen (z. B. durch `U1234567` bzw. bei Paper `DU1234567`) — sie steht im Kopf jedes Tagesberichts und an jedem Trade. Steht ein Konto-Pseudonym oder Name in der Datei, auch diesen ersetzen. Trade-IDs, Kurse und Mengen bitte lassen, sonst ist der Fall nicht nachvollziehbar.
3. Den Fall melden: kurz beschreiben, was passiert ist (z. B. „Reverse Split 1:10 auf XYZ während offener Short-Put-Position“), was das Journal anzeigt und was richtig wäre — die anonymisierte Datei anhängen. Am besten im **Forum**, in dem das TradingJournal vorgestellt wurde; wiederkehrende Fragen fließen dort in eine FAQ ein. Wer ein GitHub-Konto hat, kann stattdessen im Repository `TradingJournal-Weaver` ein Issue mit der Vorlage „[Beispiel für ungetesteten Fall](#)“ anlegen — sie fragt Fall-Nummer, Ablauf und die Anonymisierung ab.



Vorsicht

Die Flex-Query-XML enthält vollständige Handelsdaten. Vor dem Hochladen prüfen, dass die Kontonummer wirklich überall ersetzt ist (Suchen nach den letzten vier Ziffern). Prüfcodes (Tokens) stehen in der XML nicht, gehören aber auch sonst nie in einen Forenbeitrag oder ein Issue.