GUI-TRADE-EXPORT-1 Plan-Review

TradeLogs JSON/XML Export aus Filament

Datum: 2026-05-14 (UTC) | Master HEAD: `31ef276` | Phase: P1-Backlog

PLAN-REVIEW **READ-ONLY** **GO IMPLEMENTATION möglich** **3 Operator-Decisions**

> **Kurzfassung.** 49 TradeLog-Rows, 30 Spalten, 13 Indexe, 272 KB total — extrem klein. **metadata_json** enthält 16 unkritische Schlüssel (entry_price, pnl, order_id, ...), **0 Sensitive-Pattern-Treffer** (live geprüft). Filament 5 native `ExportAction` unterstützt nur CSV/XLSX → für JSON/XML brauchst du eine Custom-Action. **Empfehlung**: dünne Custom-Action in `ListTradeLogs::getHeaderActions()` + ein `TradeLogExporter`-Service mit Streaming-Response. **Admin-Gate fehlt aktuell** in der Resource — muss als Pre-Condition mit `canAccess()` nachgezogen werden.

1. Boundary-Snapshot

Item	Wert
master HEAD	`31ef276` (lokal)
Bot/Worker/GUI	alle running, healthy
Bot main.py PID	195
cmd 13 / mp / history	cancelled / 0 / 0
BINANCE_TESTNET	true
Worker Heartbeat age	9 s

2. trade_logs Live-State

Metrik	Wert
Row Count	**49**
Table-Size	32 KB
Total-Size (mit Indexes)	272 KB
Date-Range	2026-05-08 07:11 → 2026-05-14 05:08
Sensitive-Pattern in metadata_json (api_key|api_secret|token|password|dsn|authorization|secret)	**0 Rows**

49 Rows = vollständiger Export passt locker in einen HTTP-Response. Streaming ist nicht zwingend wegen Größe, aber empfohlen wegen **Speicher-Hygiene** (verhindert Disk-Spool und Memory-Doppel-Buffering).

3. trade_logs Schema (DB-IST) vs Operator-Wunschfelder

Operator-Wunsch	DB-Spalte	Bemerkung
id	`id` bigint	direkt
timestamp / created_at	`created_at`, `opened_at`, `closed_at`	3 Zeitstempel im Export anbieten
symbol	`symbol`	direkt
side / action	**NICHT direkt im Schema**	TradeLog modelliert *geschlossene Round-Trips*, kein einzelnes Order-Side. Empfehlung: nicht im Export — verwirrend.
quantity	`quantity` numeric(24,12)	direkt
price	`entry_price` + `exit_price`	beide exportieren
order_value	`position_value`	direkt
fee	`fees`	direkt
pnl	`realized_pnl`	direkt
pnl_pct	`realized_pnl_pct`	direkt
status	`status`	direkt
strategy	`strategy_id`	direkt
subkind	`strategy_group`	Mapping: subkind = strategy_group (t1_core / t2_pump_dump / t3_copy_trading / legacy_unknown)
decision_log_id	`decision_id`	direkt — FK auf decision_logs
order_id	nur in `metadata_json.order_id`	Ausziehen: `metadata_json->'order_id'`
exchange	nicht direkt — `source` oder `mode`	Empfehlung: `source` (z.B. 'binance_testnet')
environment	`environment`	direkt (testnet/mainnet)
reason	`exit_reason`	direkt (take_profit/stop_loss/manual_close/...)

metadata_json sanitized	`metadata_json`	siehe §5 Sanitizer

Zusätzliche Felder, die der Operator nicht erwähnt, aber im Schema sind: `trade_id` (UNIQUE-Constraint, sollte rein!), `position_id`, `base_asset`, `quote_asset`, `profile_id`, `slippage`, `stop_loss`, `take_profit`, `duration_seconds`, `mode`, `updated_at`

> **Empfehlung Datenfelder**: Operator-Liste 1:1 + zusätzlich `trade_id` (UNIQUE) + `position_id` + `strategy_group` + `opened_at` / `closed_at` / `duration_seconds` als Composite + `base_asset` / `quote_asset` (für Reporting-Splits). Insgesamt ~25 Felder.

4. Existing TradeLogResource — Inspect

Item	IST	SOLL für Export
`canAccess()`	**NICHT VORHANDEN** — aktuell alle authentifizierten User	Admin-Gate fehlt → in TEX-Implementation nachziehen analog ManagedProposalResource
`canCreate()`	false ✓	OK
Filter-Inventory	symbol, strategy_id, strategy_group, exit_reason, environment, winners_only, losers_only	Modal kann „aktuelle Filter übernehmen" via `$livewire->tableFilters`-Zugriff
Header Actions	`getHeaderActions(): []`	Hier kommt die Export-Action rein
Bulk Actions	`bulkActions([])`	Optional: Bulk-Export für ausgewählte Rows
Pagination	[25, 50, 100]	Export ignoriert Pagination

Filament 5 native ExportAction

> Filament's eingebaute `ExportAction` unterstützt **nur CSV + XLSX** (`ExportFormat`-Enum). Sie nutzt asynchrone Job-Batches + Database-Notifications + eine `exports`-Tabelle (DB-Migration nötig). Für 49 Rows und JSON/XML wäre das schwergewichtig und nicht passend. → **Custom-Action ist der bessere Weg**.

5. Sanitizing-Strategie für metadata_json

Aktuell-Inventory: **16 unkritische Top-Level-Keys**: `entry_price`, `entry_time`, `exit_price`, `exit_time`, `fee_buy`, `fee_sell`, `total_fees`, `gross_proceeds`, `order_id`, `pnl`, `pnl_pct`, `position_id`, `quantity`, `reason`, `strategy_group`, `symbol`. **0 Rows** mit Sensitive-Pattern.

Trotzdem als **defensive Maßnahme** empfohlen (zukünftige Trades könnten anders aussehen):

```
// Pseudo-Code Plan
$DENY_REGEX = '/(api_key|api_secret|token|password|passwd|pass|dsn|x-api-key|
                authorization|bearer|secret|credential|private_key|salt|hash|
                jwt|cookie|session)/i';

function sanitize(array $meta): array {
    foreach ($meta as $k => $v) {
        if (preg_match($DENY_REGEX, $k)) {
            $meta[$k] = '[REDACTED]';
        } elseif (is_array($v)) {
            $meta[$k] = sanitize($v);  // recursive
        } elseif (is_string($v) && strlen($v) > 4096) {
            $meta[$k] = substr($v, 0, 4096) . '[TRUNCATED]';
        }
    }
    return $meta;
}
```

Drei Sanitize-Modi (Operator-Decision-3 weiter unten):

Modus	Verhalten
S1 Whitelist	nur die 16 bekannten Keys exportieren, alles andere droppen — sicherster Pfad
S2 Denylist (oben)	alle Keys exportieren außer Pattern-Match → `[REDACTED]` — zukunftssicher, weniger restriktiv
S3 metadata_json komplett aus	option für Operator pro Export — Modal-Checkbox „include metadata_json"

6. Export-Architektur (Vorschlag)

Komponenten

```
gui/app/Filament/Resources/TradeLogResource.php
   + canAccess() : Admin-only Gate (analog ManagedProposalResource)

gui/app/Filament/Resources/TradeLogResource/Pages/ListTradeLogs.php
   + getHeaderActions() : [TradeLogExportAction::make()]

gui/app/Filament/Actions/TradeLogExportAction.php   ← NEU
   - Action::make('export')->icon->color
   - ->form([
        DatePicker::make('date_from'),
        DatePicker::make('date_to'),
        Select::make('symbol')->options(...)->searchable(),
```

```
        Select::make('side')... evtl. NICHT (siehe §3),
        Select::make('status'),
        Select::make('strategy_id'),
        Select::make('environment')->default('testnet'),
        Select::make('format')->options(['json'=>'JSON','xml'=>'XML'])->required(),
        Toggle::make('include_metadata_json')->default(true),
        Toggle::make('use_current_filters')->default(false),
      ])
    - ->action(function (array $data) {
        $service = app(TradeLogExporter::class);
        return $service->stream($data);
      })

gui/app/Services/Exports/TradeLogExporter.php   ← NEU
   - public function stream(array $opts): StreamedResponse
   - chunked Query mit cursor() (memory-safe)
   - format-switch JSON / XML
   - Sanitizer recursive
   - Header: Content-Disposition attachment + filename mit UTC-Zeitstempel
   - keine Datei auf Disk, kein /srv/shares
```

Streaming-Pattern (kein Disk-Write)

```
return response()->streamDownload(
    function () use ($query, $format) {
        // 1. öffne Output-Stream php://output
        // 2. write Header (z.B. <?xml ... <trades> oder { "trades": [ )
        // 3. cursor()-Loop, je Row -> sanitize -> encode -> output
        // 4. write Footer
    },
    'trade-logs-' . now()->utc()->format('Ymd-His') . '.' . $format,
    ['Content-Type' => $mime, 'X-Robots-Tag' => 'noindex']
);
```

Filename: `trade-logs-20260514-104923.json` / `.xml` (UTC-Zeitstempel-Format wie operator-spezifiziert).

XML-Schema (Vorschlag)

```
<?xml version="1.0" encoding="UTF-8"?>
<trade_logs export_generated_at="2026-05-14T10:49:23Z"
            row_count="49"
            filter_summary="…">
  <trade>
    <id>42</id>
    <trade_id>…</trade_id>
    <symbol>BTCUSDT</symbol>
    <entry_price>30000.00000000</entry_price>
    ...
    <metadata>
      <order_id>…</order_id>
      ...
    </metadata>
  </trade>
</trade_logs>
```

JSON-Schema (Vorschlag)

```
{
  "export_generated_at": "2026-05-14T10:49:23Z",
  "row_count": 49,
  "filter_summary": { "date_from": "…", "symbol": "…", … },
  "trades": [
    {
      "id": 42, "trade_id": "…", "symbol": "BTCUSDT",
      "entry_price": "30000.00000000", "exit_price": "…",
      ...,
      "metadata": { "order_id": "…", ... }
    },
    ...
  ]
}
```

7. UX-Vorschlag

1. **Export-Button** oben rechts in der Tabelle (Filament HeaderAction-Pattern), Icon: `heroicon-o-arrow-down-tray`, Label „Export", Color: `primary`.
2. **Modal-Schritte**:
   - (a) Format-Wahl (JSON/XML radio)
   - (b) Filter-Section collapsable (alle Filter optional)
   - (c) Toggle „include metadata_json" (default ON)
   - (d) Toggle „use current table filters" — übernimmt aktuelle Filter aus dem Livewire-State (Operator-Komfort)
   - (e) Submit-Button „Generate Export"
3. **Feedback**: Browser triggert Download direkt nach Submit (kein Notification-Flow nötig wegen kleiner Datenmenge).
4. **Empty-State**: wenn Filter 0 Rows liefern → Modal-Notification „Keine TradeLogs im gewählten Bereich. Export abgebrochen".

8. Performance-Plan

Aspekt	Plan
Row-Limit	Aktuell 49 → keine Limit-Notwendigkeit. Defensive: `limit(50_000)` hardcoded mit Warning-Log bei Hit
Memory	Cursor-Iteration (`$query->cursor()`), max 1 Row gleichzeitig im PHP-Memory
Streaming	Output direkt nach `php://output` , kein Buffering, kein Disk
Timeout	Bei aktueller Größe irrelevant; defensiv `set_time_limit(300)` in der Action
DB-Last	49 Rows + Indexes → einstellige ms. Vernachlässigbar.

9. Security-Plan

Anforderung	Umsetzung
Admin-only	`canAccess()` + `Action::make('export')->visible(fn () => UserRole::Admin === auth()->user()->role)`
MFA-authenticated	bereits durch Filament Panel-Auth + `requiresMultiFactorAuthentication()` in AdminPanelProvider
CSRF	Filament-Actions sind automatisch CSRF-protected (Livewire-Layer)
Keine Public-URL	Action triggert direkt einen StreamedResponse, kein Persistent-File, kein Public-Routing
Kein Disk-Write	nur `php://output` , keine `Storage::put`
Sanitize metadata_json	S1 oder S2 Modus (siehe §5)
Audit-Event	OPTIONAL (siehe §11)

10. Audit-Event-Frage (Operator-Decision D2)

Operator-Vorschlag im Master-Prompt: `event_type = gui.trade_logs_export_requested` .

Option	Vorteil	Cost
A — nur Laravel-Log	kein DB-Write, simpel	nicht durchsuchbar in GUI
B — audit_events INSERT	durchsuchbar, vollständig auditierbar	1× DB-Write pro Export; benötigt Migration NICHT (Tabelle existiert seit MH-INCIDENT-Recovery), nur Code für Event-Persistierung
C — beides	belt-and-suspenders	Code-Doppel-Effort

Empfehlung B: `audit_events` existiert (laut `\d audit_events` aus früheren Phasen). Schema-Footprint = 0 (kein Migrations-Block), nur ein INSERT als App-User (hat INSERT-Privs). Spalten typisch: `user_id, event_type, payload_json, created_at` .

11. Test-Plan

Test	Typ
Admin sieht Export-Action	Feature-Test
Non-Admin (User-Role) sieht Action NICHT	Feature-Test
JSON-Export Content-Type = `application/json`	Feature
XML-Export Content-Type = `application/xml`	Feature
Date-Filter date_from/date_to wirkt (factory-driven)	Feature
Symbol-Filter wirkt	Feature
metadata_json mit forged `api_key` -Schlüssel → output enthält `[REDACTED]`	Unit (Sanitizer)
Export schreibt KEINE Datei nach `/srv/shares` (filesystem-Watch im Test)	Feature
Export nutzt KEINE Bot/Worker-Dateien (grep gegen `/home/node` -Paths)	Source-Grep im Test
Filename matched Pattern `trade-logs-\d{8}-\d{6}\.(json|xml)`	Feature
Empty-Result-Handling (0 Rows → Notification)	Feature
Source-Grep gegen Secret-Patterns im fertigen Output	Feature (vollständige Test-Pipeline)
Full GUI Suite via Safe-Runner	`bash gui/scripts/run_tests_safe.sh`

12. Betroffene Files (Implementation-Phase)

Operation	Datei
edit	`gui/app/Filament/Resources/TradeLogResource.php` (canAccess + 1 line)
edit	`gui/app/Filament/Resources/TradeLogResource/Pages/ListTradeLogs.php` (HeaderAction registrieren)
NEW	`gui/app/Filament/Actions/TradeLogExportAction.php`

NEW	`gui/app/Services/Exports/TradeLogExporter.php`
NEW	`gui/app/Services/Exports/TradeLogSanitizer.php` (optional, kann in Exporter inline)
NEW	`gui/tests/Feature/Filament/TradeLogExportTest.php`
NEW	`gui/tests/Unit/TradeLogSanitizerTest.php`
keine Migration	(audit_events Tabelle existiert bereits — kein Schema-Change)
keine Bot/Worker-Files	0× Touch

13. Operator-Decisions D1–D3

1. **D1 — Sanitizing-Modus**: S1 (Whitelist, nur 16 bekannte Keys) / S2 (Denylist Regex) / S3 (metadata_json komplett optional per Toggle)?
   Empfehlung: **S1 + Toggle** — Whitelist als default, mit Operator-Override per Modal-Toggle „include all metadata_json (raw)".
2. **D2 — Audit-Event**: A (nur log) / B (audit_events INSERT) / C (beides)?
   Empfehlung: **B**.
3. **D3 — „side / action" im Export**: weglassen oder synthetisch aus realized_pnl ableiten oder per metadata_json-Lookup?
   Empfehlung: **weglassen** — TradeLog modelliert geschlossene Round-Trips, nicht Order-Side. `strategy_group` übernimmt die „Kategorie-Aufteilung"-Rolle.

14. GO/NO-GO & Reihenfolge

> **GO für SEC-1c-3c-fenster-paralleles Implementation** — Operator-Wunsch ist eine reine Read-only-Funktion, berührt KEINE Trading-Logik / Bot/Worker / Migration. Kann während des 7-Tage-Stabilitätsfensters (bis 2026-05-21) laufen, ohne SEC-1c-3c/3d-Themen anzufassen.

Pre-Condition	Status
Plan-Review (dieses Doc)	✓
Operator-Decisions D1-D3	offen
TradeLog Admin-Gate (canAccess())	fehlt — wird mit Implementation eingezogen
audit_events Tabelle	existiert (kein Migrations-Block)
Safe-Runner aktiv	✓ (`gui/scripts/run_tests_safe.sh`)

Aufwand-Schätzung Implementation: ~4-6h Code + ~2h Tests + 30min GUI-Container-Recreate + Operator-Smoke-Test.

15. Mainnet-Relevanz

Niedrig. Pure Read-only-Export berührt keinen Trading-Pfad. Sinnvolles Vor-Mainnet-Tool für Strategie-Validation + Steuer/Audit-Export. Nicht-Blocker für MH-7.

16. Boundaries dieser Plan-Review-Phase

- 0× Code geschrieben
- 0× Migration / DB-Write
- 0× Container-Restart / docker cp
- 0× Bot/Worker-Touch
- 0× Push (master `31ef276` unverändert)
- 0× Mainnet
- 0× Secret-Output (metadata_json-Inspect war Boolean-only Pattern-Match)

Erstellt: 2026-05-14 (UTC) · Phase: GUI-TRADE-EXPORT-1 Plan-Review · Master HEAD: `31ef276` · Backlog-Pin: `backlog_gui_trade_export_1.md`