AUTO-ARBITRAGE — Backend Architecture, Math, and Integration Guide

A practical specification for adding executable cross-exchange arbitrage to Coinmatcher v2 (FastAPI + Next.js)

Author: Internal Research

Purpose: This document defines the algorithms, endpoints, data contracts, and risk controls for a production-ready AUTO-ARBITRAGE module. It includes a profitability calculator, executable order routing design, and integration steps with the existing lead–lag analytics.

1) Overview

AUTO-ARBITRAGE executes simultaneous buy/sell legs on mispriced markets across multiple exchanges. It does not rely on mid-transfer price stability. Funds are pre-deployed on venues; transfers are used only for periodic rebalancing. The module exposes (a) a Profitability Evaluator and (b) an Execution Engine. The UI surfaces greenlit opportunities with explicit net margin after fees, slippage, and safety buffers.

Core Requirements

- Depth-based pricing: use L2 order books on both venues to compute executable VWAP for a given order size.
- Fee awareness: apply venue-specific maker/taker, min-notional, tick size, and lot size filters.
- No mid-trade transfers: simultaneous legs only; periodic rebalancing outside trading loop.
- Deterministic risk controls: per-symbol notional caps, latency guards, venue health checks, kill-switch.
- Auditable: every decision logged with inputs/outputs for post-trade analysis.

2) System Architecture

The AUTO-ARBITRAGE module attaches to Coinmatcher v2:

```
backend/
  api/
    auto_arb.py            # REST endpoints
  services/
    arb_router.py          # routing and order placement
    arb_calc.py            # profitability evaluator
    arb_inventory.py       # balance/inventory tracking
    arb_rebalance.py       # periodic cross-venue rebalancer
  data/
    venues/                # websocket adapters (binance, okx, kucoin)
    fee_schedules/         # maker/taker & filters
frontend/
  pages/auto-arbitrage.tsx # UI surface for opportunities and controls
  components/OrderbookWidget.tsx
  components/ArbOpportunityCard.tsx
```

Process Flow

1) WebSockets stream L2 books into Redis/Postgres. 2) arb_calc builds executable spreads for candidate sizes. 3) If net edge exceeds threshold, arb_router submits IOC/PostOnly orders simultaneously. 4) arb_inventory updates balances; arb_rebalance runs off-peak to even inventories.

3) Data Contracts (schemas)

Order Book Snapshot (normalized):


```
{
  "ts": 1699999999123,               # ms timestamp
  "venue": "binance",
  "symbol": "NEIRO/USDT",
  "best_bid": 0.00518,
  "best_ask": 0.00519,
  "bids": [{"p":0.00518,"q":12000.0}, {"p":0.00517,"q":14000.0}, ...],
  "asks": [{"p":0.00519,"q":15000.0}, {"p":0.00520,"q":16000.0}, ...],
  "lot_size": 1.0e-6,                # base step
  "tick_size": 1.0e-6,               # price step
  "min_notional": 5.0                # in quote
}
```


Opportunity Payload (UI <-> Backend):


```
{
  "symbol": "NEIRO/USDT",
  "buy_venue": "okx",
  "sell_venue": "kucoin",
  "order_size_quote": 1000.0,
  "assume_maker": false,
  "use_transfer_mode": false,
  "vol_bps_per_min": 12.0,
  "expected_transfer_min": 0.0,
  "safety_buffer_bps": 10.0,
  "max_slippage_bps": 5.0
}
```

4) Math & Profitability

Let Q be the base quantity to trade. From the buy venue asks, compute the VWAP needed to fill Q: P_buy_eff. From the sell venue bids, compute P_sell_eff. Maker/taker fees are fA and fB. Mid-price m = (best_ask_buy + best_bid_sell) / 2.

```
(edge per unit) = P_sell_eff * (1 - fB) - P_buy_eff * (1 + fA)

Net PnL (quote) = Q * (edge per unit)
                  - withdraw_fee_quote (if transferring)
                  - latency_cost

latency_cost = (latency_bps/10000) * Q * m
latency_bps  = clamp( max( 2 * vol_bps_per_min * expected_transfer_min, 50 ), 300 )

Break-even (bps) ≈ (fA + fB)*1e4 + slippage_bps + safety_buffer_bps
Executable spread (bps) = ((P_sell_eff - P_buy_eff)/m) * 1e4
```

Depth-Fill VWAP:

```
def vwap_to_fill(levels, target_base):
    filled, cost = 0.0, 0.0
    for lvl in levels:  # levels sorted best-first
        take = min(target_base - filled, lvl['q'])
        cost += take * lvl['p']
        filled += take
        if filled >= target_base:
            break
    if filled < target_base: return None
    return cost/filled
```

5) FastAPI Endpoints

Mount these under /api/v1/auto_arb

```
POST /evaluate
Request: Opportunity Payload + latest order books
Response:
{
  "exec_spread_bps": 42.1,
  "slippage_bps": 6.5,
  "latency_bps": 0.0,
  "break_even_bps": 38.0,
  "net_pnl_quote": 4.27,
  "net_margin_pct": 0.427,
  "decision": "EXECUTE" | "SKIP",
  "notes": "Used 3 ask levels on okx; 4 bid levels on kucoin"
}

POST /execute
Request: same as evaluate + "size_mode": "quote"|"base"
Action: places simultaneous orders on both venues via venue adapters, returns order ids & fi
Response: { "status":"submitted", "buy_order_id":"...", "sell_order_id":"..." }

GET /opportunities
Params: symbol, min_edge_bps
Action: streams ranked candidates from in-memory calc service.
```

Security Notes:

Require per-user API keys with withdrawals disabled, IP-whitelist, and role-based ACL. Log every request/response (without secrets).

6) Deterministic Profitability Calculator (Python)

```
def depth_fill_price(levels, target_base):
    filled, cost = 0.0, 0.0
    best = levels[0]['p']
    for lvl in levels:
        if target_base <= 0: break
        take = min(target_base, lvl['q'])
        cost += take * lvl['p']
        target_base -= take
    if target_base > 1e-12: return None, None
    vwap = cost / sum(l['q'] for l in levels[:1])  # simplified best ref
    slippage_bps = abs(vwap - best) / best * 1e4
    return vwap, slippage_bps

def arb_eval(order_size_quote, asks_buy, bids_sell, f_buy, f_sell,
             withdraw_fee=0.0, expected_transfer_min=0.0, vol_bps_per_min=0.0,
             safety_bps=10.0):
    # initial base estimate
    base_guess = order_size_quote / asks_buy[0]['p']
    P_buy, slipA = depth_fill_price(asks_buy, base_guess)
    if P_buy is None: return None
    base_qty = order_size_quote / P_buy
    P_sell, slipB = depth_fill_price(bids_sell, base_qty)
    if P_sell is None: return None

    mid = 0.5 * (asks_buy[0]['p'] + bids_sell[0]['p'])
    gross = base_qty * (P_sell*(1 - f_sell) - P_buy*(1 + f_buy))

    latency_bps = 0.0
    if expected_transfer_min > 0:
        latency_bps = min(max(2*vol_bps_per_min*expected_transfer_min, 50.0), 300.0)

    latency_cost = (latency_bps/1e4) * base_qty * mid
    net = gross - withdraw_fee - latency_cost
    exec_spread_bps = ((P_sell - P_buy)/mid) * 1e4
    break_even_bps = (f_buy + f_sell)*1e4 + (slipA + slipB) + safety_bps

    return {
        "exec_spread_bps": exec_spread_bps,
        "slippage_bps": slipA + slipB,
        "latency_bps": latency_bps,
        "break_even_bps": break_even_bps,
        "net_pnl_quote": net,
        "net_margin_pct": 100*net/order_size_quote
    }
```

Use this function in a FastAPI route, and mirror it in a deterministic TypeScript helper for the frontend preview.

7) Frontend Integration (Next.js)

Add an AUTO-ARBITRAGE page that shows ranked opportunities. For each candidate, call /evaluate with chosen size (e.g., 200, 500, 1000 USDT) and render net margin. Persist user presets.

```
// pages/auto-arbitrage.tsx (sketch)
useEffect(() => {
  const ws = new WebSocket(process.env.NEXT_PUBLIC_OPPS_STREAM);
  ws.onmessage = (msg) => setOpps(JSON.parse(msg.data));
  return () => ws.close();
}, []);

async function evaluate(opp, size) {
  const res = await fetch('/api/v1/auto_arb/evaluate', {
    method: 'POST',
    headers: {'Content-Type': 'application/json'},
    body: JSON.stringify({...opp, order_size_quote: size})
  });
  return res.json();
}
```

Show explicit warnings for low volume (<$100k/24h), shallow depth, or unrealistic spreads.

8) Rebalancing

Trade without transfers, then periodically rebalance inventory:

```
- Identify drift of base/quote per venue.
- Choose cheapest chain and time window.
- Move small amounts; avoid during peak volatility.
- Optionally hedge with perps to stay neutral while funds in flight.
```

9) Security & Compliance

• Store API keys encrypted; withdrawals disabled. • IP whitelist keys with venues. • Role-based perms; every action auditable. • Depending on jurisdiction, executing trades for users may require licensing—keep features personal-use unless you have counsel.

10) Prompt for Website Builder Bot (to wire the feature)

```
ROLE: Integration Engineer
OBJECTIVE: Add an AUTO-ARBITRAGE module to the Coinmatcher v2 stack.

STEPS:
1) Backend
  - Create files: backend/services/arb_calc.py, backend/services/arb_router.py, backend/api/
  - Implement arb_eval() from spec; expose POST /evaluate and POST /execute
  - Use existing Redis channels for order books; normalize symbols and fees
  - Add risk caps: per-trade notional, per-minute notional, venue health checks
2) Frontend
  - New page /auto-arbitrage with stream of ranked opps
  - Button 'Evaluate' for sizes [200, 500, 1000 USDT]; show net margin & decision
  - 'Execute' disabled unless decision == EXECUTE and user has API keys on both venues
3) Secrets
  - Add encrypted key storage; trade-only keys; withdrawals disabled; IP whitelist
4) QA
  - Paper-trading mode with mocked fills
  - Unit tests for arb_eval math and order routing fallbacks
OUTPUTS: PR with code, env var docs, and a runbook.
```

11) Runbook (Ops)

```
Start:
- docker-compose up -d
- Confirm WS order books flowing (Grafana dashboards)
- /api/v1/auto_arb/health -> OK
- Paper trading: true; set caps low

Deploy:
- Switch to maker/taker tiers in config
- Enable execution with tiny notional caps
- Monitor: fills, rejects, PnL, error rates

Rollback:
- Flip execution enable flag off
- Kill-switch if venue health degraded
```

Appendix A: Example Fee/Filter Table

```
binance: taker=0.001, maker=0.001, min_notional=5 USDT
okx:     taker=0.001, maker=0.0008
kucoin:  taker=0.001, maker=0.001
```

Appendix B: Sanity Filters for Fake Spreads

```
- Exclude pairs with 24h vol < $100k per venue
- Require 5x your order size within ±0.5% of top of book on both venues
- Require spread > break_even + 10 bps for N consecutive seconds (e.g., N=5)
```