KLTBN

Architecture & Developer Reference

C++20 · ImGui / ImPlot · OpenGL · SDL2 · libcurl · OpenSSL

Version 1.0.0 · February 2026

1. Overview

Kltbn is a desktop algorithmic trading terminal written in C++20. It provides real-time market data streaming, manual order entry, and an automated rule-based strategy engine — all within a single-binary desktop application.

1.1 Key Capabilities

- Connect to 11 cryptocurrency exchanges via REST + WebSocket (Binance Spot/Futures, Bybit, OKX, Kraken, MEXC, BitMEX, Bitfinex, Gate.io, Bitget, TigerX)
- Live price streaming with tick-by-tick trade feed
- Candlestick chart (ImPlot) with 9 time presets: 1m → 1d, 1M, 1Y, ALL
- Tiger-style manual trading panel: Market / Limit / Stop / Stop-Limit, Take-Profit %, Stop-Loss %
- Visual rule engine: IF price / IF command executed → BUY / SELL / STOP-LOSS / TAKE-PROFIT
- Persistent strategy storage as human-readable JSON files
- AES-256-GCM encrypted credential store, machine-id bound key derivation (PBKDF2-SHA256)
- TLS 1.2+ enforced on all connections; wss:// / https:// URL validation
- UI scale: 100% / 150% / 200%, DPI-aware font rebuild
- 186-test GoogleTest suite covering all non-UI units

1.2 Technology Stack

Component	Technology
UI Rendering	Dear ImGui 1.90.6, ImPlot
Window / OpenGL	SDL2 + OpenGL 3 (GLSL 130)
Networking (REST)	libcurl with TLS 1.2+ hardening (TlsConfig)

Networking (WebSocket)	libcurl WebSocket API (7.86+), SecureWebSocket wrapper
Cryptography	OpenSSL — HMAC-SHA256/384/512, AES-256-GCM, RSA-SHA256, PBKDF2
JSON	nlohmann/json 3.11.3
Build System	CMake 3.20+, C++20
Testing	GoogleTest 1.14.0 via FetchContent
CI	GitHub Actions (ubuntu-24.04, macos-latest)

2. Architecture

The application is structured in four layers. Data and control flow strictly downward; the UI layer owns the engine and client, never the other way around.

2.1 Layer Diagram

```
┌──────────────────────────────────────────────────────┐
│  UI LAYER            MainWindow                       │
│  (ImGui / SDL2)      renderFrame / render*Panel       │
│                      strategy builder, trading module │
├──────────────────────────────────────────────────────┤
│  ENGINE LAYER        StrategyEngine                   │
│                      feedTick → evaluate → fire       │
│                      feedOrderUpdate → state machine  │
│                      StrategyStorage  (JSON files)    │
│                      ExchangeRegistry (factory)       │
├──────────────────────────────────────────────────────┤
```

```
│  EXCHANGE LAYER    IExchangeClient (interface)       │
│                    BaseExchangeClient (CURL + WS)    │
│                    11 concrete adapters              │
├──────────────────────────────────────────────────────┤
│  INFRA / UTILS     SecureConfig  CryptoConfig        │
│                    TlsConfig     RsaSigner  Logger   │
└──────────────────────────────────────────────────────┘
```

2.2 Startup Sequence

- main() creates MainWindow, calls run()
- initSDL(): creates SDL2 window with OpenGL context (HiDPI-aware)
- initImGui(): binds ImGui SDL2 + OpenGL3 backends, loads font, applies dark theme
- run(): enters render loop — polls SDL events, calls renderFrame() each frame, swaps buffer
- renderFrame(): draws full-screen ImGui window, delegates to render*() sub-functions
- Connect button: ExchangeRegistry::create(creds) → client->connect() → new StrategyEngine(*client)
- StrategyStorage::loadAll() seeds engine with persisted strategies on every connect

2.3 Real-Time Data Flow

```
Exchange WebSocket
    │  (price tick arrives on wsThread)
    ▼
BaseExchangeClient::onWsMessage()
    │  parses JSON, builds Quote struct
    ▼
QuoteCallback  (lambda set by MainWindow::subscribePair)
    ├── MainWindow::onQuote()        updates market state, tick ring, trade feed
    └── StrategyEngine::feedTick()  mutex-locked, evaluates all running strategies
             │
```

```
            ├── IF_PRICE: compare(price, condition.priceValue)
            └── IF_CMD_EXECUTED: check referenced command.state == EXECUTED
                    │  (if fires && budgetOk)
                    ▼
            StrategyEngine::fireCommand()
                    │  places REST order via m_client
                    └── registers orderId → m_orderMap


Exchange REST / WebSocket (order fill notification)
    ▼
OrderCallback  (set in renderConnectModal)
    ├── StrategyEngine::feedOrderUpdate()  updates command state machine
    └── MainWindow::upsertOrder()          adds/updates order in orders table
```

3. Module Reference

3.1 IExchangeClient (src/core/IExchangeClient.h)

Pure abstract interface that all exchange adapters implement. Every other module depends only on this interface, making exchange adapters fully interchangeable.

Data Types

Method / Field	Returns	Purpose
`Quote`	`struct`	Latest tick: symbol, pair, latestPrice, OHLC, volume, timestamp (ms)

`Bar`	`struct`	One OHLC candlestick bar: time (epoch s), open/high/low/close, volume
`Order`	`struct`	Order snapshot: orderId, symbol, side, type, status, qty, price, filledQty, avgFillPrice
`AccountInfo`	`struct`	Account snapshot: netLiquidation, availableFunds, buyingPower, cashBalance, currency, accountId
`Position`	`struct`	Open position: symbol, quantity, avgCost, marketValue, unrealizedPnl
`ExchangeCredentials`	`struct`	Connect parameters: exchangeId, apiKey, apiSecret, passphrase, testnet

Interface Methods

Method / Field	Returns	Purpose
`connect()`	`bool`	Authenticate and open connections. Returns false on failure.
`disconnect()`	`void`	Gracefully close REST and WebSocket connections.
`isConnected()`	`bool`	Thread-safe connectivity check.
`getQuote(symbol)`	`Quote`	REST: fetch latest price for a single symbol.
`getQuotes(symbols)`	`vector<Quote>`	REST: batch quote fetch.
`getKlines(symbol, interval, limit)`	`vector<Bar>`	REST: fetch OHLC history. interval: "1m"..."1d". Default limit 100.
`placeMarketOrder(sym, side, qty)`	`Order`	REST: place market order. side = "BUY" or "SELL".
`placeLimitOrder(sym, side, qty, price)`	`Order`	REST: place limit order at specified price.
`placeStopOrder(sym, side, qty, stop)`	`Order`	REST: place stop order triggered at stop price.
`cancelOrder(sym, orderId)`	`bool`	REST: cancel an open order. Returns true on success.
`getOrder(sym, orderId)`	`Order`	REST: query current order status.
`getAccountInfo()`	`AccountInfo`	REST: fetch full account balance snapshot.
`getPositions()`	`vector<Position>`	REST: list all open positions.
`subscribeQuotes(symbols, cb)`	`void`	WebSocket: subscribe to live price stream, fires cb on every tick.

`unsubscribeQuotes(symbols)`	`void`	WebSocket: unsubscribe from price stream.
`setOrderCallback(cb)`	`void`	Register callback for incoming order fill/cancel notifications.
`exchangeId()`	`string`	Short identifier, e.g. "BINANCE", "BYBIT".
`toPairString(base, quote)`	`string`	Format symbol per exchange convention: BTCUSDT / BTC-USDT / BTC/USDT.

3.2 BaseExchangeClient (src/core/exchanges/BaseExchangeClient.h)

CRTP-style base class that implements all networking plumbing. Concrete adapters override only the exchange-specific parsing logic.

REST Networking

Method / Field	Returns	Purpose
`sendRequest(method, path, params, sign)`	`json`	Build and execute a CURL request. sign=true appends HMAC signature + timestamp. Mutex-protected.
`sendRequestEx(method, path, body, headers)`	`json`	Extended variant accepting a raw body and custom headers (used by Kraken, OKX, Bitget).
`hmacSha256/384/512(data)`	`string`	Compute HMAC digest over data using the stored API secret. Returns hex string.
`buildQuery(params)`	`string`	URL-encode a key=value list into a query string.
`nowMs()`	`long long`	Current time in milliseconds since epoch, used for request timestamps.

WebSocket

Method / Field	Returns	Purpose
`wsConnect(url)`	`bool`	Creates SecureWebSocket, starts wsLoop() on a detached thread.
`wsDisconnect()`	`void`	Signals wsRunning=false, joins thread, destroys SecureWebSocket.
`wsSend(msg)`	`bool`	Thread-safe forward to SecureWebSocket::send().

`wsLoop()`	`void`	Background thread: polls SecureWebSocket::receive(), calls onWsMessage().
`onWsMessage(msg)*`	`void`	Pure virtual. Each adapter parses its own JSON message format here.

3.3 Exchange Adapters

Class	ID	Auth	Testnet	Notes
`BinanceClient`	`BINANCE`	`HMAC-SHA256`	Yes	Full symbol as BTCUSDT. Spot
`BinanceFutClient`	`BINANCE-FUT`	`HMAC-SHA256`	Yes	Perpetual futures, fapi.binance.com
`KrakenClient`	`KRAKEN`	`HMAC-SHA512`	No	API-Sign header, nonce-based replay guard
`BybitClient`	`BYBIT`	`HMAC-SHA256`	Yes	Unified account, timestamp window
`OkxClient`	`OKX`	`HMAC-SHA256`	Yes	Passphrase required. Symbol: BTC-USDT
`MexcClient`	`MEXC`	`HMAC-SHA256`	No	Standard REST + WS
`BitmexClient`	`BITMEX`	`HMAC-SHA256`	No	Perpetual contracts
`BitfinexClient`	`BITFINEX`	`HMAC-SHA384`	No	Symbol prefix: tBTCUSD
`GateioClient`	`GATEIO`	`HMAC-SHA512`	No	Symbol: BTC_USDT
`BitgetClient`	`BITGET`	`HMAC-SHA256`	No	Passphrase required
`TigerXClient`	`TIGERX`	`RSA-SHA256`	No	Uses RsaSigner with PEM private key

3.4 SecureWebSocket (src/core/SecureWebSocket.h)

Thin TLS WebSocket wrapper built on libcurl's WebSocket API (7.86+). Enforces TLS 1.2+ via TlsConfig::harden(). Frames are buffered in an internal queue; BaseExchangeClient's wsLoop() polls them with a 500 ms timeout.

Method / Field	Returns	Purpose
`send(text)`	`bool`	Enqueue a text frame for the remote. Thread-safe.

`receive(out, timeoutMs)`	`bool`	Block until a message arrives or timeout elapses. Returns false on timeout.
`close()`	`void`	Send WebSocket CLOSE frame and mark socket closed.
`isOpen()`	`bool`	Atomic flag; safe to poll from any thread.

3.5 ExchangeRegistry (src/core/ExchangeRegistry.h)

Static registry of all supported exchanges. Serves two purposes: providing metadata for the Connect UI dialog, and acting as the factory that instantiates the correct client class.

ExchangeInfo Fields

Field	Purpose
`id / name / group`	Unique identifier, display name, exchange group ("Spot" / "Futures")
`needsKey / needsSecret`	Controls which credential fields the UI shows
`needsPassphrase`	OKX and Bitget require a 3rd secret (passphrase)
`needsPrivateKey`	TigerX uses RSA PEM file path instead of API secret
`supportsTestnet`	Enables the testnet checkbox in Connect dialog
`restUrl / websocketUrl`	Production endpoints used by the client on connect
`testnetRestUrl / testnetWsUrl`	Sandbox endpoints; empty string = no testnet support
`makerFee / takerFee`	Base fee % shown in the Connect dialog fee summary
`feeNote / feeUrl`	Human-readable fee tiers note and link to official schedule

Factory

ExchangeRegistry::create(creds) switches on creds.exchangeId (case-insensitive), constructs and returns the matching concrete client wrapped in unique_ptr<IExchangeClient>. Throws std::runtime_error for unknown IDs.

3.6 Strategy Domain Model (src/core/Strategy.h)

Plain data structures with no virtual dispatch. All strategy state lives in these structs; the engine and storage layers act on them.

Strategy	Top-level record: id, name, pair, maxCoins budget, conditions list, running flag, coinsInUse counter
StrategyCondition	One IF→THEN rule: condition type (IF_PRICE / IF_COMMAND_EXECUTED), comparison, priceValue or refCommandId, the thenCommand to fire, and triggered flag
StrategyCommand	One order action: BUY / SELL / STOP_LOSS / TAKE_PROFIT, amount (base coins), price (0 = market), execution state machine, orderId, fill info
Balance	Exchange account snapshot: currency, total, inUse, free() computed as total − inUse

Enumerations

Enum	Values
`CommandType`	BUY · SELL · STOP_LOSS · TAKE_PROFIT
`CommandState`	IDLE → PENDING → EXECUTED / CANCELLED / FAILED
`ConditionType`	IF_PRICE · IF_COMMAND_EXECUTED
`PriceComparison`	ABOVE · BELOW · EQUAL (tolerance ±0.01)

Strategy Helpers

Method / Field	Returns	Purpose
`nextConditionId()`	`int`	Returns max(condition.id) + 1. Used by UI when adding new conditions.
`nextCommandId()`	`int`	Returns max(command.id) + 1.
`resetRuntime()`	`void`	Clears running, coinsInUse, resets all triggered flags and command states to IDLE. Called on startStrategy().

3.7 StrategyEngine (src/core/StrategyEngine.h/.cpp)

Thread-safe strategy evaluator. Holds the authoritative strategy list; MainWindow polls a copy via snapshot() when the dirty flag is set.

Lifecycle

Method / Field	Returns	Purpose
`StrategyEngine(client)`	`ctor`	Stores reference to IExchangeClient. Does not connect.
`addStrategy(s)`	`void`	Insert or replace by ID. Sets dirty flag.
`removeStrategy(id)`	`bool`	Remove by ID. Returns false if not found.
`startStrategy(id)`	`bool`	Calls resetRuntime(), sets running=true. Returns false if already running.
`stopStrategy(id)`	`bool`	Sets running=false.
`stopAll()`	`void`	Stops all strategies. Called from destructor.

Tick Processing (called from WebSocket thread)

Method / Field	Returns	Purpose
`feedTick(pair, price)`	`void`	For each running strategy whose pair matches: evaluateStrategy(s, price).
`evaluateStrategy(s, price)`	`void`	Iterates conditions. Skips triggered or non-IDLE. Evaluates IF_PRICE or IF_COMMAND_EXECUTED. If fires AND budgetOk: calls fireCommand.
`budgetOk(s, amount)`	`bool`	Returns true if maxCoins ≤ 0 (unlimited) or (coinsInUse + amount) ≤ maxCoins.
`fireCommand(s, cond)`	`void`	Places REST order via m_client. Records orderId in m_orderMap. Sets state=PENDING. On exception: state=FAILED, resets triggered.

Order State Machine

feedOrderUpdate(order) looks up the orderId in m_orderMap to find which strategy + condition it belongs to, then transitions the command state:

Incoming status	New CommandState	Side effect
`"FILLED" / "FULLY_FILLED"`	`EXECUTED`	filledQty and avgFillPrice recorded
`"CANCELLED"`	`CANCELLED`	coinsInUse -= amount (floor 0)
`"REJECTED"`	`FAILED`	coinsInUse -= amount (floor 0)
`anything else`	`PENDING`	No coin adjustment

Snapshot & Dirty Flag

Method / Field	Returns	Purpose
`snapshot()`	`vector<Strategy>`	Returns a deep copy of m_strategies under lock. MainWindow caches this as m_stratList.
`dirty()`	`bool`	Atomically clears and returns the dirty flag. MainWindow calls this every frame.
`refreshBalance()`	`void`	Calls getAccountInfo() on the exchange, updates balance.total. Sums coinsInUse from all running strategies.

3.8 StrategyStorage (src/core/StrategyStorage.h/.cpp)

Static utility class. Strategies are stored as individual JSON files under ~/.local/share/kindaLikeTigerButNot/strategies/<id>.json. Files are human-readable so users can edit or share them.

File Format

```
{
  "id":          "a1b2c3d4e5f6a7b8",
  "name":        "BTC Breakout",
  "pair":        "BTC/USDT",
  "maxCoins":    0.5,
  "conditions": [
    {
      "id": 1, "type": 0, "comparison": 0, "priceValue": 50000,
      "refCommandId": 0,
      "thenCommand": { "id": 1, "type": 0, "amount": 0.1, "price": 0 }
    }
  ]
}
```

Note: runtime fields (triggered, state, orderId, filledAmount, avgFillPrice) are intentionally NOT serialised. Strategies always resume from a clean IDLE state.

Method / Field	Returns	Purpose
`storageDir()`	`filesystem::path`	Returns ~/.local/share/kindaLikeTigerButNot/strategies/ creating it if absent.
`save(strat)`	`string`	Serialises to JSON. Assigns a 16-hex-char random ID if strat.id is empty. Overwrites existing file with same ID. Returns path written.
`load(path)`	`Strategy`	Parses a JSON file. Throws on invalid JSON or missing fields.
`loadAll()`	`vector<Strategy>`	Calls listFiles() then load() on each. Skips malformed files (logs error, continues).
`remove(id)`	`bool`	Deletes <id>.json. Returns false if file does not exist.
`listFiles()`	`vector<string>`	Returns sorted list of all *.json paths in storageDir().

3.9 Security Utilities

CryptoConfig (src/utils/CryptoConfig.h/.cpp)

Provides AES-256-GCM authenticated encryption for the on-disk credential file. The encryption key is derived with PBKDF2-SHA256 from the machine's hardware identifier combined with a random per-file salt, binding decryption to the originating machine.

JSON Blob Format

```
{
  "v":    1,             // schema version
  "salt": "<base64>",   // 16-byte PBKDF2 salt (random per file)
  "iv":   "<base64>",   // 12-byte GCM nonce  (random per encrypt)
  "tag":  "<base64>",   // 16-byte GCM auth tag
  "ct":   "<base64>"    // ciphertext (same length as plaintext)
}
```

Key Derivation

machineId() reads /etc/machine-id on Linux or the IOPlatformSerialNumber on macOS. deriveKey() feeds machineId + random salt into PBKDF2-SHA256 with 100,000 iterations to produce a 256-bit key. The same key is never produced on a different host.

Security scope: protects against offline credential theft from backups or untargeted malware. Does not protect an attacker with live system access and the ability to read the machine-id.

Method / Field	Returns	Purpose
`encrypt(plaintext)`	`string`	RAND_bytes(salt, iv) → PBKDF2 key derivation → AES-256-GCM EncryptUpdate + Final → base64 JSON blob.
`decrypt(blob)`	`string`	JSON parse → PBKDF2 key derivation → AES-256-GCM DecryptUpdate → SetTag → Final. Throws std::runtime_error on authentication failure (tampered file).
`isEncrypted(blob)`	`bool`	Quick structural check: blob starts with "{" and contains "v" + "ct" fields.

TlsConfig (src/utils/TlsConfig.h/.cpp)

Applied to every CURL handle immediately after curl_easy_init(). Centralises TLS settings — no TLS options should be set anywhere else.

Method / Field	Returns	Purpose

`harden(curl)`	`void`	Sets CURLOPT_SSL_VERIFYPEER=1, CURLOPT_SSL_VERIFYHOST=2, CURLOPT_SSLVERSION=CURL_SSLVERSION_TLSv1_2. Restricts ciphers to ECDHE-RSA-AES128-GCM-SHA256 and equivalents.
`requireSecureScheme(url)`	`void`	Accepts https:// and wss:// (case-insensitive). Throws std::invalid_argument for http://, ws://, or anything else.

SecureConfig (src/utils/SecureConfig.h/.cpp)

Persists ExchangeCredentials to ~/.config/kindaLikeTigerButNot/config.enc (chmod 600). Transparently migrates from the legacy plaintext config.json on first save: the old file is zero-filled then deleted to reduce forensic recoverability.

Method / Field	Returns	Purpose
`save(creds)`	`void`	Serialises creds → JSON string → CryptoConfig::encrypt() → writes config.enc. Sets file permissions to 0600.
`load()`	`ExchangeCredentials`	Reads config.enc, calls CryptoConfig::decrypt(), JSON-parses. Falls back to legacy config.json if .enc is absent, then migrates.
`exists()`	`bool`	Returns true if either config.enc or config.json is present.

RsaSigner (src/utils/RsaSigner.h)

Header-only utility used by TigerXClient for RSA-SHA256 request signing required by the Tiger Brokers API.

Method / Field	Returns	Purpose
`loadPemFile(path)`	`string`	Reads PEM text from file path. Returns empty string on failure.
`signSha256(content, pemKey)`	`string`	EVP_DigestSignInit(SHA256) → EVP_DigestSignUpdate → EVP_DigestSignFinal → base64 encode. Throws on OpenSSL errors.

3.10 Logger (src/utils/Logger.h)

Singleton ring-buffer logger. Thread-safe. Always writes to stderr. The UI's trade feed and debug panels can register sinks to display log entries in-app.

Method / Field	Returns	Purpose
`instance()`	`Logger&`	Meyers singleton, safe for first call from any thread.
`log(level, fmt, args)`	`void`	std::format-based variadic template. Skips if level < minLevel. Appends to 4096-entry ring buffer. Notifies all sinks.
`addSink(fn)`	`void`	Register a callback (e.g. UI overlay) receiving every LogEntry after it is added to the ring.
`entries()`	`vector<LogEntry>`	Reference to the internal ring buffer (read under lock by caller).
`setMinLevel(l)`	`void`	Filter: DEBUG / INFO / WARN / ERR. Default: DEBUG.

Convenience Macros

```
LOG_DEBUG("feedTick pair={} price={:.2f}", pair, price);
LOG_INFO ("Strategy '{}' started", s.name);
LOG_WARN ("refreshBalance: {}", e.what());
LOG_ERROR("[CryptoConfig] GCM authentication tag mismatch");
```

4. UI Layer (src/ui/MainWindow.h/.cpp)

MainWindow is the single top-level UI class. It owns the IExchangeClient and StrategyEngine instances. All ImGui rendering is immediate-mode: state is read from member variables and written back on user interaction within the same frame.

4.1 Frame Layout

```
┌───────────────────────────────────────────────────────────────────┐
│ TOP BAR     [Pair input] [Subscribe]    [UI%]  [Connect]          │
```

```
├──────────────────────────────────────────────────────────── │
│ BALANCE BAR   Total: $X  In use: $Y  Free: $Z  [Refresh]  │
├─────────────────┬──────────────────────┬─────────────────────┤
│ LEFT PANEL      │ CENTER PANEL         │ RIGHT PANEL         │
│ Strategy list   │ Strategy builder     │ Trading module      │
│ + New / Load    │ Conditions / Commands│ (Tiger-style)       │
│ ▶ Start ■ Stop  │ Save / Delete        │ Cancel Buy/Sell/All│
│                 │                      │ Reverse position    │
├─────────────────┴──────────────────────┴─────────────────────┤
│ BOTTOM PANEL  [Orders] [Trades] [Feed]      ↕ drag resize   │
└──────────────────────────────────────────────────────────────┘
```

4.2 Key Render Functions

Method / Field	Returns	Purpose
`renderTopBar()`	`void`	Pair input box + Subscribe button. UI scale popup (100/150/200%). Connect/Disconnect button. Chart window toggle.
`renderBalanceBar()`	`void`	Displays Balance.total, .inUse, .free(). Refresh button calls engine->refreshBalance().
`renderLeftPanel()`	`void`	Scrollable list of strategies from m_stratList. Selecting sets m_selIdx/m_selId and loads builder. New/Load File/Start/Stop/Delete buttons.
`renderCenterPanel()`	`void`	Strategy builder: name, pair, maxCoins fields. Condition rows via renderConditionRow(). Save button with confirm modal.
`renderConditionRow(cond, idx)`	`void`	One IF→THEN row: condition type combo, price/comparison fields or command reference, then-command type/amount/price. State badge.
`renderRightPanel()`	`void`	Hosts renderTradingModule(). Draggable vertical splitter sets m_rightW.

`renderTradingModule()`	`void`	Tiger-style order panel: type selector, qty presets, price fields, TP/SL %, slippage. Buy/Sell buttons. Cancel Buy/Sell/All. Reverse position.
`renderBottomPanel()`	`void`	Tab strip [Orders] [Trades] [Feed] with vertical drag resize. Routes to render*Table/Feed.
`renderOrdersTable()`	`void`	Sortable table of m_orders. Filter: All / Active only. Inline Cancel button per row.
`renderTradesTable()`	`void`	Table of m_trades (filled orders only). Colour-coded side column.
`renderTradeFeed()`	`void`	Scrolling market-print feed from m_feed. Min-volume filter. Pause/Resume toggle.
`renderChartWindow()`	`void`	Floating OHLC chart. Pair input, Load button (explicit only). Interval pills 1m…1d, 1M, 1Y, ALL. ImPlot candlestick + live tick overlay. Auto-refresh by interval.
`renderConnectModal()`	`void`	Modal dialog: exchange dropdown, dynamic credential fields from ExchangeInfo, fee summary. On Connect: ExchangeRegistry::create() + client->connect() + new StrategyEngine.

4.3 Real-Time State

m_market (PairState)	Latest quote: pair, price, volume. Updated by onQuote() under m_marketMtx.
m_ticks (deque<TickPt>)	Ring of last 300 {time, price} points. Used for live-tick overlay on chart.
m_bars (vector<OhlcBar>)	OHLC bars from fetchChartData(). Cleared on explicit Load / interval change. Never auto-cleared on pair text change.
m_orders / m_trades	Upserted by upsertOrder() on every OrderCallback. Mutex-protected.
m_feed (deque<FeedEntry>)	Market prints from onQuote(). Capped at MAX_FEED=500. Pausable.
m_stratList	Deep copy of engine state, refreshed when engine->dirty() returns true.

4.4 fetchChartData()

Launched on a detached std::thread to avoid blocking the render loop. Fetch is triggered only when:

- User clicks "Load" or presses Enter in the pair field (explicitLoad)
- An interval pill is clicked (clears m_bars → autoFetch triggers)

- Bars are empty for the already-fetched pair (covers initial open and ⟳ Refresh)
- Auto-refresh timer expires: 30s for 1m, 120s for 5m, 300s for all others

Changing the pair text field alone does NOT trigger a fetch — Load must be clicked. This prevents stale data from being shown while the user types.

4.5 UI Scale

applyScale(scale) rebuilds the ImGui font at scale × 13.5pt from DejaVuSansMono.ttf (falls back to ImGui default). All pixel dimensions are multiplied by m_uiScale at the call site. The scale popup is an ImGui::BeginPopup anchored below the button — no custom window management required.

5. Build System & CI

5.1 CMake Options

Method / Field	Returns	Purpose
`BUILD_TESTS=ON`	`option`	Include tests/ subdirectory. On by default.
`CMAKE_BUILD_TYPE`	`string`	Release for distribution; Debug for test runs.

5.2 Quick Start

```
# 1. Fetch vendored dependencies (ImGui, ImPlot, nlohmann/json)

./scripts/setup_deps.sh



# 2. Configure + build

cmake -B build -DCMAKE_BUILD_TYPE=Release

cmake --build build -j$(nproc)



# 3. Run tests
```

```
ctest --test-dir build/tests --output-on-failure --parallel


# 4. Package
./scripts/build_appimage.sh build 1.0.0   # Linux
./scripts/build_dmg.sh      build 1.0.0   # macOS
```

5.3 Test Suite

186 GoogleTest cases covering all non-UI units. Tests use a MockExchangeClient that captures placed orders and allows injecting errors. Storage tests redirect HOME to a mkdtemp directory.

File	Tests	Coverage
`test_strategy.cpp`	29	Enum names, Balance.free(), nextId, resetRuntime, default values
`test_strategy_engine.cpp`	53	CRUD, lifecycle, IF_PRICE/IF_CMD_EXECUTED evaluation, budget, fireCommand order types, feedOrderUpdate state machine, snapshot, dirty, refreshBalance
`test_strategy_storage.cpp`	26	ID generation, roundtrip serialisation, overwrite, error cases, loadAll, remove, listFiles, storageDir creation
`test_tls_config.cpp`	22	requireSecureScheme accepts https/wss variants, rejects http/ws/ftp/empty, error message content, harden() null/idempotent
`test_crypto_config.cpp`	31	isEncrypted, encrypt output structure, roundtrip (empty/short/long/unicode/binary), decrypt error cases (tampered ct/tag, wrong version, missing fields)
`test_exchange_registry.cpp`	25	all() catalogue integrity (URLs, fees, no duplicates), find() case-insensitive, names(), create() unknown throws

5.4 GitHub Actions CI (.github/workflows/cmake-multi-platform.yml)

Runs on push and pull_request to main. Matrix: ubuntu-24.04 and macos-latest.

Step	Command

Setup	`./scripts/setup_deps.sh — installs system packages + clones third_party`
Configure	`cmake -B build -DCMAKE_BUILD_TYPE=Release -DBUILD_TESTS=ON`
Build	`cmake --build build --config Release --parallel`
Test	`ctest --test-dir build/tests --output-on-failure --parallel --timeout 60`
AppImage (Linux)	`./scripts/build_appimage.sh build 1.0.0 → upload artifact`
DMG (macOS)	`./scripts/build_dmg.sh build 1.0.0 → upload artifact`

6. Data Flows Summary

6.1 Connection Sequence

```
User fills Connect dialog
  → ExchangeCredentials built from form fields
  → SecureConfig::save(creds)           // encrypted to disk
  → ExchangeRegistry::create(creds)     // instantiates concrete client
  → client->connect()                   // REST auth handshake + WS open
  → new StrategyEngine(*client)
  → client->setOrderCallback(...)       // wires order fills to engine + UI
  → StrategyStorage::loadAll()          // seeds engine from disk
  → engine->refreshBalance()            // initial balance fetch
  → client->getPositions()              // seed position panel
```

6.2 Strategy Execution Flow

```
User builds strategy in CENTER PANEL → clicks Save
```

```
  → StrategyStorage::save(builder)     // writes JSON file
  → engine->addStrategy(builder)       // registered in engine

User clicks ▶ Start
  → engine->startStrategy(id)
       → s.resetRuntime()              // clear triggered / state / coins
       → s.running = true

WebSocket tick arrives
  → BaseExchangeClient::wsLoop()       // background thread
  → onWsMessage() → QuoteCallback
  → engine->feedTick(pair, price)
       → evaluateStrategy(s, price)
            → condition fires + budget OK
            → fireCommand(s, cond)
                 → client->placeMarketOrder / placeLimitOrder / placeStopOrder
                 → cmd.state = PENDING
                 → m_orderMap[orderId] = {stratId, condId}

Order fill arrives via OrderCallback
  → engine->feedOrderUpdate(order)
       → lookup orderId in m_orderMap
       → cmd.state = EXECUTED / CANCELLED / FAILED
       → adjust coinsInUse
  → upsertOrder(order)                 // update UI orders table
```

6.3 Credential Persistence

```
Save:
  ExchangeCredentials
    → json::dump()                     // plaintext JSON string
    → CryptoConfig::encrypt()          // AES-256-GCM
         → RAND_bytes(salt[16], iv[12])
         → PBKDF2-SHA256(machineId+salt, 100000) → key[32]
         → EVP_EncryptUpdate + Final   // ciphertext
         → EVP_CTRL_GCM_GET_TAG       // auth tag[16]
         → base64 JSON blob
    → write ~/.config/kltbn/config.enc (chmod 600)


Load:
  Read config.enc
    → CryptoConfig::decrypt(blob)
         → PBKDF2-SHA256(machineId+salt) → key
         → EVP_DecryptFinal (tag verify)
         → throws if tag mismatch (tampered file)
    → json::parse() → ExchangeCredentials
```

7. How to Add a New Exchange

The architecture is designed so adding an exchange requires touching only three files.

Step 1 — Create the client class

```
// src/core/exchanges/NewExClient.h
#pragma once
#include "BaseExchangeClient.h"
namespace kltbn {
class NewExClient : public BaseExchangeClient {
public:
    explicit NewExClient(const ExchangeCredentials& c);
    bool connect() override;
    void disconnect() override;
    // ... implement all pure virtuals from IExchangeClient ...
    std::string exchangeId()  const override { return "NEWEX"; }
    std::string displayName() const override { return "New Exchange"; }
protected:
    void onWsMessage(const std::string& msg) override;
};
} // namespace kltbn
```

Step 2 — Register in ExchangeRegistry

Add an ExchangeInfo entry to the static LIST in ExchangeRegistry.cpp, and add a case to the create() factory switch.

Step 3 — Add to CMakeLists.txt

Add src/core/exchanges/NewExClient.cpp to APP_SOURCES (root CMakeLists.txt) and to CORE_SOURCES (tests/CMakeLists.txt).

No other changes are needed. The Connect dialog, strategy engine, and tests all discover the new exchange automatically.

8. Glossary

Term	Definition
Tick	A single price update received from the WebSocket stream.
OHLC / Bar	One candlestick: Open, High, Low, Close price in a given time interval.
Strategy	A named collection of IF→THEN rules evaluated against price ticks.
Condition	One rule: IF price > X, IF price < X, IF command Y executed.
Command	One order action attached to a condition: BUY / SELL / STOP-LOSS / TAKE-PROFIT.
coinsInUse	Running total of base-currency amount committed by PENDING or EXECUTED commands.
maxCoins	Per-strategy budget cap. 0 = unlimited.
dirty flag	Atomic bool in StrategyEngine; set on any state change. MainWindow polls it to know when to refresh its copy.
HMAC-SHA256	Hash-based message authentication code used to sign REST requests.
AES-256-GCM	Authenticated encryption standard. GCM provides both confidentiality and integrity via the auth tag.
PBKDF2	Password-Based Key Derivation Function 2. Slows brute-force attacks via iteration count.
machineId	Hardware-bound identifier (/etc/machine-id on Linux, IOPlatformSerialNumber on macOS). Binds decryption to the originating machine.
ImGui	Immediate-mode GUI library: state is read and written on every frame rather than stored in widget objects.
ImPlot	ImGui extension for scientific/financial plots. Custom build adds inline property API.
DXA	Document XML A-unit: 1/20 of a typographic point (1440 DXA = 1 inch). Used in docx layout.