**📘 TradeSentry AI System Dossier**

*A Modular, Cloud-Native Strategy Automation Ecosystem*

**🔷 0. System Overview**

**TradeSentry AI** is a next-generation algorithmic trading AI system designed for modern retail and professional traders. It allows users to seamlessly activate pre-built automated trading strategies — including:

* **DRLFusion**: Deep Reinforcement Learning ensemble model.
* **SMEdge**: Smart Money Concepts (CHoCH, Order Blocks, Market Structure).
* **SMGrid**: Volatility-aware grid trading strategy.

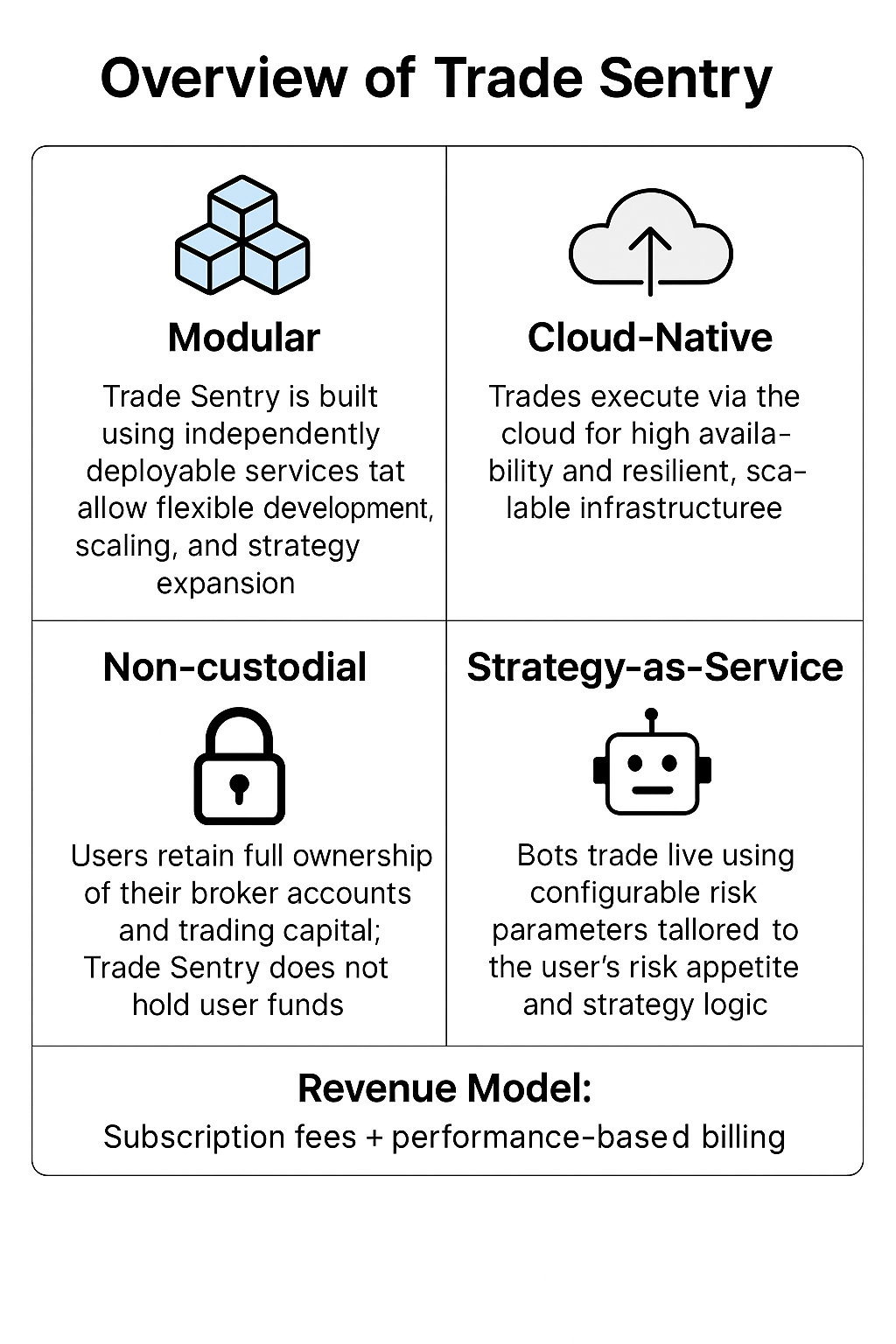


Figure : Overview of TradeSentry AI

The platform is fully **non-custodial**, meaning it does not hold or control user funds. All trading activity is conducted through **API-based connections** to user-specified broker accounts (e.g., MetaTrader 5, OANDA, Alpaca, Binance).

**Key Attributes:**

**🔐 Non-Custodial Execution**

Users retain 100% control over their broker accounts and funds. **TradeSentry AI** operates by placing and monitoring trades using **broker-issued API credentials** (MT5 login/server credentials, REST tokens, etc.).

**🤖 Strategy-as-a-Service (SaaS)**

Users select from multiple trading bots, configure their parameters (e.g., risk % per trade, SL/TP type, execution frequency), and monitor strategy execution in real-time. Each bot is modular, upgradeable, and independently deployable.

**💼 Dual Revenue Model**

1. **Subscription Fee**: Monthly access to the platform.
2. **Performance-Based Billing**: A fixed % (e.g., 30%) of the **net profit per trade** is automatically deducted from the user's **prepaid wallet** after each successful trade.

A diagram of a system architecture

AI-generated content may be incorrect.

Figure : TradeSentry AI System Architecture Overview

**🔷 1. Modular Architecture Overview of TradeSentry AI**

**TradeSentry AI** is structured around a **microservices-first architecture** to support independent scaling, modular development, and fault isolation.

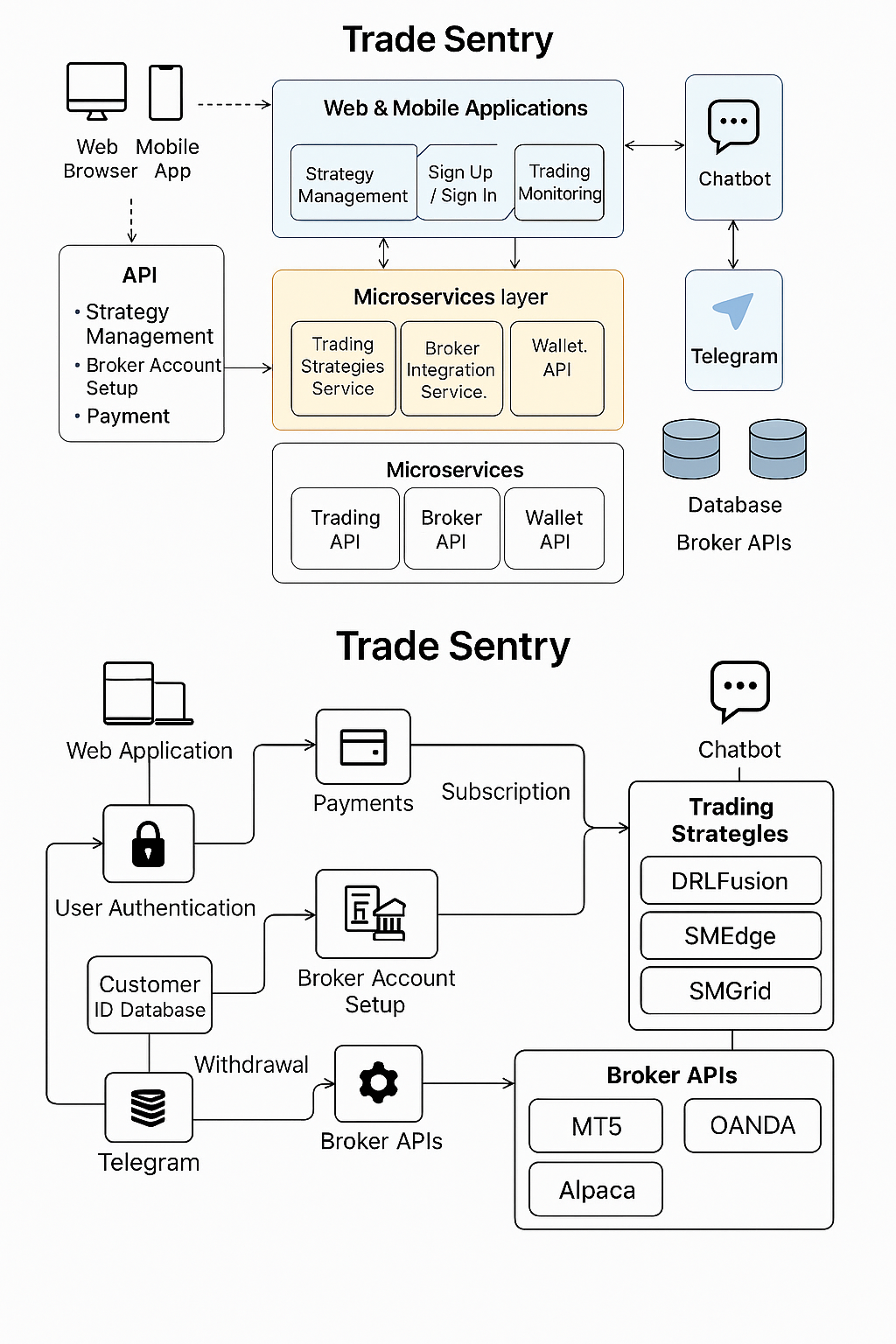


Figure : TradeSentry AI Modular Architecture Overview

**🧩 Web & Mobile Applications**

These serve as the **primary user interfaces**, offering:

* User registration, login (via OAuth2/JWT).
* Strategy Marketplace: Explore available strategies bots with performance statistics and risk profiles.
* Strategy Configuration Wizard: Set lot size, symbol(s), SL/TP method, trailing logic, etc.
* Real-time Trade Dashboard: View open positions, PnL, executed orders, and bot activity.
* Wallet & Billing View: Top-up wallet, view balance, audit logs, and invoices.

Built with frameworks like React (Web) and React Native/Flutter (Mobile), they connect to the backend via the API Gateway.

**🌐 API Gateway**

The **central entry point** for all client requests. It provides:

* Secure, rate-limited access to backend microservices.
* Request validation, token verification, and routing.
* Access to:
  + Strategy Management endpoints.
  + Broker Integration endpoints.
  + Wallet Management endpoints.
  + Monitoring and audit queries.

Backed by FastAPI/Nginx or Azure API Gateway with OAuth2 token authorization.

**⚙️ Microservices Layer**

Each core functionality is isolated into dedicated services:

**1. TradingStrategiesService**

* Loads and executes the logic for any selected strategy (DRLFusion, SMEdge, SMGrid, etc).
* Converts price data into signals using multi-timeframe aggregation.
* Calculates entries and exits.
* Stores trade metadata (e.g., direction, SL/TP, entry price).

**2. BrokerIntegrationService**

* Broker-specific handlers for MT5, OANDA, Alpaca, Binance, etc.
* Unified interface for:
  + Submitting market/limit/stop orders.
  + Querying account balance and equity.
  + Fetching open positions.
  + Getting historical OHLCV data.
* Uses Python SDK, REST APIs, or direct terminal bridges.

**3. TradingMonitoringService**

* Periodically or event-driven checks for:
  + SL/TP triggers.
  + Stop-out conditions.
  + Manual exits or trailing stop movements.
* Pushes updates to the user dashboard and Telegram.
* Ensures correct lifecycle state of each trade.

**4. WalletService**

* Maintains wallet balances per user.
* Deducts profit share % automatically after successful trades.
* If balance is insufficient:
  + Creates an invoice.
  + Notifies the user via Telegram.
  + Pauses bot execution.
* Supports payment webhooks (Stripe, PayPal).

**🔧 Infrastructure Microservices**

**a. Trading API**

* Strategy logic orchestration.
* SL/TP calculation engine (ATR, fixed, R-multiple).
* Entry signal standardization.

**b. Broker API**

* Abstracts broker differences into one common trading interface.
* Safeguards credentials via key vault integrations.

**c. Wallet API**

* Top-up (credit) wallet.
* Check balance before executing any trade.
* Log deductions and failed attempts.

**📡 Chatbot & Telegram**

**Chatbot**

* Powered by GPT for contextual help.
* Supports navigation queries, config help, and onboarding support.
* Available both on-platform and on Telegram.

**Telegram**

* Receives real-time alerts:
  + Trade open/close
  + SL/TP hits
  + Wallet deductions
  + Invoice due
  + Bot paused/unpaused

**🗄️ Central Database**

Built on PostgreSQL or TimescaleDB with ACID compliance.

**Key Tables:**

* users
* user\_wallets
* trade\_audit\_logs
* strategy\_configs
* invoices
* broker\_credentials (AES-encrypted)

**🔷 2. Trade Sentry Workflow & User Lifecycle**

1. **User Registration & Payment**
   * Registers on Web/Mobile.
   * Pays initial subscription fee.
2. **Broker Linking**
   * Securely connects broker account via API.
   * Credentials are encrypted and stored.
3. **Bot Configuration**
   * Selects strategy (DRLFusion, SMEdge, SMGrid, etc).
   * Chooses symbol, lot, SL/TP method, risk %.
4. **Execution & Monitoring**
   * Bot fetches market data.
   * Executes trade via BrokerIntegrationService.
   * TradingMonitoringService watches for exits.
5. **Post-Trade Actions**
   * On profit:
     + Audit log created.
     + WalletService deducts platform share.
   * On loss:
     + Only audit log recorded.
6. **Wallet Billing & Enforcement**
   * If wallet is low:
     + Invoice is created.
     + Telegram alert sent.
     + Bot is paused until wallet is topped up.

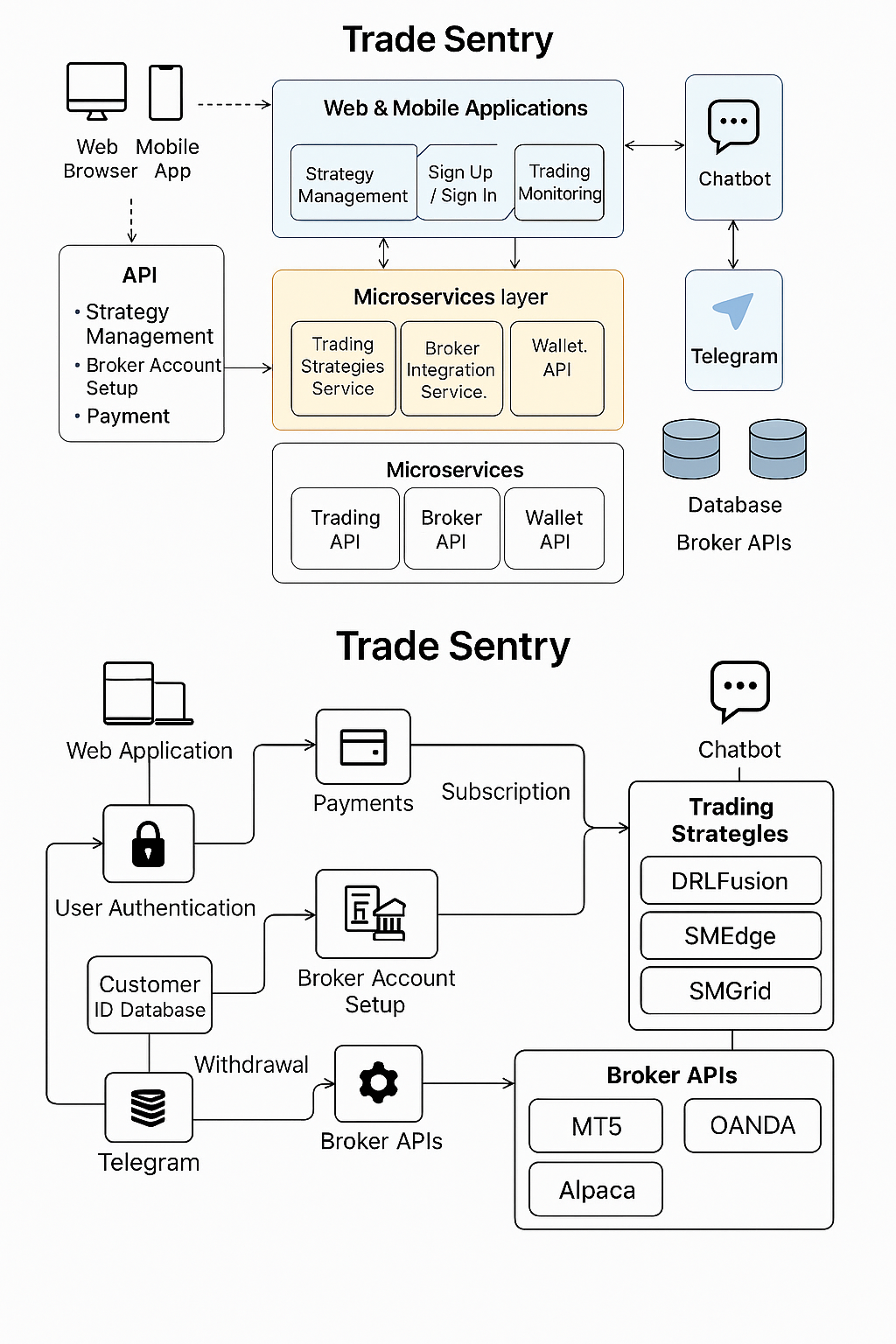


Figure : TradeSentry AI Workflow & User Lifecycle

**🔷 3. End-to-End Technical Architecture of TradeSentry AI**

This section provides a detailed breakdown of TradeSentry AI’s system architecture, organized into three core layers — user interaction, API and microservices, and backend infrastructure — illustrating how frontend interfaces, trading logic, and backend systems are seamlessly integrated to support secure, scalable, and automated trading operations.

**🔹 A. User Interaction Layer**

**✅ Web & Mobile UI**

* **Frontend Technologies**: ReactJS (Web), React Native or Flutter (Mobile)
* **Functions**:
  + Browse strategy marketplace
  + Set parameters for bots (lot size, symbol, SL/TP method)
  + Manage wallet: top-ups, view deductions and invoices
  + Monitor trade performance and active positions

**✅ Chatbot Integration**

* Smart assistant integrated in-app and via Telegram
* Examples:
  + “How do I fund my wallet?”
  + “Why was my bot paused?”
  + “Show last trade PnL”

**✅ Telegram Bot**

* Push-based notification channel
* Message Types:
  + Trade opened/closed
  + Wallet balance updates
  + Bot state change (started, paused, stopped)
  + Payment reminder for invoices

**🔹 B. API Gateway & Microservices**

**✅ API Gateway**

* Stateless, RESTful architecture using FastAPI
* JWT-based authentication
* Throttling & role-based access control
* All requests to strategy start/stop, broker link, wallet, etc., are routed here

**✅ Microservices**

**1. TradingStrategiesService**

* Strategy core logic
* Supports:
  + Real-time execution
  + MTF (Multi-timeframe) alignment
  + Stop management
  + Risk control

**2. BrokerIntegrationService**

* Uses SDKs and HTTP interfaces for:
  + Order execution
  + Balance & equity fetching
  + Candle and tick retrieval

**3. TradingMonitoringService**

* Loops through user strategies and:
  + Evaluates trade exit conditions
  + Handles timeouts or SL/TP
  + Notifies on execution outcomes

**4. WalletService**

* Manages virtual balances
* Deducts % profit from wallet on each profitable trade
* Generates invoices for unpaid profit shares
* Connects with Stripe or PayPal for wallet top-ups

**🔹 C. Backend Infrastructure Layer**

**✅ Wallet API**

* Top-up endpoint (webhook-enabled)
* Balance checker before trade execution
* Logs:
  + Debit events
  + Failures
  + Balance updates

**✅ Broker API**

* Supports:
  + submit\_order()
  + close\_order()
  + get\_positions()
  + get\_ohlcv()
* Separate adapters for MT5, OANDA, Alpaca, and Binance

**✅ Database (PostgreSQL + Redis + TimescaleDB)**

* user\_wallets: balance per user
* trade\_audit\_logs: full trade lifecycle data
* invoices: unpaid, pending, paid status tracking
* user\_strategies: tracks active bots per user

**🔷 4. Performance Fee Enforcement Flow**

**Key Enforcement Workflow:**

1. A trade closes with profit.
2. The system computes net profit (after fees, spreads).
3. WalletService: ***Is Wallet Balance ≥ Performance Fee?***
   * ✅ **Yes:**
     + Deduct Performance Fee from User Wallet
     + Log to Audit DB (i.e., trade\_audit\_logs table)
     + Strategy Continues
   * ❌ **No:**
     + Generate/issue Invoice
     + Send Telegram Alert and Dashboard Warning
     + Pause Strategy (without affecting open trades)
     + User updates Wallet to trigger Strategy Resumption

**Why This Works:**

* **User retains custody** of funds.
* Enforcement happens through **platform logic and access control**, not fund control.
* Performance-based revenue model remains legally enforceable with **Terms of Use**.

**🔷 5. Wallet & Profit Audit Schema**

**📑 user\_wallets**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| user\_id | UUID | Foreign key to users |
| balance | FLOAT | Current wallet amount |
| currency | VARCHAR | e.g., USD |
| last\_updated | TIMESTAMP | Last transaction update |

**📑 trade\_audit\_logs**

|  |  |  |
| --- | --- | --- |
| **Field** | **Type** | **Description** |
| trade\_id | UUID | Unique trade log ID |
| user\_id | UUID | User executing trade |
| strategy\_key | VARCHAR | e.g., 'drlfusion', 'smedge' |
| entry\_price | FLOAT | Entry price |
| exit\_price | FLOAT | Exit price |
| gross\_profit | FLOAT | Raw trade profit |
| platform\_fee | FLOAT | Deducted % (e.g., 30%) |
| debit\_status | ENUM | success / failed / pending |
| created\_at | TIMESTAMP | Trade close time |

**🔷 6. Implementation Roadmap**

|  |  |  |
| --- | --- | --- |
| Phase | Duration | Key Milestones |
| MVP | 1–6 wks | DRLFusion + wallet integration + basic monitoring |
| Beta | 6–8 wks | Add SMEdge, Telegram bot, test wallet fee enforcement |
| Launch | 9–10 wks | Full strategy marketplace, Stripe integration, bot suspension |
| Scale-Up | 11–16 wks | Affiliate system, compliance, pro user API, multi-bot scaling |

**🔷 7. Terms of Use Addendum: Wallet & Billing**

**🧾 7.1 Wallet-Based Profit Share**

Users agree to pre-fund a wallet on TradeSentry AI. The wallet will be automatically debited for the profit share percentage on successful trades.

**📉 7.2 Bot Suspension Policy**

If wallet balance < required fee:

* The bot is paused
* User is notified via UI and Telegram
* An invoice is generated with 3-day grace period

**✅ 7.3 Consent**

Users digitally sign this policy during onboarding.  
A checkbox stating:

“I agree to the Terms of Use, including wallet-based performance billing.”

is presented and stored in the audit log.