Forex Agent Architecture Report

Project: Autonomous Forex Trading System **Architecture Pattern:** Modular Pipeline Micro-Component **Date:** 2026-01-21

1. High-Level Architectural Overview

The Forex Agent is designed as a **modular, sequential pipeline**. Unlike monolithic trading bots, this system separates concerns into distinct, loosely coupled layers. This ensures that any single component (e.g., the Strategy or the Broker Provider) can be swapped out without affecting the rest of the system.

Core Philosophy

1. **Modularity:** Each component has a strict input/output contract.
2. **Safety First:** Risk management is an independent gatekeeper.
3. **Broker Agnostic:** The core logic does not know which broker is executing the trade.

---

2. Component Detail Analysis

2.1. Data Layer (The Foundation)

Responsibility: Reliable ingestion and normalization of market data. * **Raw Ingestion:** Connects to external APIs (currently mocked or IG Markets) to fetch Candlestick data (OHLCV). * **Normalization:** Converts varied API responses into a unified internal DataFrame schema (`timestamp`, `open`, `high`, `low`, `close`, `volume`). * **Persistence:** Saves raw and processed data to `data/` directory for auditability and backtesting.

2.2. Signal Engine (The Brain)

Responsibility: Pure market analysis and decision making. * **Input:** Normalized DataFrames. * **Logic:** Executes technical strategies (e.g., `BaselineSMACross`). The logic is strictly mathematical—it does not know about account balances or risk limits. * **Output:** `SignalIntent` objects containing direction (LONG/SHORT), confidence scores, and raw entry/stop suggestions.

2.3. Risk Management Layer (The Gatekeeper)

Responsibility: Protecting capital and enforcing rules. * **Strict Separation:** This layer sits *between* the Signal Engine and the Execution. * **Validation:** Checks signals against: * **Max Daily Loss:** Stops all trading if a threshold (e.g., 2%) is reached. * **Position Sizing:** Calculates exact lot sizes based on account equity and risk-per-trade (e.g., 1%). * **Drawdown Limits:** Prevents over-leveraging. * **Output:** `RiskIntent` (Approved) or `Rejection` (Denied).

2.4. Execution Layer (The Hands)

Responsibility: Interfacing with the external world (Brokers). * **Adapter Pattern:** Uses an abstract `BrokerInterface`. * **Implementations:** * `MockBroker`: For paper trading and pipeline verification. * `IGBroker`: For real-market connectivity. * **Router Logic:** Takes a `RiskIntent` and routes it to the active broker adapter, translating internal units to broker-specific contracts (e.g., converting 100,000 units to "1 Lot").

2.5. Orchestration & Observability (The Nervous System)

Responsibility: Scheduling and Monitoring. * **Runner:** `run_cycle.py` (Planned) executes the full pipeline in a single pass. * **Scheduling:** Designed to run via Systemd, N8N, or Cron triggers. * **Logging:** Centralized structured logging for debugging. * **Notebook:** Automated PDF reporting (this system) for progress tracking.

---

3. Data Flow Diagram

`Raw Data API` -> **[Data Layer]** -> `Normalized DF` -> **[Signal Engine]** -> `Signal Intent` -> **[Risk Layer]** -> `Order Intent` -> **[Execution Router]** -> `Broker API`

4. Future Architecture Readiness

The current architecture is built to support: * **Multi-Strategy:** Running multiple strategies in parallel. * **Database Integration:** Swapping CSV storage for TimeScaleDB or Postgres. * **Dashboarding:** Exposing a read-only API for a Streamlit or React frontend.