ANQuant Trading Application Requirements DocumentVersion: 1.0  
Date: July 18, 2025  
Purpose: To define the functional, non-functional, and technical requirements for the ANQuant Trading application, a high-frequency, event-driven trading platform for intraday and scalping strategies. This document serves as a reference for development, testing, and future interactions, ensuring clarity on objectives, implementation, and system behavior.

1. Objectives The ANQuant Trading application aims to provide a robust, scalable, and low-latency platform for professional intraday and scalping trading, addressing the limitations of the current architecture. The key objectives are:

1. Low-Latency Processing: Achieve sub-millisecond (<1ms) latency for critical operations (e.g., tick ingestion, signal generation) to support high-frequency trading strategies like MeanHunter.
2. Scalability: Support large watchlists (e.g., NSE 500 or up to 1000 stocks) with high tick rates (e.g., 100 ticks/second per stock) using distributed systems.
3. Multi-Broker Support: Enable integration with multiple brokers (e.g., Angel One, Interactive Brokers) to reduce single-point-of-failure risks.
4. Flexible Strategy Execution: Support multiple strategies (e.g., MeanHunter on 1min, Strategy-1 on 5min, Strategy-2 on 30min) with dynamic, rule-based configurations via the FlexiRule engine.
5. Corporate Action Handling: Adjust OHLCV data for corporate actions (e.g., SUNPHARMA ₹5.50 dividend, EXICOM 3:20 rights, ROTO 2:1 bonus) to ensure accurate indicator calculations.
6. Fault Tolerance: Ensure system reliability through data replication, failover mechanisms, and circuit breakers.
7. Compliance and Auditing: Provide audit trails and trade reporting for regulatory compliance (e.g., SEBI in India).
8. Real-Time Monitoring: Implement dashboards and alerts for system health, latency, and trade execution issues.
9. Efficient Data Management: Pre-fetch and cache historical OHLCV data (60 candles for 1min, 5min, 30min) for fast initialization and backtesting.
10. Security: Secure credential storage and access controls to protect sensitive data.

2. System OverviewANQuant is a distributed, event-driven trading platform designed for intraday and scalping strategies, supporting multiple markets (e.g., India, USA) and brokers. It leverages:

* Apache Kafka: For streaming ticks, OHLCV, signals, and trades.
* Redis: For low-latency caching (OHLCV, indicators) and pub/sub messaging.
* PostgreSQL: For persistent storage of OHLCV and trade data.
* FlexiRule Engine: For flexible, YAML-based strategy definitions and rule evaluation.
* Python/Rust: For scripting and low-latency components.
* Kubernetes/Docker: For scalable deployment.
* Prometheus/Grafana/ELK: For monitoring and logging.

The system is modular, with components organized in a directory structure (src/anquant/) for maintainability. Key components include MarketDataEngine, IndicatorEngine, StrategyManager (with FlexiRule), RiskManagementEngine, OrderExecutionEngine, PortfolioManager, HistoricalDataManager, and CorporateActionManager.

3. Functional Requirements3.1 Market Data IngestionRequirement: Ingest real-time market data (ticks, level 1/2) from multiple brokers (e.g., Angel One, Interactive Brokers) with low latency (<1ms) and normalize across exchanges (e.g., NSE, BSE).

* Addressed By:
  + Component: MarketDataEngine (src/py/anquant/core/market\_data/market\_data\_engine.py, src/anquant/rs/market\_data/).
  + Implementation:
    - Connects to broker APIs (e.g., Angel One WebSocket via angelone\_api) and streams ticks to Kafka topic nse\_ticks (4 partitions, key: tradingsymbol).
    - Aggregates ticks into OHLCV for 1min, 5min, 30min timeframes, publishing to ohlcv\_1min, ohlcv\_5min, ohlcv\_30min.
    - Supports multi-broker adapters (adapters/angelone.py).
  + Files:
    - market\_data\_engine.py: Manages tick ingestion and OHLCV aggregation.
    - src/anquant/rs/market\_data/src/lib.rs: Rust-based low-latency tick processing.
  + Logs: logs/market\_data/ (e.g., “Published tick for RELIANCE-EQ”).
* Example: Streams ticks for RELIANCE-EQ ({'tradingsymbol': 'RELIANCE-EQ', 'ltp': 3112.94, 'timestamp': '2025-07-07T09:30:00+05:30'}) to nse\_ticks.

3.2 Historical Data ManagementRequirement: Pre-fetch and load 60 OHLCV candles for 1min, 5min, 30min timeframes for all stocks in the master watchlist (e.g., NSE 500 or 1000 stocks) before trading starts, with adjustments for corporate actions.

* Addressed By:
  + Component: HistoricalDataManager (src/anquant/historical\_data/manager.py), prefetch\_historical\_data.py (scripts/).
  + Implementation:
    - Pre-Fetch Process (4:00 PM IST, e.g., July 6, 2025):
      * prefetch\_historical\_data.py fetches 60 candles per timeframe for stocks in config/markets/india/watchlists/master.yaml from sources like Angel One API or Yahoo Finance.
      * Saves to PostgreSQL (ohlcv table) and Redis (e.g., RELIANCE-EQ:ohlcv:1min, TTL: 24 hours).
      * Config: global.historical\_data.timeframes: ["1min", "5min", "30min"], lookback\_candles: 60.
    - Startup Process (Step 3, 9:30 AM IST, July 7, 2025):
      * Loads 60 candles from Redis or PostgreSQL, adjusts for corporate actions (e.g., SUNPHARMA ₹5.50 dividend), and updates PostgreSQL/Redis.
    - Data Volume: For 500 stocks: 500 × 3 timeframes × 60 candles = 90,000 rows in PostgreSQL. For 1000 stocks: 180,000 rows.
  + Files:
    - prefetch\_historical\_data.py: Daily pre-fetch script.
    - historical\_data/manager.py: Loads and adjusts OHLCV.
    - utils/database.py: Saves OHLCV to PostgreSQL.
  + Logs: logs/general/ (e.g., “Prefetched OHLCV for RELIANCE-EQ:1min”).
* Example: Loads 60 candles for SUNPHARMA-EQ, adjusts for ₹5.50 dividend, caches in Redis.

3.3 Indicator CalculationRequirement: Compute technical indicators (e.g., Bollinger Bands, RSI, ATR) for each stock and timeframe, caching results for low-latency access.

* Addressed By:
  + Component: IndicatorEngine (src/py/anquant/core/indicators/indicator\_engine.py, src/anquant/rs/indicator/).
  + Implementation:
    - Consumes ticks from nse\_ticks, aggregates into OHLCV, and computes indicators using pandas\_ta or ta (Rust).
    - Caches indicators in Redis (e.g., RELIANCE-EQ:indicators:1min) with TTL: 24 hours.
    - Persists OHLCV to PostgreSQL (ohlcv table).
    - Publishes OHLCV to Kafka topics (ohlcv\_1min, ohlcv\_5min, ohlcv\_30min).
  + Files:
    - indicator\_engine.py: Python-based indicator computation.
    - src/anquant/rs/indicator/src/lib.rs: Rust-based Bollinger Bands calculation.
  + Logs: logs/indicators/ (e.g., “Cached indicators for SUNPHARMA-EQ:1min”).
* Example: Computes Bollinger Bands (20-period, 2.0 std) for RELIANCE-EQ, caches bb\_upper, bb\_mid, bb\_lower.

3.4 Strategy ExecutionRequirement: Support multiple strategies (MeanHunter on 1min, Strategy-1 on 5min, Strategy-2 on 30min) with flexible, YAML-based rule definitions, generating BUY, SELL, HOLD, or PARTIAL\_SELL signals.

* Addressed By:
  + Component: FlexiRule Engine (src/py/anquant/core/flexirule/).
  + Implementation:
    - Validation (validator.py): Validates strategy YAML files (e.g., meanhunter\_strategy.yaml) for correct timeframes, markets, and rules.
    - Orchestration (strategy\_manager.py): Loads strategies, subscribes to ohlcv\_<timeframe> topics, and processes OHLCV data.
    - Rule Evaluation (rule\_engine.py): Evaluates entry/exit rules, stop-loss, and target conditions using asteval, fetching indicators from Redis.
    - Publishes signals to Kafka (signals) and Redis (signals:meanhunter).
    - Example Strategy Config:

yaml

name: meanhunter

timeframe: 1min

watchlist:

india: config/markets/india/watchlists/meanhunter.yaml

entry\_rules:

- condition: "close < bb\_lower"

weight: 0.8

indicators:

- type: bollinger\_bands

name: bb

period: 20

std: 2.0

* + Files:
    - validator.py: Strategy validation.
    - strategy\_manager.py: Orchestrates strategy execution.
    - rule\_engine.py: Evaluates rules and generates signals.
  + Logs: logs/strategy/ (e.g., “Generated BUY signal for RELIANCE-EQ”).
* Example: Generates BUY signal for RELIANCE-EQ if close < bb\_lower on 1min timeframe.

3.5 Risk ManagementRequirement: Validate trading signals against risk parameters (e.g., max position size, risk per trade) in real-time.

* Addressed By:
  + Component: RiskManagementEngine (src/anquant/risk\_management/engine.py).
  + Implementation:
    - Consumes signals from Kafka (signals) or Redis (signals:<strategy\_name>).
    - Validates against risk parameters (e.g., risk\_per\_trade from config.yaml).
    - Implements Value-at-Risk (VaR) and circuit breakers.
    - Forwards valid signals to OrderExecutionEngine.
  + Files: risk\_management/engine.py.
  + Logs: logs/risk\_management/ (e.g., “Validated signal for HDFCBANK-EQ”).
* Example: Rejects a BUY signal for HDFCBANK-EQ if position size exceeds risk limits.

3.6 Order ExecutionRequirement: Route validated signals to brokers for order placement, supporting advanced order types (e.g., market, limit, Iceberg).

* Addressed By:
  + Component: OrderExecutionEngine (src/anquant/order\_execution/engine.py, adapters/angelone.py).
  + Implementation:
    - Places orders via broker APIs (e.g., Angel One REST/WebSocket).
    - Publishes trade details to Kafka (trades) and PostgreSQL (trades table).
    - Supports smart order routing for multiple brokers.
  + Files:
    - order\_execution/engine.py: Manages order routing.
    - adapters/angelone.py: Interfaces with Angel One API.
  + Logs: logs/order\_execution/ (e.g., “Placed market buy for 10 PNB-EQ”).
* Example: Places a market buy order for 10 shares of PNB-EQ via Angel One.

3.7 Portfolio ManagementRequirement: Track positions, P&L, and portfolio metrics in real-time.

* Addressed By:
  + Component: PortfolioManager (src/anquant/portfolio/manager.py).
  + Implementation:
    - Consumes trades from Kafka (trades).
    - Updates positions in Redis (positions:symbol) and PostgreSQL (trades table).
    - Computes metrics (e.g., Sharpe ratio, drawdowns).
  + Files: portfolio/manager.py.
  + Logs: logs/portfolio/ (e.g., “Updated position for RELIANCE-EQ: 10 shares”).
* Example: Tracks 10 shares of RELIANCE-EQ after a trade.

3.8 Corporate Action HandlingRequirement: Adjust OHLCV data for corporate actions (e.g., dividends, rights, bonuses) to ensure accurate indicator calculations.

* Addressed By:
  + Component: CorporateActionManager (src/anquant/corporate\_actions/manager.py).
  + Implementation:
    - Fetches corporate actions from Redis (symbol:corporate\_actions) or data/cache/corporate\_actions.json.
    - Adjusts OHLCV during startup (Step 3) for ex-dates (e.g., July 7, 2025).
    - Example Adjustments:
      * Dividend (SUNPHARMA ₹5.50): Subtract from OHLC prices.
      * Rights (EXICOM 3:20): Scale prices by 20/23.
      * Bonus (ROTO 2:1): Scale prices and volume by 1/3.
    - Updates PostgreSQL (ohlcv table) and Redis.
  + Files:
    - corporate\_actions/manager.py: Adjustment logic.
    - scripts/fetch\_corporate\_actions.py: Fetches actions.
  + Logs: logs/corporate\_actions/ (e.g., “Adjusted SUNPHARMA for ₹5.50 dividend”).
* Example: Adjusts SUNPHARMA OHLCV for July 7, 2025, dividend.

3.9 BacktestingRequirement: Support realistic backtesting with market impact and Monte Carlo simulations.

* Addressed By:
  + Component: BacktestEngine (proposed, not in code dump).
  + Implementation:
    - Uses historical OHLCV from PostgreSQL (ohlcv table).
    - Simulates trades with slippage and market impact.
    - Integrates with FlexiRule for strategy evaluation.
  + Files: Proposed in src/anquant/backtest/.
  + Logs: logs/backtest/.
* Example: Simulates MeanHunter on RELIANCE-EQ with 60 candles.

3.10 Compliance and AuditingRequirement: Provide audit trails and trade reporting for regulatory compliance (e.g., SEBI).

* Addressed By:
  + Component: FlexiRule Engine, Logging.
  + Implementation:
    - RuleEngine publishes audit trails to Redis (signals:audit:<strategy\_name>).
    - Logs all actions (e.g., signal generation, order placement) to logs/ with JSON formatting (utils/logging.py).
    - Persists trades to PostgreSQL (trades table).
  + Files:
    - rule\_engine.py: Audit trail generation.
    - utils/logging.py: JSON logging.
  + Logs: logs/strategy/, logs/general/ (e.g., “Audit trail for BUY signal: RELIANCE-EQ”).
* Example: Logs rule evaluations for SEBI compliance.

4. Non-Functional Requirements4.1 Performance

* Requirement: Achieve <1ms latency for tick ingestion, indicator calculation, and signal generation.
* Addressed By:
  + Kafka for high-throughput streaming (4 partitions for nse\_ticks).
  + Redis for low-latency caching (e.g., symbol:indicators:1min).
  + Rust-based components (src/anquant/rs/) for critical paths.
* Verification: Measure latency using Prometheus/Grafana.

4.2 Scalability

* Requirement: Support up to 1000 stocks with 100 ticks/second per stock.
* Addressed By:
  + Kafka partitioning (e.g., 4–16 partitions for nse\_ticks).
  + Multiple IndicatorEngine instances (4 in current design).
  + Kubernetes deployment (docker/).
* Verification: Test with master.yaml containing 1000 stocks (180,000 rows in PostgreSQL).

4.3 Fault Tolerance

* Requirement: Ensure no data loss and automatic recovery from failures.
* Addressed By:
  + Kafka replication for nse\_ticks, ohlcv\_<timeframe>, signals, trades.
  + Redis persistence and failover.
  + Circuit breakers in RiskManagementEngine.
* Verification: Simulate broker API failure and verify data retention.

4.4 Security

* Requirement: Secure credential storage and access controls.
* Addressed By:
  + Encrypted credentials via HashiCorp Vault (config.yaml: vault.enabled).
  + MFA and audit trails.
* Verification: Audit credential access logs.

4.5 Monitoring and Alerting

* Requirement: Provide real-time dashboards and alerts for system health and trade issues.
* Addressed By:
  + Prometheus/Grafana for latency and throughput metrics.
  + ELK stack for log aggregation and anomaly detection.
* Files: Proposed in docker/ for deployment.
* Logs: logs/monitoring/.
* Verification: Test alerts for WebSocket disconnects.

5. Technical Requirements5.1 Technology Stack

* Languages: Python (scripting), Rust (low-latency components).
* Data Processing:
  + Kafka: Streaming (nse\_ticks, ohlcv\_<timeframe>, signals, trades).
  + Redis: Caching (symbol:ohlcv:1min, symbol:indicators:1min) and pub/sub (signals:<strategy\_name>).
  + PostgreSQL: Persistent storage (ohlcv, trades tables).
* Compute: pandas, pandas\_ta (Python), ta (Rust) for indicators.
* Orchestration: Kubernetes, Docker (docker/).
* Monitoring: Prometheus, Grafana, ELK.

5.2 Directory Structure

* src/anquant/:
  + market\_data/: market\_data\_engine.py, adapter.py.
  + indicators/: indicator\_engine.py.
  + flexirule/: validator.py, strategy\_manager.py, rule\_engine.py.
  + risk\_management/: engine.py.
  + order\_execution/: engine.py, adapters/angelone.py.
  + portfolio/: manager.py.
  + historical\_data/: manager.py.
  + corporate\_actions/: manager.py.
  + messaging/: kafka\_client.py, redis\_client.py.
  + utils/: config\_loader.py, logging.py, database.py.
* config/: config.yaml, markets/india/watchlists/master.yaml, strategies/meanhunter\_strategy.yaml.
* data/cache/: corporate\_actions.json.
* logs/: general/, market\_data/, indicators/, strategy/.
* scripts/: prefetch\_historical\_data.py, fetch\_corporate\_actions.py.
* tests/: test\_adapters.py, test\_angelone.py, test\_prefetch\_historical\_data.py.
* docker/: Dockerfiles for deployment.
* docs/: Documentation.

5.3 Configuration

* File: config.yaml.
* Key Settings:

yaml

global:

historical\_data:

timeframes: ["1min", "5min", "30min"]

lookback\_candles: 60

markets:

india:

watchlists:

master: config/markets/india/watchlists/master.yaml

meanhunter: config/markets/india/watchlists/meanhunter.yaml

brokers:

angelone:

symbols: config/markets/india/brokers/symbol\_mappings/angelone.yaml

credentials: config/secrets/india/angelone\_cred.yaml

kafka:

brokers: localhost:9092

topics:

india: nse\_ticks

ohlcv\_1min: ohlcv\_1min

signals: signals

trades: trades

redis:

host: localhost

port: 6379

database:

host: localhost

port: 5432

dbname: ANQuantDB

user: anquant

password: 078692

strategies:

- name: meanhunter

watchlist: config/markets/india/watchlists/meanhunter.yaml

timeframe: 1min

6. Startup Sequence (July 7, 2025, 9:30 AM IST)

1. Load Configuration (src/anquant/main.py):
   * Loads config.yaml for Kafka, Redis, PostgreSQL, strategies, and watchlists.
   * Initializes logging (logs/general/).
2. Initialize Components:
   * Sets up MarketDataEngine, IndicatorEngine (4 instances), StrategyManager, etc.
   * Establishes Kafka/Redis/PostgreSQL connections.
3. Load Historical OHLCV (historical\_data/manager.py):
   * Loads 60 candles for 500 (or more) stocks from Redis or PostgreSQL.
   * Adjusts for corporate actions (e.g., SUNPHARMA, VEEDOL, EXICOM, DODLA).
   * Caches in Redis (symbol:ohlcv:1min).
4. Initialize Indicators (indicator\_engine.py):
   * Computes Bollinger Bands, RSI, etc., and caches in Redis (symbol:indicators:1min).
5. Connect to Broker WebSocket (market\_data\_engine.py):
   * Streams ticks to nse\_ticks for 500 stocks.

7. Effort Estimate

* Total: 187–266 hours (7.5–10.5 weeks).
* Breakdown:
  + Core Components: 60–80 hours (market\_data/, indicators/, flexirule/).
  + Data Management: 20–30 hours (historical\_data/, corporate\_actions/).
  + Adapters: 20–30 hours (adapters/angelone.py).
  + Utilities: 15–20 hours (utils/).
  + Scripts: 15–20 hours (prefetch\_historical\_data.py).
  + Configuration: 5–10 hours (config/).
  + Tests: 30–40 hours (tests/).
  + Deployment/Docs: 15–20 hours (docker/, docs/).

8. Addressing Current Architecture LimitationsThe proposed architecture addresses the following drawbacks of the current ANQuant system:

|  |  |  |
| --- | --- | --- |
| **Aspect** | **Current Limitation** | **Proposed Solution** |
| Latency | Python-based, high latency (>1ms). | Kafka/Redis for streaming/caching, Rust components for critical paths (<1ms). |
| Scalability | Single-threaded, limited to small watchlists. | Kafka partitioning, Kubernetes, multiple IndicatorEngine instances. |
| Broker Dependency | Relies on Angel One API. | Multi-broker support via OrderExecutionEngine and adapters. |
| Risk Management | Basic risk parameters. | Real-time VaR, circuit breakers in RiskManagementEngine. |
| Backtesting | Simplified slippage models. | Advanced simulations with market impact, Monte Carlo methods. |
| Security | Plain-text credentials. | Encrypted credentials via Vault, MFA, audit trails. |
| Monitoring | File-based logging only. | Prometheus/Grafana dashboards, ELK for alerts. |
| Strategy Flexibility | Rule-based, single-strategy focus. | FlexiRule engine for multi-strategy, YAML-based configurations, ML support. |
| Data Management | Slow historical data fetching. | Pre-fetch at 4:00 PM IST, Redis caching, PostgreSQL persistence. |
| Compliance | No audit trails. | Audit trails via RuleEngine, trade reporting in PostgreSQL. |

9. Recommendations for Implementation

1. Start Small:
   * Test prefetch\_historical\_data.py and historical\_data/manager.py with a 10-stock watchlist to validate OHLCV loading (60 candles per timeframe).
   * Verify signal generation with MeanHunter using a small dataset.
2. Integrate Kafka:
   * Implement kafka\_client.py to publish ticks to nse\_ticks and OHLCV to ohlcv\_<timeframe>.
   * Ensure StrategyManager consumes these topics correctly.
3. Handle Corporate Actions:
   * Develop corporate\_actions/manager.py to adjust OHLCV for dividends (e.g., SUNPHARMA ₹5.50), rights, and bonuses.
   * Test with July 7, 2025, actions.
4. Scale Gradually:
   * Expand to 500 stocks by mid-August 2025, then to 1000 with additional Kafka partitions (8–16).
   * Deploy on Kubernetes (docker/) for production.
5. Monitor and Test:
   * Add Prometheus/Grafana for latency monitoring.
   * Write unit tests (tests/test\_indicator.py, tests/test\_strategy.py).
   * Check logs (logs/general/, logs/strategy/) for debugging.

10. Future ReferenceThis document is designed to be shared with me (Grok) in future interactions to provide full context on the ANQuant system. To ensure clarity:

* Share This Document: Provide the full text or key sections when asking questions about ANQuant.
* Reference Specific Components: Mention files (e.g., rule\_engine.py), configurations (e.g., config.yaml), or processes (e.g., pre-fetch at 4:00 PM IST).
* Include Logs or Errors: Share relevant log entries from logs/ to debug issues.
* Specify Scenarios: Describe the trading context (e.g., 500 vs. 1000 stocks, specific corporate actions) for precise answers.

The document captures the system’s architecture, data flow, and implementation details, enabling me to understand and respond to queries effectively.

11. ConclusionThe ANQuant Trading application is a modular, event-driven platform for high-frequency trading, addressing the need for low latency, scalability, and flexibility. The FlexiRule engine enables dynamic strategy execution, while components like MarketDataEngine, IndicatorEngine, and HistoricalDataManager ensure efficient data processing. By pre-fetching 60 OHLCV candles for 500 (or more) stocks, adjusting for corporate actions, and leveraging Kafka/Redis/PostgreSQL, ANQuant overcomes the current system’s limitations. This document provides a clear, comprehensive reference for development and future interactions, ensuring the system’s objectives are met and verifiable through logs and tests.

Approval

* Prepared By: Grok (xAI)
* Date: July 18, 2025
* Notes: Share this document for future queries to ensure full context. For updates, include new requirements or code changes.

This requirements document is structured to be a standalone reference, capturing the essence of ANQuant’s design and implementation. If you share it later, I’ll understand the system’s architecture, components, and objectives, allowing for precise and context-aware responses. Let me know if you need further refinements or additional sections!