**Quant Model Development - System Design Document**

**1. Project Overview**

**1.1 Objective**

Develop a quantitative trading model with an end-to-end implementation, including data analysis, strategy development, order execution, and risk management using Python and object-oriented programming principles.

**1.2 Scope**

* Fetch and preprocess market data
* Conduct exploratory data analysis (EDA) and feature engineering
* Develop and backtest trading strategies
* Implement order execution and broker API integration
* Manage portfolio risk and optimize allocations
* Deploy an automated trading system

**2. Requirement Gathering**

**2.1 Functional Requirements**

| **ID** | **Requirement Description** |
| --- | --- |
| FR-1 | System should fetch real-time and historical market data from APIs (Yahoo Finance, Alpaca, Binance, etc.) |
| FR-2 | Implement and test multiple trading strategies (e.g., Mean Reversion, Momentum) |
| FR-3 | Perform backtesting using historical data to validate strategies |
| FR-4 | Execute trades through broker APIs (Interactive Brokers, Alpaca, Binance) |
| FR-5 | Implement risk management (Stop Loss, Position Sizing, Value at Risk) |
| FR-6 | Provide real-time performance tracking and reporting |
| FR-7 | Deploy the system for automated trading with alerts & logging |

**2.2 Non-Functional Requirements**

| **ID** | **Requirement Description** |
| --- | --- |
| NFR-1 | System should be modular and extensible for adding new strategies |
| NFR-2 | Data processing should be optimized for low-latency trading |
| NFR-3 | Ensure logging and error handling for debugging |
| NFR-4 | Use database storage for trade history and performance tracking |
| NFR-5 | Secure API keys and sensitive credentials |

**2.3 Constraints**

* API rate limits for real-time data fetching
* Execution speed (latency may affect high-frequency trading strategies)
* Broker API restrictions for trade placement

**3. System Architecture**

**3.1 High-Level Architecture**

The system consists of the following modules:

1. **Data Layer**: Fetches and stores market data
2. **Analysis Layer**: Conducts data processing and strategy evaluation
3. **Execution Layer**: Manages order execution and trade placement
4. **Risk Management Layer**: Implements portfolio optimization and stop-loss mechanisms
5. **Reporting Layer**: Generates trade performance reports

**4. Class Diagram**

**4.1 UML Class Diagram Description**

**Main Classes and Responsibilities:**

* MarketDataFetcher → Fetches real-time/historical market data
* Strategy (Abstract Class) → Parent class for all strategies (Momentum, Mean Reversion)
* Backtester → Simulates strategy performance on historical data
* TradeExecution → Handles order placement with a broker API
* RiskManager → Implements risk control measures
* PerformanceTracker → Monitors P&L and generates reports

**4.2 Class Diagram (Text Representation)**

+-------------------+

| MarketDataFetcher |

|-------------------|

| +fetch\_data() |

+-------------------+

|

v

+-------------------+

| Strategy (Abstract) |

|-------------------|

| +generate\_signal() |

+-------------------+

|

v

+------------------+ +----------------+ +------------------+

| MomentumStrategy | | MeanReversion | | Backtester |

|------------------| |----------------| |------------------|

| +generate\_signal()| | +generate\_signal()| | +run\_backtest() |

+------------------+ +----------------+ +------------------+

|

v

+------------------+

| TradeExecution |

|------------------|

| +execute\_order() |

+------------------+

|

v

+------------------+

| RiskManager |

|------------------|

| +apply\_risk\_rules() |

+------------------+

|

v

+------------------+

| PerformanceTracker |

|------------------|

| +track\_performance() |

+------------------+

**5. Data Flow Diagram (DFD)**

**5.1 Data Flow Overview**

1. **Market Data Ingestion** → Fetch real-time/historical data → Store in database
2. **Strategy Processing** → Analyze data & generate buy/sell signals
3. **Backtesting Module** → Simulate strategy performance on historical data
4. **Order Execution** → Send orders to broker API
5. **Risk Management** → Adjust positions based on risk factors
6. **Performance Tracking** → Log trades and generate reports

**6. Technology Stack**

| **Component** | **Technology** |
| --- | --- |
| Programming Language | Python |
| Data Fetching | yfinance, ccxt, Alpaca API |
| Backtesting Framework | Backtrader, Zipline |
| Database | PostgreSQL, SQLite |
| Execution API | Alpaca, Binance, Interactive Brokers |
| Visualization | Matplotlib, Plotly, Dash |
| Deployment | AWS, Docker, Airflow (for automation) |

**7. Phased Development Plan**

**Phase 1: Data Collection & Processing**

* Implement MarketDataFetcher class
* Store market data in a structured format

**Phase 2: Strategy Development & Backtesting**

* Develop Strategy abstract class and specific strategies
* Create Backtester to evaluate performance

**Phase 3: Order Execution & API Integration**

* Implement TradeExecution for broker interaction
* Add logging for executed orders

**Phase 4: Risk Management & Optimization**

* Develop RiskManager for position sizing and loss protection

**Phase 5: Deployment & Monitoring**

* Automate data fetching, trading, and reporting
* Implement a monitoring dashboard

**8. Conclusion**

This document serves as a structured guide for the development of a quantitative trading system, outlining key phases, class structure, and system design. Further refinements can be made as strategies evolve and real-world constraints become more apparent.