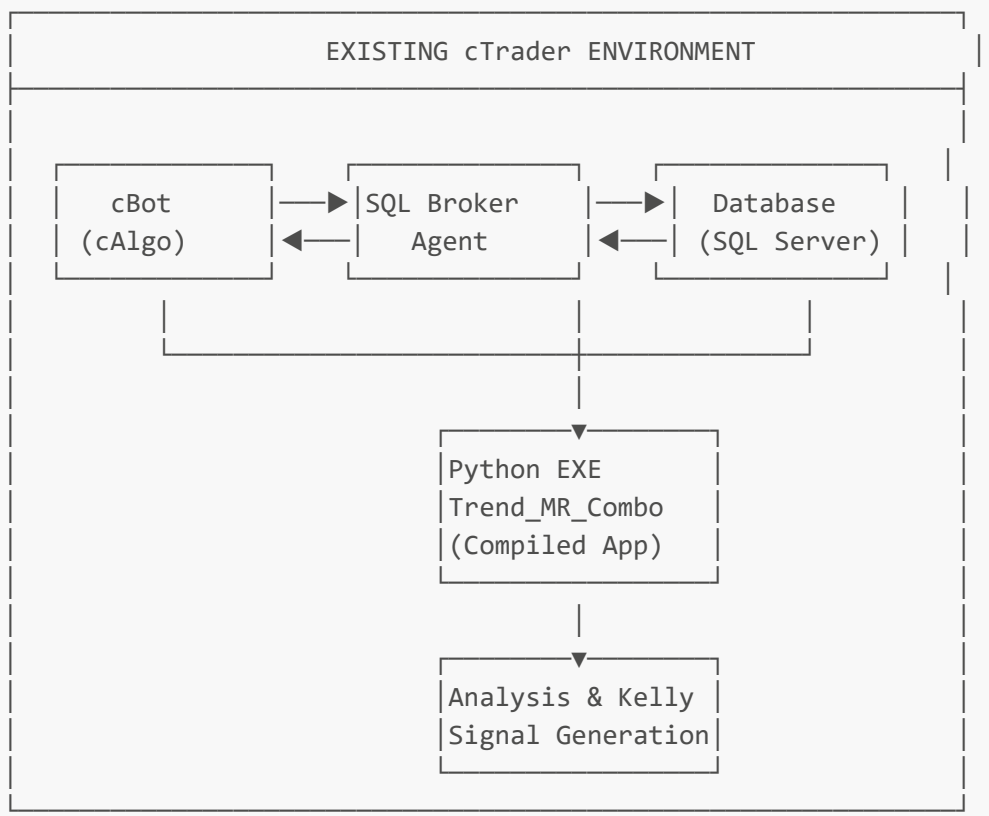
ARCHITECTURE - Trend_MR_Combo System

Обновленная архитектура системы:



Компоненты системы:

1. cBot (cAlgo) - УЖЕ СУЩЕСТВУЕТ

```
Функции:
- Запускается в cTrader
- OnStart(): логирует старт в БД
- OnTick(): мониторит сигналы каждые 500ms
- Исполняет ордера
- Управляет позициями
- Останавливается только при достижении drawdown
```

2. SQL Broker Agent - УЖЕ СУЩЕСТВУЕТ

```
Функции:
- Принимает команды от cBot
- Запускает Python приложение
```

```
- Управляет коммуникацией между cBot и Python
- Мониторит статус системы
```

3. Database (SQL Server) - УЖЕ СУЩЕСТВУЕТ

```
Существующие таблицы:
- trd.tradingSignals (signalID, assetID, volume, direction, orderPrice, stopLoss,
takeProfit, timeCreated, expiry, status, executionType, executionID,
executionTime)
- trd.assets (assetID, symbol, ...)
- trd.currentPositions
- Другие таблицы торговой системы

Нужно добавить:
- trd.strategyLogs (ID, logText, recorded)
- trd.regimeHistory (regimeID, regimeType, confidence, timestamp, indicators)
- trd.kellyCalculations (calculationID, symbol, kellyFraction, positionSize,
riskAmount, timestamp)
```

4. Python Application (Trend_MR_Combo) - НОВЫЙ КОМПОНЕНТ

```
Скомпилированное приложение (.exe) с модулями:

A. Market Regime Analyzer
   ├── Mean Reversion Detection (ADF, Hurst, RSI)
   ├── Trend Detection (MA Crossovers, Momentum)
   ├── Random Walk Detection
   └── Regime Confidence Scoring

B. Kelly Position Sizer
   ├── Kelly Criterion Calculator
   ├── Portfolio Optimization
   ├── Risk Management
   └── Position Size Calculator

C. Signal Generator
   ├── Signal Creation Based on Regime
   ├── Stop Loss/Take Profit Calculation
   ├── Signal Validation
   └── Database Integration

D. Data Manager
   ├── Market Data Feed
   ├── Historical Data Analysis
   ├── Real-time Price Updates
   └── Cache Management
```

Процесс работы системы:

Phase 1: Инициализация

```
1. cBot стартует → логирует в БД
2. SQL Broker Agent запускает Python приложение
3. Python приложение инициализирует:
   - Загружает конфигурацию
   - Подключается к БД
   - Загружает исторические данные
   - Кэширует начальные индикаторы
```

Phase 2: Непрерывный анализ

```
4. Python приложение каждые N секунд:
   ├── Получает рыночные данные
   ├── Анализирует режимы (Mean Reversion / Trend / Random Walk)
   ├── Рассчитывает Kelly fraction для каждого инструмента
   ├── Генерирует торговые сигналы
   └── Записывает сигналы в БД (tradingSignals)
```

Phase 3: Исполнение

```
5. cBot каждые 500ms:
   ├── Проверяет новые сигналы в БД
   ├── Исполняет ордера с Kelly-размерами
   ├── Управляет позициями (стоп-лосс, тейк-профит)
   └── Обновляет статусы в БД
```

Phase 4: Мониторинг и адаптация

```
6. Python приложение мониторит:
   ├── Эффективность стратегии
   ├── Изменения рыночных режимов
   ├── Kelly fractions пересчет
   └── Обновляет логи и статистику
```

Интерфейсы данных:

1. Сигналы (tradingSignals):

```
-- Структура как в вашем примере:
signalID | assetID | volume | direction | orderPrice | stopLoss | takeProfit |
timeCreated | expiry | status | executionType | executionID | executionTime
```

2. Расчеты Kelly (новая таблица):

```
CREATE TABLE trd.kellyCalculations (
    calculationID INT IDENTITY(1,1) PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    regimeType VARCHAR(20) NOT NULL, -- 'MEAN_REVERSION', 'TREND'
    winRate DECIMAL(5,4) NOT NULL,
    winLossRatio DECIMAL(5,2) NOT NULL,
    kellyFraction DECIMAL(5,4) NOT NULL, -- 0.0833 = 8.33%
    positionSize DECIMAL(18,6) NOT NULL,
    riskAmount DECIMAL(18,2) NOT NULL,
    confidence DECIMAL(5,4) NOT NULL,
    timestamp DATETIME DEFAULT GETDATE(),
    INDEX idx_symbol_time (symbol, timestamp)
);
```

3. История режимов (новая таблица):

```
CREATE TABLE trd.regimeHistory (
    regimeID INT IDENTITY(1,1) PRIMARY KEY,
    symbol VARCHAR(20) NOT NULL,
    regimeType VARCHAR(20) NOT NULL,
    confidence DECIMAL(5,4) NOT NULL,
    adfPValue DECIMAL(10,6) NULL,
    hurstExponent DECIMAL(10,6) NULL,
    rsiValue DECIMAL(10,2) NULL,
    maDiffPercent DECIMAL(10,6) NULL,
    timestamp DATETIME DEFAULT GETDATE(),
    INDEX idx_symbol_regime (symbol, regimeType, timestamp)
);
```

4. Логи стратегии (новая таблица):

```
CREATE TABLE trd.strategyLogs (
    ID INT IDENTITY(1,1) PRIMARY KEY,
    logText VARCHAR(255) NOT NULL,
    logLevel VARCHAR(20) DEFAULT 'INFO', -- INFO, WARNING, ERROR
    module VARCHAR(50) NULL, -- 'RegimeDetector', 'KellySizer', 'SignalGenerator'
    recorded DATETIME DEFAULT GETDATE(),
    INDEX idx_time_level (recorded, logLevel)
);
```

Алгоритмы расчета:

A. Определение режима:

```
def determine_regime(symbol_data):
    # 1. Mean Reversion признаки:
    #    - ADF p-value < 0.05
    #    - Hurst exponent < 0.45
    #    - RSI в диапазоне 30-70

    # 2. Trend признаки:
    #    - MA Fast > MA Slow на >0.5%
    #    - Hurst exponent > 0.55
    #    - RSI экстремальный (<30 или >70)

    # 3. Random Walk:
    #    - Hurst ≈ 0.5
    #    - ADF p-value > 0.1
    #    - Слабые трендовые сигналы
```

B. Kelly расчет:

```
def calculate_kelly(regime_type, win_rate, win_loss_ratio, confidence):
    # Kelly формула: f* = p/a - q/b
    # где p = win_rate, q = 1-p
    # b = win_loss_ratio, a = 1

    kelly_raw = (win_rate / 1) - ((1 - win_rate) / win_loss_ratio)

    # Ограничения:
    # - Максимум 8.33% для Mean Reversion (полный Kelly)
    # - Максимум 4.17% для Trend (половина Kelly)
    # - Учет confidence режима

    return adjusted_kelly
```

C. Размер позиции:

```
def calculate_position_size(symbol, kelly_fraction, current_price, stop_loss):
    # Риск на сделку = Капитал × Kelly fraction
    risk_amount = account_capital * kelly_fraction

    # Риск на единицу = |Цена - СтопЛосс|
    risk_per_unit = abs(current_price - stop_loss)

    # Количество единиц = Риск на сделку / Риск на единицу
    position_units = risk_amount / risk_per_unit

    # Конвертация в объем (лоты)
    volume = convert_to_volume(symbol, position_units)

    return volume
```

Конфигурация системы:

Файл config.json:

```
{
    "database": {
        "connection_string": "Driver={SQL
Server};Server=localhost\\SQLEXPRESS;Database=cTrader;Trusted_Connection=yes;",
        "signal_table": "trd.tradingSignals",
        "log_table": "trd.strategyLogs"
    },
    "trading": {
        "symbols": ["XAUUSD", "XAGUSD", "XPTUSD"],
        "initial_capital": 50000,
        "max_leverage": 30,
        "max_drawdown_percent": 20,
        "update_interval_seconds": 60
    },
    "regime_detection": {
        "mean_reversion_thresholds": {
            "adf_pvalue": 0.05,
            "hurst_exponent": 0.45,
            "rsi_lower": 30,
            "rsi_upper": 70
        },
        "trend_thresholds": {
            "ma_crossover_percent": 0.005,
            "hurst_exponent": 0.55,
            "min_confidence": 0.6
        }
    },
    "kelly": {
        "mean_reversion_params": {
            "win_rate": 0.55,
            "win_loss_ratio": 1.5,
            "max_kelly_fraction": 0.0833
        },
        "trend_params": {
            "win_rate": 0.45,
            "win_loss_ratio": 2.0,
            "max_kelly_fraction": 0.0417
        }
    }
}
```

Последовательность разработки:

Этап 1: Интеграция с существующей системой

1. Создать таблицы БД (`strategyLogs`, `regimeHistory`, `kellyCalculations`)
2. Настроить SQL Broker Agent для запуска Python приложения
3. Реализовать базовую коммуникацию cBot ↔ БД ↔ Python

Этап 2: Реализация анализа режимов

1. Market Regime Detector (ADF, Hurst, MA, RSI)
2. Режимный классификатор с confidence scoring
3. Логирование режимов в БД

Этап 3: Kelly расчет и sizing

1. Kelly Criterion calculator с учетом режима
2. Position size calculator
3. Risk management модуль

Этап 4: Signal generation

1. Генератор сигналов на основе режима и Kelly
2. Stop loss / take profit calculator
3. Валидация и запись сигналов в БД

Этап 5: Оптимизация и мониторинг

1. Backtesting модуль
2. Performance analyzer
3. Real-time dashboard для мониторинга

Требования к компиляции Python:

PyInstaller конфигурация:

```
# build_exe.py
import PyInstaller.__main__
import os

PyInstaller.__main__.run([
    'trend_mr_combo.py',
    '--onefile',
    '--name=TrendMRCombo',
    '--add-data=config.json:.',
    '--hidden-import=pandas',
    '--hidden-import=numpy',
    '--hidden-import=statsmodels',
    '--hidden-import=scipy',
    '--hidden-import=pyodbc',
    '--clean',
    '--noconsole'  # Для скрытия консоли в production
])
```

Структура приложения после компиляции:

```
TrendMRCombo.exe
├── Внутренние зависимости
├── config.json (внешний конфиг)
└── Логи в %APPDATA%\TrendMRCombo\logs\
```

Мониторинг и диагностика:

Dashboard показатели:

1. Текущий режим для каждого символа
2. Kelly fractions и размеры позиций
3. Количество активных сигналов
4. Общий P&L системы
5. Confidence scores режимов

Аварийные процедуры:

1. Автоматический stop при достижении max drawdown
2. Fallback на Random Walk при низком confidence
3. Логирование всех ошибок в БД
4. SMS/Email алерты при критических ошибках

Следующие шаги разработки:

1. **Создать недостающие таблицы в БД**
2. **Реализовать базовый Market Regime Detector**
3. **Интегрировать Kelly Calculator**
4. **Настроить SQL Broker Agent для запуска Python**
5. **Протестировать коммуникацию cBot ↔ БД**
6. **Добавить логирование и мониторинг**
7. **Оптимизация и backtesting**

Эта архитектура сохраняет существующую инфраструктуру и добавляет только необходимые компоненты для Trend_MR_Combo стратегии.