Мне нужно, структурировать наш проект, в проекте все разбросано не понятно как, все нужно проверить и правильно расположить!

у нас

systemetech.ru

www-root

/var/www/www-root/data/www/systemetech.ru

79.133.183.86

crypto\_top\_bd\_mysql

PHP FastCGI (Nginx + PHP-FPM) 8.1.2 (native)

/etc/crypto/config/.env

# API настройки

BYBIT\_API\_KEY=P0PKvshbBlaJjc7p0t

BYBIT\_API\_SECRET=BS0zOOEgzhLcEqr9E6h2EuWFD91FOus7nkTw

BYBIT\_TESTNET=true # ОБЯЗАТЕЛЬНО начинаем с testnet!

# База данных

DB\_HOST=localhost

DB\_NAME=crypto\_top\_bd\_mysql

DB\_USER=crypto\_top\_admin

DB\_PASSWORD=LSM6\_PqnuZ10nvEdzfe6

# Redis (установим позже)

REDIS\_HOST=localhost

REDIS\_PORT=6379

# Торговые параметры

TRADING\_SYMBOL=BTCUSDT

INITIAL\_CAPITAL=1000 # USD для testnet

MAX\_POSITION\_SIZE\_PERCENT=5 # 5% от капитала на сделку

STOP\_LOSS\_PERCENT=2

TAKE\_PROFIT\_PERCENT=4

# Human behavior

ENABLE\_HUMAN\_MODE=true

MIN\_DELAY\_SECONDS=0.5

MAX\_DELAY\_SECONDS=3.0

# Веб-интерфейс

WEB\_HOST=0.0.0.0

WEB\_PORT=8000

# Telegram уведомления

TELEGRAM\_BOT\_TOKEN=7821491963:AAHHT3nGM4qhn8FQu1goYXZN6TQ9h7l1WQ8

TELEGRAM\_CHAT\_ID=-4889364288

# Безопасность

SECRET\_KEY=your-very-long-random-secret-key-change-this-in-production

# Торговые пары (через запятую)

TRADING\_PAIRS=BTCUSDT,ETHUSDT,BNBUSDT,SOLUSDT

# Максимум открытых позиций

MAX\_POSITIONS=1

# Стратегии

ENABLE\_MULTI\_INDICATOR=true

ENABLE\_SCALPING=true

# Риск менеджмент

MIN\_RISK\_REWARD\_RATIO=2.0

MAX\_DAILY\_TRADES=10

НУжна полная структурированная часть кода.

напиши сначала структуру где и какие файлы должны быть!

потом напиши что не так и что нужно исправить где есть несоответствия, нужен единый код, без дублирований, все должно быть едино!

я знаю что в структуре есть повторяющийся функционал, так вот, нужен идеальный код! чтобы все работало как единый механизм!

Цель проекта, анализировать валюты, использовать лучшие стратегии на открытие сделок, выставлять стоп лосы и тейк профиты, и чтобы все это выглядело как будто это делает человек, чтобы бота не заблокировали!  
  
  
твой первый ответ   
  
Давайте структурируем ваш проект правильно. Сначала покажу идеальную структуру, затем проанализирую проблемы и предложу решения.

**📁 ИДЕАЛЬНАЯ СТРУКТУРА ПРОЕКТА**

/var/www/www-root/data/www/systemetech.ru/

│

├── 📁 src/ # Основной код приложения

│ ├── 📁 core/ # Ядро системы

│ │ ├── \_\_init\_\_.py

│ │ ├── config.py # Единая конфигурация

│ │ ├── database.py # Подключение к БД

│ │ ├── models.py # SQLAlchemy модели

│ │ └── exceptions.py # Кастомные исключения

│ │

│ ├── 📁 bot/ # Торговый бот

│ │ ├── \_\_init\_\_.py

│ │ ├── manager.py # ЕДИНЫЙ менеджер бота

│ │ ├── trader.py # Торговая логика

│ │ └── risk\_manager.py # Управление рисками

│ │

│ ├── 📁 exchange/ # Работа с биржей

│ │ ├── \_\_init\_\_.py

│ │ ├── client.py # ЕДИНЫЙ клиент Bybit

│ │ └── humanizer.py # Имитация человека

│ │

│ ├── 📁 strategies/ # Торговые стратегии

│ │ ├── \_\_init\_\_.py

│ │ ├── base.py # Базовый класс стратегии

│ │ ├── momentum.py # Momentum стратегия

│ │ ├── multi\_indicator.py # Мульти-индикаторная

│ │ └── scalping.py # Скальпинг

│ │

│ ├── 📁 analysis/ # Анализ рынка

│ │ ├── \_\_init\_\_.py

│ │ ├── market\_analyzer.py # Анализ рынка

│ │ ├── indicators.py # Технические индикаторы

│ │ └── signals.py # Генерация сигналов

│ │

│ ├── 📁 notifications/ # Уведомления

│ │ ├── \_\_init\_\_.py

│ │ └── telegram.py # Telegram уведомления

│ │

│ └── 📁 web/ # Веб-интерфейс

│ ├── \_\_init\_\_.py

│ ├── app.py # FastAPI приложение

│ ├── auth.py # Аутентификация

│ ├── api.py # API endpoints

│ └── websocket.py # WebSocket handlers

│

├── 📁 scripts/ # Скрипты управления

│ ├── install.sh # Установка

│ ├── start.sh # Запуск системы

│ ├── stop.sh # Остановка

│ └── monitor.sh # Мониторинг

│

├── 📁 tests/ # Тесты

│ ├── test\_strategies.py

│ ├── test\_exchange.py

│ └── test\_bot.py

│

├── 📁 logs/ # Логи

│ ├── trading.log

│ ├── web.log

│ └── errors.log

│

├── 📁 data/ # Данные

│ ├── 📁 cache/ # Кэш

│ └── 📁 backups/ # Бэкапы

│

├── 📁 docs/ # Документация

│ ├── README.md

│ ├── SETUP.md

│ └── API.md

│

├── 📁 static/ # Статика для веб

│ ├── 📁 css/

│ └── 📁 js/

│

├── 📁 templates/ # HTML шаблоны

│

├── main.py # ЕДИНАЯ точка входа

├── requirements.txt # Зависимости

├── .env -> /etc/crypto/config/.env # Символическая ссылка

└── .gitignore

**🔍 АНАЛИЗ ТЕКУЩИХ ПРОБЛЕМ**

**1. Дублирование менеджеров бота**

❌ ПРОБЛЕМА:

- bot\_manager.py

- process\_bot\_manager.py

- unified\_bot\_manager.py

- state\_manager.py

✅ РЕШЕНИЕ: Один единый BotManager в src/bot/manager.py

**2. Множество точек входа**

❌ ПРОБЛЕМА:

- main.py

- main\_simple.py

- main\_advanced.py

✅ РЕШЕНИЕ: Один main.py с параметрами запуска

**3. Разбросанные файлы проверки**

❌ ПРОБЛЕМА:

- test\_installation.py

- test\_installation\_fixed.py

- test\_setup.py

- check\_install.py

- check\_database.py

- check\_system.sh

✅ РЕШЕНИЕ: Один scripts/check\_system.py

**4. Дублирование клиентов биржи**

❌ ПРОБЛЕМА:

- bybit\_client.py с классом HumanizedBybitClient

- Прямое использование ccxt в разных местах

✅ РЕШЕНИЕ: Один ExchangeClient с HumanBehavior миксином

**📝 ЕДИНЫЙ КОД БЕЗ ДУБЛИРОВАНИЯ## 🛠️ ЧТО НУЖНО ИСПРАВИТЬ**

**1. Объединение менеджеров## 🔍 КОНКРЕТНЫЕ ПРОБЛЕМЫ И РЕШЕНИЯ**

**Проблема 1: Множественные менеджеры бота**

❌ ТЕКУЩЕЕ СОСТОЯНИЕ:

- bot\_manager.py - основной менеджер с дублированием start/stop

- process\_bot\_manager.py - управление процессами

- unified\_bot\_manager.py - попытка объединения

- state\_manager.py - управление состоянием

✅ РЕШЕНИЕ: Один BotManager который:

- Управляет жизненным циклом бота

- Следит за состоянием

- Управляет процессами

- Синхронизирует с БД

```### Проблема 2: Дублирование клиентов биржи## 🚀 ФИНАЛЬНЫЕ ШАГИ РЕФАКТОРИНГА

мы внесли правки и запросили у тебя снова помощь по доработке  
  
ниже наш запрос снова  
«Я обратился к тебе с таким запросом. "Мне нужно, структурировать наш проект, в проекте все разбросано не понятно как, все нужно проверить и правильно расположить! у нас systemetech.ru www-root /var/www/www-root/data/www/systemetech.ru 79.133.183.86 crypto\_top\_bd\_mysql PHP FastCGI (Nginx + PHP-FPM) 8.1.2 (native) /etc/crypto/config/.env # API настройки BYBIT\_API\_KEY=P0PKvshbBlaJjc7p0t BYBIT\_API\_SECRET=BS0zOOEgzhLcEqr9E6h2EuWFD91FOus7nkTw BYBIT\_TESTNET=true # ОБЯЗАТЕЛЬНО начинаем с testnet! # База данных DB\_HOST=localhost DB\_NAME=crypto\_top\_bd\_mysql DB\_USER=crypto\_top\_admin DB\_PASSWORD=LSM6\_PqnuZ10nvEdzfe6 # Redis (установим позже) REDIS\_HOST=localhost REDIS\_PORT=6379 # Торговые параметры TRADING\_SYMBOL=BTCUSDT INITIAL\_CAPITAL=1000 # USD для testnet MAX\_POSITION\_SIZE\_PERCENT=5 # 5% от капитала на сделку STOP\_LOSS\_PERCENT=2 TAKE\_PROFIT\_PERCENT=4 # Human behavior ENABLE\_HUMAN\_MODE=true MIN\_DELAY\_SECONDS=0.5 MAX\_DELAY\_SECONDS=3.0 # Веб-интерфейс WEB\_HOST=0.0.0.0 WEB\_PORT=8000 # Telegram уведомления TELEGRAM\_BOT\_TOKEN=7821491963:AAHHT3nGM4qhn8FQu1goYXZN6TQ9h7l1WQ8 TELEGRAM\_CHAT\_ID=-4889364288 # Безопасность SECRET\_KEY=your-very-long-random-secret-key-change-this-in-production # Торговые пары (через запятую) TRADING\_PAIRS=BTCUSDT,ETHUSDT,BNBUSDT,SOLUSDT # Максимум открытых позиций MAX\_POSITIONS=1 # Стратегии ENABLE\_MULTI\_INDICATOR=true ENABLE\_SCALPING=true # Риск менеджмент MIN\_RISK\_REWARD\_RATIO=2.0 MAX\_DAILY\_TRADES=10 НУжна полная структурированная часть кода. напиши сначала структуру где и какие файлы должны быть! потом напиши что не так и что нужно исправить где есть несоответствия, нужен единый код, без дублирований, все должно быть едино! я знаю что в структуре есть повторяющийся функционал, так вот, нужен идеальный код! чтобы все работало как единый механизм! Цель проекта, анализировать валюты, использовать лучшие стратегии на открытие сделок, выставлять стоп лосы и тейк профиты, и чтобы все это выглядело как будто это делает человек, чтобы бота не заблокировали!" ты вроде бы как попытался сделать лучше, но лучше не получилось, вот твой ответ : Проверь что уже сделано, что сделано не правильно, и в каждом коде, пиши что сделать, в какую папку и как назвать файл. Допиши код целиком, чтобы все соответствовало нашему проекту! проект должен анализировать валюты которые мы выберем в интерфейсе. через интерфейс со входом через app должен осуществятся выход, и там должны быть абсолютно все данные по включению бота, анализам, всем сделкам и т.д !

в первый раз ты написал это Давайте структурируем ваш проект правильно. Сначала покажу идеальную структуру, затем проанализирую проблемы и предложу решения.

«  
  
ты ответил нам   
  
вот таким сообщением   
  
Продолжаю с того места, где остановился. Дописываю файл **src/web/api.py**:

**1️⃣1️⃣ src/web/api.py (продолжение)**

# ... продолжение кода ...

# Статистика по сделкам

trades = db.query(Trade).filter(Trade.created\_at >= start\_date).all()

total\_trades = len(trades)

profitable\_trades = len([t for t in trades if t.profit and t.profit > 0])

total\_profit = sum(t.profit or 0 for t in trades)

# Win rate

win\_rate = (profitable\_trades / total\_trades \* 100) if total\_trades > 0 else 0

# Средняя прибыль/убыток

avg\_profit = total\_profit / total\_trades if total\_trades > 0 else 0

# Лучшая и худшая сделка

best\_trade = max(trades, key=lambda t: t.profit or 0) if trades else None

worst\_trade = min(trades, key=lambda t: t.profit or 0) if trades else None

# Статистика по парам

pair\_stats = {}

for trade in trades:

if trade.symbol not in pair\_stats:

pair\_stats[trade.symbol] = {

'total': 0,

'profitable': 0,

'profit': 0

}

pair\_stats[trade.symbol]['total'] += 1

if trade.profit and trade.profit > 0:

pair\_stats[trade.symbol]['profitable'] += 1

pair\_stats[trade.symbol]['profit'] += trade.profit or 0

return {

'period': period,

'total\_trades': total\_trades,

'profitable\_trades': profitable\_trades,

'win\_rate': win\_rate,

'total\_profit': total\_profit,

'average\_profit': avg\_profit,

'best\_trade': {

'symbol': best\_trade.symbol,

'profit': best\_trade.profit,

'date': best\_trade.created\_at

} if best\_trade else None,

'worst\_trade': {

'symbol': worst\_trade.symbol,

'profit': worst\_trade.profit,

'date': worst\_trade.created\_at

} if worst\_trade else None,

'pair\_statistics': pair\_stats

}

@router.post("/position/{symbol}/close")

async def close\_position(

symbol: str,

current\_user: User = Depends(get\_current\_user)

):

"""Закрыть позицию вручную"""

if not current\_user.is\_admin:

raise HTTPException(status\_code=403, detail="Только администраторы могут закрывать позиции")

success, message = await bot\_manager.close\_position(symbol)

if not success:

raise HTTPException(status\_code=404, detail=message)

return {"status": "closed", "message": message}

@router.get("/dashboard")

async def get\_dashboard\_data(

db: Session = Depends(get\_db),

current\_user: User = Depends(get\_current\_user)

):

"""Получить все данные для дашборда"""

bot\_status = bot\_manager.get\_status()

balance = await get\_balance(db, current\_user)

statistics = await get\_statistics("day", db, current\_user)

recent\_trades = await get\_trades(0, 10, db, current\_user)

recent\_signals = await get\_signals(0, 10, db, current\_user)

return {

'bot\_status': bot\_status,

'balance': balance,

'statistics': statistics,

'recent\_trades': recent\_trades,

'recent\_signals': recent\_signals,

'timestamp': datetime.utcnow()

}

**1️⃣2️⃣ src/web/auth.py - Аутентификация**

"""

Модуль аутентификации

Путь: /var/www/www-root/data/www/systemetech.ru/src/web/auth.py

"""

from fastapi import APIRouter, Depends, HTTPException, status

from fastapi.security import OAuth2PasswordBearer, OAuth2PasswordRequestForm

from passlib.context import CryptContext

from datetime import datetime, timedelta

from typing import Optional

import jwt

from sqlalchemy.orm import Session

from ..core.config import config

from ..core.database import get\_db

from ..core.models import User

router = APIRouter()

# Настройки безопасности

pwd\_context = CryptContext(schemes=["bcrypt"], deprecated="auto")

oauth2\_scheme = OAuth2PasswordBearer(tokenUrl="/auth/token")

ALGORITHM = "HS256"

ACCESS\_TOKEN\_EXPIRE\_MINUTES = 30

def verify\_password(plain\_password: str, hashed\_password: str) -> bool:

"""Проверка пароля"""

return pwd\_context.verify(plain\_password, hashed\_password)

def get\_password\_hash(password: str) -> str:

"""Хеширование пароля"""

return pwd\_context.hash(password)

def create\_access\_token(data: dict, expires\_delta: Optional[timedelta] = None):

"""Создание JWT токена"""

to\_encode = data.copy()

if expires\_delta:

expire = datetime.utcnow() + expires\_delta

else:

expire = datetime.utcnow() + timedelta(minutes=15)

to\_encode.update({"exp": expire})

encoded\_jwt = jwt.encode(to\_encode, config.SECRET\_KEY, algorithm=ALGORITHM)

return encoded\_jwt

async def get\_current\_user(

token: str = Depends(oauth2\_scheme),

db: Session = Depends(get\_db)

) -> User:

"""Получение текущего пользователя из токена"""

credentials\_exception = HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Could not validate credentials",

headers={"WWW-Authenticate": "Bearer"},

)

try:

payload = jwt.decode(token, config.SECRET\_KEY, algorithms=[ALGORITHM])

username: str = payload.get("sub")

if username is None:

raise credentials\_exception

except jwt.PyJWTError:

raise credentials\_exception

user = db.query(User).filter(User.username == username).first()

if user is None:

raise credentials\_exception

if user.is\_blocked:

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="User is blocked"

)

return user

@router.post("/token")

async def login(

form\_data: OAuth2PasswordRequestForm = Depends(),

db: Session = Depends(get\_db)

):

"""Вход в систему"""

user = db.query(User).filter(User.username == form\_data.username).first()

if not user:

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Incorrect username or password",

headers={"WWW-Authenticate": "Bearer"},

)

# Проверяем блокировку

if user.is\_blocked:

raise HTTPException(

status\_code=status.HTTP\_403\_FORBIDDEN,

detail="User is blocked"

)

# Проверяем пароль

if not verify\_password(form\_data.password, user.hashed\_password):

# Увеличиваем счетчик неудачных попыток

user.failed\_login\_attempts += 1

# Блокируем после 5 попыток

if user.failed\_login\_attempts >= 5:

user.is\_blocked = True

user.blocked\_at = datetime.utcnow()

db.commit()

raise HTTPException(

status\_code=status.HTTP\_401\_UNAUTHORIZED,

detail="Incorrect username or password",

headers={"WWW-Authenticate": "Bearer"},

)

# Сбрасываем счетчик при успешном входе

user.failed\_login\_attempts = 0

user.last\_login = datetime.utcnow()

db.commit()

# Создаем токен

access\_token\_expires = timedelta(minutes=ACCESS\_TOKEN\_EXPIRE\_MINUTES)

access\_token = create\_access\_token(

data={"sub": user.username}, expires\_delta=access\_token\_expires

)

return {"access\_token": access\_token, "token\_type": "bearer"}

@router.post("/register")

async def register(

username: str,

password: str,

email: Optional[str] = None,

db: Session = Depends(get\_db)

):

"""Регистрация нового пользователя"""

# Проверяем существование пользователя

if db.query(User).filter(User.username == username).first():

raise HTTPException(

status\_code=400,

detail="Username already registered"

)

# Создаем пользователя

hashed\_password = get\_password\_hash(password)

user = User(

username=username,

hashed\_password=hashed\_password,

email=email

)

db.add(user)

db.commit()

db.refresh(user)

return {"message": "User created successfully", "username": username}

@router.get("/me")

async def get\_current\_user\_info(current\_user: User = Depends(get\_current\_user)):

"""Получить информацию о текущем пользователе"""

return {

"username": current\_user.username,

"email": current\_user.email,

"is\_admin": current\_user.is\_admin,

"created\_at": current\_user.created\_at,

"last\_login": current\_user.last\_login

}

**1️⃣3️⃣ src/web/websocket.py - WebSocket менеджер**

"""

WebSocket менеджер для real-time обновлений

Путь: /var/www/www-root/data/www/systemetech.ru/src/web/websocket.py

"""

from typing import Dict, List

from fastapi import WebSocket

import json

import asyncio

import logging

logger = logging.getLogger(\_\_name\_\_)

class WebSocketManager:

"""Менеджер WebSocket соединений"""

def \_\_init\_\_(self):

self.active\_connections: Dict[str, WebSocket] = {}

self.\_broadcast\_task = None

async def connect(self, websocket: WebSocket, client\_id: str):

"""Подключение нового клиента"""

await websocket.accept()

self.active\_connections[client\_id] = websocket

logger.info(f"WebSocket клиент {client\_id} подключен")

# Отправляем приветственное сообщение

await self.send\_personal\_message(

{"type": "connection", "message": "Connected to Crypto Bot"},

client\_id

)

def disconnect(self, client\_id: str):

"""Отключение клиента"""

if client\_id in self.active\_connections:

del self.active\_connections[client\_id]

logger.info(f"WebSocket клиент {client\_id} отключен")

async def send\_personal\_message(self, message: dict, client\_id: str):

"""Отправка сообщения конкретному клиенту"""

if client\_id in self.active\_connections:

try:

await self.active\_connections[client\_id].send\_json(message)

except Exception as e:

logger.error(f"Ошибка отправки сообщения клиенту {client\_id}: {e}")

self.disconnect(client\_id)

async def broadcast(self, message: dict):

"""Отправка сообщения всем подключенным клиентам"""

disconnected\_clients = []

for client\_id, connection in self.active\_connections.items():

try:

await connection.send\_json(message)

except Exception as e:

logger.error(f"Ошибка отправки broadcast клиенту {client\_id}: {e}")

disconnected\_clients.append(client\_id)

# Удаляем отключенных клиентов

for client\_id in disconnected\_clients:

self.disconnect(client\_id)

async def start\_broadcast\_loop(self):

"""Запуск цикла рассылки обновлений"""

from ..bot.manager import bot\_manager

from ..core.database import SessionLocal

from ..core.models import Trade, Signal, Balance

while True:

try:

# Получаем статус бота

bot\_status = bot\_manager.get\_status()

# Получаем последние данные из БД

db = SessionLocal()

try:

# Последние сделки

recent\_trades = db.query(Trade).order\_by(

Trade.created\_at.desc()

).limit(5).all()

# Последние сигналы

recent\_signals = db.query(Signal).order\_by(

Signal.created\_at.desc()

).limit(5).all()

# Текущий баланс

latest\_balance = db.query(Balance).filter(

Balance.currency == 'USDT'

).order\_by(Balance.timestamp.desc()).first()

finally:

db.close()

# Формируем сообщение

update\_message = {

"type": "update",

"data": {

"bot\_status": bot\_status,

"recent\_trades": [

{

"symbol": t.symbol,

"side": t.side.value,

"status": t.status.value,

"profit": t.profit,

"created\_at": t.created\_at.isoformat()

}

for t in recent\_trades

],

"recent\_signals": [

{

"symbol": s.symbol,

"action": s.action,

"confidence": s.confidence,

"created\_at": s.created\_at.isoformat()

}

for s in recent\_signals

],

"balance": {

"total": latest\_balance.total if latest\_balance else 0,

"free": latest\_balance.free if latest\_balance else 0

}

}

}

# Отправляем всем клиентам

await self.broadcast(update\_message)

# Пауза между обновлениями

await asyncio.sleep(5)

except Exception as e:

logger.error(f"Ошибка в broadcast loop: {e}")

await asyncio.sleep(10)

# Глобальный экземпляр

ws\_manager = WebSocketManager()

**1️⃣4️⃣ src/web/dashboard.py - HTML дашборд**

"""

HTML дашборд

Путь: /var/www/www-root/data/www/systemetech.ru/src/web/dashboard.py

"""

def get\_dashboard\_html() -> str:

"""Возвращает HTML код дашборда"""

return """

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Crypto Trading Bot Dashboard</title>

<style>

\* {

margin: 0;

padding: 0;

box-sizing: border-box;

}

body {

font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, sans-serif;

background: #0a0e27;

color: #fff;

line-height: 1.6;

}

.container {

max-width: 1400px;

margin: 0 auto;

padding: 20px;

}

header {

background: rgba(255,255,255,0.05);

padding: 20px 0;

margin-bottom: 30px;

border-radius: 10px;

}

.header-content {

display: flex;

justify-content: space-between;

align-items: center;

padding: 0 30px;

}

h1 {

font-size: 28px;

background: linear-gradient(45deg, #667eea, #764ba2);

-webkit-background-clip: text;

-webkit-text-fill-color: transparent;

}

.user-info {

display: flex;

align-items: center;

gap: 20px;

}

.btn {

background: linear-gradient(45deg, #667eea, #764ba2);

color: white;

border: none;

padding: 10px 20px;

border-radius: 5px;

cursor: pointer;

font-size: 14px;

transition: transform 0.2s;

}

.btn:hover {

transform: translateY(-2px);

}

.btn-danger {

background: linear-gradient(45deg, #f093fb, #f5576c);

}

.grid {

display: grid;

grid-template-columns: repeat(auto-fit, minmax(300px, 1fr));

gap: 20px;

margin-bottom: 30px;

}

.card {

background: rgba(255,255,255,0.05);

border-radius: 10px;

padding: 25px;

border: 1px solid rgba(255,255,255,0.1);

transition: transform 0.2s;

}

.card:hover {

transform: translateY(-5px);

border-color: rgba(102, 126, 234, 0.5);

}

.card h3 {

font-size: 14px;

color: #888;

margin-bottom: 10px;

text-transform: uppercase;

letter-spacing: 1px;

}

.metric {

font-size: 32px;

font-weight: 700;

margin-bottom: 5px;

}

.positive {

color: #4ade80;

}

.negative {

color: #f87171;

}

.status-badge {

display: inline-block;

padding: 5px 15px;

border-radius: 20px;

font-size: 12px;

font-weight: 600;

text-transform: uppercase;

}

.status-running {

background: rgba(74, 222, 128, 0.2);

color: #4ade80;

}

.status-stopped {

background: rgba(248, 113, 113, 0.2);

color: #f87171;

}

.controls {

display: flex;

gap: 10px;

margin-top: 20px;

}

.table-container {

background: rgba(255,255,255,0.05);

border-radius: 10px;

padding: 20px;

overflow-x: auto;

}

table {

width: 100%;

border-collapse: collapse;

}

th, td {

padding: 12px;

text-align: left;

border-bottom: 1px solid rgba(255,255,255,0.1);

}

th {

font-weight: 600;

color: #888;

text-transform: uppercase;

font-size: 12px;

letter-spacing: 1px;

}

.pairs-selector {

display: grid;

grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));

gap: 10px;

margin: 20px 0;

}

.pair-checkbox {

display: flex;

align-items: center;

padding: 10px;

background: rgba(255,255,255,0.05);

border-radius: 5px;

cursor: pointer;

}

.pair-checkbox:hover {

background: rgba(255,255,255,0.1);

}

.pair-checkbox input {

margin-right: 8px;

}

#connection-status {

width: 10px;

height: 10px;

border-radius: 50%;

background: #f87171;

display: inline-block;

margin-right: 5px;

}

#connection-status.connected {

background: #4ade80;

animation: pulse 2s infinite;

}

@keyframes pulse {

0% {

box-shadow: 0 0 0 0 rgba(74, 222, 128, 0.7);

}

70% {

box-shadow: 0 0 0 10px rgba(74, 222, 128, 0);

}

100% {

box-shadow: 0 0 0 0 rgba(74, 222, 128, 0);

}

}

.modal {

display: none;

position: fixed;

z-index: 1000;

left: 0;

top: 0;

width: 100%;

height: 100%;

background: rgba(0,0,0,0.8);

}

.modal-content {

background: #1a1f36;

margin: 5% auto;

padding: 30px;

width: 90%;

max-width: 500px;

border-radius: 10px;

position: relative;

}

.close {

position: absolute;

right: 20px;

top: 20px;

font-size: 28px;

cursor: pointer;

color: #888;

}

.close:hover {

color: #fff;

}

.login-form {

display: flex;

flex-direction: column;

gap: 15px;

}

.login-form input {

background: rgba(255,255,255,0.05);

border: 1px solid rgba(255,255,255,0.1);

padding: 12px;

border-radius: 5px;

color: #fff;

font-size: 16px;

}

.login-form input:focus {

outline: none;

border-color: #667eea;

}

.chart-container {

height: 400px;

margin-top: 20px;

}

</style>

</head>

<body>

<div class="container">

<header>

<div class="header-content">

<h1>🤖 Crypto Trading Bot</h1>

<div class="user-info">

<span id="connection-status"></span>

<span id="username">Not logged in</span>

<button class="btn" onclick="logout()">Logout</button>

</div>

</div>

</header>

<div class="grid">

<div class="card">

<h3>Bot Status</h3>

<div id="bot-status" class="status-badge status-stopped">Stopped</div>

<div class="controls">

<button id="start-btn" class="btn" onclick="startBot()">Start Bot</button>

<button id="stop-btn" class="btn btn-danger" onclick="stopBot()" style="display:none;">Stop Bot</button>

</div>

</div>

<div class="card">

<h3>Balance (USDT)</h3>

<div id="balance" class="metric">$0.00</div>

<small id="balance-change">+0.00%</small>

</div>

<div class="card">

<h3>Daily P&L</h3>

<div id="daily-pnl" class="metric">$0.00</div>

<small id="pnl-percent">+0.00%</small>

</div>

<div class="card">

<h3>Open Positions</h3>

<div id="open-positions" class="metric">0</div>

<small>Active trades</small>

</div>

<div class="card">

<h3>Win Rate</h3>

<div id="win-rate" class="metric">0%</div>

<small id="total-trades">0 trades today</small>

</div>

<div class="card">

<h3>Active Pairs</h3>

<div id="active-pairs" class="metric">0</div>

<small>Trading pairs</small>

</div>

</div>

<div class="card">

<h3>Trading Pairs</h3>

<div class="pairs-selector" id="pairs-selector"></div>

<button class="btn" onclick="updatePairs()">Update Pairs</button>

</div>

<div class="table-container">

<h3>Recent Trades</h3>

<table id="trades-table">

<thead>

<tr>

<th>Time</th>

<th>Pair</th>

<th>Side</th>

<th>Entry</th>

<th>Exit</th>

<th>Profit</th>

<th>Status</th>

</tr>

</thead>

<tbody id="trades-tbody">

</tbody>

</table>

</div>

<div class="table-container" style="margin-top: 20px;">

<h3>Recent Signals</h3>

<table id="signals-table">

<thead>

<tr>

<th>Time</th>

<th>Pair</th>

<th>Action</th>

<th>Confidence</th>

<th>Reason</th>

</tr>

</thead>

<tbody id="signals-tbody">

</tbody>

</table>

</div>

</div>

<!-- Login Modal -->

<div id="loginModal" class="modal">

<div class="modal-content">

<span class="close" onclick="closeLoginModal()">&times;</span>

<h2>Login</h2>

<form class="login-form" onsubmit="login(event)">

<input type="text" id="login-username" placeholder="Username" required>

<input type="password" id="login-password" placeholder="Password" required>

<button type="submit" class="btn">Login</button>

</form>

</div>

</div>

<script>

let ws = null;

let token = localStorage.getItem('token');

const API\_URL = window.location.origin;

// Проверка авторизации при загрузке

window.onload = async function() {

if (!token) {

showLoginModal();

} else {

await checkAuth();

connectWebSocket();

loadDashboardData();

}

};

function showLoginModal() {

document.getElementById('loginModal').style.display = 'block';

}

function closeLoginModal() {

document.getElementById('loginModal').style.display = 'none';

}

async function login(event) {

event.preventDefault();

const username = document.getElementById('login-username').value;

const password = document.getElementById('login-password').value;

const formData = new FormData();

formData.append('username', username);

formData.append('password', password);

try {

const response = await fetch(`${API\_URL}/auth/token`, {

method: 'POST',

body: formData

});

if (response.ok) {

const data = await response.json();

token = data.access\_token;

localStorage.setItem('token', token);

closeLoginModal();

document.getElementById('username').textContent = username;

connectWebSocket();

loadDashboardData();

} else {

alert('Login failed');

}

} catch (error) {

alert('Connection error');

}

}

async function checkAuth() {

try {

const response = await fetch(`${API\_URL}/auth/me`, {

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

const user = await response.json();

document.getElementById('username').textContent = user.username;

} else {

showLoginModal();

}

} catch (error) {

showLoginModal();

}

}

function logout() {

localStorage.removeItem('token');

if (ws) ws.close();

window.location.reload();

}

function connectWebSocket() {

const wsUrl = `ws://${window.location.host}/ws/${generateClientId()}`;

ws = new WebSocket(wsUrl);

ws.onopen = () => {

document.getElementById('connection-status').classList.add('connected');

};

ws.onclose = () => {

document.getElementById('connection-status').classList.remove('connected');

// Переподключение через 5 секунд

setTimeout(connectWebSocket, 5000);

};

ws.onmessage = (event) => {

const data = JSON.parse(event.data);

if (data.type === 'update') {

updateDashboard(data.data);

}

};

}

function generateClientId() {

return Math.random().toString(36).substr(2, 9);

}

async function loadDashboardData() {

try {

const response = await fetch(`${API\_URL}/api/dashboard`, {

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

const data = await response.json();

updateFromApiData(data);

}

} catch (error) {

console.error('Error loading dashboard data:', error);

}

// Загружаем список пар

loadTradingPairs();

}

async function loadTradingPairs() {

try {

const response = await fetch(`${API\_URL}/api/pairs`, {

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

const pairs = await response.json();

displayTradingPairs(pairs);

}

} catch (error) {

console.error('Error loading pairs:', error);

}

}

function displayTradingPairs(pairs) {

const container = document.getElementById('pairs-selector');

container.innerHTML = '';

const allPairs = ['BTCUSDT', 'ETHUSDT', 'BNBUSDT', 'SOLUSDT', 'ADAUSDT', 'XRPUSDT', 'DOTUSDT', 'DOGEUSDT'];

allPairs.forEach(symbol => {

const pairDiv = document.createElement('div');

pairDiv.className = 'pair-checkbox';

const isActive = pairs.some(p => p.symbol === symbol && p.is\_active);

pairDiv.innerHTML = `

<input type="checkbox" id="pair-${symbol}" value="${symbol}" ${isActive ? 'checked' : ''}>

<label for="pair-${symbol}">${symbol}</label>

`;

container.appendChild(pairDiv);

});

}

function updateFromApiData(data) {

const { bot\_status, balance, statistics, recent\_trades, recent\_signals } = data;

// Обновляем статус бота

if (bot\_status) {

updateBotStatus(bot\_status.is\_running);

document.getElementById('active-pairs').textContent = bot\_status.active\_pairs.length;

document.getElementById('open-positions').textContent = bot\_status.open\_positions;

}

// Обновляем баланс

if (balance && balance.USDT) {

document.getElementById('balance').textContent = `$${balance.USDT.total.toFixed(2)}`;

}

// Обновляем статистику

if (statistics) {

document.getElementById('daily-pnl').textContent = `$${statistics.total\_profit.toFixed(2)}`;

document.getElementById('daily-pnl').className = statistics.total\_profit >= 0 ? 'metric positive' : 'metric negative';

const pnlPercent = statistics.total\_profit / (balance?.USDT?.total || 1000) \* 100;

document.getElementById('pnl-percent').textContent = `${pnlPercent >= 0 ? '+' : ''}${pnlPercent.toFixed(2)}%`;

document.getElementById('win-rate').textContent = `${statistics.win\_rate.toFixed(1)}%`;

document.getElementById('total-trades').textContent = `${statistics.total\_trades} trades today`;

}

// Обновляем таблицы

if (recent\_trades) updateTradesTable(recent\_trades);

if (recent\_signals) updateSignalsTable(recent\_signals);

}

function updateDashboard(data) {

// Обновление от WebSocket

if (data.bot\_status) {

updateBotStatus(data.bot\_status.is\_running);

}

if (data.balance) {

document.getElementById('balance').textContent = `$${data.balance.total.toFixed(2)}`;

}

if (data.recent\_trades) {

updateTradesTable(data.recent\_trades);

}

if (data.recent\_signals) {

updateSignalsTable(data.recent\_signals);

}

}

function updateBotStatus(isRunning) {

const statusEl = document.getElementById('bot-status');

const startBtn = document.getElementById('start-btn');

const stopBtn = document.getElementById('stop-btn');

if (isRunning) {

statusEl.textContent = 'Running';

statusEl.className = 'status-badge status-running';

startBtn.style.display = 'none';

stopBtn.style.display = 'block';

} else {

statusEl.textContent = 'Stopped';

statusEl.className = 'status-badge status-stopped';

startBtn.style.display = 'block';

stopBtn.style.display = 'none';

}

}

function updateTradesTable(trades) {

const tbody = document.getElementById('trades-tbody');

tbody.innerHTML = '';

trades.slice(0, 10).forEach(trade => {

const row = tbody.insertRow();

row.insertCell(0).textContent = new Date(trade.created\_at).toLocaleString();

row.insertCell(1).textContent = trade.symbol;

row.insertCell(2).textContent = trade.side;

row.insertCell(3).textContent = `$${trade.entry\_price.toFixed(2)}`;

row.insertCell(4).textContent = trade.exit\_price ? `$${trade.exit\_price.toFixed(2)}` : '-';

const profitCell = row.insertCell(5);

if (trade.profit !== null) {

profitCell.textContent = `$${trade.profit.toFixed(2)}`;

profitCell.className = trade.profit >= 0 ? 'positive' : 'negative';

} else {

profitCell.textContent = '-';

}

row.insertCell(6).textContent = trade.status;

});

}

function updateSignalsTable(signals) {

const tbody = document.getElementById('signals-tbody');

tbody.innerHTML = '';

signals.slice(0, 10).forEach(signal => {

const row = tbody.insertRow();

row.insertCell(0).textContent = new Date(signal.created\_at).toLocaleString();

row.insertCell(1).textContent = signal.symbol;

row.insertCell(2).textContent = signal.action;

row.insertCell(3).textContent = `${(signal.confidence \* 100).toFixed(1)}%`;

row.insertCell(4).textContent = signal.reason || '-';

});

}

async function startBot() {

try {

const response = await fetch(`${API\_URL}/api/bot/start`, {

method: 'POST',

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

alert('Bot started successfully');

loadDashboardData();

} else {

const error = await response.json();

alert(`Failed to start bot: ${error.detail}`);

}

} catch (error) {

alert('Connection error');

}

}

async function stopBot() {

try {

const response = await fetch(`${API\_URL}/api/bot/stop`, {

method: 'POST',

headers: {

'Authorization': `Bearer ${token}`

}

});

if (response.ok) {

alert('Bot stopped successfully');

loadDashboardData();

} else {

const error = await response.json();

alert(`Failed to stop bot: ${error.detail}`);

}

} catch (error) {

alert('Connection error');

}

}

async function updatePairs() {

const checkboxes = document.querySelectorAll('#pairs-selector input[type="checkbox"]:checked');

const pairs = Array.from(checkboxes).map(cb => cb.value);

if (pairs.length === 0) {

alert('Please select at least one trading pair');

return;

}

try {

const response = await fetch(`${API\_URL}/api/pairs`, {

method: 'POST',

headers: {

'Authorization': `Bearer ${token}`,

'Content-Type': 'application/json'

},

body: JSON.stringify(pairs)

});

if (response.ok) {

alert('Trading pairs updated successfully');

loadDashboardData();

} else {

const error = await response.json();

alert(`Failed to update pairs: ${error.detail}`);

}

} catch (error) {

alert('Connection error');

}

}

// Обновление данных каждые 30 секунд

setInterval(loadDashboardData, 30000);

</script>

</body>

</html>

"""

**1️⃣5️⃣ src/bot/trader.py - Торговая логика**

"""

Модуль для исполнения торговых операций

Путь: /var/www/www-root/data/www/systemetech.ru/src/bot/trader.py

"""

import logging

from typing import Optional, Dict

from datetime import datetime

from ..core.models import Trade, Signal, OrderSide, TradeStatus

from ..core.database import SessionLocal

from ..core.config import config

from ..exchange.client import ExchangeClient

logger = logging.getLogger(\_\_name\_\_)

class Trader:

"""Класс для исполнения торговых операций"""

def \_\_init\_\_(self, exchange: ExchangeClient):

self.exchange = exchange

async def execute\_signal(self, signal: Signal) -> Optional[Trade]:

"""Исполнение торгового сигнала"""

try:

# Получаем текущий баланс

balance = await self.exchange.fetch\_balance()

usdt\_balance = balance.get('USDT', {}).get('free', 0)

if usdt\_balance < 10: # Минимальный баланс

logger.warning(f"Недостаточный баланс: {usdt\_balance} USDT")

return None

# Рассчитываем размер позиции

position\_size\_usdt = usdt\_balance \* (config.MAX\_POSITION\_SIZE\_PERCENT / 100)

# Получаем текущую цену

ticker = await self.exchange.fetch\_ticker(signal.symbol)

current\_price = ticker['last']

# Рассчитываем количество

amount = self.exchange.calculate\_position\_size(

signal.symbol,

position\_size\_usdt,

config.MAX\_POSITION\_SIZE\_PERCENT

)

if amount <= 0:

logger.warning(f"Рассчитанный размер позиции слишком мал: {amount}")

return None

# Создаем ордер

order = await self.exchange.create\_order(

symbol=signal.symbol,

side=signal.action,

amount=amount,

order\_type='market'

)

if not order:

logger.error("Не удалось создать ордер")

return None

# Создаем запись о сделке

trade = Trade(

symbol=signal.symbol,

side=OrderSide[signal.action],

entry\_price=order.get('price', current\_price),

quantity=amount,

status=TradeStatus.OPEN,

strategy=signal.strategy,

stop\_loss=signal.stop\_loss,

take\_profit=signal.take\_profit,

created\_at=datetime.utcnow()

)

# Сохраняем в БД

db = SessionLocal()

try:

db.add(trade)

db.commit()

db.refresh(trade)

finally:

db.close()

logger.info(f"✅ Открыта позиция: {trade.side.value} {trade.quantity} {trade.symbol} @ {trade.entry\_price}")

return trade

except Exception as e:

logger.error(f"❌ Ошибка исполнения сигнала: {e}")

return None

async def close\_position(self, trade: Trade, current\_price: float) -> bool:

"""Закрытие позиции"""

try:

# Определяем направление закрывающего ордера

close\_side = 'SELL' if trade.side == OrderSide.BUY else 'BUY'

# Создаем закрывающий ордер

order = await self.exchange.create\_order(

symbol=trade.symbol,

side=close\_side,

amount=trade.quantity,

order\_type='market'

)

if not order:

logger.error("Не удалось создать закрывающий ордер")

return False

logger.info(f"✅ Закрыта позиция: {trade.symbol} @ {current\_price}")

return True

except Exception as e:

logger.error(f"❌ Ошибка закрытия позиции: {e}")

return False

**1️⃣6️⃣ src/bot/risk\_manager.py - Управление рисками**

"""

Модуль управления рисками

Путь: /var/www/www-root/data/www/systemetech.ru/src/bot/risk\_manager.py

"""

import logging

from typing import Dict, List

from datetime import datetime, timedelta

from ..core.config import config

from ..core.models import Signal, Trade

logger = logging.getLogger(\_\_name\_\_)

class RiskManager:

"""Управление рисками"""

def \_\_init\_\_(self):

self.max\_positions = config.MAX\_POSITIONS

self.max\_daily\_loss\_percent = 5.0 # Максимальный дневной убыток 5%

self.max\_position\_size\_percent = config.MAX\_POSITION\_SIZE\_PERCENT

self.min\_risk\_reward\_ratio = config.MIN\_RISK\_REWARD\_RATIO

# Статистика

self.daily\_stats = {

'trades': 0,

'wins': 0,

'losses': 0,

'total\_profit': 0,

'last\_reset': datetime.now().date()

}

def check\_signal(self, signal: Signal, open\_positions: Dict[str, Trade], current\_balance: float) -> bool:

"""Проверка сигнала на соответствие правилам риск-менеджмента"""

# Сбрасываем дневную статистику если новый день

self.\_reset\_daily\_stats\_if\_needed()

# 1. Проверка количества открытых позиций

if len(open\_positions) >= self.max\_positions:

logger.info(f"🚫 Достигнут лимит открытых позиций: {len(open\_positions)}/{self.max\_positions}")

return False

# 2. Проверка дневного лимита сделок

if self.daily\_stats['trades'] >= config.MAX\_DAILY\_TRADES:

logger.info(f"🚫 Достигнут дневной лимит сделок: {self.daily\_stats['trades']}/{config.MAX\_DAILY\_TRADES}")

return False

# 3. Проверка дневного убытка

daily\_loss\_percent = (self.daily\_stats['total\_profit'] / current\_balance) \* 100

if daily\_loss\_percent < -self.max\_daily\_loss\_percent:

logger.info(f"🚫 Достигнут дневной лимит убытков: {daily\_loss\_percent:.2f}%")

return False

# 4. Проверка риск/прибыль

if signal.stop\_loss and signal.take\_profit:

risk = abs(signal.price - signal.stop\_loss)

reward = abs(signal.take\_profit - signal.price)

if risk > 0:

risk\_reward\_ratio = reward / risk

if risk\_reward\_ratio < self.min\_risk\_reward\_ratio:

logger.info(f"🚫 Низкое соотношение риск/прибыль: {risk\_reward\_ratio:.2f}")

return False

# 5. Проверка на дублирование позиции

if signal.symbol in open\_positions:

logger.info(f"🚫 Позиция по {signal.symbol} уже открыта")

return False

# 6. Проверка минимальной уверенности

if signal.confidence < 0.6:

logger.info(f"🚫 Низкая уверенность в сигнале: {signal.confidence:.2f}")

return False

logger.info(f"✅ Сигнал прошел проверку риск-менеджмента")

return True

def update\_statistics(self, result: str, profit: float):

"""Обновление статистики"""

self.daily\_stats['trades'] += 1

if result == 'win':

self.daily\_stats['wins'] += 1

else:

self.daily\_stats['losses'] += 1

self.daily\_stats['total\_profit'] += profit

win\_rate = (self.daily\_stats['wins'] / self.daily\_stats['trades']) \* 100 if self.daily\_stats['trades'] > 0 else 0

logger.info(f"📊 Статистика дня: Сделок: {self.daily\_stats['trades']}, "

f"Win rate: {win\_rate:.1f}%, P&L: ${self.daily\_stats['total\_profit']:.2f}")

def \_reset\_daily\_stats\_if\_needed(self):

"""Сброс дневной статистики если новый день"""

today = datetime.now().date()

if today != self.daily\_stats['last\_reset']:

logger.info("📅 Новый торговый день - сброс статистики")

self.daily\_stats = {

'trades': 0,

'wins': 0,

'losses': 0,

'total\_profit': 0,

'last\_reset': today

}

def calculate\_position\_size(self, signal: Signal, balance: float) -> float:

"""Расчет размера позиции с учетом риска"""

# Базовый размер позиции

position\_size = balance \* (self.max\_position\_size\_percent / 100)

# Корректировка на основе уверенности

position\_size \*= signal.confidence

# Корректировка на основе текущей статистики

if self.daily\_stats['trades'] > 5:

win\_rate = self.daily\_stats['wins'] / self.daily\_stats['trades']

if win\_rate < 0.3: # Если win rate < 30%, уменьшаем размер

position\_size \*= 0.5

# Не больше максимального размера

max\_size = balance \* (self.max\_position\_size\_percent / 100)

position\_size = min(position\_size, max\_size)

return position\_size

**1️⃣7️⃣ src/analysis/market\_analyzer.py - Анализ рынка**

"""

Анализатор рынка

Путь: /var/www/www-root/data/www/systemetech.ru/src/analysis/market\_analyzer.py

"""

import logging

import pandas as pd

from typing import Dict, List, Optional

from datetime import datetime, timedelta

from ..exchange.client import exchange\_client

logger = logging.getLogger(\_\_name\_\_)

class MarketAnalyzer:

"""Анализ рыночных данных"""

def \_\_init\_\_(self):

self.exchange = exchange\_client

self.cache = {} # Кэш данных

self.cache\_ttl = 60 # TTL кэша в секундах

async def analyze\_symbol(self, symbol: str) -> Optional[Dict]:

"""Анализ конкретного символа"""

try:

# Проверяем кэш

if self.\_is\_cache\_valid(symbol):

logger.debug(f"Используем кэш для {symbol}")

return self.cache[symbol]['data']

# Получаем данные

ohlcv = await self.exchange.fetch\_ohlcv(symbol, '5m', 200)

if not ohlcv:

logger.warning(f"Нет данных для {symbol}")

return None

# Преобразуем в DataFrame

df = pd.DataFrame(

ohlcv,

columns=['timestamp', 'open', 'high', 'low', 'close', 'volume']

)

df['timestamp'] = pd.to\_datetime(df['timestamp'], unit='ms')

df.set\_index('timestamp', inplace=True)

# Получаем текущую цену

ticker = await self.exchange.fetch\_ticker(symbol)

current\_price = ticker['last']

# Рассчитываем дополнительные метрики

volatility = self.calculate\_volatility(df)

trend = self.detect\_trend(df)

support\_resistance = self.find\_support\_resistance(df)

volume\_profile = self.analyze\_volume(df)

result = {

'df': df,

'current\_price': current\_price,

'volatility': volatility,

'trend': trend,

'support': support\_resistance['support'],

'resistance': support\_resistance['resistance'],

'volume\_analysis': volume\_profile,

'timestamp': datetime.now()

}

# Сохраняем в кэш

self.cache[symbol] = {

'data': result,

'timestamp': datetime.now()

}

return result

except Exception as e:

logger.error(f"Ошибка анализа {symbol}: {e}")

return None

def calculate\_volatility(self, df: pd.DataFrame) -> Dict:

"""Расчет волатильности"""

# Дневная волатильность

returns = df['close'].pct\_change().dropna()

daily\_volatility = returns.std()

# ATR (Average True Range)

high\_low = df['high'] - df['low']

high\_close = abs(df['high'] - df['close'].shift())

low\_close = abs(df['low'] - df['close'].shift())

true\_range = pd.concat([high\_low, high\_close, low\_close], axis=1).max(axis=1)

atr = true\_range.rolling(window=14).mean().iloc[-1]

# Bollinger Bands width

sma = df['close'].rolling(window=20).mean()

std = df['close'].rolling(window=20).std()

bb\_width = (std \* 2) / sma

current\_bb\_width = bb\_width.iloc[-1]

return {

'daily': daily\_volatility,

'atr': atr,

'bb\_width': current\_bb\_width,

'is\_high': daily\_volatility > 0.02 # Высокая волатильность > 2%

}

def detect\_trend(self, df: pd.DataFrame) -> Dict:

"""Определение тренда"""

# Скользящие средние

sma\_20 = df['close'].rolling(window=20).mean()

sma\_50 = df['close'].rolling(window=50).mean()

sma\_200 = df['close'].rolling(window=200).mean()

current\_price = df['close'].iloc[-1]

# Определение тренда по MA

if current\_price > sma\_20.iloc[-1] > sma\_50.iloc[-1]:

trend\_ma = 'UPTREND'

elif current\_price < sma\_20.iloc[-1] < sma\_50.iloc[-1]:

trend\_ma = 'DOWNTREND'

else:

trend\_ma = 'SIDEWAYS'

# Тренд по High/Low

recent\_highs = df['high'].rolling(window=20).max()

recent\_lows = df['low'].rolling(window=20).min()

higher\_highs = recent\_highs.iloc[-1] > recent\_highs.iloc[-20]

higher\_lows = recent\_lows.iloc[-1] > recent\_lows.iloc[-20]

if higher\_highs and higher\_lows:

trend\_hl = 'UPTREND'

elif not higher\_highs and not higher\_lows:

trend\_hl = 'DOWNTREND'

else:

trend\_hl = 'SIDEWAYS'

# Сила тренда

trend\_strength = abs(sma\_20.iloc[-1] - sma\_50.iloc[-1]) / sma\_50.iloc[-1] \* 100

return {

'direction': trend\_ma,

'direction\_hl': trend\_hl,

'strength': trend\_strength,

'sma\_20': sma\_20.iloc[-1],

'sma\_50': sma\_50.iloc[-1],

'sma\_200': sma\_200.iloc[-1] if not pd.isna(sma\_200.iloc[-1]) else None

}

def find\_support\_resistance(self, df: pd.DataFrame) -> Dict:

"""Поиск уровней поддержки и сопротивления"""

# Простой метод - локальные минимумы и максимумы

window = 20

# Поддержка - локальные минимумы

lows = df['low'].rolling(window=window).min()

support\_levels = []

for i in range(window, len(df) - window):

if df['low'].iloc[i] == lows.iloc[i]:

support\_levels.append(df['low'].iloc[i])

# Сопротивление - локальные максимумы

highs = df['high'].rolling(window=window).max()

resistance\_levels = []

for i in range(window, len(df) - window):

if df['high'].iloc[i] == highs.iloc[i]:

resistance\_levels.append(df['high'].iloc[i])

# Кластеризация уровней

current\_price = df['close'].iloc[-1]

# Ближайшие уровни

nearest\_support = None

nearest\_resistance = None

if support\_levels:

below\_price = [s for s in support\_levels if s < current\_price]

if below\_price:

nearest\_support = max(below\_price)

if resistance\_levels:

above\_price = [r for r in resistance\_levels if r > current\_price]

if above\_price:

nearest\_resistance = min(above\_price)

return {

'support': nearest\_support,

'resistance': nearest\_resistance,

'all\_supports': sorted(set(support\_levels))[-5:], # Последние 5 уровней

'all\_resistances': sorted(set(resistance\_levels))[:5] # Первые 5 уровней

}

def analyze\_volume(self, df: pd.DataFrame) -> Dict:

"""Анализ объемов"""

# Средний объем

avg\_volume = df['volume'].rolling(window=20).mean()

current\_volume = df['volume'].iloc[-1]

volume\_ratio = current\_volume / avg\_volume.iloc[-1] if avg\_volume.iloc[-1] > 0 else 1

# Volume trend

volume\_sma\_5 = df['volume'].rolling(window=5).mean()

volume\_sma\_20 = df['volume'].rolling(window=20).mean()

volume\_trend = 'INCREASING' if volume\_sma\_5.iloc[-1] > volume\_sma\_20.iloc[-1] else 'DECREASING'

# Price-Volume correlation

price\_change = df['close'].pct\_change()

volume\_change = df['volume'].pct\_change()

correlation = price\_change.corr(volume\_change)

return {

'current': current\_volume,

'average': avg\_volume.iloc[-1],

'ratio': volume\_ratio,

'trend': volume\_trend,

'price\_correlation': correlation,

'is\_high': volume\_ratio > 1.5,

'is\_low': volume\_ratio < 0.5

}

def \_is\_cache\_valid(self, symbol: str) -> bool:

"""Проверка валидности кэша"""

if symbol not in self.cache:

return False

cache\_time = self.cache[symbol]['timestamp']

return (datetime.now() - cache\_time).total\_seconds() < self.cache\_ttl

**1️⃣8️⃣ src/notifications/telegram.py - Уведомления**

"""

Telegram уведомления

Путь: /var/www/www-root/data/www/systemetech.ru/src/notifications/telegram.py

"""

import os

import asyncio

import logging

from datetime import datetime, timedelta

from typing import Optional, List

import aiohttp

from ..core.config import config

logger = logging.getLogger(\_\_name\_\_)

class TelegramNotifier:

"""Отправка уведомлений в Telegram"""

def \_\_init\_\_(self):

self.bot\_token = config.TELEGRAM\_BOT\_TOKEN

self.chat\_id = config.TELEGRAM\_CHAT\_ID

self.enabled = bool(self.bot\_token and self.chat\_id)

if not self.enabled:

logger.warning("⚠️ Telegram уведомления отключены (не настроены токен или chat\_id)")

self.base\_url = f"https://api.telegram.org/bot{self.bot\_token}"

async def send\_message(self, text: str, parse\_mode: str = 'HTML', disable\_notification: bool = False):

"""Отправка сообщения"""

if not self.enabled:

return

try:

async with aiohttp.ClientSession() as session:

url = f"{self.base\_url}/sendMessage"

data = {

'chat\_id': self.chat\_id,

'text': text,

'parse\_mode': parse\_mode,

'disable\_notification': disable\_notification

}

async with session.post(url, json=data) as response:

if response.status != 200:

error\_text = await response.text()

logger.error(f"❌ Ошибка отправки в Telegram: {error\_text}")

except Exception as e:

logger.error(f"❌ Ошибка отправки в Telegram: {e}")

async def send\_startup\_message(self, pairs: List[str], mode: str):

"""Уведомление о запуске бота"""

text = f"""🚀 <b>Бот запущен</b>

📊 Режим: <code>{mode}</code>

💱 Пары: <code>{', '.join(pairs)}</code>

⏰ Время: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

🌐 Веб: <a href="http://systemetech.ru:8000">Открыть дашборд</a>

<i>Удачной торговли!</i>"""

await self.send\_message(text)

async def send\_shutdown\_message(self, runtime: Optional[timedelta], trades\_count: int):

"""Уведомление об остановке бота"""

runtime\_str = str(runtime).split('.')[0] if runtime else "Неизвестно"

text = f"""🛑 <b>Бот остановлен</b>

⏱️ Время работы: <code>{runtime\_str}</code>

📊 Сделок за сессию: <code>{trades\_count}</code>

⏰ Время: {datetime.now().strftime('%Y-%m-%d %H:%M:%S')}

<i>До встречи!</i>"""

await self.send\_message(text)

async def send\_trade\_opened(self, symbol: str, side: str, amount: float, price: float):

"""Уведомление об открытии сделки"""

emoji = "🟢" if side == "BUY" else "🔴"

text = f"""{emoji} <b>Открыта позиция</b>

💱 Пара: <code>{symbol}</code>

📊 Сторона: <code>{side}</code>

📈 Количество: <code>{amount:.4f}</code>

💵 Цена входа: <code>${price:.2f}</code>

⏰ Время: {datetime.now().strftime('%H:%M:%S')}"""

await self.send\_message(text)

async def send\_trade\_closed(self, symbol: str, side: str, profit: float, reason: str):

"""Уведомление о закрытии сделки"""

emoji = "✅" if profit > 0 else "❌"

profit\_emoji = "📈" if profit > 0 else "📉"

text = f"""{emoji} <b>Закрыта позиция</b>

💱 Пара: <code>{symbol}</code>

📊 Сторона: <code>{side}</code>

{profit\_emoji} Прибыль: <code>${profit:.2f}</code>

📝 Причина: <i>{reason}</i>

⏰ Время: {datetime.now().strftime('%H:%M:%S')}"""

await self.send\_message(text)

async def send\_daily\_report(self, stats: Dict):

"""Ежедневный отчет"""

text = f"""📊 <b>Дневной отчет</b>

📅 Дата: {datetime.now().strftime('%Y-%m-%d')}

📈 <b>Статистика:</b>

• Всего сделок: <code>{stats.get('total\_trades', 0)}</code>

• Прибыльных: <code>{stats.get('profitable\_trades', 0)}</code>

• Убыточных: <code>{stats.get('losing\_trades', 0)}</code>

• Win Rate: <code>{stats.get('win\_rate', 0):.1f}%</code>

💰 <b>Финансы:</b>

• Общая прибыль: <code>${stats.get('total\_profit', 0):.2f}</code>

• Средняя прибыль: <code>${stats.get('avg\_profit', 0):.2f}</code>

• Лучшая сделка: <code>${stats.get('best\_trade', 0):.2f}</code>

• Худшая сделка: <code>${stats.get('worst\_trade', 0):.2f}</code>

💼 <b>Баланс:</b>

• Начальный: <code>${stats.get('start\_balance', 0):.2f}</code>

• Текущий: <code>${stats.get('current\_balance', 0):.2f}</code>

• Изменение: <code>{stats.get('balance\_change', 0):.2f}%</code>

🏆 <b>Лучшая пара:</b> <code>{stats.get('best\_pair', 'N/A')}</code>

<i>Хорошего дня!</i>"""

await self.send\_message(text, disable\_notification=True)

async def send\_error(self, error: str):

"""Уведомление об ошибке"""

# Ограничиваем длину сообщения

if len(error) > 500:

error = error[:497] + "..."

text = f"""🚨 <b>Ошибка</b>

<code>{error}</code>

⏰ Время: {datetime.now().strftime('%H:%M:%S')}"""

await self.send\_message(text)

async def send\_warning(self, warning: str):

"""Уведомление о предупреждении"""

text = f"""⚠️ <b>Предупреждение</b>

{warning}

⏰ Время: {datetime.now().strftime('%H:%M:%S')}"""

await self.send\_message(text, disable\_notification=True)

# Глобальный экземпляр

telegram\_notifier = TelegramNotifier()

**1️⃣9️⃣ src/strategies/momentum.py - Momentum стратегия**

"""

Momentum стратегия

Путь: /var/www/www-root/data/www/systemetech.ru/src/strategies/momentum.py

"""

import pandas as pd

import numpy as np

from ta.momentum import RSIIndicator, ROCIndicator

from ta.trend import EMAIndicator

from ta.volatility import AverageTrueRange

import logging

from .base import BaseStrategy, TradingSignal

logger = logging.getLogger(\_\_name\_\_)

class MomentumStrategy(BaseStrategy):

"""

Простая momentum стратегия

Торгует по направлению сильного движения

"""

def \_\_init\_\_(self):

super().\_\_init\_\_("momentum")

self.rsi\_period = 14

self.ema\_fast = 9

self.ema\_slow = 21

self.roc\_period = 10

self.min\_momentum\_score = 0.6

async def analyze(self, df: pd.DataFrame, symbol: str) -> TradingSignal:

"""Анализ momentum"""

if not self.validate\_dataframe(df):

return TradingSignal('WAIT', 0, 0, reason='Недостаточно данных')

try:

# Рассчитываем индикаторы

indicators = self.\_calculate\_indicators(df)

# Анализируем momentum

momentum\_score = self.\_analyze\_momentum(indicators)

# Принимаем решение

return self.\_make\_decision(momentum\_score, indicators, df)

except Exception as e:

logger.error(f"Ошибка анализа momentum для {symbol}: {e}")

return TradingSignal('WAIT', 0, 0, reason=f'Ошибка анализа: {e}')

def \_calculate\_indicators(self, df: pd.DataFrame) -> Dict:

"""Расчет индикаторов momentum"""

indicators = {}

# RSI

rsi = RSIIndicator(df['close'], window=self.rsi\_period)

indicators['rsi'] = rsi.rsi().iloc[-1]

indicators['rsi\_prev'] = rsi.rsi().iloc[-2]

# EMA

indicators['ema\_fast'] = EMAIndicator(df['close'], window=self.ema\_fast).ema\_indicator().iloc[-1]

indicators['ema\_slow'] = EMAIndicator(df['close'], window=self.ema\_slow).ema\_indicator().iloc[-1]

# Rate of Change

roc = ROCIndicator(df['close'], window=self.roc\_period)

indicators['roc'] = roc.roc().iloc[-1]

# Price momentum

indicators['price\_change\_5'] = ((df['close'].iloc[-1] - df['close'].iloc[-6]) / df['close'].iloc[-6]) \* 100

indicators['price\_change\_10'] = ((df['close'].iloc[-1] - df['close'].iloc[-11]) / df['close'].iloc[-11]) \* 100

# Volume momentum

indicators['volume\_ratio'] = df['volume'].iloc[-1] / df['volume'].rolling(window=20).mean().iloc[-1]

# ATR для volatility

atr = AverageTrueRange(df['high'], df['low'], df['close'])

indicators['atr'] = atr.average\_true\_range().iloc[-1]

# Текущая цена

indicators['current\_price'] = df['close'].iloc[-1]

return indicators

def \_analyze\_momentum(self, indicators: Dict) -> Dict:

"""Анализ силы momentum"""

momentum\_score = {

'direction': None,

'strength': 0,

'components': []

}

bullish\_score = 0

bearish\_score = 0

# RSI momentum

if indicators['rsi'] > 50 and indicators['rsi'] > indicators['rsi\_prev']:

bullish\_score += 0.2

momentum\_score['components'].append('RSI восходящий')

elif indicators['rsi'] < 50 and indicators['rsi'] < indicators['rsi\_prev']:

bearish\_score += 0.2

momentum\_score['components'].append('RSI нисходящий')

# EMA momentum

if indicators['ema\_fast'] > indicators['ema\_slow']:

bullish\_score += 0.25

momentum\_score['components'].append('EMA бычий крест')

else:

bearish\_score += 0.25

momentum\_score['components'].append('EMA медвежий крест')

# Price momentum

if indicators['price\_change\_5'] > 1 and indicators['price\_change\_10'] > 2:

bullish\_score += 0.3

momentum\_score['components'].append(f"Рост цены {indicators['price\_change\_5']:.1f}%")

elif indicators['price\_change\_5'] < -1 and indicators['price\_change\_10'] < -2:

bearish\_score += 0.3

momentum\_score['components'].append(f"Падение цены {indicators['price\_change\_5']:.1f}%")

# ROC momentum

if indicators['roc'] > 2:

bullish\_score += 0.15

momentum\_score['components'].append('Сильный ROC')

elif indicators['roc'] < -2:

bearish\_score += 0.15

momentum\_score['components'].append('Слабый ROC')

# Volume confirmation

if indicators['volume\_ratio'] > 1.5:

if bullish\_score > bearish\_score:

bullish\_score += 0.1

else:

bearish\_score += 0.1

momentum\_score['components'].append('Высокий объем')

# Определяем направление и силу

if bullish\_score > bearish\_score:

momentum\_score['direction'] = 'BULLISH'

momentum\_score['strength'] = bullish\_score

else:

momentum\_score['direction'] = 'BEARISH'

momentum\_score['strength'] = bearish\_score

return momentum\_score

def \_make\_decision(self, momentum\_score: Dict, indicators: Dict, df: pd.DataFrame) -> TradingSignal:

"""Принятие решения на основе momentum"""

if momentum\_score['strength'] < self.min\_momentum\_score:

return TradingSignal(

action='WAIT',

confidence=0,

price=indicators['current\_price'],

reason=f"Слабый momentum: {momentum\_score['strength']:.2f}"

)

# Определяем действие

if momentum\_score['direction'] == 'BULLISH':

action = 'BUY'

else:

action = 'SELL'

# Расчет уровней

atr = indicators['atr']

stop\_loss = self.calculate\_stop\_loss(indicators['current\_price'], action, atr, 2.0)

take\_profit = self.calculate\_take\_profit(indicators['current\_price'], action, atr, 3.0)

risk\_reward = self.calculate\_risk\_reward(indicators['current\_price'], stop\_loss, take\_profit)

# Формируем причину

reason = f"Momentum {momentum\_score['direction']}: {', '.join(momentum\_score['components'][:2])}"

return TradingSignal(

action=action,

confidence=min(0.9, momentum\_score['strength']),

price=indicators['current\_price'],

stop\_loss=stop\_loss,

take\_profit=take\_profit,

reason=reason,

risk\_reward\_ratio=risk\_reward,

indicators=indicators

)

**2️⃣0️⃣ src/strategies/scalping.py - Скальпинг стратегия**

"""

Скальпинг стратегия

Путь: /var/www/www-root/data/www/systemetech.ru/src/strategies/scalping.py

"""

import pandas as pd

import numpy as np

from ta.momentum import RSIIndicator

from ta.volatility import BollingerBands, AverageTrueRange

from ta.volume import VolumeWeightedAveragePrice

import logging

from .base import BaseStrategy, TradingSignal

logger = logging.getLogger(\_\_name\_\_)

class ScalpingStrategy(BaseStrategy):

"""

Скальпинг стратегия для быстрых сделок

Работает на малых таймфреймах с жесткими стопами

"""

def \_\_init\_\_(self):

super().\_\_init\_\_("scalping")

self.bb\_period = 20

self.bb\_std = 2

self.rsi\_period = 7 # Быстрый RSI

self.min\_profit\_percent = 0.3 # Минимум 0.3% профита

self.max\_loss\_percent = 0.5 # Максимум 0.5% убытка

self.min\_volume\_ratio = 1.2 # Минимальный объем

async def analyze(self, df: pd.DataFrame, symbol: str) -> TradingSignal:

"""Анализ для скальпинга"""

if not self.validate\_dataframe(df):

return TradingSignal('WAIT', 0, 0, reason='Недостаточно данных')

try:

# Рассчитываем индикаторы

indicators = self.\_calculate\_indicators(df)

# Проверяем условия для скальпинга

scalp\_signal = self.\_check\_scalping\_conditions(indicators)

# Принимаем решение

return self.\_make\_scalping\_decision(scalp\_signal, indicators)

except Exception as e:

logger.error(f"Ошибка анализа скальпинга для {symbol}: {e}")

return TradingSignal('WAIT', 0, 0, reason=f'Ошибка анализа: {e}')

def \_calculate\_indicators(self, df: pd.DataFrame) -> Dict:

"""Расчет индикаторов для скальпинга"""

indicators = {}

# Bollinger Bands

bb = BollingerBands(df['close'], window=self.bb\_period, window\_dev=self.bb\_std)

indicators['bb\_upper'] = bb.bollinger\_hband().iloc[-1]

indicators['bb\_lower'] = bb.bollinger\_lband().iloc[-1]

indicators['bb\_middle'] = bb.bollinger\_mavg().iloc[-1]

indicators['bb\_width'] = bb.bollinger\_wband().iloc[-1]

indicators['bb\_percent'] = bb.bollinger\_pband().iloc[-1]

# RSI

rsi = RSIIndicator(df['close'], window=self.rsi\_period)

indicators['rsi'] = rsi.rsi().iloc[-1]

# VWAP

vwap = VolumeWeightedAveragePrice(df['high'], df['low'], df['close'], df['volume'])

indicators['vwap'] = vwap.volume\_weighted\_average\_price().iloc[-1]

# ATR для расчета стопов

atr = AverageTrueRange(df['high'], df['low'], df['close'], window=14)

indicators['atr'] = atr.average\_true\_range().iloc[-1]

indicators['atr\_percent'] = (indicators['atr'] / df['close'].iloc[-1]) \* 100

# Volume analysis

indicators['volume\_sma'] = df['volume'].rolling(window=20).mean().iloc[-1]

indicators['volume\_ratio'] = df['volume'].iloc[-1] / indicators['volume\_sma']

# Price action

indicators['current\_price'] = df['close'].iloc[-1]

indicators['price\_range'] = df['high'].iloc[-1] - df['low'].iloc[-1]

indicators['candle\_body'] = abs(df['close'].iloc[-1] - df['open'].iloc[-1])

indicators['upper\_wick'] = df['high'].iloc[-1] - max(df['close'].iloc[-1], df['open'].iloc[-1])

indicators['lower\_wick'] = min(df['close'].iloc[-1], df['open'].iloc[-1]) - df['low'].iloc[-1]

# Микро-тренд (последние 5 свечей)

indicators['micro\_trend'] = 'UP' if df['close'].iloc[-1] > df['close'].iloc[-5] else 'DOWN'

return indicators

def \_check\_scalping\_conditions(self, indicators: Dict) -> Dict:

"""Проверка условий для скальпинга"""

signal = {

'direction': None,

'strength': 0,

'entry\_type': None,

'reasons': []

}

# Проверяем волатильность - для скальпинга нужна умеренная волатильность

if indicators['atr\_percent'] > 3:

signal['reasons'].append('Слишком высокая волатильность')

return signal

if indicators['atr\_percent'] < 0.5:

signal['reasons'].append('Слишком низкая волатильность')

return signal

# Проверяем объем

if indicators['volume\_ratio'] < self.min\_volume\_ratio:

signal['reasons'].append('Недостаточный объем')

return signal

# === УСЛОВИЯ ДЛЯ ПОКУПКИ ===

# 1. Отскок от нижней BB

if (indicators['bb\_percent'] < 0.2 and

indicators['rsi'] < 35 and

indicators['lower\_wick'] > indicators['candle\_body'] \* 1.5): # Длинная нижняя тень

signal['direction'] = 'BUY'

signal['entry\_type'] = 'BB\_BOUNCE'

signal['strength'] = 0.8

signal['reasons'].append('Отскок от нижней BB с подтверждением')

# 2. Прорыв VWAP вверх с объемом

elif (indicators['current\_price'] > indicators['vwap'] and

indicators['current\_price'] < indicators['vwap'] \* 1.002 and # Только что пробили

indicators['volume\_ratio'] > 1.5 and

indicators['micro\_trend'] == 'UP'):

signal['direction'] = 'BUY'

signal['entry\_type'] = 'VWAP\_BREAK'

signal['strength'] = 0.7

signal['reasons'].append('Прорыв VWAP с объемом')

# === УСЛОВИЯ ДЛЯ ПРОДАЖИ ===

# 1. Отскок от верхней BB

elif (indicators['bb\_percent'] > 0.8 and

indicators['rsi'] > 65 and

indicators['upper\_wick'] > indicators['candle\_body'] \* 1.5): # Длинная верхняя тень

signal['direction'] = 'SELL'

signal['entry\_type'] = 'BB\_BOUNCE'

signal['strength'] = 0.8

signal['reasons'].append('Отскок от верхней BB с подтверждением')

# 2. Пробой VWAP вниз с объемом

elif (indicators['current\_price'] < indicators['vwap'] and

indicators['current\_price'] > indicators['vwap'] \* 0.998 and # Только что пробили

indicators['volume\_ratio'] > 1.5 and

indicators['micro\_trend'] == 'DOWN'):

signal['direction'] = 'SELL'

signal['entry\_type'] = 'VWAP\_BREAK'

signal['strength'] = 0.7

signal['reasons'].append('Пробой VWAP с объемом')

return signal

def \_make\_scalping\_decision(self, scalp\_signal: Dict, indicators: Dict) -> TradingSignal:

"""Принятие решения для скальпинга"""

if not scalp\_signal['direction']:

reason = scalp\_signal['reasons'][0] if scalp\_signal['reasons'] else 'Нет сигнала'

return TradingSignal(

action='WAIT',

confidence=0,

price=indicators['current\_price'],

reason=reason

)

# Расчет жестких уровней для скальпинга

current\_price = indicators['current\_price']

if scalp\_signal['direction'] == 'BUY':

# Для покупки

stop\_loss = current\_price \* (1 - self.max\_loss\_percent / 100)

take\_profit = current\_price \* (1 + self.min\_profit\_percent / 100)

else:

# Для продажи

stop\_loss = current\_price \* (1 + self.max\_loss\_percent / 100)

take\_profit = current\_price \* (1 - self.min\_profit\_percent / 100)

# Проверяем risk/reward

risk\_reward = self.calculate\_risk\_reward(current\_price, stop\_loss, take\_profit)

if risk\_reward < 0.6: # Для скальпинга допустим меньший R:R

return TradingSignal(

action='WAIT',

confidence=0,

price=current\_price,

reason=f"Низкий R:R для скальпинга: {risk\_reward:.2f}"

)

# Формируем сигнал

return TradingSignal(

action=scalp\_signal['direction'],

confidence=scalp\_signal['strength'],

price=current\_price,

stop\_loss=stop\_loss,

take\_profit=take\_profit,

reason=f"Скальпинг: {scalp\_signal['reasons'][0]}",

risk\_reward\_ratio=risk\_reward,

indicators=indicators

)

**2️⃣1️⃣ main.py - Единая точка входа**

#!/usr/bin/env python3

"""

Crypto Trading Bot v3.0 - Единая точка входа

Путь: /var/www/www-root/data/www/systemetech.ru/main.py

"""

import asyncio

import argparse

import logging

import signal

import sys

from pathlib import Path

# Добавляем корневую директорию в путь

sys.path.append(str(Path(\_\_file\_\_).parent))

from src.core.config import config

from src.bot.manager import bot\_manager

from src.web.websocket import ws\_manager

# Настройка логирования

logging.basicConfig(

level=logging.INFO,

format='%(asctime)s - %(name)s - %(levelname)s - %(message)s',

handlers=[

logging.FileHandler('logs/trading.log'),

logging.StreamHandler()

]

)

logger = logging.getLogger(\_\_name\_\_)

# Глобальная переменная для остановки

stop\_event = asyncio.Event()

def signal\_handler(sig, frame):

"""Обработчик сигналов системы"""

logger.info(f"📡 Получен сигнал {sig}, инициируем graceful shutdown...")

stop\_event.set()

# Регистрируем обработчики сигналов

signal.signal(signal.SIGINT, signal\_handler)

signal.signal(signal.SIGTERM, signal\_handler)

async def run\_bot():

"""Запуск торгового бота"""

logger.info("==================================================")

logger.info("🤖 Crypto Trading Bot v3.0")

logger.info("==================================================")

logger.info("🚀 Запуск торгового бота...")

try:

# Запускаем бота

success, message = await bot\_manager.start()

if not success:

logger.error(f"❌ Не удалось запустить бота: {message}")

return

# Ждем сигнала остановки

logger.info("🔄 Бот запущен. Нажмите Ctrl+C для остановки")

await stop\_event.wait()

logger.info("🛑 Получен сигнал остановки, инициируем shutdown...")

except Exception as e:

logger.error(f"💥 Критическая ошибка: {e}", exc\_info=True)

finally:

# Останавливаем бота

logger.info("🛑 Остановка бота...")

await bot\_manager.stop()

logger.info("✅ Бот остановлен корректно")

def run\_web():

"""Запуск веб-интерфейса"""

logger.info("==================================================")

logger.info("🌐 Crypto Trading Bot Web Interface")

logger.info("==================================================")

logger.info(f"🌐 Запуск веб-интерфейса на {config.WEB\_HOST}:{config.WEB\_PORT}...")

import uvicorn

from src.web.app import app

# Запускаем фоновую задачу для WebSocket broadcast

asyncio.create\_task(ws\_manager.start\_broadcast\_loop())

uvicorn.run(

app,

host=config.WEB\_HOST,

port=config.WEB\_PORT,

log\_level="info"

)

async def run\_all():

"""Запуск всей системы"""

logger.info("==================================================")

logger.info("🚀 Crypto Trading Bot v3.0 - Full System")

logger.info("==================================================")

# Запускаем веб в отдельном процессе

import multiprocessing

web\_process = multiprocessing.Process(target=run\_web)

web\_process.start()

# Небольшая задержка чтобы веб успел запуститься

await asyncio.sleep(2)

try:

мы все внесли, но ты не дописал main.py   
и возможно другие коды. Проанализируй наш код, в репозитории сейчас наш старый код, для понимания тебе, но сейчас в новой версии все точно так как ты написал,   
т.е не нужно переписывать эти коды, просто проанализировать что у нас было и что нужно еще сделать.  
  
по этому, я прошу тебя дописать main до конца.  
далее как все это собрать в виртуальном окружении.  
далее как проверить все что у нас настроено  
как запустить интерфейс, как ранее писал, из интерфейса все должно работать. Т.е я запускаю интерфейс и все делаю там!  
  
по этому допиши что нужно еще сделать, пропиши все что нужно. В том числе какие команды выполнить! Структура и код должен быть единым целым.!  
также нужно выделить те файлы которые нужно удалить из проекта как лишние!