```
import os
from datetime import datetime, timedelta
import yfinance as yf
from fredapi import Fred
from swarms import Agent, AgentRearrange
from swarm_models import OpenAIChat
import logging
from dotenv import load_dotenv
import asyncio
import aiohttp
from ratelimit import limits, sleep_and_retry


# Load environment variables
load_dotenv()


# Set up logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s - %(levelname)s - %(message)s",
)
logger = logging.getLogger(__name__)


# Get the OpenAI API key from the environment variable
api_key = os.getenv("GROQ_API_KEY")


# Model
```

```
model = OpenAIChat(
    openai_api_base="https://api.groq.com/openai/v1",
    openai_api_key=api_key,
    model_name="llama-3.1-70b-versatile",
    temperature=0.1,
)

# API keys
POLYGON_API_KEY = os.getenv("POLYGON_API_KEY")
FRED_API_KEY = os.getenv("FRED_API_KEY")
OPENAI_API_KEY = os.getenv("OPENAI_API_KEY")

# Initialize FRED client
fred_client = Fred(api_key=FRED_API_KEY)

# Polygon API base URL
POLYGON_BASE_URL = "https://api.polygon.io"


# Rate limiting decorators
@sleep_and_retry
@limits(
    calls=5, period=60
)  # Adjust these values based on your Polygon API tier
async def call_polygon_api(session, endpoint, params=None):
    url = f"{POLYGON_BASE_URL}{endpoint}"
    params = params or {}
    params["apiKey"] = POLYGON_API_KEY
    async with session.get(url, params=params) as response:
        response.raise_for_status()
        return await response.json()


@sleep_and_retry
@limits(calls=120, period=60)  # FRED allows 120 requests per minute
def call_fred_api(func, *args, **kwargs):
    return func(*args, **kwargs)


# Yahoo Finance Integration
async def get_yahoo_finance_data(
    session, ticker, period="1d", interval="1m"
):
    try:
        stock = yf.Ticker(ticker)
        hist = await asyncio.to_thread(
            stock.history, period=period, interval=interval
        )
        info = await asyncio.to_thread(lambda: stock.info)
        return hist, info
    except Exception as e:
        logger.error(
            f"Error fetching Yahoo Finance data for {ticker}: {e}"
        )
        return None, None


async def get_yahoo_finance_realtime(session, ticker):
    try:
        stock = yf.Ticker(ticker)
        return await asyncio.to_thread(lambda: stock.fast_info)
    except Exception as e:
        logger.error(
            f"Error fetching Yahoo Finance realtime data for {ticker}: {e}"
        )
        return None


# Polygon.io Integration
async def get_polygon_realtime_data(session, ticker):
    try:
        trades = await call_polygon_api(
            session, f"/v2/last/trade/{ticker}"
        )
        quotes = await call_polygon_api(
            session, f"/v2/last/nbbo/{ticker}"
        )
        return trades, quotes
    except Exception as e:
        logger.error(
            f"Error fetching Polygon.io realtime data for {ticker}: {e}"
        )
        return None, None


async def get_polygon_news(session, ticker, limit=10):
    try:
        news = await call_polygon_api(
            session,
            "/v2/reference/news",
            params={"ticker": ticker, "limit": limit},
        )
        return news.get("results", [])
    except Exception as e:
        logger.error(
            f"Error fetching Polygon.io news for {ticker}: {e}"
        )
        return []


# FRED Integration
async def get_fred_data(session, series_id, start_date, end_date):
    try:
        data = await asyncio.to_thread(
            call_fred_api,
            fred_client.get_series,
            series_id,
            start_date,
            end_date,
        )
        return data
    except Exception as e:
        logger.error(f"Error fetching FRED data for {series_id}: {e}")
        return None


async def get_fred_realtime(session, series_ids):
    try:
        data = {}
        for series_id in series_ids:
            series = await asyncio.to_thread(
                call_fred_api, fred_client.get_series, series_id
            )
            data[series_id] = series.iloc[
                -1
            ]  # Get the most recent value
        return data
    except Exception as e:
        logger.error(f"Error fetching FRED realtime data: {e}")
        return {}
```

```
# Creating Specialized Agents
stock_agent = Agent(
    agent_name="StockAgent",
    system_prompt="""You are an expert stock analyst. Your task is to analyze real-time stock data and provide insights.
    Consider price movements, trading volume, and any available company information.
    Provide a concise summary of the stock's current status and any notable trends or events.""",
    llm=model,
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

market_agent = Agent(
    agent_name="MarketAgent",
    system_prompt="""You are a market analysis expert. Your task is to analyze overall market conditions using real-time data.
    Consider major indices, sector performance, and market-wide trends.
    Provide a concise summary of current market conditions and any significant developments.""",
    llm=model,
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

macro_agent = Agent(
    agent_name="MacroAgent",
    system_prompt="""You are a macroeconomic analysis expert. Your task is to analyze key
economic indicators and provide insights on the overall economic situation.
    Consider GDP growth, inflation rates, unemployment figures, and other relevant economic data.
    Provide a concise summary of the current economic situation and any potential impacts on
financial markets.""",
    llm=model,
    max_loops=1,
    dashboard=False,
    streaming_on=True,
    verbose=True,
)

news_agent = Agent(
    agent_name="NewsAgent",
    system_prompt="""You are a financial news analyst. Your task is to analyze recent news articles
related to specific stocks or the overall market.
    Consider the potential impact of news events on stock prices or market trends.
    Provide a concise summary of key news items and their potential market implications.""",
    llm=model,
    max_loops=1,
    dashboard=False,
    streaming_on=True,

    verbose=True,

)


# Building the Multi-Agent System

agents = [stock_agent, market_agent, macro_agent, news_agent]

flow = "StockAgent -> MarketAgent -> MacroAgent -> NewsAgent"


agent_system = AgentRearrange(agents=agents, flow=flow)




# Real-Time Data Analysis

async def real_time_analysis(session, ticker):

    logger.info(f"Starting real-time analysis for {ticker}")


    # Fetch real-time data

    yf_data, yf_info = await get_yahoo_finance_data(session, ticker)

    yf_realtime = await get_yahoo_finance_realtime(session, ticker)

    polygon_trades, polygon_quotes = await get_polygon_realtime_data(

        session, ticker

    )

    polygon_news = await get_polygon_news(session, ticker)

    fred_data = await get_fred_realtime(

        session, ["GDP", "UNRATE", "CPIAUCSL"]

    )
    # Prepare input for the multi-agent system
    input_data = f"""
    Yahoo Finance Data:
    {yf_realtime}

    Recent Stock History:
    {yf_data.tail().to_string() if yf_data is not None else 'Data unavailable'}

    Polygon.io Trade Data:
    {polygon_trades}

    Polygon.io Quote Data:
    {polygon_quotes}

    Recent News:
    {polygon_news[:3] if polygon_news else 'No recent news available'}

    Economic Indicators:
    {fred_data}

      Analyze this real-time financial data for {ticker}. Provide insights on the stock's performance,
overall market conditions, relevant economic factors, and any significant news that might impact the
stock or market.
    """

    # Run the multi-agent analysis
    try:
        analysis = agent_system.run(input_data)
        logger.info(f"Analysis completed for {ticker}")
        return analysis
    except Exception as e:
        logger.error(
            f"Error during multi-agent analysis for {ticker}: {e}"
        )
        return f"Error during analysis: {e}"



# Advanced Use Cases
async def compare_stocks(session, tickers):
    results = {}
    for ticker in tickers:
        results[ticker] = await real_time_analysis(session, ticker)

    comparison_prompt = f"""
    Compare the following stocks based on the provided analyses:
    {results}

    Highlight key differences and similarities. Provide a ranking of these stocks based on their current
performance and future prospects.
    """

    try:
        comparison = agent_system.run(comparison_prompt)

        logger.info(f"Stock comparison completed for {tickers}")

        return comparison

    except Exception as e:

        logger.error(f"Error during stock comparison: {e}")

        return f"Error during comparison: {e}"




async def sector_analysis(session, sector):

    sector_stocks = {

        "Technology": ["AAPL", "MSFT", "GOOGL", "AMZN", "NVDA"],

        "Finance": ["JPM", "BAC", "WFC", "C", "GS"],

        "Healthcare": ["JNJ", "UNH", "PFE", "ABT", "MRK"],

        "Consumer Goods": ["PG", "KO", "PEP", "COST", "WMT"],

        "Energy": ["XOM", "CVX", "COP", "SLB", "EOG"],

    }


    if sector not in sector_stocks:

        return f"Sector '{sector}' not found. Available sectors: {', '.join(sector_stocks.keys())}"


    stocks = sector_stocks[sector][:5]


    sector_data = {}

    for stock in stocks:

        sector_data[stock] = await real_time_analysis(session, stock)
    sector_prompt = f"""
    Analyze the {sector} sector based on the following data from its top stocks:
    {sector_data}

    Provide insights on:
    1. Overall sector performance
    2. Key trends within the sector
    3. Top performing stocks and why they're outperforming
    4. Any challenges or opportunities facing the sector
    """

    try:
        analysis = agent_system.run(sector_prompt)
        logger.info(f"Sector analysis completed for {sector}")
        return analysis
    except Exception as e:
        logger.error(
            f"Error during sector analysis for {sector}: {e}"
        )
        return f"Error during sector analysis: {e}"


async def economic_impact_analysis(session, indicator, threshold):
    # Fetch historical data for the indicator
    end_date = datetime.now().strftime("%Y-%m-%d")
    start_date = (datetime.now() - timedelta(days=365)).strftime(
        "%Y-%m-%d"
    )
    indicator_data = await get_fred_data(
        session, indicator, start_date, end_date
    )

    if indicator_data is None or len(indicator_data) < 2:
        return f"Insufficient data for indicator {indicator}"

    # Check if the latest value crosses the threshold
    latest_value = indicator_data.iloc[-1]
    previous_value = indicator_data.iloc[-2]
    crossed_threshold = (
        latest_value > threshold and previous_value <= threshold
    ) or (latest_value < threshold and previous_value >= threshold)

    if crossed_threshold:
        impact_prompt = f"""
         The economic indicator {indicator} has crossed the threshold of {threshold}. Its current value is
{latest_value}.

        Historical data:
        {indicator_data.tail().to_string()}

        Analyze the potential impacts of this change on:
        1. Overall economic conditions
        2. Different market sectors
        3. Specific types of stocks (e.g., growth vs. value)
        4. Other economic indicators

        Provide a comprehensive analysis of the potential consequences and any recommended
actions for investors.
        """

        try:
            analysis = agent_system.run(impact_prompt)
            logger.info(
                f"Economic impact analysis completed for {indicator}"
            )
            return analysis
        except Exception as e:
            logger.error(
                f"Error during economic impact analysis for {indicator}: {e}"
            )
            return f"Error during economic impact analysis: {e}"
    else:
            return f"The {indicator} indicator has not crossed the threshold of {threshold}. Current value:
{latest_value}"


async def main():
    async with aiohttp.ClientSession() as session:
```

```
    # Example usage
    analysis_result = await real_time_analysis(session, "AAPL")
    print("Single Stock Analysis:")
    print(analysis_result)

    comparison_result = await compare_stocks(
        session, ["AAPL", "GOOGL", "MSFT"]
    )
    print("\nStock Comparison:")
    print(comparison_result)

    tech_sector_analysis = await sector_analysis(
        session, "Technology"
    )
    print("\nTechnology Sector Analysis:")
    print(tech_sector_analysis)

    gdp_impact = await economic_impact_analysis(
        session, "GDP", 22000
    )
    print("\nEconomic Impact Analysis:")
    print(gdp_impact)


if __name__ == "__main__":
    asyncio.run(main())
```