```
from typing import List, Dict
from dataclasses import dataclass
from datetime import datetime
import asyncio
import aiohttp
from loguru import logger
from swarms import Agent
from pathlib import Path
import json


@dataclass
class CryptoData:
    """Real-time cryptocurrency data structure"""

    symbol: str
    current_price: float
    market_cap: float
    total_volume: float
    price_change_24h: float
    market_cap_rank: int


class DataFetcher:
    """Handles real-time data fetching from CoinGecko"""
```

```
def __init__(self):
    self.base_url = "https://api.coingecko.com/api/v3"
    self.session = None

async def _init_session(self):
    if self.session is None:
        self.session = aiohttp.ClientSession()

async def close(self):
    if self.session:
        await self.session.close()
        self.session = None

async def get_market_data(
    self, limit: int = 20
) -> List[CryptoData]:
    """Fetch market data for top cryptocurrencies"""
    await self._init_session()

    url = f"{self.base_url}/coins/markets"
    params = {
        "vs_currency": "usd",
        "order": "market_cap_desc",
        "per_page": str(limit),
        "page": "1",
        "sparkline": "false",
}


try:
    async with self.session.get(
        url, params=params
    ) as response:
        if response.status != 200:
            logger.error(
                f"API Error {response.status}: {await response.text()}"
            )
            return []

        data = await response.json()
        crypto_data = []

        for coin in data:
            try:
                crypto_data.append(
                    CryptoData(
                        symbol=str(
                            coin.get("symbol", "")
                        ).upper(),
                        current_price=float(
                            coin.get("current_price", 0)
                        ),
                        market_cap=float(
                            coin.get("market_cap", 0)
                        ),
                        total_volume=float(
                            coin.get("total_volume", 0)
                        ),
                        price_change_24h=float(
                            coin.get("price_change_24h", 0)
                        ),
                        market_cap_rank=int(
                            coin.get("market_cap_rank", 0)
                        ),
                    )
                )
            except (ValueError, TypeError) as e:
                logger.error(
                    f"Error processing coin data: {str(e)}"
                )
                continue

        logger.info(
            f"Successfully fetched data for {len(crypto_data)} coins"
        )
        return crypto_data

except Exception as e:
    logger.error(f"Exception in get_market_data: {str(e)}")
```

```
            return []


class CryptoSwarmSystem:
    def __init__(self):
        self.agents = self._initialize_agents()
        self.data_fetcher = DataFetcher()
        logger.info("Crypto Swarm System initialized")

    def _initialize_agents(self) -> Dict[str, Agent]:
        """Initialize different specialized agents"""
        base_config = {
            "max_loops": 1,
            "autosave": True,
            "dashboard": False,
            "verbose": True,
            "dynamic_temperature_enabled": True,
            "retry_attempts": 3,
            "context_length": 200000,
            "return_step_meta": False,
            "output_type": "string",
            "streaming_on": False,
        }

        agents = {
            "price_analyst": Agent(
            agent_name="Price-Analysis-Agent",
                system_prompt="""Analyze the given cryptocurrency price data and provide insights
about:
            1. Price trends and movements
            2. Notable price actions
            3. Potential support/resistance levels""",
            saved_state_path="price_agent.json",
            user_name="price_analyzer",
            **base_config,
        ),
        "volume_analyst": Agent(
            agent_name="Volume-Analysis-Agent",
              system_prompt="""Analyze the given cryptocurrency volume data and provide insights
about:
            1. Volume trends
            2. Notable volume spikes
            3. Market participation levels""",
            saved_state_path="volume_agent.json",
            user_name="volume_analyzer",
            **base_config,
        ),
        "market_analyst": Agent(
            agent_name="Market-Analysis-Agent",
              system_prompt="""Analyze the overall cryptocurrency market data and provide insights
about:
            1. Market trends
            2. Market dominance
            3. Notable market movements""",
            saved_state_path="market_agent.json",
            user_name="market_analyzer",
            **base_config,
        ),
    }
    return agents

async def analyze_market(self) -> Dict:
    """Run real-time market analysis using all agents"""
    try:
        # Fetch market data
        logger.info("Fetching market data for top 20 coins")
        crypto_data = await self.data_fetcher.get_market_data(20)

        if not crypto_data:
            return {
                "error": "Failed to fetch market data",
                "timestamp": datetime.now().isoformat(),
            }

        # Run analysis with each agent
        results = {}
        for agent_name, agent in self.agents.items():
            logger.info(f"Running {agent_name} analysis")
            analysis = self._run_agent_analysis(
                agent, crypto_data
            )
            results[agent_name] = analysis

        return {
            "timestamp": datetime.now().isoformat(),
            "market_data": {
                coin.symbol: {
                    "price": coin.current_price,
                    "market_cap": coin.market_cap,
                    "volume": coin.total_volume,
                    "price_change_24h": coin.price_change_24h,
                    "rank": coin.market_cap_rank,
                }
                for coin in crypto_data
            },
            "analysis": results,
        }

    except Exception as e:
        logger.error(f"Error in market analysis: {str(e)}")
        return {
            "error": str(e),
            "timestamp": datetime.now().isoformat(),
        }
```

```
def _run_agent_analysis(
    self, agent: Agent, crypto_data: List[CryptoData]
) -> str:
    """Run analysis for a single agent"""
    try:
        data_str = json.dumps(
            [
                {
                    "symbol": cd.symbol,
                    "price": cd.current_price,
                    "market_cap": cd.market_cap,
                    "volume": cd.total_volume,
                    "price_change_24h": cd.price_change_24h,
                    "rank": cd.market_cap_rank,
                }
                for cd in crypto_data
            ],
            indent=2,
        )

        prompt = f"""Analyze this real-time cryptocurrency market data and provide detailed insights:
        {data_str}"""

        return agent.run(prompt)
        except Exception as e:
            logger.error(f"Error in {agent.agent_name}: {str(e)}")
            return f"Error: {str(e)}"


async def main():
    # Create output directory
    Path("reports").mkdir(exist_ok=True)

    # Initialize the swarm system
    swarm = CryptoSwarmSystem()

    while True:
        try:
            # Run analysis
            report = await swarm.analyze_market()

            # Save report
            timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
            report_path = f"reports/market_analysis_{timestamp}.json"

            with open(report_path, "w") as f:
                json.dump(report, f, indent=2, default=str)

            logger.info(
                f"Analysis complete. Report saved to {report_path}"
            )


            # Wait before next analysis
            await asyncio.sleep(300)  # 5 minutes


        except Exception as e:
            logger.error(f"Error in main loop: {str(e)}")
            await asyncio.sleep(60)  # Wait 1 minute before retrying
        finally:
            if swarm.data_fetcher.session:
                await swarm.data_fetcher.close()



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