```
import os

import sys

import datetime

from typing import List, Dict, Any, Optional


from swarms import Agent

from swarm_models import OpenAIChat

from swarms.prompts.finance_agent_sys_prompt import (

    FINANCIAL_AGENT_SYS_PROMPT,

)


from pulsar import Client, Producer

from pydantic import BaseModel, Field

from loguru import logger


# Configure Loguru logger

logger.remove()

logger.add(sys.stderr, level="INFO")

logger.add("swarm_logs.log", rotation="10 MB", level="DEBUG")


# Apache Pulsar configuration

PULSAR_SERVICE_URL = os.getenv(

    "PULSAR_SERVICE_URL", "pulsar://localhost:6650"

)
```

```
# Define Pydantic schemas for structured output
class AgentOutputMetadata(BaseModel):
    agent_name: str
    task: str
    timestamp: datetime.datetime
    status: str


class AgentOutputData(BaseModel):
    output: str
    additional_info: Optional[Dict[str, Any]] = None


class AgentOutputSchema(BaseModel):
    metadata: AgentOutputMetadata
    data: AgentOutputData


class SwarmOutputSchema(BaseModel):
    results: List[AgentOutputSchema] = Field(default_factory=list)


# SwarmManager class to manage agents and tasks
class SwarmManager:
    def __init__(
        self,
        agents: List[Agent],

        pulsar_service_url: str = PULSAR_SERVICE_URL,

    ):

        """

        Initializes the SwarmManager with a list of agents and Pulsar service URL.



        :param agents: List of Agent instances.

        :param pulsar_service_url: URL of the Apache Pulsar service.

        """

        self.agents = agents

        self.pulsar_service_url = pulsar_service_url

        self.client: Optional[Client] = None

        self.producers: Dict[str, Producer] = {}

        self.swarm_results = SwarmOutputSchema()


    def connect_pulsar(self) -> None:

        """

        Establishes connection to the Apache Pulsar service.

        """

        try:

            self.client = Client(

                self.pulsar_service_url, operation_timeout_seconds=30

            )

            logger.info(

                f"Connected to Pulsar service at {self.pulsar_service_url}"

            )
            except Exception as e:
                logger.error(f"Failed to connect to Pulsar service: {e}")
                raise

        def initialize_producers(self) -> None:
            """
            Initializes Pulsar producers for each agent.
            """
            if not self.client:
                logger.error("Pulsar client is not connected.")
                raise ConnectionError("Pulsar client is not connected.")

            for agent in self.agents:
                try:
                    topic = f"{agent.agent_name}_topic"
                    producer = self.client.create_producer(topic)
                    self.producers[agent.agent_name] = producer
                    logger.debug(
                        f"Initialized producer for agent {agent.agent_name} on topic {topic}"
                    )
                except Exception as e:
                    logger.error(
                        f"Failed to create producer for agent {agent.agent_name}: {e}"
                    )
                    raise
```

```
def run_task(self, agent: Agent, task: str) -> AgentOutputSchema:
    """
    Executes a task using the specified agent and returns the structured output.

    :param agent: The Agent instance to execute the task.
    :param task: The task string to be executed.
    :return: AgentOutputSchema containing the result and metadata.
    """
    logger.info(
        f"Agent {agent.agent_name} is starting task: {task}"
    )
    timestamp = datetime.datetime.utcnow()

    try:
        output = agent.run(task)
        status = "Success"
        logger.info(
            f"Agent {agent.agent_name} completed task successfully."
        )
    except Exception as e:
        output = str(e)
        status = "Failed"
        logger.error(
            f"Agent {agent.agent_name} failed to complete task: {e}"
        )
```

```
metadata = AgentOutputMetadata(
    agent_name=agent.agent_name,
    task=task,
    timestamp=timestamp,
    status=status,
)


data = AgentOutputData(output=output)


agent_output = AgentOutputSchema(metadata=metadata, data=data)


# Publish result to Pulsar topic
try:
    producer = self.producers.get(agent.agent_name)
    if producer:
        producer.send(agent_output.json().encode("utf-8"))
        logger.debug(
            f"Published output for agent {agent.agent_name} to Pulsar topic."
        )
    else:
        logger.warning(
            f"No producer found for agent {agent.agent_name}. Skipping publish step."
        )
except Exception as e:
    logger.error(
        f"Failed to publish output for agent {agent.agent_name}: {e}"
```

```
            )

        return agent_output

    def run(self, task: str) -> SwarmOutputSchema:
        """
        Runs the swarm by executing the task across all agents sequentially and returns aggregated results.

        :param task: The task string to be executed by the swarm.
        :return: SwarmOutputSchema containing results from all agents.
        """
        try:
            self.connect_pulsar()
            self.initialize_producers()

            for agent in self.agents:
                result = self.run_task(agent, task)
                self.swarm_results.results.append(result)

            logger.info("Swarm run completed successfully.")
            return self.swarm_results

        except Exception as e:
            logger.error(f"Swarm run encountered an error: {e}")
            raise
        finally:
            if self.client:
                self.client.close()
                logger.info("Pulsar client connection closed.")


# Example usage
if __name__ == "__main__":
    # Initialize OpenAIChat model
    api_key = os.getenv("OPENAI_API_KEY")
    if not api_key:
        logger.error(
            "OPENAI_API_KEY environment variable is not set."
        )
        sys.exit(1)

    model = OpenAIChat(
        api_key=api_key, model_name="gpt-4", temperature=0.1
    )

    # Define agents
    agent1 = Agent(
        agent_name="Financial-Analysis-Agent",
        system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
        llm=model,
        max_loops=1,

        autosave=True,

        dashboard=False,

        verbose=True,

        dynamic_temperature_enabled=True,

        saved_state_path="finance_agent.json",

        user_name="swarms_corp",

        retry_attempts=1,

        context_length=2000,

        return_step_meta=False,

    )


    agent2 = Agent(

        agent_name="Market-Analysis-Agent",

        system_prompt=FINANCIAL_AGENT_SYS_PROMPT,

        llm=model,

        max_loops=1,

        autosave=True,

        dashboard=False,

        verbose=True,

        dynamic_temperature_enabled=True,

        saved_state_path="market_agent.json",

        user_name="swarms_corp",

        retry_attempts=1,

        context_length=2000,

        return_step_meta=False,
        )


        # Initialize and run swarm
        swarm = SwarmManager(agents=[agent1, agent2])
        task_description = "How can I establish a ROTH IRA to buy stocks and get a tax break? What are the criteria?"
        results = swarm.run(task_description)


        # Output results
        print(results.json(indent=4))
```