```
import asyncio
import os
import uuid
from concurrent.futures import ThreadPoolExecutor
from typing import List

from pydantic import BaseModel

from swarms import Agent
from swarm_models import OpenAIChat
from swarms.prompts.finance_agent_sys_prompt import (
    FINANCIAL_AGENT_SYS_PROMPT,
)
from swarms.utils.file_processing import create_file_in_folder

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

# Create an instance of the OpenAIChat class
model = OpenAIChat(
    api_key=api_key, model_name="gpt-4o-mini", temperature=0.1
)


# Pydantic schema for logging the responses
class AgentResponseLog(BaseModel):
```

```
    layer: int
    agent_name: str
    response: str


class FinalAgentResponse(BaseModel):
    layer: int
    agent_name: str
    response: str


class MixtureOfAgentsLog(BaseModel):
    run_id: str = uuid.uuid4().hex
    task: str
    logs: List[AgentResponseLog]
    final_agent_response: FinalAgentResponse
    total_layers: int = 3
    agents_per_layer: int = 3


# Define the function for the Mixture-of-Agents structure using async internally
def mixture_of_agents(
    agents: List[Agent],
    task: str,
    number_of_agents_per_layer: int = 3,
    save_file_name: str = "mixture_of_agents.json",
```

```
    autosave: bool = True,
    layers: int = 3,
    final_agent: Agent = None,
) -> dict:
    residual = task

    async def run_agents_sequentially(agents, input_task):
        results = []
        for agent in agents:
            result = await loop.run_in_executor(
                executor, agent.run, input_task
            )
            results.append(result)
            input_task += (
                f" Agent: {agent.agent_name}, Response: {result}"
            )
        return results

    loop = asyncio.new_event_loop()
    asyncio.set_event_loop(loop)
    logs = []

    with ThreadPoolExecutor(max_workers=os.cpu_count()) as executor:
        # Process each layer
        for layer in range(
            layers
```

```
):  # 3 layers as per original example
    current_agents = agents[
        layer
        * number_of_agents_per_layer : (layer + 1)
        * number_of_agents_per_layer
    ]
    output_layer = loop.run_until_complete(
        run_agents_sequentially(current_agents, task)
    )

    # Log responses
    for i, output in enumerate(output_layer):
        logs.append(
            AgentResponseLog(
                layer=layer + 1,
                agent_name=current_agents[i].agent_name,
                response=output,
            )
        )

    # Prepare task for the next layer by including all outputs
    task = " ".join(
        [
            f"Agent: {log.agent_name}, Response: {log.response}"
            for log in logs
        ]
```

```
        )

    # Now run the final agent after all layers are completed
    final_agent_input = task
    final_output = loop.run_until_complete(
        loop.run_in_executor(
            executor, final_agent.run, final_agent_input
        )
    )

loop.close()

# Create the log object and return as JSON
mixture_log = MixtureOfAgentsLog(
    task=residual,
    logs=logs,
    total_layers=layers,
    agents_per_layer=number_of_agents_per_layer,
    final_agent_response=FinalAgentResponse(
        layer=layers + 1,  # Last layer
        agent_name=final_agent.agent_name,
        response=final_output,
    ),
)

if autosave:
```

```
        create_file_in_folder(
            os.getenv("WORKSPACE_DIR"),
            save_file_name,
            mixture_log.model_dump_json(indent=4),
        )

    return mixture_log.model_dump_json(indent=4)



# Create a list of agents
agents = [
    Agent(
        agent_name=f"Agent_{i+1}",
        system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
        llm=model,
        max_loops=1,
        verbose=True,
        saved_state_path=f"Agent_{i+1}.json",
        user_name="swarms_corp",
        retry_attempts=1,
        context_length=200000,
    )
    for i in range(3)  # Adjusted to include all 9 agents
]

director_agent = Agent(
```

```
    agent_name="Final Agent",
    system_prompt=FINANCIAL_AGENT_SYS_PROMPT,
    llm=model,
    max_loops=1,
    verbose=True,
    user_name="swarms_corp",
    retry_attempts=1,
)

# Define the task and number of agents per layer
task = "How can I establish a ROTH IRA to buy stocks and get a tax break? What are the criteria?"
number_of_agents_per_layer = 3

# Run the Mixture-of-Agents structure and get the JSON log
final_result_json = mixture_of_agents(
    agents,
    task,
    number_of_agents_per_layer,
    final_agent=director_agent,
    layers=3,
)
print(final_result_json)
```