```
import os
import requests
from loguru import logger
from swarms import Agent
from swarm_models import OpenAIChat
from pydantic import BaseModel, Field
from typing import Optional
from datetime import datetime
from dotenv import load_dotenv

load_dotenv()

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

# Define the system prompt for the pharmaceutical analysis agent
PHARMACEUTICAL_AGENT_SYS_PROMPT = """
You are an expert pharmaceutical data analyst. Your task is to analyze chemical and protein data to provide detailed insights into their potential interactions and uses in drug development. Use the provided data and ensure your analysis is scientifically accurate, reliable, and considers potential side effects and clinical trials.

Always answer in a structured, detailed format. Consider the following information when analyzing:
- Chemical: {chemical_title}, Molecular Formula: {chemical_formula}
- Protein: {protein_name}, Function: {protein_function}
```

```
Your goal is to provide a comprehensive understanding of how these chemical compounds might interact with the protein and their potential use cases in medicine, considering real-world clinical scenarios.
"""



# Pydantic Model for chemical data
class ChemicalData(BaseModel):
    title: Optional[str] = Field(None, title="Chemical Title")
    molecular_formula: Optional[str] = Field(
        None, title="Molecular Formula"
    )
    isomeric_smiles: Optional[str] = Field(
        None, title="Isomeric SMILES"
    )



# Pydantic Model for protein data
class ProteinData(BaseModel):
    entry_name: Optional[str] = Field(
        None, title="Protein Entry Name"
    )
    function: Optional[str] = Field(None, title="Protein Function")



# Pydantic Model for the analysis output
```

```
class AnalysisOutput(BaseModel):
    analysis_id: str = Field(..., title="Unique Analysis ID")
    timestamp: str = Field(..., title="Timestamp of the analysis")
    chemical_data: Optional[ChemicalData] = Field(
        None, title="Chemical Data"
    )
    protein_data: Optional[ProteinData] = Field(
        None, title="Protein Data"
    )
    analysis_result: Optional[str] = Field(
        None, title="Result from the agent analysis"
    )



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


# Initialize the Swarms Agent
agent = Agent(
    agent_name="Pharmaceutical-Analysis-Agent",
    # system_prompt=PHARMACEUTICAL_AGENT_SYS_PROMPT,
    llm=model,
    max_loops=1,
    autosave=True,
    dashboard=False,
    verbose=True,
    dynamic_temperature_enabled=True,
    saved_state_path="pharmaceutical_agent.json",
    user_name="swarms_corp",
    retry_attempts=1,
    context_length=200000,
    return_step_meta=False,
)


class PharmaDataIntegration:
    def __init__(self):
        """
        Initializes the integration class for Swarms and public pharmaceutical APIs (PubChem, UniProt).
        """
        pass

    @logger.catch
    def fetch_chemical_data(self, compound_id: str) -> ChemicalData:
        """
        Fetch chemical data from the PubChem API based on compound ID. No API key is required.

        :param compound_id: The PubChem compound ID to fetch data for.
        :return: Pydantic model containing chemical data.
        """
                                                                            url                =
f"https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/cid/{compound_id}/property/Title,MolecularF
ormula,IsomericSMILES/JSON"
        logger.debug(
            f"Fetching chemical data for compound ID: {compound_id}"
        )
        response = requests.get(url)
        if response.status_code == 200:
            logger.info(
                f"Successfully fetched chemical data for compound ID: {compound_id}"
            )
            data = (
                response.json()
                .get("PropertyTable", {})
                .get("Properties", [{}])[0]
            )
            return ChemicalData(
                title=data.get("Title", "Unknown Chemical"),
                molecular_formula=data.get(
                    "MolecularFormula", "Unknown Formula"
                ),
                isomeric_smiles=data.get(
                    "IsomericSMILES", "Unknown SMILES"
                ),
            )
        else:
            logger.error(
                    f"Failed to fetch chemical data for compound ID: {compound_id}, Status Code:
{response.status_code}"
            )
            return ChemicalData()

    @logger.catch
    def fetch_protein_data(self, protein_id: str) -> ProteinData:
        """
        Fetch protein data from the UniProt API based on protein ID. No API key is required.

        :param protein_id: The UniProt protein ID to fetch data for.
        :return: Pydantic model containing protein data.
        """
        url = f"https://www.uniprot.org/uniprot/{protein_id}.json"
        logger.debug(
            f"Fetching protein data for protein ID: {protein_id}"
        )
        response = requests.get(url)
        if response.status_code == 200:
            logger.info(
                f"Successfully fetched protein data for protein ID: {protein_id}"
            )
            data = response.json()
            return ProteinData(
                entry_name=data.get("entryName", "Unknown Protein"),
                function=data.get("function", "Unknown Function"),
            )
        else:
            logger.error(
                f"Failed to fetch protein data for protein ID: {protein_id}, Status Code:
{response.status_code}"
            )
            return ProteinData()

    @logger.catch
    def analyze_data_with_swarms_agent(
        self,
        chemical_data: Optional[ChemicalData],
        protein_data: Optional[ProteinData],
    ) -> str:
        """
        Use the Swarms Agent to analyze the fetched chemical and protein data.

        :param chemical_data: Data fetched from PubChem about the chemical.
        :param protein_data: Data fetched from UniProt about the protein.
        :return: Analysis result from the Swarms Agent.
        """
        # Fill in the system prompt with the actual data
        agent_input = PHARMACEUTICAL_AGENT_SYS_PROMPT.format(
            chemical_title=(
                chemical_data.title if chemical_data else "Unknown"
            ),
            chemical_formula=(
                chemical_data.molecular_formula
                if chemical_data
                else "Unknown"
            ),
            protein_name=(
                protein_data.entry_name if protein_data else "Unknown"
            ),
            protein_function=(
                protein_data.function if protein_data else "Unknown"
            ),
        )

        logger.debug(
            "Running Swarms Agent with the provided chemical and protein data."
        )
        out = agent.run(agent_input)
        logger.info(f"Swarms Agent analysis result: {out}")
        return out

    @logger.catch
    def run(
        self,
        task: str,
        protein_id: Optional[str] = None,
        compound_id: Optional[str] = None,
        *args,
        **kwargs,
    ) -> AnalysisOutput:
        """
        The main method that dynamically handles task, protein, and chemical analysis.

        :param task: Natural language task that guides the analysis (e.g., "Analyze the effects of this protein").
        :param protein_id: (Optional) Protein ID from UniProt.
        :param compound_id: (Optional) Compound ID from PubChem.
        :return: JSON output with chemical, protein, and analysis data.
        """
        chemical_data = None
        protein_data = None

        # Dynamic task handling
        if "protein" in task.lower() and protein_id:
            logger.debug(f"Task is protein-related: {task}")
            protein_data = self.fetch_protein_data(protein_id)
            logger.info(protein_data)

        if "chemical" in task.lower() and compound_id:
            logger.debug(f"Task is chemical-related: {task}")
            chemical_data = self.fetch_chemical_data(compound_id)
        # Analyze data using the Swarms Agent
        analysis_result = self.analyze_data_with_swarms_agent(
            chemical_data, protein_data
        )

        # Create the output model
        output = AnalysisOutput(
            analysis_id=f"{compound_id or 'unknown'}-{protein_id or 'unknown'}",
            timestamp=datetime.utcnow().isoformat(),
            chemical_data=chemical_data,
            protein_data=protein_data,
            analysis_result=analysis_result,
        )

        # Log the JSON output
        # logger.info(f"Final analysis result as JSON: {output.json(indent=2)}")

        # Return the structured JSON output
        return output.model_dump_json(indent=4)


# Example usage:
if __name__ == "__main__":
    pharma_integration = PharmaDataIntegration()
```

```
# Example: Analyze the effects of a specific protein and chemical compound
result = pharma_integration.run(
    task="Analyze this compound and provide an analysis",
    # protein_id="P12345",
    compound_id="19833",
)

# Print the result in JSON format
print(result)
```