```
import os

from dotenv import load_dotenv
from termcolor import colored

from swarm_models import OpenAIChat
from swarms.prompts.code_interpreter import CODE_INTERPRETER
from swarms.prompts.programming import DOCUMENTATION_SOP, TEST_SOP
from swarms.structs import Agent

load_dotenv()



FEATURE = (
    "Implement an all-new signup system in typescript using supabase"
)

CODEBASE = """
import React, { useState } from 'react';
import UpperPanel from './UpperPanel';
import LowerPanel from './LowerPanel';

const MainPanel = () => {
  const [promptInstructionForLowerPanel, setPromptInstructionForLowerPanel] = useState('');
  const [formData, setFormData] = useState('');
  const [isLoading, setIsLoading] = useState(false);
```

```
  return (
    <div className="flex h-screen">
      <UpperPanel setPromptInstructionForLowerPanel={setPromptInstructionForLowerPanel}
      isLoading={isLoading}
      setIsLoading={setIsLoading}
      />
      <LowerPanel promptInstruction={promptInstructionForLowerPanel} isLoading={isLoading} />
    </div>
  );
};

export default MainPanel;
```

"""

```
# Load the environment variables
api_key = os.getenv("OPENAI_API_KEY")

# Initialize the language agent
llm = OpenAIChat(
    model_name="gpt-4",
    openai_api_key=api_key,
    temperature=0.5,
    max_tokens=4000,
```

```
)

# Product Manager Agent init
product_manager_agent = Agent(
    llm=llm, max_loops=1, sop=CODE_INTERPRETER, autosave=True
)

# Initialize the agent with the language agent
feature_implementer_frontend = Agent(
    llm=llm, max_loops=1, sop=CODE_INTERPRETER, autosave=True
)

# Create another agent for a different task
feature_implementer_backend = Agent(
    llm=llm, max_loops=1, sop=CODE_INTERPRETER, autosave=True
)

# Create another agent for a different task
tester_agent = Agent(
    llm=llm, max_loops=1, sop=TEST_SOP, autosave=True
)

# Create another agent for a different task
documenting_agent = Agent(
    llm=llm, max_loops=1, sop=DOCUMENTATION_SOP, autosave=True
)
```

```
# Product Agent prompt
def feature_codebase_product_agentprompt(
    feature: str, codebase: str
) -> str:
    prompt = (
        "Create an algorithmic pseudocode for an all-new feature:"
        f" {feature} based on this codebase: {codebase}"
    )
    return prompt



# Product Manager Agent
product_manager_out = product_manager_agent.run(
    feature_codebase_product_agentprompt(FEATURE, CODEBASE)
)
print(
    colored(
        (
            "---------------------------- Product Manager Plan:"
            f" {product_manager_out}"
        ),
        "cyan",
    )
)
```

```
# Feature Implementer  Agent
agent1_out = feature_implementer_frontend.run(
    f"Create the backend code for {FEATURE} in markdown based off of"
    f" this algorithmic pseudocode: {product_manager_out} the logic"
    f" based on the following codebase: {CODEBASE}"
)
print(
    colored(
        (
            "---------------------- Feature Implementer Code logic:"
            f" {agent1_out}"
        ),
        "cyan",
    )
)

# Tester agent
tester_agent_out = tester_agent.run(
    f"Create tests for the following code: {agent1_out}"
)
print(
    colored(
        (
            "---------------------------- Tests for the logic:"
            f" {tester_agent_out}"
```

```
        ),
        "green",
    )
)



# Documentation Agent
documenter_agent_out = documenting_agent.run(
    f"Document the following code: {agent1_out}"
)
print(
    colored(
        (
            "---------------------------- Documentation for the"
            f" logic: {documenter_agent_out}"
        ),
        "yellow",
    )
)
```