```
import inspect
import os
import threading
from typing import Callable, List

from swarms.prompts.documentation import DOCUMENTATION_WRITER_SOP
from swarms import Agent
from swarm_models import OpenAIChat
from swarms.utils.loguru_logger import logger
import concurrent

#########
from swarms.utils.file_processing import (
    load_json,
    sanitize_file_path,
    zip_workspace,
    create_file_in_folder,
    zip_folders,
)


class PythonDocumentationSwarm:
    """
    A class for automating the documentation process for Python classes.

    Args:
        agents (List[Agent]): A list of agents used for processing the documentation.
        max_loops (int, optional): The maximum number of loops to run. Defaults to 4.
        docs_module_name (str, optional): The name of the module where the documentation will be saved. Defaults to "swarms.structs".
        docs_directory (str, optional): The directory where the documentation will be saved. Defaults to "docs/swarms/tokenizers".

    Attributes:
        agents (List[Agent]): A list of agents used for processing the documentation.
        max_loops (int): The maximum number of loops to run.
        docs_module_name (str): The name of the module where the documentation will be saved.
        docs_directory (str): The directory where the documentation will be saved.
    """

    def __init__(
        self,
        agents: List[Agent],
        max_loops: int = 4,
        docs_module_name: str = "swarms.utils",
        docs_directory: str = "docs/swarms/utils",
        *args,
        **kwargs,
    ):
        super().__init__(*args, **kwargs)
        self.agents = agents
        self.max_loops = max_loops
        self.docs_module_name = docs_module_name
        self.docs_directory = docs_directory

        # Initialize agent name logging
        logger.info(
            "Agents used for documentation:"
            f" {', '.join([agent.name for agent in agents])}"
        )

        # Create the directory if it doesn't exist
        dir_path = self.docs_directory
        os.makedirs(dir_path, exist_ok=True)
        logger.info(f"Documentation directory created at {dir_path}.")

    def process_documentation(self, item):
        """
        Process the documentation for a given class using OpenAI model and save it in a Markdown file.

        Args:
            item: The class or function for which the documentation needs to be processed.
        """
        try:
            doc = inspect.getdoc(item)
            source = inspect.getsource(item)
            is_class = inspect.isclass(item)
            item_type = "Class Name" if is_class else "Name"
            input_content = (
                f"{item_type}:"
                f" {item.__name__}\n\nDocumentation:\n{doc}\n\nSource"
                f" Code:\n{source}"
            )

              # Process with OpenAI model (assuming the model's __call__ method takes this input and
returns processed content)
            for agent in self.agents:
                processed_content = agent(
                    DOCUMENTATION_WRITER_SOP(
                        input_content, self.docs_module_name
                    )
                )

            doc_content = f"{processed_content}\n"

            # Create the directory if it doesn't exist
            dir_path = self.docs_directory
            os.makedirs(dir_path, exist_ok=True)

            # Write the processed documentation to a Markdown file
            file_path = os.path.join(
                dir_path, f"{item.__name__.lower()}.md"
            )
            with open(file_path, "w") as file:
                file.write(doc_content)

            logger.info(
                f"Documentation generated for {item.__name__}."
            )
        except Exception as e:
            logger.error(
                f"Error processing documentation for {item.__name__}."
            )
            logger.error(e)

    def run(self, python_items: List[Callable]):
        """
        Run the documentation process for a list of Python items.

        Args:
                python_items (List[Callable]): A list of Python classes or functions for which the
documentation needs to be generated.
        """
        try:
            threads = []
            for item in python_items:
                thread = threading.Thread(
                    target=self.process_documentation, args=(item,)
                )
                threads.append(thread)
                thread.start()

            # Wait for all threads to complete
            for thread in threads:
                thread.join()

            logger.info(
                "Documentation generated in 'swarms.structs'"
                " directory."
            )
        except Exception as e:
            logger.error("Error running documentation process.")
            logger.error(e)

    def run_concurrently(self, python_items: List[Callable]):
        try:
            with concurrent.futures.ThreadPoolExecutor() as executor:
                executor.map(self.process_documentation, python_items)

            logger.info(
                "Documentation generated in 'swarms.structs'"
                " directory."
            )
        except Exception as e:
            logger.error("Error running documentation process.")
            logger.error(e)



# Example usage
# Initialize the agents
agent = Agent(
    llm=OpenAIChat(max_tokens=3000),
    agent_name="Documentation Agent",
    system_prompt=(
        "You write documentation for Python items functions and"
        " classes, return in markdown"
    ),
    max_loops=1,
)

# Initialize the documentation swarm
doc_swarm = PythonDocumentationSwarm(
    agents=[agent],
    max_loops=1,
    docs_module_name="swarms.structs",
    docs_directory="docs/swarms/tokenizers",
)

# Run the documentation process
doc_swarm.run(
    [
        load_json,
        sanitize_file_path,
        zip_workspace,
        create_file_in_folder,
        zip_folders,
    ]
)
```