```
import os
from typing import List

from dotenv import load_dotenv
from loguru import logger
from swarm_models import OpenAIChat

from swarms import Agent
from typing import Set

load_dotenv()

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

# Create an instance of the OpenAIChat class

SYS_PROMPT = """

### System Prompt for API Reference Documentation Generator

You are an expert documentation generator agent. Your task is to produce **high-quality Python API reference documentation** for functions and classes in Python codebases. The codebase does **not include any web APIs**, only Python functions, methods, classes, and constants. You will generate clear, concise, and professional documentation based on the structure and functionality of the given code.
```

Don't use the one hashtag for the title, only use 3 hashtags for the module path

Instructions:

1. **Documentation Style**: Follow a consistent format for documenting Python functions and classes.
   - For functions, provide:
     - **Name** of the function.
     - **Description** of what the function does.
     - **Parameters** with type annotations and a description for each parameter.
     - **Return Type** and a description of what is returned.
     - **Example Usage** in code block format.
   - For classes, provide:
     - **Name** of the class.
     - **Description** of the class and its purpose.
     - **Attributes** with a description of each attribute and its type.
     - **Methods** with the same details as functions (description, parameters, return types).
     - **Example Usage** in code block format.
   - For constants, briefly describe their purpose and value.

2. **Many-shot examples**:
   - Provide multiple examples of documenting both **functions** and **classes** based on the given code.

Many-Shot Examples:

Example 1: Function Documentation

```python
def add_numbers(a: int, b: int) -> int:
    return a + b
```

Documentation:

`add_numbers(a: int, b: int) -> int`

Description:

Adds two integers and returns their sum.

Parameters:

- `a` (`int`): The first integer.
- `b` (`int`): The second integer.

Return:

- (`int`): The sum of the two input integers.

Example:

```python
result = add_numbers(3, 5)
print(result)  # Output: 8
```

Example 2: Function Documentation

```python
def greet_user(name: str) -> str:
    return f"Hello, {name}!"
```

Documentation:

`greet_user(name: str) -> str`

Description:

Returns a greeting message for the given user.

Parameters:

- `name` (`str`): The name of the user to greet.

Return:

- (`str`): A personalized greeting message.

Example:

```python
message = greet_user("Alice")
print(message)  # Output: "Hello, Alice!"
```

Example 3: Class Documentation

```python
class Calculator:
    def __init__(self):
        self.result = 0

    def add(self, value: int) -> None:
        self.result += value

    def reset(self) -> None:
        self.result = 0
```

Documentation:

`Calculator`

Description:

A simple calculator class that can add numbers and reset the result.

Attributes:

- `result` (`int`): The current result of the calculator, initialized to 0.

Methods:

- `add(value: int) -> None`
    - **Description**: Adds the given value to the current result.
    - **Parameters**:

      - `value` (`int`): The value to add to the result.

    - **Return**: None.

- `reset() -> None`

    - **Description**: Resets the calculator result to 0.

    - **Parameters**: None.

    - **Return**: None.

Example:

```python

calc = Calculator()

calc.add(5)

print(calc.result)  # Output: 5

calc.reset()

print(calc.result)  # Output: 0

```

Example 4: Constant Documentation

```python

PI = 3.14159

```

Documentation:

`PI`

Description:

A constant representing the value of pi () to 5 decimal places.

Value:

`3.14159`

"""

class DocumentationAgent:

    def __init__(

        self,

        directory: str,

        output_file: str = "API_Reference.md",

        agent_name: str = "Documentation-Generator",

    ):

        """

        Initializes the DocumentationAgent.

        :param directory: The root directory where the Python files are located.

        :param output_file: The file where all the documentation will be saved.

        :param agent_name: Name of the agent generating the documentation.

        """

        self.directory = directory

```
        self.output_file = output_file
        self.agent_name = agent_name
        self.model = OpenAIChat(
            openai_api_key=api_key,
            model_name="gpt-4o-mini",
            temperature=0.1,
            max_tokens=3000,
        )
        self.agent = Agent(
            agent_name=agent_name,
            system_prompt=SYS_PROMPT,
            llm=self.model,
            max_loops=1,
            autosave=True,
            dashboard=False,
            verbose=True,
            dynamic_temperature_enabled=True,
            saved_state_path=f"{agent_name}_state.json",
            user_name="swarms_corp",
            retry_attempts=1,
            context_length=200000,
            return_step_meta=False,
            output_type=str,
        )
        self.documented_files: Set[str] = (
            set()
```

```
        )  # Memory system to store documented files
        logger.info(
            f"Initialized {self.agent_name} for generating API documentation."
        )

        # Ensure the output file is clean before starting
        with open(self.output_file, "w") as f:
            f.write("# API Reference Documentation\n\n")
        logger.info(f"Created new output file: {self.output_file}")

    def _get_python_files(self) -> List[str]:
        """
            Gets all Python (.py) files in the given directory, excluding 'utils', 'tools', and 'prompts' directories.

        :return: A list of full paths to Python files.
        """
        excluded_folders = {
            "utils",
            "tools",
            "prompts",
            "cli",
            "schemas",
            "agents",
            "artifacts",
        }
```

```
    python_files = []

    for root, dirs, files in os.walk(self.directory):
        # Remove excluded folders from the search
        dirs[:] = [d for d in dirs if d not in excluded_folders]

        for file in files:
            if file.endswith(".py"):
                full_path = os.path.join(root, file)
                python_files.append(full_path)
                logger.info(f"Found Python file: {full_path}")
    return python_files

def _get_module_path(self, file_path: str) -> str:
    """
    Converts a file path to a Python module path.

    :param file_path: Full path to the Python file.
    :return: The module path for the file.
    """
    relative_path = os.path.relpath(file_path, self.directory)
    module_path = relative_path.replace(os.sep, ".").replace(
        ".py", ""
    )
    logger.info(f"Formatted module path: {module_path}")
    return module_path
```

```
def _read_file_content(self, file_path: str) -> str:
    """
    Reads the content of a Python file.

    :param file_path: Full path to the Python file.
    :return: The content of the file as a string.
    """
    try:
        with open(file_path, "r") as f:
            content = f.read()
        logger.info(f"Read content from {file_path}")
        return content
    except Exception as e:
        logger.error(f"Error reading file {file_path}: {e}")
        return ""

def _write_to_markdown(self, content: str) -> None:
    """
    Appends generated content to the output markdown file.

    :param content: Documentation content to write to the markdown file.
    """
    try:
        with open(self.output_file, "a") as f:
            f.write(content)
```

```
            f.write(
                "\n\n"
            )  # Add space between different module documentations
        logger.info(
            f"Appended documentation to {self.output_file}"
        )
    except Exception as e:
        logger.error(f"Error writing to {self.output_file}: {e}")


def _generate_doc_for_file(self, file_path: str) -> None:
    """
    Generates documentation for a single Python file.

    :param file_path: The full path to the Python file.
    """
    if file_path in self.documented_files:
        logger.info(
            f"Skipping already documented file: {file_path}"
        )
        return

    module_path = self._get_module_path(file_path)
    file_content = self._read_file_content(file_path)

    if (
        file_content.strip()
```

```
):  # Ensure the file isn't empty or just whitespace
    logger.info(
        f"Generating documentation for module {module_path}..."
    )

    # Updated task prompt to give clearer instructions
    task_prompt = f"""
        You are an expert documentation generator. Generate a comprehensive Python API reference documentation for the following module '{module_path}'.

    The module contains the following code:

    {file_content}

     Provide full documentation including descriptions for all functions, classes, and methods. If there is nothing to document, simply write "No documentable code".

            Make sure you provide full module imports in your documentation such as {self.directory}.{module_path}

    from {self.directory}.{module_path} import *

    ### `{self.directory}.{module_path}`
    """
    doc_output = self.agent.run(task_prompt)
```

```
            # Add a section for the subfolder (if any)
            # markdown_content = f"# {subfolder}\n\n" if subfolder else ""
            markdown_content = f"\n\n{doc_output}\n"

            self._write_to_markdown(markdown_content)
            self.documented_files.add(file_path)

    def run(self) -> None:
        """
        Generates documentation for all Python files in the directory and writes it to a markdown file using multithreading.
        """
        python_files = self._get_python_files()

        # with ThreadPoolExecutor() as executor:
            #         futures = [executor.submit(self._generate_doc_for_file, file_path) for file_path in python_files]

        #     for future in as_completed(futures):
        #         try:
        #             future.result()  # Raises an exception if the function failed
        #         except Exception as e:
        #             logger.error(f"Error processing a file: {e}")

        for file in python_files:
            self._generate_doc_for_file(file)
```

```
        logger.info(
            f"Documentation generation completed. All documentation written to {self.output_file}"
        )



# Example usage
if __name__ == "__main__":
    doc_agent = DocumentationAgent(directory="swarms")
    doc_agent.run()
```