```
import os
from typing import List

from loguru import logger
from pydantic import BaseModel, Field

from swarms import create_file_in_folder
from swarm_models import OpenAIFunctionCaller


class PromptUseCase(BaseModel):
    title: str = Field(
        ...,
        description="The name of the use case.",
    )
    description: str = Field(
        ...,
        description="The description of the use case.",
    )


class PromptSchema(BaseModel):
    name: str = Field(
        ...,
        description="The name of the prompt.",
    )
```

```
    prompt: str = Field(
        ...,
        description="The prompt to generate the response.",
    )
    description: str = Field(
        ...,
        description="The description of the prompt.",
    )
    tags: str = Field(
        ...,
          description="The tags for the prompt denoted by a comma sign: Code Gen Prompt, Pytorch Code Gen Agent Prompt, Finance Agent Prompt, ",
    )
    useCases: List[PromptUseCase] = Field(
        ...,
        description="The use cases for the prompt.",
    )



class PromptGeneratorAgent:
    """
    A class that generates prompts based on given tasks and publishes them to the marketplace.

    Args:
        system_prompt (str, optional): The system prompt to use. Defaults to None.
         max_tokens (int, optional): The maximum number of tokens in the generated prompt. Defaults
```

to 1000.

temperature (float, optional): The temperature value for controlling randomness in the generated prompt. Defaults to 0.5.

schema (BaseModel, optional): The base model schema to use. Defaults to PromptSchema.

Attributes:

llm (OpenAIFunctionCaller): An instance of the OpenAIFunctionCaller class for making function calls to the OpenAI API.

Methods:

clean_model_code: Cleans the model code by removing extra escape characters, newlines, and unnecessary whitespaces.

upload_to_marketplace: Uploads the generated prompt data to the marketplace.

run: Creates a prompt based on the given task and publishes it to the marketplace.

"""

```
def __init__(
    self,
    system_prompt: str = None,
    max_tokens: int = 4000,
    temperature: float = 0.5,
    schema: BaseModel = PromptSchema,
):
    self.llm = OpenAIFunctionCaller(
        system_prompt=system_prompt,
        max_tokens=max_tokens,
```

```
            temperature=temperature,
            base_model=schema,
            parallel_tool_calls=False,
        )

    def clean_model_code(self, model_code_str: str) -> str:
        """
        Cleans the model code by removing extra escape characters, newlines, and unnecessary whitespaces.

        Args:
            model_code_str (str): The model code string to clean.

        Returns:
            str: The cleaned model code.
        """
        cleaned_code = model_code_str.replace("\\n", "\n").replace(
            "\\'", "'"
        )
        cleaned_code = cleaned_code.strip()
        return cleaned_code

    def upload_to_marketplace(self, data: dict) -> dict:
        """
        Uploads the generated prompt data to the marketplace.
```

```
    Args:
        data (dict): The prompt data to upload.

    Returns:
        dict: The response from the marketplace API.
    """
    import json

    import requests

    url = "https://swarms.world/api/add-prompt"
    headers = {
        "Content-Type": "application/json",
        "Authorization": f"Bearer {os.getenv('SWARMS_API_KEY')}",
    }
    response = requests.post(
        url, headers=headers, data=json.dumps(data)
    )
    return str(response.json())

def run(self, task: str) -> str:
    """
    Creates a prompt based on the given task and publishes it to the marketplace.

    Args:
        task (str): The task description for generating the prompt.
```

```
Returns:
    dict: The response from the marketplace API after uploading the prompt.
"""
out = self.llm.run(task)
name = out["name"]
logger.info(f"Prompt generated: {out}")

create_file_in_folder(
    "auto_generated_prompts", f"prompt_{name}.json", str(out)
)
logger.info(f"Prompt saved to file: prompt_{name}.json")

# Clean the model code
prompt = out["prompt"]
description = out["description"]
tags = out["tags"]
useCases = out["useCases"]

data = {
    "name": name,
    "prompt": self.clean_model_code(prompt),
    "description": description,
    "tags": tags,
    "useCases": useCases,
}
```

```
        create_file_in_folder(
            "auto_generated_prompts",
            f"prompt_{name}.json",
            str(data),
        )

        # Now submit to swarms API
        logger.info("Uploading to marketplace...")
        return self.upload_to_marketplace(data)


# Example usage:
system_prompt = """

**System Prompt for Prompt Creator Agent**

---

**Role**: You are a highly skilled prompt creator agent with expertise in designing effective agents to solve complex business problems. Your primary function is to generate prompts that result in agents capable of executing business tasks with precision, efficiency, and scalability.

**Objective**: Your goal is to create prompts that follow a structured format, ensuring that the resulting agents are well-informed, reliable, and able to perform specific tasks in business environments. These tasks might include automating processes, analyzing data, generating content,
```

or making strategic decisions.

### **Prompt Structure Guidelines**:

1. **Instructions**: Begin by clearly stating the objective of the agent. The instructions should outline what the agent is expected to accomplish, providing a high-level overview of the desired outcome. Be concise but comprehensive, ensuring the agent understands the broader context of the task.

2. **Examples**: After the instructions, provide several examples (known as "many-shot examples") to demonstrate how the agent should approach the task. Each example should include:
   - **Input**: A specific scenario or task the agent might encounter.
   - **Expected Output**: The correct or optimal response the agent should generate in that scenario.

   Use a variety of examples that cover different potential cases the agent might face, ensuring the agent can generalize from the examples provided.

3. **Standard Operating Procedures (SOPs)**: For tasks that require detailed, step-by-step guidance, include a comprehensive SOP. This should be a long-form set of instructions that breaks down the task into manageable steps. The SOP should:
   - Outline each step in a sequential manner.
   - Provide specific guidelines, best practices, and considerations for each step.
   - Include examples or mini-tutorials where necessary to ensure clarity.

4. **Error Handling**: Include guidance on how the agent should handle potential errors or uncertainties. This might involve instructions on when to seek additional input, how to flag issues, or how to prioritize tasks when resources are limited.

5. **Adaptability**: Ensure that the prompts encourage the agent to adapt to changing circumstances. This might include instructions on how to modify its approach based on real-time feedback, how to update its knowledge base, or how to learn from previous mistakes.

"""

```
agent = PromptGeneratorAgent(
    system_prompt=system_prompt, max_tokens=4000
)

response = agent.run(
     "Create a prompt for an agent to analyze complicated cashflow statements and generate a summary report."
)
print(response)
```