```
from typing import List, Dict, Any
import uuid
from pydantic import BaseModel
from typing import Optional
import time

# ID
id = str(uuid.uuid4())



class InputOpenAISpec(BaseModel):
    """OpenAI Spec for the model"""

    model: Optional[str] = "gpt-3.5-turbo"
    max_new_tokens: Optional[int] = 100
    prompt: Optional[str] = ""
    stream: Optional[bool] = False
    sampling_params: Optional[Dict[str, Any]] = None
    best_of: Optional[int] = 1
    echo: Optional[bool] = False
    frequency_penalty: Optional[float] = 0.0
    logit_bias: Optional[Dict[str, Any]] = None
    logprobs: Optional[int] = None
    max_tokens: Optional[int] = None
    n: Optional[int] = 1
    presence_penalty: Optional[float] = 0.0
```

```
    seed: Optional[int] = None
    stop: Optional[str] = None
    suffix: Optional[str] = None
    temperature: Optional[float] = 0.0
    top_k: Optional[int] = 0
    top_p: Optional[float] = 1.0
    user: Optional[str] = None


class OutputOpenAISpec(BaseModel):
    """OpenAI Spec for the model"""

    id: Optional[str] = id
    created: Optional[int] = int(time.time())
    object: Optional[str] = None
    system_fingerpoint: Optional[str] = None
    model: Optional[str] = "gpt-3.5-turbo"
    max_new_tokens: Optional[int] = 100
    prompt: Optional[str] = ""
    stream: Optional[bool] = False
    sampling_params: Optional[Dict[str, Any]] = None
    best_of: Optional[int] = 1
    echo: Optional[bool] = False
    frequency_penalty: Optional[float] = 0.0
    logit_bias: Optional[Dict[str, Any]] = None
    logprobs: Optional[int] = None
```

max_tokens: Optional[int] = None

n: Optional[int] = 1

presence_penalty: Optional[float] = 0.0

seed: Optional[int] = None

stop: Optional[str] = None

suffix: Optional[str] = None

temperature: Optional[float] = 0.0

top_k: Optional[int] = 0

top_p: Optional[float] = 1.0

user: Optional[str] = None

usage: Optional[Dict[str, Any]] = None

completion_tokens: Optional[int] = None

prompt_tokens: Optional[int] = None

total_tokens: Optional[int] = None

choices: Optional[List[Dict[str, Any]]] = None

finish_reason: Optional[str] = None

index: Optional[int] = None

logprobs: Optional[Dict[str, Any]] = None

text: Optional[str] = None

conversation_id: Optional[str] = None

response_id: Optional[str] = None

timestamp: Optional[str] = None

status: Optional[str] = None

error: Optional[str] = None

error_message: Optional[str] = None

error_code: Optional[int] = None

```
    error_details: Optional[str] = None
    error_traceback: Optional[str] = None
    error_cause: Optional[str] = None
    error_type: Optional[str] = None
    error_context: Optional[str] = None


class OpenAIAPIWrapper:
    """
    A wrapper class for the OpenAI API.

    This class provides methods to set and get the input and output specifications for the OpenAI API.
    """

    def __init__(self):
        self.input_spec = InputOpenAISpec()
        self.output_spec = OutputOpenAISpec()

    def set_input_spec(self, **kwargs):
        """
        Set the input specification for the OpenAI API.

        Args:
            **kwargs: Keyword arguments representing the input specification attributes and their values.
```

```
        """
        for key, value in kwargs.items():
            if hasattr(self.input_spec, key):
                setattr(self.input_spec, key, value)

    def set_output_spec(self, **kwargs):
        """
        Set the output specification for the OpenAI API.

        Args:
            **kwargs: Keyword arguments representing the output specification attributes and their values.
        """
        for key, value in kwargs.items():
            if hasattr(self.output_spec, key):
                setattr(self.output_spec, key, value)

    def get_input_spec(self):
        """
        Get the input specification for the OpenAI API.

        Returns:
            str: JSON representation of the input specification.
        """
        return self.input_spec.json()
```

```
def get_output_spec(self):
    """
    Get the output specification for the OpenAI API.


    Returns:
        str: JSON representation of the output specification.
    """
    return self.output_spec.json()
```