```
from loguru import logger
import subprocess
from pydantic import BaseModel, Field, ValidationError
from typing import List, Optional, Union



try:
    import ollama
except ImportError:
    logger.error("Failed to import ollama")
    subprocess.run(["pip", "install", "ollama"])
    import ollama



class Message(BaseModel):
    role: str = Field(
        ...,
        pattern="^(user|system|assistant)$",
        description="The role of the message sender.",
    )
    content: str = Field(
        ..., min_length=1, description="The content of the message."
    )



class OllamaModel:
```

```
def __init__(
    self,
    model_name: str,
    host: Optional[str] = None,
    timeout: int = 30,
    stream: bool = False,
):
    """
    Initializes the OllamaModel with the model name and optional parameters.

    Args:
        model_name (str): The name of the model to interact with (e.g., 'llama3.1').
        host (str, optional): The Ollama host to connect to. Defaults to None.
        timeout (int, optional): Timeout for the requests. Defaults to 30 seconds.
        stream (bool, optional): Enable streaming for responses. Defaults to False.
    """
    self.model_name = model_name
    self.host = host
    self.timeout = timeout
    self.stream = stream

    self.client = ollama.Client(host=host) if host else None

def validate_messages(
    self, messages: List[Message]
) -> List[dict]:
```

```
    """
    Validates the list of messages using Pydantic schema.

    Args:
        messages (List[Message]): List of messages to validate.

    Returns:
        List[dict]: Validated messages in dictionary format.
    """
    try:
        return [message.dict() for message in messages]
    except ValidationError as e:
        print(f"Validation error: {e}")
        return []

def chat(
    self, messages: List[Message], *args, **kwargs
) -> Union[str, None]:
    """Executes the chat task."""
    validated_messages = self.validate_messages(messages)
    if not validated_messages:
        return None

    if self.stream:
        stream = ollama.chat(
            model=self.model_name,
```

```
            messages=validated_messages,
            stream=True,
            *args,
            **kwargs,
        )
        for chunk in stream:
            print(chunk["message"]["content"], end="", flush=True)
    else:
        response = ollama.chat(
            model=self.model_name, messages=validated_messages
        )
        return response["message"]["content"]

def generate(self, prompt: str) -> Optional[str]:
    """Generates text based on a prompt."""
    if len(prompt) == 0:
        print("Prompt cannot be empty.")
        return None

    response = ollama.generate(
        model=self.model_name, prompt=prompt
    )
    return response.get("message", {}).get("content", None)

def list_models(self) -> List[str]:
    """Lists available models."""
```

```
    return ollama.list()

def show_model(self) -> dict:
    """Shows details of the current model."""
    return ollama.show(self.model_name)

def create_model(self, modelfile: str) -> dict:
    """Creates a new model from a modelfile."""
    return ollama.create(
        model=self.model_name, modelfile=modelfile
    )

def delete_model(self) -> bool:
    """Deletes the current model."""
    try:
        ollama.delete(self.model_name)
        return True
    except ollama.ResponseError as e:
        print(f"Error deleting model: {e}")
        return False

def run(self, task: str, *args, **kwargs):
    """
    Executes the task based on the task string.

    Args:
```

```
    task (str): The task to execute, such as 'chat', 'generate', etc.
"""
return ollama.generate(
    model=self.model_name, prompt=task, *args, **kwargs
)
```