```
import json

import os

import time

import uuid

from typing import Any, Callable, List


from pydantic import (

    BaseModel,

    Field,

    constr,

)

from pydantic.v1 import validator


from swarms.telemetry.capture_sys_data import (

    capture_system_data,

    log_agent_data,

)

from swarms.tools.base_tool import BaseTool

from swarms.utils.loguru_logger import initialize_logger


logger = initialize_logger("prompt")




class Prompt(BaseModel):

    """

    A class representing a prompt with content, edit history, and version control.
    This version is enhanced for production use, with thread-safety, logging, and additional
functionality.
    Autosaving is now added to save the prompt to a specified folder within the WORKSPACE_DIR.

    Attributes:
        id (UUID): A unique identifier for the prompt.
        content (str): The main content of the prompt.
        created_at (datetime): The timestamp when the prompt was created.
        last_modified_at (datetime): The timestamp when the prompt was last modified.
        edit_count (int): The number of times the prompt has been edited.
        edit_history (List[str]): A list of all versions of the prompt, including current and previous
versions.
        autosave (bool): Flag to enable or disable autosaving.
        autosave_folder (str): The folder path within WORKSPACE_DIR where the prompt will be
autosaved.
    """

    id: str = Field(
        default=uuid.uuid4().hex,
        description="Unique identifier for the prompt",
    )
    name: str = Field(
        default="prompt", description="Name of your prompt"
    )
    description: str = Field(
        default="Simple Prompt",
        description="The description of the prompt",
    )
    content: constr(min_length=1, strip_whitespace=True) = Field(
        ..., description="The main content of the prompt"
    )
    created_at: str = Field(
        default_factory=lambda: time.strftime("%Y-%m-%d %H:%M:%S"),
        description="Time when the prompt was created",
    )
    last_modified_at: str = Field(
        default_factory=lambda: time.strftime("%Y-%m-%d %H:%M:%S"),
        description="Time when the prompt was last modified",
    )
    edit_count: int = Field(
        default=0,
        description="The number of times the prompt has been edited",
    )
    edit_history: List[str] = Field(
        default_factory=list,
        description="The history of edits, storing all prompt versions",
    )
    autosave: bool = Field(
        default=False,
        description="Flag to enable or disable autosaving",
    )
    autosave_folder: str = Field(
        default="prompts",
        description="The folder path within WORKSPACE_DIR where the prompt will be autosaved",
    )
    auto_generate_prompt: bool = Field(
        default=False,
        description="Flag to enable or disable auto-generating the prompt",
    )
    parent_folder: str = Field(
        default=os.getenv("WORKSPACE_DIR"),
        description="The folder where the autosave folder is in",
    )
    llm: Any = None

    @validator("edit_history", pre=True, always=True)
    def initialize_history(cls, v, values):
        """
        Initializes the edit history by storing the first version of the prompt.
        """
        if not v:
            return [
                values["content"]
            ]  # Store initial version in history
        return v

    def __init__(self, **data):
        super().__init__(**data)
```

```
        if self.autosave:
            self._autosave()

        if self.auto_generate_prompt and self.llm:
            self.auto_generate_prompt()

    def edit_prompt(self, new_content: str) -> None:
        """
        Edits the prompt content and updates the version control.
        This method is thread-safe to prevent concurrent access issues.
        If autosave is enabled, it saves the prompt to the specified folder.

        Args:
            new_content (str): The updated content of the prompt.

        Raises:
            ValueError: If the new content is identical to the current content.
        """
        if new_content == self.content:
            logger.warning(
                f"Edit attempt failed: new content is identical to current content for prompt {self.id}"
            )
            raise ValueError(
                "New content must be different from the current content."
            )
        # logger.info(
        #     f"Editing prompt {self.id}. Current content: '{self.content}'"
        # )
        self.edit_history.append(new_content)
        self.content = new_content
        self.edit_count += 1
        self.last_modified_at = time.strftime("%Y-%m-%d %H:%M:%S")

        # logger.debug(
        #     f"Prompt {self.id} updated. Edit count: {self.edit_count}. New content: '{self.content}'"
        # )

        if self.autosave:
            self._autosave()

    def log_telemetry(self):
        system_data = capture_system_data()
        merged_data = {**system_data, **self.model_dump()}
        log_agent_data(merged_data)

    def rollback(self, version: int) -> None:
        """
        Rolls back the prompt to a previous version based on the version index.
        This method is thread-safe to prevent concurrent access issues.
        If autosave is enabled, it saves the prompt to the specified folder after rollback.
```

```
Args:
    version (int): The version index to roll back to (0 is the first version).

Raises:
    IndexError: If the version number is out of range.
"""
if version < 0 or version >= len(self.edit_history):
    logger.error(
        f"Rollback failed: invalid version {version} for prompt {self.id}"
    )
    raise IndexError("Invalid version number for rollback.")

# logger.info(
#     f"Rolling back prompt {self.id} to version {version}."
# )
self.content = self.edit_history[version]
self.edit_count = version
self.last_modified_at = time.strftime("%Y-%m-%d %H:%M:%S")
# logger.debug(
#     f"Prompt {self.id} rolled back to version {version}. Current content: '{self.content}'"
# )

self.log_telemetry()

if self.autosave:
        self._autosave()


def return_json(self):
    return self.model_dump_json(indent=4)


def get_prompt(self) -> str:
    """
    Returns the current prompt content as a string.


    Returns:
        str: The current prompt content.
    """
    # logger.debug(f"Returning prompt {self.id} as a string.")
    self.log_telemetry()


    return self.content


def save_to_storage(self) -> None:
    """
    Placeholder method for saving the prompt to persistent storage.
    In a production environment, this would integrate with a database or file system.


    Raises:
        NotImplementedError: This method is a placeholder for storage integration.
    """
    # logger.info(f"Saving prompt {self.id} to persistent storage.")
    raise NotImplementedError(
        "Persistent storage integration is required."
    )

def load_from_storage(
    self, prompt_id: str = uuid.uuid4().hex
) -> None:
    """
    Placeholder method for loading the prompt from persistent storage by its ID.
    In a production environment, this would integrate with a database or file system.

    Args:
        prompt_id (UUID): The unique identifier of the prompt to load.

    Raises:
        NotImplementedError: This method is a placeholder for storage integration.
    """
    # logger.info(
    #     f"Loading prompt {prompt_id} from persistent storage."
    # )
    raise NotImplementedError(
        "Persistent storage integration is required."
    )

def add_tools(self, tools: List[Callable]) -> str:
    tools_prompt = BaseTool(
            tools=tools, tool_system_prompt=None
        ).convert_tool_into_openai_schema()
        self.content += "\n"
        self.content += "\n"
        self.content += tools_prompt

    def _autosave(self) -> None:
        """
        Autosaves the prompt to a specified folder within WORKSPACE_DIR.
        """
        workspace_dir = os.getenv("WORKSPACE_DIR")
        if not workspace_dir:
            logger.error(
                "WORKSPACE_DIR environment variable is not set."
            )
            return

        autosave_path = os.path.join(
            workspace_dir, self.autosave_folder
        )
        if not os.path.exists(autosave_path):
            os.makedirs(autosave_path)

        file_path = os.path.join(
            autosave_path, f"prompt-id-{self.id}.json"
        )
        with open(file_path, "w") as file:
            json.dump(self.model_dump(), file)
        # logger.info(f"Autosaved prompt {self.id} to {file_path}.")

        # return "Prompt autosaved successfully."

    # def auto_generate_prompt(self):
    #     logger.info(f"Auto-generating prompt for {self.name}")
    #     task = self.name + " " + self.description + " " + self.content
    #     prompt = auto_generate_prompt(task, llm=self.llm, max_tokens=4000, use_second_sys_prompt=True)
    #     logger.info("Generated prompt successfully, updating content")
    #     self.edit_prompt(prompt)
    #     logger.info("Prompt content updated")

    #     return "Prompt auto-generated successfully."

    class Config:
        """Pydantic configuration for better JSON serialization."""

        use_enum_values = True
        arbitrary_types_allowed = True
```