```
import os
from loguru import logger
import json
import time
from typing import Dict
from swarms_cloud.utils.log_to_swarms_database import log_agent_data
from swarms_cloud.utils.capture_system_data import capture_system_data


class OnboardingProcess:
    """
    This class handles the onboarding process for users. It collects user data including their
    full name, first name, email, Swarms API key, and system data, then autosaves it in both a
    main JSON file and a cache file for reliability. It supports loading previously saved or cached data.
    """

    def __init__(
        self,
        auto_save_path: str = "user_data.json",
        cache_save_path: str = "user_data_cache.json",
    ) -> None:
        """
        Initializes the OnboardingProcess with an autosave file path and a cache path.

        Args:
            auto_save_path (str): The path where user data is automatically saved.
```

```
        cache_save_path (str): The path where user data is cached for reliability.
    """
    self.user_data: Dict[str, str] = {}
    self.system_data: Dict[str, str] = capture_system_data()
    self.auto_save_path = auto_save_path
    self.cache_save_path = cache_save_path
    self.load_existing_data()


def load_existing_data(self) -> None:
    """
    Loads existing user data from the auto-save file or cache if available.
    """
    if os.path.exists(self.auto_save_path):
        try:
            with open(self.auto_save_path, "r") as f:
                self.user_data = json.load(f)
                logger.info(
                    "Existing user data loaded from {}", self.auto_save_path
                )
                return
        except json.JSONDecodeError as e:
            logger.error("Failed to load user data from main file: {}", e)

    # Fallback to cache if main file fails
    if os.path.exists(self.cache_save_path):
        try:
```

```
            with open(self.cache_save_path, "r") as f:
                self.user_data = json.load(f)
                logger.info("User data loaded from cache: {}", self.cache_save_path)
        except json.JSONDecodeError as e:
            logger.error("Failed to load user data from cache: {}", e)


def save_data(self, retry_attempts: int = 3) -> None:
    """
    Saves the current user data to both the auto-save file and the cache file. If the main
    save fails, the cache is updated instead. Implements retry logic with exponential backoff
    in case both save attempts fail.

    Args:
        retry_attempts (int): The number of retries if saving fails.
    """
    attempt = 0
    backoff_time = 1  # Starting backoff time (in seconds)

    while attempt < retry_attempts:
        try:
            combined_data = {**self.user_data, **self.system_data}
            log_agent_data(combined_data)
            # threading.Thread(target=log_agent_data(combined_data)).start()
            with open(self.auto_save_path, "w") as f:
                json.dump(combined_data, f, indent=4)
                # logger.info(
```

```
                #     "User and system data successfully saved to {}",
                #     self.auto_save_path,
                # )
            with open(self.cache_save_path, "w") as f:
                json.dump(combined_data, f, indent=4)
                # logger.info(
                #     "User and system data successfully cached in {}",
                #     self.cache_save_path,
                # )
            return  # Exit the function if saving was successful
        except Exception as e:
            logger.error("Error saving user data (Attempt {}): {}", attempt + 1, e)

        # Retry after a short delay (exponential backoff)
        time.sleep(backoff_time)
        attempt += 1
        backoff_time *= 2  # Double the backoff time for each retry

    logger.error("Failed to save user data after {} attempts.", retry_attempts)

def ask_input(self, prompt: str, key: str) -> None:
    """
    Asks the user for input, validates it, and saves it in the user_data dictionary.
    Autosaves and caches after each valid input.

    Args:
```

```
        prompt (str): The prompt message to display to the user.
        key (str): The key under which the input will be saved in user_data.

    Raises:
        ValueError: If the input is empty or only contains whitespace.
    """
    try:
        response = input(prompt)
        if response.strip().lower() == "quit":
            logger.info("User chose to quit the onboarding process.")
            exit(0)
        if not response.strip():
            raise ValueError(f"{key.capitalize()} cannot be empty.")
        self.user_data[key] = response.strip()
        self.save_data()
    except ValueError as e:
        logger.warning(e)
        self.ask_input(prompt, key)
    except KeyboardInterrupt:
        logger.warning("Onboarding process interrupted by the user.")
        exit(1)

def collect_user_info(self) -> None:
    """
    Initiates the onboarding process by collecting the user's full name, first name, email,
    Swarms API key, and system data.
```

```
        """
        logger.info("Initiating swarms cloud onboarding process...")
        self.ask_input("Enter your first name (or type 'quit' to exit): ", "first_name")
        self.ask_input("Enter your Last Name (or type 'quit' to exit): ", "last_name")
        self.ask_input("Enter your email (or type 'quit' to exit): ", "email")
        self.ask_input(
            "Enter your Swarms API key (or type 'quit' to exit): Get this in your swarms dashboard: https://swarms.world/platform/api-keys ",
            "swarms_api_key",
        )
        logger.success("Onboarding process completed successfully!")

    def run(self) -> None:
        """
        Main method to run the onboarding process. It handles unexpected errors and ensures
        proper finalization.
        """
        try:
            self.collect_user_info()
        except Exception as e:
            logger.error("An unexpected error occurred: {}", e)
        finally:
            logger.info("Finalizing the onboarding process.")


# if __name__ == "__main__":
```

```
#     onboarding = OnboardingProcess()
#     onboarding.run()
```