voice_assistant.py

```python
import os
import logging
import time
import speech_recognition as sr
import pyttsx3
from command_processor import CommandProcessor

class VoiceAssistant:
    """
    Voice Assistant class that handles speech recognition,
    command processing, and response generation.
    """

    def __init__(self):
        """Initialize the voice assistant with necessary components."""
        self.logger = logging.getLogger(__name__)

        # Initialize the speech recognizer
        self.recognizer = sr.Recognizer()
        self.recognizer.energy_threshold = 4000
        self.recognizer.dynamic_energy_threshold = True

        # Initialize the text-to-speech engine
        self.engine = pyttsx3.init()
        self.engine.setProperty('rate', 150)  # Speed of speech

        # Initialize the command processor
        self.command_processor = CommandProcessor()

        # Internal state tracking
        self.is_listening = False
        self.command_history = []

        self.logger.debug("Voice Assistant initialized")

    def start_listening(self):
        """Begin the listening process."""
        self.is_listening = True
        self.logger.debug("Listening started")

    def stop_listening(self):
        """Stop the listening process."""
        self.is_listening = False
        self.logger.debug("Listening stopped")

    def process_audio(self, audio_data):
        """
        Process audio data to extract and execute commands.
```

```
        Args:
            audio_data: The audio data to process

        Returns:
            dict: The result of the command execution
        """
        try:
            # Convert audio data to text using speech recognition
            text = self.recognizer.recognize_google(audio_data)
            self.logger.debug(f"Recognized text: {text}")

            # Process the command
            return self.process_text_command(text)

        except sr.UnknownValueError:
            message = "Sorry, I couldn't understand the audio"
            self.speak(message)
            return {"command": None, "status": "error", "message": message}

        except sr.RequestError as e:
            message = f"Could not request results; {e}"
            self.speak(message)
            return {"command": None, "status": "error", "message": message}

        except Exception as e:
            message = f"Error processing audio: {e}"
            self.speak(message)
            self.logger.error(message)
            return {"command": None, "status": "error", "message": message}

    def process_text_command(self, text):
        """
        Process a text command to extract and execute commands.

        Args:
            text: The text command to process

        Returns:
            dict: The result of the command execution
        """
        try:
            # Use the command processor to understand and execute the command
            result = self.command_processor.process_command(text)

            # Store the command in history
            self.command_history.append({
                "timestamp": time.time(),
                "command": text,
                "result": result
```

```
            })

            # Generate and speak the response
            if result["status"] == "success":
                response = f"I've {result['action']} as requested."
                self.speak(response)
            else:
                response = f"Sorry, I couldn't {result['action']}. {result['message']}"
                self.speak(response)

            return {
                "command": text,
                "status": result["status"],
                "message": response,
                "action": result["action"],
                "details": result.get("details", {})
            }

        except Exception as e:
            message = f"Error processing command: {e}"
            self.speak(message)
            self.logger.error(message)
            return {"command": text, "status": "error", "message": message}

    def speak(self, text):
        """
        Convert text to speech and speak it.

        Args:
            text: The text to speak
        """
        self.logger.debug(f"Speaking: {text}")
        self.engine.say(text)
        self.engine.runAndWait()

    def get_command_history(self):
        """
        Get the history of commands processed by the assistant.

        Returns:
            list: The command history
        """
        return self.command_history

    def recognize_speech_from_mic(self, microphone, timeout=5):
        """
        Transcribe speech from recorded from `microphone`.

        Args:
            microphone: A microphone instance from the speech_recognition module
```

```
            timeout: Maximum number of seconds to listen for

        Returns:
            dict: A dictionary with transcription results
        """
        if not self.is_listening:
            return {"success": False, "error": "Assistant is not in listening mode"}

        # Check that recognizer and microphone arguments are appropriate type
        if not isinstance(microphone, sr.Microphone):
            raise TypeError("`microphone` must be a Microphone instance")

        # Adjust for ambient noise and record audio from the microphone
        with microphone as source:
            self.logger.debug("Adjusting for ambient noise")
            self.recognizer.adjust_for_ambient_noise(source)
            self.logger.debug("Listening for speech")
            try:
                audio = self.recognizer.listen(source, timeout=timeout)
            except sr.WaitTimeoutError:
                return {"success": False, "error": "Listening timed out"}

        # Try recognizing the speech in the recording
        try:
            self.logger.debug("Recognizing speech")
            text = self.recognizer.recognize_google(audio)
            return {"success": True, "text": text}
        except sr.UnknownValueError:
            return {"success": False, "error": "Speech was unintelligible"}
        except sr.RequestError as e:
            return {"success": False, "error": f"API unavailable: {e}"}
```