Project Name:

Voice-Recognition-for-Smart-Assistant

Code:

```
import speech_recognition as sr
import pyttsx3
import datetime
import webbrowser
import os
import subprocess
import requests
import json
import pyjokes
import pyperclip
import wikipedia
import random
from time import sleep
import threading
import time
import psutil
import win32com.client
import schedule
import sys
from pathlib import Path
```

```
# Add system tray support
try:
    import pystray
    from PIL import Image, ImageDraw
    TRAY_AVAILABLE = True
except ImportError:
    TRAY_AVAILABLE = False
    print("pystray not available. Install with: pip install pystray pillow")

class VoiceAssistant:
    def __init__(self):
        try:
            # Initialize text-to-speech engine
            self.engine = pyttsx3.init()

            # Initialize speech recognizer
            self.recognizer = sr.Recognizer()

            # Set microphone as source
            self.microphone = sr.Microphone()

            # Configure recognizer settings
            self.recognizer.energy_threshold = 4000
            self.recognizer.dynamic_energy_threshold = True
            self.recognizer.pause_threshold = 0.8
```

```
            self.wake_word = "assistant"
            self.exit_word = "goodbye"
            self.is_listening = False
            self.is_running = True
            self.background_mode = True

            self.quotes = [
                "The only way to do great work is to love what you do. - Steve Jobs",
                "Innovation distinguishes between a leader and a follower. - Steve Jobs",
                "Stay hungry, stay foolish. - Steve Jobs",
                "The future belongs to those who believe in the beauty of their dreams. - Eleanor Roosevelt",
                "Success is not final, failure is not fatal: it is the courage to continue that counts. - Winston Churchill"
            ]

            # Create log file
            self.log_file = Path("voice_assistant.log")
            self.log("Voice Assistant initialized successfully")

        except Exception as e:
            self.log(f"Error initializing Voice Assistant: {e}")
            raise

    def log(self, message):
        """Log messages to file and console"""
```

```
        timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        log_message = f"[{timestamp}] {message}"
        print(log_message)

        try:
            with open(self.log_file, "a", encoding='utf-8') as f:
                f.write(log_message + "\n")
        except:
            pass

    def create_tray_icon(self):
        """Create system tray icon"""
        if not TRAY_AVAILABLE:
            return None

        # Create a simple icon
        image = Image.new('RGB', (64, 64), color='blue')
        dc = ImageDraw.Draw(image)
        dc.ellipse([16, 16, 48, 48], fill='white')

        # Create menu
        menu = pystray.Menu(
            pystray.MenuItem("Status: Running", lambda: None, enabled=False),
            pystray.MenuItem("Show Log", self.show_log),
            pystray.MenuItem("Restart", self.restart),
            pystray.MenuItem("Exit", self.stop)
```

```
        )

        return pystray.Icon("Voice Assistant", image, "Voice Assistant", menu)

    def show_log(self, icon=None, item=None):
        """Show the log file"""
        try:
            if self.log_file.exists():
                os.startfile(str(self.log_file))
            else:
                self.log("Log file not found")
        except Exception as e:
            self.log(f"Error opening log: {e}")

    def restart(self, icon=None, item=None):
        """Restart the assistant"""
        self.log("Restarting voice assistant...")
        self.is_running = False
        if icon:
            icon.stop()
        # Restart the program
        python = sys.executable
        os.execl(python, python, *sys.argv)

    def stop(self, icon=None, item=None):
        """Stop the assistant"""
```

```
        self.log("Stopping voice assistant...")
        self.is_running = False
        if icon:
            icon.stop()

    def speak(self, text):
        """Convert text to speech"""
        try:
            self.log(f"Speaking: {text}")
            self.engine.say(text)
            self.engine.runAndWait()
        except Exception as e:
            self.log(f"Error in speech: {e}")

    def boot_up(self):
        """Display boot-up message"""
        current_time = datetime.datetime.now()
        hour = current_time.hour

        user_name = os.getenv('USERNAME', 'User')

        boot_message = f"Voice Assistant started in background mode"

        if 5 <= hour < 12:
            boot_message += " - Good morning!"
        elif 12 <= hour < 17:
```

```
            boot_message += " - Good afternoon!"
        elif 17 <= hour < 21:
            boot_message += " - Good evening!"
        else:
            boot_message += " - Good night!"

        self.log(boot_message)
        self.speak("Voice assistant is running in background. Say assistant to wake me up.")

    def get_weather(self, city):
        """Get current weather for a city"""
        try:
            API_KEY = "44bb1c57f29e742dbced10cab66b7b87"
            url = f"http://api.openweathermap.org/data/2.5/weather?q={city}&appid={API_KEY}&units=metric"
            response = requests.get(url)
            data = response.json()
            if data["cod"] == 200:
                temp = data["main"]["temp"]
                desc = data["weather"][0]["description"]
                return f"The temperature in {city} is {temp}°C with {desc}"
            return "Sorry, I couldn't get the weather information."
        except Exception as e:
            return "Sorry, there was an error getting the weather information."
```

```
    def play_music(self):
        """Play music from a default directory"""
        music_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)),
"assets", "music")
        if os.path.exists(music_dir):
            songs = os.listdir(music_dir)
            if songs:
                os.startfile(os.path.join(music_dir, random.choice(songs)))
                return "Playing music"
        return "No music files found"


    def set_alarm(self, time_str):
        """Set an alarm for the specified time"""
        try:
            try:
                alarm_time = datetime.datetime.strptime(time_str, "%H:%M").time()
            except ValueError:
                return "Please provide time in HH:MM format (e.g., 14:30 for 2:30
PM)"

            current_time = datetime.datetime.now().time()

            if alarm_time < current_time:
                return "Please set an alarm for a future time"

            current_datetime = datetime.datetime.now()
```

```
        alarm_datetime = datetime.datetime.combine(current_datetime.date(),
alarm_time)
        if alarm_datetime < current_datetime:
            alarm_datetime += datetime.timedelta(days=1)

        time_until_alarm = alarm_datetime - current_datetime
        hours, remainder = divmod(time_until_alarm.seconds, 3600)
        minutes, _ = divmod(remainder, 60)

        def alarm_ring():
            try:
                while True:
                    current = datetime.datetime.now().time()
                    if current.hour == alarm_time.hour and current.minute ==
alarm_time.minute:
                        for _ in range(3):
                            self.speak("Alarm! Wake up!")
                            time.sleep(1)
                        break
                    time.sleep(5)
            except Exception as e:
                self.log(f"Error in alarm thread: {str(e)}")

        alarm_thread = threading.Thread(target=alarm_ring, daemon=True)
        alarm_thread.start()

        if hours > 0:
```

```
            return f"Alarm set for {time_str}. It will ring in {hours} hours and {minutes} minutes."
        else:
            return f"Alarm set for {time_str}. It will ring in {minutes} minutes."

    except Exception as e:
        self.log(f"Error setting alarm: {str(e)}")
        return "Sorry, I couldn't set the alarm"

def take_note(self, note):
    """Save a note to a file"""
    try:
        notes_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), "notes")
        os.makedirs(notes_dir, exist_ok=True)

        notes_file = os.path.join(notes_dir, "notes.txt")

        with open(notes_file, "a", encoding='utf-8') as f:
            timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
            f.write(f"[{timestamp}] {note}\n")

        return f"Note saved successfully"
    except Exception as e:
        self.log(f"Error saving note: {str(e)}")
        return "Sorry, I couldn't save the note"
```

```
    def set_reminder(self, task, time_str):
        """Set a reminder for a specific task"""
        try:
            reminders_dir = os.path.join(os.path.dirname(os.path.abspath(__file__)), "reminders")
            os.makedirs(reminders_dir, exist_ok=True)

            try:
                reminder_time = datetime.datetime.strptime(time_str, "%H:%M").time()
            except ValueError:
                return "Please provide time in HH:MM format (e.g., 14:30 for 2:30 PM)"

            reminders_file = os.path.join(reminders_dir, "reminders.txt")

            with open(reminders_file, "a", encoding='utf-8') as f:
                timestamp = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
                f.write(f"[{timestamp}] Reminder: {task} at {time_str}\n")

            def check_reminders():
                try:
                    while True:
                        current_time = datetime.datetime.now().time()
                        if current_time.hour == reminder_time.hour and current_time.minute == reminder_time.minute:
```

```
                    self.speak(f"Reminder: {task}")
                    break
                time.sleep(10)
            except Exception as e:
                self.log(f"Error in reminder thread: {str(e)}")

        reminder_thread = threading.Thread(target=check_reminders, daemon=True)
        reminder_thread.start()

        return f"Reminder set for {task} at {time_str}"
    except Exception as e:
        self.log(f"Error setting reminder: {str(e)}")
        return "Sorry, I couldn't set the reminder"

def read_clipboard(self):
    """Read the contents of the clipboard"""
    try:
        clipboard_content = pyperclip.paste()

        if not clipboard_content:
            return "The clipboard is empty"

        if len(clipboard_content) > 200:
            return f"Clipboard content (truncated): {clipboard_content[:200]}..."

        return f"Clipboard content: {clipboard_content}"
```

```
        except Exception as e:
            self.log(f"Error reading clipboard: {str(e)}")
            return "Sorry, I couldn't read the clipboard"

    def open_application(self, app_name):
        """Open various applications"""
        apps = {
            "youtube": "https://www.youtube.com",
            "google": "https://www.google.com",
            "calculator": "calc.exe",
            "notepad": "notepad.exe",
            "visual studio code": "code.exe"
        }

        if app_name.lower() in apps:
            if app_name.lower() in ["youtube", "google"]:
                webbrowser.open(apps[app_name.lower()])
            else:
                subprocess.Popen(apps[app_name.lower()])
            return f"Opening {app_name}"
        return "Application not found"

    def search_wikipedia(self, query):
        """Search Wikipedia for information"""
        try:
            result = wikipedia.summary(query, sentences=2)
```

```
            return result
        except:
            return "Sorry, I couldn't find that information"

    def tell_joke(self):
        """Tell a random joke"""
        return pyjokes.get_joke()

    def tell_quote(self):
        """Tell a random quote"""
        return random.choice(self.quotes)

    def get_feeling(self):
        """Tell current feeling based on system resources"""
        cpu_percent = psutil.cpu_percent()
        memory_percent = psutil.virtual_memory().percent

        if cpu_percent > 80 or memory_percent > 80:
            return "I'm feeling a bit stressed, the system is under heavy load"
        elif cpu_percent > 50 or memory_percent > 50:
            return "I'm feeling okay, but the system is moderately busy"
        else:
            return "I'm feeling great! The system is running smoothly"

    def greet(self):
        """Greet the user based on time of day"""
```

```
        hour = datetime.datetime.now().hour
        user_name = os.getenv('USERNAME', 'User')

        if 5 <= hour < 12:
            return f"Good morning {user_name}! How can I help you today?"
        elif 12 <= hour < 17:
            return f"Good afternoon {user_name}! What can I do for you?"
        elif 17 <= hour < 21:
            return f"Good evening {user_name}! How may I assist you?"
        else:
            return f"Good night {user_name}! What can I help you with?"

    def listen(self):
        """Listen for voice commands with improved error handling and feedback"""
        try:
            with self.microphone as source:
                self.recognizer.adjust_for_ambient_noise(source, duration=0.5)

                try:
                    audio = self.recognizer.listen(source, timeout=3, phrase_time_limit=3)

                    try:
                        text = self.recognizer.recognize_google(audio)
                        self.log(f"Raw recognized text: '{text}'")
                        return text.lower().strip()
```

```
                except sr.UnknownValueError:
                    return ""
                except sr.RequestError as e:
                    self.log(f"Speech recognition service error: {e}")
                    return ""

            except sr.WaitTimeoutError:
                return ""

    except Exception as e:
        self.log(f"Error during listening: {e}")
        return ""

def process_command(self, command):
    """Process voice commands with improved response handling"""
    try:
        if not command:
            return "I didn't catch that. Could you please repeat?"

        command = command.lower().strip()
        self.log(f"Processing command: '{command}'")

        if self.exit_word in command:
            self.is_running = False
            return "Goodbye! Have a great day!"
```

```
        if "weather" in command:
            try:
                city = command.split("weather in ")[-1].strip()
                return self.get_weather(city)
            except:
                return "Please specify a city for weather information"

        elif "play music" in command:
            return self.play_music()

        elif "set alarm" in command:
            try:
                time_str = command.split("set alarm for ")[-1].strip()
                time_str = time_str.split()[0]
                self.log(f"Setting alarm for: {time_str}")
                response = self.set_alarm(time_str)
                return response
            except Exception as e:
                self.log(f"Error setting alarm: {str(e)}")
                return "Please specify the time in HH:MM format (e.g., set alarm for
14:30)"

        elif "take note" in command:
            try:
                note = command.split("take note ")[-1].strip()
                return self.take_note(note)
            except:
```

```
            return "Please specify what note to take"

        elif "set reminder" in command:
            try:
                reminder_text = command.split("set reminder ")[-1].strip()
                if " at " not in reminder_text:
                    return "Please specify the reminder and time (e.g., 'set reminder buy groceries at 14:30')"

                task, time_str = reminder_text.split(" at ")
                task = task.strip()
                time_str = time_str.strip()
                return self.set_reminder(task, time_str)
            except Exception as e:
                self.log(f"Error setting reminder: {str(e)}")
                return "Please specify the reminder and time correctly"

        elif "read clipboard" in command:
            return self.read_clipboard()

        elif "open" in command:
            try:
                app = command.split("open ")[-1].strip()
                return self.open_application(app)
            except:
                return "Please specify which application to open"
```

```
        elif "search" in command:
            try:
                query = command.split("search ")[-1].strip()
                return self.search_wikipedia(query)
            except:
                return "Please specify what to search for"

        elif "tell joke" in command:
            return self.tell_joke()

        elif "tell quote" in command:
            return self.tell_quote()

        elif "how are you" in command:
            return self.get_feeling()

        elif "hello" in command or "hi" in command:
            return self.greet()

        else:
            return "I didn't understand that command. Please try again."

    except Exception as e:
        self.log(f"Error processing command: {str(e)}")
        return "Sorry, I encountered an error processing your command"
```

```
def run_background(self):
    """Run the assistant in background mode"""
    try:
        self.boot_up()

        # Create system tray icon
        icon = self.create_tray_icon()

        # Start background listening thread
        def background_listener():
            while self.is_running:
                try:
                    command = self.listen()

                    if command and self.wake_word in command:
                        self.log("Wake word detected!")
                        self.is_listening = True
                        self.speak("Yes, how can I help you?")

                        while self.is_listening and self.is_running:
                            command = self.listen()

                            if command:
                                response = self.process_command(command)
                                self.speak(response)
```

```
                        if not self.is_running:
                            break
                        self.is_listening = False

                except Exception as e:
                    self.log(f"Error in background listener: {str(e)}")
                    time.sleep(1)

        # Start background thread
        listener_thread = threading.Thread(target=background_listener,
daemon=True)
        listener_thread.start()

        # Run system tray
        if icon:
            icon.run()
        else:
            # If no tray, just keep running
            while self.is_running:
                time.sleep(1)

    except Exception as e:
        self.log(f"Critical error in background run: {str(e)}")
    finally:
        self.log("Voice Assistant stopped")

if __name__ == "__main__":
```

```
try:
    assistant = VoiceAssistant()
    assistant.run_background()
except Exception as e:
    print(f"Fatal error: {e}")
    input("Press Enter to exit...")
```

--- 😊 ---