Membuat File Aplikasi Voicebot Prosa terintegrasi Mycroft Precise pada Ubuntu 16.04 LTS

Note : Metode pemasangan berlaku umum sehingga dapat diterapkan pada OS Ubuntu berbagai arsitektur termasuk Ubuntu Arm dan Ubuntu Aarch64

1. Pastikan bahwa terminal sedang berjalan dalam Virtual Environment Python:

```
isro@isro-vmware:~/qcom_emulator$ source env/bin/activate
(env) isro@isro-vmware:~/qcom_emulator$
```

2. Buat folder baru:

```
$ mkdir prosa-voicebot
$ cd prosa-voicebot
```

3. Buat script Python (terlampir):

```
$ nano voicebot-mycroft.py
```

4. Buat file spec pyinstaller (terlampir):

```
$ nano voicebot-mycroft.spec
```

5. Build aplikasi Voicebot Prosa:

```
$ pyinstaller --clean -y voicebot-mycroft.spec
```

```
(env) isro@isro-vmware:~/qcom_emulator/prosa-voicebot$ ls dist/voicebot-mycroft/
audioop.cpython-35m-x86_64-linux-gnu.so
base_library.zip
_bz2.cpython-35m-x86_64-linux-gnu.so
certifi
_cffi_backend.cpython-35m-x86_64-linux-gnu.so
_codecs_cn.cpython-35m-x86_64-linux-gnu.so
_codecs_hk.cpython-35m-x86_64-linux-gnu.so
_codecs_iso2022.cpython-35m-x86_64-linux-gnu.so
_codecs_jp.cpython-35m-x86_64-linux-gnu.so
_codecs_kr.cpython-35m-x86_64-linux-gnu.so
_codecs_tw.cpython-35m-x86_64-linux-gnu.so
cryptography
cryptography-3.2.1-py3.5.egg-info
_ctypes.cpython-35m-x86_64-linux-gnu.so
_decimal.cpython-35m-x86_64-linux-gnu.so
_hashlib.cpython-35m-x86_64-linux-gnu.so
include
```

Perhatikan bahwa hasil build tersimpan pada folder voicebot-mycroft di dalam folder dist. Salin satu folder voicebot-mycroft secara utuh untuk dapat menjalankan aplikasi voicebot-mycroft di dalam folder tersebut.

Script Python Aplikasi Voicebot Prosa terintegrasi Mycroft Precise

```
#!/usr/bin/env python3

from precise_runner import PreciseEngine, PreciseRunner
from urllib.request import urlopen
from subprocess import call
import json
import logging
import math
import audioop
import collections
import pyaudio
import requests
import wave
import io
import sys

waiting = 1

class AudioSource(object):
    def __init__(self):
        raise NotImplementedError("this is an abstract class")

    def __enter__(self):
        raise NotImplementedError("this is an abstract class")

    def __exit__(self, exc_type, exc_value, traceback):
        raise NotImplementedError("this is an abstract class")

class AudioData(object):
    def __init__(self, frame_data, sample_rate, sample_width):
        assert sample_rate > 0
        assert sample_width % 1 == 0 and 1 <= sample_width <= 4
        self.frame_data = frame_data
        self.sample_rate = sample_rate
        self.sample_width = int(sample_width)

    def get_raw_data(self, convert_rate=None, convert_width=None):
        assert convert_rate is None or convert_rate > 0
        assert convert_width is None or (convert_width % 1 == 0 and 1 <=
convert_width <= 4)
        raw_data = self.frame_data
        if self.sample_width == 1:
            raw_data = audioop.bias(raw_data, 1, -128)
        if convert_rate is not None and self.sample_rate != convert_rate:
            raw_data, _ = audioop.ratecv(raw_data, self.sample_width, 1,
self.sample_rate, convert_rate, None)
        if convert_width is not None and self.sample_width != convert_width:
            if convert_width == 3:
                raw_data = audioop.lin2lin(raw_data, self.sample_width, 4)
                try:
                    audioop.bias(b"", 3, 0)
                except audioop.error:
                    raw_data = b"".join(raw_data[i + 1:i + 4] for i in range(0,
len(raw_data), 4))
                else:
                    raw_data = audioop.lin2lin(raw_data, self.sample_width,
convert_width)
            else:
                raw_data = audioop.lin2lin(raw_data, self.sample_width,
convert_width)
        if convert_width == 1:
            raw_data = audioop.bias(raw_data, 1, 128)
        return raw_data
```

```
    def get_wav_data(self, convert_rate=None, convert_width=None):
        raw_data = self.get_raw_data(convert_rate, convert_width)
        sample_rate = self.sample_rate if convert_rate is None else convert_rate
        sample_width = self.sample_width if convert_width is None else 
convert_width
        with io.BytesIO() as wav_file:
            wav_writer = wave.open(wav_file, "wb")
            try:
                wav_writer.setframerate(sample_rate)
                wav_writer.setsampwidth(sample_width)
                wav_writer.setnchannels(1)
                wav_writer.writeframes(raw_data)
                wav_data = wav_file.getvalue()
            finally:
                wav_writer.close()
        return wav_data

class Microphone(AudioSource):

    def __init__(self, device_index=None, sample_rate=None, chunk_size=1024):
        assert device_index is None or isinstance(device_index, int)
        assert sample_rate is None or (isinstance(sample_rate, int) and 
sample_rate > 0)
        assert isinstance(chunk_size, int) and chunk_size > 0
        self.pyaudio_module = self.get_pyaudio()
        audio = self.pyaudio_module.PyAudio()
        try:
            count = audio.get_device_count()
            if device_index is not None:
                assert 0 <= device_index < count
            if sample_rate is None:
                device_info = audio.get_device_info_by_index(device_index) if 
device_index is not None else audio.get_default_input_device_info()
                assert isinstance(device_info.get("defaultSampleRate"), (float, 
int)) and device_info["defaultSampleRate"] > 0
                sample_rate = int(device_info["defaultSampleRate"])
        except Exception:
            audio.terminate()
            raise
        self.device_index = device_index
        self.format = self.pyaudio_module.paInt16
        self.SAMPLE_WIDTH = self.pyaudio_module.get_sample_size(self.format)
        self.SAMPLE_RATE = sample_rate
        self.CHUNK = chunk_size
        self.audio = None
        self.stream = None

    @staticmethod
    def get_pyaudio():
        try:
            import pyaudio
        except ImportError:
            raise AttributeError("Could not find PyAudio; check installation")
        from distutils.version import LooseVersion
        if LooseVersion(pyaudio.__version__) < LooseVersion("0.2.11"):
            raise AttributeError("PyAudio 0.2.11 or later is required (found 
version {})".format(pyaudio.__version__))
        return pyaudio

    @staticmethod
    def list_microphone_names():
        audio = Microphone.get_pyaudio().PyAudio()
        try:
            result = []
            for i in range(audio.get_device_count()):
                device_info = audio.get_device_info_by_index(i)
                result.append(device_info.get("name"))
```

```
        finally:
            audio.terminate()
        return result

    def __enter__(self):
        assert self.stream is None
        self.audio = self.pyaudio_module.PyAudio()
        try:
            self.stream = Microphone.MicrophoneStream(
                self.audio.open(
                    input_device_index=self.device_index, channels=1,
                    format=self.format, rate=self.SAMPLE_RATE,
frames_per_buffer=self.CHUNK,
                    input=True,
                )
            )
        except Exception:
            self.audio.terminate()
            raise
        return self

    def __exit__(self, exc_type, exc_value, traceback):
        try:
            self.stream.close()
        finally:
            self.stream = None
            self.audio.terminate()

    class MicrophoneStream(object):
        def __init__(self, pyaudio_stream):
            self.pyaudio_stream = pyaudio_stream

        def read(self, size):
            return self.pyaudio_stream.read(size, exception_on_overflow=False)

        def close(self):
            try:
                if not self.pyaudio_stream.is_stopped():
                    self.pyaudio_stream.stop_stream()
            finally:
                self.pyaudio_stream.close()

class Client(AudioSource):

    def __init__(self):
        self.energy_threshold = 300
        self.dynamic_energy_threshold = True
        self.dynamic_energy_adjustment_damping = 0.15
        self.dynamic_energy_ratio = 1.5
        self.pause_threshold = 0.8
        self.operation_timeout = None
        self.phrase_threshold = 0.3
        self.non_speaking_duration = 0.5

    def adjust_for_ambient_noise(self, source, duration=1):
        assert isinstance(source, AudioSource)
        assert source.stream is not None
        assert self.pause_threshold >= self.non_speaking_duration >= 0
        seconds_per_buffer = (source.CHUNK + 0.0) / source.SAMPLE_RATE
        elapsed_time = 0
        while True:
            elapsed_time += seconds_per_buffer
            if elapsed_time > duration: break
            buffer = source.stream.read(source.CHUNK)
            energy = audioop.rms(buffer, source.SAMPLE_WIDTH)
            damping = self.dynamic_energy_adjustment_damping **
seconds_per_buffer
            target_energy = energy * self.dynamic_energy_ratio
```

```
            self.energy_threshold = self.energy_threshold * damping +
target_energy * (1 - damping)

    def listen(self, source, timeout=None, phrase_time_limit=None):
        assert isinstance(source, AudioSource)
        assert source.stream is not None
        assert self.pause_threshold >= self.non_speaking_duration >= 0
        seconds_per_buffer = float(source.CHUNK) / source.SAMPLE_RATE
        pause_buffer_count = int(math.ceil(self.pause_threshold /
seconds_per_buffer))
        phrase_buffer_count = int(math.ceil(self.phrase_threshold /
seconds_per_buffer))
        non_speaking_buffer_count = int(math.ceil(self.non_speaking_duration /
seconds_per_buffer))
        elapsed_time = 0
        buffer = b""
        while True:
            frames = collections.deque()
            while True:
                elapsed_time += seconds_per_buffer
                if timeout and elapsed_time > timeout:
                    break
                buffer = source.stream.read(source.CHUNK)
                if len(buffer) == 0: break
                frames.append(buffer)
                if len(frames) > non_speaking_buffer_count:
                    frames.popleft()
                energy = audioop.rms(buffer, source.SAMPLE_WIDTH)
                if energy > self.energy_threshold: break
                if self.dynamic_energy_threshold:
                    damping = self.dynamic_energy_adjustment_damping **
seconds_per_buffer
                    target_energy = energy * self.dynamic_energy_ratio
                    self.energy_threshold = self.energy_threshold * damping +
target_energy * (1 - damping)
            pause_count, phrase_count = 0, 0
            phrase_start_time = elapsed_time
            while True:
                elapsed_time += seconds_per_buffer
                if phrase_time_limit and elapsed_time - phrase_start_time >
phrase_time_limit:
                    break
                buffer = source.stream.read(source.CHUNK)
                if len(buffer) == 0: break
                frames.append(buffer)
                phrase_count += 1
                energy = audioop.rms(buffer, source.SAMPLE_WIDTH)
                if energy > self.energy_threshold:
                    pause_count = 0
                else:
                    pause_count += 1
                if pause_count > pause_buffer_count:
                    break
            phrase_count -= pause_count
            if phrase_count >= phrase_buffer_count or len(buffer) == 0: break
        for i in range(pause_count - non_speaking_buffer_count): frames.pop()
        frame_data = b"".join(frames)
        return AudioData(frame_data, source.SAMPLE_RATE, source.SAMPLE_WIDTH)

def Prosa_Authorization():
    try:
        with open("/data/prosa-auth.json", "r") as f:
            prosa_auth = json.load(f)
            headers = {"Authorization": "Bearer
{}".format(prosa_auth["access_token"])}
            text = {"session_id" : prosa_auth["session_id"], "message" :
"Assalamu'alaikum"}
```

```
            response = requests.post(url="http://35.198.196.217:5027/chat",
headers=headers, json=text)
            if response.status_code != 200:
                logging.error("{}".format(response.text))
                logging.error("VoiceBot API Test Failed")
                bottest = json.loads(response.text)
                if "message" in bottest:
                    if bottest["message"] == "Session expired":
                        raise Exception("Session expired")
            else:
                return prosa_auth
    except:
        response =
requests.post(url="http://35.198.196.217:5027/login?username=speaker&password=sma
rtspeaker")
        if response.status_code != 200:
            logging.error("{}".format(response.text))
            logging.error("VoiceBot API Login Failed")
        else:
            login = json.loads(response.text)
            headers = {"Authorization": "Bearer
{}".format(login["access_token"])}
            response = requests.get(url="http://35.198.196.217:5027/start-chat",
headers=headers)
            if response.status_code != 200:
                logging.error("{}".format(response.text))
                logging.error("VoiceBot API Session Failed")
            else:
                session = json.loads(response.text)
                prosa_auth = {"access_token" : login["access_token"],
"session_id" : session["session_id"]}
                with open("/data/prosa-auth.json", "w") as f:
                    json.dump(prosa_auth, f)
                return prosa_auth
    return None

def Prosa_Session(prosa_auth, audio):
    if "access_token" in prosa_auth and "session_id" in prosa_auth:
        headers = {"Authorization": "Bearer
{}".format(prosa_auth["access_token"])}
        data = {"session_id" : prosa_auth["session_id"]}
        files = {"audio": audio.get_wav_data()}
        response = requests.post(url="http://35.198.196.217:5027/audio-chat",
headers=headers, data=data, files=files)
        if response.status_code != 200:
            logging.error("{}".format(response.text))
            logging.error("VoiceBot API Session Failed")
        else:
            botreply = json.loads(response.text)
            if "response" in botreply:
                logging.info("Transcript of user request:
{}".format(botreply["response"]["text_transcript"]))
                logging.info("Transcript of chatbot response:
{}".format(botreply["response"]["chatbot_response"][-1]))
            if "audio_url" in botreply:
                audio_url = botreply["audio_url"]
                return audio_url
    else:
        logging.error("VoiceBot API Authorization Failed")
    return None

def Audio_Record(source):
    Client().adjust_for_ambient_noise(source)
    logging.info("Recording audio request.")
    call(["adk-message-send",
"led_indicate_direction_pattern{pattern:1,direction:50}"])
    audio = Client().listen(source, timeout=10)
    logging.info("End of audio request detected.")
```

```
    logging.info("Stopping recording.")
    call(["adk-message-send", "led_start_pattern{pattern:16}"])
    return audio

def Audio_Play(audio_bytes):
    logging.info("Playing voicebot response.")
    call(["adk-message-send", "led_start_pattern{pattern:2}"])
    with io.BytesIO() as wav_file:
        wav_file = io.BytesIO(audio_bytes)
        wf = wave.open(wav_file, "rb")
        p = pyaudio.PyAudio()
        stream = p.open(
            format = p.get_format_from_width(wf.getsampwidth()),
            channels = wf.getnchannels(),
            rate = wf.getframerate(),
            output = True
        )
        chunk = 1024
        data = wf.readframes(chunk)
        while data != b'':
            stream.write(data)
            data = wf.readframes(chunk)
        stream.close()
        p.terminate()
        logging.info("Finished playing voicebot response.")
        call(["adk-message-send",
"led_indicate_direction_pattern{pattern:17,direction:0}"])

def on_act():
    global waiting
    waiting = 0

with Microphone() as source:
    verbose = False
    logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)

    try:
        with open("/data/voicebot-mycroft.json", "r") as json_file:
            prosa_config = json.load(json_file)
            engine_path = prosa_config["engine_path"]
            model_path = prosa_config["model_path"]
            trigger_level = prosa_config["trigger_level"]
            sensitivity = prosa_config["sensitivity"]
    except Exception as e:
        logging.error("Error loading voicebot-mycroft.json: %s", e)
        sys.exit(-1)

    try:
        # initiate precise engine with mycroft model
        engine = PreciseEngine(engine_path, model_path)

        # initiate precise runner that will listen, predict, and detect wakeword
        runner = PreciseRunner(engine, on_activation=on_act,
trigger_level=trigger_level, sensitivity=sensitivity)

        # start runner
        runner.start()
    except Exception as e:
        logging.error("Wake Word Engine Error: %s", e)
        sys.exit(-1)

    call(["adk-message-send", "led_start_pattern{pattern:7}"])
    while True:
        logging.info("Waiting Wake Word")
        while waiting == 1:
            pass
        prosa_auth = Prosa_Authorization()
        if prosa_auth != None:
```

```
            audio = Audio_Record(source)
            audio_url = Prosa_Session(prosa_auth, audio)
            if audio_url != None:
                try:
                    audio_file = urlopen(audio_url)
                    audio_bytes = audio_file.read()
                    Audio_Play(audio_bytes)
                except Exception as e:
                    logging.error("{}".format(e))
        waiting = 1
```

Perhatikan bahwa '/data/prosa-auth.json' adalah letak file data otorisasi untuk mengakses API sedangkan '/data/voicebot-mycroft.json' adalah letak file konfigurasi.

File Spec PyInstaller Aplikasi Voicebot Prosa terintegrasi Mycroft Precise

```
# -*- mode: python ; coding: utf-8 -*-

block_cipher = None

a = Analysis(['voicebot-mycroft.py'],
             pathex=['.'],
             binaries=[('/usr/lib/x86_64-linux-gnu/libxcb.so.1','.')],
             datas=[],
             hiddenimports=['_portaudio'],
             hookspath=[],
             runtime_hooks=[],
             excludes=[],
             win_no_prefer_redirects=False,
             win_private_assemblies=False,
             cipher=block_cipher,
             noarchive=False)
pyz = PYZ(a.pure, a.zipped_data,
             cipher=block_cipher)
exe = EXE(pyz,
          a.scripts,
          [],
          exclude_binaries=True,
          name='voicebot-mycroft',
          debug=False,
          bootloader_ignore_signals=False,
          strip=False,
          upx=True,
          console=True )
coll = COLLECT(exe,
               a.binaries,
               a.zipfiles,
               a.datas,
               strip=False,
               upx=True,
               upx_exclude=[],
               name='voicebot-mycroft')
```

Perhatikan bahwa letak file '/usr/lib/x86_64-linux-gnu/libxcb.so.1' akan berbeda untuk arsitektur sistem yang berbeda, yaitu:

- Ubuntu Desktop (x86_64) : '/usr/lib/x86_64-linux-gnu/libxcb.so.1'
- Ubuntu Arm (ARM 32) : '/usr/lib/arm-linux-gnueabihf/libxcb.so.1'
- Ubuntu Aarch64 (ARM 64) : '/usr/lib/aarch64-linux-gnu/libxcb.so.1'