Membuat File Aplikasi Voicebot Prosa 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:



1. Buat folder baru:

|  |
| --- |
| $ mkdir prosa-voicebot $ cd prosa-voicebot |

1. Buat script Python (terlampir):

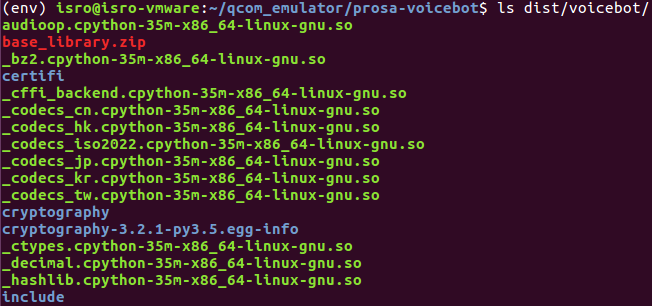
|  |
| --- |
| $ nano voicebot.py |

1. Buat file spec pyinstaller (terlampir):

|  |
| --- |
| $ nano voicebot.spec |

1. Build aplikasi Voicebot Prosa:

|  |
| --- |
| $ pyinstaller --clean -y voicebot.spec |



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

Script Python Aplikasi Voicebot Prosa

|  |
| --- |
| #!/usr/bin/env python3  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  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=smartspeaker")  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}"])  with Microphone() as source:  verbose = False  logging.basicConfig(level=logging.DEBUG if verbose else logging.INFO)  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)) |

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

File Spec PyInstaller Aplikasi Voicebot Prosa

|  |
| --- |
| # -\*- mode: python ; coding: utf-8 -\*-  block\_cipher = None  a = Analysis(['voicebot.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',  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') |

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'