**EDITH (VIRTUAL ASSISTANT)**

A

Mini Project Report

Submitted in partial fulfilment of the

Requirements for the award of the Degree of

**BACHELOR OF ENGINEERING**

IN

**COMPUTER SCIENCE & ENGINEERING**

By

**SAI SUMAN CHITTURI**

**1602-18-733-097**

**PRANEETH KAPILA**

**1602-18-733-116**

****

**Department of Computer Science & Engineering**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Ibrahimbagh, Hyderabad-31**

**2020-2021**

**Vasavi College of Engineering (Autonomous)**

**(Affiliated to Osmania University)**

**Hyderabad-500 031**

**ACKNOWLEDGEMENT**

We are thankful to the College Management for Encouraging us to do our Mini Project titled Meta Stream. We extend our heart-felt gratitude to our Professor, M. Sashi Kumar and our Head of the Dept. Dr. T. Adilakshmi for their Invaluable Guidance and Support. Their guidance was unforgettable and their constructive suggestions helped us in finishing the project Effectively.

**TABLE OF CONTENTS**

List of Figures 4

Abstract 5

Introduction 6

Software Requirements Specifications 7

System Design 8

Pseudo Algorithm/Implementation/Code 10

Results/Screenshots 20

Performance Testing 27

Conclusion and Future Work 29

References 30

Full Project Demonstration (YouTube) 30

**LIST OF FIGURES**

1. Overall Architecture 8

2. Google Assistant Backend Services 8

3. Speech-to-text Module 8

4. Text-to-Speech Module 9

5. Music Playback Module 9

6. News Module 9

7. Screenshots 20

* Weather Updates 20
* Traffic Updates 21
* Directions to a place 22
* Setting a Reminder 23
* Ring my phone feature 24
* Jokes and Facts 25
* News and Sports Updates 25
* Music Playback 26

8. Performance Testing 27

* Errors Encountered 27
* Latency by API 27
* Overall Latency 28
* Traffic Response 28

**EDITH (VIRTUAL ASSISTANT)**

**Abstract**

The main aim of the work is to develop an economically effective and performance wise efficient virtual assistant using Raspberry Pi for home automation based on the concepts of Internet of Things, Speech Recognition, Natural Language Processing and Artificial Intelligence. People who are using it can give voice inputs and the device responds through voice commands by itself. It can fetch the date, time, weather, play your favourite music and fetch search results from the internet.

The Raspberry Pi processes the speech inputs online given by the user through the mic and converts it into text and executes the command. The whole project is put in action through a python script which includes online Speech to Text conversion and Text to Speech conversion codes written.

The device will respond to the user in a casual manner so that the user has a friendly experience with the device and feels it like his or her own assistant. This device makes the day-by-day processes easier.

**Introduction**

People today are living the busiest life. While managing lot of activities, they keep forgetting few. They find it difficult to plan and manage activities. Moreover, under such lifestyle, they are not having enough access to immediate Entertainment.

This creates a need for a Virtual Assistant. Virtual Assistants, today, are capable of planning activities, setting reminders, alarms, playing music, telling jokes and facts etc.

To Solve these problems, we built a Virtual Assistant (Edith) which is capable of planning activities, setting reminders, alarms, playing music, telling jokes and facts etc.

This work is constructed based on the basic concepts on Internet of things (IoT) and Natural Language Processing (NLP). This system is designed to provide a user-friendly experience as well as an easier interface so that anyone can use this effortlessly. This project has been implemented with the help of Raspberry Pi 3 Model B, Google Assistant API, News API and Web scraping tools.

Natural Language Processing is simply a bridge-way that reduces the distance between human communication and machine communication. The main objective of NLP is to make the machines understand the natural human language so that the usage becomes very much comfortable. In technical terms, NLP is the algorithm which analyses and synthesizes human speech. This algorithm is based on artificial intelligence and computational linguistics.

**SOFTWARE REQUIREMENTS SPECIFICATION**

The software is designed to be light-weighted so that it doesn’t be a burden on the machine running it. This system is being build keeping in mind the generally available hardware and software compatibility. Here are the minimum hardware and software requirements for Edith - virtual assistant.

**Hardware Requirements:**

* Raspberry Pi 3 B+
* USB Microphone
* USB Speaker
* Interfacing cables – HDMI, Ethernet Cable

**Software Requirements:**

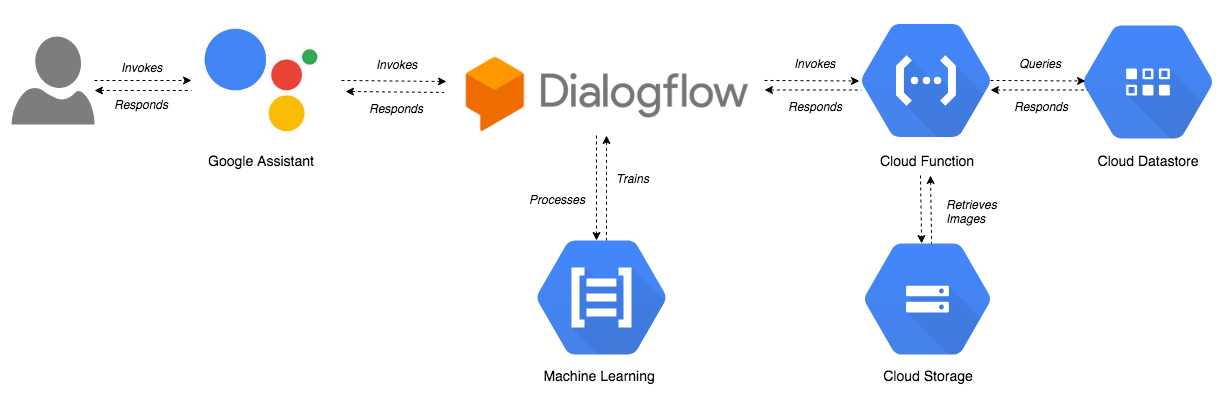
* Raspbian OS
* Python 3
* Google Voice Assistant API must be configured in the machine
* Chromium based Web Browsers
* Python3 Modules Required
  + GRPC Client
  + Google OAuth lib tool
  + gTTS – Google Text to speech
  + Play sound
  + Pafy - YouTube Scraper
  + Speech Recognition
  + Python3 Virtual Environment
  + Port Audio 19
  + PyAudio

**SYSTEM DESIGN**

1. Overall Architecture



1. Google Assistant Backend Services



1. Speech to text Module



1. Text to Speech Module



1. Music Playback Module 6. News Module

**PSEUDO ALGORITHM / IMPLEMENTATION / CODE**

**Code:**

1. **Assitant.py**

"""Sample that implements a gRPC client for the Google Assistant API."""

**import** concurrent**.**futures

**import** json

**import** logging

**import** os

**import** os**.**path

**import** pathlib2 **as** pathlib

**import** sys

**import** time

**import** uuid

**import** click

**import** grpc

**import** google**.**auth**.**transport**.**grpc

**import** google**.**auth**.**transport**.**requests

**import** google**.**oauth2**.**credentials

**from** google**.**assistant**.**embedded**.**v1alpha2 **import** **(**

embedded\_assistant\_pb2**,**

embedded\_assistant\_pb2\_grpc

**)**

**from** tenacity **import** retry**,** stop\_after\_attempt**,** retry\_if\_exception

**try:**

**from** **.** **import** **(**

assistant\_helpers**,**

audio\_helpers**,**

browser\_helpers**,**

device\_helpers

**)**

**except** **(SystemError,** **ImportError):**

**import** assistant\_helpers

**import** audio\_helpers

**import** browser\_helpers

**import** device\_helpers

**from** Play\_Music **import** play\_song

**from** News **import** NewsFromBBC

ASSISTANT\_API\_ENDPOINT **=** 'embeddedassistant.googleapis.com'

END\_OF\_UTTERANCE **=** embedded\_assistant\_pb2**.**AssistResponse**.**END\_OF\_UTTERANCE

DIALOG\_FOLLOW\_ON **=** embedded\_assistant\_pb2**.**DialogStateOut**.**DIALOG\_FOLLOW\_ON

CLOSE\_MICROPHONE **=** embedded\_assistant\_pb2**.**DialogStateOut**.**CLOSE\_MICROPHONE

PLAYING **=** embedded\_assistant\_pb2**.**ScreenOutConfig**.**PLAYING

DEFAULT\_GRPC\_DEADLINE **=** 60 **\*** 3 **+** 5

**class** **SampleAssistant(object):**

"""Sample Assistant that supports conversations and device actions.

Args:

device\_model\_id: identifier of the device model.

device\_id: identifier of the registered device instance.

conversation\_stream(ConversationStream): audio stream

for recording query and playing back assistant answer.

channel: authorized gRPC channel for connection to the

Google Assistant API.

deadline\_sec: gRPC deadline in seconds for Google Assistant API call.

device\_handler: callback for device actions.

"""

**def** \_\_init\_\_**(**self**,** language\_code**,** device\_model\_id**,** device\_id**,**

conversation\_stream**,** display**,**

channel**,** deadline\_sec**,** device\_handler**):**

self**.**language\_code **=** language\_code

self**.**device\_model\_id **=** device\_model\_id

self**.**device\_id **=** device\_id

self**.**conversation\_stream **=** conversation\_stream

self**.**display **=** display

# Opaque blob provided in AssistResponse that,

# when provided in a follow-up AssistRequest,

# gives the Assistant a context marker within the current state

# of the multi-Assist()-RPC "conversation".

# This value, along with MicrophoneMode, supports a more natural

# "conversation" with the Assistant.

self**.**conversation\_state **=** **None**

# Force reset of first conversation.

self**.**is\_new\_conversation **=** **True**

# Create Google Assistant API gRPC client.

self**.**assistant **=** embedded\_assistant\_pb2\_grpc**.**EmbeddedAssistantStub**(**

channel

**)**

self**.**deadline **=** deadline\_sec

self**.**device\_handler **=** device\_handler

**def** \_\_enter\_\_**(**self**):**

**return** self

**def** \_\_exit\_\_**(**self**,** etype**,** e**,** traceback**):**

**if** e**:**

**return** **False**

self**.**conversation\_stream**.**close**()**

**def** is\_grpc\_error\_unavailable**(**e**):**

is\_grpc\_error **=** **isinstance(**e**,** grpc**.**RpcError**)**

**if** is\_grpc\_error **and** **(**e**.**code**()** **==** grpc**.**StatusCode**.**UNAVAILABLE**):**

logging**.**error**(**'grpc unavailable error: %s'**,** e**)**

**return** **True**

**return** **False**

@retry**(**reraise**=True,** stop**=**stop\_after\_attempt**(**3**),**

retry**=**retry\_if\_exception**(**is\_grpc\_error\_unavailable**))**

**def** assist**(**self**):**

"""Send a voice request to the Assistant and playback the response.

Returns: True if conversation should continue.

"""

continue\_conversation **=** **False**

device\_actions\_futures **=** **[]**

self**.**conversation\_stream**.**start\_recording**()**

logging**.**info**(**'Recording audio request.'**)**

**def** iter\_log\_assist\_requests**():**

**for** c **in** self**.**gen\_assist\_requests**():**

assistant\_helpers**.**log\_assist\_request\_without\_audio**(**c**)**

**yield** c

logging**.**debug**(**'Reached end of AssistRequest iteration.'**)**

# This generator yields AssistResponse proto messages

# received from the gRPC Google Assistant API.

user\_transcripts **=** **[**'0'**]**

**for** resp **in** self**.**assistant**.**Assist**(**iter\_log\_assist\_requests**(),**

self**.**deadline**):**

assistant\_helpers**.**log\_assist\_response\_without\_audio**(**resp**)**

**if** resp**.**event\_type **==** END\_OF\_UTTERANCE**:**

logging**.**info**(**'End of audio request detected.'**)**

logging**.**info**(**'Stopping recording.'**)**

self**.**conversation\_stream**.**stop\_recording**()**

**if** resp**.**speech\_results**:**

user\_transcripts**[**0**]** **=** ' '**.**join**(**r**.**transcript

**for** r **in** resp**.**speech\_results**)**

logging**.**info**(**'Transcript of user request: "%s".'**,**

' '**.**join**(**r**.**transcript

**for** r **in** resp**.**speech\_results**))**

**if** **len(**resp**.**audio\_out**.**audio\_data**)** **>** 0**:**

**if** **not** self**.**conversation\_stream**.**playing**:**

self**.**conversation\_stream**.**stop\_recording**()**

self**.**conversation\_stream**.**start\_playback**()**

logging**.**info**(**'Playing assistant response.'**)**

**if** "news" **in** user\_transcripts**[**0**]:**

NewsFromBBC**()**

**exit(**0**)**

**elif** "play" **in** user\_transcripts**[**0**]** **and** "game" **not** **in** user\_transcripts**[**0**]:**

song\_name **=** user\_transcripts**[**0**][**5**:]**

song\_name **=** song\_name**.**replace**(**" "**,** "+"**)**

play\_song**(**song\_name**)**

**else:**

self**.**conversation\_stream**.**write**(**resp**.**audio\_out**.**audio\_data**)**

self**.**conversation\_stream**.**write**(**resp**.**audio\_out**.**audio\_data**)**

**if** resp**.**dialog\_state\_out**.**conversation\_state**:**

conversation\_state **=** resp**.**dialog\_state\_out**.**conversation\_state

logging**.**debug**(**'Updating conversation state.'**)**

self**.**conversation\_state **=** conversation\_state

**if** resp**.**dialog\_state\_out**.**volume\_percentage **!=** 0**:**

volume\_percentage **=** resp**.**dialog\_state\_out**.**volume\_percentage

logging**.**info**(**'Setting volume to %s%%'**,** volume\_percentage**)**

self**.**conversation\_stream**.**volume\_percentage **=** volume\_percentage

**if** resp**.**dialog\_state\_out**.**microphone\_mode **==** DIALOG\_FOLLOW\_ON**:**

continue\_conversation **=** **True**

logging**.**info**(**'Expecting follow-on query from user.'**)**

**elif** resp**.**dialog\_state\_out**.**microphone\_mode **==** CLOSE\_MICROPHONE**:**

continue\_conversation **=** **False**

**if** resp**.**device\_action**.**device\_request\_json**:**

device\_request **=** json**.**loads**(**

resp**.**device\_action**.**device\_request\_json

**)**

fs **=** self**.**device\_handler**(**device\_request**)**

**if** fs**:**

device\_actions\_futures**.**extend**(**fs**)**

**if** self**.**display **and** resp**.**screen\_out**.**data**:**

system\_browser **=** browser\_helpers**.**system\_browser

system\_browser**.**display**(**resp**.**screen\_out**.**data**)**

**if** **len(**device\_actions\_futures**):**

logging**.**info**(**'Waiting for device executions to complete.'**)**

concurrent**.**futures**.**wait**(**device\_actions\_futures**)**

logging**.**info**(**'Finished playing assistant response.'**)**

self**.**conversation\_stream**.**stop\_playback**()**

**return** continue\_conversation

**def** gen\_assist\_requests**(**self**):**

"""Yields: AssistRequest messages to send to the API."""

config **=** embedded\_assistant\_pb2**.**AssistConfig**(**

audio\_in\_config**=**embedded\_assistant\_pb2**.**AudioInConfig**(**

encoding**=**'LINEAR16'**,**

sample\_rate\_hertz**=**self**.**conversation\_stream**.**sample\_rate**,**

**),**

audio\_out\_config**=**embedded\_assistant\_pb2**.**AudioOutConfig**(**

encoding**=**'LINEAR16'**,**

sample\_rate\_hertz**=**self**.**conversation\_stream**.**sample\_rate**,**

volume\_percentage**=**self**.**conversation\_stream**.**volume\_percentage**,**

**),**

dialog\_state\_in**=**embedded\_assistant\_pb2**.**DialogStateIn**(**

language\_code**=**self**.**language\_code**,**

conversation\_state**=**self**.**conversation\_state**,**

is\_new\_conversation**=**self**.**is\_new\_conversation**,**

**),**

device\_config**=**embedded\_assistant\_pb2**.**DeviceConfig**(**

device\_id**=**self**.**device\_id**,**

device\_model\_id**=**self**.**device\_model\_id**,**

**)**

**)**

**if** self**.**display**:**

config**.**screen\_out\_config**.**screen\_mode **=** PLAYING

# Continue current conversation with later requests.

self**.**is\_new\_conversation **=** **False**

# The first AssistRequest must contain the AssistConfig

# and no audio data.

**yield** embedded\_assistant\_pb2**.**AssistRequest**(**config**=**config**)**

**for** data **in** self**.**conversation\_stream**:**

# Subsequent requests need audio data, but not config.

**yield** embedded\_assistant\_pb2**.**AssistRequest**(**audio\_in**=**data**)**

@click**.**command**()**

@click**.**option**(**'--api-endpoint'**,** default**=**ASSISTANT\_API\_ENDPOINT**,**

metavar**=**'<api endpoint>'**,** show\_default**=True,**

**help=**'Address of Google Assistant API service.'**)**

@click**.**option**(**'--credentials'**,**

metavar**=**'<credentials>'**,** show\_default**=True,**

default**=**os**.**path**.**join**(**click**.**get\_app\_dir**(**'google-oauthlib-tool'**),**

'credentials.json'**),**

**help=**'Path to read OAuth2 credentials.'**)**

@click**.**option**(**'--project-id'**,**

metavar**=**'<project id>'**,**

**help=(**'Google Developer Project ID used for registration '

'if --device-id is not specified'**))**

@click**.**option**(**'--device-model-id'**,**

metavar**=**'<device model id>'**,**

**help=((**'Unique device model identifier, '

'if not specifed, it is read from --device-config'**)))**

@click**.**option**(**'--device-id'**,**

metavar**=**'<device id>'**,**

**help=((**'Unique registered device instance identifier, '

'if not specified, it is read from --device-config, '

'if no device\_config found: a new device is registered '

'using a unique id and a new device config is saved'**)))**

@click**.**option**(**'--device-config'**,** show\_default**=True,**

metavar**=**'<device config>'**,**

default**=**os**.**path**.**join**(**

click**.**get\_app\_dir**(**'googlesamples-assistant'**),**

'device\_config.json'**),**

**help=**'Path to save and restore the device configuration'**)**

@click**.**option**(**'--lang'**,** show\_default**=True,**

metavar**=**'<language code>'**,**

default**=**'en-US'**,**

**help=**'Language code of the Assistant'**)**

@click**.**option**(**'--display'**,** is\_flag**=True,** default**=False,**

**help=**'Enable visual display of Assistant responses in HTML.'**)**

@click**.**option**(**'--verbose'**,** '-v'**,** is\_flag**=True,** default**=False,**

**help=**'Verbose logging.'**)**

@click**.**option**(**'--input-audio-file'**,** '-i'**,**

metavar**=**'<input file>'**,**

**help=**'Path to input audio file. '

'If missing, uses audio capture'**)**

@click**.**option**(**'--output-audio-file'**,** '-o'**,**

metavar**=**'<output file>'**,**

**help=**'Path to output audio file. '

'If missing, uses audio playback'**)**

@click**.**option**(**'--audio-sample-rate'**,**

default**=**audio\_helpers**.**DEFAULT\_AUDIO\_SAMPLE\_RATE**,**

metavar**=**'<audio sample rate>'**,** show\_default**=True,**

**help=**'Audio sample rate in hertz.'**)**

@click**.**option**(**'--audio-sample-width'**,**

default**=**audio\_helpers**.**DEFAULT\_AUDIO\_SAMPLE\_WIDTH**,**

metavar**=**'<audio sample width>'**,** show\_default**=True,**

**help=**'Audio sample width in bytes.'**)**

@click**.**option**(**'--audio-iter-size'**,**

default**=**audio\_helpers**.**DEFAULT\_AUDIO\_ITER\_SIZE**,**

metavar**=**'<audio iter size>'**,** show\_default**=True,**

**help=**'Size of each read during audio stream iteration in bytes.'**)**

@click**.**option**(**'--audio-block-size'**,**

default**=**audio\_helpers**.**DEFAULT\_AUDIO\_DEVICE\_BLOCK\_SIZE**,**

metavar**=**'<audio block size>'**,** show\_default**=True,**

**help=(**'Block size in bytes for each audio device '

'read and write operation.'**))**

@click**.**option**(**'--audio-flush-size'**,**

default**=**audio\_helpers**.**DEFAULT\_AUDIO\_DEVICE\_FLUSH\_SIZE**,**

metavar**=**'<audio flush size>'**,** show\_default**=True,**

**help=(**'Size of silence data in bytes written '

'during flush operation'**))**

@click**.**option**(**'--grpc-deadline'**,** default**=**DEFAULT\_GRPC\_DEADLINE**,**

metavar**=**'<grpc deadline>'**,** show\_default**=True,**

**help=**'gRPC deadline in seconds'**)**

@click**.**option**(**'--once'**,** default**=False,** is\_flag**=True,**

**help=**'Force termination after a single conversation.'**)**

**def** main**(**api\_endpoint**,** credentials**,** project\_id**,**

device\_model\_id**,** device\_id**,** device\_config**,**

lang**,** display**,** verbose**,**

input\_audio\_file**,** output\_audio\_file**,**

audio\_sample\_rate**,** audio\_sample\_width**,**

audio\_iter\_size**,** audio\_block\_size**,** audio\_flush\_size**,**

grpc\_deadline**,** once**,** **\***args**,** **\*\***kwargs**):**

"""Samples for the Google Assistant API.

Examples:

Run the sample with microphone input and speaker output:

$ python -m googlesamples.assistant

Run the sample with file input and speaker output:

$ python -m googlesamples.assistant -i <input file>

Run the sample with file input and output:

$ python -m googlesamples.assistant -i <input file> -o <output file>

"""

# Setup logging.

logging**.**basicConfig**(**level**=**logging**.**DEBUG **if** verbose **else** logging**.**INFO**)**

# Load OAuth 2.0 credentials.

**try:**

**with** **open(**credentials**,** 'r'**)** **as** f**:**

credentials **=** google**.**oauth2**.**credentials**.**Credentials**(**token**=None,**

**\*\***json**.**load**(**f**))**

http\_request **=** google**.**auth**.**transport**.**requests**.**Request**()**

credentials**.**refresh**(**http\_request**)**

**except** **Exception** **as** e**:**

logging**.**error**(**'Error loading credentials: %s'**,** e**)**

logging**.**error**(**'Run google-oauthlib-tool to initialize '

'new OAuth 2.0 credentials.'**)**

sys**.exit(-**1**)**

# Create an authorized gRPC channel.

grpc\_channel **=** google**.**auth**.**transport**.**grpc**.**secure\_authorized\_channel**(**

credentials**,** http\_request**,** api\_endpoint**)**

logging**.**info**(**'Connecting to %s'**,** api\_endpoint**)**

# Configure audio source and sink.

audio\_device **=** **None**

**if** input\_audio\_file**:**

audio\_source **=** audio\_helpers**.**WaveSource**(**

**open(**input\_audio\_file**,** 'rb'**),**

sample\_rate**=**audio\_sample\_rate**,**

sample\_width**=**audio\_sample\_width

**)**

**else:**

audio\_source **=** audio\_device **=** **(**

audio\_device **or** audio\_helpers**.**SoundDeviceStream**(**

sample\_rate**=**audio\_sample\_rate**,**

sample\_width**=**audio\_sample\_width**,**

block\_size**=**audio\_block\_size**,**

flush\_size**=**audio\_flush\_size

**)**

**)**

**if** output\_audio\_file**:**

audio\_sink **=** audio\_helpers**.**WaveSink**(**

**open(**output\_audio\_file**,** 'wb'**),**

sample\_rate**=**audio\_sample\_rate**,**

sample\_width**=**audio\_sample\_width

**)**

**else:**

audio\_sink **=** audio\_device **=** **(**

audio\_device **or** audio\_helpers**.**SoundDeviceStream**(**

sample\_rate**=**audio\_sample\_rate**,**

sample\_width**=**audio\_sample\_width**,**

block\_size**=**audio\_block\_size**,**

flush\_size**=**audio\_flush\_size

**)**

**)**

# Create conversation stream with the given audio source and sink.

conversation\_stream **=** audio\_helpers**.**ConversationStream**(**

source**=**audio\_source**,**

sink**=**audio\_sink**,**

iter\_size**=**audio\_iter\_size**,**

sample\_width**=**audio\_sample\_width**,**

**)**

**if** **not** device\_id **or** **not** device\_model\_id**:**

**try:**

**with** **open(**device\_config**)** **as** f**:**

device **=** json**.**load**(**f**)**

device\_id **=** device**[**'id'**]**

device\_model\_id **=** device**[**'model\_id'**]**

logging**.**info**(**"Using device model %s and device id %s"**,**

device\_model\_id**,**

device\_id**)**

**except** **Exception** **as** e**:**

logging**.**warning**(**'Device config not found: %s' **%** e**)**

logging**.**info**(**'Registering device'**)**

**if** **not** device\_model\_id**:**

logging**.**error**(**'Option --device-model-id required '

'when registering a device instance.'**)**

sys**.exit(-**1**)**

**if** **not** project\_id**:**

logging**.**error**(**'Option --project-id required '

'when registering a device instance.'**)**

sys**.exit(-**1**)**

device\_base\_url **=** **(**

'https://%s/v1alpha2/projects/%s/devices' **%** **(**api\_endpoint**,**

project\_id**)**

**)**

device\_id **=** **str(**uuid**.**uuid1**())**

payload **=** **{**

'id'**:** device\_id**,**

'model\_id'**:** device\_model\_id**,**

'client\_type'**:** 'SDK\_SERVICE'

**}**

session **=** google**.**auth**.**transport**.**requests**.**AuthorizedSession**(**

credentials

**)**

r **=** session**.**post**(**device\_base\_url**,** data**=**json**.**dumps**(**payload**))**

**if** r**.**status\_code **!=** 200**:**

logging**.**error**(**'Failed to register device: %s'**,** r**.**text**)**

sys**.exit(-**1**)**

logging**.**info**(**'Device registered: %s'**,** device\_id**)**

pathlib**.**Path**(**os**.**path**.**dirname**(**device\_config**)).**mkdir**(**exist\_ok**=True)**

**with** **open(**device\_config**,** 'w'**)** **as** f**:**

json**.**dump**(**payload**,** f**)**

device\_handler **=** device\_helpers**.**DeviceRequestHandler**(**device\_id**)**

@device\_handler**.**command**(**'action.devices.commands.OnOff'**)**

**def** onoff**(**on**):**

**if** on**:**

logging**.**info**(**'Turning device on'**)**

**else:**

logging**.**info**(**'Turning device off'**)**

@device\_handler**.**command**(**'com.example.commands.BlinkLight'**)**

**def** blink**(**speed**,** number**):**

logging**.**info**(**'Blinking device %s times.' **%** number**)**

delay **=** 1

**if** speed **==** "SLOWLY"**:**

delay **=** 2

**elif** speed **==** "QUICKLY"**:**

delay **=** 0.5

**for** i **in** **range(int(**number**)):**

logging**.**info**(**'Device is blinking.'**)**

time**.**sleep**(**delay**)**

**with** SampleAssistant**(**lang**,** device\_model\_id**,** device\_id**,**

conversation\_stream**,** display**,**

grpc\_channel**,** grpc\_deadline**,**

device\_handler**)** **as** assistant**:**

# If file arguments are supplied:

# exit after the first turn of the conversation.

**if** input\_audio\_file **or** output\_audio\_file**:**

assistant**.**assist**()**

**return**

# If no file arguments supplied:

# keep recording voice requests using the microphone

# and playing back assistant response using the speaker.

# When the once flag is set, don't wait for a trigger. Otherwise, wait.

wait\_for\_user\_trigger **=** **not** once

**while** **True:**

**if** wait\_for\_user\_trigger**:**

click**.**pause**(**info**=**'Press Enter to send a new request...'**)**

continue\_conversation **=** assistant**.**assist**()**

# wait for user trigger if there is no follow-up turn in

# the conversation.

wait\_for\_user\_trigger **=** **not** continue\_conversation

# If we only want one conversation, break.

**if** once **and** **(not** continue\_conversation**):**

**break**

**if** \_\_name\_\_ **==** '\_\_main\_\_'**:**

main**()**

1. **Music\_Playback.py**

# This program plays the music and videos from YouTube

**import** os # For system calls

**import** signal # For SIGTERM value

**import** pafy # Used to collect duration of video

**import** re # Used to parse the webpage of search results

**import** urllib**.**request # To create a request to URL

**import** urllib**.**parse # To encode URLs

**import** subprocess # To Play song on Chrome

**import** time # For sleep

**def** play\_song**(**msg**):**

# song name from user

song **=** urllib**.**parse**.**urlencode**({**"search\_query" **:** msg**})**

# fetch the ?v=query\_string

result **=** urllib**.**request**.**urlopen**(**"http://www.youtube.com/results?" **+** song**)**

# make the url of the first result song

search\_results **=** re**.**findall**(**r'\/watch\?v=(.{11})'**,** result**.**read**().**decode**())**

# make the final url of song; selects the very first result from youtube result

url **=** "https://www.youtube.com/watch?v=" **+** search\_results**[**0**]**

# Extract the length of video from Metadata

video **=** pafy**.**new**(**url**)**

length **=** video**.**length

# Start a chrome session with the Video URL

pid **=** subprocess**.**Popen**(**"google-chrome " **+** url**,** shell **=** **True).**pid

# Wait until the video finishes

time**.**sleep**(**length **+** 5**)**

# Terminate the Chrome session

os**.**killpg**(**os**.**getpgid**(**pid**),** signal**.**SIGTERM**)**

1. **News.py**

# Reads out News Headlines from BBC

**import** requests

**import** gtts

**import** playsound

**import** os

**def** NewsFromBBC**():**

# BBC news api

# Following query parameters are used

# source, sortBy and apiKey

query\_params **=** **{**

"source"**:** "bbc-news"**,**

"sortBy"**:** "top"**,**

"apiKey"**:** "<Your API Key>"

**}**

# URL of News API

main\_url **=** " https://newsapi.org/v1/articles"

# fetching data in json format

res **=** requests**.**get**(**main\_url**,** params**=**query\_params**)**

open\_bbc\_page **=** res**.**json**()**

# getting all articles in a string article

article **=** open\_bbc\_page**[**"articles"**]**

# empty list which will

# contain all trending news

results **=** **[]**

**for** ar **in** article**:**

results**.**append**(**ar**[**"title"**])**

**for** i **in** **range(len(**results**)):**

# Convert Text to Speech using Google Text-to-speech Engine

myobj **=** gtts**.**gTTS**(**text **=** results**[**i**],** lang **=** 'en'**)**

# Save it in Mp3 Format

myobj**.**save**(**"News.mp3"**)**

# Play the music file

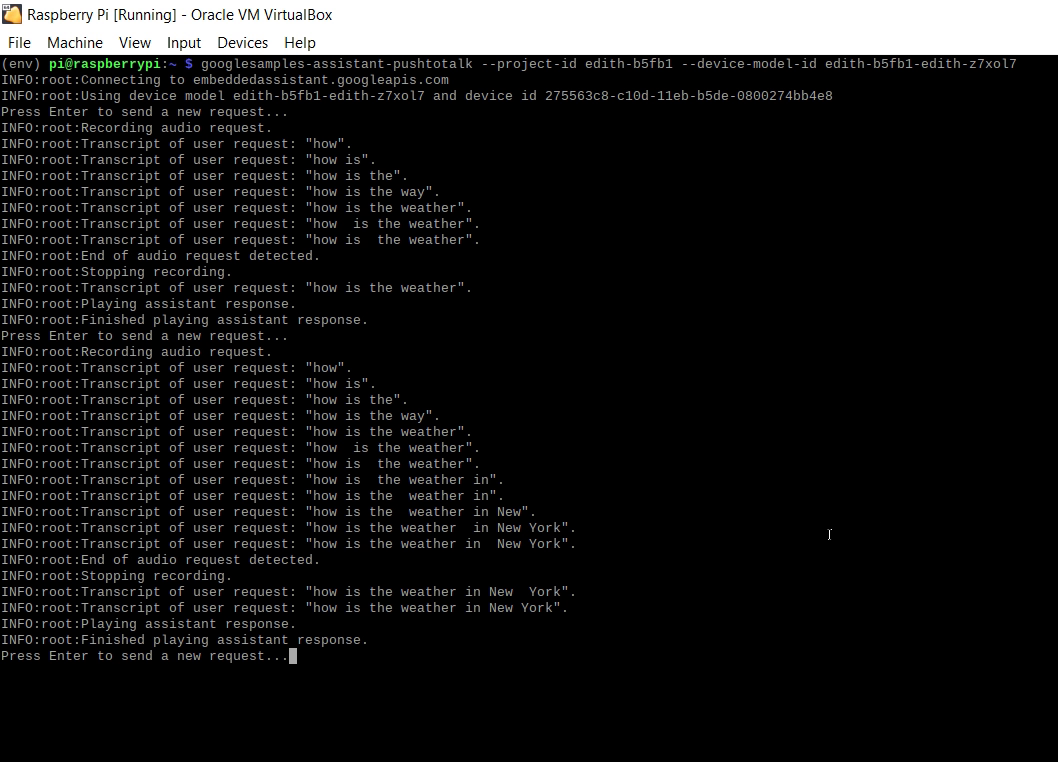
playsound**.**playsound**(**"News.mp3"**)**

# Remove the music file after reading

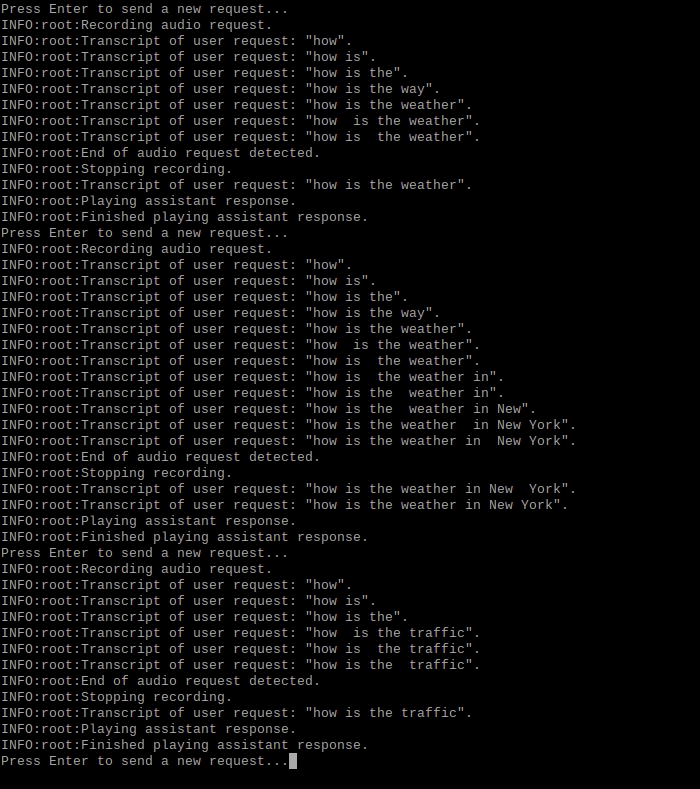
os**.**system**(**"rm News.mp3"**)**

**SCREENSHOTS**

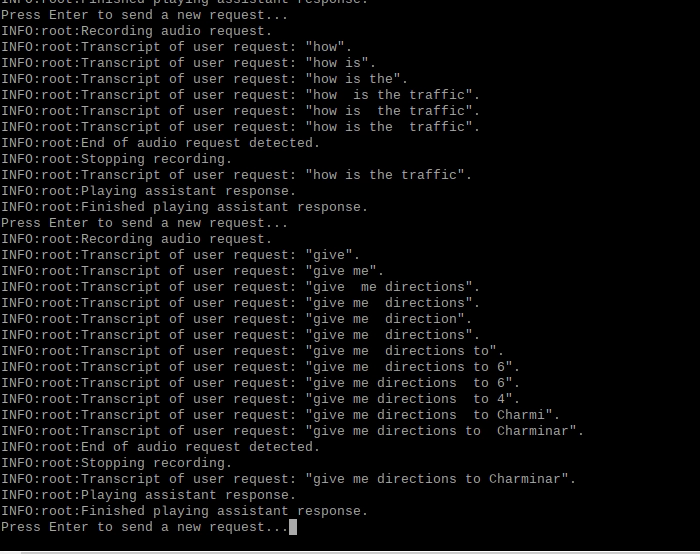
1. Weather Updates



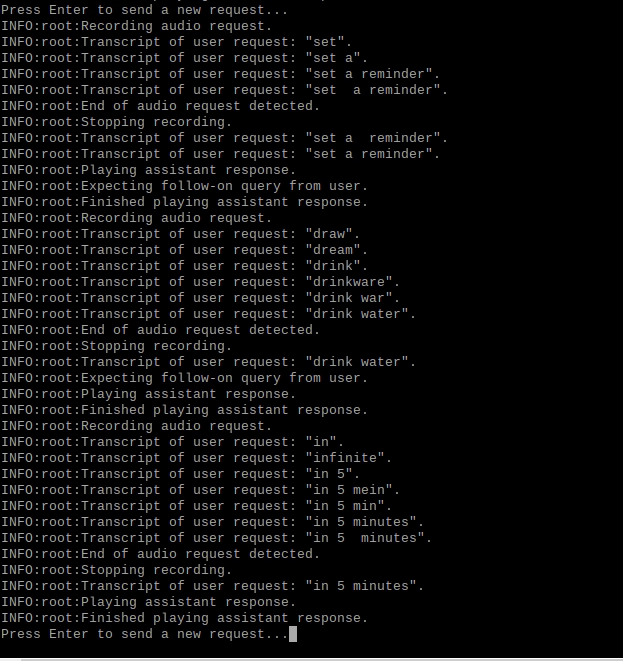
1. Traffic Updates



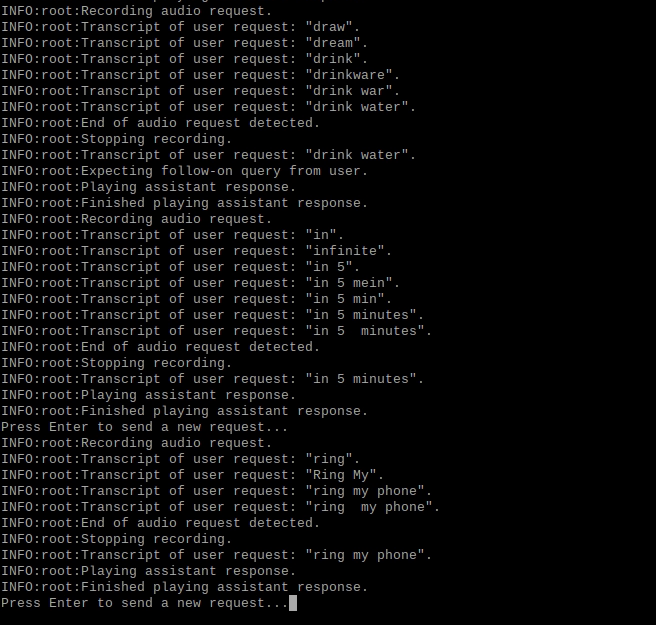
1. Directions to a place



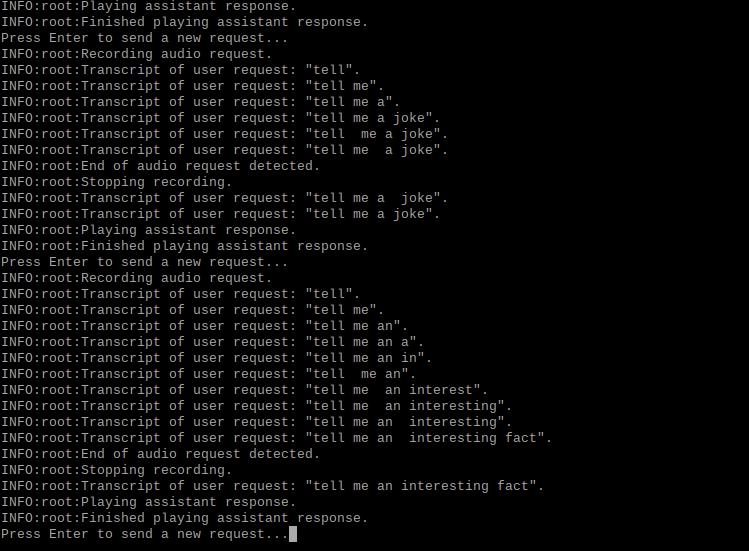
1. Setting Reminder



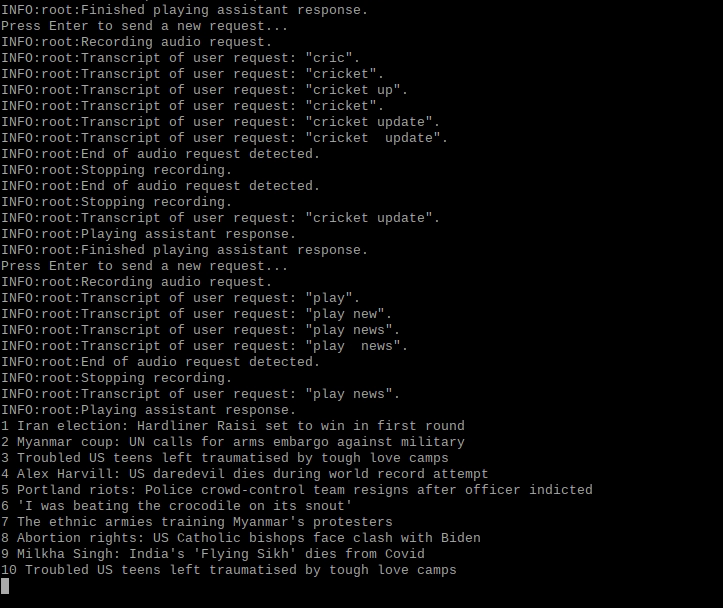
1. Ring my phone feature



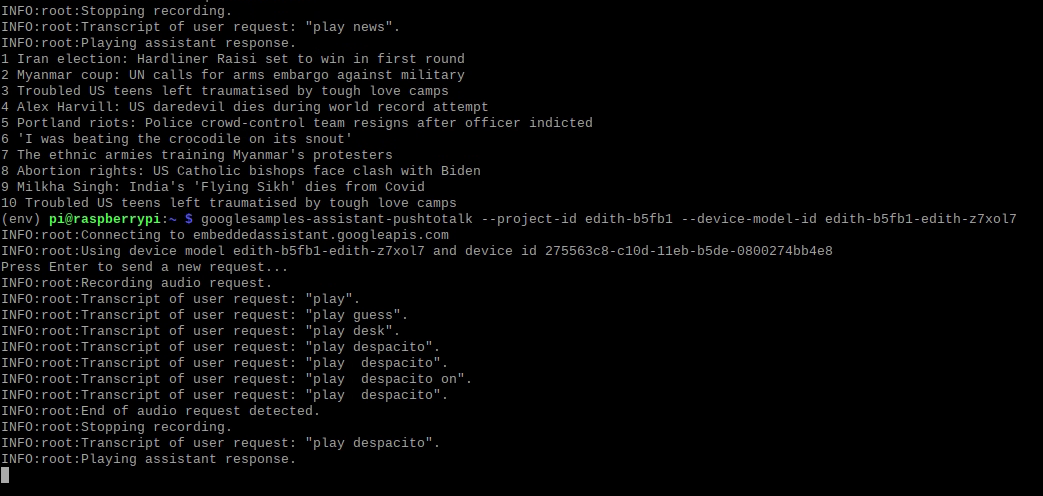
1. Jokes and Facts

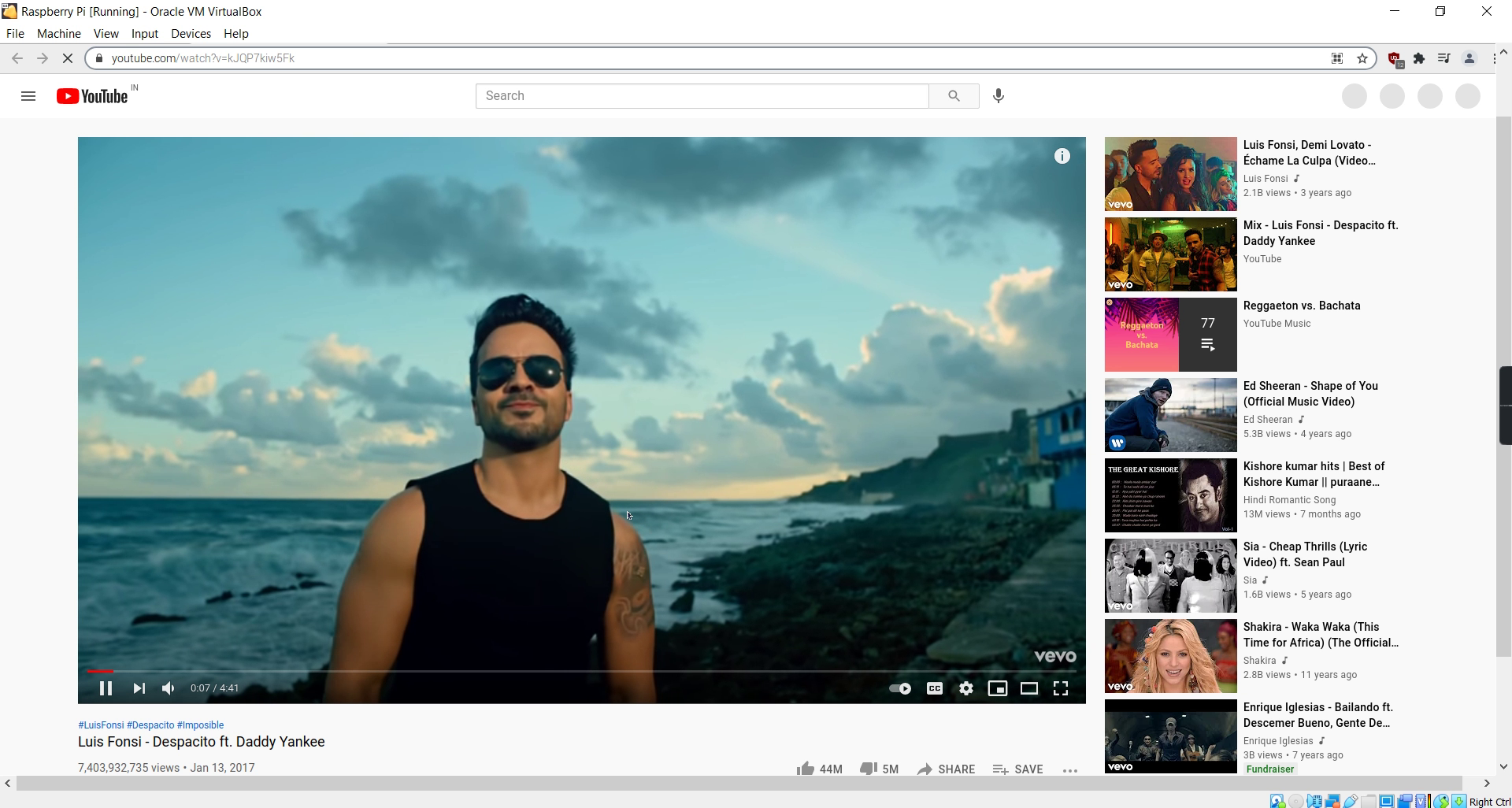


1. News and Sports Updates



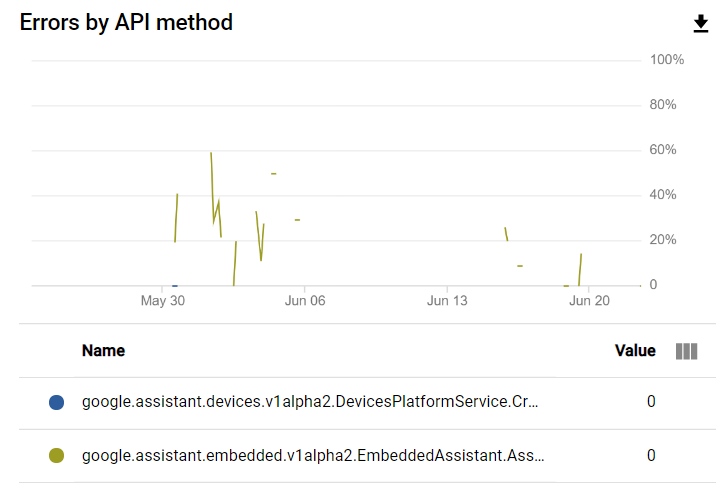
1. Music Playback



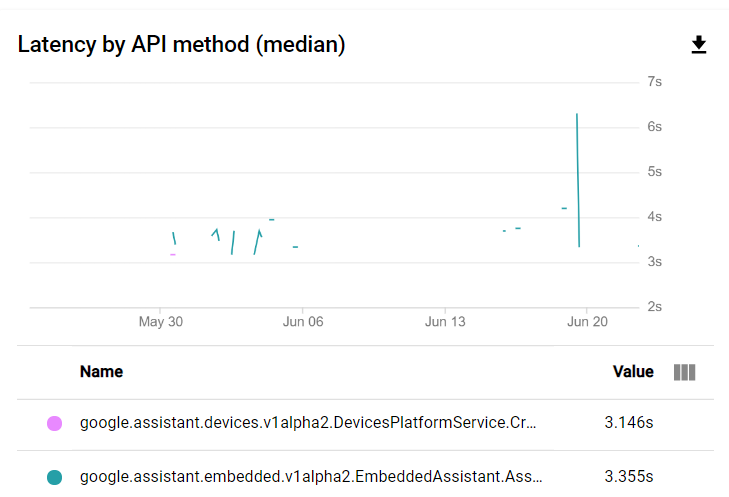


**PERFORMANCE TESTING**

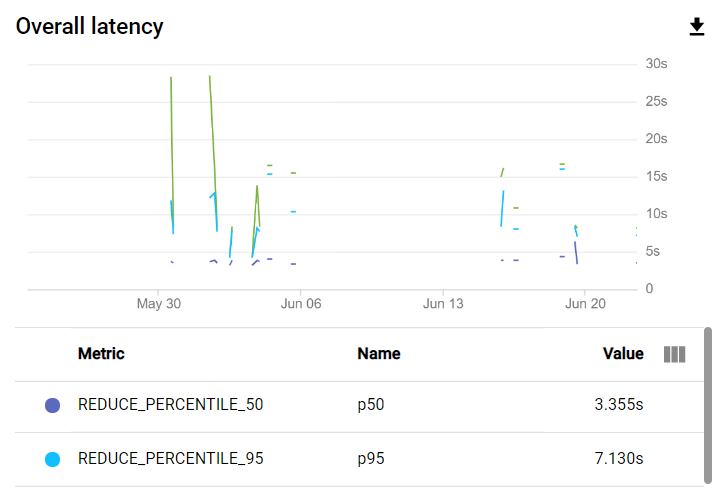
1. Errors encountered



1. Latency by API



1. Overall Latency



1. Traffic Response



**CONCLUSION AND FUTURE WORK**

**Conclusion**

The project **“Edith”** has been developed as per the requirement specification. The complete functionality has been thoroughly tested, to eliminate bugs and enhance the user experience.

The design, implementation and the output reports are presented in this project report. The entire project was meticulously designed to ensure seamless user experience and easier incorporation of future modules.

**Future Work**

The goals of this project were purposely kept within what was believed to be attainable within the allotted timeline and resources. As such, many improvements can be made upon this initial design. That being said, it is felt that the design could be replicated to a much larger scale. The following are the features we wish to add in the future:

* Extend the project to support Home Automation like turning on lights, controlling devices with voice etc.
* Extend to support Messages, SMS and Emails.
* Sending Voice Mails, Play Radio.
* Tracking parcels/orders.
* Booking Movie/Travel Tickets.
* Make purchases in apps.

**REFERENCES**

[1]. <https://developers.google.com/assistant/sdk/guides/library/python>

[2]. <https://github.com/googlesamples/assistant-sdk-python>

[3]. <https://pypi.org/>

[4]. <https://cloud.google.com/apis/docs/getting-started>

[5]. <https://www.raspberrypi.org/forums/>

[6]. <https://ubuntuforums.org/>

[7]. <https://stackoverflow.com/>

[8]. <https://www.geeksforgeeks.org/>

**FULL PROJECT DEMONSTRATION**

<https://www.youtube.com/watch?v=CJJJV1EhqXg>