# ARTEFACT DESIGN & DEVELOPMENT

## Introduction

Chatbots are very common now a days as we interact with social media platforms, online shopping recommenders, Q&A sessions, and in fact a person with smartphone device have easy access to Google assistant in android devices, Cortana in Windows supported devices and Siri in iOS devices marking the highest peak of artificial intelligence in the form of new generation interactive intelligent chatbots. Even with such advancements in the development of chatbots, there is still much room for working on specific fields instead of working on a generic purpose chatbot and for that a relocation assistance providing chatbot is developed.

The chatterbot supported database module which enables real time data storage of the chatbot conversations marks a new way of feeding data to the intelligent chatbot on real time basis.

The chatbot answers some normally asked questions from the already trained datasets stored in the database, if the asked question has no confident answer available in the database then query is forwarded to the secondary database that is an external database (wolframalpha) for getting answer in response to the user query. Even if the answer is not there in the secondary database then Google search will invoke and get results from the Google search engine.

## Chatterbot Database search

We have trained data models searched and gathered from several sources in the form of .txt, .JSON, .CSV, .YML and XML files which then converted into three final formats (.TXT, .YML, and .JSON) for which the code was implemented to read through the lines and train data i.e (question answer sessions, informative content, end-to-end chats) to the chatterbot corpus which get stored in a single SQLite3 database file.

Another feature supported by Chatterbot Python library is that it provides a way of storing the real time conversations being made on the chatbot app into the same database which makes the database more room for the possible combinations of questions aligned with responded answers and, they definitely increase the capability of chatbot to get best match from the database that is getting trained on real time basis.

## Wolframalpha search

An external API integrated with the chatbot application that has a support from ‘itertools’ and ‘google-search’ API’s are of vast scope which is being used as a secondary database to the relocation-assist chatbot application.

## Google search

The 3rd and the most important source of information and data for the chatbot is Python-google-search-API. Python based google-search API is implemented in a way to use the query received from the user end to be sent to google search and out of the possible google search answers, informative content from the top 3 search results is combined and concatenated in a string before it is sent as a response to the user’s query.

It has some pre-trained intents to understand the context of the question and answer only the specific search result as a result in case of a specific question like i.e., when was Jesus born?, When was World War two started? And related more, instead of concatenating the top 3 search results.

## Architecture

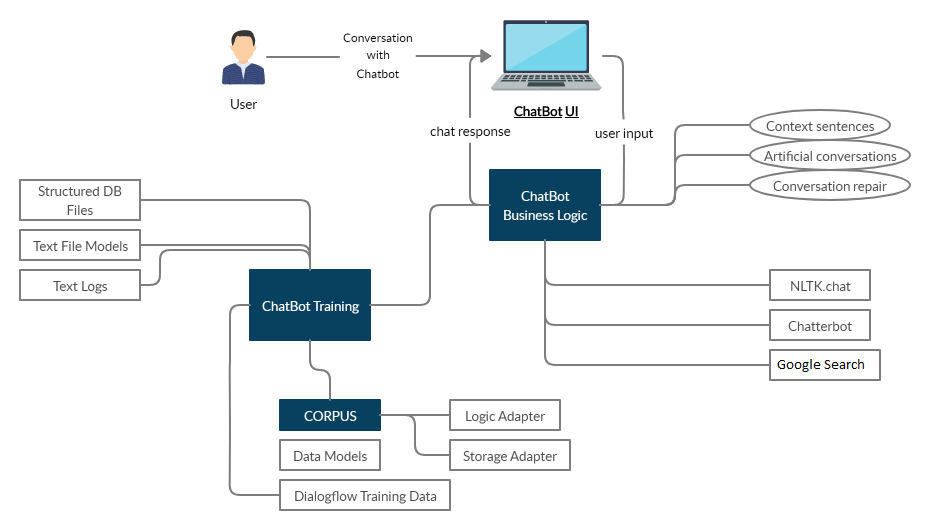
****

Figure 8 System Architecture

## Data flow

Data flow actually depends totally on the phases a query received from the server side has passed through. Following steps are involved up to 3 levels in getting a satisfied answer to a specific user query:

Get response from Chatterbot trained database

Get response from Wolframalpha secondary database if not found from the Chatterbot DB

Send query to Google-search-API’ for getting response from Google search

If not found, send a pre-defined error message to represent: no data found

## File Structure

Answers to the user queries are searched from the three data sources i.e., chatterbot database, external database ‘wolframalpha’, and google search result. The chatterbot database created consists of the onetime trained data models which in our case are .txt, .JSON, and .YML files and the conversation logs being recorded from the chatbot sessions.

JSON module is imported in the source code as a library to support reading the .JSON files and get them trained to the Chatterbot database

Glob is imported as a library to support reading the .TXT files in a specific directory and train them one by one into Chatterbot database

Chatterbot corpus trainer is imported as a library to support training of the .YML files into Chatterbot database

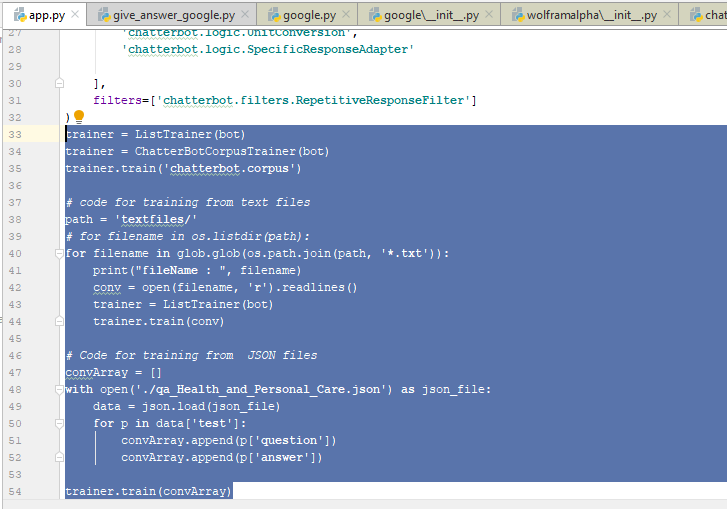
****

Figure 9 Screenshot for Corpus trainer

## Design

Covering the limitations of contribution in relocation assistance, we have worked with Python Chatterbot API for implementing a customized chatbot application backed with Flask and SQLite3 with an interactive Frontend interface implemented in HTML, JavaScript to render as a Flask web app at the default IP address at pot 5000. Later on the IP address got changed when deployed on production server for real time testing. With the power of Python 3.7 modules, several solutions were analysed including implementation with NLTK, Dialog Flow, integration with IBM Watson datasets for chatbot implementation but out of all, the customization of chatbot by integrating it with an external database ‘wolframalpha’ proved to be more enhanced way of getting stage-1, stage-2 and then stage-3 response when google search API for python got integrated with the custom developed chatbot application.

### **Chatterbot**

Chatterbot is a powerful Python library with advanced support for implementing a chatbot application with the help of following modules:

Storage Adapters: Used for storing data into database

Supports SQLite3 Database Storage

Supports MongoDB storage

Supports SQL storage

Logical Adapters: Used for deciding the answer

Best Match Logical Adapter: Supports the best match for a query on the basis of the semantics of received user query.

Mathematical Evaluation Adapter: This logical adapter supports the mathematical calculations and intelligently recognizes if there is some calculation query received from the user’s statement.

Time Logic Adapter: This logical adapter provides the current time if asked by the user.

Specific Response Adapter: If you want a specific response to be given to a special case then this adapter is of use.

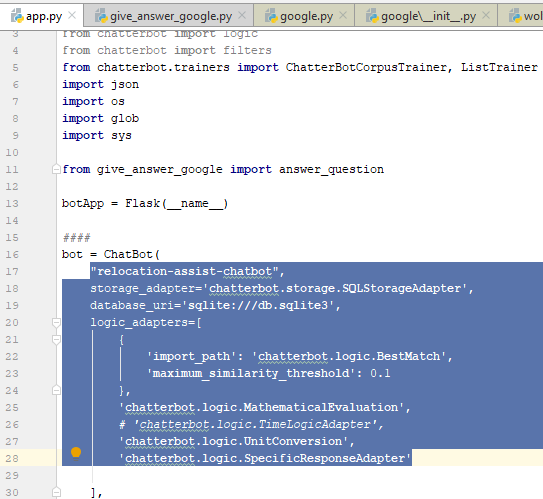


Figure 10 Screenshot for Logic adapters

Trainer: Chatterbot trainer module supports corpus trainer and list trainer sub modules for training of the data models.

Real Time training: The Chatterbot corpus trainer has an advanced feature of real time training of conversation data from the user sessions being stored in the chatbot database via Storage adapter.

### **Wolframalpha API**

Wolframalpha search API is used as a secondary source of chatbot for finding a suitable answer for the user query. The user query is received as a statement to be queried in the Wolframalpha Database for finding the answer to that specific query.

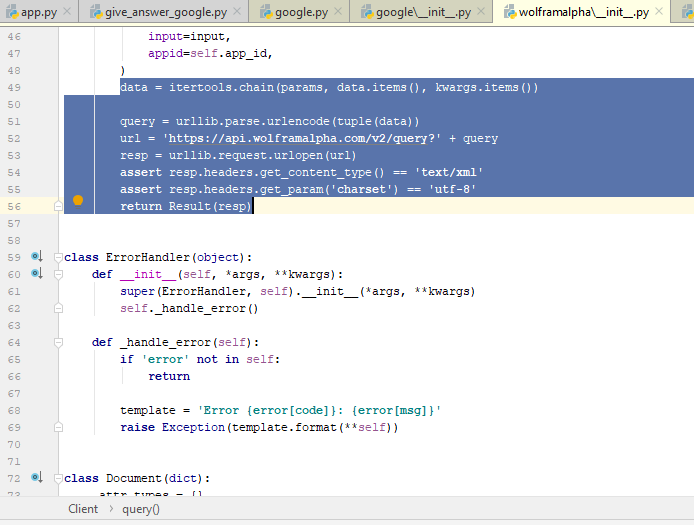


Figure 11Screenshot wolframalpha

The API works after an app token obtained by registering for the API is added in the source code.

### **Google search**

Returning the search results from google search engine

Concatenating the top 3 search results

Forming up a string of top 3 search results

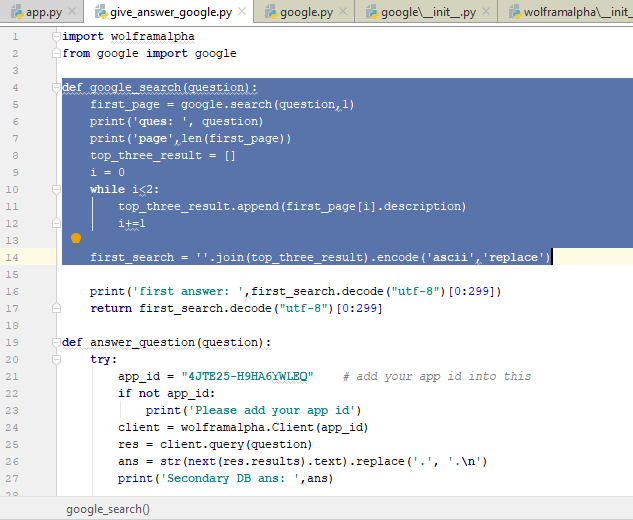
****

Figure 12 Screenshot Google search

Checking the limit of characters for string to send the character size in response.

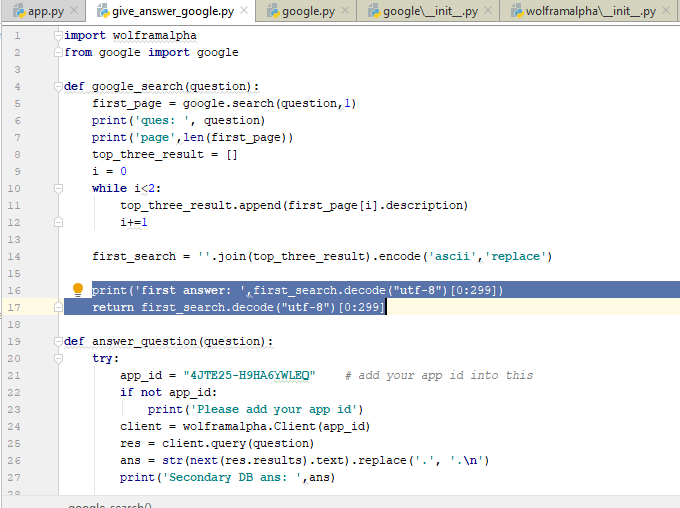
****

Figure 13 Screenshot Character limit

Specific answer added in the code is to be displayed if the answer to the query posted by the user is not found in all of the above 3 sources.

### **Response confidence**

Interacting with the main source code implemented in Python utilizing the Chatterbot API, the extended libraries are integrated in such a way that the responses generated as a response to the user query from primary and secondary databases are filtered through a response confidence to check whether the response his of a satisfied confidence level to be sent as a response to the user or not. If not then query is forwarded further for google search.

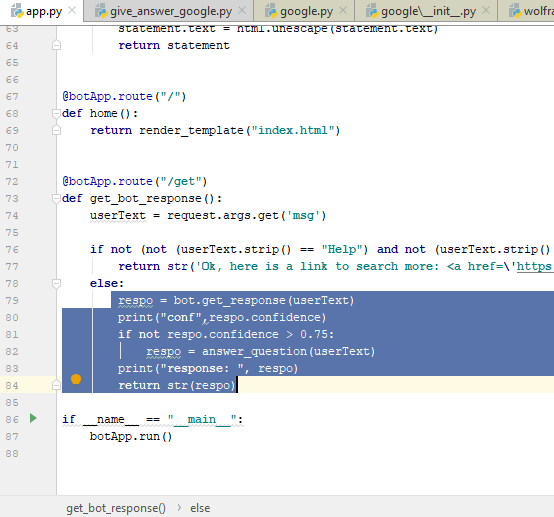


Figure 14 Screenshot Response confidence

## Server Deployment

The chatbot implemented and tested is deployed to a production server to be made available for the real user testing. The live link to deployed relocation-assist chatbot is shared with the targeted people of scope and their responses and interaction experience with the chatbot is recoded with the help of a survey form.

### **Server setup**

A digital ocean droplet was created for installing a dedicated server dependencies for making the application live. Ubuntu 18.04 was installed with x64 OS on the server droplet.

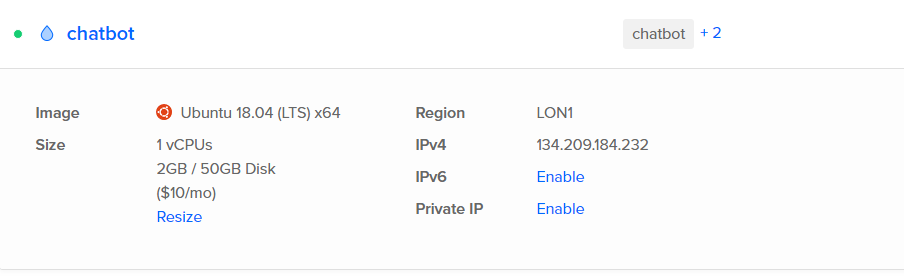


Figure 15 Server setup

Establishing server connection via Putty:

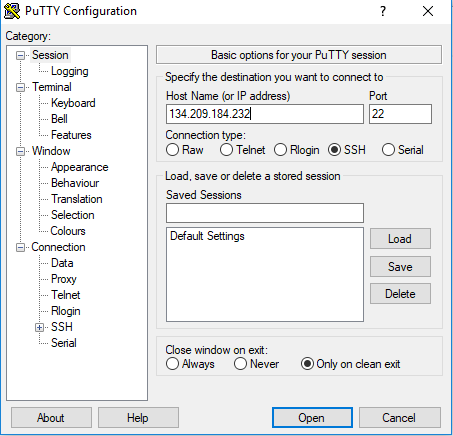


Figure 16 Server Connection

Dependencies required are put into the requirements.txt as ‘freeze’ for being installed on the server:

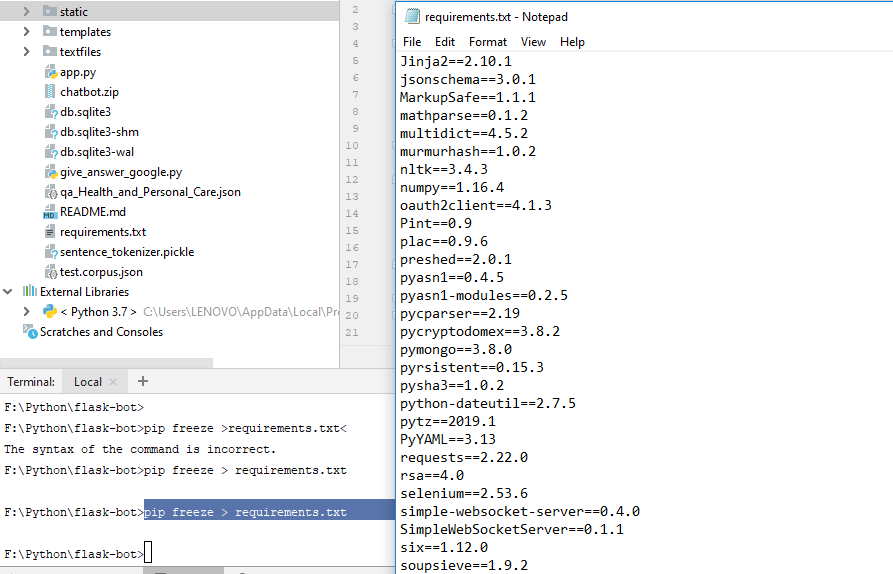


Figure 17 Dependencies

Dependencies were installed one by one including:

Python 3.7.3

Flask 1.1.1

Pip3

Chatterbot

Wolframalpha API

Google

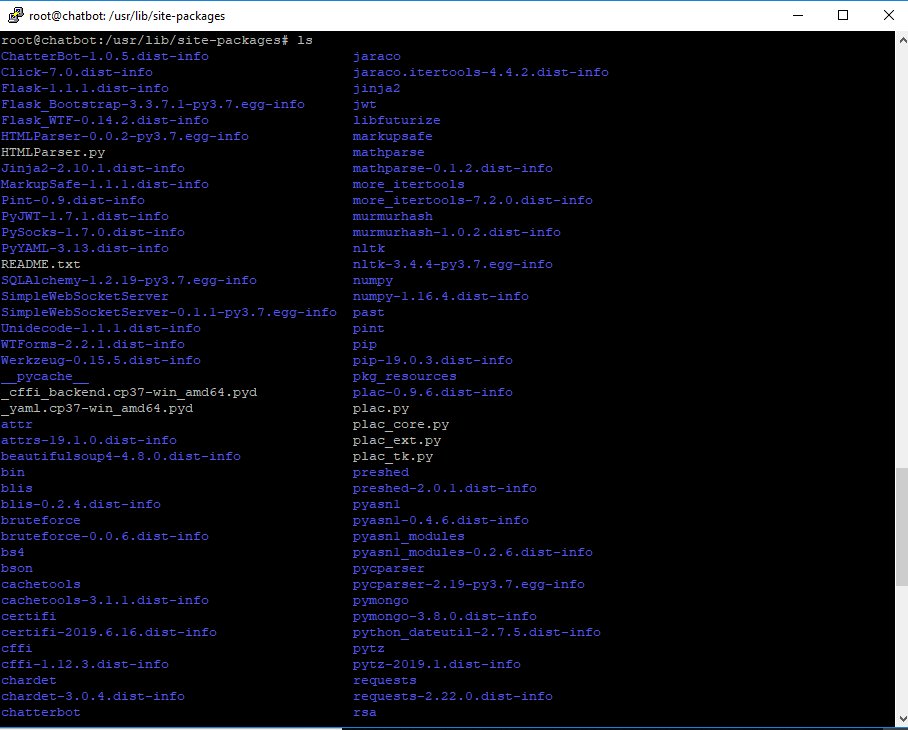


Figure 18 Dependencies

And others were transferred directly transferred from the local machine as running locally via FileZilla.

### **Chatbot deployment**

Chatbot application is then deployed onto the production server by transferring all the source files via FileZilla. You can see the source files moved to the server in the below snip shot.

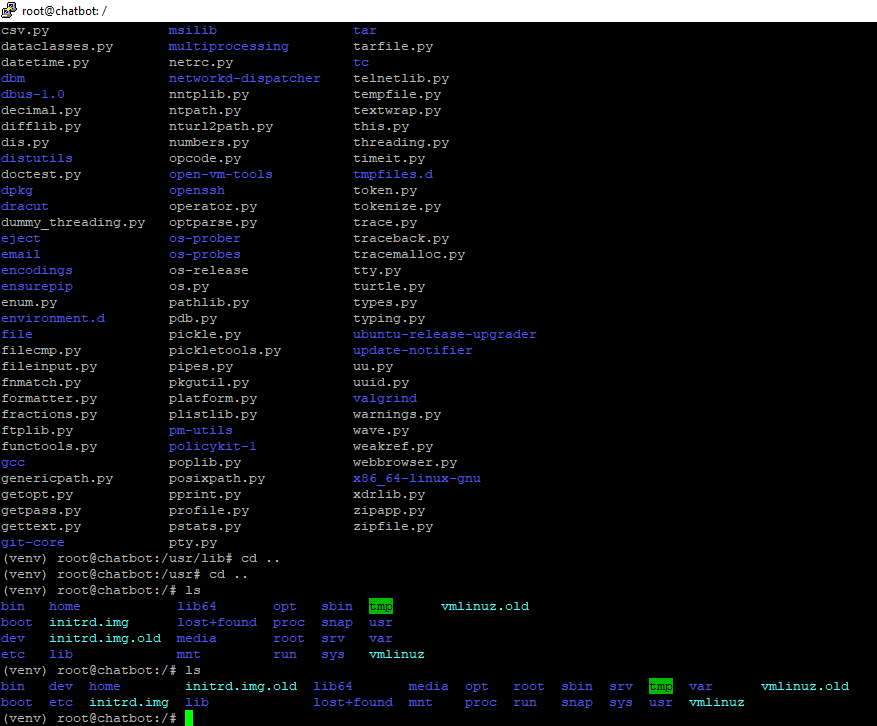


Figure 19 Chatbot deployment

### **Chatbot running on server**

Chatbot is then ran on the server by initiating a server screen so in case we exit from the server connection, the application should remain in running state in a specific screen.

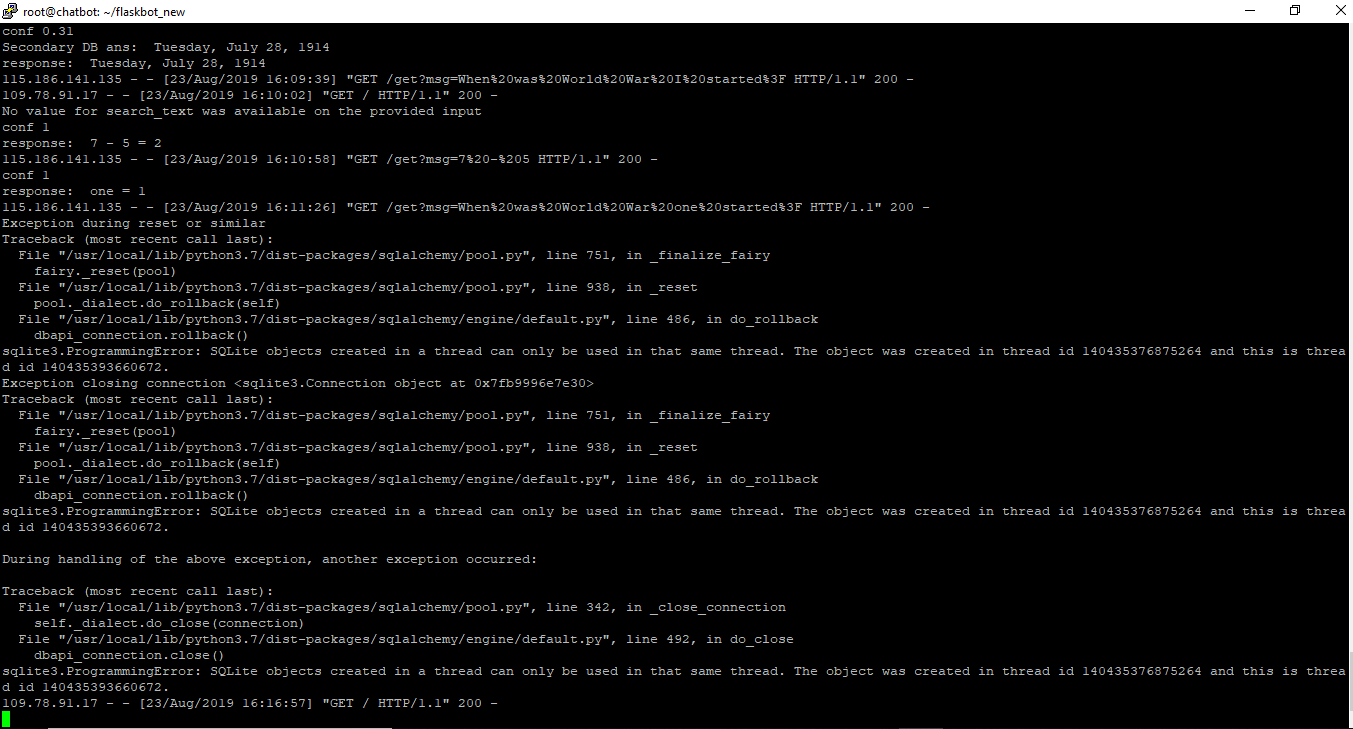


Figure 20 Server status

### **Future work**

In future, we look forward to enhance the source structure of the working modules and implement an API to be provided open for the people to integrate this relocation-assist chatbots with their websites. The tourism businesses would benefit mainly from this chatbot application s they would have an extra feature for their customers to interact with relocation-assist chatbot to learn about the city they are going to explore instead of spending money on the travel guides.

We came across finding a 38 GB file containing 1 billion Reddit conversation sessions but, due to limitation of advanced software tools and shortage of time it was difficult to extract data out of this huge file and train to the database but, we look forward to find a way to work around with this important data to be trained to the chatbot in near future

## App.py Code:

from flask import Flask, render\_template, request

from chatterbot import ChatBot

from chatterbot import logic

from chatterbot import filters

from chatterbot.trainers import ChatterBotCorpusTrainer, ListTrainer

import json

import os

import glob

import sys

from give\_answer\_google import answer\_question

botApp = Flask(\_\_name\_\_)

####

bot = ChatBot(

"relocation-assist-chatbot",

storage\_adapter='chatterbot.storage.SQLStorageAdapter',

database\_uri='sqlite:///db.sqlite3',

logic\_adapters=[

{

'import\_path': 'chatterbot.logic.BestMatch',

'maximum\_similarity\_threshold': 0.1

},

'chatterbot.logic.MathematicalEvaluation',

# 'chatterbot.logic.TimeLogicAdapter',

'chatterbot.logic.UnitConversion',

'chatterbot.logic.SpecificResponseAdapter'

],

filters=['chatterbot.filters.RepetitiveResponseFilter']

)

trainer = ListTrainer(bot)

trainer = ChatterBotCorpusTrainer(bot)

trainer.train('chatterbot.corpus')

# code for training from text files

path = 'textfiles/'

# for filename in os.listdir(path):

for filename in glob.glob(os.path.join(path, '\*.txt')):

print("fileName : ", filename)

conv = open(filename, 'r', encoding="utf8", errors='ignore').readlines()

trainer = ListTrainer(bot)

trainer.train(conv)

# Code for training from JSON files

convArray = []

with open('./qa\_Health\_and\_Personal\_Care.json') as json\_file:

data = json.load(json\_file)

for p in data['test']:

convArray.append(p['question'])

convArray.append(p['answer'])

trainer.train(convArray)

###### PreProcessor ######

def unescape(bot, statement):

if sys.version\_info[0] < 3:

from HTMLParser import HTMLParser

html = HTMLParser()

else:

import html

statement.text = html.unescape(statement.text)

return statement

@botApp.route("/")

def home():

return render\_template("index.html")

@botApp.route("/get")

def get\_bot\_response():

userText = request.args.get('msg')

if not (not (userText.strip() == "Help") and not (userText.strip() == "help")) or (userText.strip() == "Help!"):

return str('Ok, here is a link to search more: <a href=\'https://www.google.com\'>www.google.com</a>')

else:

respo = bot.get\_response(userText)

print("conf",respo.confidence)

if not respo.confidence > 0.75:

respo = answer\_question(userText)

print("response: ", respo)

return str(respo)

if \_\_name\_\_ == "\_\_main\_\_":

botApp.run()

## give\_anser\_google.py Code:

import wolframalpha

import google

def google\_search(question):

first\_page = google.search(question,1)

print('ques: ', question)

print('page',len(first\_page))

top\_three\_result = []

i = 0

while i<1:

top\_three\_result.append(first\_page[i].description)

i+=1

first\_search = ''.join(top\_three\_result).encode('ascii','replace')

print('first answer: ',first\_search.decode("utf-8")[0:299])

return first\_search.decode("utf-8")[0:299]

def answer\_question(question):

try:

app\_id = "4JTE25-H9HA6YWLEQ" # add your app id into this

if not app\_id:

print('Please add your app id')

client = wolframalpha.Client(app\_id)

res = client.query(question)

ans = str(next(res.results).text).replace('.', '.\n')

print('Secondary DB ans: ',ans)

if ans == 'None' or ans == '(data not available)' or ans == '(information not available)':

print('none-google')

ans = google\_search(question)

print('google answer: ', ans)

return ans

except:

try:

print('except-google')

ans = google\_search(question)

print('google answ: ', ans)

return ans

except:

return str('Ok, here is a link to search more: <a href=\'https://www.google.com\'>www.google.com</a>')

## index.html Code:

<!DOCTYPE html>

<html>

<head>

<link rel="stylesheet" type="text/css" href="/static/style.css">

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<head>

<link

rel="shortcut icon"

type="image/x-icon"

<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.js"></script>

<style>

body {

font-family: calibri;

}

h1 {

background-color: blue;

display: inline-block;

font-size: 3em;

margin: 0;

padding: 14px;

}

h3 {

color: black;

font-size: 20px;

margin-top: 3px;

text-align: center;

}

#chatbox {

margin-left: auto;

margin-right: auto;

width: 40%;

margin-top: 60px;

}

#userInput {

margin-left: auto;

margin-right: auto;

width: 40%;

margin-top: 60px;

}

#textInput {

width: 90%;

border: none;

border-bottom: 3px solid black;

font-family: monospace;

font-size: 17px;

}

.userText {

color: white;

font-family: monospace;

font-size: 17px;

text-align: right;

line-height: 30px;

}

.userText span {

background-color: #808080;

padding: 10px;

border-radius: 2px;

}

.botText {

color: white;

font-family: monospace;

font-size: 17px;

text-align: left;

line-height: 30px;

}

.botText span {

background-color: #4169e1;

padding: 10px;

border-radius: 2px;

}

#tidbit {

position: absolute;

bottom: 0;

right: 0;

width: 300px;

}

.chat-box {

margin-left: auto;

margin-right: auto;

width: 78%;

margin-top: 60px;

border: 1px solid green;

}

.box {

border: 2px solid black;

}

</style>

</head>

<body>

<center>

<h1>

Your Relocation assist ChatBot

</h1>

</center>

<div class="box"></div>

<div class="chat-box">

<div>

<div id="chatbox">

<p class="botText">

<span>Hi! I'm a ChatBot and I'm here to assist you.. </span>

</p>

</div>

<div id="userInput">

<input id="textInput" type="text" name="msg" placeholder="Message" />

</div>

</div>

</div>

<script>

function getBotResponse() {

var rawText = $("#textInput").val();

var userHtml = '<p class="userText"><span>' + rawText + "</span></p>";

$("#textInput").val("");

$("#chatbox").append(userHtml);

document

.getElementById("userInput")

.scrollIntoView({ block: "start", behavior: "smooth" });

$.get("/get", { msg: rawText }).done(function(data) {

var botHtml = '<p class="botText"><span>' + data + "</span></p>";

$("#chatbox").append(botHtml);

document

.getElementById("userInput")

.scrollIntoView({ block: "start", behavior: "smooth" });

});

}

$("#textInput").keypress(function(e) {

if (e.which == 13) {

getBotResponse();

}

});

</script>

</div>

</body>

</html>