|  |
| --- |
| **Project Showcase** |

1. <https://www.linkedin.com/in/henri-georges-demanou-tchamdjou> link to my LinkedIn profile
2. <https://github.com/henri500/Python-Discord-bot/wiki>

**Introduction:**

This aim of this project was created a chat bot that uses the discord interface to interact with uses that have a discord account. At the end of this prototype, the chat bot has the following features:

* Stream music from YouTube in to the discord voice channel they a u user is currently in
* Find navigation routes to place users want
* Provide news headlines concerning sports and more from different sources depending on the user's choice
* Give information on sport team such as foundation date, league and the colour of the team's kit

**My Role:**

In this project, apart from helping my team members with some coding problems I was in charge of the following tasks:

* Create the interface between the discord API and the main script of the Chabot
* Bring together the code written by my team mates to work with together.
* Handle message received from users and determine what action the Chabot would take. An example of this is how I filter commands from common text.
* Used Open Weather API to retrieve weather information according to the user's location.

**How I does it work?**

Because we wanted the Chabot’s main interface to be discord, we had to import it and create a discord client object as such:

TESS=discord.Client() #--where TESS is the client object that we create.

TESS.run() #--Using the discord run function that connects the machine running the scripts to the discord server placed at the end of the script.

Now that the bot is up and running, I used the *@Client.event* function provided by discord API to detect incoming messages. To be able to differentiate if the text received what a command or general text I wrote the following code:

**if** ((message.content[**0**])=='!' **and** ('play' **in** message.content)): #--checking if an input is a command

**if** TESS.is\_voice\_connected(server) == False: #--checking if we already have a voice connection to the voice client

await TESS.join\_voice\_channel(channel)

TESS\_voice=TESS.voice\_client\_in(server)

**print**('joined voice channel')

message.content=message.content.strip('!')

message.content=message.content.strip('play'

first\_link=get\_vid\_link(message.content)

song\_lenght=int(lenght\_song(first\_link))

vid='https://youtube.com{}'.format(first\_link)

song\_player=await TESS\_voice.create\_ytdl\_player(vid) #--Creates a stream player in a new thread in the background from YouTube.

players\_instances[server.id]=song\_player #-- Storing an instance of the song play so that it can accessed later

song\_player.start()

In the above example, if the first character of the in-coming message is '!' and if it contains the string 'play', then I consider this message to be a command. In this case, the command is to play a song. I used the same method to identify other commands such as stop and resume.

Now that I know the message is a "play” command, I checked if the there was a voice connection between the user and the bot in other to stream audio to the channel the user is in. I removed the command prefix ("!" and "play") from the message and passed it to the format\_message function which remove characters such as question marks. In order to play the correct song required by the user, I wrote a get\_vid\_link function that uses requests and BeautifulSoup modules to search for a song/artist on YouTube and returns the id of the first video that appears on a YouTube search:

**def** **get\_vid\_link**(artist): #--the argument passed to this function could also be a song.

'''this function that takes in a string and searches it on youtube's website. It gets the first link on the page and return a video id '''

url="https://www.youtube.com/results?search\_query=" + artist

response=requests.get(url)

links=response.text #--converting response object to text:

data=BeautifulSoup(links,'html.parser')#--convert text in html format

list\_links=[]

**for** tag **in** data.find\_all('a'): #--retrieving every link from the <a> tag in data

hf=tag.get('href')

**if** '/watch?v' **in** hf:

list\_links.append(hf)

**if** len(list\_links)==**1**:

**return**(list\_links[**0**]) #--selecting the first link.

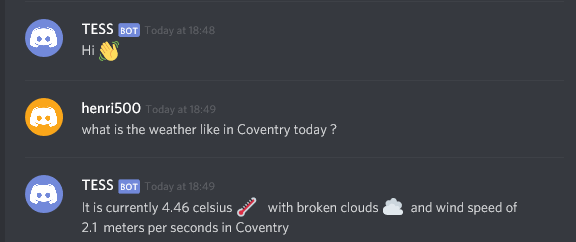
**break**

**else**:

**return** False

***Why does it return only the first link?*** Well based on youtube's searching algorithm which is very accurate, the first link on the page would very likely be the song or artist’s most popular song (this will be the case if the user only enters the name of an artist.) the user requested. This can of course be improved in later prototypes by giving the choice to the user to select between the first five results returned by the YouTube search engine. Finally, I used the discord function (*create\_stream\_player()*) to create a new thread in which a background song player will be initialized.

Another example of the Features this Chat bot has is to displaying weather conditions of a specific location. The Screenshot below shows this feature in action:



To achieve this, I used [Wit.ai](https://wit.ai/) set of similar sentences which all ask for the weather.

Wi.ai is a natural language API which helped me to train a bot to extract meaning out of text. It does that by comparing the user’s message to a set of predefined sentences with the same intend.

An extract of the code I wrote to get weather information can be seen below:

**def** **get\_weather**(location):

'''make a weather query to API and uses the location as a parameter and returns the weather as a list of key info as a list'''

weather\_info=[]

url='https://api.openweathermap.org/data/2.5/weather?q={}&appid=e5329dd2891f08e84c9dc73b913bcab0&units=metric'.format(location)

response=requests.get(url)

data=response.json()

overview =(data['weather'][**0**]['main'])

description=(data['weather'][**0**]['description'])

temp=(data['main']['temp'])

wind\_speed = (data['wind']['speed'])

location=data['name']

weather\_info.append(overview) #--- summary description

weather\_info.append(temp) #-- temperature

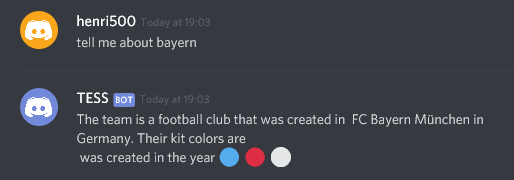
weather\_info.append(description)

weather\_info.append(wind\_speed)

weather\_info.append(location)

**return** weather\_info

In addition to given weather conditions it can also give some information a football clubs as follows :



**What I have learnt while working on this project?**

Throughout the course of this project I have gained many valuable skills. The first one is to be able to work within a team. I always thought it was easier to take on projects individually but doing this group project made me realise that in reality the best work is often done as a team. This is because as a team we learn from each other and complement each other’s skills. Secondly, having to work with a deadline improved my time management skills. This was very crucial because I needed to prioritize time spent on some features of the Chabot and implement them within a fixed amount of time. It also helped me to put into practice what I have learnt about agile development. At the beginning of the project I could not install the discord python library. After some research, I realised it was due to fact that I was not using the correct python version(python3.6). As a result, I had to use a Virtual environment in PyCharm Editor so that I could easily check the versions and required module for my projects. This was a good lesson for me because with a Virtual environment, I can test my code on a specific platform, relocate my entire project to a different machine while keeping the same configuration.

In addition to that, being a team leader in this project helped me development leadership skills. This was the case because I had to make sure everyone knew what they had to do, if we were on Woking on schedule or not.

Apart from personal development, I also improve on my technical skills. For example, I in order to get data about the weather I had to query an API and filter the data I needed. I also had a brief introduction to Asynchronous programming in python and how powerful it is in designing and optimizing programs.

**Project Evaluation:**

Personally, one of the good aspect of this prototype is the fact it’s able to work well with external entities such as APIs. But on the other side it lacks in terms of complexity. What I mean is that although it executes queries well, it does not correctly account for some errors. Errors that could include; the API’s servers not responding and more. To solve this problem in feature prototypes the use of error handling in python will be crucial. Secondly, this prototype only reacts to commands. To improve this aspect, we can use libraries such Kera, TensorFlow create a neuronal network that can be trained on conversation datasets to be able to proactively reply to users without having to rely on the Wit API to do this for us. Some examples that we of datasets we could use are Cornell movie dialogs or even reddit comments. Toi go even further we could do sentiment analysis on the user’s messages to determine if they are sad or happy. If they happened to be sad we could suggest them to watch a movie or some other entertaining activity. This can be done with the help of libraries such as Text Blob to classify sentences into good or bad.

|  |
| --- |
| **Python Code** |

<https://github.coventry.ac.uk/demanouh/CHATBOT>

Code that I wrote (Any code highlighted in yellow was not used):

**import** **requests** #this is an external library that i will use to make http requests to the Weather API

**def** **get\_weather**(location):

'''make a weather query to API and uses the location as a parameter and returns the weather as a list of key info'''

weather\_info=[]

url='https://api.openweathermap.org/data/2.5/weather?q={}&appid=e5329dd2891f08e84c9dc73b913bcab0&units=metric'.format(location)

response=requests.get(url)

data=response.json()

overview =(data['weather'][**0**]['main'])

description=(data['weather'][**0**]['description'])

temp=(data['main']['temp'])

wind\_speed = (data['wind']['speed'])

location=data['name']

weather\_info.append(overview) #--- summary description

weather\_info.append(temp) #-- temperature

weather\_info.append(description)

weather\_info.append(wind\_speed)

weather\_info.append(location)

**return** weather\_info

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

get\_weather('london')

----------------------------------------------------------------------------------------------------------------------------------------------

**from** **bs4** **import** BeautifulSoup #--found at https://www.crummy.com/software/BeautifulSoup/bs4/doc/

youtube\_api\_v3\_key='AIzaSyALTsGjGJz0LDgYCw5jI7SltfQbUTRUUl0' #used to access youtube api for music duation

**def** **get\_vid\_link**(artist):

'''this function takesin a string and search youtube for that string. it gets the first link on the page and return an id(video id )'''

url="https://www.youtube.com/results?search\_query=" + artist

response=requests.get(url) #---request.get() is a a funtion from http://docs.python-requests.org/en/master/user/quickstart/#passing-parameters-in-urls

links=response.text #--coverting response object to text:

#---The code below is a modified version that can be found in the BeatifoulSoup Docs at :----------

#https://www.crummy.com/software/BeautifulSoup/bs4/doc/

data=BeautifulSoup(links,'html.parser')

list\_links=[]

**for** tag **in** data.find\_all('a'):#--retrieving every link from the <a> tag in data

hf=tag.get('href')

**if** '/watch?v' **in** hf:

list\_links.append(hf)

**if** len(list\_links)==**1**:

**return**(list\_links[**0**]) #--selecting the first link as basedon youtube's relevance filter to input

**break**

#----END of code inspired from https://www.crummy.com/software/BeautifulSoup/bs4/doc/ -------------

**def** **extract\_id**(link):

id\_sum=link.split('=')

id=id\_sum[**1**]

**return** id

**def** **lenght\_song**(vid\_id):

'''this function takes in the id of a vieo and returns the lenght of the vidoe as an interger'''

vid\_id=extract\_id(vid\_id)

url="https://www.googleapis.com/youtube/v3/videos?id="+vid\_id+"&key="+youtube\_api\_v3\_key+"&part=contentDetails"

response=requests.get(url)

response=response.json()

vid\_length=response['items'][**0**]['contentDetails']['duration']

vid\_length\_list=[int(num) **for** num **in** vid\_length **if** num.isdigit()]

total\_time=(int(vid\_length\_list[**0**])\***60**)+int((vid\_length\_list[**1**]))

#print(total\_time/100)

**return** total\_time

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

**print**(lenght\_song('watch?v=2tJipvjt4w4'))

t=(lenght\_song('watch?v=2tJipvjt4w4'))

time.sleep(t/**100**)

**print**('time done')

---------------------------------------------------I will add more of the code I have written Later -------------------------------------------------------------------------------------------

(Never use screenshots of your code. Your code should be presented as text. There are many good tools to help you format your code such as <http://hilite.me>)